Satellite Communications Toolbox

Reference

MATLAB®

R2023a

MathWorks®

How to Contact MathWorks

Latest news:	`www.mathworks.com`
Sales and services:	`www.mathworks.com/sales_and_services`
User community:	`www.mathworks.com/matlabcentral`
Technical support:	`www.mathworks.com/support/contact_us`
Phone:	508-647-7000

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

Satellite Communications Toolbox Reference

Revision History

March 2021	Online only	New for Version 1.0 (Release 2021a)
September 2021	Online only	Revised for Version 1.1 (Release 2021b)
March 2022	Online only	Revised for Version 1.2 (Release 2022a)
September 2022	Online only	Revised for Version 1.3 (Release 2022b)
March 2023	Online only	Revised for Version 1.4 (Release 2023a)

Contents

Apps

Satellite Link Budget Analyzer

Analyze link budgets for satellite communications

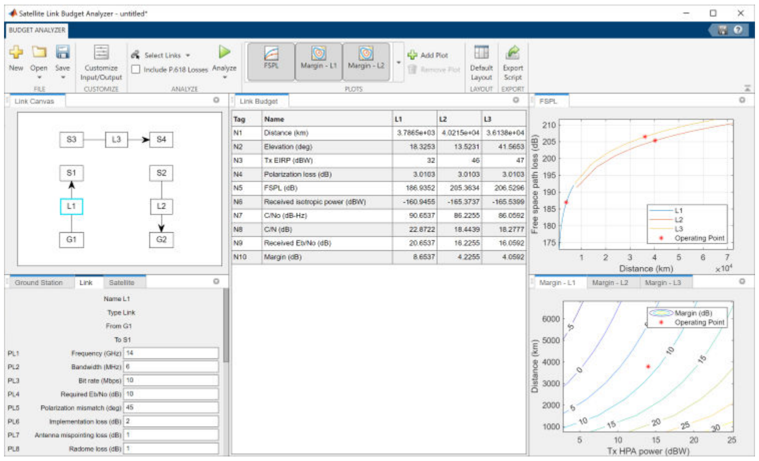
Description

The **Satellite Link Budget Analyzer** app enables you to analyze link budgets for satellite communications.

Using the app, you can:

- Analyze link budgets by specifying input properties related to the location, transmitter, and receiver characteristics of satellites and ground stations, as well as atmospheric conditions for links.
- Design a satellite communications link to meet a minimum link margin requirement.
- Gain insight into intermediate link budget computations.
- Calculate, compare, and visualize results across a sweep of multiple parameterized design constraints.

For more information, see "Get Started with Satellite Link Budget Analyzer App".



Tag	Name	L1	L2	L3
N1	Distance (km)	3.7865e+03	4.0215e+04	3.6138e+04
N2	Elevation (deg)	18.3253	13.5231	41.5653
N3	Tx EIRP (dBW)	32	46	47
N4	Polarization loss (dB)	3.0103	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634	206.5296
N6	Received isotropic power (dBW)	-160.9455	-165.3737	-165.5399
N7	C/No (dB-Hz)	90.6537	86.2255	86.0592
N8	C/N (dB)	22.8722	18.4439	18.2777
N9	Received Eb/No (dB)	20.6537	16.2255	16.0592
N10	Margin (dB)	8.6537	4.2255	4.0592

Name L1
Type Link
From G1
To S1

PL1	Frequency (GHz)	14
PL2	Bandwidth (MHz)	6
PL3	Bit rate (Mbps)	10
PL4	Required Eb/No (dB)	10
PL5	Polarization mismatch (deg)	45
PL6	Implementation loss (dB)	2
PL7	Antenna mispointing loss (dB)	1
PL8	Radome loss (dB)	1

Open the Satellite Link Budget Analyzer App

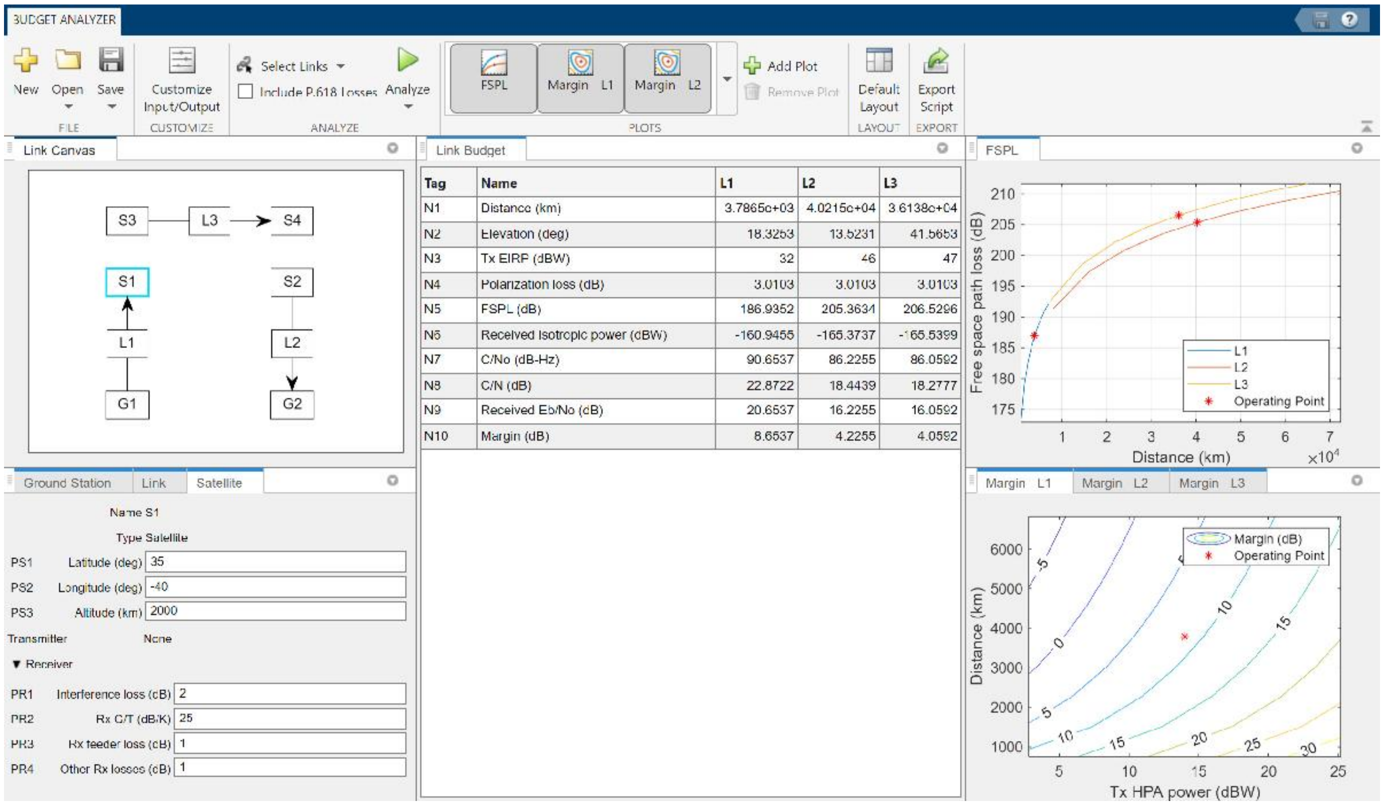
MATLAB® Toolstrip: On the **Apps** tab, under **Signal Processing and Communications**, click the app icon. Satellite Link Budget Anal...

MATLAB Command Prompt: Enter `satelliteLinkBudgetAnalyzer`.

Examples

Show Default Satellite Link Budget App Configuration

This example shows the default configuration that appears when you open the **Satellite Link Budget Analyzer** app. The figure shows the displayed results and plots, which analyze the default satellite communications link.



Link Budget

Tag	Name	L1	L2	L3
N1	Distance (km)	3.7865e+03	4.0215e+04	3.6138e+04
N2	Elevation (deg)	18.3253	13.5231	41.5653
N3	Tx EIRP (dBW)	32	46	47
N4	Polarization loss (dB)	3.0103	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634	206.5296
N6	Received isotropic power (dBW)	-160.9455	-165.3737	-165.5399
N7	C/No (dB-Hz)	90.6537	86.2255	86.0592
N8	C/N (dB)	22.8722	18.4439	18.2777
N9	Received Eb/No (dB)	20.6537	16.2255	16.0592
N10	Margin (dB)	8.6537	4.2255	4.0592

Ground Station | Link | Satellite

Name S1

Type Satellite

PS1	Latitude (deg)	35
PS2	Longitude (deg)	-40
PS3	Altitude (km)	2000

Transmitter None

▼ Receiver

PR1	Interference loss (dB)	2
PR2	Rx G/T (dB/K)	25
PR3	Rx feeder loss (dB)	1
PR4	Other Rx losses (dB)	1

The upper-left pane of the app shows the **Link Canvas** tab, which displays this default configuration:

- Link L1 is an uplink connecting ground station G1 to satellite S1
- Link L3 is a crosslink connecting satellite S3 to satellite S4
- Link L2 is a downlink connecting satellite S2 to ground station G2

The lower-left pane of the app shows the **Ground Station**, **Link**, and **Satellite** tabs. In these tabs, you can adjust the property settings for each entity in the configured links. To view or adjust the property settings of an entity, bring that entity into focus by selecting it in the **Link Canvas** tab.

The center pane of the app shows the computed link budget results in the **Link Budget** tab.

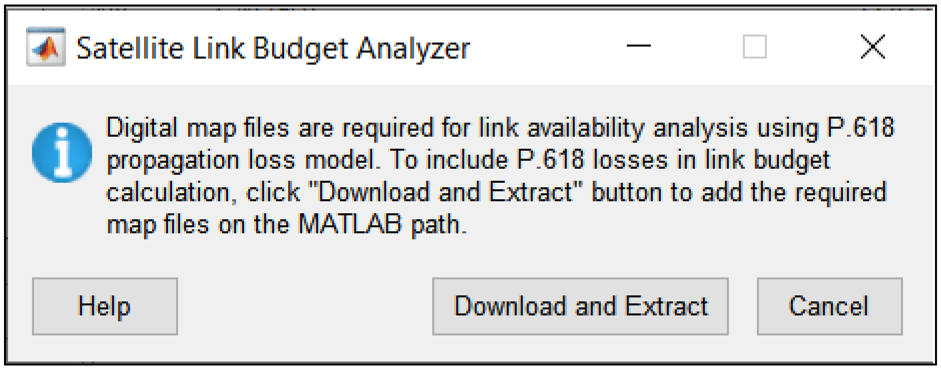
The right pane of the app window shows these plots:

- Free-space path loss for links L1, L2, and L3 in the upper-right area (`FSPL` tab).
- Link margins for links L1, L2, and L3 in separate tabbed plots in the lower-right area (`Margin-L1`, `Margin-L2`, and `Margin-L3` tabs, respectively).

Configuration Including P.618 Link Availability Analysis

The app supports analyzing the satellite communications link availability through the propagation loss model defined in Recommendation ITU-R P.618-13. For details on the P.618 propagation loss model, see “Earth-Space Propagation Losses”.

To include ITU-R P.618 propagation losses for availability analysis, select the **Include P.618 Losses** checkbox on the **Budget Analyzer** tab. If the MAT-files with digital maps are not available on the path, the following dialog box appears. Click the **Download and Extract** button to add the required map files to the MATLAB path.



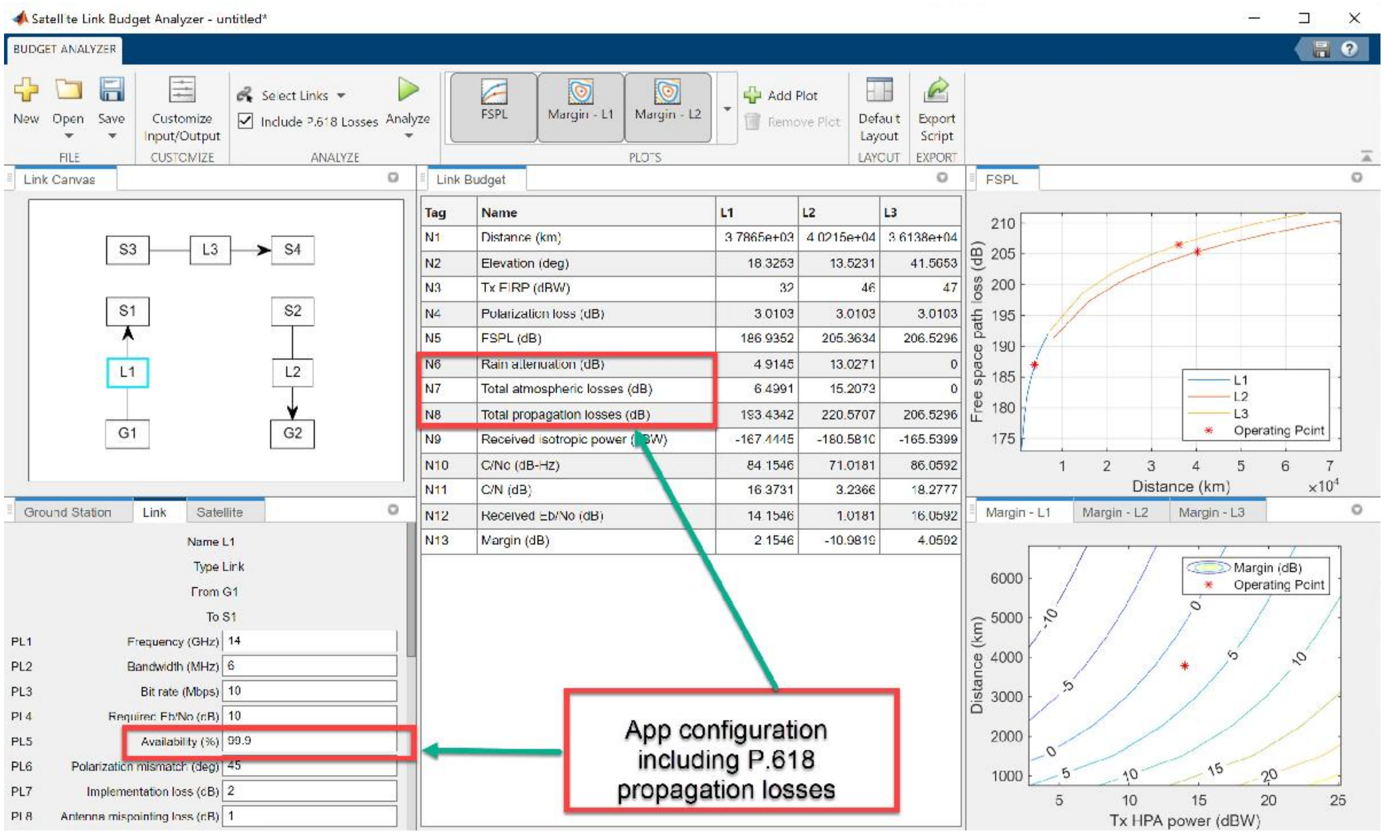
Digital map files are required for link availability analysis using P.618 propagation loss model. To include P.618 losses in link budget calculation, click "Download and Extract" button to add the required map files on the MATLAB path.

Alternatively, you can download and unpack the MAT-files by entering this code at the MATLAB command prompt.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

If you have an already customized link budget inputs/outputs, including availability will restore the link budget to its default inputs/outputs. A different dialog box opens up in this case where you can choose to either cancel or consent to the inclusion of ITU-R P.618 propagation losses for availability analysis.

This figure shows the updates to the configuration in the **Link Budget** (tags `N6`, `N7`, and `N8`) and **Link** (tag `PL5`) tabs, after the MAT-files are added to the MATLAB path.



Link Budget

Tag	Name	L1	L2	L3
N1	Distance (km)	3.7865e+03	4.0215e+04	3.6138e+04
N2	Elevation (deg)	18.3253	13.5231	41.5653
N3	Tx EIRP (dBW)	32	46	47
N4	Polarization loss (dB)	3.0103	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634	206.5296
N6	Rain attenuation (dB)	4.9145	13.0271	0
N7	Total atmospheric losses (dB)	6.4991	15.2073	0
N8	Total propagation losses (dB)	193.4342	220.5707	206.5296
N9	Received isotropic power (dBW)	-167.4445	-180.5810	-165.5399
N10	C/No (dB-Hz)	84.1546	71.0181	86.0592
N11	C/N (dB)	16.3731	3.2366	18.2777
N12	Received Eb/No (dB)	14.1546	1.0181	16.0592
N13	Margin (dB)	2.1546	-10.9819	4.0592

Ground Station | Link | Satellite

Name L1
Type Link
From G1
To S1

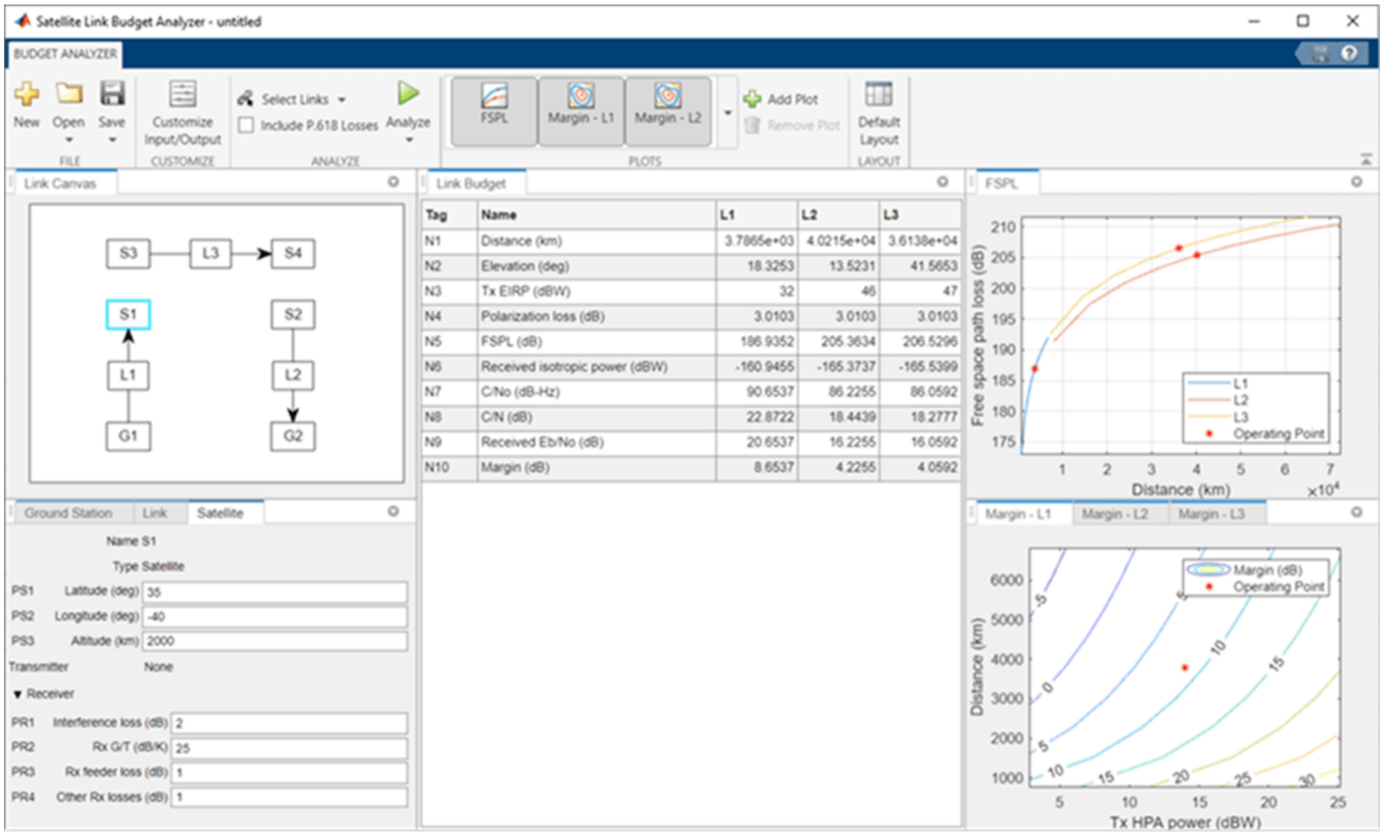
PL1	Frequency (GHz)	14
PL2	Bandwidth (MHz)	6
PL3	Bit rate (Mbps)	10
PL4	Required Eb/No (dB)	10
PL5	Availability (%)	99.9
PL6	Polarization mismatch (deg)	45
PL7	Implementation loss (dB)	2
PL8	Antenna mispointing loss (dB)	1

In this app, `Total atmospheric losses` (tag N7) calculation with P.618 propagation model assumes the antenna type as parabolic.

Customize Inputs and Outputs

Customize the **Properties** and **Results** tabs in the **Satellite Link Budget Analyzer** app using the **Customize Input/Output** tab.

Open the **Satellite Link Budget Analyzer** app. These figures show the default configuration on the **Budget Analyzer** and **Customize Input/Output** tabs.



Tag	Name	L1	L2	L3
N1	Distance (km)	3.7865e+03	4.0215e+04	3.6138e+04
N2	Elevation (deg)	18.3253	13.5231	41.5653
N3	Tx EIRP (dBW)	32	46	47
N4	Polarization loss (dB)	3.0103	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634	206.5296
N6	Received isotropic power (dBW)	-160.9455	-165.3737	-165.5399
N7	C/No (dB-Hz)	90.6537	86.2255	86.0592
N8	C/N (dB)	22.8722	18.4439	18.2777
N9	Received Eb/No (dB)	20.6537	16.2255	16.0592
N10	Margin (dB)	8.6537	4.2255	4.0592

Ground Station | Link | Satellite

Name S1

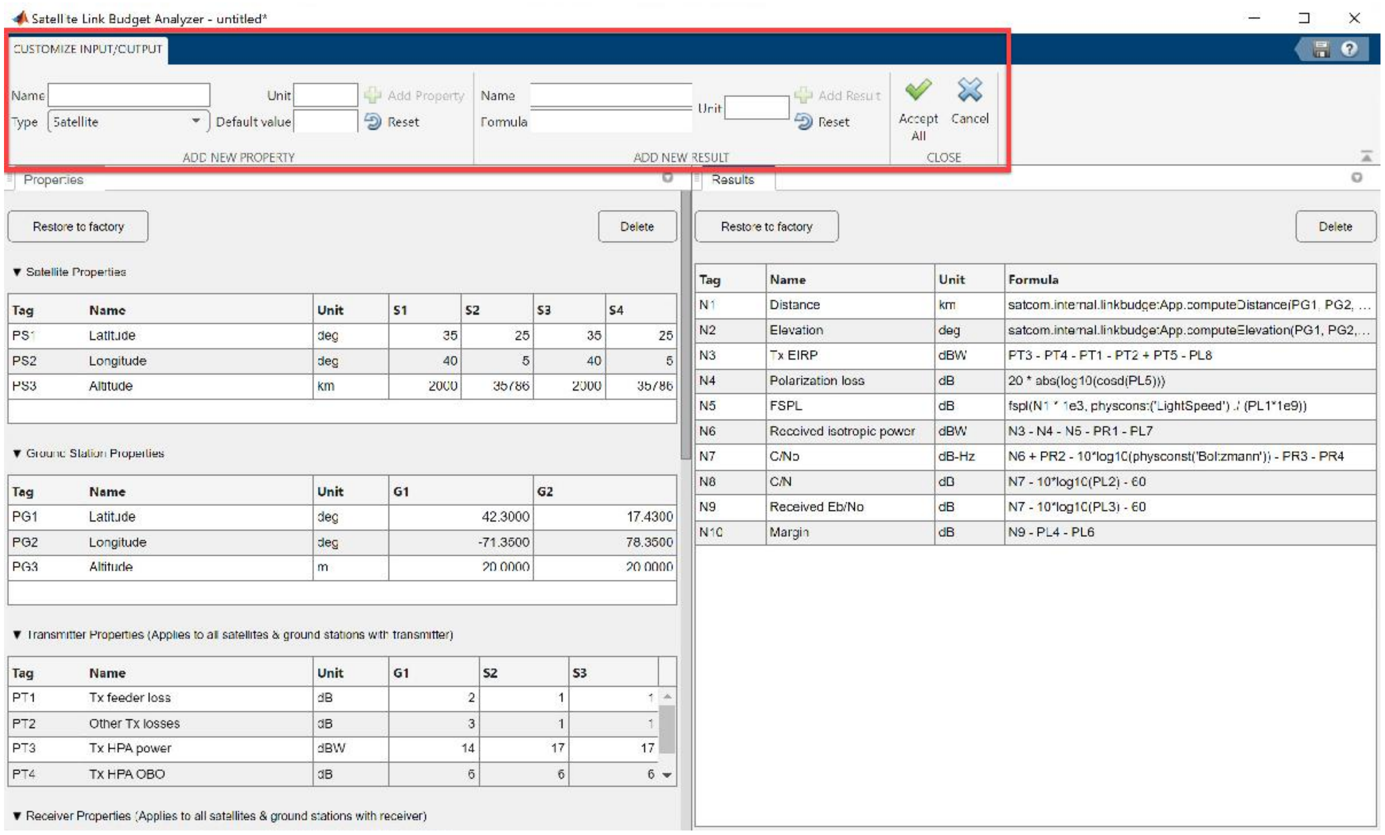
Type Satellite

PS1	Latitude (deg)	35
PS2	Longitude (deg)	-40
PS3	Altitude (km)	2000

Transmitter None

▼ Receiver

PR1	Interference loss (dB)	2
PR2	Rx G/T (dB/K)	25
PR3	Rx feeder loss (dB)	1
PR4	Other Rx losses (dB)	1



▼ Satellite Properties

Tag	Name	Unit	S1	S2	S3	S4
PS1	Latitude	deg	35	25	35	25
PS2	Longitude	deg	40	5	40	5
PS3	Altitude	km	2000	35786	2000	35786

▼ Ground Station Properties

Tag	Name	Unit	G1	G2
PG1	Latitude	deg	42.3000	17.4300
PG2	Longitude	deg	-71.3500	78.3500
PG3	Altitude	m	20.0000	20.0000

▼ Transmitter Properties (Applies to all satellites & ground stations with transmitter)

Tag	Name	Unit	G1	S2	S3
PT1	Tx feeder loss	dB	2	1	1
PT2	Other Tx losses	dB	3	1	1
PT3	Tx HPA power	dBW	14	17	17
PT4	Tx HPA OBO	dB	6	6	6

▼ Receiver Properties (Applies to all satellites & ground stations with receiver)

Tag	Name	Unit	Formula
N1	Distance	km	satcom.internal.linkbudgetApp.computeDistance(PG1, PG2, ...
N2	Elevation	deg	satcom.internal.linkbudgetApp.computeElevation(PG1, PG2, ...
N3	Tx EIRP	dBW	PT3 - PT4 - PT1 - PT2 + PT5 - PL8
N4	Polarization loss	dB	20 * abs(log10(cosd(PL5)))
N5	FSPL	dB	fspl(N1 * 1e3, physconst('LightSpeed') ./ (PL1*1e9))
N6	Received isotropic power	dBW	N3 - N4 - N5 - PR1 - PL7
N7	C/No	dB-Hz	N6 + PR2 - 10*log10(physconst('Boltzmann')) - PR3 - PR4
N8	C/N	dB	N7 - 10*log10(PL2) - 60
N9	Received Eb/No	dB	N7 - 10*log10(PL3) - 60
N10	Margin	dB	N9 - PL4 - PL6

On the **Customize Input/Output** tab:

- Use the options in the **Add New Property** section to add new properties.
- Use the options in the **Add New Result** section to add new results.
- Use the buttons in the **Close** section to accept or cancel the changes.

To delete a property or result, select it and click **Delete** in the respective section.

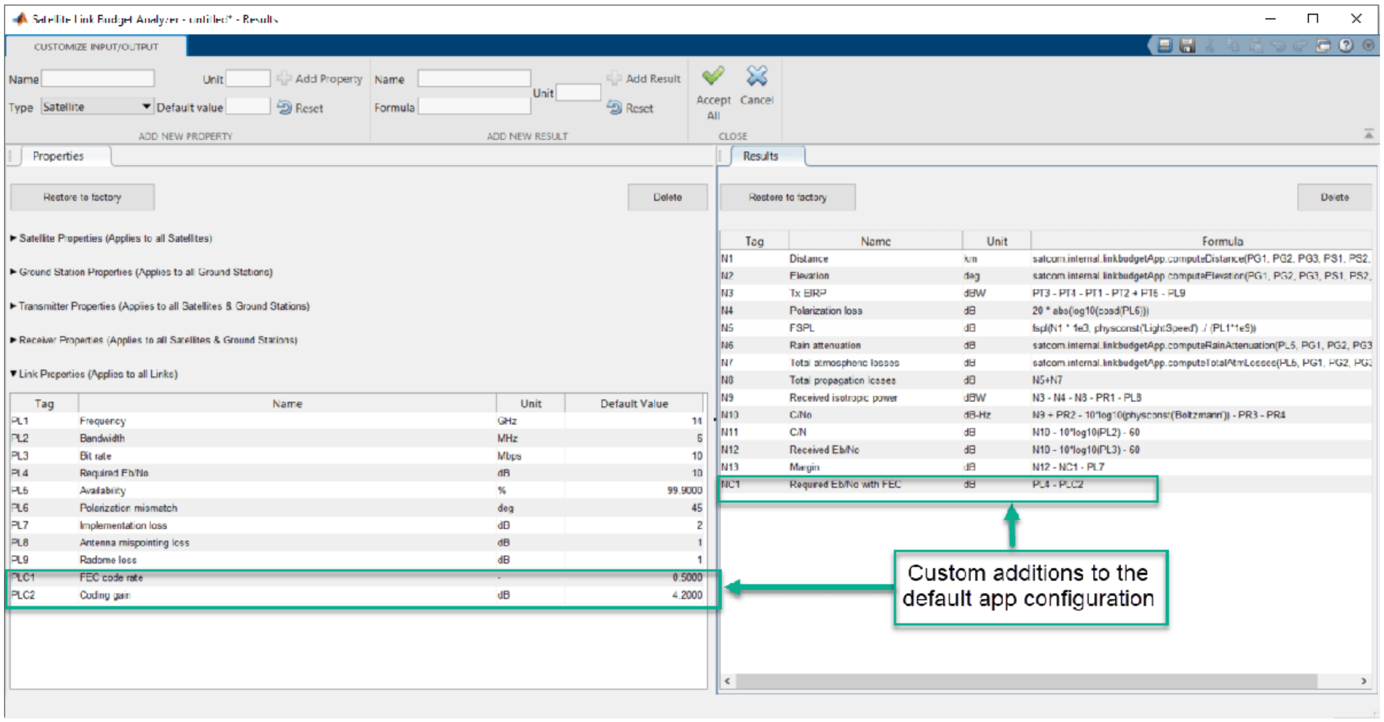
Add Customized Properties and Results

Add customized properties and results by following these steps.

1. Add a new link property, `FEC code rate`. In the **Add New Property** section of the **Customize Input/Output tab**, select `Link` from the **Type** list. In the **Unit** box, type `-`. In the **Default value** box, type `0.5`. Click **Add Property**. The **Link Properties** section of the **Properties** tab now includes `FEC code rate` (tag `PLC1`).
2. Add another link property, `Coding gain`. Select `Link` from the **Type** list. In the **Unit** box, type `dB`. In the **Default value** box, type `4.2`. Click **Add Property**. The **Link Properties** section of the **Properties** tab now includes `Coding gain` (tag `PLC2`).
3. Add a new result, `Required Eb/No with FEC`. In the **Add New Result** section of the **Customize Input/Output tab**, type `PL4 - PLC2` (`Required Eb/No` - `Coding gain`) in the **Formula** box. In the **Unit** box, type `dB`. Click **Add Result**. The **Results** tab now includes `Required Eb/No with FEC` (tag `NC1`).
4. The formula for `Margin` (tag `N13`) on the **Results** tab is changed to use `NC1` instead of `PL4`.

5 In the **Close** section of the app toolstrip, accept all the changes.

This figure shows these updates in the **Properties** and **Results** tabs.



▸ Satellite Properties (Applies to all Satellites)

▸ Ground Station Properties (Applies to all Ground Stations)

▸ Transmitter Properties (Applies to all Satellites & Ground Stations)

▸ Receiver Properties (Applies to all Satellites & Ground Stations)

▾ Link Properties (Applies to all Links)

Tag	Name	Unit	Default Value
PL1	Frequency	GHz	14
PL2	Bandwidth	MHz	6
PL3	Bit rate	Mbps	10
PL4	Required Eb/No	dB	10
PL5	Availability	%	99.9000
PL6	Polarization mismatch	deg	45
PL7	Implementation loss	dB	2
PL8	Antenna mispointing loss	dB	1
PL9	Radome loss	dB	1
PLC1	FEC code rate	-	0.5000
PLC2	Coding gain	dB	4.2000

Tag	Name	Unit	Formula
N1	Distance	km	satcom.internal.linkbudgetApp.computeDistance(PG1, PG2, PG3, PS1, PS2,
N2	Elevation	deg	satcom.internal.linkbudgetApp.computeElevation(PG1, PG2, PG3, PS1, PS2,
N3	Tx EIRP	dBW	PT3 - PT4 - PT1 - PT2 + PT5 - PL9
N4	Polarization loss	dB	20 * abs(log10(cosd(PL6)))
N5	FSPL	dB	fspl(N1 * 1e3, physconst('LightSpeed') ./ (PL1*1e9))
N6	Rain attenuation	dB	satcom.internal.linkbudgetApp.computeRainAttenuation(PL5, PG1, PG2, PG3
N7	Total atmospheric losses	dB	satcom.internal.linkbudgetApp.computeTotalAtmLosses(PL5, PG1, PG2, PG3
N8	Total propagation losses	dB	N5+N7
N9	Received isotropic power	dBW	N3 - N4 - N8 - PR1 - PL8
N10	C/No	dB-Hz	N9 + PR2 - 10*log10(physconst('Boltzmann')) - PR3 - PR4
N11	C/N	dB	N10 - 10*log10(PL2) - 60
N12	Received Eb/No	dB	N10 - 10*log10(PL3) - 60
N13	Margin	dB	N12 - NC1 - PL7
NC1	Required Eb/No with FEC	dB	PL4 - PLC2

Delete Existing Results

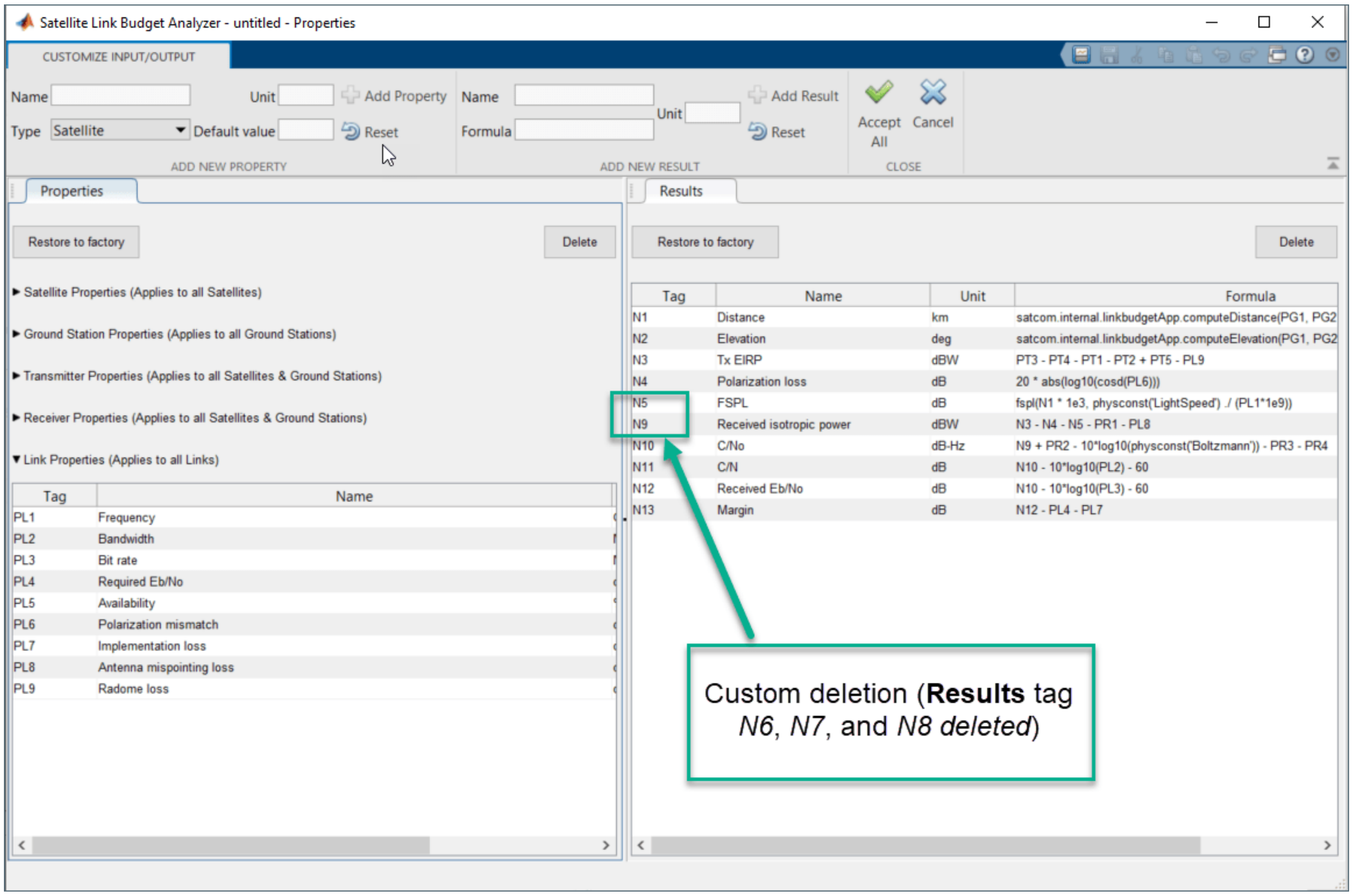
Delete existing link analysis results by following these steps.

1 In the **Results** tab, select `Rain attenuation` (tag N6) and click **Delete** in this tab. Repeat this process for `Total atmospheric losses` (tag N7) and `Total propagation losses` (tag N8).

2 The formula for `Received isotropic power` (tag N9) on the **Results** tab is changed to use N5 instead of N8.

3 In the **Close** section of the app toolstrip, accept all the changes.

This figure shows these updates in the **Results** tab.


Satellite Link Budget Analyzer - untitled - Properties
CUSTOMIZE INPUT/OUTPUT
Name
Type
Satellite
Unit
Default value
Add Property
Reset
ADD NEW PROPERTY
Name
Formula
Unit
Add Result
Reset
ADD NEW RESULT
Accept All
Cancel
CLOSE
Properties
Restore to factory
Delete
Satellite Properties (Applies to all Satellites)
Ground Station Properties (Applies to all Ground Stations)
Transmitter Properties (Applies to all Satellites & Ground Stations)
Receiver Properties (Applies to all Satellites & Ground Stations)
Link Properties (Applies to all Links)
Results
Restore to factory
Delete
Custom deletion (**Results** tag *N6*, *N7*, and *N8 deleted*)

Tag	Name
PL1	Frequency
PL2	Bandwidth
PL3	Bit rate
PL4	Required Eb/No
PL5	Availability
PL6	Polarization mismatch
PL7	Implementation loss
PL8	Antenna mispointing loss
PL9	Radome loss

Tag	Name	Unit	Formula
N1	Distance	km	satcom.internal.linkbudgetApp.computeDistance(PG1, PG2
N2	Elevation	deg	satcom.internal.linkbudgetApp.computeElevation(PG1, PG2
N3	Tx EIRP	dBW	PT3 - PT4 - PT1 - PT2 + PT5 - PL9
N4	Polarization loss	dB	20 * abs(log10(cosd(PL6)))
N5	FSPL	dB	fspl(N1 * 1e3, physconst('LightSpeed') ./ (PL1*1e9))
N9	Received isotropic power	dBW	N3 - N4 - N5 - PR1 - PL8
N10	C/No	dB-Hz	N9 + PR2 - 10*log10(physconst('Boltzmann')) - PR3 - PR4
N11	C/N	dB	N10 - 10*log10(PL2) - 60
N12	Received Eb/No	dB	N10 - 10*log10(PL3) - 60
N13	Margin	dB	N12 - PL4 - PL7

Create Custom Plots

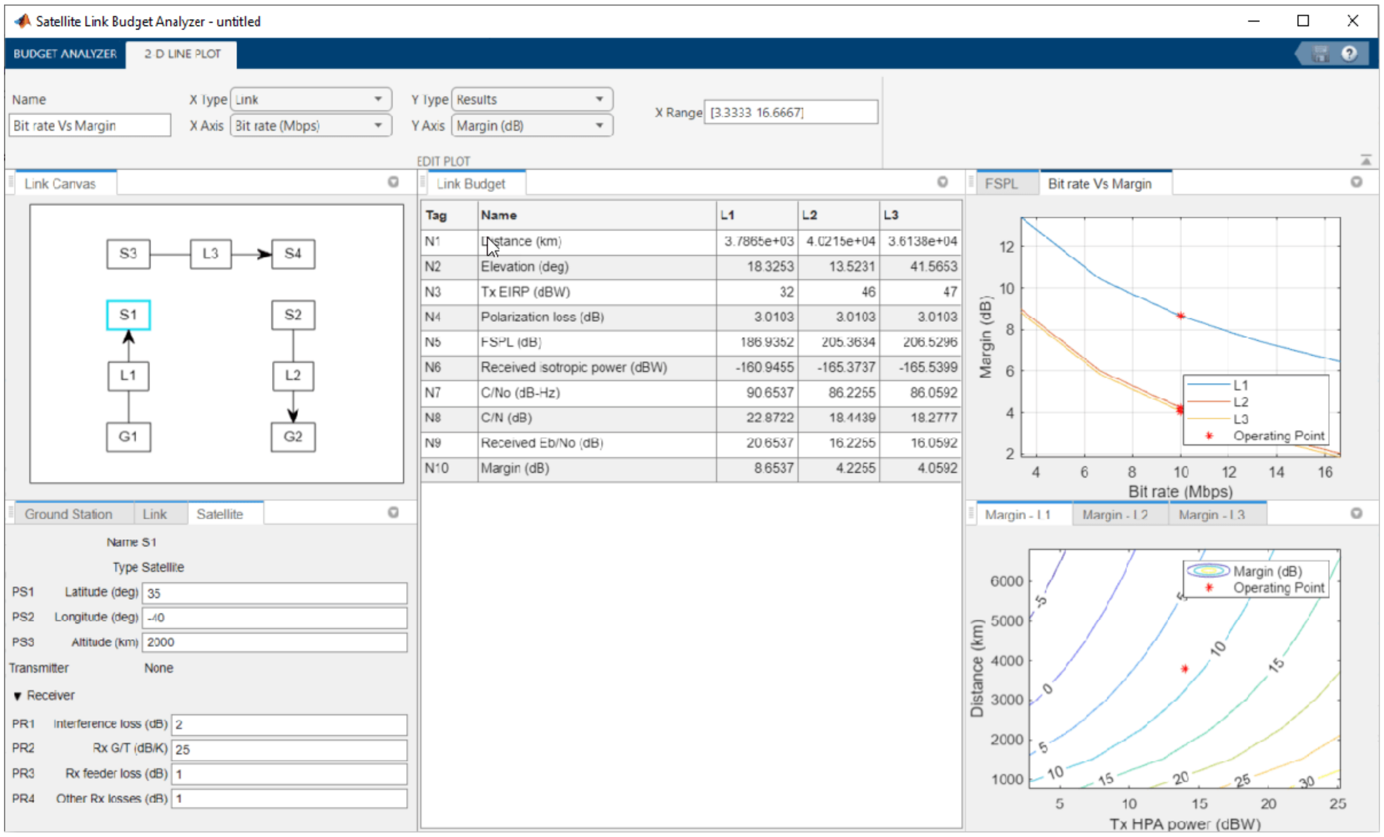
Create custom 2-D line plots in the **Satellite Link Budget Analyzer** app for sensitivity analysis.

Add Custom 2-D Line Plots

Add customized 2-D line plots by following the steps in this example.

1. On the **Budget Analyzer** toolstrip, click the **Add Plot** button in the **Plots** section.
2. In the pop-up window, type `Bit rate Vs Margin` in the **Name** tab. Select `Link` in the **X Type** list and `Bit rate (Mbps)` in the **X Axis** list. Select `Results` in the **Y Type** list and `Margin (dB)` in the **Y Axis** list.
3. Click **Add**.

A new **2-D Line Plot** tab opens up next to the **Budget Analyzer** toolstrip. This figure shows the new custom plot visible in the upper right corner of the app window, next to the default FSPL plot.



Name	Bit rate Vs Margin
X Type	Link
X Axis	Bit rate (Mbps)
Y Type	Results
Y Axis	Margin (dB)
X Range	[3.3333 16.6667]

Link Budget

Tag	Name	L1	L2	L3
N1	Distance (km)	3.7865e+03	4.0215e+04	3.6138e+04
N2	Elevation (deg)	18.3253	13.5231	41.5653
N3	Tx EIRP (dBW)	32	46	47
N4	Polarization loss (dB)	3.0103	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634	206.5296
N6	Received isotropic power (dBW)	-160.9455	-165.3737	-165.5399
N7	C/No (dB-Hz)	90.6537	86.2255	86.0592
N8	C/N (dB)	22.8722	18.4439	18.2777
N9	Received Eb/No (dB)	20.6537	16.2255	16.0592
N10	Margin (dB)	8.6537	4.2255	4.0592

Ground Station | Link | Satellite

	Name	S1
	Type	Satellite
PS1	Latitude (deg)	35
PS2	Longitude (deg)	-40
PS3	Altitude (km)	2000
Transmitter		None
▼ Receiver		
PR1	Interference loss (dB)	2
PR2	Rx G/T (dB/K)	25
PR3	Rx feeder loss (dB)	1
PR4	Other Rx losses (dB)	1

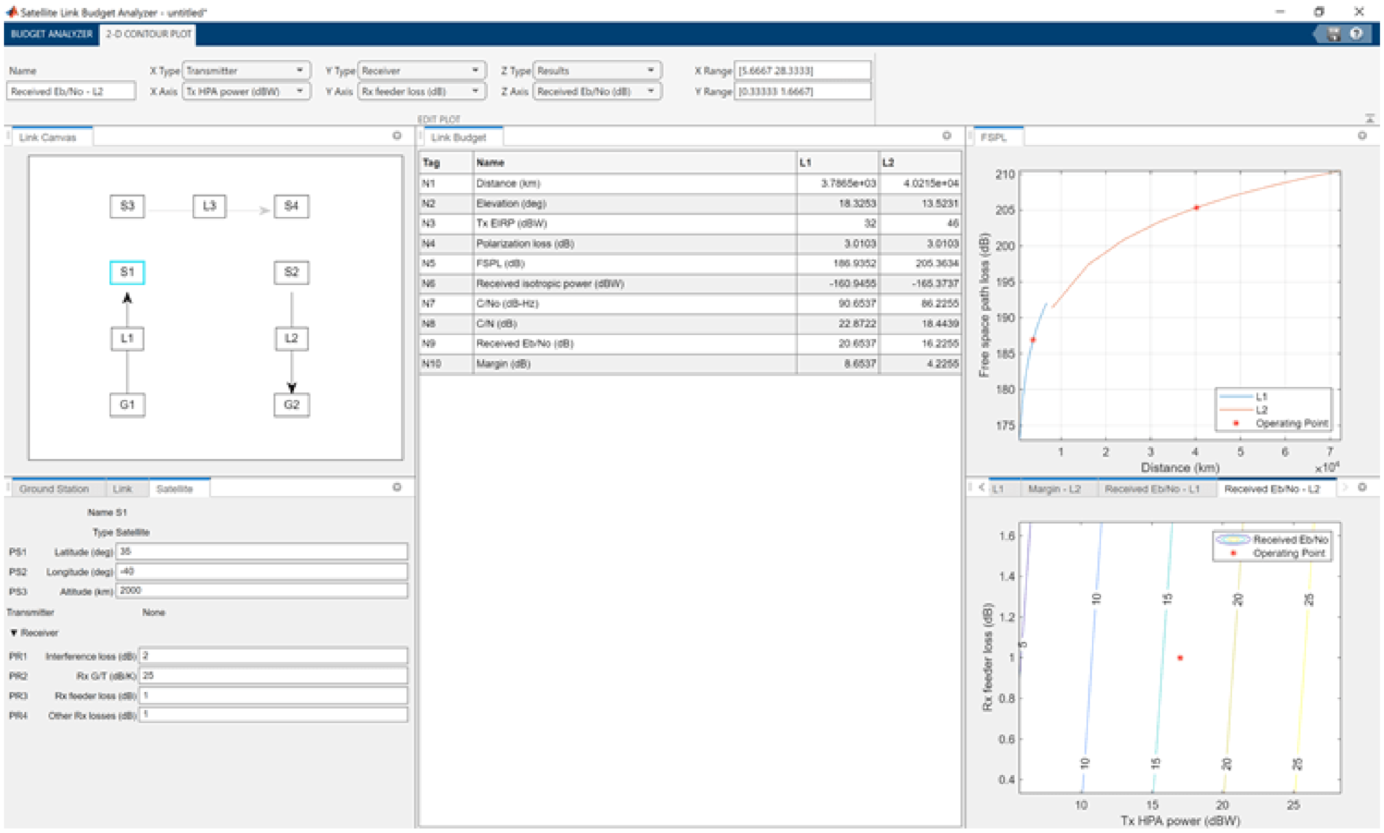
You can directly modify all the tabs (**Name**, **X Type, X Axis**, **Y Type**, **Y Axis**, and **X range**) that appear in the **2-D Line Plot** contextual tab.

Add Custom 2-D Contour Plots

To add customized 2-D contour plots, follow the steps in this example.

1. On the Budget Analyzer toolstrip, in the Analyze section, click **Select Links** and select **L1** and **L2**.
2. In the **Plots** section, click **Add Plot**.
3. In the Add Plot dialog, set type to **2-D Contour Plot.**
4. Specify **Name** as `Received Eb/No`.
5. Set **X Type** to `Transmitter` and **X Axis** to `Tx HPA power (dBW)`.
6. Set **Y Type** to `Receiver` and **Y Axis** to `Rx feeder loss (dB)`.
7. Set **Z Type** to Results and **Z Axis** to `Received Eb/No (dB)`.
8. Click **Add**.

A new **2-D Contour Plot** tab for L2 appears next to the **Budget Analyzer** tab. This figure shows the new custom plots visible in the lower-right pane of the app window, next to the default **Margin - L2** plot.



Name: Received Eb/No - L2
X Type: Transmitter
X Axis: Tx HPA power (dBW)
Y Type: Receiver
Y Axis: Rx feeder loss (dB)
Z Type: Results
Z Axis: Received Eb/No (dB)
X Range: [5.6667 28.3333]
Y Range: [0.33333 1.6667]

Tag	Name	L1	L2
N1	Distance (km)	3.7865e+03	4.0215e+04
N2	Elevation (deg)	18.3253	13.5231
N3	Tx EIRP (dBW)	32	46
N4	Polarization loss (dB)	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634
N6	Received isotropic power (dBW)	-160.9455	-165.3737
N7	C/No (dB-Hz)	90.6537	86.2255
N8	C/N (dB)	22.8722	18.4439
N9	Received Eb/No (dB)	20.6537	16.2255
N10	Margin (dB)	8.6537	4.2255

Name S1
Type Satellite
PS1 Latitude (deg) 35
PS2 Longitude (deg) -40
PS3 Altitude (km) 2000
Transmitter None
▼ Receiver
PR1 Interference loss (dB) 2
PR2 Rx G/T (dB/K) 25
PR3 Rx feeder loss (dB) 1
PR4 Other Rx losses (dB) 1

You can directly modify all the fields (**Name**, **X Type**, **X Axis**, **Y Type**, **Y Axis**, **Z Type**, **Z Axis**, **X Range** and **Y Range**) that appear in the **2-D Contour Plot** tab. To access these fields for L1, in the lower-right pane, select the **Received Eb/No - L1** tab.

Delete Custom Plots

To delete a custom plot, follow these steps.

1. Switch to the **Budget Analyzer** toolstrip.
2. On the toolstrip, click the **Remove Plot** button in the **Plots** section.
3. In the pop-up window, select the custom plot to delete and click **Remove Plot**.

Visibility Control for All Plots

To control the visibility of any default or custom plot, select the specific plot from the drop-down list present in the **Plots** section of the **Budget Analyzer** toolstrip.

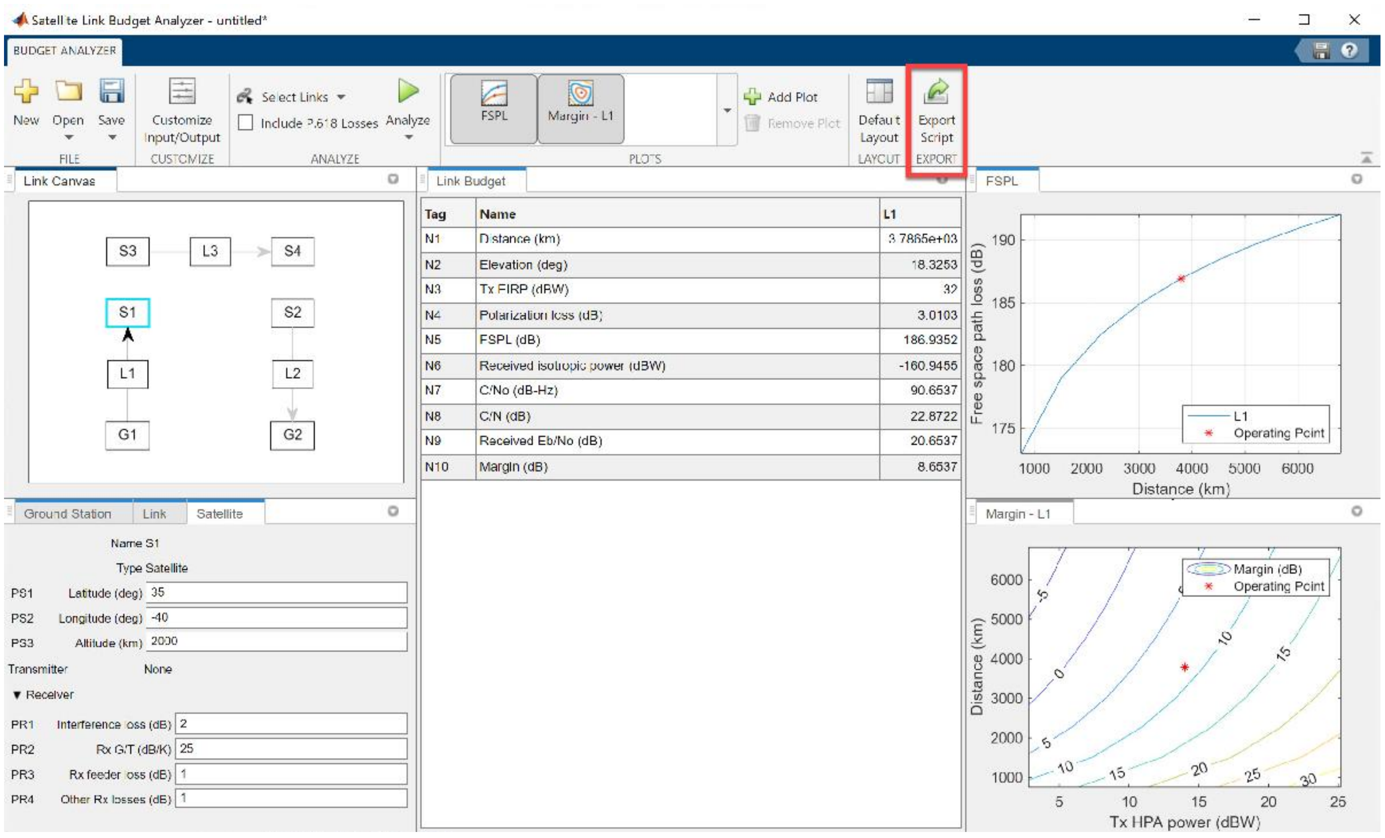
By default, this app selects all plots and makes them visible. To hide any plot, clear it. Reselect it to make it visible again.

If you modify the app layout, you can click the **Default Layout** button on the **Budget Analyzer** toolstrip to return to the default layout.

Note that this app does not save the custom plots for a session.

Generate MATLAB Script

To generate the equivalent MATLAB code for sensitivity analysis, click **Export Script**. You can modify the exported script to study the impact of link budget parameters on link performance.



Tag	Name	L1
N1	Distance (km)	3.7865e+03
N2	Elevation (deg)	18.3253
N3	Tx EIRP (dBW)	32
N4	Polarization loss (dB)	3.0103
N5	FSPL (dB)	186.9352
N6	Received isotropic power (dBW)	-160.9455
N7	C/No (dB-Hz)	90.6537
N8	C/N (dB)	22.8722
N9	Received Eb/No (dB)	20.6537
N10	Margin (dB)	8.6537

Name S1

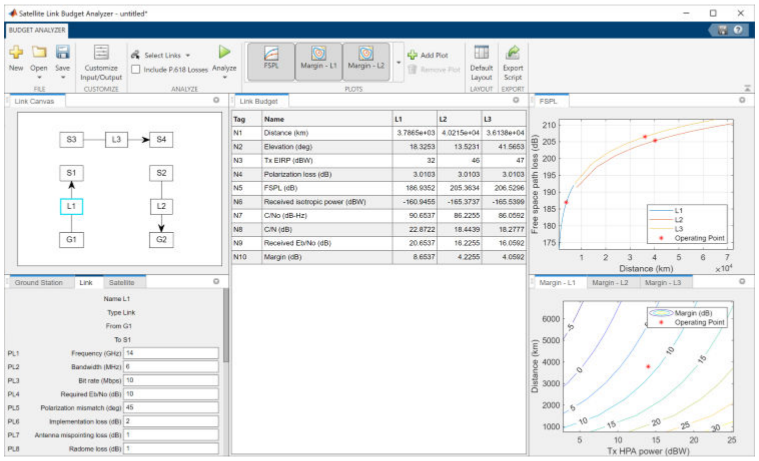
Type Satellite

PS1	Latitude (deg)	35
PS2	Longitude (deg)	-40
PS3	Altitude (km)	2000
Transmitter		None
▼ Receiver		
PR1	Interference loss (dB)	2
PR2	Rx G/T (dB/K)	25
PR3	Rx feeder loss (dB)	1
PR4	Other Rx losses (dB)	1

Parameters

`BUDGET ANALYZER` — Link budget configuration
tab

This figure shows the **BUDGET ANALYZER** tab with the factory default configuration.



Tag	Name	L1	L2	L3
N1	Distance (km)	3.7865e+03	4.0215e+04	3.6138e+04
N2	Elevation (deg)	18.3253	13.5231	41.5653
N3	Tx EIRP (dBW)	32	46	47
N4	Polarization loss (dB)	3.0103	3.0103	3.0103
N5	FSPL (dB)	186.9352	205.3634	206.5296
N6	Received isotropic power (dBW)	-160.9455	-165.3737	-165.5399
N7	C/No (dB-Hz)	90.6537	86.2255	86.0592
N8	C/N (dB)	22.8722	18.4439	18.2777
N9	Received Eb/No (dB)	20.6537	16.2255	16.0592
N10	Margin (dB)	8.6537	4.2255	4.0592

Name L1
Type Link
From G1
To S1

PL1	Frequency (GHz)	14
PL2	Bandwidth (MHz)	6
PL3	Bit rate (Mbps)	10
PL4	Required Eb/No (dB)	10
PL5	Polarization mismatch (deg)	45
PL6	Implementation loss (dB)	2
PL7	Antenna mispointing loss (dB)	1
PL8	Radome loss (dB)	1

Use the **Ground Station**, **Link**, and **Satellite** tabs to adjust property settings for the link budget entities shown in the **Link Canvas** tab.

`Ground Station` — Ground station location, transmitter, and receiver settings
tab

Select the **Ground Station** tab to set the location, transmitter, and receiver settings for the ground station highlighted in the **Link Canvas** tab. For information about customizing satellite, ground station, transmitter, receiver, and link properties, and the link budget result computations, see `CUSTOMIZE INPUT/OUTPUT`.

`Satellite` — Satellite location, transmitter, and receiver settings
tab

Select the **Satellite** tab to set the location, transmitter, and receiver settings for the satellite highlighted in the **Link Canvas** tab. For information about customizing satellite, ground station, transmitter, receiver, and link properties, and the link budget result computations, see `CUSTOMIZE INPUT/OUTPUT`.

`Link` — Link characteristics
tab

Select the **Link** tab to set link characteristics for the link highlighted in the **Link Canvas** tab. For information about customizing satellite, ground station, transmitter, receiver, and link properties, and the link budget result computations, see `CUSTOMIZE INPUT/OUTPUT`.

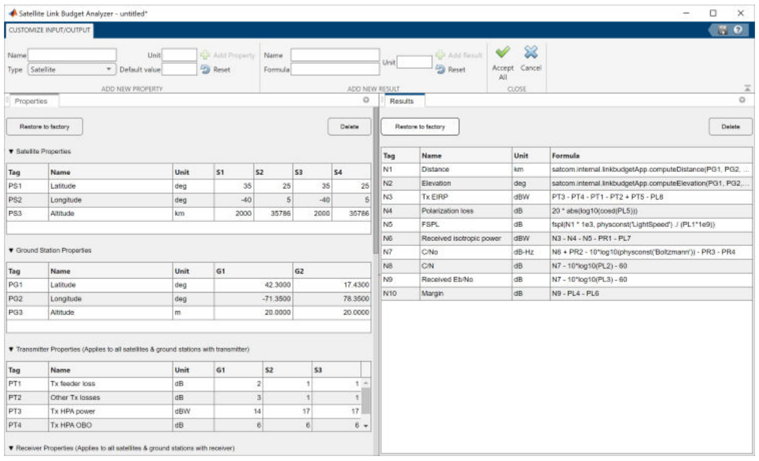
`Customize Input/Output` — Customize input properties and computations used for output
tab

To view or customize input properties and computations used for output, on the **BUDGET ANALYZER** tab, click **Customize Input/Output** to switch to the **CUSTOMIZE INPUT/OUTPUT** tab. In the **CUSTOMIZE INPUT/OUTPUT** tab, you can

- Change settings of the satellite, ground station, transmitter, receiver, and link properties from the factory default inputs
- Add and delete satellite, ground station, transmitter, receiver, and link input properties
- Add, delete, and modify formulas used to compute link budget output results

`CUSTOMIZE INPUT/OUTPUT` — Customize link budget computations
tab

This figure shows the **CUSTOMIZE INPUT/OUTPUT** tab with the factory default configuration.



▼ Satellite Properties

Tag	Name	Unit	S1	S2	S3	S4
PS1	Latitude	deg	35	25	35	25
PS2	Longitude	deg	-40	5	-40	5
PS3	Altitude	km	2000	35786	2000	35786

▼ Ground Station Properties

Tag	Name	Unit	G1	G2
PG1	Latitude	deg	42.3000	17.4300
PG2	Longitude	deg	-71.3500	78.3500
PG3	Altitude	m	20.0000	20.0000

▼ Transmitter Properties (Applies to all satellites & ground stations with transmitter)

Tag	Name	Unit	G1	S2	S3
PT1	Tx feeder loss	dB	2	1	1
PT2	Other Tx losses	dB	3	1	1
PT3	Tx HPA power	dBW	14	17	17
PT4	Tx HPA OBO	dB	6	6	6

▼ Receiver Properties (Applies to all satellites & ground stations with receiver)

Tag	Name	Unit	Formula
N1	Distance	km	satcom.internal.linkbudgetApp.computeDistance(PG1, PG2, ...
N2	Elevation	deg	satcom.internal.linkbudgetApp.computeElevation(PG1, PG2,...
N3	Tx EIRP	dBW	PT3 - PT4 - PT1 - PT2 + PT5 - PL8
N4	Polarization loss	dB	20 * abs(log10(cosd(PL5)))
N5	FSPL	dB	fspl(N1 * 1e3, physconst('LightSpeed') ./ (PL1*1e9))
N6	Received isotropic power	dBW	N3 - N4 - N5 - PR1 - PL7
N7	C/No	dB-Hz	N6 + PR2 - 10*log10(physconst('Boltzmann')) - PR3 - PR4
N8	C/N	dB	N7 - 10*log10(PL2) - 60
N9	Received Eb/No	dB	N7 - 10*log10(PL3) - 60
N10	Margin	dB	N9 - PL4 - PL6

In the **CUSTOMIZE INPUT/OUTPUT** tab, you can

- Use the **Properties** tab to change settings of the satellite, ground station, transmitter, receiver, and link properties from the factory default inputs. You can also add and delete satellite, ground station, transmitter, receiver, and link input properties. On the **Properties** tab, you can use the **Restore to factory** button to load the factory default property configuration in the current app session.
- Use the **Results** tab to add, delete, and modify formulas used to compute link budget output results. On the **Results** tab, you can use the **Restore to factory** button to load the factory default results configuration in the current app session.

Programmatic Use

`satelliteLinkBudgetAnalyzer` opens the **Satellite Link Budget Analyzer** app.

Version History

Introduced in R2021a

See Also

Functions
`fspl` | `satelliteCNR`

Objects
`satelliteScenario` | `satelliteCNRConfig`

Topics
"Get Started with Satellite Link Budget Analyzer App"

Functions

bocmod

Binary offset carrier modulation

Syntax

```
y = bocmod(x,m,n)
y = bocmod(x,m,n,halfcyclesps)
y = bocmod(x,m,n,halfcyclesps,phasing)
```

Description

`y = bocmod(x,m,n)` performs binary offset carrier (BOC) modulation on the input bits `x` by using a square wave and returns the modulated symbols `y`. `m` is the square wave frequency indicator. `n` is the input bit rate indicator.

By default, the phasing of the square wave is set to the phase of the sine curve.

`y = bocmod(x,m,n,halfcyclesps)` specifies the number of samples per half cycle of the square wave.

`y = bocmod(x,m,n,halfcyclesps,phasing)` specifies the phase of the square wave.

Examples

Apply BOC Modulation Using Default Phasing

Generate a random stream of input data bits to modulate.

```
numBits = 5;
bits = randi([0,1],numBits,1);
```

Set the values of `m` and `n` for the subcarrier square wave.

```
m = 2; % Square wave frequency is m*1.023e6 Hz
n = 2; % Square wave input bit rate is n*1.023e6 Hz
```

Modulate the input bits with the square wave using the BOC modulation technique.

```
sym = bocmod(bits,m,n) % Default phasing is of a sine curve
```

```
sym = 20×1

    -1
    -1
     1
     1
    -1
    -1
     1
     1
     1
```

```
        1
        ⋮
```

Apply BOC Modulation Using Cosine Curve Phasing

Generate a random stream of input data bits to modulate.

```
numBits = 10;
bits = randi([0,1],numBits,1);
```

Set the values of `m` and `n` for the subcarrier square wave. Also specify the number of samples per square wave half cycle, `spshc`.

```
m = 5;
n = 2;
spshc = 4;
```

Modulate the input bits with the square wave using the BOC modulation technique.

```
sym = bocmod(bits,m,n,spshc,"cos");
```

Input Arguments

`x` — Input bits
column vector of binary values

Input bits, specified as a column vector of binary values.

The function maps an input bit value of `0` to `+1` and an input bit value of `1` to `-1`. It then multiplies the mapped symbols with a square wave by using the BOC modulation technique.

Data Types: `double` | `int8` | `logical`

`m` — Square wave frequency indicator
positive scalar

Square wave frequency indicator, specified as a positive scalar.

The frequency of square wave is `m*1.023e6` Hz.

Note The value of `2*m/n` must always be an integer. This value represents the number of square wave half cycles per input bit, `x`.

Data Types: `double`

`n` — Input bit rate indicator
positive scalar

Input bit rate indicator, specified as a positive scalar.

The input bit rate is `n*1.023e6` Hz.

Data Types: `double`

`halfcyclesps` — Number of samples per half cycle of the square wave
2 (default) | integer greater than or equal to 2

Number of samples per half cycle of the square wave, specified as an integer greater than or equal to 2.

Data Types: `double` | `uint8`

`phasing` — Phase of square wave
`"sin"` (default) | `"cos"`

Phase of the square wave, specified as `"sin"` or `"cos"`.

- `"sin"` — Set the phase of the square wave to the phase of a sine curve.
- `"cos"` — Set the phase of the square wave to the phase of a cosine curve.

Data Types: `char` | `string`

Output Arguments

y — BOC modulated symbols
column vector

BOC modulated symbols, returned as a column vector. The length of the vector is equal to `length(x)*halfcyclesps*2*m/n`. If you do not specify `halfcyclesps`, the value of `y` is 2 by default.

The data type of the returned modulated symbols is same as that of the input bits, `x`.

Version History

Introduced in R2022a

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`gnssCACode` | `gpsPCode`

ccsdsRSEncode

Encode CCSDS-compliant RS codes

Syntax

```
code = ccsdsRSEncode(msg,k)
code = ccsdsRSEncode(msg,k,i)
code = ccsdsRSEncode(msg,k,i,s)
```

Description

`code = ccsdsRSEncode(msg,k)` encodes the message in `msg` by using a (255, `k`) Reed-Solomon (RS) encoder, as defined in Consultative Committee for Space Data Systems (CCSDS) 131.0-B-3 Section 4 [1]. `k` is the message length. `code` is in dual basis form, as the function assumes that the input to the CCSDS RS encoder is in dual basis form. For more details on dual basis representation, see CCSDS 131.0-B-3 Section 4.4.2 [1].

For a description of CCSDS RS code construction, see “CCSDS RS Code Construction” on page 2-8.

`code = ccsdsRSEncode(msg,k,i)` specifies the interleaving depth, `i`. `msg` consists of `i` RS message symbols of length `k`.

`code = ccsdsRSEncode(msg,k,i,s)` encodes the shortened input message of length `s` with interleaving depth `i`.

Examples

Encode Message Using Full-Length CCSDS RS Encoder

Encode a message using a Consultative Committee for Space Data Systems (CCSDS) Reed-Solomon (RS) encoder.

Specify the message length, `k`, and the interleaving depth, `i`.

```
k = 239;
i = 3;
```

Generate a column vector of random message symbols. The length of the message is product of message length, `k`, and interleaving depth, `i`.

```
msg = randi([0 255],k*i,1);
size(msg)
```

```
ans = 1×2

   717     1
```

Encode the message by using CCSDS RS encoder.

```
code = ccsdsRSEncode(msg,k,i);
```

Verify that the length of the encoded codeword is 255 times the value of the interleaving depth.

```
size(code)
```

```
ans = 1×2

   765     1
```

Encode Shortened Message Using CCSDS RS Encoder

Encode a message using a Consultative Committee for Space Data Systems (CCSDS) Reed-Solomon (RS) encoder with message shortening.

Specify the message length, `k`, interleaving depth, `i`, and the shortened message length, `s`.

```
k = 223;
i = 2;
s = 146;
```

Generate a column vector of random message bits. The length for the shortened message bits is eight times the product of shortened message length, `s`, and the interleaving depth, `i`.

```
msg = logical(randi([0 1],s*i*8,1));
```

Encode the shortened message by using a CCSDS RS encoder.

```
code = ccsdsRSEncode(msg,k,i,s);
```

Verify that the length of the encoded codeword is equal to (8*`i`*(255 – `k` + `s`).

```
size(code)
```

```
ans = 1×2

        2848           1
```

Input Arguments

`msg` — Input message
column vector of logical bits | column vector of integers in the range [0, 255]

Input message, specified as a column vector of logical bits or a column vector of integers in the range [0, 255]. The size of the column vector depends on the data type of the input message.

Input Message Type	Size of `msg`	
	Data Type of `msg` Is logical	Data Type of `msg` Is uint8 or double
Full-length input message	8*`k`	`k`

Input Message Type	Size of msg	
	Data Type of msg Is logical	Data Type of msg Is uint8 or double
Interleaved input message	8*k*i	k*i
Shortened input message	8*s*i	s*i

Data Types: double | uint8 | logical

k — Message length
223 | 239

Message length, specified as 223 or 239.

Data Types: double

i — Interleaving depth
1 (default) | 2 | 3 | 4 | 5 | 8

Interleaving depth, specified as 1, 2, 3, 4, 5, or 8. The default value, 1, corresponds to no interleaving.

msg consists of i RS message symbols of length k.

Data Types: double

s — Shortened message length
k (default) | integer in the range [1, k]

Shortened message length, specified as an integer in the range [1, k].

Data Types: double

Output Arguments

code — CCSDS RS encoded message
column vector

CCSDS RS encoded message, returned as a column vector. The data type of code is same as that of the input message, msg. The size of the column vector depends on the data type of the input message.

Input Message Type	Size of code	
	Data Type of msg Is logical	Data Type of msg Is uint8 or double
Full length input message	8*255	255
Interleaved input message	8*255*i	255*i
Shortened input message	8*i*(255 – k + s)	i*(255 – k + s)

More About

CCSDS RS Code Construction

CCSDS RS codes are powerful burst error-correcting codes used as forward error-correcting (FEC) codes.

The CCSDS RS encoder accepts full-length or shortened messages.

Construction of Full-Length Message CCSDS RS Codes

For full-length input messages the input column vector length is a product of the interleaving depth (*i*) and the message length (*k*).

Encoding in CCSDS RS codes is done row-wise. The encoding results in an *i*-by-*n* vector that includes parity bits added to the end of each row. *n* is the codeword length, which is fixed to 255 symbols according to CCSDS 131.0-B-3 Section 4 [1].

Construction of Shortened Message CCSDS RS Codes

For shortened input messages, the input column vector length is a product of the interleaving depth (*i*) and the shortened message length (*s*). The shortened message vector prepends padding the beginning of the message vector with zeros. The resulting vector is an *i*-by-*k* vector.

Encoding in CCSDS RS codes is done row-wise. The encoding results in an *i*-by-*n* vector that includes parity bits added to the end of each row.

Version History

Introduced in R2021a

References

[1] TM Synchronization and Channel Coding. *Recommendation for Space Data System Standards*. CCSDS 131.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, September 2017.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsRSDecode`

Objects

`ccsdsTMWaveformGenerator` | `comm.RSEncoder`

ccsdsRSDecode

Decode CCSDS-complaint RS codes

Syntax

```
[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k)
[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i)
[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i,s)
```

Description

`[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k)` decode the received signal in `code` by using a (255, `k`) Reed-Solomon (RS) decoder with the generator polynomial, as defined in the Consultative Committee for Space Data Systems (CCSDS) 131.0-B-3 Section 4 [1]. `k` is the number of symbols in the decoded message. The function returns the decoded message `code`, `decoded`, the number of corrected errors, `cnumerr`, and the corrected version of `code`, `ccode`.

For a description of CCSDS RS code decoding, see “CCSDS RS Code Decoding” on page 2-12.

`[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i)` specifies the interleaving depth, `i`. `code` consists of `i` RS codewords of length 255 bytes.

`[decoded,cnumerr,ccode] = ccsdsRSDecode(code,k,i,s)` specifies the shortened message length, `s`.

Examples

Encode and Decode Full-length CCSDS RS Encoded Message

Generate a full-length encoded Reed-Solomon (RS) codeword, introduce random errors, and decode the result using a Consultative Committee for Space Data Systems (CCSDS) RS decoder.

Generate a random message of length `k`.

```
k = 223;
msg = randi([0 255],k,1);
```

Encode the message by using a CCSDS RS encoder.

```
code = ccsdsRSEncode(msg,k);
```

Generate 15 random error symbols and 15 unique random locations to insert these errors.

```
err = randi([1 255],15,1);
errLoc = randperm(255,15);
errVec = zeros(255,1);
errVec(errLoc) = err;
```

Introduce error symbols in the encoded message.

```
rxBytes = bitxor(code,errVec);
```

Decode the encoded symbols introduced with errors by using CCSDS RS decoder.

```
[decoded,v,ccode] = ccsdsRSDecode(rxBytes, k);
```

Display the number of corrected errors.

```
disp(v)
```

```
    15
```

Decode CCSDS RS Codeword with Burst Errors

Generate an full-length encoded Reed-Solomon (RS) codeword, introduce burst of erros, and decode the result using a Consultative Committee for Space Data Systems (CCSDS) RS decoder.

Specify the message length `k` and interleaving depth, `i`.

```
k = 239;
i = 5;
```

Generate a column vector of random message bits. Encode the shortened message by using a CCSDS RS encoder.

```
msg = randi([0 255],k*i,1);
code = ccsdsRSEncode(msg,k,i);
```

Generate 30 random error symbols.

```
err = randi([1 255],30,1);
errVec = zeros(255*i,1);
```

Introduce burst errors from location 52 to 81.

```
errVec(52:81) = err;
rxBytes = bitxor(code,errVec);
```

Decode the encoded symbols introduced with burst errors by using a CCSDS RS decoder.

```
[decoded,v,ccode] = ccsdsRSDecode(rxBytes,k,i);
```

Display the number of corrected errors.

```
disp(v)
```

```
    30
```

Input Arguments

code — Encoded message
column vector of integers in the range [0, 255]

Encoded message, specified as a column vector of integers in the range [0, 255].

The elements and the size of the column vector depends on the data type of the input message.

- For a logical data type, each element in the vector is either 0 or 1.
- For a uint8 or double data type, each element is an integer symbol in GF(2^m), in the range [0, 255]. *m* is the number of bits in each symbol.

Input Message Type	**Size of code**	
	Data Type of code Is logical	**Data Type of code Is uint8 or double**
Full length input message	8*255	255
Interleaved input message	8*255*i	255*i
Shortened input message	8*i*(255 – k + s)	i*(255 – k + s)

Data Types: `double` | `uint8` | `logical`

k — Number of symbols in decoded message
223 | 239

Number of symbols in the decoded message, specified as `223` or `239`.

Data Types: `double`

i — Interleaving depth
1 (default) | 2 | 3 | 4 | 5 | 8

Interleaving depth, specified as `1`, `2`, `3`, `4`, `5`, or `8`. The default value, `1`, corresponds to no interleaving.

`code` consists of `i` RS codewords of length 255 bytes.

Data Types: `double`

s — Shortened message length
`k` (default) | integer in the range [1, `k`]

Shortened message length, specified as an integer in the range [1, `k`].

Data Types: `double`

Output Arguments

decoded — Decoded message
column vector

Decoded message, returned as a column vector. Each element represents decoding the corresponding element in input `code`. The data type of `decoded` is the same as that of `code`.

The size of the column vector depends on the data type of `code`.

Input Message Type	**Size of decoded**	
	Data Type of code Is logical	**Data Type of code Is uint8 or double**
Full length input message	`8*k`	`k`
Interleaved input message	`8*k*i`	`k*i`
Shortened input message	`8*s*i`	`s*i`

When the value of output `cnumerr` is `–1`, `decoded` is equal to the first `k` elements of `code`.

`cnumerr` — Number of corrected errors
integer in the range [`-1`, (*n* – `k`) / 2]

Number of corrected errors, returned as an integer in the range [`-1`, (*n* – `k`) / 2], where *n* is the codeword length. The value of *n* is set to `255` according to CCSDS 131.0-B-3 Section 4 [1].

A value of `–1` in `cnumerr` indicates the failure of the decoder to correct the errors.

`ccode` — Corrected version of code
column vector

Corrected version of code, returned as a column vector. The length of `ccode` is same as the length of `code`. The data type of `ccode` is the same as that of `code`.

When the value of output `cnumerr` is `–1`, `ccode` is equal to `code`.

More About

CCSDS RS Code Decoding

CCSDS RS codes are powerful burst error-correcting codes. These are most commonly used as forward error-correcting (FEC) codes, as they detects and correct errors on the symbol level.

Decoding Full-Length Message CCSDS RS Codes

Like encoding, decoding of CCSDS RS codes is also done row-wise. The input vector length is a product of interleaving depth (*i*) and codeword length (*n*). *n* is fixed to 255 symbols according to CCSDS 131.0-B-3 Section 4 [1]. The input vector is composed of message and parity symbols.

Decoding Shortened Message CCSDS RS Codes

Like encoding, the decoding of CCSDS RS codes is also done row-wise. The input vector length is a product of the interleaving depth (*i*) and the value calculated by *n*-*k*+*s*. The input vector is composed of shortened message and parity symbols.

Version History

Introduced in R2021a

References

[1] TM Synchronization and Channel Coding. *Recommendation for Space Data System Standards*. CCSDS 131.0-B-3. Blue Book. Issue 3. Washington, D.C.: CCSDS, September 2017.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`ccsdsRSEncode`

Objects
`ccsdsTMWaveformGenerator` | `comm.RSDecoder`

ccsdsSCPPMEncode

Encode CCSDS-compliant SCPPM codes

Syntax

```
sym = ccsdsSCPPMEncode(msg,m)
[sym,info] = ccsdsSCPPMEncode(msg,m)
```

Description

`sym = ccsdsSCPPMEncode(msg,m)` encodes the message in `msg` by using a serially concatenated pulse position modulation (SCPPM) encoder, as defined by the Consultative Committee for Space Data Systems (CCSDS) 142.0-B-1 section 3.8 [1]. `m` is the modulation order.

`[sym,info] = ccsdsSCPPMEncode(msg,m)` also returns the outer convolutional encoder information `info`.

Examples

Encode Message for Single Frame Using CCSDS SCPPM Encoder

Encode a message for a single frame using a CCSDS SCPPM encoder.

Use a `comm.CRCGenerator` System object™ to generate CRC code bits and append these bits to the input data.

Note: As specified in CCSDS 142.0-B-1 section 3.6:

- Fix the generator polynomial for the CRC algorithm to `"x^32+x^29+x^18+x^14+x^3+1"`.
- Set the initial states of the internal shift register to `1`.

```
infoSize = 7526;                                    % Information block size without CRC
crc32Generator = comm.CRCGenerator(...
                    Polynomial = "x^32+x^29+x^18+x^14+x^3+1", ...
                    InitialConditions = 1);
crcIn = randi([0 1],infoSize,1);
crcOut = crc32Generator(crcIn);                     % Codeword frame with CRC bits appended
```

Add termination bits to terminate the outer convolutional encoder, as specified in CCSDS 142.0-B-1 section 3.7.

```
msg = [crcOut; 0; 0];
```

Specify the modulation order, and then encode the message using the CCSDS SCPPM encoder.

```
m = 6;
[sym,info] = ccsdsSCPPMEncode(msg,m);
```

Display the outer convolutional encoder information `info`.

```
info
```

```
info = struct with fields:
           OuterEncoderCodeRate: "1/2"
    OuterEncoderPuncturePattern: [1 1 0 1 1 0]
```

Encode Message for Multiple Frames Using CCSDS SCPPM Encoder

Encode a message for multiple frames with constant frame length and modulation order using a CCSDS SCPPM encoder.

Use a `comm.CRCGenerator` System object™ to generate CRC code bits and append these bits to the input data.

Note: As specified in CCSDS 142.0-B-1 section 3.6:

- Fix the generator polynomial for the CRC algorithm to `"x^32+x^29+x^18+x^14+x^3+1"`.
- Set the initial states of the internal shift register to `1`.

```
infoSize = 10046;                        % Information block size without CRC
numFrames = 10;
m =  8;                                  % Modulation order
data = randi([0 1],infoSize,numFrames);
crc32Generator = comm.CRCGenerator(...
           Polynomial = "x^32+x^29+x^18+x^14+x^3+1", ...
           InitialConditions = 1);
sym = zeros(15120/m,numFrames);
```

Generate CRC and add termination bits to encode the message using the CCSDS SCPPM encoder.

```
for frmIdx = 1:numFrames
    crcData = crc32Generator(data(:,frmIdx));   % Generate CRC
    msgIn = [crcData; 0; 0];                    % Add termination bits
    sym(:,frmIdx) = ccsdsSCPPMEncode(msgIn,m);
end
```

Input Arguments

`msg` — Input message
binary column vector

Input message, specified as a binary column vector of length 5040, 7560, or 10080 bits. The input message includes both cyclic redundancy check (CRC) and termination bits, as described in CCSDS 142.0-B-1 sections 3.6 and 3.7 [1].

Data Types: `double` | `int8` | `logical`

`m` — Modulation order
integer in range [2, 8]

Modulation order, specified as an integer in the range [2, 8]. This value specifies the number of bits mapped to one constellation symbol.

Data Types: `double` | `uint8`

Output Arguments

`sym` — CCSDS SCPPM encoded message
column vector

CCSDS SCPPM encoded message, returned as a column vector. The data type of the encoded message depends on the input message data type.

- If `msg` is of data type `double`, `sym` is of data type `double`.
- If `msg` is of data type `logical` or `int8`, `sym` is of data type `uint8`.

The length of this column vector is fixed to 15120 / `m`.

Data Types: `double` | `uint8`

`info` — Outer convolutional encoder information
structure

Outer convolutional encoder information, returned as a structure with these fields.

Structure Field	Description
`OuterEncoderCodeRate`	Outer convolutional encoder code rate, returned as a string scalar.
`OuterEncoderPuncturePattern`	Outer convolutional encoder puncture pattern, returned as a logical vector of these six elements in order: [P_0 P_1 P_2 P_3 P_4 P_5], as described in CCSDS 142.0-B-1 section 3.8.2.3.1 [1].

Data Types: `struct`

Version History
Introduced in R2022b

References

[1] The Consultative Committee for Space Data Systems. *Optical Communications Coding and Synchronization, Recommended Standard, Issue 1.* CCSDS 142.0-B-1. Washington, D.C.: CCSDS, August 2019. https://public.ccsds.org/Pubs/142x0b1.pdf.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also
`ccsdsRSEncode` | `ccsdsRSDecode` | `ccsdsSCPPMDecode`

ccsdsSCPPMDecode

Decode CCSDS-compliant SCPPM codes

Syntax

```
[decoded,crcErr] = ccsdsSCPPMDecode(code,r,m)
[decoded,crcErr] = ccsdsSCPPMDecode(code,r,m,maxIter)
```

Description

`[decoded,crcErr] = ccsdsSCPPMDecode(code,r,m)` decodes the soft bits in `code` using a Consultative Committee for Space Data Systems (CCSDS)-compliant serially concatenated pulse position modulation (SCPPM) decoder, as defined in the Coded Modulation for the Deep-Space Optical Channel, volume 42-161 section III.A [1]. `r` is the outer convolutional encoder code rate and `m` is the modulation order. The function returns the decoded data `decoded` which includes the cyclic redundancy check (CRC) and termination bits. The function also returns the CRC error status `crcErr`.

`[decoded,crcErr] = ccsdsSCPPMDecode(code,r,m,maxIter)` also specifies the maximum number of decoding iterations.

Examples

Transmit and Decode CCSDS SCPPM Encoded Data

Transmit CCSDS SCPPM encoded data over a noiseless channel, and then decode this data using a CCSDS SCPPM decoder with a maximum of eight decoding iterations.

Encode a message using a CCSDS SCPPM encoder for a single frame. Use a `comm.CRCGenerator` System object™ to generate CRC code bits and append these bits to the input data.

Note: As specified in CCSDS 142.0-B-1 section 3.6:

- Fix the generator polynomial for the CRC algorithm to `"x^32+x^29+x^18+x^14+x^3+1"`.
- Set the initial states of the internal shift register to `1`.

```
infoSize = 7526;                       % Information block size without CRC
crc32Generator = comm.CRCGenerator(...
         Polynomial = "x^32+x^29+x^18+x^14+x^3+1", ...
         InitialConditions = 1);
crcIn = randi([0 1],infoSize,1);
crcOut = crc32Generator(crcIn);        % Codeword frame with CRC bits appended
```

Add termination bits to terminate the outer convolutional encoder.

```
msg = [crcOut; 0; 0];
```

Specify the modulation order, and then encode the message using the CCSDS SCPPM encoder.

```
mod = 6;
[sym,info] = ccsdsSCPPMEncode(msg,mod);
```

Display the outer convolutional encoder information `info`.

```
info
```

```
info = struct with fields:
           OuterEncoderCodeRate: "1/2"
    OuterEncoderPuncturePattern: [1 1 0 1 1 0]
```

Modulate the data using the M-ary pulse position modulation (PPM) technique, and pass the modulated data through a noiseless channel.

```
M = 2^mod;                                % M-ary PPM
modData = zeros(length(sym)*M,1);
mapIdx = (0:length(sym)-1)'*M + sym + 1; % Modulate data
modData(mapIdx) = 1;
code = 2*modData - 1;                     % Pass through noiseless channel
```

Set the maximum number of decoding iterations to eight. Decode the encoded SCPPM data.

```
maxIter = 8;
r = info.OuterEncoderCodeRate;
decoded = ccsdsSCPPMDecode(code,r,mod,maxIter);
```

Display the total number of bit errors in the decoded data.

```
fprintf("Number of bit errors = %f\n",sum(msg~=decoded));
```

```
Number of bit errors = 0.000000
```

Input Arguments

code — Encoded soft bits

column vector

Encoded soft bits, specified as a column vector.

The function considers negative soft input bits to be `0`s and positive soft input bits to be `1`s.

Data Types: `double`

r — Outer convolutional encoder code rate

`"1/3"` | `"1/2"` | `"2/3"`

Outer convolutional encoder code rate, specified as `"1/3"`, `"1/2"`, or `"2/3"`.

Data Types: `char` | `string`

m — Modulation order

integer in the range [2, 8]

Modulation order, specified as an integer in the range [2, 8]. This value specifies the number of bits mapped to one constellation symbol.

Data Types: `double` | `uint8`

`maxIter` — Maximum number of decoding iterations
`10` (default) | positive integer

Maximum number of decoding iterations, specified as a positive integer.

Data Types: `double` | `uint8`

Output Arguments

decoded — Decoded information bits
binary column vector

Decoded information bits, returned as a binary column vector. This data includes CRC and termination bits.

Data Types: `int8`

`crcErr` — CRC error status
`false` or `0` | `true` or `1`

CRC error status, returned as one of these numeric or logical values.

- `0` (`false`) — Returned when early termination using CRC passes.
- `1` (`true`) — Returned when CRC fails after completing the maximum number of decoding iterations.

Data Types: `logical`

Version History

Introduced in R2022b

References

[1] Moision, B., and J. Hamkins. "Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation." *The Interplanetary Network Progress Report,* vol. 42-161 (May 15, 2005): 1–25. https://ipnpr.jpl.nasa.gov/progress_report/42-161/161T.pdf.

[2] The Consultative Committee for Space Data Systems. *Optical Communications Coding and Synchronization, Recommended Standard, Issue 1.* CCSDS 142.0-B-1. Washington, D.C.: CCSDS, August 2019. https://public.ccsds.org/Pubs/142x0b1.pdf.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`ccsdsSCPPMEncode` | `ccsdsRSDecode` | `ccsdsRSEncode`

ccsdsTCIdealReceiver

Ideal receiver for CCSDS TC waveform

Syntax

```
bits = ccsdsTCIdealReceiver(waveform,cfg)
bits = ccsdsTCIdealReceiver(waveform,cfg,Name,Value)
```

Description

`bits = ccsdsTCIdealReceiver(waveform,cfg)` recovers transfer frames from a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) `waveform`, generated using the `ccsdsTCWaveform` function. Output `bits` is the recovered bits for the given format configuration `cfg`.

`bits = ccsdsTCIdealReceiver(waveform,cfg,Name,Value)` specifies options using one or more name-value pairs. For example, `'NoiseVariance',1e-11` specifies the noise variance of additive white Gaussian noise (AWGN) on the received waveform as 1e-11.

Examples

Recover Transfer Frame from CCSDS TC Waveform

Recover the transfer frame from the Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) waveform.

Create a CCSDS TC object and specify the object properties.

```
cfg = ccsdsTCConfig;
cfg.HasRandomizer = 1;
cfg.SamplesPerSymbol = 12;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

             DataFormat: "CLTU"
          ChannelCoding: "BCH"
          HasRandomizer: 1
             Modulation: "PCM/PSK/PM"
              PCMFormat: "NRZ-L"
        ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
             SymbolRate: 4000
       SamplesPerSymbol: 12

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Specify the transfer frame length and generate the CCSDS TC waveform for the transfer frame.

```
transferFrameLength = 12; % Number of octets in each transfer frame
data = randi([0 1],8*transferFrameLength,1); % bits in the transfer frame
waveform = ccsdsTCWaveform(data,cfg);
```

Recover the transfer frame from the CCSDS TC waveform

```
decodedBits = ccsdsTCIdealReceiver(waveform,cfg,'DecodingMode',"error detecting");
```

Check if the transfer frame is recovered successfully.

```
rxBits = decodedBits{1};
bits = rxBits((1:8*transferFrameLength)');
isequal(bits,data)
```

```
ans = logical
   1
```

Input Arguments

`waveform` — Received time-domain signal

column vector

Received time-domain signal, consisting of complex in-phase quadrature (IQ) samples, specified as a column vector. The `waveform` input is a CCSDS TC waveform.

A CCSDS TC waveform can contain one or more communications link transmission units (CLTUs). Each CLTU can contain one or more transfer frames.

Data Types: `single` | `double`
Complex Number Support: Yes

`cfg` — Format configuration object

`ccsdsTCConfig` object

Format configuration object, specified as `ccsdsTCConfig` object. The properties of this object determine the parameters required for CCSDS TC waveform generation and reception.

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: `ccsdsTCIdealReceiver(waveform,cfg,'NoiseVariance',1e-11)` specifies the noise variance of AWGN on the received waveform as 1e-11.

`NoiseVariance` — Noise variance of AWGN

`1e-10` (default) | positive scalar

Noise variance of AWGN that is added to the input IQ symbols of the waveform, specified as a positive scalar.

Dependencies

To enable this name-value pair, set the `ChannelCoding` property of the `cfg` input to `"LDPC"`.

Data Types: `double`

`DecodingMode` — Decoding mode
`"error correcting"` (default) | `"error detecting"`

Decoding mode to decode the Bose Chaudhuri Hocquenghem (BCH) encoded codewords, specified as `"error correcting"` or `"error detecting"`.

`'DecodingMode'` defines the allowed number of errors in the start sequence of the CLTU. In error detecting mode, the allowed number of errors in the start sequence is zero. In error correcting mode, the allowed number of errors in the start sequence is one.

Dependencies

To enable this name-value pair, set the `ChannelCoding` property of the `cfg` input to `"BCH"`.

Data Types: `char` | `string`

`DetectionThreshold` — Threshold to detect start sequence
`0.7` (default) | scalar in the range [0.5, 1]

Threshold to detect the start sequence, by calculating the normalized correlation metric with the known start sequence, specified as a scalar in the range [0.5, 1]. When the computed normalized correlation metric is greater than or equal to `'DetectionThreshold'`, the start sequence of the CLTU is detected.

Dependencies

To enable this name-value pair, set the `ChannelCoding` property of the `cfg` input to `"LDPC"`.

Data Types: `double`

Output Arguments

`bits` — Recovered transfer frames
cell array of column vectors

Recovered transfer frames, returned as a cell array of column vectors. Each element of the cell array is of data type `int8`.

Bits in the cell array of one or more column vectors, corresponds to the number of CLTUs present in the `waveform` input. Recovered transfer frames of CLTUs can contain fill bits. The fill bits removal procedure is not performed in the TC synchronization and channel coding sublayer.

Data Types: `int8` | `cell`

Version History
Introduced in R2021a

References

[1] CCSDS 231.0-B-3. Blue Book. Issue 3. "TC Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-29. Blue Book. Issue 29. "Radio Frequency and Modulation Systems - Part 1". *Earth Stations and Spacecraft*. Washington, D.C.: CCSDS, September 2019.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`ccsdsTCWaveform`

Objects
`ccsdsTCConfig`

dvbrcs2TurboEncode

Encode DVB-RCS2-compliant turbo codes

Syntax

```
code = dvbrcs2TurboEncode(msg,r,permparams)
```

Description

`code = dvbrcs2TurboEncode(msg,r,permparams)` encodes the message `msg` by using a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) standard-compliant duo-binary turbo encoder, as defined in ETSI EN 301 545-2 V1.2.1 Section 7.3.5.1 [1]. `r` is the code rate, and `permparams` specifies the permutation control parameters that the function uses to interleave the input message. Output `code` contains the DVB-RCS2-encoded message.

Examples

Encode Message Using DVB-RCS2 Turbo Encoder

Encode a message using a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) duo-binary turbo encoder, with constant code rate and frame length.

Specify the frame length, code rate, and permutation control parameters.

```
frameLen = 40*8;           % Payload length in bits
r = "3/4";
permParams = [17 9 5 14 1];
```

Generate a column vector of random binary data.

```
msg = randi([0 1],frameLen,1);
```

Encode the message by using DVB-RCS2 turbo encoder.

```
code = dvbrcs2TurboEncode(msg,r,permParams);
```

Encode Message Using DVB-RCS2 Turbo Encoder with Variable Code Rates and Frame Lengths

Encode a message using a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) duo-binary turbo encoder, with variable code rates and frame lengths.

Specify the frame lengths, code rates, and permutation control parameters.

```
frameLen = [10*8 100*8 49*8];     % Payload length in bits
r = {'1/3','1/2','2/3'};
permParams = [31 1 3 4 2];
```

Generate the column vectors of binary data and encode the message using DVB-RCS2 turbo encoder.

```
% Initialize output as a 3-by-1 cell array
code = cell(length(r),1);
for frmIdx = 1:length(frameLen)
    msg = randi([0 1],frameLen(frmIdx),1);
    code{frmIdx} = dvbrcs2TurboEncode(msg,r{frmIdx},permParams);
end
```

Input Arguments

`msg` — Input message
binary-valued column vector

Input message, specified as a binary-valued column vector. The length of this column vector must be in the range [1, 65,535] bytes.

Data Types: `double` | `int8` | `logical`

`r` — Code rate
`"1/3"` | `"1/2"` | `"2/3"` | `"3/4"` | `"4/5"` | `"5/6"` | `"6/7"` | `"7/8"`

Code rate, specified as one of these values.

- `"1/3"`
- `"1/2"`
- `"2/3"`
- `"3/4"`
- `"4/5"`
- `"5/6"`
- `"6/7"`
- `"7/8"`

Data Types: `char` | `string`

`permparams` — Permutation control parameters
vector

Permutation control parameters that the function uses to interleave the input message, specified as a vector of these five elements in order: P, Q_0, Q_1, Q_2, and Q_3. P must be in the range [9, 255], and Q_0, Q_1, Q_2, and Q_3 must be in the range [0, 15].

To generate unique interleaver indices, the value of P must be coprime to half of the length of the input `msg`.

Data Types: `double` | `uint8`

Output Arguments

`code` — DVB-RCS2-encoded message
binary-valued column vector

DVB-RCS2-encoded message, returned as a binary-valued column vector. The data type of the `code` is same as that of the input `msg`.

Data Types: `double` | `int8` | `logical`

Version History

Introduced in R2021b

References

[1] EN 301 545-2 - V1.2.1. *Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard (etsi.org).*

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dvbrcs2TurboDecode`

Objects
`dvbrcs2WaveformGenerator` | `comm.TurboEncoder`

dvbrcs2TurboDecode

Decode DVB-RCS2-compliant turbo codes

Syntax

```
decoded = dvbrcs2TurboDecode(code,r,permparams)
decoded = dvbrcs2TurboDecode(code,r,permparams,numiter)
```

Description

`decoded = dvbrcs2TurboDecode(code,r,permparams)` decodes the soft bits in `code` by using a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) standard-compliant duo-binary turbo decoder, as defined in ETSI EN 301 545-2 V1.2.1 Section 7.3.5.1 [1]. `r` is the code rate, and `permparams` are the permutation control parameters that the function uses to interleave the input soft bits data.

`decoded = dvbrcs2TurboDecode(code,r,permparams,numiter)` specifies the number of decoding iterations.

Examples

Transmit and Decode DVB-RCS2 Encoded Data

Transmit a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) encoded signal through an additive white Gaussian noise (AWGN) channel, and then decode it using a DVB-RCS2 duo-binary turbo decoder.

Specify the frame length, code rate, and permutation control parameters.

```
frameLen = 100*8;         % Payload length in bits
r = "2/3";
permParams = [37 0 2 0 2];
```

Generate a column vector of random binary data, and then encode the message by using a DVB-RCS2 turbo encoder.

```
msg = randi([0 1],frameLen,1);
code = dvbrcs2TurboEncode(msg,r,permParams);
```

Modulate the encoded message, and then pass it through an AWGN channel.

```
modCode = qammod(code,16,'gray', ...
             'InputType','bit', ...
             'UnitAveragePower',true);  % 16QAM Modulation
snrdB = 10;                             % SNR
receivedCode = awgn(modCode,snrdB);
```

Demodulated the received signal.

```
noiseVar = 10.^(-snrdB/10);                   % Noise variance
demodLLR = qamdemod(receivedCode,16,'gray', ...
              'OutputType','llr', ...
              'UnitAveragePower',true, ...
              'NoiseVariance',noiseVar);      % 16QAM Demodulation
```

Decode the demodulated soft bits by using a DVB-RCS2 turbo decoder.

```
decoded = dvbrcs2TurboDecode(-1*demodLLR,r, ...
                             permParams);
```

Display the erroneous bits.

```
fprintf('Number of bit errors = %f\n',sum(msg~=decoded))
```

```
Number of bit errors = 0.000000
```

Calculate BER for DVB-RCS2 Encode-Decode Chain

Calculate bit error rate (BER) for a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) encode-decode chain.

Specify the frame length, code rate, and permutation control parameters.

```
frameLen = 25*8;           % Payload length in bits
r = "3/4";
permParams = [19 13 2 9 15];
```

Define the simulation parameters.

```
snrdB = 6;                 % SNR
nVar = 10.^(-snrdB/10);    % Noise variance
errorRate = comm.ErrorRate; % Calculates BER
```

Run the encode-decode chain simulation for 10 frames and calculate the BER.

```
for frmIdx = 1:10
    msg = randi([0 1],frameLen,1);
    code = dvbrcs2TurboEncode(msg,r,permParams);
    modCode = qammod(code,4,[0 2 3 1], ...
                'InputType','bit', ...
                'UnitAveragePower',true);         % QPSK Modulation
    receivedOut = awgn(modCode, snrdB);
    demodOut = qamdemod(receivedOut,4,[0 2 3 1], ...
                'OutputType','llr', ...
                'UnitAveragePower',true, ...
                'NoiseVariance',nVar);            % QPSK Demodulation
    decoded = dvbrcs2TurboDecode(-1*demodOut,r, ...
                             permParams);
    errorStats = errorRate(int8(msg),decoded);
end
```

Display the bit error rate.

```
fprintf('Error rate = %f\n',errorStats(1));
```

```
Error rate = 0.003500
```

```
fprintf('Number of errors detected = %f\n',errorStats(2));
```

```
Number of errors detected = 7.000000
```

```
fprintf('Total bits compared = %f\n',errorStats(3));
```

```
Total bits compared = 2000.000000
```

Input Arguments

code — Encoded soft bits

column vector

Encoded soft bits, specified as a column vector.

Data Types: `double`

r — Code rate

`"1/3"` | `"1/2"` | `"2/3"` | `"3/4"` | `"4/5"` | `"5/6"` | `"6/7"` | `"7/8"`

Code rate, specified as one of these values.

- `"1/3"`
- `"1/2"`
- `"2/3"`
- `"3/4"`
- `"4/5"`
- `"5/6"`
- `"6/7"`
- `"7/8"`

Data Types: `char` | `string`

`permparams` — Permutation control parameters

vector

Permutation control parameters that the function uses to interleave the input soft bits data, specified as a vector of these five elements in order: P, Q_0, Q_1, Q_2, and Q_3. P must be in the range [9, 255], and Q_0, Q_1, Q_2, and Q_3 must be in the range [0, 15].

To generate unique interleaver indices, the value of P must be co-prime to `floor`((*inputmsglen* x `r`)/2). *inputmsglen* is the length of the input message, before encoding.

Data Types: `double` | `uint8`

`numiter` — Number of decoding iterations

8 (default) | positive integer

Number of decoding iterations, specified as a positive integer.

Data Types: `double` | `uint8`

Output Arguments

decoded — Decoded message
binary-valued column vector

Decoded message, returned as a binary-valued column vector.

Data Types: `int8`

Version History

Introduced in R2021b

References

[1] EN 301 545-2 - V1.2.1. *Digital Video Broadcasting (DVB); Second Generation DVB Interactive Satellite System (DVB-RCS2); Part 2: Lower Layers for Satellite standard (etsi.org).*

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dvbrcs2TurboEncode` | `dvbrcs2BitRecover`

Objects
`dvbrcs2RecoveryConfig` | `comm.TurboDecoder`

dvbs2BitRecover

Recover bits for DVB-S2 PL frames

Syntax

```
[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR)
[BITS,NUMFRAMESLOST,PKTCRCSTATUS] = dvbs2BitRecover(RXFRAME,NVAR)
[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR,EARLYTERM)
```

Description

`[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR)` recovers user packets (UPs) or a continuous data stream, `BITS`, and the number of lost baseband frames, `NUMFRAMESLOST`. Input `RXFRAME` is the received complex in-phase quadrature (IQ) symbols in the form of physical layer (PL) frames of a Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission. Input `NVAR` is the noise variance estimate, used to calculate soft bits.

`[BITS,NUMFRAMESLOST,PKTCRCSTATUS] = dvbs2BitRecover(RXFRAME,NVAR)` also returns the UP cyclic redundancy check (CRC) status.

`[BITS,NUMFRAMESLOST] = dvbs2BitRecover(RXFRAME,NVAR,EARLYTERM)` uses low-density parity-check (LDPC) decoding termination criterion, `EARLYTERM`, to recover data bits, `BITS`.

Examples

Recover Data Bits from Transport Stream DVB-S2 Transmission

Recover user packets (UPs) for multiple physical layer (PL) frames in a single transport stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream. Create a DVB-S2 System object.

```
nFrames = 2;
s2WaveGen = dvbs2WaveformGenerator;
```

Create the bit vector of information bits, `data`, of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
```

```
numPkts = s2WaveGen.MinNumPackets*nFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate the DVB-S2 time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 1;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor',sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs. Display the number of frames lost and the UP cyclic redundancy check (CRC) status.

```
[bits,FramesLost,pktCRCStat] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10));
disp(FramesLost)
```

```
     0
```

```
disp(pktCRCStat)
```

```
    {20×1 logical}
```

Recover Data Bits from Generic Stream DVB-S2 Transmission with Early Termination Enabled

Recover user bits in a multi-input generic stream (GS) Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission with variable modulation and coding scheme.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
nFrames = 1;
```

Create a DVB-S2 System object with variable coding and modulation configuration for a multi-input GS. Specify the modulation scheme and forward error correction (FEC) rate (MODCOD) and the data field length (DFL).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.StreamFormat = "GS";
s2WaveGen.NumInputStreams = 3;
s2WaveGen.MODCOD = [10 15 6];    % QPSK 8/9, 8PSK 5/6, and QPSK 2/3
s2WaveGen.DFL = [44500 51387 42960];
```

Create a bit vector of input information bits for each input stream.

```
data =  cell(s2WaveGen.NumInputStreams,1);
for i = 1:s2WaveGen.NumInputStreams
   data{i} = randi([0 1],s2WaveGen.DFL(i)*nFrames,1);
end
```

Generate the DVB-S2 time-domain waveform with the input information bits. Flush the transmit filter to handle the filter delay and recover the complete frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform. Specify the samples per symbol for the baseband filter.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 10;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor',sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover user bits. Enable early termination of the low-density parity-codes (LDPC) decoder.

```
[bits,FramesLost] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10),1);
```

Display the number of frames lost and the number of bit errors in each stream.

```
fprintf('Number of frames lost = %d\n',FramesLost)
```

```
Number of frames lost = 0
```

```
for i = 1:s2WaveGen.NumInputStreams
        fprintf('Number of bit errors in stream %d = %d\n',i, ...
            sum(data{i}~=bits{i}))
end
```

```
Number of bit errors in stream 1 = 0
Number of bit errors in stream 2 = 0
Number of bit errors in stream 3 = 0
```

Recover Data Bits from Transport Stream DVB-S2 Transmission with ISSYI Enabled

Recover user packets (UPs) in a multi-input transport stream (TS) Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission with constant coding and modulation.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 1;
```

Create a DVB-S2 System object with constant coding and modulation configuration for a multi-input TS. Specify a short forward error correction (FEC) frame format and enable the input stream synchronization (ISSY).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.NumInputStreams = 3;
s2WaveGen.FECFrame = "short";
s2WaveGen.MODCOD = 10;       % QPSK 8/9
s2WaveGen.DFL   = 13920;
s2WaveGen.ISSYI = true;
```

Create a bit vector of information bits of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';      % Sync byte for TS packet is 47 Hex
pktLen = 1496;                      % UP length without sync bits is 1496
data =  cell(1,s2WaveGen.NumInputStreams);
for i = 1:s2WaveGen.NumInputStreams
   numPkts = s2WaveGen.MinNumPackets(i)*numFrames;
   txRawPkts = randi([0 1],pktLen,numPkts);
   ISSY = randi([0 1],16,numPkts);   % ISCRFormat is 'short' by default
                                     % 'short' implies the default length of ISSY as 2 bytes
   txPkts = [repmat(syncBits,1,numPkts); txRawPkts; ISSY]; % ISSY is appended at the end of UP
   data{i} = txPkts(:);
end
```

Generate the DVB-S2 time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform. Specify the samples per symbol for the baseband filter.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 12;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn (txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor', sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs. Display the number of frames lost and the number of bit errors in each stream.

```
[bits,FramesLost,pktCRCStat] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10));
fprintf('Number of frames lost = %d\n',FramesLost)
```

```
Number of frames lost = 0
```

```
for i = 1:s2WaveGen.NumInputStreams
    fprintf('Number of bit errors in stream %d = %d\n',i, ...
    numel(pktCRCStat{i})-sum(pktCRCStat{i}))
end
```

```
Number of bit errors in stream 1 = 0
Number of bit errors in stream 2 = 0
Number of bit errors in stream 3 = 0
```

Input Arguments

RXFRAME — Received IQ symbols from PL frames of DVB-S2 transmission

column vector

Received IQ symbols from PL frames of a DVB-S2 single-input or multi-input transmission, specified as a column vector. RXFRAME can contain one or multiple PL frames.

The length of RXFRAME depends on the value of the properties FECFrame, MODCOD, and HasPilots of the dvbs2WaveformGenerator System object™.

Data Types: double
Complex Number Support: Yes

`NVAR` — Noise variance estimate
nonnegative scalar

Noise variance estimate that the function adds to the input IQ symbols, specified as a nonnegative scalar. `NVAR` is used as a scaling factor to calculate the soft bits from the IQ symbols.

When you specify `NVAR` as `0`, the function uses a value of 1e-5, which corresponds to a signal-to-noise ratio (SNR) of 50 dB.

Data Types: `double`

`EARLYTERM` — Flag for early termination of LDPC decoder
`0` or `false` (default) | `1` or `true`

Flag for early termination of the LDPC decoder when all parity-checks are satisfied, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). When set to `1` (`true`), the LDPC decoder is terminated when all parity checks are satisfied.

When you set this value to `0` (`false`), the maximum decoding iteration limit is 50.

Data Types: `logical`

Output Arguments

`BITS` — Recovered data bits
cell array of column vectors

Recovered data bits, returned as a cell array of column vectors. Each element of the cell array is of data type `int8`. This output can be either UPs or generic data stream, depending of the `StreamFormat` property of the `dvbs2WaveformGenerator` System object.

For a multi-input stream transmission, each element of the cell array corresponds to an individual input stream.

Data Types: `cell`

`NUMFRAMESLOST` — Number of lost baseband frames
nonnegative integer

Number of lost baseband frames, returned as a nonnegative integer. If the baseband header CRC fails, the frame is considered lost.

Data Types: `double`

`PKTCRCSTATUS` — UP CRC status
cell array of column vectors

UP CRC status, returned as a cell array of column vectors. Each element of the cell array is of data type `logical`. For a multi-input stream transmission, each element of the cell array corresponds to an individual input stream.

Dependencies

`PKTCRCSTATUS` applies for only the input streams where the value of the `UPL` property of `dvbs2WaveformGenerator` System object is nonzero.

Data Types: `cell`

Version History

Introduced in R2021a

References

[1] ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`dvbs2WaveformGenerator`

dvbs2xBitRecover

Recover bits for DVB-S2X PL frames

Syntax

```
bits = dvbs2xBitRecover(rxdata,nvar)
[bits,numFramesLost,pktCRCStatus,decodedParams] = dvbs2xBitRecover(rxdata,
nvar)
[ ___ ] = dvbs2xBitRecover(rxdata,nvar,Name=Value)
```

Description

`bits = dvbs2xBitRecover(rxdata,nvar)` recovers user packets (UPs) or a continuous bit stream, `bits`. Input `rxdata` is the received complex in-phase quadrature (IQ) symbols in the form of physical layer (PL) frames of a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) single or multi-input stream transmission. Input `nvar` is the noise variance estimate, which the function uses to calculate soft bits.

`[bits,numFramesLost,pktCRCStatus,decodedParams] = dvbs2xBitRecover(rxdata, nvar)` also returns the number of frames lost `numFramesLost` UP cyclic redundancy check (CRC) status `pktCRCStatus` and a structure `decodedParams` in which the fields indicate the parameters of the processed PL frames.

`[ ___ ] = dvbs2xBitRecover(rxdata,nvar,Name=Value)` specifies one or more optional name-value arguments. For example, `Mode="wideband"` sets the mode of operation for the bit recovery to the wideband frame processing mode.

Examples

Recover Data Bits from Transport Stream DVB-S2X Transmission in Regular Mode

Recover UPs for a given PL frame in a single transport stream (TS) DVB-S2X transmission in regular mode.

This example uses MAT files with LDPC parity matrices. If the MAT files are not available on the path, download and unzip the MAT files by entering this code at the MATLAB command line.

```
if ~exist("dvbs2xLDPCParityMatrices.mat","file")
    if ~exist("s2xLDPCParityMatrices.zip","file")
        url = "https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip";
        websave("s2xLDPCParityMatrices.zip",url);
        unzip("s2xLDPCParityMatrices.zip");
    end
addpath("s2xLDPCParityMatrices");
end
```

Specify the number of PL frames per stream and set the baseband filtering at 4 samples per symbol.

```
nFrames = 1;
s2xWaveGen = dvbs2xWaveformGenerator;
sps = 4;
```

Create a bit vector of information bits, `data`, of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';                  % Sync byte for TS packet is 47 Hex
pktLen = 1496;                                  % UP length without sync bits is 1496
numPkts = s2xWaveGen.MinNumPackets*nFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate the DVB-S2X time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2xWaveGen(data); flushFilter(s2xWaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
EsNodB = 5;
snrdB = EsNodB - 10*log10(sps);
rxWaveform = awgn(txWaveform,snrdB,"measured");
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
       "RolloffFactor",s2xWaveGen.RolloffFactor, ...
       "InputSamplesPerSymbol",sps,"DecimationFactor",sps);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxWaveform);
rxSymb = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs.

```
[rxOut,numFramesLost] = dvbs2xBitRecover(rxSymb,10^(-EsNodB/10));
```

Display the number of frames lost and the UP cyclic redundancy check (CRC) status.

```
fprintf("numFramesLost = %d\n",numFramesLost)
```

```
numFramesLost = 0
```

```
fprintf("Number of bit errors = %d\n",sum(data~=rxOut{1}))
```

```
Number of bit errors = 0
```

Recover Data Bits from Generic Stream DVB-S2X Transmission in Wideband Mode

Recover user bits in a multi-input generic stream (GS) DVB-S2X transmission, with variable modulation and coding scheme, in wideband mode.

This example uses MAT files with LDPC parity matrices. If the MAT files are not available on the path, download and unzip the MAT files by entering this code at the MATLAB command prompt.

```
if ~exist("dvbs2xLDPCParityMatrices.mat","file")
    if ~exist("s2xLDPCParityMatrices.zip","file")
        url = "https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip";
        websave("s2xLDPCParityMatrices.zip",url);
        unzip("s2xLDPCParityMatrices.zip");
    end
    addpath("s2xLDPCParityMatrices");
end
```

Specify the number of PL frames per stream and set the baseband filtering at 4 samples per symbol.

```
nFramesPerStream = 2;
sps = 4;
```

Create a DVB-S2X System object™ with variable coding and modulation configuration for a multi-input GS and set its properties.

```
s2xWaveGen = dvbs2xWaveformGenerator;
s2xWaveGen.NumInputStreams = 5;
s2xWaveGen.PLSDecimalCode = [140 132 133 141 132];
s2xWaveGen.DFL = [37168 18448 18448 37168 18448];
s2xWaveGen.StreamFormat = "GS";
s2xWaveGen.HasTimeSlicing = true;
```

Create a bit vector of input information bits for each input stream.

```
data = cell(s2xWaveGen.NumInputStreams,1);
for i = 1:s2xWaveGen.NumInputStreams
  data{i} = randi([0 1],s2xWaveGen.DFL(i)*nFramesPerStream,1);
end
```

Generate the DVB-S2X time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2xWaveGen(data); flushFilter(s2xWaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
EsNodB = 4;
snrdB = EsNodB - 10*log10(sps);
rxWaveform = awgn(txWaveform,snrdB,"measured");
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
    "RolloffFactor",s2xWaveGen.RolloffFactor, ...
    "InputSamplesPerSymbol",sps,"DecimationFactor",sps);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxWaveform);
rxSymb = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Set the time slicing number to the required PL frame value to be recovered. Recover the user bits.

```
tsn = 3;
[rxOut,numFramesLost,pktCRCStat] = dvbs2xBitRecover(rxSymb,10^(-EsNodB/10), ...
                                                     Mode="wideband", ...
                                                     TimeSlicingNumber=tsn);
```

Display the number of frames lost and the UP CRC status.

```
fprintf("numFramesLost = %d\n",numFramesLost)
```

```
numFramesLost = 0
```

```
fprintf("Number of bit errors = %d\n",sum(data{tsn}~=rxOut{1}))
```

```
Number of bit errors = 0
```

Recover Data Bits from Transport Stream DVB-S2X Transmission in VL-SNR Mode

Recover UPs for a given PL frame in a single TS DVB-S2X transmission in VL-SNR mode.

This example uses MAT files with LDPC parity matrices. If the MAT files are not available on the path, download and unzip the MAT files by entering this code at the MATLAB command prompt.

```
if ~exist("dvbs2xLDPCParityMatrices.mat","file")
    if ~exist("s2xLDPCParityMatrices.zip","file")
        url = "https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip";
        websave("s2xLDPCParityMatrices.zip",url);
        unzip("s2xLDPCParityMatrices.zip");
    end
    addpath("s2xLDPCParityMatrices");
end
```

Specify the number of PL frames per stream and set the baseband filtering at 4 samples per symbol.

```
nFrames = 1;
sps = 4;
```

Create a DVB-S2X System object™ for TS and set its properties.

```
s2xWaveGen = dvbs2xWaveformGenerator;
s2xWaveGen.PLSDecimalCode = 129;
s2xWaveGen.NumInputStreams = 1;
s2xWaveGen.DFL = 14128;
s2xWaveGen.PLScramblingIndex =4;
```

Create a bit vector of information bits, `data`, of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';                % Sync byte for TS packet is 47 Hex
pktLen = 1496;                                % UP length without sync bits is 1496
numPkts = s2xWaveGen.MinNumPackets*nFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate the DVB-S2X time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2xWaveGen(data); flushFilter(s2xWaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
EsNodB = -2;
snrdB = EsNodB - 10*log10(sps);
rxWaveform = awgn(txWaveform,snrdB,"measured");
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
    "RolloffFactor",s2xWaveGen.RolloffFactor, ...
    "InputSamplesPerSymbol",sps,"DecimationFactor",sps);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxWaveform);
rxSymb = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs.

```
[rxOut,numFramesLost,pktCRCStat,decodedParams] = dvbs2xBitRecover(rxSymb,10^(-EsNodB/10), ...
                                 Mode="vl-snr",PLScramblingIndex=4);
```

Display the number of frames lost and parameters of the processed PL frame.

```
fprintf("numFramesLost = %d\n",numFramesLost)
```

```
numFramesLost = 0
```

```
disp(decodedParams)
```

```
         PLSDecimalCode: 129
           IsDummyFrame: 0
         BBHeaderStatus: 1
        ModulationOrder: 4
         FECFrameLength: 64800
    CanonicalMODCODName: {'QPSK 2/9'}
              HasPilots: 1
```

Input Arguments

rxdata — Received IQ symbols from PL frames of DVB-S2X transmission

column vector

Received IQ symbols from the PL frames of a DVB-S2X single-input or multi-input transmission, specified as a column vector. `rxdata` can contain one or more PL frames.

The length of `rxdata` depends on the forward error correction (FEC) frame length, modulation order, mode (regular, VL-SNR, or wideband), and presence or absence of pilots specified for the PL frame.

Data Types: `double`
Complex Number Support: Yes

nvar — Noise variance estimate

nonnegative scalar

Noise variance estimate that the function adds to the input IQ symbols, specified as a nonnegative scalar. The function uses `nvar` as a scaling factor to calculate the soft bits from the IQ symbols.

When you specify `nvar` as `0`, the function uses a value of 1e–5, which corresponds to a signal-to-noise ratio (SNR) of 50 dB.

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: `Mode="wideband"` sets the mode of operation for the bit recovery to wideband frame processing mode.

`PLScramblingIndex` — PL scrambling sequence index

`0` (default) | integer in range [0, 7] | vector of integers in range [0, 7]

PL scrambling sequence index, specified as a integer in the range [0, 7] or as a vector of integers in the range [0, 7].

The function uses a gold sequence index of `PLScramblingIndex*10949`, as defined in ETSI EN 302 307-2 section 5.5.4 table 19e [1].

When you specify `Mode` as `"wideband"`, the `PLScramblingIndex` value must be equal to the scrambling sequence index value of the sequence used to scramble the required wideband PL frames.

Data Types: `double`

`EarlyTermination` — Flag for early termination of LDPC decoder

`0` or `false` (default) | `1` or `true`

Flag for early termination of the LDPC decoder when all parity-checks are satisfied, specified as a numeric or logical value of `1` (`true`) or `0` (`false`). When set to `1`, the LDPC decoder is terminated when all parity checks are satisfied.

When you set this value to `0`, the maximum decoding iteration limit is 50.

If the modulation order of the PL frame is 2, the maximum decoding iteration limit is 75.

Data Types: `logical`

`Mode` — Mode of operation for bit recovery

`"regular` (default) | `"vl-snr"` | `"wideband"`

Mode of operation for the bit recovery, specified as one of these values.

- `"regular` — Use this option to set the mode to regular frame processing.
- `"vl-snr"` — Use this option to set the mode to very low signal-to-noise ratio (VL-SNR) frame processing.
- `"wideband"` — Use this option to set the mode to wideband frame processing.

Note For this function, wideband mode does not support VL-SNR frame processing.

Data Types: `char` | `string`

`TimeSlicingNumber` — Time slicing number
`1` (default) | scalar in range [0, 253]

Time slicing number encoded into the PL header of the required wideband frames, specified as a scalar in the range [0, 253].

If the time slicing number decoded from a PL header is not equal to the required time slicing number value, the function discards the PL frame and decodes the next wideband frame PL header.

To use this argument, you must specify `Mode` as `"wideband"`.

Note In wideband mode, the number of streams is considered as number of services, where each stream is considered to be an individual service. The receiver recovers only the service specified by the `TimeSlicingNumber` argument.

Data Types: `double`

Output Arguments

`bits` — Recovered data bits
cell array of column vectors

Recovered data bits, returned as a cell array of column vectors. Each element of the cell array is of data type `int8`. This output can be either UPs or a generic data stream, depending of the StreamFormat property of the `dvbs2xWaveformGenerator` System object.

For a multi-input stream transmission, each element of the cell array corresponds to an individual input stream.

Data Types: `cell`

`numFramesLost` — Number of lost baseband frames
nonnegative integer

Number of lost baseband frames, returned as a nonnegative integer. If the baseband header CRC fails, the function considers the frame as lost.

Data Types: `double`

`pktCRCStatus` — UP CRC status
cell array of column vectors

UP CRC status, returned as a cell array of column vectors. Each element of the cell array is of data type `logical`. For a multi-input stream transmission, each element of the cell array corresponds to an individual input stream.

`pktCRCStatus` applies for only the input streams where the value of the UPL property of `dvbs2xWaveformGenerator` System object is nonzero. When UPL is set to zero, `pktCRCStatus` returns an empty output.

Data Types: `cell`

decodedParams — Parameters of the processed PL frames

structure

Parameters of the processed PL frames, returned as a structure with these fields.

Structure Field	Value	Description
PLSDecimalCode	0 or integer in the range [4, 249]	PL signaling code information derived from the PL header recovery.
IsDummyFrame	0 (false) or 1 (true)	Flag to indicate whether the decoded PL frame is a dummy frame. A value of 1 (true) indicates the decoded PL frame is a dummy frame.
BBHeaderCRCStatus	0 (false) or 1 (true)	Flag to indicate whether the baseband (BB) header CRC check passed. A value of 1 (true) indicates the header CRC is valid. This field is returned as 0 (false) for dummy frames.
ModulationOrder	Integer from the set {0, 2, 4, 8, 16, 32, 64, 128, 256}	Modulation order of the FEC frame symbols. This field is returned as 0 for dummy frames.
FECFrameLength	16200, 32400, or 64800	FEC frame length is the output length (in bits) specified to the FEC encoder. This field is returned as 0 for dummy frames.

Structure Field	Value	Description
`CanonicalMODCODName`	cell array	Canonical modulation scheme and code rate name for VL-SNR frame transmissions. This field is present only if you pass a VL-SNR frame to this ideal receiver. Valid `CanonicalMODCODName` values include these options. • `"QPSK 2/9"`, `"BPSK 1/5"`, `"BPSK 11/45"`, `"BPSK-S 1/5"`, `"BPSK-S 11/45"`, and `"BPSK 1/3"` — Applicable for VL-SNR set 1 • `"BPSK 1/5"`, `"BPSK 4/15"`, `"BPSK 1/3"`, and `"dummy"` — Applicable for VL-SNR set 2 If the frame passed to the receiver is not a VL-SNR frame, the output of this field is an empty string.
`HasPilots`	`0` (`false`) or `1` (`true`)	Flag to indicate the pilot block. A value of `1` (`true`) indicates the presence of pilots. This field is returned as `0` (`false`) for dummy frames.

In the case of a single-input stream, all structure fields except `CanonicalMODCODName` are returned as integers and as vectors in the case of a multi-input stream. For a multi-input stream, each element of a vector corresponds to an individual input stream.

Whether for a single-input stream or a multi-input stream, the function returns `CanonicalMODCODName` as a cell array, where each element of the cell array is of data type `char` or `string`.

Data Types: `struct`

Version History

Introduced in R2022b

References

[1] ETSI Standard EN 302 307-2 V1.1.1(2015-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and Other Broadband Satellite Applications; Part 2: DVB-S2 Extensions (DVB-S2X)*

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`dvbs2xWaveformGenerator` | `dvbs2WaveformGenerator`

Functions
`dvbs2BitRecover`

Topics
“End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames”
“End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames”
“End-to-End DVB-S2X Simulation with RF Impairments and Corrections in Wideband Mode”

gnssBitSynchronize

Bit synchronizer for GNSS receivers

Syntax

`[syncidx,numtr] = gnssBitSynchronize(samples,n)`

Description

`[syncidx,numtr] = gnssBitSynchronize(samples,n)` performs bit synchronization on the input samples `samples` with a window size of `n`, as defined in [2]. The function searches the input samples for the maximum number of transitions from a positive to a negative value and from a negative to a positive value, and returns the bit synchronization index `syncidx` and the number of transitions corresponding to each sample location `numtr`.

Use this function for global navigation satellite systems (GNSS) receivers that use code division multiple access (CDMA) schemes. These GNSS receivers include GPS, NavIC, and QZSS.

Examples

Bit Synchronize Signal Generated Using C/A-Code

Perform bit synchronization for a noisy signal generated from random bits using C/A-code. Assume perfect time and frequency synchronization.

Generate the transmit signal from random bits using C/A-code.

```
n = 20;                                                     % Number of C/A-code blocks per bit
numbits = 500;                                              % Number of data bits
numcachips = 1023;                                          % Number of C/A-code chips
txbits = randi([0,1],numbits,1);
cacode = 1 - 2*double(repmat(gnssCACode(1,"GPS"),n*numbits,1));
txsig = cacode.*repelem(1 - 2*txbits,numcachips*n,1);
```

Add additive white Gaussian noise (AWGN) to the generated signal.

```
snrdB = -25;                       % Signal to noise ratio in dB
rxsig = awgn(txsig,snrdB,"measured");
```

Multiply the noisy signal with time synchronized reference C/A-code and integrate the samples.

```
corrsamples = rxsig.*cacode;
integsamples = sum(reshape(corrsamples,numcachips,[]));
```

Delay the samples by a fixed offset.

```
dly = 15;                                 % Number of samples delayed
bitsyncin = [zeros(dly,1);integsamples(:)];
```

Perform bit synchronization on the input samples.

```
[syncidx,numtr] = gnssBitSynchronize(bitsyncin,n);
```

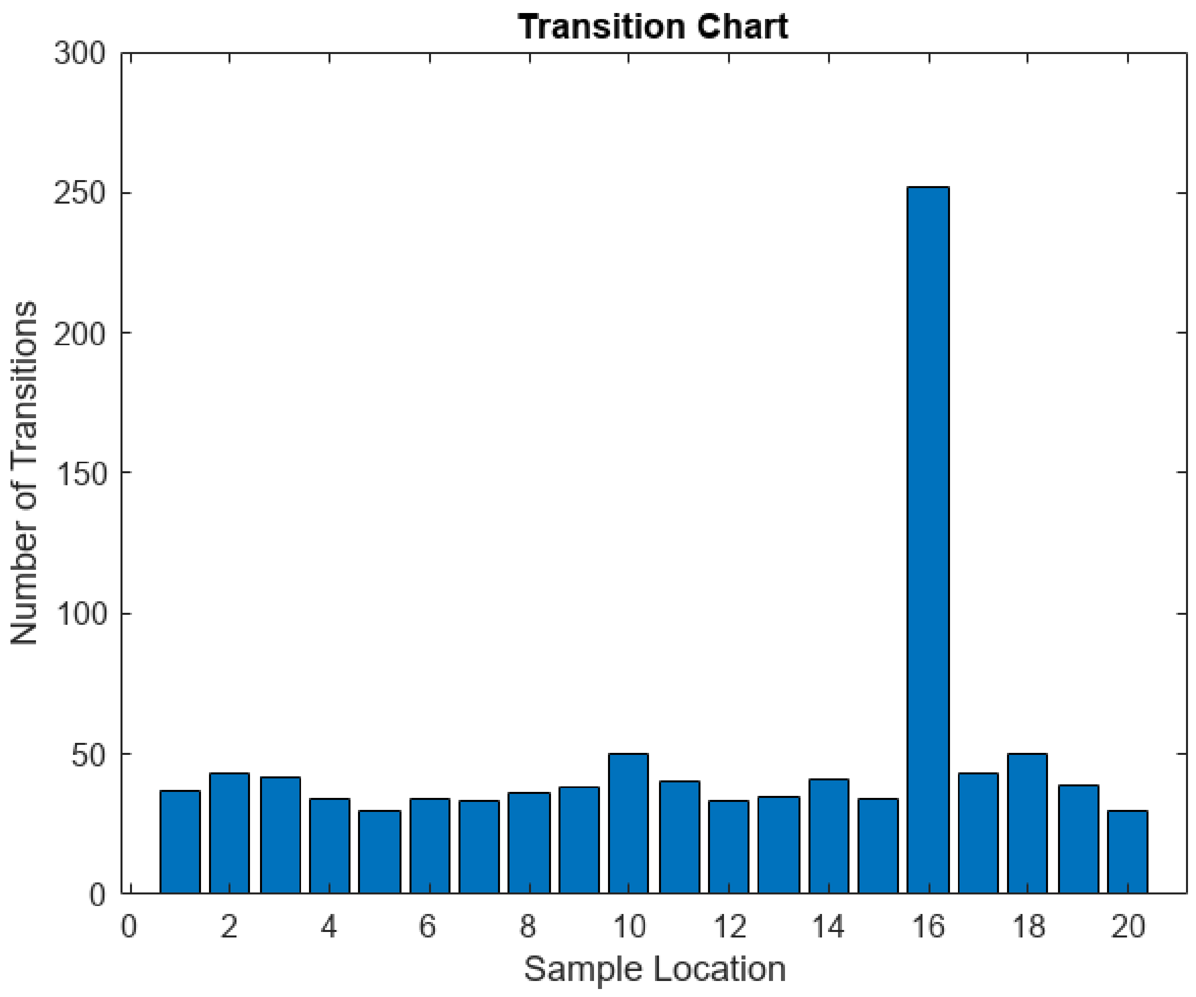
Display the value of bit synchronization index, `syncidx`. Note that it is equal to the number to samples delayed + 1.

```
syncidx
```

```
syncidx = 16
```

Plot the transition chart at each sample location.

```
bar(numtr)
xlabel("Sample Location")
ylabel("Number of Transitions")
title("Transition Chart")
```



Input Arguments

`samples` — Input samples
real-valued column vector

Input samples, specified as a real-valued column vector of length greater than or equal to `n`.

For a better estimate of the bit synchronization index, specify a large number of input samples (such as hundreds of times `n`).

Data Types: `double`

n — Window size used for searching
positive integer

Window size used for searching the input samples, specified as a positive integer. This value represents the number of spreading code blocks per bit, or the number of samples per bit.

Data Types: `double` | `uint8`

Output Arguments

`syncidx` — Bit synchronization index
integer in the range [1, `n`]

Bit synchronization index, returned as an integer in the range [1, `n`]. This index value represents the maximum number of transitions within the window size `n`.

The data type of the index is same as that of `n`.

`numtr` — Number of transitions corresponding to each sample location
column vector

Number of transitions corresponding to each sample location, returned as a column vector of length `n`.

Data Types: `double`

Version History

Introduced in R2022b

References

[1] GPS Enterprise Space & Missile Systems Center (SMC) - LAAFB. *NAVSTAR GPS Space Segment/ Navigation User Segment Interfaces*. IS-GPS-200L. El Segundo, CA: SAIC (GPS SE&I), May 14, 2020. https://navcen.uscg.gov/sites/default/files/pdf/gps/IS_GPS_200L.pdf.

[2] Kaplan, Elliott D., and C. Hegarty, eds. *Understanding GPS/GNSS: Principles and Applications.* Third edition. GNSS Technology and Applications Series. Boston ; London: Artech House, 2017.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`gnssCACode`

Objects
`gpsPCode`

Topics
"GPS Data Decode"

gnssCACode

Generate C/A-code for GPS, NavIC, and QZSS satellites

Syntax

```
code = gnssCACode(prnid,gnsstype)
```

Description

`code = gnssCACode(prnid,gnsstype)` generates coarse acquisition codes (C/A-codes) for the specified pseudo-random noise (PRN) index, `prnid`, of the satellite constellation specified by `gnsstype`.

Examples

Generate C/A-code for Multiple GPS Satellites

Specify the unique pseudo-random noise (PRN) index for three GPS satellites.

```
prnid = [43 87 10]; % 3 satellites
gnsstype = "GPS";   % Global navigation satellite constellation type
```

Generate the C/A-code for these three GPS satellites.

```
code = gnssCACode(prnid,gnsstype);
size(code)
```

```
ans = 1×2

        1023           3
```

Generate C/A-code for NavIC Satellites over Multiple Epochs

Specify the unique PRN index for two NavIC S-band satellites.

```
prnid = [2 13];
gnsstype = "NavIC S-SPS"; % S-band
```

Generate the C/A-code for these two NavIC S-band satellites.

```
code = gnssCACode(prnid,gnsstype);
```

Calculate the output for 10 C/A-code epochs.

```
numCAEpochs = 10;
fullCode = repmat(code,numCAEpochs,1);
size(fullCode)
```

```
ans = 1×2

       10230           2
```

Input Arguments

`prnid` — Satellite PRN index
integer | vector of integers

Satellite PRN index for which the function generates a C/A-code, specified as a scalar indicating a PRN index for a single satellite or a vector indicating PRN indices for multiple satellites. Valid values of PRN indices depend on the `gnsstype` input.

`gnsstype` Value	PRN Index Valid Value
`"GPS"`	integer in the range [1, 210]
`"QZSS"`	integer in the range [183, 202]
`"NavIC L5-SPS"` or `"NavIC S-SPS"`	integer in the range [1, 14]

Data Types: `double` | `uint8`

`gnsstype` — Type of global navigation satellite constellation
`"GPS"` | `"QZSS"` | `"NavIC L5-SPS"` | `"NavIC S-SPS"`

Type of global navigation satellite constellation, specified as one of these values.

- `"GPS"`
- `"QZSS"`
- `"NavIC L5-SPS"`
- `"NavIC S-SPS"`

Data Types: `char` | `string`

Output Arguments

code — Generated C/A-code
column vector | matrix

Generated C/A-code, returned as one of these options.

- Column vector of length 1023 — When you specify `prnid` as a scalar.
- Matrix — When you specify `prnid` as a vector. The number of rows of this matrix is equal to 1023, and the number of columns correspond to the length of the `prnid` vector. Each column of this matrix represents the generated C/A-code corresponding to the element in the `prnid` vector.

For detailed information on the relationship between PRN index values and the generated C/A-codes, refer to IS-GPS-200L Table 3-Ia, 3-Ib, and 6-I [1], ISRO-IRNSS-ICD-SPS-1.1 Table 7 [2], and IS-QZSS-PNT-004 Table 3.2.2-2 [3].

Version History

Introduced in R2021b

References

[1] IS-GPS-200L. "*NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*". GPS Enterprise Space & Missile Systems Center (SMC) - LAAFB, May 14, 2020.

[2] ISRO-IRNSS-ICD-SPS-1.1. "*Signal in space ICD for standard positioning service*". ISRO satellite navigation programme. August 2017.

[3] IS-QZSS-PNT-004. "*Quasi-Zenith Satellite System. Interface Specification. Satellite Positioning, Navigation and Timing Service*". Cabinet office, Government of Japan. January 25, 2021.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`gpsPCode` | `comm.GoldSequence` | `comm.PNSequence`

Topics
“GPS Waveform Generation”

p618PropagationLosses

Calculate Earth-space propagation losses, cross-polarization discrimination, and sky noise temperature

Syntax

```
[pl,xpd,tsky] = p618PropagationLosses(p618cfg)
[pl,xpd,tsky] = p618PropagationLosses(p618cfg,Name,Value)
```

Description

`[pl,xpd,tsky] = p618PropagationLosses(p618cfg)` returns Earth-space propagation losses `pl`, cross-polarization discrimination `xpd`, and sky noise temperature `tsky`, as defined in the ITU-R P.618 recommendation [1]. `p618cfg` specifies the P.618 configuration parameters.

This function requires MAT-files with digital maps from International Telecommunication Union (ITU) documents. If they are not available on the path, download and uncompress the data files from https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz to a location on the MATLAB path.

`[pl,xpd,tsky] = p618PropagationLosses(p618cfg,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Calculate Propagation Losses, Cross-Polarization Discrimination, and Sky Noise Temperature

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a default P.618 configuration object.

```
cfg = p618Config;
```

Specify the time percentage of excess for the rain attenuation per annum as 0.01 and the time percentage of excess for the total attenuation per annum as 0.001.

```
cfg.RainAnnualExceedance = 0.01;
cfg.TotalAnnualExceedance = 0.001;
```

Calculate the propagation losses, cross-polarization discrimination, and sky noise temperature.

```
[pl,xpd,tsky] = p618PropagationLosses(cfg)
```

```
pl = struct with fields:
    Ag: 0.2269
    Ac: 0.4552
    Ar: 6.7981
    As: 0.2633
    At: 15.6091


xpd = 32.8876

tsky = 267.4689
```

Calculate Earth-space Propagation Losses Using Name-Value Pair Arguments

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and untar the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a P.618 configuration object with a signal frequency of 20 GHz.

```
cfg = p618Config('Frequency',20e9);
```

Specify the surface water vapor density as $2.8\frac{g}{m^3}$, the total columnar content of the cloud liquid water as $1.4\ \frac{\text{kg}}{m^2}$, and the median value of the wet surface refractivity as 1.2. Set the earth station height as 0.5 km. Calculate the Earth-space propagation losses.

```
pl =  p618PropagationLosses(cfg,'StationHeight',0.5,...
                            'WaterVaporDensity',2.8,...
```

```
                                'TotalColumnarContent',1.4,...
                                'WetSurfaceRefractivity',1.2)
```

```
pl = struct with fields:
    Ag: 0.8649
    Ac: 1.0987
    Ar: 0.8907
    As: 0.1372
    At: 2.8590
```

Calculate Propagation Losses in Light Rainfall

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a P.618 configuration object that occupies a signal frequency of 20 GHz.

```
cfg = p618Config('Frequency',20e9);
```

Calculate the propagation losses in a light rainfall of 1 mm/hr with an earth station height of 0.75 km.

```
pl =  p618PropagationLosses(cfg,'RainRate',1,'StationHeight',0.75)
```

```
pl = struct with fields:
    Ag: 0.7996
    Ac: 0.8793
    Ar: 0.0177
    As: 0.3187
    At: 1.7514
```

Input Arguments

p618cfg — P.618 configuration
p618Config object

P.618 configuration required for the calculation of the propagation losses, cross-polarization discrimination, and sky noise temperature, specified as a `p618Config` object.

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: `'StationHeight'`,`1.5` specifies the earth station height as 1.5 km.

`StationHeight` — Height of earth station

nonnegative scalar

Height of the earth station above the mean sea level in km, specified as the comma-separated pair consisting of '`StationHeight`' and a nonnegative scalar. The maximum supported value is 100. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.1511 section 1, Annex 1 [3] to obtain the station height value.

Data Types: `double` | `single`

`Temperature` — Temperature of earth surface

nonnegative scalar

Temperature of the earth surface in kelvin, specified as the comma-separated pair consisting of '`Temperature`' and a nonnegative scalar. If the local data is not available as an input, the function uses the map of the mean annual surface temperature provided in ITU-R P.1510 section 1, Annex 1 [4] to obtain the temperature value.

Data Types: `double` | `single`

`Pressure` — Dry air pressure at earth surface

nonnegative scalar

Dry air pressure at the earth surface in hPa, specified as the comma-separated pair consisting of '`Pressure`' and a nonnegative scalar. If the local data is not available as an input, the function uses the mean annual global reference atmosphere provided in ITU-R P.835 section 1.1, Annex 1 [5] to obtain the air pressure value.

Data Types: `double` | `single`

`WaterVaporDensity` — Surface water vapor density

nonnegative scalar

Surface water vapor density in g/m^3, specified as the comma-separated pair consisting of '`WaterVaporDensity`' and a nonnegative scalar. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.836 section 1, Annex 1 [6] to estimate the value of the water vapor density.

Data Types: `double` | `single`

`IntegratedWaterVaporContent` — Integrated water vapor content

positive scalar

Integrated water vapor content exceeded for the percentage of GasAnnualExceedance in an average year, specified as the comma-separated pair consisting of '`IntegratedWaterVaporContent`' and a positive scalar. Units are in kg/m^2 or mm. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.836 section 1, Annex 2 [6] to obtain the value of the integrated water vapor content.

Data Types: `double` | `single`

`TotalColumnarContent` — Total columnar content of cloud liquid water
nonnegative scalar

Total columnar content of the cloud liquid water exceeded for the percentage of CloudAnnualExceedance in an average year, specified as the comma-separated pair consisting of '`TotalColumnarContent`' and a nonnegative scalar. Units are in kg/m^2 or mm. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.840 section 3.1, Annex 1 [7] to obtain the value of the total columnar content.

Data Types: `double` | `single`

`RainRate` — Point rainfall rate
nonnegative scalar

Point rainfall rate at the location for 0.01% of an average year, specified as the comma-separated pair consisting of '`RainRate`' and a nonnegative scalar. Units are in mm/hr. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.837, Annex 1 [8] to obtain the value of the point rainfall rate.

Data Types: `double` | `single`

`WetSurfaceRefractivity` — Median value of wet term of surface refractivity
nonnegative scalar

Median value of the wet term of the surface refractivity, specified as the comma-separated pair consisting of '`WetSurfaceRefractivity`' and a nonnegative scalar. If the local data is not available as an input, the function uses the digital maps provided in ITU-R P.453 section 2.2, Annex 1 [9] to obtain the value of the wet surface refractivity.

Data Types: `double` | `single`

`MeanRadiatingTemperature` — Atmospheric mean radiating temperature
nonnegative scalar

Atmospheric mean radiating temperature in kelvin, specified as the comma-separated pair consisting of '`MeanRadiatingTemperature`' and a nonnegative scalar. If the local data is not available as an input, the function uses an atmospheric mean radiating temperature of 275 K in the computation.

Data Types: `double` | `single`

Output Arguments

`pl` — Earth-space propagation losses information
structure

Earth-space propagation losses information, returned as a structure containing these fields.

Fields	Description
At	Total atmospheric attenuation (in dB)

Fields	Description
Ag	Gaseous attenuation (in dB)
Ac	Cloud and fog attenuation (in dB)
Ar	Rain attenuation (in dB)
As	Attenuation due to tropospheric scintillation (in dB)

xpd — Cross-polarization discrimination
scalar

Cross-polarization discrimination in (dB) not exceeded for the percentage of the RainAnnualExceedance, returned as a scalar.

`tsky` — Sky noise temperature
nonnegative scalar

Sky noise temperature (in kelvin) at the ground station antenna, returned as a nonnegative scalar.

Version History

Introduced in R2021a

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

[2] International Telecommunication Union, ITU-R Recommendation P.676 (08/2019).

[3] International Telecommunication Union, ITU-R Recommendation P.1511 (08/2019).

[4] International Telecommunication Union, ITU-R Recommendation P.1510 (06/2017).

[5] International Telecommunication Union, ITU-R Recommendation P.835 (12/2017).

[6] International Telecommunication Union, ITU-R Recommendation P.836 (12/2017).

[7] International Telecommunication Union, ITU-R Recommendation P.840 (08/2019).

[8] International Telecommunication Union, ITU-R Recommendation P.837 (06/2017).

[9] International Telecommunication Union, ITU-R Recommendation P.453 (08/2019).

[10] International Telecommunication Union, ITU-R Recommendation P.839 (09/2013).

[11] International Telecommunication Union, ITU-R Recommendation P.838 (03/2005).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports only MEX code generation.

See Also

Objects
`p618Config` | `p618SiteDiversityConfig`

Functions
`p618SiteDiversityOutage`

p618SiteDiversityOutage

Calculate outage probability due to rain attenuation with site diversity

Syntax

```
Outage = p618SiteDiversityOutage(cfgsd)
Outage = p618SiteDiversityOutage(cfgsd,Name,Value)
```

Description

`Outage = p618SiteDiversityOutage(cfgsd)` returns the outage probability due to rain attenuation with site diversity. The function calculates this value as per the ITU-R P.618 recommendation [1].

This function requires MAT-files with digital maps from International Telecommunication Union (ITU) documents. If they are not available on the path, download and uncompress the data files from https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz to a location on the MATLAB path.

`Outage = p618SiteDiversityOutage(cfgsd,Name,Value)` specifies additional options using one or more name-value pair arguments.

Examples

Calculate Outage Probability due to Rain Attenuation with Site Diversity

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and untar the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a P.618 site diversity configuration object with a signal frequency of 25 GHz.

```
cfgsd = p618SiteDiversityConfig;
cfgsd.Frequency = 25e9;
```

Specify the polarization tilt angles for two sites as [-90 90] degrees, separation between the two sites as 50 km, and attenuation threshold on the two links as [9 9] dB.

```
cfgsd.PolarizationTiltAngle = [-90 90];
cfgsd.SiteDistance = 50;
cfgsd.AttenuationThreshold = [9 9];
```

Calculate the outage probability due to rain attenuation with site diversity.

```
outage = p618SiteDiversityOutage(cfgsd)
```

```
outage = 0.0338
```

Calculate Outage Probability with Site Diversity Using Name-Value Pair Arguments

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute these commands to download and untar the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a default P.618 site diversity configuration object. Change the signal frequency to 25 GHz.

```
cfgsd = p618SiteDiversityConfig;
cfgsd.Frequency = 25e9;
```

Specify the separation between two sites as 50 km and the attenuation threshold on the two links as [9 9] dB.

```
cfgsd.SiteDistance = 50;
cfgsd.AttenuationThreshold = [9 9];
```

Calculate the outage probability for the specified site diversity configuration.

```
outage = p618SiteDiversityOutage(cfgsd,'RainAnnualExceedances',[0.01 0.05 0.2],...
                                'RainProbability1',0.3,...
                                'RainProbability2',0.5)
```

```
outage = 0.0339
```

Input Arguments

`cfgsd` — P.618 site diversity configuration
p618SiteDiversityConfig object

P.618 site diversity configuration required for the calculation of the outage probability due to rain attenuation, specified as a `p618SiteDiversityConfig` object.

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: `'RainAnnualExceedances',[0.01 0.02 0.03 0.05]` specifies the average annual time percentage of excess for the rain attenuation.

`RainAnnualExceedances` — Average annual time percentage of excess for rain attenuation
nonnegative vector

Average annual time percentage of excess for the rain attenuation, specified as the comma-separated pair consisting of `'RainAnnualExceedances'` and a nonnegative vector. The values in this vector must be less than the probability of rain at the two sites.

If the local data is not available as an input, the function uses `[0.01 0.02 0.03 0.05 0.1 0.2 0.3 0.5 1 2 3 5]` as the default vector.

Data Types: `double` | `single`

`RainAttenuations1` — Rain attenuations at site 1
nonnegative vector

Rain attenuations (in dB) at site 1, specified as the comma-separated pair consisting of `'RainAttenuations1'` and a nonnegative vector. This value specifies the rain attenuation exceeded for the percentages given in the `RainAnnualExceedances` name-value pair argument. The dimension of this value must match that of the `RainAnnualExceedances`.

If the local data is not available as an input, the function uses the method as defined in section 2.2.1.1 of the ITU-R P.618 [1] recommendation to calculate the rain attenuations for site 1.

Note If you do not specify `RainAttenuations1`, then `RainAnnualExceedances` must be in the range from 0.01% to 5%.

Data Types: `double` | `single`

`RainAttenuations2` — Rain attenuations at site 2
nonnegative vector

Rain attenuations (in dB) at site 2, specified as the comma-separated pair consisting of `'RainAttenuations2'` and a nonnegative vector. This value specifies the rain attenuation exceeded for the percentages given in the `RainAnnualExceedances` name-value pair argument. The dimension of this value must match that of the `RainAnnualExceedances`.

If the local data is not available as an input, the function uses the method as defined in section 2.2.1.1 of the ITU-R P.618 recommendation to calculate the rain attenuations for site 2.

Note If you do not specify `RainAttenuations2`, then `RainAnnualExceedances` must be in the range from 0.01% to 5%.

Data Types: `double` | `single`

`RainProbability1` — Probability of rain for site 1
nonnegative scalar

Probability of (in %) rain for site 1, specified as the comma-separated pair consisting of `'RainProbability1'` and a nonnegative scalar.

If the local measured rainfall rate data is not available as an input, the function uses the digital maps as defined in ITU-R P.837 Annex 1 [2] to calculate the rain probability for the sites.

Data Types: `double` | `single`

`RainProbability2` — Probability of rain for site 2
nonnegative scalar

Probability of (in %) rain for site 2, specified as the comma-separated pair consisting of `'RainProbability2'` and a nonnegative scalar.

If the local measured rainfall rate data is not available as an input, the function uses the digital maps as defined in ITU-R P.837 Annex 1 [2] to calculate the rain probability for the sites.

Data Types: `double` | `single`

Output Arguments

`Outage` — Outage probability due to rain attenuation with site diversity
nonnegative scalar

Outage probability due to rain attenuation with site diversity, returned as a nonnegative scalar. This argument predicts the joint probability ($P_r(A_1 \geq a_1, A_2 \geq a_2)$), where the attenuation on the path of the site 1 must exceed a_1 and the attenuation on the path of the site 2 must exceed a_2.

Version History

Introduced in R2021a

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

[2] International Telecommunication Union, ITU-R Recommendation P.837 (06/2017).

[3] International Telecommunication Union, ITU-R Recommendation P.1511 (08/2019).

[4] International Telecommunication Union, ITU-R Recommendation P.1510 (06/2017).

[5] International Telecommunication Union, ITU-R Recommendation P.836 (12/2017).

[6] International Telecommunication Union, ITU-R Recommendation P.840 (08/2019).

[7] International Telecommunication Union, ITU-R Recommendation P.453 (08/2019).

[8] International Telecommunication Union, ITU-R Recommendation P.839 (09/2013).

[9] International Telecommunication Union, ITU-R Recommendation P.838 (03/2005).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Supports only MEX code generation.

See Also

Objects
`p618Config` | `p618SiteDiversityConfig`

Functions
`p618PropagationLosses`

ccsdsTCWaveform

Generate CCSDS TC waveform

Syntax

```
waveform = ccsdsTCWaveform(bits,cfgFormat)
[waveform,encodedBits] = ccsdsTCWaveform(bits,cfgFormat)
```

Description

`waveform = ccsdsTCWaveform(bits,cfgFormat)` generates a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform, `waveform`, for the corresponding input bits, `bits`, and the given format configuration, `cfgFormat`.

`[waveform,encodedBits] = ccsdsTCWaveform(bits,cfgFormat)` also returns the bits obtained after TC synchronization and channel coding sublayer operations.

Examples

Create CCSDS TC Waveform for Multiple CLTUs

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform for multiple communications link transmission units (CLTUs).

Create a default CCSDS TC configuration object.

```
cfg = ccsdsTCConfig;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

             DataFormat: "CLTU"
          ChannelCoding: "BCH"
          HasRandomizer: 1
             Modulation: "PCM/PSK/PM"
              PCMFormat: "NRZ-L"
        ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
             SymbolRate: 4000
       SamplesPerSymbol: 10

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Specify the number of CLTUs and the transfer frame length.

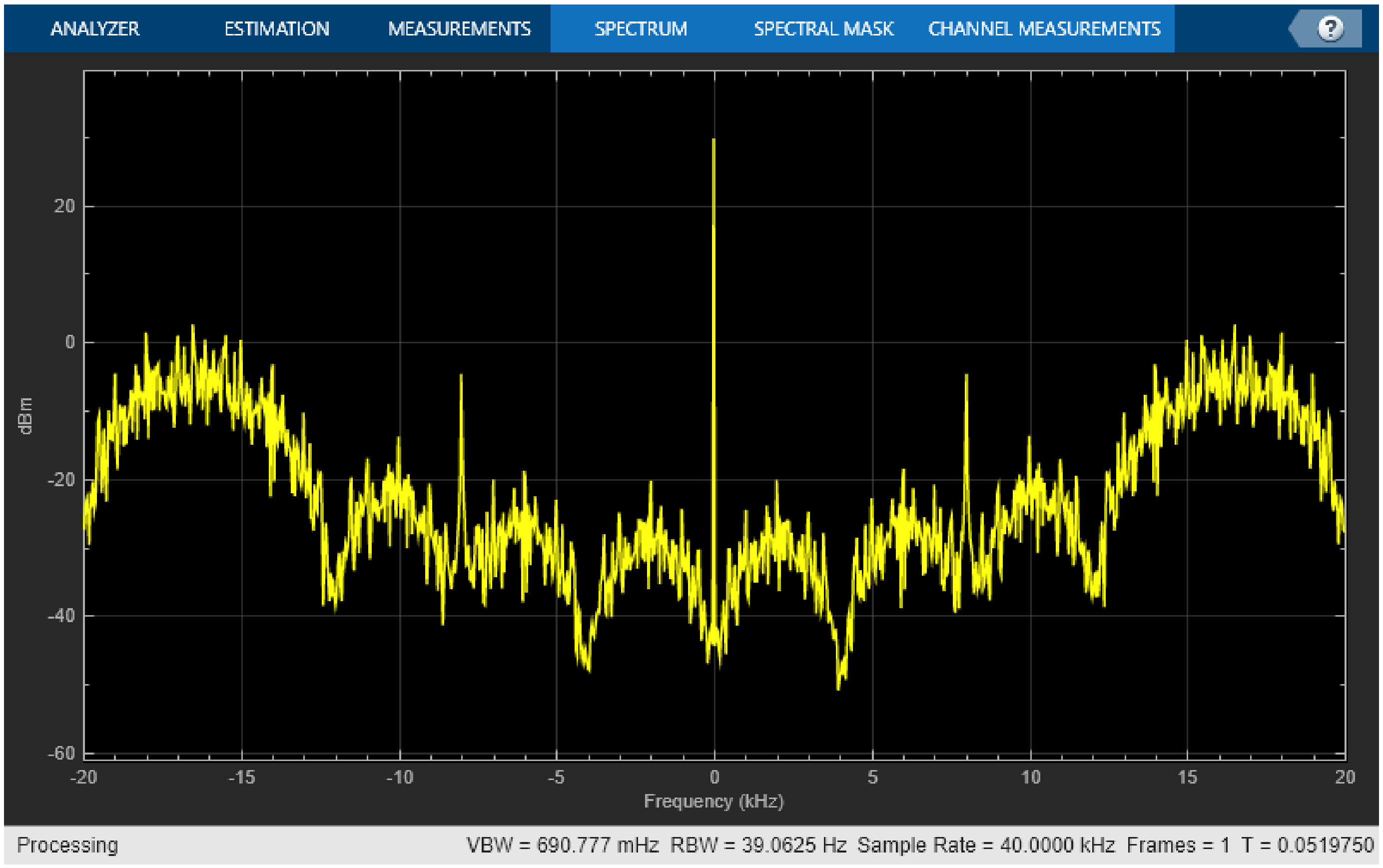
```
numCLTUs = 10;
transferFramesLength = 8; % Number of octets in each transfer frame
```

Generate the CCSDS TC time-domain waveform for the transfer frames.

```
c = cell(1,numCLTUs); % Cell array to store the generated waveform for all CLTUs
for k=1:numCLTUs
    bits = randi([0 1],8*transferFramesLength,1); % Bits in the TC transfer frame
    waveform = ccsdsTCWaveform(bits,cfg);
    c{1,k} = waveform; % Waveform for each CLTU
end
```

Create a spectrum analyzer System object to display the frequency spectrum of the generated CCSDS TC time-domain waveform from the last CLTU.

```
scope = spectrumAnalyzer;
scope.SampleRate = cfg.SamplesPerSymbol*cfg.SymbolRate;
scope(waveform)  % Last CLTU spectrum display
```



```
Processing          VBW = 690.777 mHz  RBW = 39.0625 Hz  Sample Rate = 40.0000 kHz  Frames = 1  T = 0.0519750
```

Create CCSDS TC Waveform for Acquisition Sequence

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform for a acquisition sequence with 20 octets.

Create a CCSDS TC configuration object, and then specify the object properties. Display the object properties.

```
cfg = ccsdsTCConfig;
cfg.DataFormat = "acquisition sequence";
cfg.Modulation = "PCM/PM/biphase-L";
cfg.ModulationIndex = 1.2;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

          DataFormat: "acquisition sequence"
          Modulation: "PCM/PM/biphase-L"
     ModulationIndex: 1.2000
    SamplesPerSymbol: 10
```

Generate the CCSDS TC waveform.

```
bits = repmat([0;1],8*10,1); % Alternating 1s and 0s with 0s as a starting sequence bit
waveform = ccsdsTCWaveform(bits,cfg);
```

Input Arguments

`bits` — Information bits

binary-valued column vector

Information bits, specified as a binary-valued column vector.

- When you set the DataFormat property of the `ccsdsTCConfig` object to `"CLTU"`, the length of this vector must be an integer multiple of 8.
- When you set the `DataFormat` property of the `ccsdsTCConfig` object to `"acquisition sequence"` or `"idle sequence"`, this vector must be a sequence of alternating 1s and 0s, starting with either 1 or 0.

Data Types: `double` | `int8` | `logical`

`cfgFormat` — Format configuration object

`ccsdsTCConfig` object

Format configuration object, specified as `ccsdsTCConfig` object. The properties of this object define the parameters required for CCSDS TC waveform generation.

Output Arguments

`waveform` — Generated time-domain CCSDS TC waveform

column vector

Generated time-domain CCSDS TC waveform, returned as a column vector. The `waveform` output is generated in the form of complex in-phase quadrature (IQ) samples.

Data Types: `double`

`encodedBits` — Output bits obtained after TC synchronization and channel coding sublayer operations

column vector

Output bits obtained after TC synchronization and channel coding sublayer operations, returned as a column vector.

Data Types: `double`

Version History

Introduced in R2021a

References

[1] CCSDS 231.0-B-3. Blue Book. Issue 3. "TC Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-29. Blue Book. Issue 29. "Radio Frequency and Modulation Systems - Part 1". *Earth Stations and Spacecraft*. Washington, D.C.: CCSDS, September 2019.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`ccsdsTCIdealReceiver`

Objects
`ccsdsTCConfig` | `ccsdsTMWaveformGenerator`

dvbrcs2BitRecover

Recover bits for DVB-RCS2 waveform

Syntax

`[bits,framePDUErr] = dvbrcs2BitRecover(rxdata,cfgrx,nvar)`

Description

`[bits,framePDUErr] = dvbrcs2BitRecover(rxdata,cfgrx,nvar)` recovers frame protocol data unit (PDU), `bits`, and the frame PDU cyclic redundancy check (CRC) status, `framePDUErr`. Input `rxdata` is the received complex in-phase quadrature (IQ) symbols in the form of bursts of a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) transmission. `cfgrx` is the recovery configuration object, `dvbrcs2RecoveryConfig`. `nvar` is the noise variance estimate that the function uses to calculate soft bits.

The function supports demodulation and decoding of the turbo codes with linear modulation (TC-LM), and spread spectrum and turbo codes with linear modulation (SS-TC-LM) transmission formats, with all three PDU types (logon, control, and traffic), for reference and custom waveforms.

Examples

Recover PDU from DVB-RCS2 Reference Waveform

Recover the frame PDU for a DVB-RCS2 reference waveform.

Set the properties of a DVB-RCS2 waveform generator System object™.

```
wg = dvbrcs2WaveformGenerator;
wg.TransmissionFormat = "SS-TC-LM";
wg.WaveformID = 7;
wg.SamplesPerSymbol = 2;
```

Generate a frame PDU.

```
framePDU = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2-based burst symbols.

```
txWaveform = wg(framePDU);
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = wg.SamplesPerSymbol;
EsNodB = 1;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,"measured");
```

Create and then configure the DVB-RCS2 recovery configuration object.

```
cfg = dvbrcs2RecoveryConfig;
cfg.TransmissionFormat = wg.TransmissionFormat;
cfg.WaveformID = wg.WaveformID;
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
                 'RolloffFactor',0.2, ...
                 'InputSamplesPerSymbol',sps, ...
                 'DecimationFactor',sps);
span = rxFilter.FilterSpanInSymbols;
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter([rxIn; ...
               complex(zeros(span/2*sps,1))]);
rxSymb = filtOut(span+1:end);
```

Recover user packets. Display the frame PDU cyclic redundancy check (CRC) status and the numbers of bit errors.

```
[rxOut,pduErr] = dvbrcs2BitRecover(rxSymb,cfg,10^(-EsNodB/10));
fprintf("Erroneous frame PDU = %d\n", pduErr)
```

```
Erroneous frame PDU = 0
```

```
fprintf("Number of bit errors = %d\n", sum(framePDU~=rxOut))
```

```
Number of bit errors = 0
```

Recover PDU from DVB-RCS2 Custom Waveform

Recover the frame PDU for a DVB-RCS2 custom waveform.

Set the properties of the DVB-RCS2 waveform generator System object™.

```
wg = dvbrcs2WaveformGenerator;
wg.IsCustomWaveform = true;
wg.PayloadLengthInBytes = 115;
wg.MappingScheme = "8PSK";
wg.CodeRate = "2/3";
wg.PermutationParameters = [29 6 5 0 0];
wg.UniqueWord = "3ACF08B13076";
```

Get the characteristic information about the DVB-RCS2 waveform generator.

```
info(wg)
```

```
ans = struct with fields:
              BurstLength: 476
     PayloadLengthInBytes: 115
            MappingScheme: "8PSK"
                 CodeRate: "2/3"
           PreambleLength: 8
          PostambleLength: 8
              PilotPeriod: 0
          PilotBlockLength: 1
    PermutationParameters: [29 6 5 0 0]
               UniqueWord: "3ACF08B13076"
                 PilotSum: 0
```

Generate a frame PDU.

```
framePDU = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2-based burst symbols.

```
txWaveform = wg(framePDU);
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = wg.SamplesPerSymbol;
EsNodB = 9;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Configure the DVB-RCS2 recovery configuration object.

```
cfg = dvbrcs2RecoveryConfig;
cfg.IsCustomWaveform = true;
cfg.MappingScheme = wg.MappingScheme;
cfg.CodeRate = wg.CodeRate;
cfg.PermutationParameters = wg.PermutationParameters;
```

Get burst parameters from waveform generator info method.

```
burstParams = info(wg);
cfg.BurstLength = burstParams.BurstLength;
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor',0.2, ...
    'InputSamplesPerSymbol',sps,...
    'DecimationFactor',sps);
span = rxFilter.FilterSpanInSymbols;
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter([rxIn; ...
    complex(zeros(span/2*sps,1))]);
rxSymb = filtOut(span+1:end);
```

Recover user packets. Display the frame PDU cyclic redundancy check (CRC) status and the numbers of bit errors.

```
[rxOut,pduErr] = dvbrcs2BitRecover(rxSymb,cfg,10^(-EsNodB/10));
fprintf('Erroneous frame PDU = %d\n', pduErr)
```

```
Erroneous frame PDU = 0
```

```
fprintf('Number of bit errors = %d\n', sum(framePDU~=rxOut))
```

```
Number of bit errors = 0
```

Recover PDU from Burst Configuration Parameters

Recover the frame PDU for a DVB-RCS2 waveform with specified burst configuration parameters.

Set the burst configuration paramters.

```
Rsym = 1e6;                    % Symbol rate (1 Msps)
tSlot = 2.11e-3;               % Burst time slot duration (2.11 ms)
preBurstGuardOffset = 20e-6;  % 20 microsecond
waveId = 39;                   % Waveform ID
```

Set the properties of the DVB-RCS2 waveform generator System object™.

```
wg = dvbrcs2WaveformGenerator;
wg.WaveformID = waveId;         % QPSK 6/7
```

Compute the burst parameters in terms of symbols.

```
wg.PreBurstGuardLength = ceil(preBurstGuardOffset*Rsym);
params = info(wg);
burstPayLoadDuration = params.BurstLength/Rsym;
burstPostGuard = ceil((tSlot-preBurstGuardOffset-burstPayLoadDuration)*Rsym);
wg.PostBurstGuardLength = burstPostGuard;
```

Generate the frame PDU.

```
framePDU = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2-based burst symbols

```
txWaveform = wg(framePDU);
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = wg.SamplesPerSymbol;
EsNodB = 7;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Configure the DVB-RCS2 recovery configuration object.

```
cfg = dvbrcs2RecoveryConfig;
cfg.WaveformID = wg.WaveformID;
```

Initialize a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor', 0.20, ...
    'InputSamplesPerSymbol', sps, 'DecimationFactor', sps);
span = rxFilter.FilterSpanInSymbols;
```

Apply matched filtering and remove the filter delay

```
rxBurst = rxIn(wg.PreBurstGuardLength*sps+1:end-wg.PostBurstGuardLength*sps);
filtOut = rxFilter([rxBurst; ...
    complex(zeros(span/2*sps,1))]);
rxSymb = filtOut(span+1:end);
```

Recover user packets. Display the frame PDU cyclic redundancy check (CRC) status and the numbers of bit errors.

```
[rxOut, pduErr] = dvbrcs2BitRecover(rxSymb, cfg, 10^(-EsNodB/10));
fprintf('Erroneous frame PDU = %d\n', pduErr)
```

```
Erroneous frame PDU = 0
```

```
fprintf('Number of bit errors = %d\n', sum(rxOut~=framePDU))
```

```
Number of bit errors = 0
```

Input Arguments

`rxdata` — Received complex IQ symbols

column vector

Received complex IQ symbols, specified as a column vector. `rxdata` must contain only one burst.

The type of waveform determines the length of `rxdata`.

- Reference waveform — For set values of the TransmissionFormat and WaveformID properties of the `dvbrcs2WaveformGenerator` System object, the length of input `rxdata` must be equal to the burst length parameter specified in ETSI EN 301 545-2 V1.2.1 (2014-11) Table A-1 and A-2 [1].
- Custom waveform — The length must be equal to the value of BurstLength property of the `dvbrcs2RecoveryConfig` object.

Data Types: `double`
Complex Number Support: Yes

`cfgrx` — DVB-RCS2 recovery configuration object

`dvbrcs2RecoveryConfig` object

DVB-RCS2 recovery configuration object, specified as a `dvbrcs2RecoveryConfig` object. The properties of this object specify the transmission parameters of the received waveform and the decoding parameters for the recovery of the data.

`nvar` — Noise variance estimate

nonnegative scalar

Noise variance estimate, specified as a nonnegative scalar. The function uses `nvar` as a scaling factor to calculate the soft bits from the IQ symbols.

When you specify `nvar` as `0`, the function uses a value of 1e-5, which corresponds to a signal-to-noise ratio (SNR) of 50 dB.

Data Types: `double`

Output Arguments

`bits` — Recovered frame PDU data bits

column vector

Recovered frame PDU data bits, returned as a column vector.

Data Types: `int8`

`framePDUErr` — Frame PDU CRC status
`true` or `1` | `false` or `0`

Frame PDU CRC status, returned as a numeric or logical `1` (`true`) or `0` (`false`). A value of `false` indicates the frame is erroneous.

Data Types: `logical`

Version History

Introduced in R2021b

References

[1] ETSI Standard EN 301 545-2 V1.2.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Interactive Satellite Systems (DVB-RCS2); Part 2: Lower Layers for Satellite Standard.*

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

`dvbrcs2RecoveryConfig` | `dvbrcs2WaveformGenerator`

flushFilter

Flush transmit filter

Syntax

```
out = flushFilter(obj)
```

Description

`out = flushFilter(obj)` passes zeros through the transmit filter in the input waveform generator to flush the residual data samples that remain in the filter state. The function returns the residual data samples.

You must call the input waveform generator System object (not only create the object) prior to using the `flushFilter` object function. The number of zeros passed through the transmit filter depends on the filter delay. This object function is required for the receiver simulations to recover all of the bits in the last physical layer frame.

Examples

Get DVB-S2 Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2WaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip', 'file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 1;
```

Create a Digital Video Broadcasting standard (DVB-S2) System object, and then specify its properties.

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.NumInputStreams = 2;
s2WaveGen.MODCOD = [21 16];
s2WaveGen.DFL   = 47008;
s2WaveGen.ISSYI = true;
s2WaveGen.SamplesPerSymbol = 2;
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

             StreamFormat: "TS"
          NumInputStreams: 2
                 FECFrame: "normal"
                   MODCOD: [21 16]
                      DFL: 47008
            ScalingMethod: "outer radius as 1"
                HasPilots: 0
            RolloffFactor: 0.3500
      FilterSpanInSymbols: 10
         SamplesPerSymbol: 2
                    ISSYI: true
               ISCRFormat: "short"

  Show all properties
```

Get the characteristic information about the DVB-S2 waveform generator.

```
info(s2WaveGen)
```

```
ans = struct with fields:
      ModulationScheme: {'16APSK'  '8PSK'}
    LDPCCodeIdentifier: {'5/6'  '8/9'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                        % UP length without sync bits is 1496
data =  cell(1,s2WaveGen.NumInputStreams);
for i = 1:s2WaveGen.NumInputStreams
    numPkts = s2WaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1],pktLen,numPkts);
    ISSY = randi([0 1],16,numPkts);   % ISCRFormat is 'short' by default
                                      % 'short' implies the default length of ISSY as 2 bytes
    txPkts = [repmat(syncBits,1,numPkts);txRawPkts;ISSY];    % ISSY is appended at the end of UP
    data{i} = txPkts(:);
end
```

Generate a DVB-S2 time-domain waveform using the information bits.

```
txWaveform = [s2WaveGen(data)];
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2WaveGen)
```

```
ans = 20×1 complex

   0.0153 + 0.4565i
   0.2483 + 0.5535i
   0.3527 + 0.3972i
   0.3541 - 0.0855i
   0.3505 - 0.4071i
   0.4182 - 0.1962i
   0.5068 + 0.0636i
   0.4856 - 0.1532i
   0.3523 - 0.4153i
   0.1597 - 0.2263i
      ⋮
```

Recover Data Bits from Transport Stream DVB-S2 Transmission

Recover user packets (UPs) for multiple physical layer (PL) frames in a single transport stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) transmission.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream. Create a DVB-S2 System object.

```
nFrames = 2;
s2WaveGen = dvbs2WaveformGenerator;
```

Create the bit vector of information bits, `data`, of concatenated TS UPs.

```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
numPkts = s2WaveGen.MinNumPackets*nFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate the DVB-S2 time-domain waveform using the input information bits. Flush the transmit filter to handle the filter delay and recover the complete last frame.

```
txWaveform = [s2WaveGen(data); flushFilter(s2WaveGen)];
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = s2WaveGen.SamplesPerSymbol;
EsNodB = 1;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,'measured');
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
      'RolloffFactor',s2WaveGen.RolloffFactor, ...
       'InputSamplesPerSymbol',sps,...
       'DecimationFactor',sps);
s = coeffs(rxFilter);
rxFilter.Gain = sum(s.Numerator);
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter(rxIn);
rxFrame = filtOut(rxFilter.FilterSpanInSymbols+1:end);
```

Recover UPs. Display the number of frames lost and the UP cyclic redundancy check (CRC) status.

```
[bits,FramesLost,pktCRCStat] = dvbs2BitRecover(rxFrame,10^(-EsNodB/10));
disp(FramesLost)
```

```
     0
```

```
disp(pktCRCStat)
```

```
    {20×1 logical}
```

Get DVB-S2X Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2xWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 2;
```

Create a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) System object and specify its properties. Use time slicing technique and variable coding and modulation configuration mode.

```
s2xWaveGen = dvbs2xWaveformGenerator();
s2xWaveGen.HasTimeSlicing = true;
s2xWaveGen.NumInputStreams = 2;
s2xWaveGen.PLSDecimalCode = [135 145];   % QPSK 9/20 and 8PSK 25/36
s2xWaveGen.DFL = [18048 44656];
s2xWaveGen.PLScramblingIndex = [0 1];
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

           StreamFormat: "TS"
         HasTimeSlicing: true
        NumInputStreams: 2
         PLSDecimalCode: [135 145]
                    DFL: [18048 44656]
```

```
      PLScramblingIndex: [0 1]
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4
                  ISSYI: false

  Show all properties
```

Get the characteristic information about the DVB-S2X waveform generator.

```
info(s2xWaveGen)
```

```
ans = struct with fields:
              FECFrame: {'normal'  'normal'}
      ModulationScheme: {'QPSK'  '8PSK'}
    LDPCCodeIdentifier: {'9/20'  '25/36'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets for each input stream.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1, s2xWaveGen.NumInputStreams);
for i = 1:s2xWaveGen.NumInputStreams
    numPkts = s2xWaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1], pktLen, numPkts);
    txPkts = [repmat(syncBits, 1, numPkts); txRawPkts];
    data{i} = txPkts(:);
end
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2xWaveGen)
```

```
ans = 40×1 complex

  -0.2412 - 0.0143i
  -0.2619 - 0.0861i
  -0.2726 - 0.1337i
  -0.2511 - 0.1597i
  -0.1851 - 0.1680i
  -0.0780 - 0.1602i
   0.0448 - 0.1288i
   0.1598 - 0.0751i
   0.2482 - 0.0049i
   0.3026 + 0.0702i
      ⋮
```

Get CCSDS TM Waveform Generator Information and Check Transmit Filter Delay

Get information from a `ccsdsTMWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

Create a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) System object. Set the waveform type as `synchronization and channel coding` with low-density parity-check (LDPC) channel coding. Display the properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "LDPC";
tmWaveGen.NumBitsInInformationBlock = 1024;
tmWaveGen.Modulation = "QPSK";
tmWaveGen.CodeRate = "1/2";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

             WaveformSource: "synchronization and channel coding"
              HasRandomizer: true
                     HasASM: true
                  PCMFormat: "NRZ-L"

   Channel coding
              ChannelCoding: "LDPC"
    NumBitsInInformationBlock: 1024
                   CodeRate: "1/2"
               IsLDPCOnSMTF: false

   Digital modulation and filter
                 Modulation: "QPSK"
         PulseShapingFilter: "root raised cosine"
              RolloffFactor: 0.3500
         FilterSpanInSymbols: 10
            SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

```
numTF = 20;
```

Get the characteristic information about the CCSDS TM waveform generator.

```
info(tmWaveGen)
```

```
ans = struct with fields:
         ActualCodeRate: 0.5000
        NumBitsPerSymbol: 2
    SubcarrierFrequency: []
```

Generate the input bits for the CCSDS TM waveform generator, and then generate the waveform.

```
bits = randi([0 1], tmWaveGen.NumInputBits*numTF,1);
waveform = tmWaveGen(bits);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(tmWaveGen)

ans = 100×1 complex

  -0.0772 - 0.0867i
  -0.0751 - 0.0859i
  -0.0673 - 0.0788i
  -0.0549 - 0.0654i
  -0.0388 - 0.0469i
  -0.0200 - 0.0250i
   0.0002 - 0.0012i
   0.0208 + 0.0227i
   0.0405 + 0.0453i
   0.0587 + 0.0653i
      ⋮
```

Input Arguments

obj — Waveform generator
dvbs2WaveformGenerator | dvbs2xWaveformGenerator | ccsdsTMWaveformGenerator

Waveform generator object, specified as a `dvbs2WaveformGenerator`, `dvbs2xWaveformGenerator`, or `ccsdsTMWaveformGenerator` System object.

To enable the `flushFilter` object function when you specify `obj` as a `ccsdsTMWaveformGenerator` System object, you must set these dependencies in the `ccsdsTMWaveformGenerator` object.

- Set the `WaveformSource` property to `"synchronization and channel coding"`.
- Set the `ChannelCoding` property to one of these values.
  - `"none"`
  - `"RS"`
  - `"turbo"`
  - `"LDPC"` — In this case, you must also set the `IsLDPCOnSMTF` property to `0` (`false`)
  - `"convolutional"` — In this case, you must also set the `ConvolutionalCodeRate` property to either `"1/2"` or `"2/3"`
  - `"concatenated"` — In this case, you must also set the `ConvolutionalCodeRate` property to either `"1/2"` or `"2/3"`
- Set the `Modulation` property to either `"BPSK"` or `"QPSK"`.

Output Arguments

out — Residual data samples that remain in filter state
column vector

Residual data samples that remain in the filter state, returned as a column vector. The length of the column vector is equal to the product of the `SamplesPerSymbol` and `FilterSpanInSymbols` properties of the input object, `obj`.

When you specify `obj` as `dvbs2WaveformGenerator` or `dvbs2xWaveformGenerator` System object and the `NumInputStream` property as a value greater than 1, the data fields generated from different input streams are merged in a round-robin technique into a single stream. The residual samples of the frame after the merging process are flushed out.

Data Types: `double`

Version History

Introduced in R2021a

See Also

Functions
`info`

Objects
`ccsdsTMWaveformGenerator` | `dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator`

info

Characteristic information about object

Syntax

```
s = info(obj)
```

Description

`s = info(obj)` returns a structure containing the characteristic information of the specified input object `obj`.

Examples

Get DVB-S2 Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2WaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip', 'file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 1;
```

Create a Digital Video Broadcasting standard (DVB-S2) System object, and then specify its properties.

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.NumInputStreams = 2;
s2WaveGen.MODCOD = [21 16];
s2WaveGen.DFL   = 47008;
s2WaveGen.ISSYI = true;
s2WaveGen.SamplesPerSymbol = 2;
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

           StreamFormat: "TS"
        NumInputStreams: 2
               FECFrame: "normal"
                 MODCOD: [21 16]
```

```
                    DFL: 47008
          ScalingMethod: "outer radius as 1"
              HasPilots: 0
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 2
                  ISSYI: true
             ISCRFormat: "short"

  Show all properties
```

Get the characteristic information about the DVB-S2 waveform generator.

```
info(s2WaveGen)
```

```
ans = struct with fields:
      ModulationScheme: {'16APSK'  '8PSK'}
    LDPCCodeIdentifier: {'5/6'  '8/9'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1,s2WaveGen.NumInputStreams);
for i = 1:s2WaveGen.NumInputStreams
    numPkts = s2WaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1],pktLen,numPkts);
    ISSY = randi([0 1],16,numPkts);   % ISCRFormat is 'short' by default
                                      % 'short' implies the default length of ISSY as 2 bytes
    txPkts = [repmat(syncBits,1,numPkts);txRawPkts;ISSY];    % ISSY is appended at the end of UP
    data{i} = txPkts(:);
end
```

Generate a DVB-S2 time-domain waveform using the information bits.

```
txWaveform = [s2WaveGen(data)];
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2WaveGen)
```

```
ans = 20×1 complex

   0.0153 + 0.4565i
   0.2483 + 0.5535i
   0.3527 + 0.3972i
   0.3541 - 0.0855i
   0.3505 - 0.4071i
   0.4182 - 0.1962i
   0.5068 + 0.0636i
   0.4856 - 0.1532i
   0.3523 - 0.4153i
   0.1597 - 0.2263i
      ⋮
```

Get DVB-S2X Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2xWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 2;
```

Create a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) System object and specify its properties. Use time slicing technique and variable coding and modulation configuration mode.

```
s2xWaveGen = dvbs2xWaveformGenerator();
s2xWaveGen.HasTimeSlicing = true;
s2xWaveGen.NumInputStreams = 2;
s2xWaveGen.PLSDecimalCode = [135 145];   % QPSK 9/20 and 8PSK 25/36
s2xWaveGen.DFL = [18048 44656];
s2xWaveGen.PLScramblingIndex = [0 1];
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

           StreamFormat: "TS"
         HasTimeSlicing: true
        NumInputStreams: 2
         PLSDecimalCode: [135 145]
                    DFL: [18048 44656]
      PLScramblingIndex: [0 1]
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4
                  ISSYI: false

  Show all properties
```

Get the characteristic information about the DVB-S2X waveform generator.

```
info(s2xWaveGen)
```

```
ans = struct with fields:
              FECFrame: {'normal'  'normal'}
      ModulationScheme: {'QPSK'  '8PSK'}
    LDPCCodeIdentifier: {'9/20'  '25/36'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets for each input stream.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                        % UP length without sync bits is 1496
data =  cell(1, s2xWaveGen.NumInputStreams);
for i = 1:s2xWaveGen.NumInputStreams
    numPkts = s2xWaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1], pktLen, numPkts);
    txPkts = [repmat(syncBits, 1, numPkts); txRawPkts];
    data{i} = txPkts(:);
end
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2xWaveGen)
```

```
ans = 40×1 complex

  -0.2412 - 0.0143i
  -0.2619 - 0.0861i
  -0.2726 - 0.1337i
  -0.2511 - 0.1597i
  -0.1851 - 0.1680i
  -0.0780 - 0.1602i
   0.0448 - 0.1288i
   0.1598 - 0.0751i
   0.2482 - 0.0049i
   0.3026 + 0.0702i
      ⋮
```

Get DVB-RCS2 Waveform Generator Information

Get information from a `dvbrcs2WaveformGenerator` System object by using the `info` object function.

Create a DVB-RCS2 System object, and then specify its properties.

```
wg = dvbrcs2WaveformGenerator;
wg.ContentType = "control";
wg.WaveformID = 33;
wg.FilterSpanInSymbols = 12;
disp(wg)
```

```
  dvbrcs2WaveformGenerator with properties:

      TransmissionFormat: "TC-LM"
             ContentType: "control"
        IsCustomWaveform: false
              WaveformID: 33
     PreBurstGuardLength: 0
```

```
    PostBurstGuardLength: 0
     FilterSpanInSymbols: 12
        SamplesPerSymbol: 4

   Read-only:
          FramePDULength: 784
```

Get the characteristic information about the DVB-RCS2 waveform generator.

```
info(wg)
```

```
ans = struct with fields:
              BurstLength: 566
     PayloadLengthInBytes: 100
            MappingScheme: "QPSK"
                 CodeRate: "3/4"
           PreambleLength: 32
          PostambleLength: 0
              PilotPeriod: 0
         PilotBlockLength: 0
    PermutationParameters: [23 10 8 2 1]
               UniqueWord: "0C330C0FF3F3033F"
                 PilotSum: 0
```

Get CCSDS TM Waveform Generator Information and Check Transmit Filter Delay

Get information from a `ccsdsTMWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

Create a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) System object. Set the waveform type as `synchronization and channel coding` with low-density parity-check (LDPC) channel coding. Display the properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "LDPC";
tmWaveGen.NumBitsInInformationBlock = 1024;
tmWaveGen.Modulation = "QPSK";
tmWaveGen.CodeRate = "1/2";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

               WaveformSource: "synchronization and channel coding"
                HasRandomizer: true
                       HasASM: true
                    PCMFormat: "NRZ-L"

   Channel coding
                ChannelCoding: "LDPC"
    NumBitsInInformationBlock: 1024
                     CodeRate: "1/2"
                 IsLDPCOnSMTF: false
   Digital modulation and filter
                   Modulation: "QPSK"
           PulseShapingFilter: "root raised cosine"
                RolloffFactor: 0.3500
          FilterSpanInSymbols: 10
             SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

```
numTF = 20;
```

Get the characteristic information about the CCSDS TM waveform generator.

```
info(tmWaveGen)
```

```
ans = struct with fields:
         ActualCodeRate: 0.5000
       NumBitsPerSymbol: 2
    SubcarrierFrequency: []
```

Generate the input bits for the CCSDS TM waveform generator, and then generate the waveform.

```
bits = randi([0 1], tmWaveGen.NumInputBits*numTF,1);
waveform = tmWaveGen(bits);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(tmWaveGen)
```

```
ans = 100×1 complex

  -0.0772 - 0.0867i
  -0.0751 - 0.0859i
  -0.0673 - 0.0788i
  -0.0549 - 0.0654i
  -0.0388 - 0.0469i
  -0.0200 - 0.0250i
   0.0002 - 0.0012i
   0.0208 + 0.0227i
   0.0405 + 0.0453i
   0.0587 + 0.0653i
      ⋮
```

Get ETSI Rician Channel Information

Get information from a `etsiRicianChannel` System object by using the `info` object function.

Create a European Telecommunication Standards Institute (ETSI) Rician channel System object, and then specify its properties.

```
chan = etsiRicianChannel;
chan.SampleRate = 2e5;
chan.KFactor = 10;
chan.MaximumDopplerShift = 20;
chan.NumSinusoids = 58;
disp(chan)

  etsiRicianChannel with properties:

             SampleRate: 200000
                KFactor: 10
    MaximumDopplerShift: 20

  Use get to show all properties
```

Pass data through the channel.

```
txWaveform = randi([0 1],500,1);
rxWaveform = chan(txWaveform);
```

Get the characteristic information about the ETSI Rician channel.

```
info(chan)

ans = struct with fields:
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 500
```

Get P.681-11 LMS Channel Information

Get channel information from a `p681LMSChannel` System object by using the `info` object function.

Create an ITU-R P.681-11 LMS channel System object and specify its properties.

```
chan = p681LMSChannel;
chan.SampleRate = 10e3;             % Hz
chan.MobileSpeed = 2;               % m/s
chan.Environment = "RuralWooded";
disp(chan)

  p681LMSChannel with properties:

            SampleRate: 10000
          InitialState: "Good"
      CarrierFrequency: 2.2000e+09
        ElevationAngle: 45
           MobileSpeed: 2
    AzimuthOrientation: 0
           Environment: "RuralWooded"
      ChannelFiltering: true

  Use get to show all properties
```

QPSK-modulate a random input signal, and then pass it through the channel.

```
numSamples = 2e4;
txWaveform = pskmod(randi([0 3],numSamples,1),4);
[rxWaveform,pathGains,sampleTimes,stateSeries] = chan(txWaveform);
```

Get the characteristic information about the P.681-11 LMS channel.

```
info(chan)
```

```
ans = struct with fields:
                   PathDelays: 0
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 20000
```

Transmit another QPSK-modulated random input signal through the channel

```
numSamples2 = 3e4;
txWaveform2 = pskmod(randi([0 3],numSamples2,1),4);
[rxWaveform2,pathGains2,sampleTimes2,stateSeries2] = chan(txWaveform2);
```

Observe the change in number of samples processed.

```
info(chan)
```

```
ans = struct with fields:
                   PathDelays: 0
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 50000
```

Get Lutz LMS Channel Information

Get channel information from a `LutzLMSChannel` System object by using the `info` object function.

Create a Lutz LMS channel System object and specify its properties.

```
chan = lutzLMSChannel;
chan.SampleRate = 6000;
chan.KFactor = 20;
chan.MeanStateDuration = [8 2];
disp(chan)
```

```
  lutzLMSChannel with properties:

                   SampleRate: 6000
                 InitialState: "Good"
                      KFactor: 20
               LogNormalFading: [-8.8000 3.8000]
    StateDurationDistribution: "Exponential"
            MeanStateDuration: [8 2]
          MaximumDopplerShift: 4.2807
             ChannelFiltering: true

  Show all properties
```

QPSK-modulate a random input signal, and then pass it through the channel.

```
numSamples = 2e4;
txWaveform = pskmod(randi([0 3],numSamples,1),4);
[rxWaveform,pathGains,sampleTimes,stateSeries] = chan(txWaveform);
```

Get the characteristic information about the Lutz LMS channel.

```
info(chan)
```

```
ans = struct with fields:
                   PathDelays: 0
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 20000
```

Transmit another QPSK-modulated random input signal through the channel

```
numSamples2 = 3e4;
txWaveform2 = pskmod(randi([0 3],numSamples2,1),4);
[rxWaveform2,pathGains2,sampleTimes2,stateSeries2] = chan(txWaveform2);
```

Observe the change in number of samples processed.

```
info(chan)
```

```
ans = struct with fields:
                   PathDelays: 0
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 50000
```

Get P-Code State Information

Get information from a `gpsPCode` System object™ by using the `info` object function. Observe how the precision of initial time impacts the generation of the P-code.

Create a P-code generator System object™, and then specify its properties.

```
format long
pgen = gpsPCode
```

```
pgen = 
  gpsPCode with properties:

                PRNID: 1
     OutputCodeLength: 10230
    InitialStateFormat: "seconds"
          InitialTime: 0
```

```
pgen.InitialStateFormat = "chips";
pgen.InitialNumChipsElapsed = 8388600;
```

Get the characteristic information about the P-code generator.

```
pgen.info
```

```
ans = struct with fields:
    TotalNumChipsElapsed: 8388600
     TotalSecondsElapsed: 0.820000000000000
```

Advance the time by a quarter of a P-code chip time (that is, 0.25/10.23e6).

```
pgen1 = gpsPCode;
pgen1.InitialTime = pgen.info.TotalSecondsElapsed + 0.25/10.23e6
```

```
pgen1 = 
  gpsPCode with properties:

                 PRNID: 1
      OutputCodeLength: 10230
    InitialStateFormat: "seconds"
           InitialTime: 0.820000024437928
```

```
pgen1.info
```

```
ans = struct with fields:
    TotalNumChipsElapsed: 8388600
     TotalSecondsElapsed: 0.820000000000000
```

The `info` function output shows no increment in the `TotalNumChipsElapsed` in this case, because `TotalNumChipsElapsed` is calculated internally using the function `round`.

Advance the time by half of a P-code chip time now (that is, 0.5/10.23e6).

```
pgen2 = gpsPCode;
pgen2.InitialTime = pgen.info.TotalSecondsElapsed + 0.5/10.23e6
```

```
pgen2 = 
  gpsPCode with properties:

                 PRNID: 1
      OutputCodeLength: 10230
    InitialStateFormat: "seconds"
           InitialTime: 0.820000048875855
```

```
pgen2.info
```

```
ans = struct with fields:
    TotalNumChipsElapsed: 8388601
     TotalSecondsElapsed: 0.820000097751711
```

The `info` function output now shows the `TotalNumChipsElapsed` is incremented by one, due to the internal usage of `round()` function.

Compare the output of each System object call.

```
code = pgen();
code1 = pgen1();
```

```
code2 = pgen2();
isequal(code, code1) % code and code1 are equal
```

```
ans = logical
   1
```

```
isequal(code1,code2) % code1 and code2 are unequal
```

```
ans = logical
   0
```

Get GNSS Signal Acquisition Information

Get channel information from a gnssSignalAquirer System object by using the `info` object function.

Load a precomputed GPS waveform.

```
load gnssWaveforms
```

Initialize a GNNS signal Acquirer system object.

```
gsa = gnssSignalAcquirer(IntermediateFrequency=10e6,SampleRate=38.192e6);
```

Search for 32 GPS satellites.

```
Information = gsa(gpsIFWaveform,1:32); % Information about frequency Offset, code-phase Offset, a
```

Get the characteristic information about the GNSS signal acquisition .

```
gsaInfo = info(gsa)
```

```
gsaInfo = struct with fields:
    ReferenceNoiseLevel: 1.2586e+03
```

Input Arguments

obj — Input object
`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator` | `dvbrcs2WaveformGenerator` | `ccsdsTMWaveformGenerator` | `etsiRicianChannel` | `p681LMSChannel` | `lutzLMSChannel` | `gpsPCode` | `gnssSignalAquirer`

Input object to get information from, specified as a `dvbs2WaveformGenerator`, `dvbs2xWaveformGenerator`, `dvbrcs2WaveformGenerator`, `ccsdsTMWaveformGenerator`, `etsiRicianChannel`, `p681LMSChannel`, `lutzLMSChannel`, `gnssSignalAcquirer` or `gpsPCode` System object.

Output Arguments

s — Characteristic information of specified object
structure

Characteristic information of the specified object, returned as a structure. The fields of the structure depend on the `obj` input.

- If `obj` is a `dvbs2WaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the Digital Video Broadcasting Satellite Second Generation (DVB-S2) waveform generator.

Field	Value	Description
`ModulationScheme`	String scalar (default) or cell array of character vectors	Modulation scheme, returned as a string scalar for single-input stream and a cell array of character vectors of length equal to the `NumInputStreams` property of the `dvbs2WaveformGenerator` object for multi-input streams.
`LDPCCodeIdentifier`	String scalar (default) or cell array of character vectors	LDPC code identifier used in forward error correction (FEC), returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of the `dvbs2WaveformGenerator` object for multi-input streams.

- If `obj` is a `dvbs2xWaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) waveform generator.

Field	Value	Description
`FECFrame`	String scalar (default) or cell array of character vectors	FEC frame format, returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of `dvbs2xWaveformGenerator` object for multi-input streams.
`ModulationScheme`	String scalar (default) or cell array of character vectors	Modulation scheme, returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of `dvbs2xWaveformGenerator` object for multi-input streams.

Field	Value	Description
`LDPCCodeIdentifier`	String scalar (default) or cell array of character vectors	LDPC code identifier used in forward error correction (FEC), returned as a string scalar for single-input stream and a cell array of character vectors of length equal to `NumInputStreams` property of `dvbs2xWaveformGenerator` object for multi-input streams.

- If `obj` is a `dvbrcs2WaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) waveform generator.

Field	Value	Description
`BurstLength`	positive integer	Length of the burst, in symbols, prior to the pulse shaping, returned as a positive integer.
`PayloadLengthInBytes`	integer in the range [3, 65,535]	Input data length, in bytes, to the forward error correction (FEC) encoder, returned as an integer in the range [3, 65,535].
`MappingScheme`	`"pi/2-BPSK"`, `"QPSK"`, `"8PSK"`, or `"16QAM"`	Symbol mapping and modulation scheme to generate the DVB-RCS2 waveform, returned as `"pi/2-BPSK"`, `"QPSK"`, `"8PSK"`, or `"16QAM"`.
`CodeRate`	`"1/3"`, `"1/2"`, `"2/3"`, `"3/4"`, `"4/5"`, `"5/6"`, `"6/7"`, or `"7/8"`	Code rate of the channel encoder, returned as `"1/3"`, `"1/2"`, `"2/3"`, `"3/4"`, `"4/5"`, `"5/6"`, `"6/7"`, or `"7/8"`.

Field	Value	Description
`PreambleLength`	integer in the range [0, 255]	Number of preamble symbols that are prefixed to the burst symbols prior to the modulation, returned as an integer in the range [0, 255]. When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of preamble length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of preamble length is chips.
`PostambleLength`	integer in the range [0, 255]	Number of postamble symbols that are suffixed to the burst symbols, prior to the modulation, returned as an integer in the range [0, 255]. When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of preamble length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of preamble length is chips.
`PilotPeriod`	integer in the range [0, 4095]	Pilot symbol periodicity, including the burst symbols, returned as an integer in the range [0, 4095]. This period represents the length of the sequence from the first symbol of a pilot block to the first symbol of the next pilot block in symbols or chips.
`PilotBlockLength`	integer in the range [1, 255]	Length of the pilot block, in symbols, returned as an integer in the range [1, 255].

Field	Value	Description
`PermutationParameters`	five-element vector	DVB-RCS2 turbo encoder permutation control parameters that are used to generate turbo encoder interleaver indices, returned as a five-element vector in order: P, Q_0, Q_1, Q_2, and Q_3.
`UniqueWord`	character array or string scalar	Hexadecimal string consisting of combined symbols of the preamble, one pilot block, and the postamble sequence, returned as a character array or string scalar.

- If `obj` is a `ccsdsTMWaveformGenerator` System object, the output structure has these fields, consisting of physical layer information about the Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) waveform generator.

Field	Value	Description
`ActualCodeRate`	positive scalar in range [0 1]	Numeric value of the code rate of the channel coding scheme, returned as a positive scalar in the range [0, 1]. This value is used to generate the CCSDS TM waveform.
`NumBitsPerSymbol`	positive integer	Number of bits per modulated symbol, returned as a positive integer.
`SubcarrierFrequency`	positive scalar	Subcarrier frequency, returned as a positive scalar. This field is applicable only when the `Modulation` property of `ccsdsTMWaveformGenerator` object is set to `"PCM/PSK/PM"`. For other cases, this value is returned as null.

- If `obj` is an `etsiRicianChannel` System object, the output structure has these fields, consisting of information about the fading channel.

Field	Value	Description
`ChannelFilterDelay`	`0`	Channel filter delay in samples returned as `0` always (due to flat fading nature of the channel).

Field	Value	Description
`ChannelFilterCoefficients`	1	Channel filter coefficient used to convert path gains to channel filter tap gains, returned as `1` always (as `etsiRicianChannel` describes a single path channel).
`NumSamplesProcessed`	nonnegative integer	Number of samples processed by the channel object since the last reset, returned as a positive integer.

- If `obj` is a `p681LMSChannel` System object, the output structure has these fields, consisting of information about the ITU-R P.681-11 land-mobile satellite (LMS) fading channel.

Field	Value	Description
`PathDelays`	0	Delay of discrete channel path in seconds returned as `0` always (due to flat fading nature of the channel).
`ChannelFilterDelay`	0	Channel filter delay in samples returned as `0` always (due to flat fading nature of the channel).
`ChannelFilterCoefficients`	1	Channel filter coefficient used to convert path gains to channel filter tap gains, returned as `1` always (as `p681LMSChannel` describes a single path channel).
`NumSamplesProcessed`	nonnegative integer	Number of samples processed by the channel object since the last reset, returned as a nonnegative integer.

- If `obj` is a `lutzLMSChannel` System object, the output structure has these fields, consisting of information about the Lutz LMS fading channel.

Field	Value	Description
`PathDelays`	0	Delay of discrete channel path in seconds returned as `0` always (due to flat fading nature of the channel).
`ChannelFilterDelay`	0	Channel filter delay in samples returned as `0` always (due to flat fading nature of the channel).

Field	Value	Description
`ChannelFilterCoefficients`	`1`	Channel filter coefficient used to convert path gains to channel filter tap gains, returned as `1` always (as `lutzLMSChannel` describes a single path channel).
`NumSamplesProcessed`	nonnegative integer	Number of samples processed by the channel object since the last release or reset, returned as a nonnegative integer.

- If `obj` is a `gpsPCode` System object, the output structure has these fields, consisting of state information about the GPS P-code generator.

Field	Value	Description
`TotalNumChipsElapsed`	positive integer	Total number of P-code chips that elapsed from the beginning of the week, returned as a positive integer. The beginning of a week is marked at midnight Saturday night - Sunday morning.
`TotalSecondsElapsed`	real-valued scalar	Total seconds elapsed from the beginning of the week, returned as a real-valued scalar.

- If `obj` is a `gnssSignalAcquirer` System object, the output structure has this field, consisting of information about the GNSS signal acquisition.

Field	Value	Description
ReferenceNoiseLevel	numeric scalar	This field represents the reference noise level in the incoming signal. The info function calculates the reference noise level value by correlating the incoming signal with a C/A code. This C/A code is not present in the signal. The function uses G1 code to generate a C/A-code. The reference noise level is the peak of such correlation values.

Version History

Introduced in R2021a

See Also

Functions
`flushFilter`

Objects
`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator` | `dvbrcs2WaveformGenerator` | `ccsdsTMWaveformGenerator` | `etsiRicianChannel` | `p681LMSChannel` | `lutzLMSChannel` | `gpsPCode`

read

Read next VITA 49 packet from file

Syntax

```
[signalDataPacket,contextPacket,contextPacketChangeIndex] = read(
vita49ReaderObj)
[ ___ ] = read(vita49ReaderObj,Name=Value)
```

Description

`[signalDataPacket,contextPacket,contextPacketChangeIndex] = read( vita49ReaderObj)` reads the next packet from the VMEbus International Trade Association (VITA) 49 file specified by the input VITA 49 file reader object, `vita49ReaderObj`, and returns the signal data packet, `signalDataPacket`, and context packet, `contextPacket`. The function also returns the starting indices of the signal data packets after any new context packet arrival, in `contextPacketChangeIndex`. Note that the `read` function will terminate if it finds any ill-formatted packet while reading the file.

`[ ___ ] = read(vita49ReaderObj,Name=Value)` specifies one or more optional name-value arguments, in addition to the output arguments from the previous syntax. For example, `StreamID=2` sets the stream identifier to 2.

Examples

Read Specified Number of Packets from File

Create a VITA 49 file reader object, specifying the name of a VITA 49 file and an output format for the packet timestamp.

```
vita49ReaderObj = vita49Reader("VITA49SampleData.bin");
vita49ReaderObj.OutputTimestampFormat = "seconds";
```

Specify the number of packets to be read from the file.

```
numpkt = 11;
```

Read the specified number of packets from the VITA 49 file to the MATLAB® workspace.

```
[signalDataPacket,contextPacket,contextPacketChangeIndex] = read(vita49ReaderObj,NumPackets=numpl
```

```
signalDataPacket = struct with fields:
                  PacketType: 1
                    StreamID: 0
                     ClassID: "7C386C0000"
                 PadBitCount: 0
        IntegerTimestampType: "GPS"
       IntegerTimestampValue: 1625215654
     FractionalTimestampType: "real time"
    FractionalTimestampValue: 900000344000
```

```
                     RawBytes: [1472x1 uint8]
                    IQSamples: [722x1 double]
                      Trailer: [1x1 struct]

contextPacket=1×10 struct array with fields:
    PacketType
    StreamID
    ClassID
    IntegerTimestampType
    IntegerTimestampValue
    FractionalTimestampType
    FractionalTimestampValue
    RawBytes
    ContextFieldChangeIndicator
    ReferencePointIdentifier
    Bandwidth
    IFReferenceFrequency
    RFFrequency
    RFFrequencyOffset
    IFBandOffset
    ReferenceLevel
    Gain
    OverRangeCount
    SampleRate
    TimestampAdjustment
    TimestampCalibrationTime
    StateAndEventIndicator
    SignalDataPayloadFormat
      ⋮

contextPacketChangeIndex = 1×10

     0     0     0     0     0     0     0     0     0     1
```

Read Packets in Streaming Mode from File

Create a VITA 49 file reader object, specifying the name of a VITA 49 file.

```
vita49ReaderObj = vita49Reader("VITA49SampleData.bin");
```

Set the stream identifier as 1, class identifier as "736C860000", and packet type as signal data packet.

```
pktType = "signal data";
streamID = 0;
classID = "7C386C0000";
```

In streaming mode, read the VITA 49 packets that match the specified filters to the MATLAB workspace.

```
for idx = 1:3
    signalDataPacket = read(vita49ReaderObj, ...
      PacketType=pktType,StreamID=streamID,ClassID=classID)
end

signalDataPacket = struct with fields:
                  PacketType: 1
                    StreamID: 0
                     ClassID: "7C386C0000"
                 PadBitCount: 0
        IntegerTimestampType: "GPS"
       IntegerTimestampValue: 1625215654
     FractionalTimestampType: "real time"
    FractionalTimestampValue: 900000344000
                    RawBytes: [1472x1 uint8]
                   IQSamples: [722x1 double]
                     Trailer: [1x1 struct]


signalDataPacket = struct with fields:
                  PacketType: 1
                    StreamID: 0
                     ClassID: "7C386C0000"
                 PadBitCount: 0
        IntegerTimestampType: "GPS"
       IntegerTimestampValue: 1625215654
     FractionalTimestampType: "real time"
    FractionalTimestampValue: 900042328000
                    RawBytes: [1472x1 uint8]
                   IQSamples: [722x1 double]
                     Trailer: [1x1 struct]


signalDataPacket = struct with fields:
                  PacketType: 1
                    StreamID: 0
                     ClassID: "7C386C0000"
                 PadBitCount: 0
        IntegerTimestampType: "GPS"
       IntegerTimestampValue: 1625215654
     FractionalTimestampType: "real time"
    FractionalTimestampValue: 900084248000
                    RawBytes: [1472x1 uint8]
                   IQSamples: [722x1 double]
                     Trailer: [1x1 struct]
```

Input Arguments

`vita49ReaderObj` — VITA 49 file reader
`vita49Reader` object

VITA 49 file reader, specified as a `vita49Reader` object.

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: `StreamID=2` sets the stream identifier to 2.

`PacketType` — Type of packet to be read
`"signal data"` | `"context"`

Type of packet to be read, specified as `"signal data"` or `"context"`. If you do not specify this argument, the function reads the next packet from the file.

Data Types: `char` | `string`

`NumPackets` — Number of packets to be read
`1` (default) | positive integer

Number of packets to be read, specified as a positive integer. To read all packets up to the end of the file, set `NumPackets` to `intmax("uint32")` or `intmax("uint64")`.

Data Types: `double` | `uint32` | `uint64`

`StreamID` — Stream identifier
nonnegative integer

Stream identifier, specified as a nonnegative integer.

If you specify `StreamID`, the function identifies the packets belonging to only the specified ID from the packet stream.

Data Types: `uint32`

`ClassID` — Class identifier
character vector | string scalar

Class identifier, specified as a character vector or string scalar.

A class identifier is a 5-byte-long, hex-formatted string that includes three bytes of Organizational Unique Identifier (OUI) information and two bytes of information class code.

Data Types: `char` | `string`

Output Arguments

`signalDataPacket` — Decoded signal data packet
structure

Decoded signal data packet, returned as a structure containing these fields.

Field	Description	Value
`PacketType`	Type of VITA 49 packet, returned as one of these values. • `0`, `1`, `2`, or `3` — Indicates signal time data packet • `4` or `5` — Indicates context packet	Integer in the range [0, 5]
`StreamID`	Stream ID, as specified by the name-value argument `StreamID`.	32-bit unsigned integer

Field	Description	Value
`ClassID`	Class ID, as specified by the name-value argument `ClassID`.	Character vector or string scalar
`PadBitCount`	Pad bit count is the difference between the nearest multiple of 32 and the actual number of payload bits to pack, returned as a nonnegative integer. Pad bits are required when the size and number of data items do not completely fill the last 32-bit word of a data packet.	8-bit unsigned integer
`IntegerTimestampType`	Integer timestamp type, returned as `"UTC"`, `"GPS"`, or a user-specified time-code.	Character vector or string scalar
`IntegerTimestampValue`	Integer timestamp value, in seconds. This value represents the Reference Point Time of data samples or metadata in the packet, returned to 1-second resolution accuracy.	32-bit unsigned integer
`FractionalTimestampType`	Fractional timestamp, returned as `"sample count"`, `"real time"`, or `"free running count"`. When this value is `"sample count"` or `"real time"`, the `FractionalTimestampValue` value adds resolution to the `IntegerTimestampValue` value, so that together they provide a range of years and a precision down to sample period or picoseconds, respectively. When returned as `"free running count"`, the value provides an incrementing sample count from any chosen starting time.	Character vector or string scalar
`FractionalTimestampValue`	Fractional timestamp value in picoseconds, represents the Reference Point Time of data samples or metadata in the packet, returned as a timestamp of higher resolution than the integer timestamp value.	64-bit unsigned integer
`RawBytes`	Raw payload in bytes, which is not decoded, returned as a column vector.	Column vector with 8-bit unsigned integer elements
`IQSamples`	Decoded real or complex in-phase quadrature (IQ) Cartesian samples of the signal data, returned as a column vector. This function does not support decoding of complex Polar samples.	Column vector with elements of data type `double`

Field	Description	Value
`Trailer`	Validity of the decoded data and the status of the processes producing that data, returned as a structure with these fields.	Structure

Field	Description
`CalibratedTimeIndicator`	Calibrated time indicator, returned as one of these logical values. • `1` — Indicates the timestamp in the signal data packet is calibrated to some external reference • `0` — Indicates the timestamp is free running and may be inaccurate
`ValidDataIndicator`	Valid data indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates the data packet as valid.
`ReferenceLockIndicator`	Reference lock indicator, returned as one of these logical values. • `1` — Indicates any phase-locked loops (PLL) affecting the data are locked and stable • `0` — Indicates at least one PLL is not locked and stable
`AGCIndicator`	AGC indicator, returned as one of these logical values. • `1` — Indicates AGC is active • `0` — Indicates MGC is active
`MGCIndicator`	MGC indicator, returned as one of these logical values. • `1` — Indicates MGC is active • `0` — Indicates AGC is active

Field	Description		Value
	Field	**Description**	
	`DetectedSignalIndicator`	Detected signal indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates that the data contained in the packet has some detected signal.	
	`SpectrumInversionIndicator`	Spectrum inversion indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates that the signal conveyed in the data payload has an inverted spectrum with respect to the spectrum of the signal at the system Reference Point.	
	`OverRangeIndicator`	Over range indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates that at least one data sample in the payload is invalid due to the signal exceeding the range of the data item.	
	`SampleLossIndicator`	Sample loss indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates that the packet contains at least one sample discontinuity due to processing errors or buffer overflow.	
	`AssociatedContextPacketCount`	Associated context packet count, returned as a 7-bit unsigned integer. This count includes the associated context packets transmitted by a process other than the one that transmits the signal data packet containing the count.	

Data Types: `struct`

`contextPacket` — Decoded context packet
structure

Decoded context packet, returned as a structure containing these fields.

Field	Description	Value
`PacketType`	Type of VITA 49 packet, returned as one of these values. • `0`, `1`, `2`, or `3` — Indicates signal time data packet • `4` or `5` — Indicates context packet	Integer in the range [0, 5]
`StreamID`	Stream ID, as specified by the name-value argument `StreamID`.	32-bit unsigned integer
`ClassID`	Class ID, as specified by the name-value argument `ClassID`.	Character vector or string scalar
`IntegerTimestampType`	Integer timestamp type, returned as `"UTC"`, `"GPS"`, or a user-specified time-code.	Character vector or string scalar
`IntegerTimestampValue`	Integer timestamp value, in seconds. This value represents the Reference Point Time of data samples or metadata in the packet, returned to 1-second resolution accuracy.	32-bit unsigned integer
`FractionalTimestampType`	Fractional timestamp, returned as `"sample count"`, `"real time"`, or `"free running count"`. When this value is `"sample count"` or `"real time"`, the `FractionalTimestampValue` value adds resolution to the `IntegerTimestampValue` value, so that together they provide a range of years and a precision down to sample period or picoseconds, respectively. When returned as `"free running count"`, the value provides an incrementing sample count from any chosen starting time.	Character vector or string scalar

Field	Description	Value
`FractionalTimestampValue`	Fractional timestamp value in picoseconds, represents the Reference Point Time of data samples or metadata in the packet, returned as a timestamp of higher resolution than the integer timestamp value.	64-bit unsigned integer
`RawBytes`	Raw payload in bytes, which is not decoded, returned as a column vector.	Column vector with 8-bit unsigned integer elements
`ContextFieldChangeIndicator`	Bit indicator returned as `0` (`false`) or `1` (`true`). A value of `1` indicates that at least one context field contains a new value.	`0` (`false`) or `1` (`true`)
`ReferencePointIdentifier`	Reference point ID consists of the `StreamID` of the reference point, returned as a nonnegative integer.	32-bit unsigned integer
`Bandwidth`	Usable spectrum at the output of a communication channel, returned as a nonnegative scalar in hertz.	Nonnegative scalar
`IFReferenceFrequency`	Intermediate frequency (IF) at which a carrier wave shifts as an intermediate step in transmission or reception of the signal, returned as a real scalar in hertz.	Real scalar
`RFFrequency`	Location in the signal path that corresponds to the original frequency, returned as a real scalar in hertz.	Real scalar
`RFFrequencyOffset`	Intentional slight deviation of the broadcast radio frequency (RF) to reduce the interference with other transmitters, returned as a real scalar in hertz.	Real scalar
`IFBandOffset`	IF offset from the `IFReferenceFrequency` to the center of the band, returned as a real scalar in hertz.	Real scalar

Field	Description	Value
`ReferenceLevel`	Physical signal amplitude at a reference point relative to the corresponding data sample value, returned as a real scalar in dB.	Real scalar
`Gain`	Amount of signal gain or signal attenuation from the reference point, returned as a real scalar in dB.	Real scalar
`OverRangeCount`	Number of data samples in the paired data packet whose amplitudes are beyond the range of the data item format, returned as a nonnegative integer in dB.	32-bit unsigned integer
`SampleRate`	Sampling rate of the data samples in the payload of a paired data packet stream, returned as a positive integer in hertz.	Positive integer

Field	Description	Value
`TimeStampAdjustment`	Delay used to adjust the timestamp information of the first packet of the file, returned as a structure with these fields.	Structure

Field	Description
`GlobalTimestamp`	Global timestamp indicator, returned as one of these logical values. • `1` — Indicates the timestamp details apply globally to all packets in the information stream • `0` — Indicates the timestamp details apply only to the context stream and the paired data stream
`TSECode`	Indicates the type of `TimestampEpoch` provided, returned as `"Unspecified"`, `"UTC"`, `"GPS"`, or `"POSIX"`.
`TimestampEpoch`	Conveys the time in one of these formats, depending on the value of

Field	Description	Value
	<table><tr><th>Field</th><th>Description</th></tr><tr><td></td><td>the `TSECode` field.<ul><li>If the `TSECode` field is returned as `"UTC"`, then timestamp epoch conveys time in the UTC epoch. The value returned provides the time of the start of the epoch in international system of units (SI) seconds, including leap seconds, since 1970-01-01 T00:00:00Z (UTC).</li><li>If the `TSECode` field is returned as `"GPS"`, then timestamp epoch conveys time in the GPS epoch. The value returned provides the time of the start of the epoch</li></ul></td></tr></table>	

Field	Description	Value
	<table><tr><th>Field</th><th>Description</th></tr><tr><td></td><td>used in SI seconds, since 1980-01-06 T00:00:00Z (UTC). Leap seconds are not applicable within the GPS epoch. • If the `TSECode` field is returned as `"POSIX"`, then timestamp epoch conveys time in the POSIX epoch. The value returned provides the time of the start of the epoch used in nominal seconds, since 1970-01-01 T00:00:00Z (UTC). Leap seconds are not applicable within the POSIX epoch.</td></tr><tr><td>`LeapSecondHandle`</td><td>Indicates how the leap seconds are handled in the</td></tr></table>	

Field	Description		Value
	Field	Description	
		packet timestamps, returned as `"Not applicable"`, `"normal"`, `"duplication"`, or `"overflow"`.	
	`SecScheduledPerDay`	Indicates the number of seconds in the current day denoted by the packet timestamps, returned as `0`, `86399`, `86400`, or `86401` seconds.	
	`TimeSource`	Indicates the time reference source used, returned as `"Unspecified"`, `"Atomic Clock"`, `"Satellite System"`, `"Terrestrial Radio"`, `"PTP"`, `"NTP or SNTP"`, or `"Not Defined"`.	
	`PosixTimebufferOffset`	Conveys the difference in seconds between UTC time and POSIX time, returned as a 8-bit unsigned integer. This value represents the current total	

Field	Description	Value
	Field / **Description** (Field empty) / leap seconds count.	
`TimestampCalibrationTime`	Conveys the date and time at which the timestamp in the data signal packet and context packet was confirmed accurate, returned as a nonnegative integer.	32-bit unsigned integer

Field	Description	Value
`StateAndEventIndicator`	Conveys a set of binary indications and a limited number of non-binary state indications, returned as a structure with these fields. This structure field is equivalent to be the `Trailer` field of the `signalDataPacket` output argument structure. <table><tr><th>Field</th><th>Description</th></tr><tr><td>`CalibratedTimeIndicator`</td><td>Calibrated time indicator, returned as one of these logical values. • `1` — Indicates the timestamp in the signal data packet is calibrated to some external reference • `0` — Indicates the timestamp is free running and may be inaccurate</td></tr><tr><td>`ValidDataIndicator`</td><td>Valid data indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates the data packet as valid.</td></tr><tr><td>`ReferenceLockIndicator`</td><td>Reference lock indicator, returned as</td></tr></table>	Structure

Field	Description	Value
	<table><tr><th>Field</th><th>Description</th></tr><tr><td></td><td>one of these logical values.<ul><li>`1` — Indicates any phase-locked loops (PLL) affecting the data are locked and stable</li><li>`0` — Indicates at least one PLL is not locked and stable</li></ul></td></tr><tr><td>`AGCIndicator`</td><td>AGC indicator, returned as one of these logical values.<ul><li>`1` — Indicates AGC is active</li><li>`0` — Indicates MGC is active</li></ul></td></tr><tr><td>`MGCIndicator`</td><td>MGC indicator, returned as one of these logical values.<ul><li>`1` — Indicates MGC is active</li><li>`0` — Indicates AGC is active</li></ul></td></tr><tr><td>`DetectedSignalIndicator`</td><td>Detected signal indicator, returned as a</td></tr></table>	

Field	Description		Value
	Field	Description	
		logical value of 1 or 0. When returned as 1, it indicates that the data contained in the packet has some detected signal.	
	`SpectrumInversionIndicator`	Spectrum inversion indicator, returned as a logical value of 1 or 0. When returned as 1, it indicates that the signal conveyed in the data payload has an inverted spectrum with respect to the spectrum of the signal at the system Reference Point.	
	`OverRangeIndicator`	Over range indicator, returned as a logical value of 1 or 0. When returned as 1, it indicates that at least one data sample in the payload is invalid due to the signal exceeding the range of the data item.	
	`SampleLossIndicator`	Sample loss indicator, returned as a	

Field	Description		Value
	Field	Description	
		logical value of `1` or `0`. When returned as `1`, it indicates that the packet contains at least one sample discontinuity due to processing errors or buffer overflow.	
	`AssociatedContextPacketCount`	Associated context packet count, returned as a 7-bit unsigned integer. This count includes the associated context packets transmitted by a process other than the one that transmits the signal data packet containing the count.	

Field	Description	Value
`SignalDataPayloadFormat`	Consists of the format of the real or complex signal data (of either 8, 16, or 32 bits) and the data item size, returned as a structure with these fields.	Structure

Field	Description
`LinkEfficientPacking`	Link-efficient packing indicator, returned as one of these logical values. • `1` — Indicates link-efficient packing is used in the paired data packet stream • `0` — Indicates processing-efficient packing is used
`DataSampleType`	Indicates whether the data samples are real or complex Cartesian samples.
`DataItemFormat`	Five-bit code that indicates the type of data items used in the paired data packet stream. This function supports the decimal values of `0` (signed fixed-point) and `16`

Field	Description	Value
	<table><tr><th>Field</th><th>Description</th></tr><tr><td></td><td>(unsigned fixed-point).</td></tr><tr><td>`RepeatCountIndicator`</td><td>Repeat count indicator, returned as a logical value of `1` or `0`. When returned as `1`, it indicates sample component repeating in the paired data packet stream.</td></tr><tr><td>`EventTagSize`</td><td>Event tag size field value used in the paired data packet stream, returned as a 3-bit unsigned integer.</td></tr><tr><td>`ChannelTagSize`</td><td>Channel tag size field value used in the paired data packet stream, returned as a 4-bit unsigned integer.</td></tr><tr><td>`DataItemFractionSize`</td><td>Number of bits in the fraction of a non-normalized number of the format *integer.fraction*, where the total size of the number is given by the data item size field. The value is returned as a 4-bit unsigned integer.</td></tr></table>	

Field	Description		Value
	Field	**Description**	
	`ItemPackingFieldSize`	Item packing field size, returned as a 6-bit unsigned integer. The value is one less than the actual item packing size used in the paired data packet stream.	
	`DataItemSize`	Data item size, returned as a 6-bit unsigned integer. This value is one less than the actual data item size used in the paired data packet stream.	
	`RepeatCount`	Repeat count, returned as a 16-bit unsigned integer. This value is one less than the actual repeat count used in the paired data packet stream.	
	`VectorSize`	Vector size, returned as a 16-bit unsigned integer. This value is one less than the actual vector size used in the paired data packet stream.	

Data Types: `struct`

`contextPacketChangeIndex` — Starting indices of signal data packets after any new context packet arrival
array of nonnegative integers

Starting indices of the signal data packets after any new context packet arrival, returned as an array of nonnegative integers.

When two context packets arrive one after another, the corresponding value returned is two zeros.

Data Types: `double`

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Code generation is available only when the `OutputTimestampFormat` property of the `vita49Reader` object is set to the default value of `"seconds"`.

See Also

Objects
`vita49Reader`

Functions
`reset`

Topics
"VITA 49 File Reader"

reset

Reset VITA 49 file reader to first VITA 49 packet of file

Syntax

```
reset(vita49ReaderObj)
```

Description

`reset(vita49ReaderObj)` resets the VITA 49 file reader to the first VITA 49 packet of the file. This function resets the number of signal data and context packets traversed to zero.

Examples

Reset VITA 49 File Reader

Create a VITA 49 file reader object, specifying the name of a VITA 49 file.

```
vita49ReaderObj = vita49Reader("VITA49SampleData.bin");
```

Specify the number of packets to be read.

```
numpkt = 4;
```

Read the specified number of packets to the MATLAB® workspace.

```
[signalDataPacket,contextPacket,contextPacketChangeIndex] = read(vita49ReaderObj,NumPackets = nur
```

Display the number of packets read by the object. Because you specified for the function to read 4 packets, the number of packets read is 4.

```
numPacketsRead = vita49ReaderObj.PacketsRead
```

```
numPacketsRead = 4
```

Reset the position of the VITA 49 file reader to the first packet of the VITA 49 file.

```
reset(vita49ReaderObj);
```

Display the number of packets read by the object. Since the `reset` object function resets the position of the file reader object to the first VITA 49 packet of the file, the value reflects `0` packets read. Read a packet from the object..

```
numPacketsRead1 = vita49ReaderObj.PacketsRead
```

```
numPacketsRead1 = 0
```

```
[signalDataPacket1,contextPacket1,contextPacketChangeIndex1] = read(vita49ReaderObj);
```

Display the number of packets read. Because you did not specify a number of packets to read, only the first packet of the VITA 49 file has been read.

```
numPacketsRead2 = vita49ReaderObj.PacketsRead
```

```
numPacketsRead2 = 1
```

Input Arguments

`vita49ReaderObj` — VITA 49 file reader
`vita49Reader` object

VITA 49 file reader, specified as a `vita49Reader` object.

Version History
Introduced in R2022b

See Also

Objects
`vita49Reader`

Functions
`read`

Topics
“VITA 49 File Reader”

satelliteCNR

Carrier-to-noise ratio for configured satellite link budget parameters

Syntax

```
[cn,info] = satelliteCNR(cfg)
```

Description

`[cn,info] = satelliteCNR(cfg)` computes the carrier-to-noise ratio (CNR) `cn` for the satellite link budget parameters specified in the configuration `cfg`. The function also returns the intermediate results in the CNR calculation `info` based on the link budget parameters specified in `cfg`.

Examples

Calculate CNR and Link Margin

Calculate the CNR and the received link margin for the specified link budget parameters.

Create a default CNR configuration object, and then set its properties.

```
cfg = satelliteCNRConfig;
cfg.TransmitterPower = 17;                % in dBW
cfg.TransmitterSystemLoss = 9;            % in dB
cfg.TransmitterAntennaGain = 38;          % in dBi
cfg.Distance = 40215;                     % in km
cfg.Frequency = 11;                       % in GHz

% Here, miscellaneous losses include polarization loss, interference
% loss, and antenna mispointing loss, respectively.
polLoss = 3.0103;
intLoss = 2;
antLoss = 1;
cfg.MiscellaneousLoss =  polLoss + intLoss + antLoss; % in dB
cfg.GainToNoiseTemperatureRatio = 25;                 % in dB/K
cfg.ReceiverSystemLoss = 2;                           % in dB
cfg.BitRate = 10;                                     % in Mbps
```

Display the CNR configuration object properties.

```
disp(cfg)
```

```
  satelliteCNRConfig with properties:

             TransmitterPower: 17
        TransmitterSystemLoss: 9
       TransmitterAntennaGain: 38
                     Distance: 40215
                    Frequency: 11
            MiscellaneousLoss: 6.0103
  GainToNoiseTemperatureRatio: 25
```

```
            ReceiverSystemLoss: 2
                       BitRate: 10
                    SymbolRate: 10
                     Bandwidth: 6
```

Calculate the CNR.

```
[cn,info] = satelliteCNR(cfg)
```

```
cn = 18.4440

info = struct with fields:
               TransmitterEIRP: 46
                          FSPL: 205.3634
        ReceivedIsotropicPower: -165.3737
    CarrierToNoiseDensityRatio: 86.2255
                  ReceivedEbNo: 16.2255
                  ReceivedEsNo: 16.2255
```

Compute the link margin. Assume a required energy per bit to noise power density ratio (Eb/No) of 10 dB and an implementation loss of 2 dB in the receiver.

```
reqEbNo = 10;
implLoss = 2;
margin = info.ReceivedEbNo - reqEbNo - implLoss
```

```
margin = 4.2255
```

Input Arguments

`cfg` — CNR configuration object
`satelliteCNRConfig` object

CNR configuration object, specified as a `satelliteCNRConfig` object.

Output Arguments

`cn` — Carrier-to-noise ratio
scalar

Carrier-to-noise ratio in dB, returned as a scalar.

Data Types: `double`

`info` — Intermediate results in CNR calculation
structure

Intermediate results in the CNR calculation, based on the link budget parameters specified in the configuration object `cfg` returned as a structure with these fields.

Structure Field	Description
`TransmitterEIRP`	Effective isotropic radiated power (EIRP) of the transmitter antenna, returned as a scalar. Value is in dBW.
`FSPL`	Free space path loss (FSPL) from the transmitter to the receiver antenna, returned as a scalar. Value is in dB.
`ReceivedIsotropicPower`	Received isotropic power at the receiver antenna, returned as a scalar. Value is in dBW.
`CarrierToNoiseDensityRatio`	Carrier-to-noise power density ratio (C/No), returned as a scalar. Value is in dB-Hz.
`ReceivedEbNo`	Received energy per bit to noise power density ratio (Eb/No), returned as a scalar. Value is in dB.
`ReceivedEsNo`	Received energy per symbol to noise power density ratio (Es/No), returned as a scalar. Value is in dB.

All fields in the structure are of data type `double`.

Data Types: `struct`

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

`satelliteCNRConfig`

satellite

Add satellites to satellite scenario

Syntax

```
satellite(scenario,file)
satellite(scenario,RINEXdata)
satellite(scenario,semimajoraxis,eccentricity,inclination,RAAN,
argofperiapsis,trueanomaly)
satellite(scenario,positiontable)
satellite(scenario,positiontable,velocitytable)
satellite(scenario,positiontimeseries)
satellite(scenario,positiontimeseries,velocitytimeseries)
satellite( ___ ,Name,Value)
sat = satellite( ___ )
```

Description

`satellite(scenario,file)` adds `Satellite` objects from `file` to the satellite scenario specified by `scenario`. The yaw (*z*) axes of the satellites point toward nadir and the roll (*x*) axes of the satellites align with their respective inertial velocity vectors.

`satellite(scenario,RINEXdata)` adds `Satellite` objects from `RINEXdata` to the satellite scenario specified by `scenario`.

`satellite(scenario,semimajoraxis,eccentricity,inclination,RAAN, argofperiapsis,trueanomaly)` adds a `Satellite` object from Keplerian elements defined in the Geocentric Celestial Reference Frame (GCRF) to the satellite scenario.

`satellite(scenario,positiontable)` adds a `Satellite` object from position data specified in `positiontable` (`timetable` object) to the scenario. This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite(scenario,positiontable,velocitytable)` adds a `Satellite` object from position data specified in `positiontable` (`timetable` object) and velocity data specified in `velocitytable` (`timetable` object) to the scenario. This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite(scenario,positiontimeseries)` adds a `Satellite` object from position data specified in `positiontimeseries` to the `scenario`. This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite(scenario,positiontimeseries,velocitytimeseries)` adds a `Satellite` object to the `scenario` from position (in meters) data specified in `positiontimeseries` (`timeseries` object) and velocity (in meters/second) data specified in `velocitytimeseries` (`timeseries` object). This function creates a `Satellite` with `OrbitPropagator="ephemeris"`.

`satellite( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to any input argument combination from previous syntaxes.

`sat = satellite( ___ )` returns a vector of handles to the added satellites. Specify any input argument combination from previous syntaxes.

Note When the `AutoSimulate` property of the `satelliteScenario` is `false`, you can modify the `satellite` only when the `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Examples

Visualize Satellite Trajectories

Create a satellite scenario object.

```
sc = satelliteScenario;
```

Load the satellite ephemeris timetable in the Earth Centered Earth Fixed (ECEF) coordinate frame.

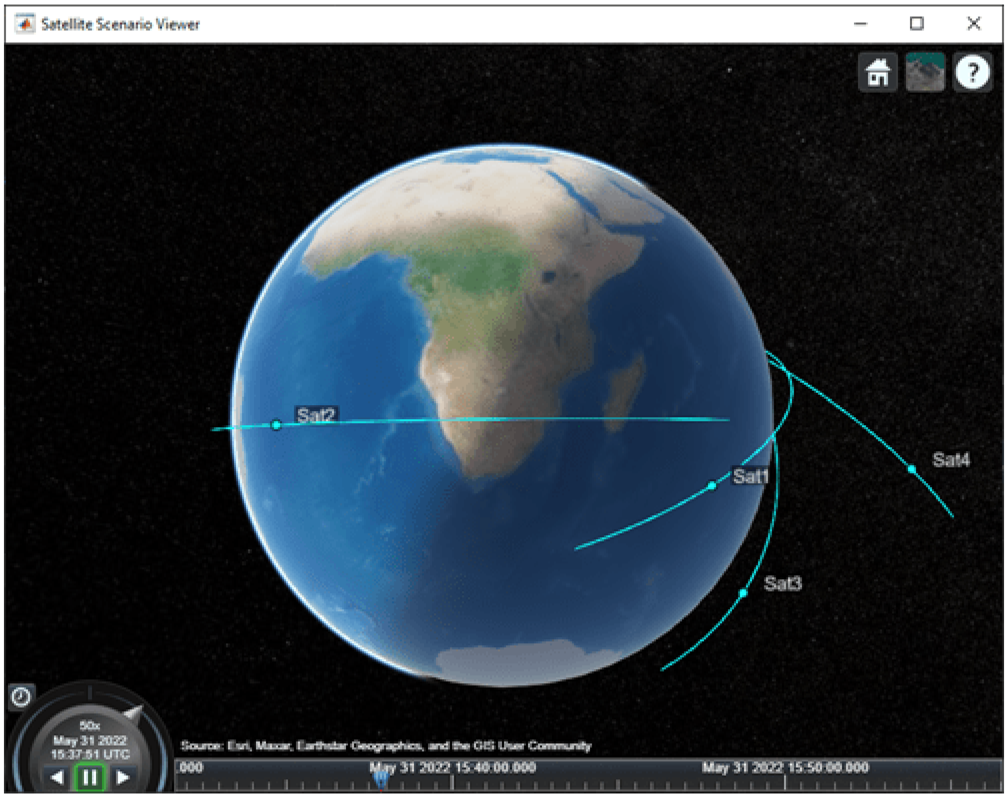
```
load("timetableSatelliteTrajectory.mat","positionTT","velocityTT");
```

Add four satellites to the scenario.

```
sat = satellite(sc,positionTT,velocityTT,"CoordinateFrame","ecef");
```

Visualize the trajectories of the satellites.

```
play(sc);
```


Satellite Scenario Viewer
Sat1
Sat2
Sat3
Sat4
50x
May 31 2022
15:37:51 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
May 31 2022 15:40:00.000
May 31 2022 15:50:00.000

Add Ground Stations to Scenario and Visualize Access Intervals

Create a satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = 10;
lon = -30;
gs = groundStation(sc,lat,lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

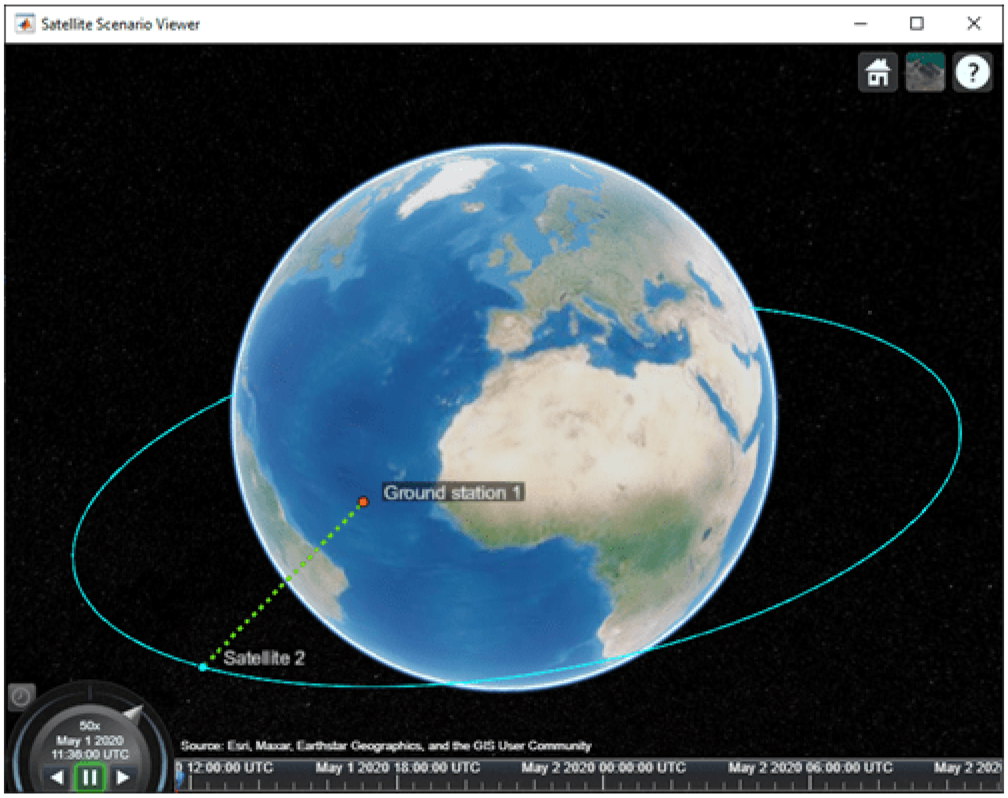
Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat,gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"           1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"           2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"           3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"           4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"           5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"           6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"           7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"           8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000;15000000];
eccentricity = [0.01;0.02];
inclination = [0;10];
rightAscensionOfAscendingNode = [0;15];
argumentOfPeriapsis = [0;30];
trueAnomaly = [0;20];

sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  1x2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

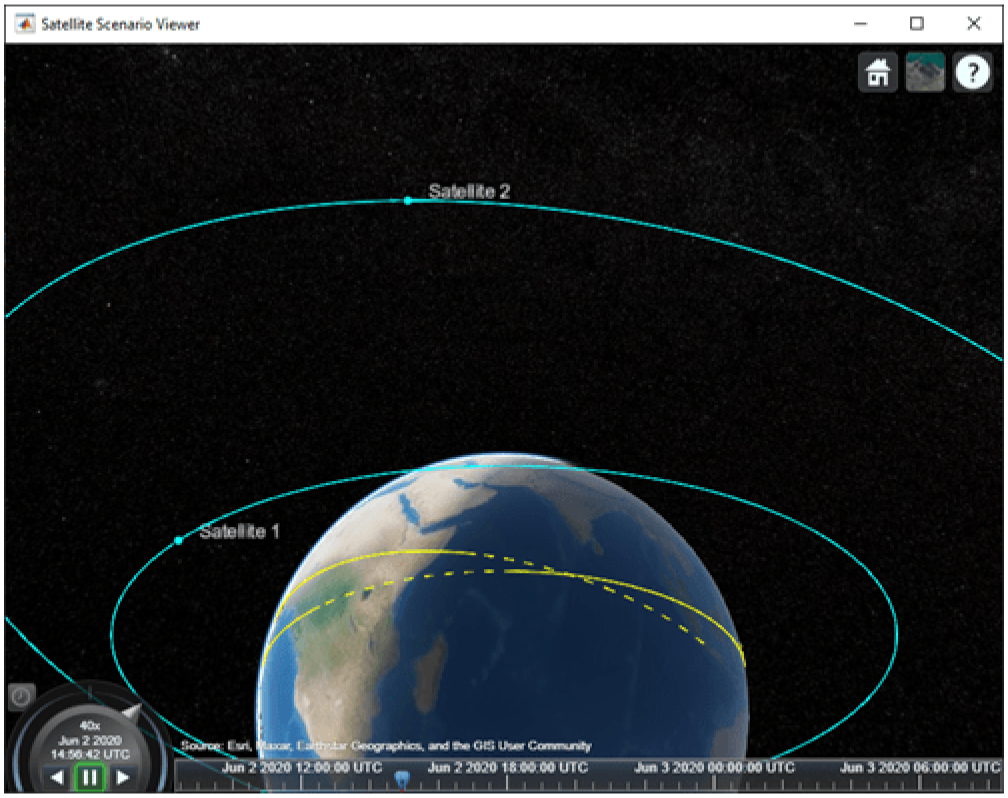
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,LeadTime=3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    LeadLineColor
    TrailLineColor
    VisibilityMode
```

Play the scenario and set the animation speed of the simulation to 40.

```
play(sc,PlaybackSpeedMultiplier=40)
```



Visualize GPS Constellation

Set up the satellite scenario.

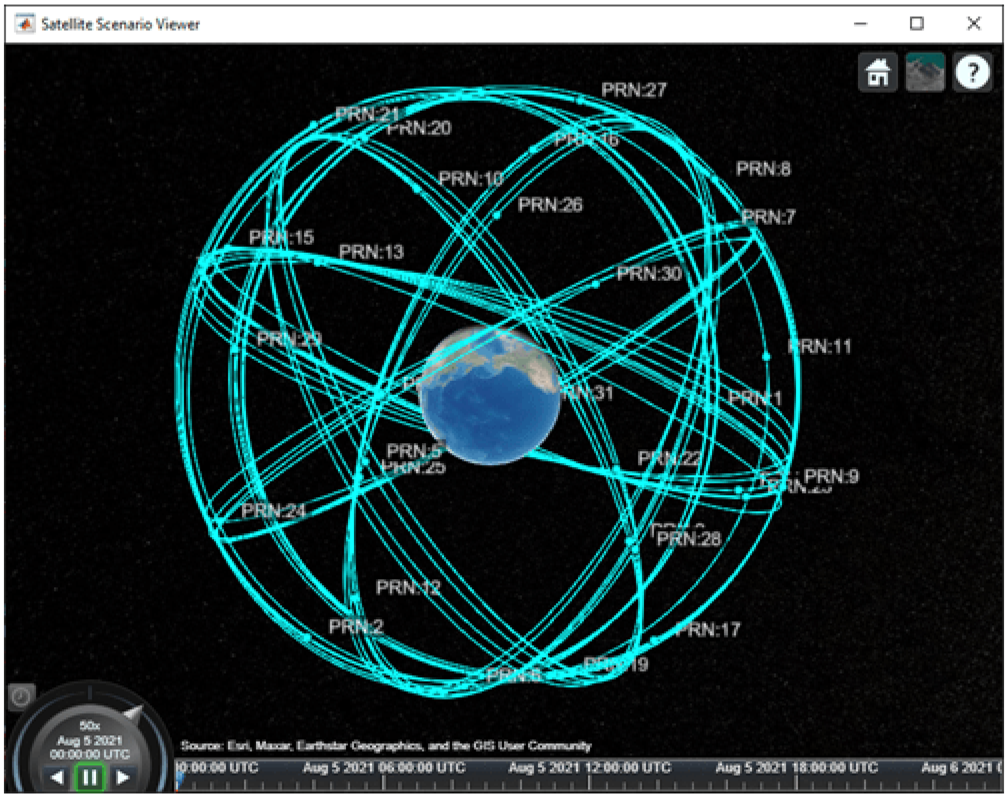
```
startTime = datetime(2021,8,5);
stopTime = startTime + days(1);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add satellites to the scenario from a SEM almanac file.

```
sat = satellite(sc,"gpsAlmanac.txt","OrbitPropagator","gps");
```

Visualize the GPS constellation.

```
v = satelliteScenarioViewer(sc);
```



Input Arguments

scenario — Satellite scenario
satelliteScenario object

Satellite scenario, specified as a `satelliteScenario` object.

file — TLE or SEM almanac file
character vector | string scalar

TLE or SEM almanac file, specified as a character vector or a string scalar. The file must exist in the current folder, in a folder on the MATLAB path, or it must include a full or relative path to a file.

For more information on TLE files, see “Two Line Element (TLE) Files”.

Data Types: `char` | `string`

RINEXdata — RINEX navigation message
structure

RINEX navigation message output from `rinexread`, specified as a structure. The navigation message must belong to GPS or Galileo constellation.

If `RINEXdata` contains multiple entries for the same satellite and `StartTime` of `scenario` is using default values, the entry corresponding to the earliest time is used. If `StartTime` is explicitly specified, the entry that is closest to `StartTime` is used.

Data Types: `struct`

Note This argument requires Navigation Toolbox™.

`semimajoraxis`, `eccentricity`, `inclination`, `RAAN`, `argofperiapsis`, `trueanomaly` — Keplerian elements defined in GCRF

comma-separated list of vectors

Keplerian elements defined in the GCRF, specified as a comma-separated list of vectors. The Keplerian elements are:

- `semimajoraxis` – This vector defines the semimajor axis of the orbit of the satellite. Each value is equal to half of the longest diameter of the orbit.
- `eccentricity` – This vector defines the shape of the orbit of the satellite.
- `inclination` – This vector defines the angle between the orbital plane and the *xy*-plane of the GCRF for each satellite in the range [0,180].
- `RAAN` (right ascension of ascending node) – This element defines the angle between the *xy*-plane of the GCRF and the direction of the ascending node, as seen from the Earth's center of mass for each satellite in the range [0,360). The ascending node is the location where the orbit crosses the *xy*-plane of the GCRF and goes above the plane.
- `argofperiapsis` (argument of periapsis) – This vector defines the angle between the direction of the ascending node and the periapsis, as seen from the Earth's center of mass in the range [0,360). Periapsis is the location on the orbit that is closest to the Earth's center of mass for each satellite.
- `trueanomaly` – This vector defines the angle between the direction of the periapsis and the current location of the satellite, as seen from the Earth's center of mass for each satellite in the range [0,360).

Note All angles defined outside the specified range is automatically converted to the corresponding value within the acceptable range.

For more information on Keplerian elements, see “Orbital Elements”.

`positiontable` — Position data

timetable | table

Position data in meters, specified as a timetable created using the `timetable` function or `table` function. The `positiontable` has exactly one monotonically increasing column of *rowTimes* (`datetime` or `duration`) and either:

- One or more columns of variables, where each column contains data for an individual satellite over time.
- One column of 2-D data, where the length of one dimension must equal 3 and the remaining dimension defines the number of satellites in the ephemeris.
- One column of 3-D data, where the length of one dimension must equal 3, one dimension is a singleton, and the remaining dimension defines the number of satellites in the ephemeris.

If *rowTimes* values are of type `duration`, time values are measured relative to the current scenario `StartTime` property. The timetable `VariableNames` property are used by default if no names are provided as an input. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value argument is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `positiontable`.

Data Types: `table` | `timetable`

velocitytable — Velocity data
timetable | table

Velocity data in meters/second, specified as a timetable created using the `timetable` function or the `table` function. The `velocitytable` has exactly one monotonically increasing column of *rowTimes* (`datetime` or `duration`), and either:

- One or more columns of variables, where each column contains data for an individual satellite over time.
- One column of 2-D data, where the length of one dimension must equal 3 and the remaining dimension defines the number of satellites in the ephemeris.
- One column of 3-D data, where the length of one dimension must equal 3, one dimension is a singleton, and the remaining dimension defines the number of satellites in the ephemeris.

If *rowTimes* values are of type `duration`, time values are measured relative to the current scenario `StartTime` property. The timetable `VariableNames` are used by default if no names are provided as an input. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value argument is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `velocitytable`.

Data Types: `table` | `timetable`

positiontimeseries — Position data
`timeseries` object | `tscollection` object

Position data in meters, specified as a `timeseries` object or a `tscollection` object.

- If the `Data` property of the `timeseries` or `tscollection` object has two dimensions, one dimension must equal 3, and the other dimension must align with the orientation of the time vector.
- If the `Data` property of the `timeseries` or `tscollection` has three dimensions, one dimension must equal 3, either the first or the last dimension must align with the orientation of the time vector, and the remaining dimension defines the number of satellites in the ephemeris.

  When `timeseries.TimeInfo.StartDate` is empty, time values are measured relative to the current scenario `StartTime` property. The timeseries `Name` property (if defined) is used by default if no names are provided as inputs. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value pair is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `positiontimeseries`.

Data Types: `timeseries` | `tscollection`

velocitytimeseries — Velocity data
`timeseries` object | `tscollection` object

Velocity data in meters/second, specified as a `timeseries` object or a `tscollection` object.

- If the `Data` property of the `timeseries` or `tscollection` object has two dimensions, one dimension must equal 3, and the other dimension must align with the orientation of the time vector.
- If the `Data` property of the `timeseries` or `tscollection` has three dimensions, one dimension must equal 3, either the first or the last dimension must align with the orientation of the time vector, and the remaining dimension defines the number of satellites in the ephemeris.

  When `timeseries.TimeInfo.StartDate` is empty, time values are measured relative to the current scenario `StartTime` property. The timeseries `Name` property (if defined) is used by default if no names are provided as inputs. Satellite states are assumed to be in the GCRF unless a `CoordinateFrame` name-value pair is provided. States are held constant in GCRF for scenario timesteps outside of the time range of `velocitytimeseries`.

Data Types: `timeseries` | `tscollection`

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.

Example: `Name = 'MySatellite'` sets the satellite name to `'MySatellite'`.

`CoordinateFrame` — Satellite state coordinate frame
`"inertial"` (default) | `"ecef"` | `"geographic"`

Satellite state coordinate frame, specified as the comma-separated pair consisting of `'CoordinateFrame'` and one of these values:

- `"inertial"` — For `timeseries` or `timetable` data, specifying this value accepts the position and velocity in the GCRF frame.
- `"ecef"` — For `timeseries` or `timetable` data, specifying this value accepts the position and velocity in the ECEF frame.
- `"geographic"` — For `timeseries` or `timetable` data, specifying this value accepts the position [*lat*, *lon*, *altitude*], where *lat* and *lon* are latitude and longitude in degrees, and *altitude* is the height above the World Geodetic System 84 (WGS 84) ellipsoid in meters.

  Velocity is in the local NED frame.

Dependencies

To enable this name value argument, ephemeris data inputs (`timetable` or `timeseries`).

Data Types: `string` | `char`

`GPSweekepoch` — GPS week epoch
date string

GPS week epoch, specified as a date string in "dd-Mmm-yyyy" or 'dd-Mmm-yyyy' format. The GPS week number specifies the reference date that the function uses when counting weeks defined in the SEM almanac file. If you do not specify `GPSweekepoch`, the function uses the date that coincides with the latest GPS week number rollover date before the start time.

This argument applies only if you use a SEM almanac file. If you specify `GPSweekepoch` and you are not using a SEM almanac file, the function ignores the argument value.

Data Types: `string` | `char`

`Viewer` — Satellite scenario viewer
vector of `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a scalar, vector, or array of `satelliteScenarioViewer` objects. If the `AutoSimulate` property of the scenario is `false`, adding a satellite to the scenario disables any previously available timeline and playback widgets.

`Name` — Satellite name
string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `satellite` function. After you call `satellite` function, this property is read-only.

Satellite name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one satellite is added, specify `Name` as a string scalar or a character vector.
- If multiple satellites are added, specify `Name` as a string scalar, character vector, string vector or a cell array of character vectors. All satellites added as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vector must equal the number of satellites being added. Each satellite is assigned the corresponding name from the vector or cell array.

The default value when satellite is added to the satellite scenario using

- Keplerian orbital elements, TLE file, timeseries, or timetable — "Satellite ID", where `ID` is assigned by the satellite scenario.
- SEM almanac file or RINEX GPS navigation data — "PRN:prnValue", where prnValue is an integer denoting the pseudorandom noise code of the satellite as specified in the SEM almanac file.
- RINEX Galileo navigation data — "GAL Sat IF: id", where "id" is the satellite ID of the Galileo satellite defined in the RINEX navigation data.

Data Types: `string`

`OrbitPropagator` — Name of orbit propagator
`"sgp4"` | `"sdp4"` | `"two-body-keplerian"` | `"ephemeris"` | `"gps"` | `"galileo"`

You can set this property on `satellite` object creation and then this property becomes read-only.

Name of the orbit propagator used for propagating the satellite position and velocity, specified as one of these options.

- If you specify the satellite using timetable, table, `timeseries`, or `tscollection`, the `OrbitPropagator` value is `"ephemeris"`.
- If you specify the satellite using a SEM almanac file or RINEX data containing a GPS navigation message, the `OrbitPropagator` value can take one of these options.
  - `"gps"` (default)
  - `"sgp4"`
  - `"sdp4"`

  - `"two-body-keplerian"`
- If you specify the satellite using the RINEX data containing a Galileo navigation message, the `OrbitPropagator` value can take one of these options.
  - `"galileo"` (default)
  - `"sgp4"`
  - `"sdp4"`
  - `"two-body-keplerian"`
- If you specify the satellite using a TLE file, the `OrbitPropagator` value can take one of these options.
  - `"two-body-keplerian"`
  - `"sgp4"`
  - `"sdp4"`

  If the orbital period is less than 225 minutes, the default `OrbitPropagator` value is `"sgp4"`. Otherwise, the default `OrbitPropagator` value is `"sdp4"`.
- If you specify the satellite using `Keplerian` elements, the `OrbitPropagator` value can take one of these options.
  - `"two-body-keplerian"`
  - `"sgp4"`
  - `"sdp4"`

If the RINEX data contains both valid GPS and Galileo navigation messages, you cannot specify `OrbitPropagator` as `"gps"` or `"galileo"` using a name-value argument. However, you can still specify it as `"two-body-keplerian"`, `"sgp4"`, or `"sdp4"`.

Output Arguments

sat — Satellite in the scenario

`Satellite` object

Satellite in the scenario, returned as a `Satellite` object belonging to the satellite scenario specified by `scenario`.

You can modify the `Satellite` object by changing its property values.

Version History

Introduced in R2021a

R2022b: RINEXdata argument added to the function

You can now add satellites via RINEX navigation data to the scenario using the `RINEXdata` input argument.

R2022a: SEM almanac file added to the file argument

You can now add satellites via SEM almanac file to the scenario using the `file` input argument.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`access` | `receiver` | `transmitter` | `show` | `play` | `hide` | `orbitalElements`

Topics
“Comparison of Orbit Propagators”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

conicalSensor

Package: `matlabshared.satellitescenario`

Add conical sensor to satellite scenario

Syntax

```
conicalSensor(parent)
conicalSensor(parent,Name=Value)
sensor = conicalSensor( ___ )
```

Description

`conicalSensor(parent)` adds a `ConicalSensor` object to each parent in the vector `parent` using default parameters. `parent` can be `Satellite` , `GroundStation` , or `Gimbal`.

`conicalSensor(parent,Name=Value)` adds conical sensors to the parents in `parent` using additional parameters specified by optional name-value arguments. For example, `'MaxViewAngle'=90` specifies a field of view angle of 90 degrees.

`sensor = conicalSensor( ___ )` returns added conical sensors as a row vector `sensor`. Specify any input argument combination from previous syntaxes.

Examples

Calculate Maximum Revisit Time of Satellite

Create a satellite scenario with a start time of 15-June-2021 8:55:00 AM UTC and a stop time of five days later. Set the simulation sample time to `60` seconds.

```
startTime = datetime(2021,6,21,8,55,0);
stopTime = startTime + days(5);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 21-Jun-2021 08:55:00
          StopTime: 26-Jun-2021 08:55:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario using Keplerian orbital elements.

```
semiMajorAxis = 7878137;                                                              % me
eccentricity = 0;
inclination = 50;                                                                     % deg
rightAscensionOfAscendingNode = 0;                                                    % deg
argumentOfPeriapsis = 0;                                                              % deg
trueAnomaly = 50;                                                                     % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  Satellite with properties:

               Name:  Satellite 1
                 ID:  1
     ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
            Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
       Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
          Receivers:  [1x0 satcom.satellitescenario.Receiver]
           Accesses:  [1x0 matlabshared.satellitescenario.Access]
        GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
              Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator:  sgp4
        MarkerColor:  [0.059 1 1]
         MarkerSize:  6
          ShowLabel:  true
     LabelFontColor:  [1 1 1]
      LabelFontSize:  15
```

Add a ground station, which represents the location to be photographed, to the scenario.

```
gs = groundStation(sc,Name="Location to Photograph", ...
    Latitude=42.3001,Longitude=-71.3504)                 % degrees
```

```
gs = 
  GroundStation with properties:

                 Name:  Location to Photograph
                   ID:  2
             Latitude:  42.3 degrees
            Longitude:  -71.35 degrees
             Altitude:  0 meters
    MinElevationAngle:  0 degrees
       ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
              Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
         Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
            Receivers:  [1x0 satcom.satellitescenario.Receiver]
             Accesses:  [1x0 matlabshared.satellitescenario.Access]
          MarkerColor:  [1 0.4118 0.1608]
           MarkerSize:  6
            ShowLabel:  true
       LabelFontColor:  [1 1 1]
        LabelFontSize:  15
```

Add a gimbal to the satellite. You can steer this gimbal independently of the satellite.

```
g = gimbal(sat)
```

```
g = 
  Gimbal with properties:

               Name:  Gimbal 3
                 ID:  3
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
      ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
        Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
           Receivers:  [1x0 satcom.satellitescenario.Receiver]
```

Track the location to be photographed using the gimbal.

```
pointAt(g,gs);
```

Add a conical sensor to the gimbal. This sensor represents the camera. Set the field of view to 60 degrees.

```
camSensor = conicalSensor(g,MaxViewAngle=60)
```

```
camSensor = 
  ConicalSensor with properties:

               Name:  Conical sensor 4
                 ID:  4
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
        MaxViewAngle:  60 degrees
            Accesses:  [1x0 matlabshared.satellitescenario.Access]
         FieldOfView:  [0x0 matlabshared.satellitescenario.FieldOfView]
```

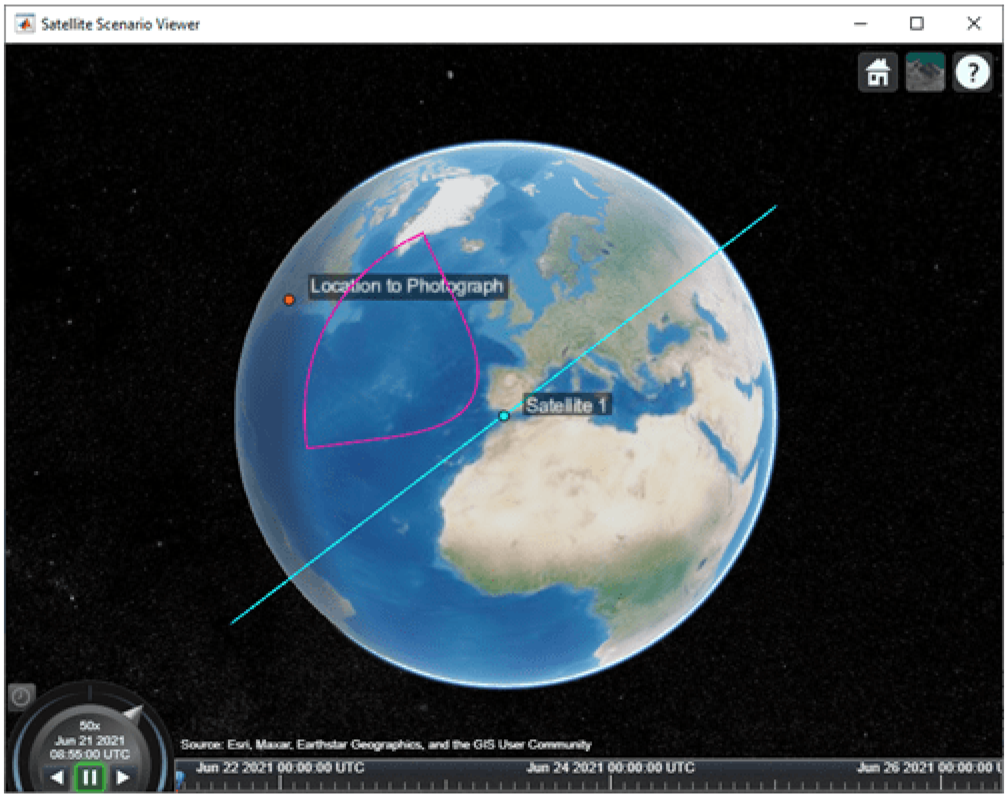
Add access analysis to the conical sensor between the camera and the location to be photographed.

```
ac = access(camSensor,gs)
```

```
ac = 
  Access with properties:

    Sequence:  [4 2]
    LineWidth:  3
    LineColor:  [0.3922 0.8314 0.0745]
```

Visualize the field of view of the camera by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
fieldOfView(camSensor);
```



Determine the intervals during which the camera can see the geographical site.

```
t = accessIntervals(ac)
```

```
t=35×8 table
         Source                   Target              IntervalNumber         StartTime
    ___________________    _______________________    ______________    ____________________

    "Conical sensor 4"     "Location to Photograph"          1          21-Jun-2021 10:38:00
    "Conical sensor 4"     "Location to Photograph"          2          21-Jun-2021 12:36:00
    "Conical sensor 4"     "Location to Photograph"          3          21-Jun-2021 14:37:00
    "Conical sensor 4"     "Location to Photograph"          4          21-Jun-2021 16:41:00
    "Conical sensor 4"     "Location to Photograph"          5          21-Jun-2021 18:44:00
    "Conical sensor 4"     "Location to Photograph"          6          21-Jun-2021 20:46:00
    "Conical sensor 4"     "Location to Photograph"          7          21-Jun-2021 22:50:00
    "Conical sensor 4"     "Location to Photograph"          8          22-Jun-2021 09:51:00
    "Conical sensor 4"     "Location to Photograph"          9          22-Jun-2021 11:46:00
    "Conical sensor 4"     "Location to Photograph"         10          22-Jun-2021 13:46:00
    "Conical sensor 4"     "Location to Photograph"         11          22-Jun-2021 15:50:00
    "Conical sensor 4"     "Location to Photograph"         12          22-Jun-2021 17:53:00
    "Conical sensor 4"     "Location to Photograph"         13          22-Jun-2021 19:55:00
    "Conical sensor 4"     "Location to Photograph"         14          22-Jun-2021 21:58:00
    "Conical sensor 4"     "Location to Photograph"         15          23-Jun-2021 10:56:00
    "Conical sensor 4"     "Location to Photograph"         16          23-Jun-2021 12:56:00
      ⋮
```

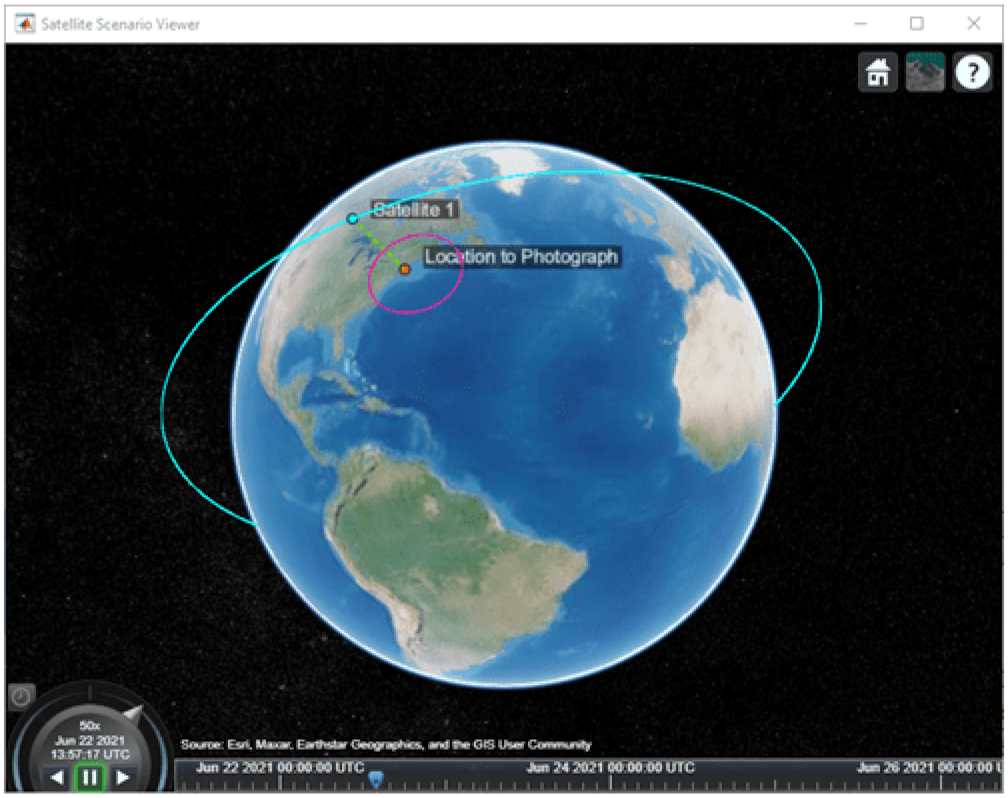
Calculate the maximum revisit time in hours.

```
startTimes = t.StartTime;
endTimes = t.EndTime;
revisitTimes = hours(startTimes(2:end) - endTimes(1:end-1));
maxRevisitTime = max(revisitTimes)                               % hours
```

```
maxRevisitTime = 12.6667
```

Visualize the revisit times that the camera photographs of the location.

```
play(sc);
```



Input Arguments

parent — Element of scenario to which conical sensor is added
scalar | vector

Element of scenario in which to add the conical sensor, specified as a scalar or vector of satellites, ground stations, or gimbals. The number of conical sensors to add is determined by the size of the inputs.

- If `parent` is a scalar, all conical sensors are added to the parent.
- If `parent` is a vector of parents and the number of conical sensors specified is one, that conical sensor is added to each parent.
- If `parent` is a vector of parents and the number of conical sensors specified is more than one, the number of conical sensors must equal the number of parents in `parent` and each parent gets one conical sensor.

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: `'MountingAngle'=[20;35;10]` sets the yaw, pitch, and roll angles of the conical sensor to 20, 35, and 10 degrees, respectively.

Note The size of the name-value arguments defines the number of conical sensors that you can specify. To understand how to specify multiple conical sensors, refer to these properties.

Name — Conical sensor name

`"Conical sensor *idx*"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `conicalSensor` function. After you call the `conicalSensor` function, this property is read-only.

Conical sensor name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one conical sensor, specify `Name` as a string scalar or a character vector.
- If you are adding multiple conical sensors, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All conical sensors that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of conical sensors that you are adding. Each conical sensor is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent

`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one conical sensor, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of conical sensor.
- If you are adding multiple conical sensors, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified conical sensors. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the conical sensors. The columns correspond to the mounting location of each specified conical sensor and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one conical sensor, `MountingAngles` is a three-element vector.
- If you are adding multiple conical sensors, `MountingAngles` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified conical sensors. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the conical sensors. The columns correspond to the mounting angles of each specified conical sensor and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

`MaxViewAngle` — Field of view angle
`30` (default) | scalar in the range [0, 180] | vector

Field of view angle in degrees, specified as a scalar in the range [0, 180] or a vector.

- If you add one conical sensor, `MaxViewAngle` must be a scalar.
- If you add multiple conical sensors, `MaxViewAngle` can be a scalar or a vector. When `MaxViewAngle` is a scalar, the same field of view angle is assigned to all conical sensors that you are adding. When `MaxViewAngle` is a vector, the length of `MaxViewAngle` must equal the number of conical sensors in the parent. Each element of `MaxViewAngle` is assigned to the specified corresponding conical sensor.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify `MaxViewAngle` while the `SimulationStatus` is `NotStarted` or `InProgress`.

Data Types: `double`

Output Arguments

`sensor` — Conical sensor
`row vector` object

Conical sensors attached to `parent`, returned as a row vector.

When the `AutoSimulate` property of the satellite scenario is `false`, you can call `conicalSensor` function only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Version History
Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `groundStation` | `access` | `gimbal` | `satellite`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

play

Package: `matlabshared.satellitescenario`

Play satellite scenario simulation results on viewer

Syntax

```
play(scenario)
play(viewer)
play(scenario,Name=Value)
```

Description

`play(scenario)` plays simulation results of the satellite scenario, `scenario`, on the Satellite Scenario Viewer. When the `AutoSimulate` property of the satellite scenario is `true`, the simulation is automatically performed from `StartTime` to `StopTime` using a step size specified by `SampleTime`, and the results are played on the viewer. Otherwise, the results calculated for `SimulationTime` are played on the viewer. Calling the `play` function enables the widgets on the viewer.

`play(viewer)` plays the satellite scenario simulation results on the Satellite Scenario Viewer specified by `viewer`.

`play(scenario,Name=Value)` specifies additional options using one or more name-value arguments.

Examples

Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000;15000000];
eccentricity = [0.01;0.02];
inclination = [0;10];
rightAscensionOfAscendingNode = [0;15];
argumentOfPeriapsis = [0;30];
trueAnomaly = [0;20];

sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly)
```

```
sat =
  1x2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

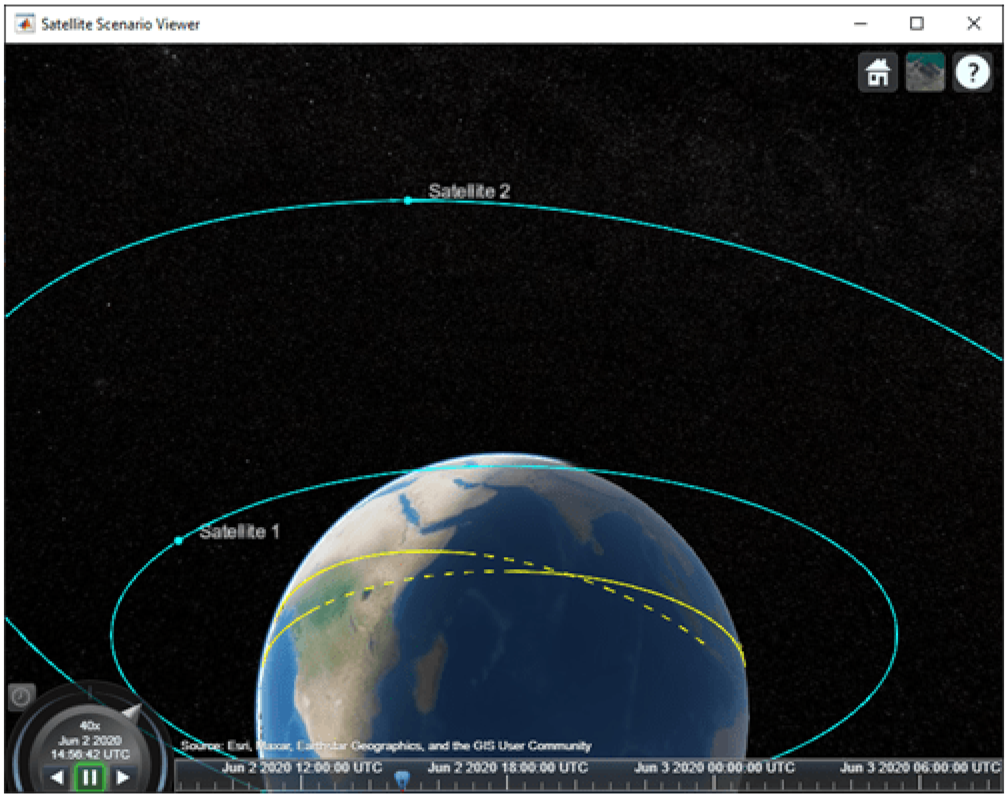
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,LeadTime=3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    LeadLineColor
    TrailLineColor
    VisibilityMode
```

Play the scenario and set the animation speed of the simulation to 40.

```
play(sc,PlaybackSpeedMultiplier=40)
```



Input Arguments

`scenario` — Satellite scenario
`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object.

`viewer` — Viewer playing simulation results
scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Viewer playing the simulation results, specified as a scalar `satelliteScenarioViewer` object or an array of `satelliteScenarioViewer` objects.

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.

Example: `PlaybackSpeedMultiplier=30` plays the animation 30 times faster than real time.

`Viewer` — Satellite Scenario Viewer
all viewers associated with `satelliteScenarioViewer` (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object or an array of `satelliteScenarioViewer` objects.

`PlaybackSpeedMultiplier` — Speed of animation
`50` (default) | positive scalar

Speed of animation in the viewer relative to real time, specified as a positive scalar.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`hide` | `show` | `satellite` | `access` | `groundStation` | `restart`

Topics
“Satellite Constellation Access to Ground Station”
“Comparison of Orbit Propagators”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

pointAt

Package: `matlabshared.satellitescenario`

Point satellite at target

Syntax

```
pointAt(sat,coordinates)
pointAt(sat,target)
pointAt(sat,'nadir')
pointAt(sat,attitudetable)
pointAt(sat,attitudetable,Name=Value)
pointAt(sat,attitudetimeseries)
pointAt(sat,attitudetimeseries,Name=Value)

pointAt(gimbal,'none')
pointAt(gimbal,coordinates)
pointAt(gimbal,target)
pointAt(gimbal,'nadir')
pointAt(gimbal,steeringtable)
pointAt(gimbal,steeringtimeseries)
```

Description

Satellite Object

`pointAt(sat,coordinates)` steers the satellites in the vector `sat` toward the geographical coordinates [latitude; longitude; altitude] specified by `coordinates`.

`pointAt(sat,target)` steers the satellites specified by `sat` toward the specified `target`. The input `target` can be another satellite or ground station.

`pointAt(sat,'nadir')` steers the satellites specified by the row vector `sat` toward the nadir direction. `'nadir'` is the default pointing direction.

`pointAt(sat,attitudetable)` sets the attitude of the satellite `sat` such that it follows the attitudes provided in `attitudetable`, which is a MATLAB `timetable` object.

`pointAt(sat,attitudetable,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous `attitudetable` syntax. For example, to interpret the provided attitude values as the rotation from the Geocentric Celestial Reference Frame (GCRF) to the body frame, set `CoordinateFrame` to `inertial`.

`pointAt(sat,attitudetimeseries)` sets the attitudes of the satellite `sat` such that it follows the attitude provided in `attitudetimeseries`, which is a MATLAB `timeseries` object.

`pointAt(sat,attitudetimeseries,Name=Value)` specifies options using one or more name-value arguments in addition to the input arguments in the previous `attitudetimeseries` syntax. For example, to interpret the provided attitude values as the rotation from the GCRF to the body frame, set `CoordinateFrame` to `inertial`.

Gimbal Object

`pointAt(gimbal,'none')` sets the gimbal angles (gimbal azimuth and gimbal elevation) of the gimbals in the vector `gimbal` to zero. This setting is the default.

`pointAt(gimbal,coordinates)` steers the gimbals in the vector `gimbal` toward the geographical coordinates [latitude; longitude; altitude] specified by `coordinates`.

`pointAt(gimbal,target)` steers the gimbals in the vector `gimbal` toward the specified `target`.

`pointAt(gimbal,'nadir')` steers the gimbals specified by the row vector `gimbal` toward the nadir direction of their parents, namely, their parent's latitude, longitude, and 0 meter altitude.

`pointAt(gimbal,steeringtable)` sets the orientation of the gimbals to align with the azimuth and elevation angles provided in `steeringtable`, which is a MATLAB `timetable` object.

`pointAt(gimbal,steeringtimeseries)` sets the orientation of the gimbals to align with the azimuth and elevation angles provided in `steeringtimeseries`, which is MATLAB `timeseries` object.

Examples

Steer Ground Station Gimbal to Point at Satellite

Create a satellite scenario object.

```
startTime = datetime(2021,6,10);                     % 10 June 2021, 12:00 AM UTC
stopTime = datetime(2021,6,11);                      % 11 June 2021, 12:00 AM UTC
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                            % meters
eccentricity = 0;
inclination = 10;                                    % degrees
rightAscensionOfAscendingNode = 0;                   % degrees
argumentOfPeriapsis = 0;                             % degrees
trueAnomaly = 0;                                     % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a ground station to the scenario.

```
latitude = 42.3501;                        % degrees
longitude = -71.3504;                      % degrees
gs = groundStation(sc,latitude,longitude);
```

Add a gimbal to the ground station.

```
g = gimbal(gs,MountingLocation=[0; 0; -1],MountingAngles=[0; 180; 0]);
```

Add a conical sensor to the gimbal.

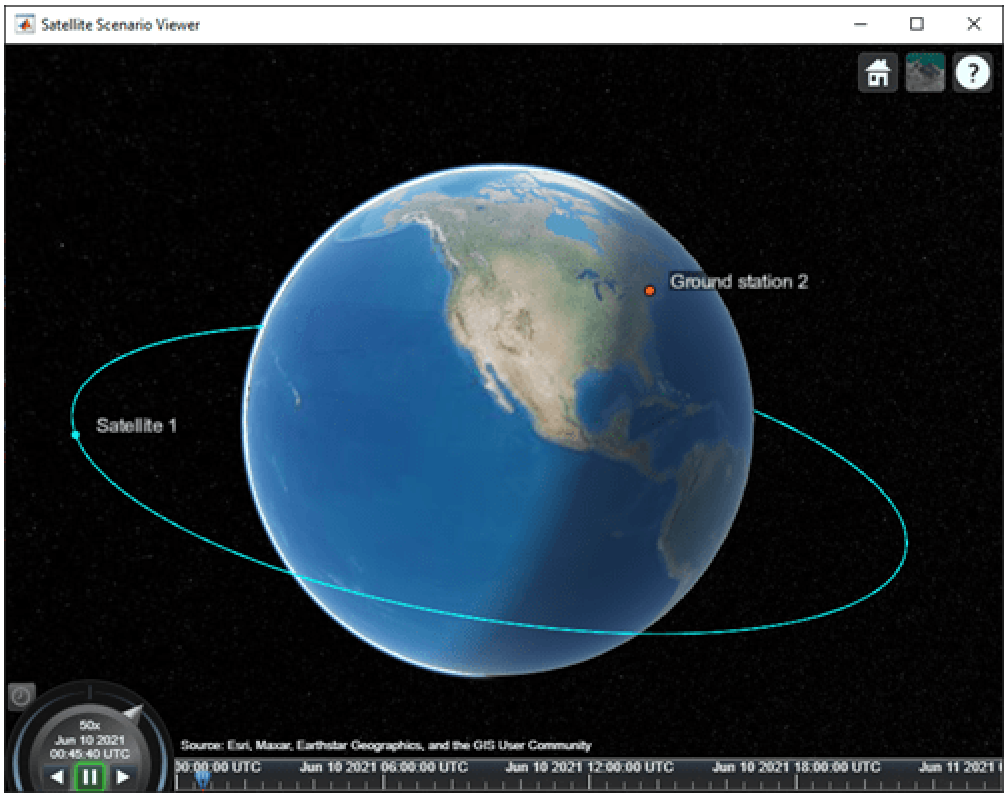
```
c = conicalSensor(g,MountingLocation=[0; 0; 0.5]);
```

Point the gimbal at the satellite.

```
pointAt(g,sat);
```

Visualize the scenario by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```

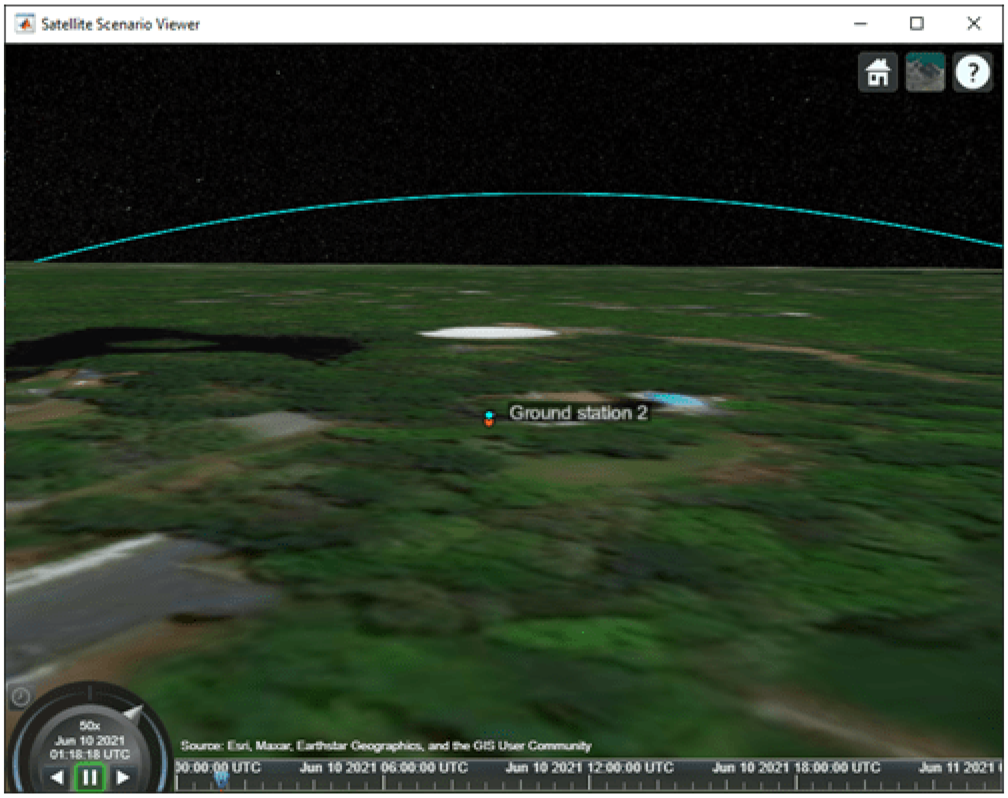


Play the scenario.

```
play(sc);
```

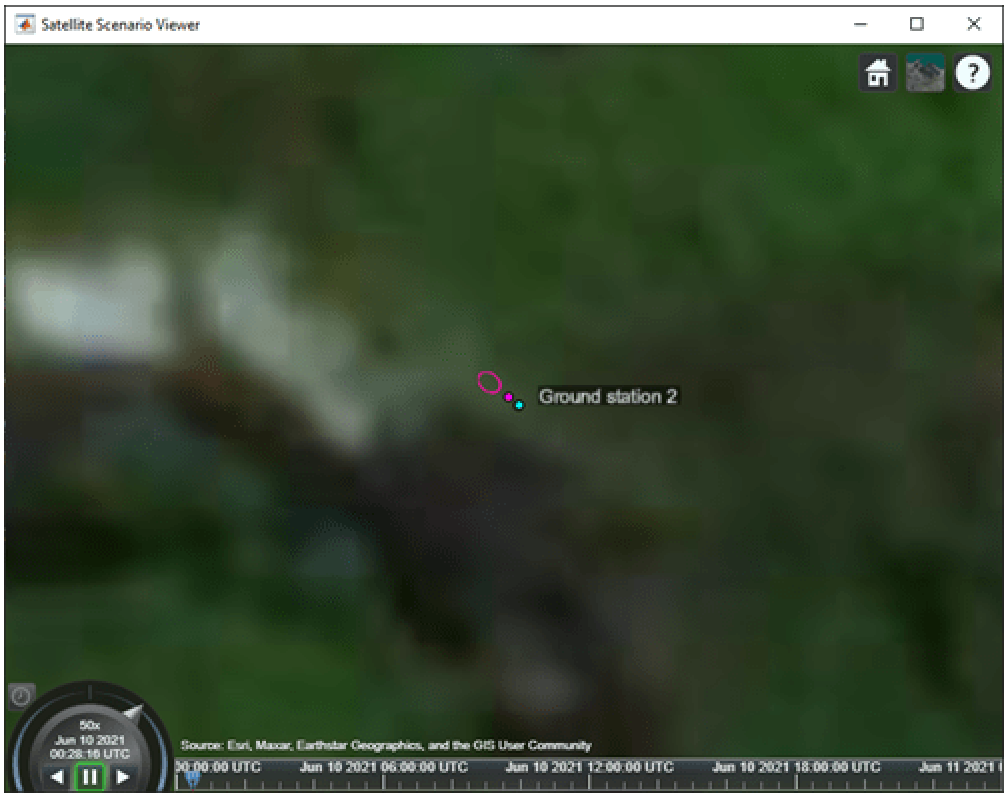
Set the ground station as the camera target.

```
camtarget(v,gs);
```



Visualize the field of view of the conical sensor and observe the change in orientation of the conical sensor.

```
fieldOfView(c);
```



Input Arguments

`sat` — Satellite
scalar | vector

Satellite object, specified as either a scalar or a vector.

`gimbal` — Gimbal
scalar | vector

Gimbal object, specified as either a scalar or a vector.

`coordinates` — Geographical coordinates of satellite or gimbal target
three-element vector | 2-D array

Geographical coordinates of the satellite or gimbal target, specified as a three-element vector or a 2-D array.

- When the coordinates are a three-element vector, the elements of the vector correspond to the latitude, longitude, and altitude, in that order, and all satellites or gimbals are steered to point at this location.
- When the coordinates are a 2-D array, the number of rows must equal 3 and the number of columns must equal the number of satellites in `satellite` or the number of gimbals in `gimbal`. The rows correspond to the latitude, longitude, and altitude, in that order, and each column represents the pointing coordinates of the corresponding satellite in the vector `satellite` or

gimbal in the vector `gimbal`. The latitudes and longitudes are specified in degrees and the altitudes are specified in meters, which represent the height above the surface of the Earth.

`target` — Target
scalar | vector

Target at which input `satellite` or `gimbal` is pointed, specified as a scalar or a vector. The input `target` can be another satellite or a ground station.

- When `target` is a scalar, all satellites or gimbals point to the specified `target`.
- When `target` is a vector, the length of `target` must equal the number of satellites in `sat` or the number of gimbals in `gimbal`. Each element in `target` represents the pointing target of a satellite in `sat` or a gimbal in `gimbal`.

`attitudetable` — MATLAB `timetable` for satellite attitude
`timetable` object

MATLAB `timetable` with exactly one monotonically increasing column of `rowTimes` (datetime or duration).

- If `sat` contains a single satellite, the table must contain one data column of scalar-first quaternions [1-by-4] or ZYX Euler angles [1-by-3].
- If `sat` is an array of satellites, each data row must contain either:
  - Multiple columns, where each column contains data for an individual satellite over time.
  - One column of 2-D data, where the length of one dimension must equal 3 or 4, depending on whether Euler angles or quaternions are used, and the remaining dimension must have length equal to the number of satellites in `sat`.
  - One column of 3-D data, where the length of one dimension must equal 3 or 4, depending on whether Euler angles or quaternions are used, one dimension is a singleton, and the remaining dimension must have length equal to the number of satellites in `sat`.

Euler angles represent passive, intrinsic rotations in degrees, using the ZYX rotation order. If the provided `rowTimes` are of type `duration`, time values are measured relative to the current scenario `StartTime` property.

The function assumes that satellite attitudes represent the transformation from the GCRF to the body frame, unless you specify a `CoordinateFrame` name-value argument. For scenario timesteps outside of the time range of `attitudetable`, the function uses `nadir` by default, unless you specify an `ExtrapolationMethod` name-value argument.

`attitudetimeseries` — MATLAB `timeseries` for satellite attitude
`timeseries` object

MATLAB `timeseries` containing scalar-first quaternions or ZYX Euler angles.

- If the `Data` property of `timeseries` has two dimensions, the length of one dimension must equal 3 or 4, depending on whether Euler angles or quaternions are used, and the other dimension must align with the orientation of the time vector.
- If `sat` is an array of satellites, the `Data` property of `timeseries` must have three dimensions where the length of one dimension must equal 3 or 4, depending on whether Euler angles or quaternions are used, either the first or the last dimension must align with the orientation of the time vector, and the remaining dimension must align with the number of satellites in `sat`.

Euler angles represent passive, intrinsic rotations in degrees, using the ZYX rotation order. When `timeseries.TimeInfo.StartDate` is empty, time values are measured relative to the current scenario `StartTime` property.

The function assumes that satellite attitudes represent the transformation from the Geocentric Celestial Reference Frame (GCRF) to the body frame, unless you specify a `CoordinateFrame` name-value argument. For scenario timesteps outside of the time range of `attitudetable`, the function uses `nadir` by default, unless you specify an `ExtrapolationMethod` name-value argument.

`steeringtable` — MATLAB `timetable` for satellite steering
`timetable` object

MATLAB `timetable` with exactly one monotonically increasing column of `rowTimes` (datetime or duration).

- If `gimbal` contains a single gimbal, the table must contain one data column of azimuth and elevation angles in degrees [1-by-2].
- If `gimbal` is an array of gimbals, each data row must contain either:
  - Multiple columns, where each column contains data for an individual gimbal over time.
  - One column of 2-D data, where the length of one dimension must equal 2 and the remaining dimension must have length equal to the number of gimbals in `gimbal`.
  - One column of 3-D data, where the length of one dimension must equal 2, one dimension is a singleton, and the remaining dimension must have length equal to the number of gimbals in `gimbal`.

Specify the azimuth and elevation angles in degrees. If the provided `rowTimes` are of type `duration`, time values are measured relative to the current scenario `StartTime` property.

`steeringtimeseries` — MATLAB `timeseries`for satellite steering
`timeseries` object

MATLAB `timeseries` timeseries containing azimuth and elevation in degrees [1-by-2].

- If the `Data` property of `timeseries` has two dimensions, the length of one dimension must equal 2 and the other dimension must align with the orientation of the time vector.
- If `gimbal` is an array of gimbals, the `Data` property of `timeseries` must have three dimensions where:
  - The length of one dimension must equal 2.
  - Either the first or the last dimension must align with the orientation of the time vector.
  - The remaining dimension must align with the number of gimbals in `gimbal`.

When `timeseries.TimeInfo.StartDate` is empty, time values are measured relative to the current scenario `StartTime` property.

Name-Value argument 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: `pointAt(sat,attTT,CoordinateFrame="inertial")` interprets the provided attitude values as the rotation from the Geocentric Celestial Reference Frame (GCRF) to the body frame.

`CoordinateFrame` — Coordinate frame of custom attitude inputs
`inertial` (default) | `ecef` | `ned`

Coordinate frame of custom attitude inputs, specified as one of these options.

- `inertial` — Interprets the provided attitude values as the rotation from the GCRF to the body frame.
- `ecef` — Interprets the provided attitude values as the rotation from the Earth-Centered-Earth-Fixed (ECEF) frame to the body frame.
- `ned` — Interprets the provided attitude values as the rotation from the North-East-Down (NED) frame to the body frame.

Data Types: `char` | `string`

`ExtrapolationMethod` — Default behavior for attitude
`nadir` (default) | `fixed`

Default behavior for attitude, specified as:

- `nadir` — Sets the attitude of the satellite `sat` such that the yaw axis points in the nadir direction.
- `fixed` — Keeps the attitude constant with respect to the GCRF at the closest time value for which data is provided in the custom attitude data.

The scenario uses this setting for scenario time steps that lie outside the provided custom attitude time range. If you do not provide `ExtrapolationMethod`, the function returns a warning when the scenario time is out of range of the custom attitude time range.

Data Types: `char` | `string`

`Format` — Format of attitude data provided
`quaternion` (default) | `euler`

Format of attitude data provided, specified as one of these options.

- `quaternion` — Interprets the provided attitude values as scalar-first quaternions. Quaternions represent passive rotations from `CoordinateFrame` to the body frame.
- `euler` — Interprets the provided attitude values as Euler angles, in degrees. Euler angles represent passive, intrinsic rotations from `CoordinateFrame` to the body frame using the ZYX rotation order and are provided in that order.

Data Types: `char` | `string`

Note When the `AutoSimulate` property of the satellite scenario is `false`, you can call the `pointAt` function as long as the `SimulationStatus` is `NotStarted` or `InProgress`.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `access` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

camroll

Package: `matlabshared.satellitescenario`

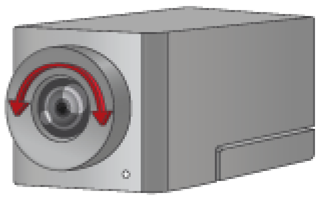
Set or get roll angle of camera for Satellite Scenario Viewer

Syntax

```
camroll(viewer,roll)
outRoll = camroll(viewer,___)
```

Description

`camroll(viewer,roll)` sets the roll angle of the camera for the Satellite Scenario Viewer. Setting the roll angle rotates the camera around its *x*-axis.



`outRoll = camroll(viewer,___)` returns the roll angle of the camera. If the second input is `roll`, then the function sets the output equal to the input roll.

Examples

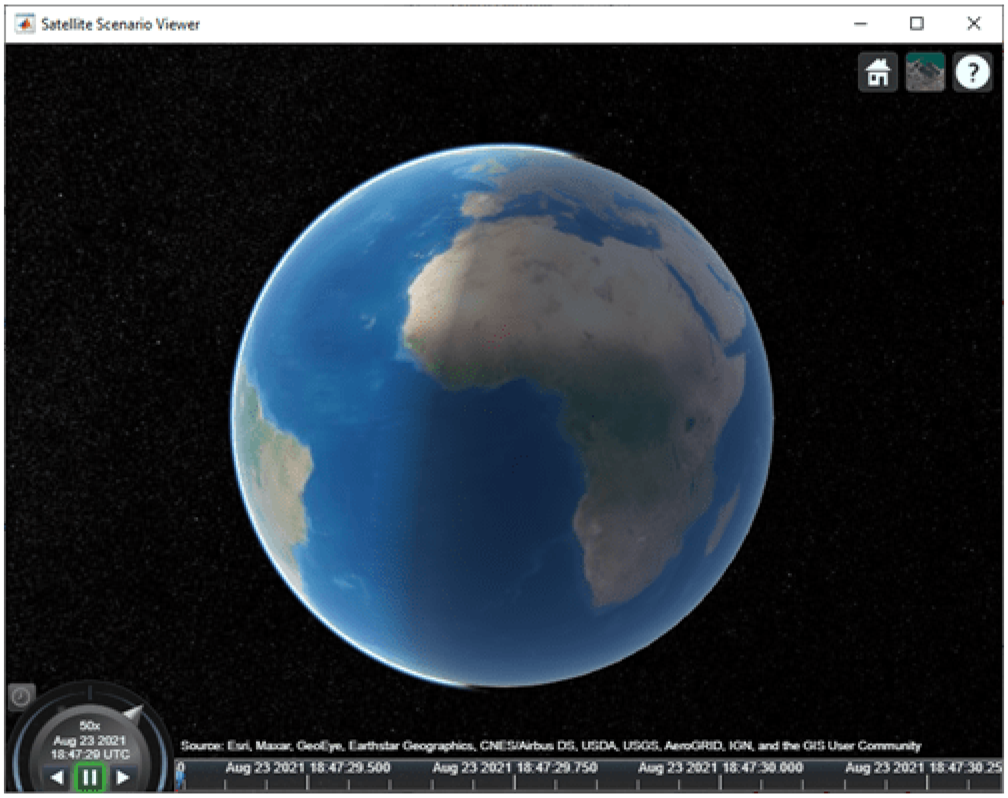
Set Camera Roll Angle of Satellite Scenario Viewer

Create a satellite scenario object.

```
sc = satelliteScenario;
```

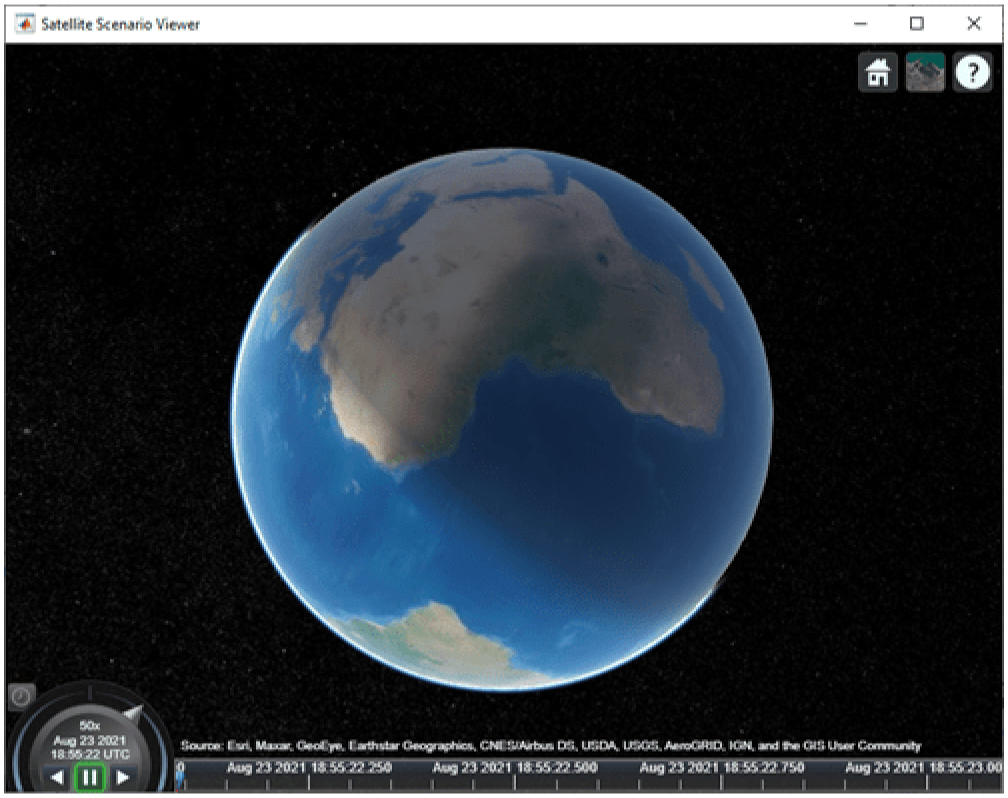
Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```


Satellite Scenario Viewer
50x
Aug 23 2021
18:47:29 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
Aug 23 2021 18:47:29.500
Aug 23 2021 18:47:29.750
Aug 23 2021 18:47:30.000
Aug 23 2021 18:47:30.25

Set the roll angle of the camera in the Satellite Scenario Viewer to 60 degrees.

```
roll = 60;      % degrees
camroll(v,roll);
```


Satellite Scenario Viewer
50x
Aug 23 2021
18:55:22 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
Aug 23 2021 18:55:22.250
Aug 23 2021 18:55:22.500
Aug 23 2021 18:55:22.750
Aug 23 2021 18:55:23.00

Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[1]

`roll` — Roll angle of camera
numeric scalar in the range [–360, 360]

Roll angle of the camera in degrees, specified as a scalar in the range [–360, 360].

Output Arguments

`outRoll` — Camera roll angle
numeric scalar in the range [-360, 360]

Camera roll angle in degrees, returned as a numeric scalar.

Tips

- When the pitch angle is near –90 (the default value) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle might

1 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks®.

change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `campitch` | `campos` | `hideAll` | `camtarget` | `camheight` | `camheading`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

campitch

Package: `matlabshared.satellitescenario`

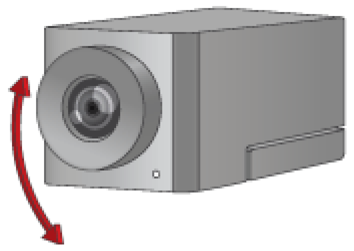
Set or get pitch angle of camera for Satellite Scenario Viewer

Syntax

```
campitch(viewer,pitch)
outPitch = campitch(viewer,___)
```

Description

`campitch(viewer,pitch)` sets the pitch angle of the camera for the specified Satellite Scenario Viewer. Setting the pitch angle tilts the camera up or down about its *y*-axis.



`outPitch = campitch(viewer,___)` returns the pitch angle of the camera. If the second input is `pitch`, then the function sets the output equal to the input pitch.

Examples

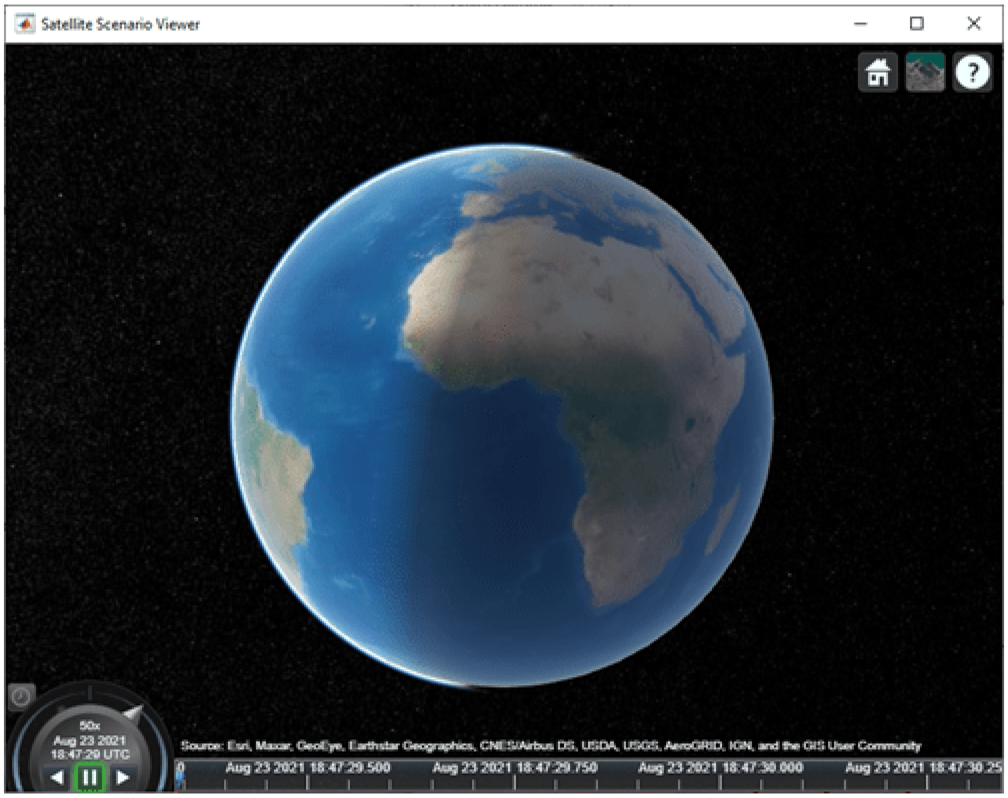
Set Camera Pitch Angle of Satellite Scenario Viewer

Create a satellite scenario object.

```
sc = satelliteScenario;
```

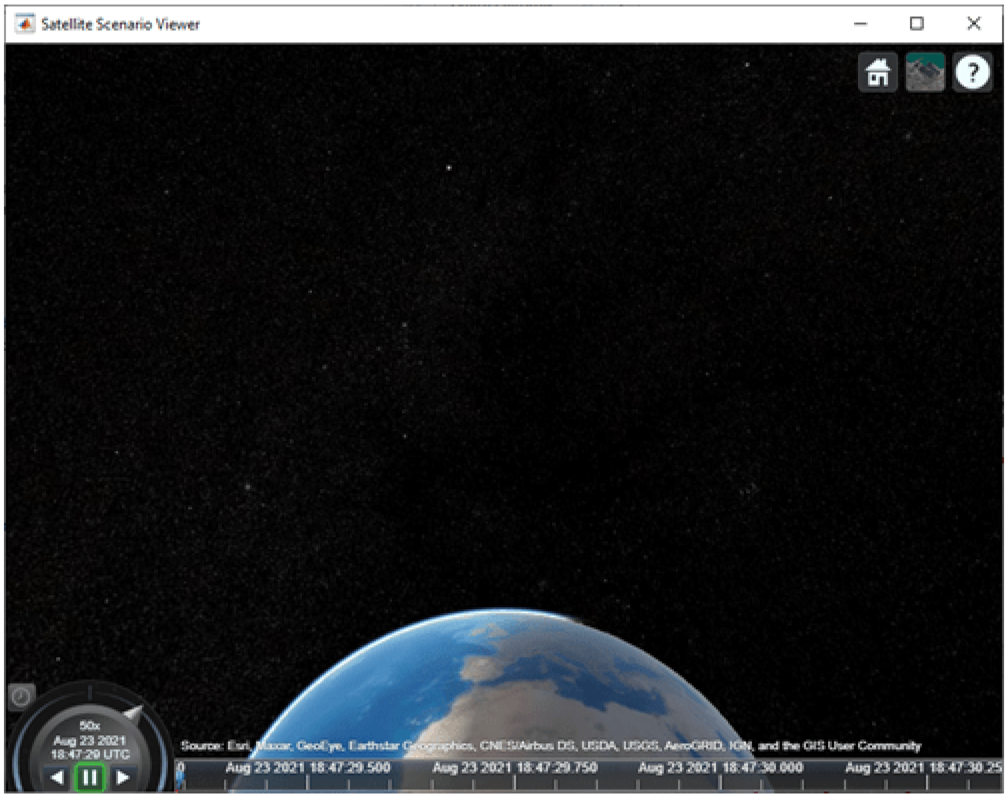
Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```


Satellite Scenario Viewer
50x
Aug 23 2021
18:47:29 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
Aug 23 2021 18:47:29.500
Aug 23 2021 18:47:29.750
Aug 23 2021 18:47:30.000
Aug 23 2021 18:47:30.25

In the Satellite Scenario Viewer, set the pitch angle of the camera to −60 degrees.

```
pitch = -60;      % degrees
campitch(v,pitch);
```


Satellite Scenario Viewer
50x
Aug 23 2021
18:47:29 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
Aug 23 2021 18:47:29.500
Aug 23 2021 18:47:29.750
Aug 23 2021 18:47:30.000

Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[2]

`pitch` — Pitch angle of camera
-90 (default) | scalar the in the range [–90, 90]

Pitch angle of the camera in degrees, specified as a scalar in the range [–90, 90]. By default, the pitch angle is –90 degrees, which means that camera points directly toward the surface of the globe.

Output Arguments

`outPitch` — Camera pitch angle
numeric scalar in the range [-90, 90]

Camera pitch angle in degrees, returned as a numeric scalar.

2 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Tips

- When the pitch angle is near –90 (the default value) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle might change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `camroll` | `campos` | `hideAll` | `camtarget` | `camheading`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

campos

Package: `matlabshared.satellitescenario`

Set or get position of camera for Satellite Scenario Viewer

Syntax

```
campos(viewer,lat,lon)
campos(viewer,lat,lon,height)
campos(viewer)
[latOut,lonOut,heightOut] = campos( ___ )
```

Description

`campos(viewer,lat,lon)` sets the latitude and longitude of the camera for the specified Satellite Scenario Viewer.

`campos(viewer,lat,lon,height)` sets the latitude, longitude, and ellipsoidal height of the camera. If you want to set only the height of the camera, use the `camheight` function instead.

`campos(viewer)` displays the latitude, longitude, and ellipsoidal height of the camera as a three-element vector. `satelliteScenarioViewer` objects use the WGS84 reference ellipsoid.

`[latOut,lonOut,heightOut] = campos( ___ )` sets the position and then returns the latitude, longitude, and height of the camera. Specify any input argument combinations from previous syntaxes.

Examples

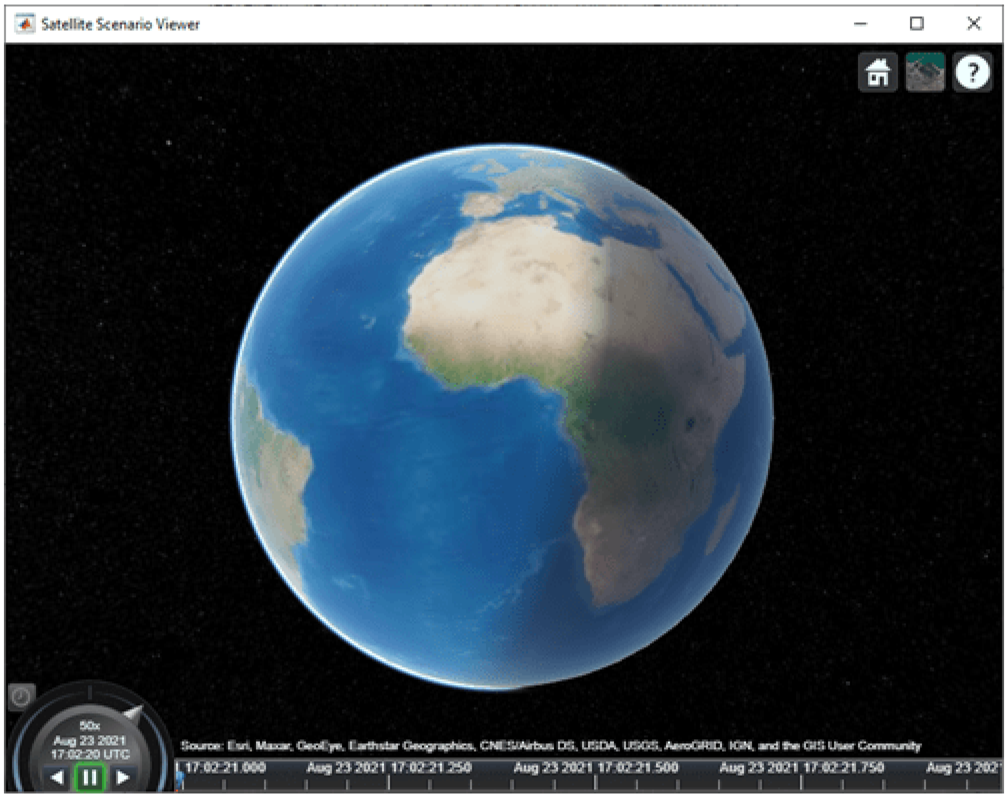
Reposition Camera of Satellite Scenario Viewer

Create a satellite scenario object.

```
sc = satelliteScenario;
```

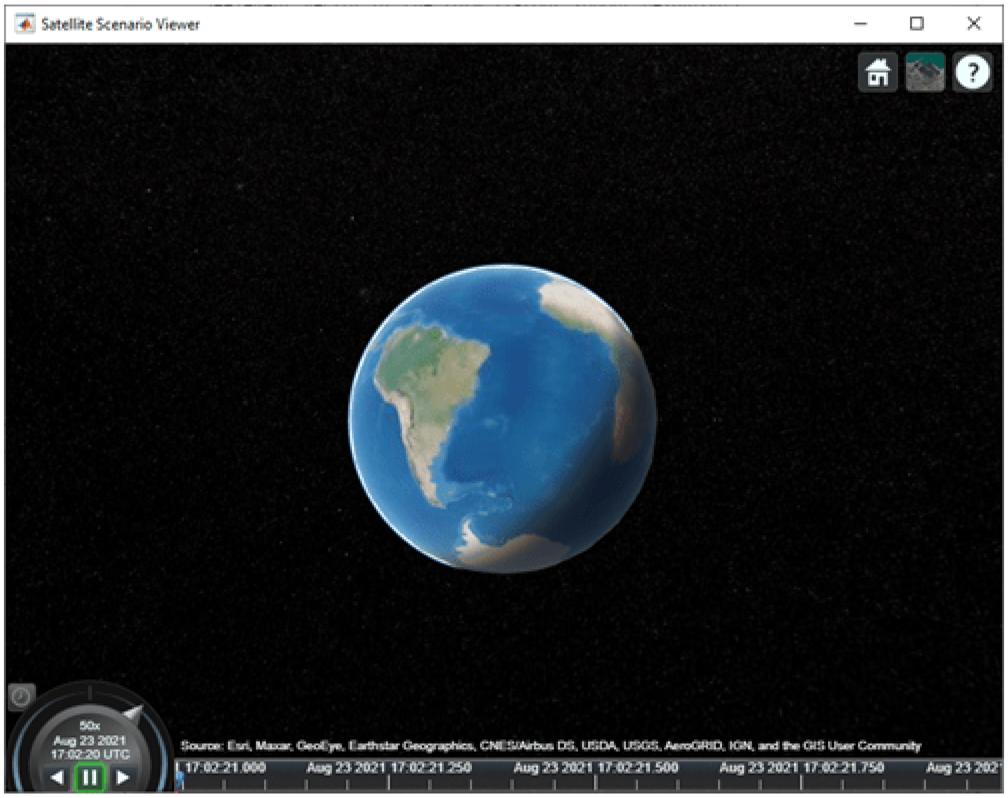
Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```


Satellite Scenario Viewer
50x
Aug 23 2021
17:02:20 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
17:02:21.000
Aug 23 2021 17:02:21.250
Aug 23 2021 17:02:21.500
Aug 23 2021 17:02:21.750

In the Satellite Scenario Viewer, set the latitude and longitude of the camera to −30 degrees and the height of the camera to 30,000 km.

```
latitude = -30;                    % degrees
longitude = -30;                   % degrees
height = 30000000;                 % meters
campos(v,latitude,longitude,height)
```


Satellite Scenario Viewer
50x
Aug 23 2021
17:02:20 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
17:02:21.000
Aug 23 2021 17:02:21.250
Aug 23 2021 17:02:21.500
Aug 23 2021 17:02:21.750

Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[3]

`lat` — Geodetic latitude of camera
`0` (default) | scalar in the range [-90, 90].

Geodetic latitude of the camera in degrees, specified as a scalar in the range [–90, 90].

`lon` — Geodetic longitude of camera
`0` (default) | scalar in the range [-360, 360].

Geodetic longitude of the camera in degrees, specified as a scalar in the range [–360, 360].

`height` — Ellipsoidal height of camera
`15000000` (default) | numeric scalar

Ellipsoidal height of the camera in meters, specified as a numeric scalar.

If you specify the height such that the camera is level with or below the surface of the Earth, then the `campos` function sets the height to a value one meter above the surface.

3 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Output Arguments

`latOut` — Geodetic latitude of camera
numeric scalar

Geodetic latitude of the camera in degrees, returned as a numeric scalar.

`lonOut` — Geodetic longitude of camera
numeric scalar

Geodetic longitude of the camera in degrees, returned as a numeric scalar.

`heightOut` — Ellipsoidal height of camera
numeric scalar

Ellipsoidal height of the camera in meters, returned as a numeric scalar. For more information about ellipsoidal height, see "Geodetic Coordinates".

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `camroll` | `campitch` | `hideAll` | `camtarget` | `camheight` | `camheading`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

camheading

Package: `matlabshared.satellitescenario`

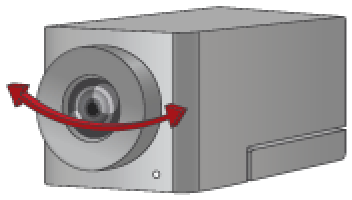
Set or get heading angle of camera for Satellite Scenario Viewer

Syntax

```
camheading(viewer,heading)
outHeading = camheading(viewer,___)
```

Description

`camheading(viewer,heading)` sets the heading angle of the camera for the specified Satellite Scenario Viewer. Setting the heading angle shifts the camera left or right about its *z*-axis.



`outHeading = camheading(viewer,___)` returns the heading angle of the camera. If the second input is `heading`, then the function sets the output equal to the input heading.

Examples

Set Camera Heading Angle of Satellite Scenario Viewer

Create a satellite scenario object.

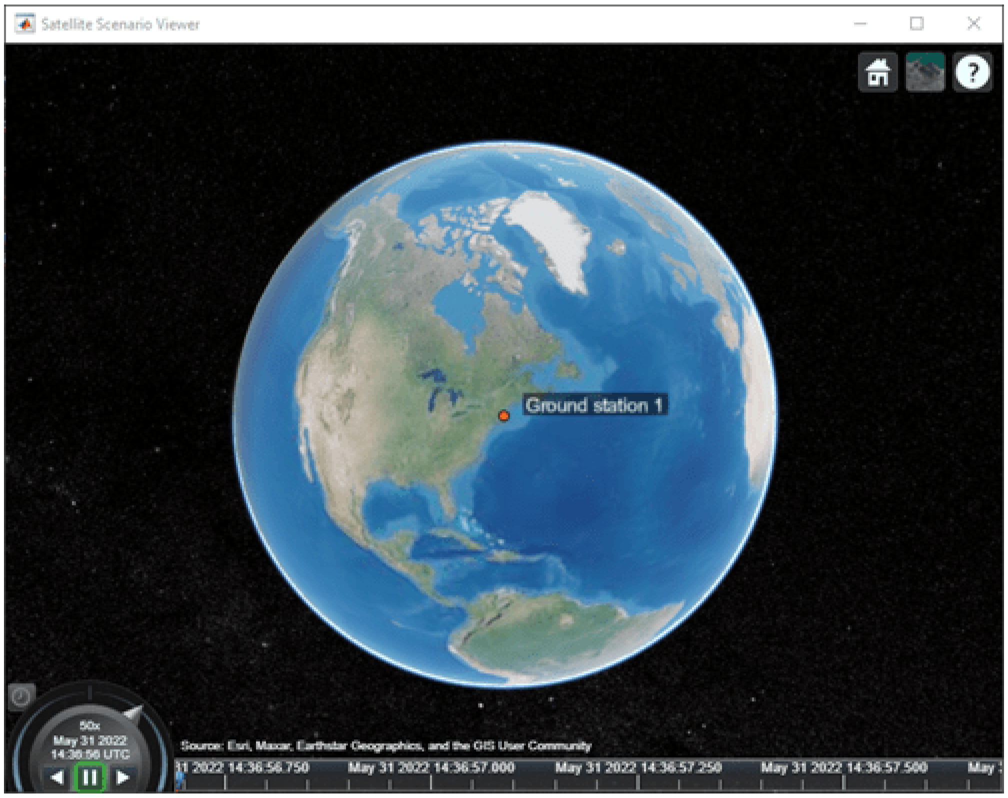
```
sc = satelliteScenario;
```

Add a ground station to the scenario.

```
latitude = 42.3001;                         % degrees
longitude = -71.3504;                       % degrees
groundStation(sc,latitude+0.05,longitude);
```

Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```

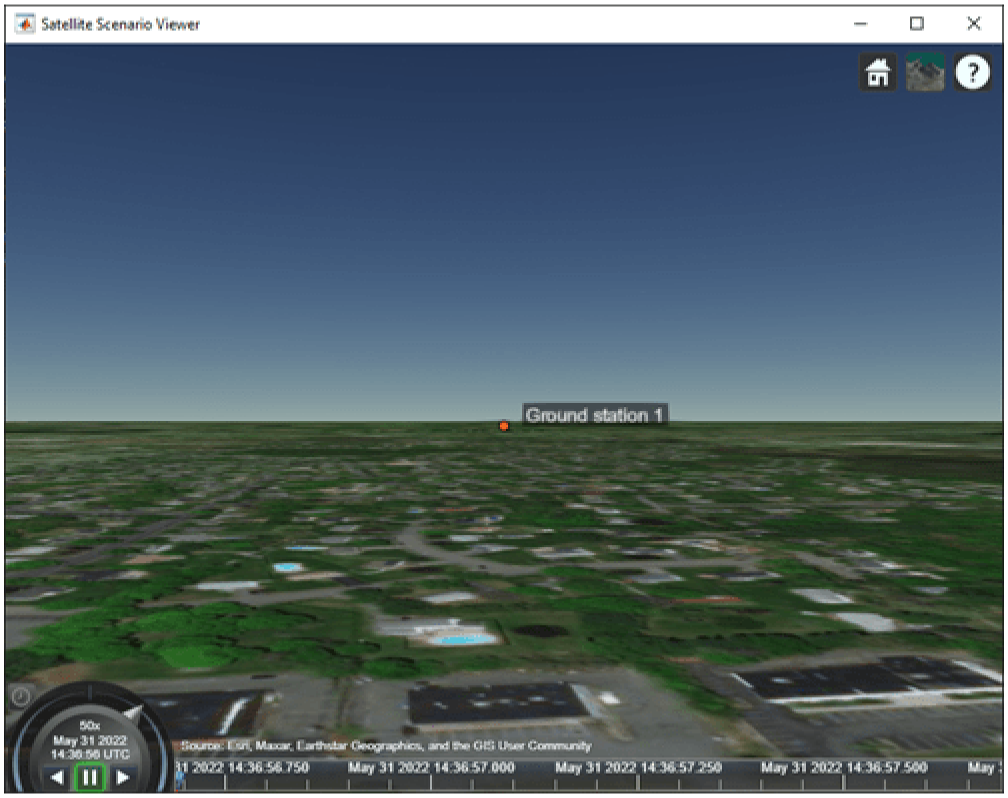

Satellite Scenario Viewer
Ground station 1
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
50x
May 31 2022
14:36:56 UTC

In the Satellite Scenario Viewer, set the height of the camera to 50 meters.

```
height = 50;                                    % meters
campos(v,latitude,longitude,height);
pause(2);
```

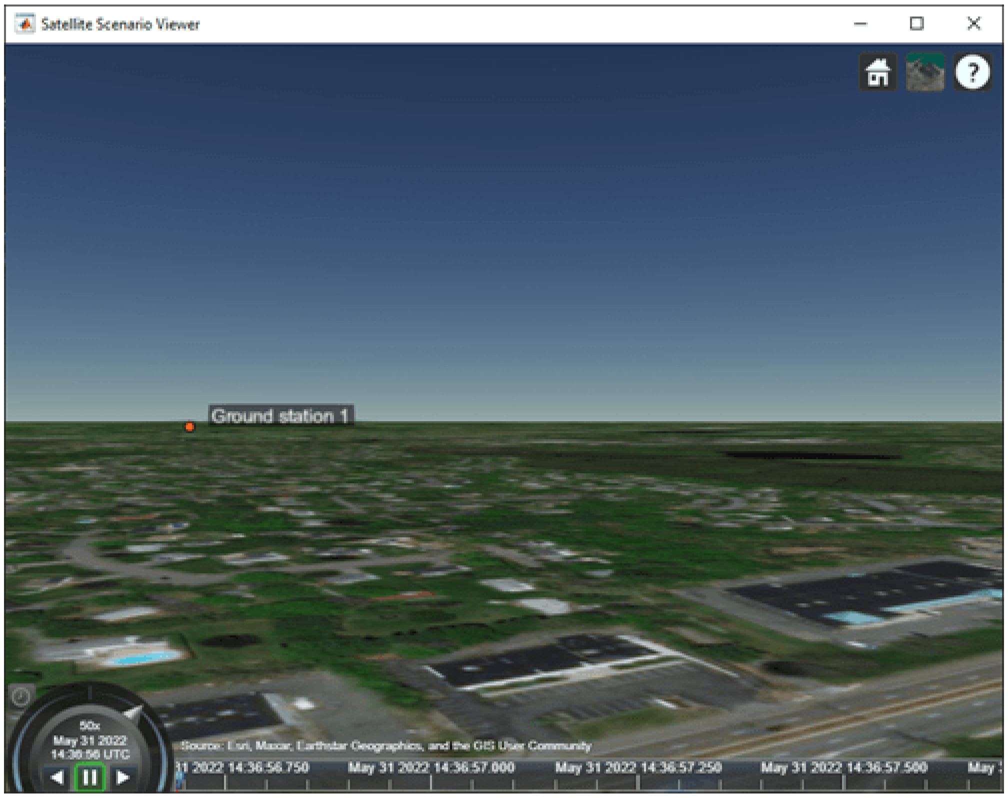
Set the pitch angle of the camera to 0 degrees.

```
pitch = 0;
campitch(v,pitch);
pause(2);
```



Set the heading angle of the camera to 20 degrees.

```
heading = 20;         % degrees
camheading(v,heading);
```



Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[4]

`heading` — Heading angle of camera
`360` (default) | numeric scalar in the range [–360, 360]

Heading angle of the camera in degrees, specified as a scalar value in the range [–360, 360].

Output Arguments

`outHeading` — Camera heading angle
numeric scalar in the range [-360, 360]

Camera heading angle in degrees, returned as a numeric scalar.

Tips

- When the pitch angle is near –90 (the default value) or 90 degrees, the camera loses one rotational degree of freedom. As a result, when you change the roll angle, the heading angle might

4 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

change instead. This phenomenon is called gimbal lock. To avoid the effects of gimbal lock, call the `camheading` function instead of the `camroll` function.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `camroll` | `campitch` | `campos` | `hideAll` | `camtarget` | `camheight`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

camheight

Package: `matlabshared.satellitescenario`

Set or get height of camera for Satellite Scenario Viewer

Syntax

```
camheight(viewer,height)
heightOut = camheight(viewer,___)
```

Description

`camheight(viewer,height)` sets the ellipsoidal height of the camera for the specified Satellite Scenario Viewer.

`heightOut = camheight(viewer,___)` returns the ellipsoidal height of the camera. If the second input is `height`, then the function sets the output equal to the input height.

Examples

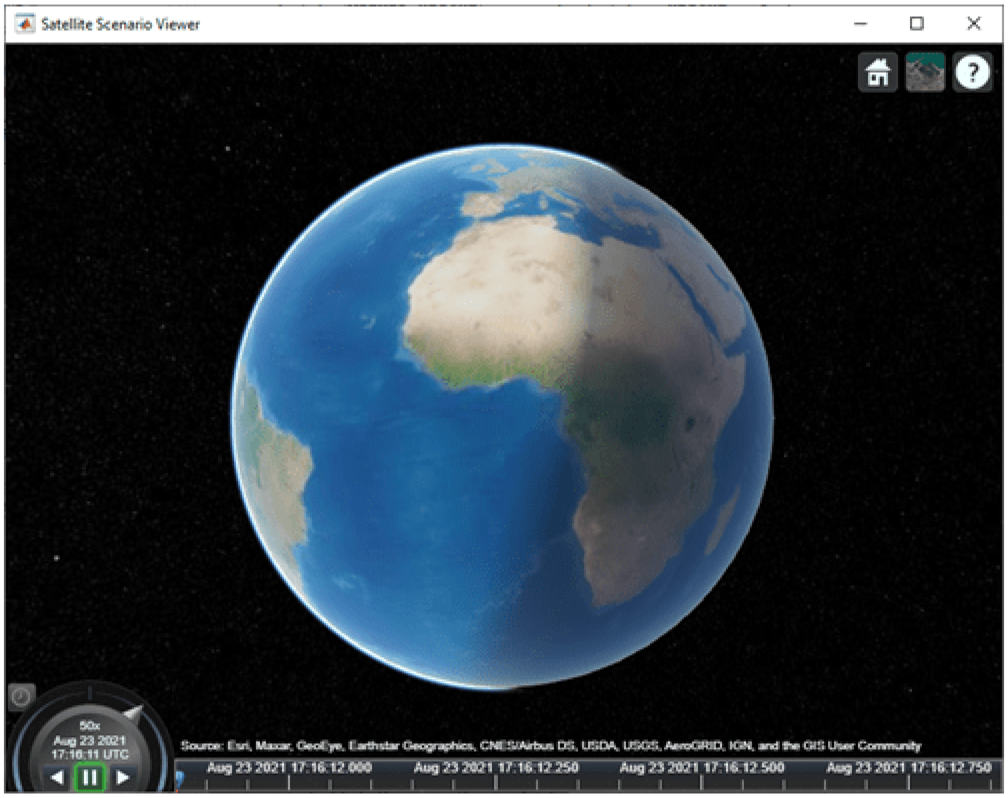
Retrieve Camera Height of Satellite Scenario Viewer

Create a satellite scenario object.

```
sc = satelliteScenario;
```

Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```


Satellite Scenario Viewer
50x
Aug 23 2021
17:16:11 UTC
Source: Esri, Maxar, GeoEye, Earthstar Geographics, CNES/Airbus DS, USDA, USGS, AeroGRID, IGN, and the GIS User Community
Aug 23 2021 17:16:12.000
Aug 23 2021 17:16:12.250
Aug 23 2021 17:16:12.500
Aug 23 2021 17:16:12.750

Retrieve the height of the camera in the Satellite Scenario Viewer.

```
height = camheight(v)
```

```
height = 15000000
```

Input Arguments

viewer — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[5]

height — Ellipsoidal height of camera
`15000000` (default) | numeric scalar

Ellipsoidal height of the camera in meters, specified as a numeric scalar. `satelliteScenarioViewer` objects use the WGS84 reference ellipsoid. For more information about ellipsoidal height, see “Geodetic Coordinates”.

If you specify the height so that the camera is level with or below the surface of the Earth, then the `camheight` function sets the height to a value one meter above the surface.

5 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `camroll` | `campitch` | `campos` | `hideAll` | `camtarget` | `camheading`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

camtarget

Package: `matlabshared.satellitescenario`

Set camera target for Satellite Scenario Viewer

Syntax

```
camtarget(viewer,target)
```

Description

`camtarget(viewer,target)` focuses the camera on the input satellite or ground station. The camera follows `target`. You can reset the camera target by using `camtarget` with a different satellite or ground station or by double-clicking anywhere in the map.

Examples

Set Camera Target to Satellite

Create a satellite scenario object.

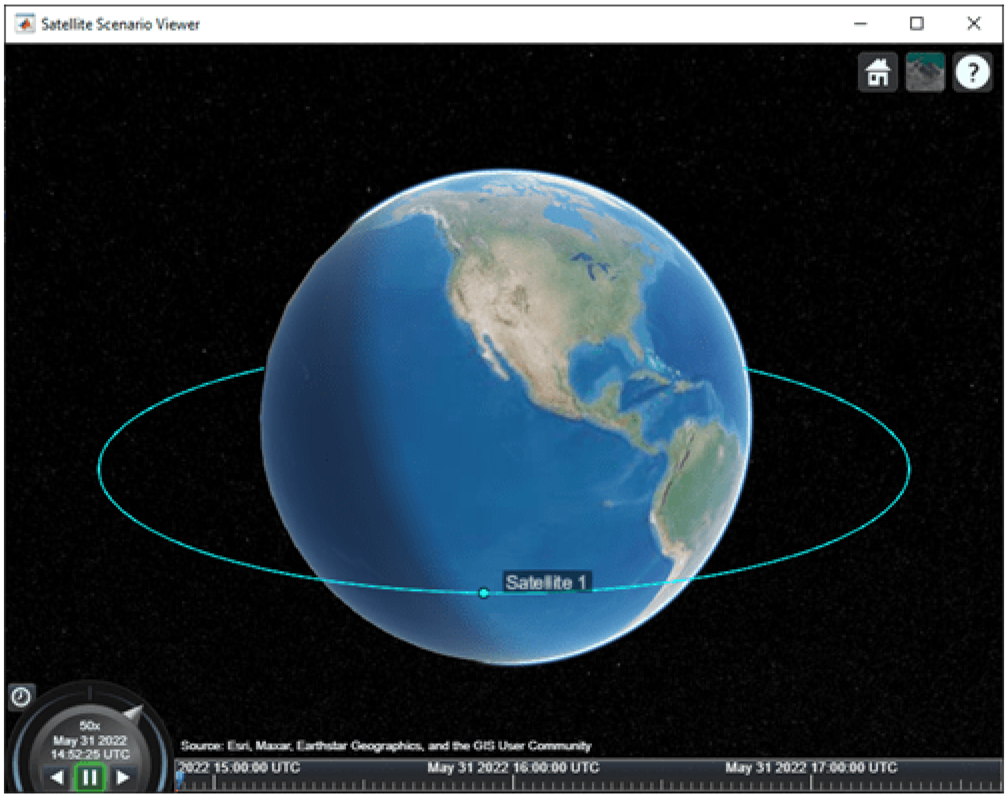
```
sc = satelliteScenario;
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                           % meters
eccentricity = 0;
inclination = 0;                                    % degrees
rightAscensionOfAscendingNode = 0;                  % degrees
argumentOfPeriapsis = 0;                            % degrees
trueAnomaly = 0;                                    % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
      inclination,rightAscensionOfAscendingNode, ...
      argumentOfPeriapsis,trueAnomaly);
```

Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```

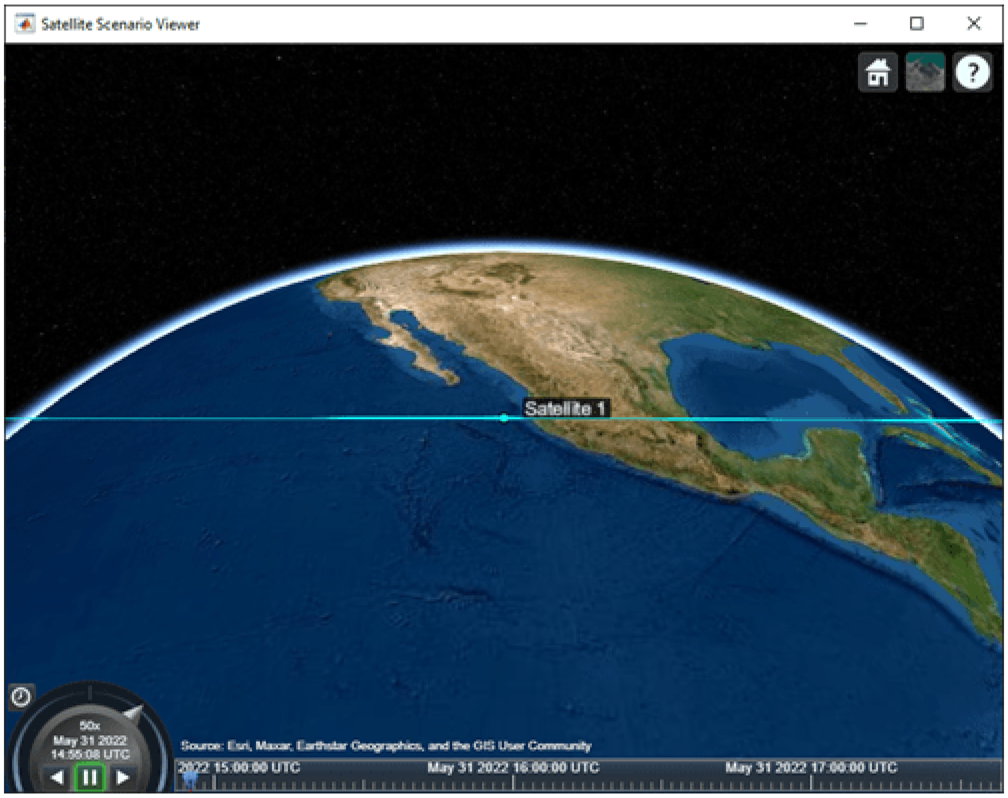


Play the scenario in the viewer.

```
play(sc,"Viewer",v);
```

Set the camera target to the satellite.

```
camtarget(v,sat);
```



Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[6]

`target` — Camera target
`Satellite` object | `GroundStation` object

Camera target, specified as a scalar `Satellite` or `GroundStation` object.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `camroll` | `campitch` | `campos` | `hideAll` | `camheight` | `camheading`

6 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

Topics

“Satellite Scenario Key Concepts”

“Satellite Scenario Basics”

hideAll

Package: `matlabshared.satellitescenario`

Hide all graphics in satellite scenario viewer

Syntax

```
hideAll(viewer)
```

Description

`hideAll(viewer)` hides all graphics in the specified satellite scenario viewer.

Examples

Hide All Graphics from Satellite Scenario Viewer

Create a satellite scenario object.

```
sc = satelliteScenario;
```

Add satellites to the scenario.

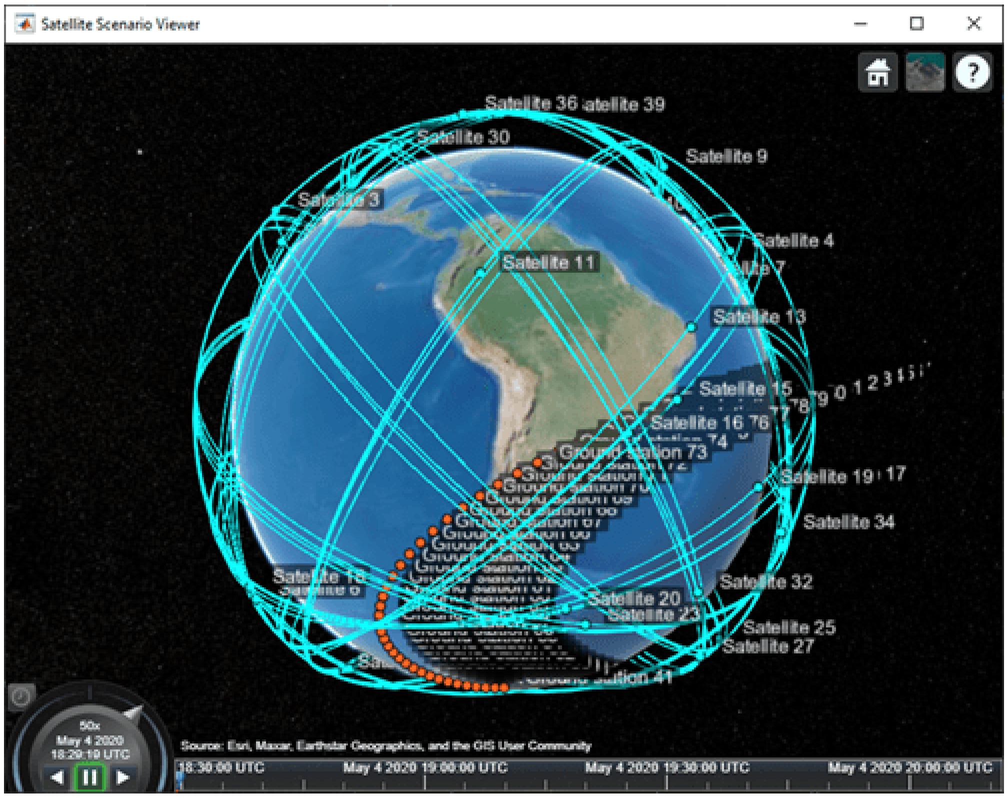
```
tleFile = "leoSatelliteConstellation.tle";
sats = satellite(sc,tleFile);
```

Add a hundred ground stations to the scenario.

```
latitudes = linspace(-90,90,100);            % degrees
longitudes = linspace(-180,180,100);         % degrees
gss = groundStation(sc,latitudes,longitudes);
```

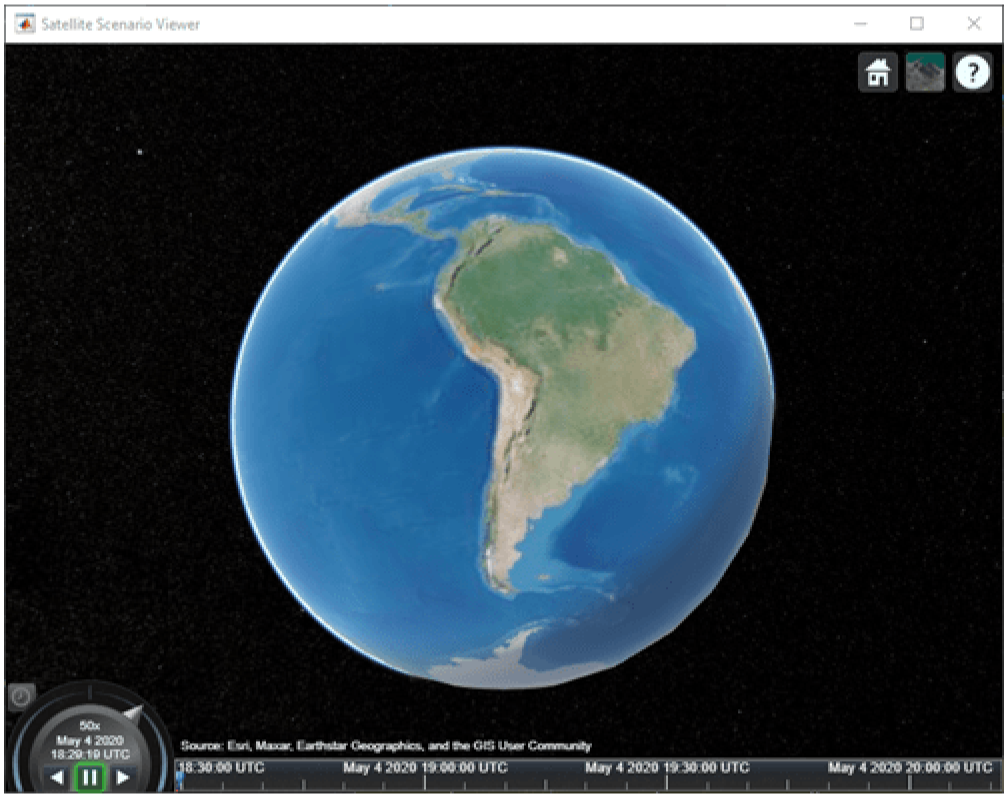
Visualize the scenario using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```



Hide all graphics in the viewer.

```
hideAll(v);
```



Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[7]

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `campos` | `camroll` | `campitch` | `camheading` | `camheight` | `camtarget` | `access` | `groundStation` | `conicalSensor` | `showAll`

Topics
"Comparison of Orbit Propagators"

7 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

showAll

Package: `matlabshared.satellitescenario`

Show all graphics in viewer

Syntax

```
showAll(viewer)
```

Description

`showAll(viewer)` shows all graphics in the specified satellite scenario viewer.

Examples

Show All Hidden Satellite Scenario Objects

Create a satellite scenario object.

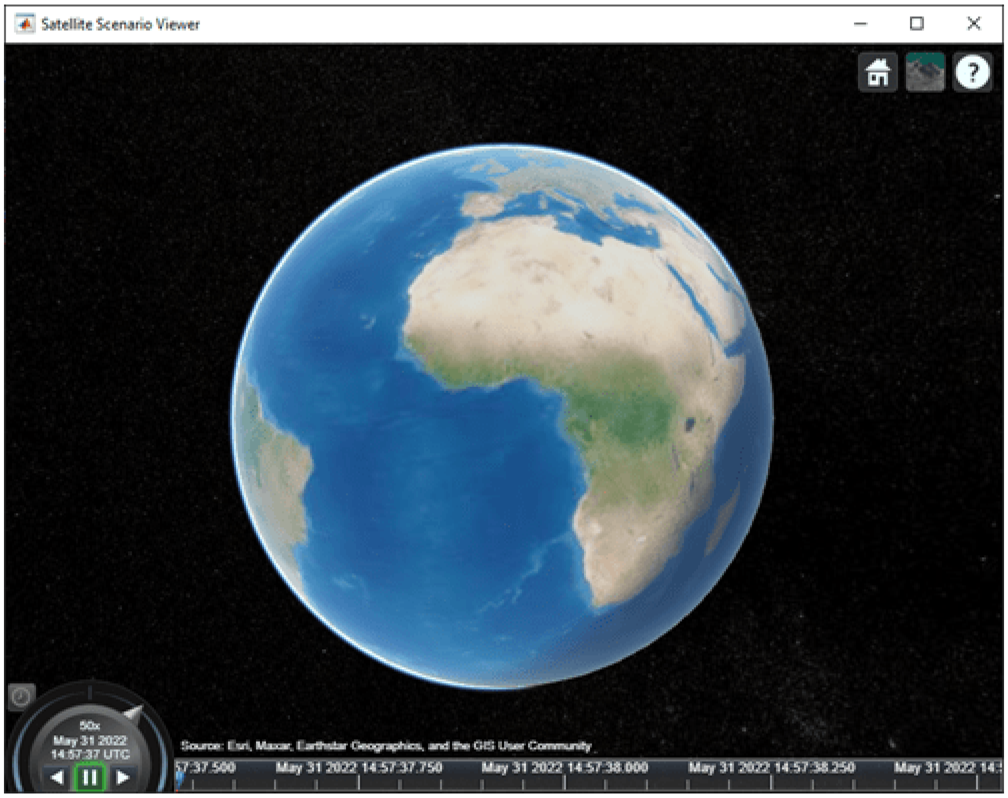
```
sc = satelliteScenario;
```

Specify not to automatically show scenario entities in an open Satellite Scenario Viewer.

```
sc.AutoShow = false;
```

Launch the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```



Add a constellation of satellites to the scenario.

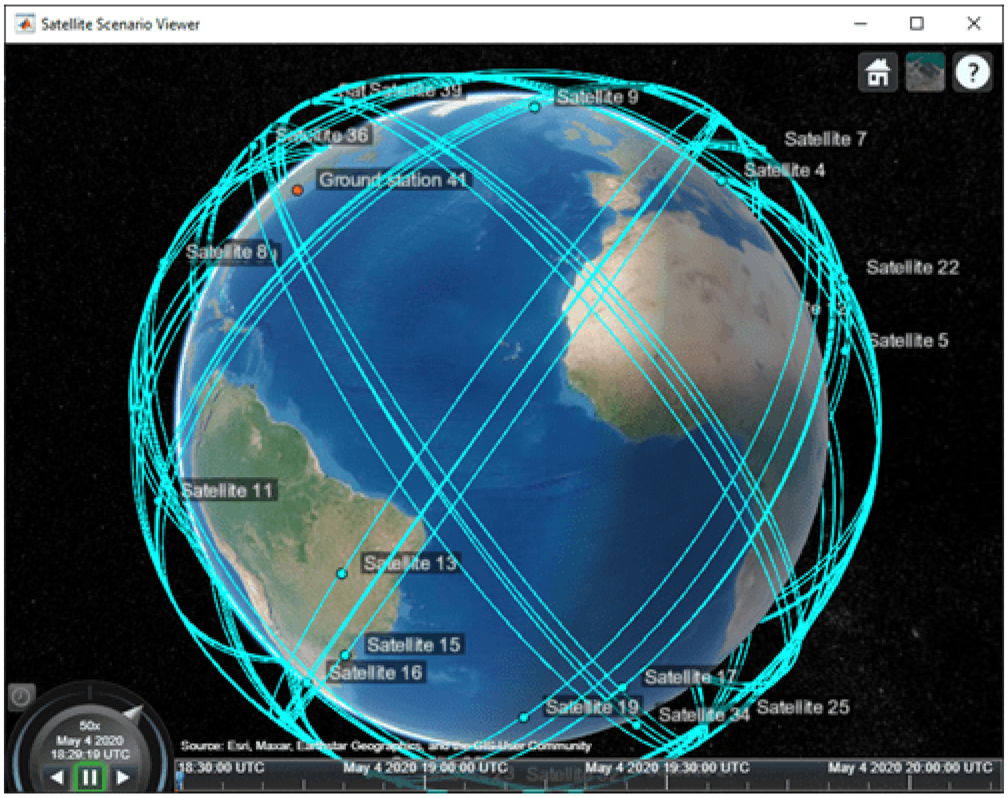
```
tleFile = "leoSatelliteConstellation.tle";
sat = satellite(sc,tleFile);
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Visualize the satellite scenario objects using the Satellite Scenario Viewer.

```
showAll(v);
```



Input Arguments

`viewer` — Satellite Scenario Viewer
`satelliteScenarioViewer` object

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object.[8]

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor`

Functions
`show` | `play` | `hide` | `campos` | `camroll` | `campitch` | `camheading` | `camheight` | `camtarget`

Topics
"Comparison of Orbit Propagators"

8 Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by MathWorks.

“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

accessPercentage

Package: `matlabshared.satellitescenario`

Percentage of time when access exists between first and last node in access analysis

Syntax

```
acpercent = accessPercentage(ac)
```

Description

`acpercent = accessPercentage(ac)` returns the percentage of time from start time to stop time of the satellite scenario when access exists between the first and last node of each `access` object in the input vector, `ac`.

Examples

Calculate Access Percentage Between Ground Station and Satellites

Create a satellite scenario object.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Add satellites to the scenario.

```
semiMajorAxis = [10000000 10000000];                 % meters
eccentricity = [0 0];
inclination = [0 30];                                % degrees
rightAscensionOfAscendingNode = [0 0];               % degrees
argumentOfPeriapsis = [0 0];                         % degrees
trueAnomaly = [0 10];                                % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
      inclination,rightAscensionOfAscendingNode, ...
      argumentOfPeriapsis,trueAnomaly);
```

Add access analysis between the ground station and each satellite.

```
access(gs,sat(1));
access(gs,sat(2));
```

Obtain the access percentage between the ground station and each satellite.

```
ac = gs.Accesses;
acPercent = accessPercentage(ac)
acPercent = 2×1

   15.0000
   14.9306
```

Input Arguments

ac — Access analysis
row vector of `Access` objects

Access analysis, specified as a row vector of `Access` objects.

Outputs Arguments

`acpercent` — Access percentage
row vector of nonnegative numbers

Access percentage, returned as a row vector of nonnegative numbers.

Note When the `AutoSimulate` property of satellite scenario is `true`, the access percentage corresponds to the duration between `StartTime` and `StopTime`. When the property is `false`, the access percentage corresponds to the duration between `StartTime` and `SimulationTime`.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `groundStation` | `satellite` | `access` | `accessIntervals` | `accessStatus`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

linkPercentage

Package: `satcom.satellitescenario`

Percentage of time when link between first and last node in link analysis is closed

Syntax

```
lp = linkPercentage(lnk)
```

Description

`lp = linkPercentage(lnk)` returns the percentage of time from start time to stop time of the satellite scenario when the link between the first and last node is closed.

Examples

Calculate Uplink Percentage Between Ground Station and Satellite

Create a satellite scenario object.

```
startTime = datetime(2020,11,13,7,25,0);
stopTime = startTime + days(1);
sampleTime = 60;                                      % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                              % meters
eccentricity = 0;
inclination = 10;                                      % degrees
rightAscensionOfAscendingNode = 0;                     % degrees
argumentOfPeriapsis = 0;                               % degrees
trueAnomaly = 210;                                     % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a receiver to the satellite.

```
rx = receiver(sat);
```

Add a ground station to the scenario.

```
latitude = 0;                             % degrees
longitude = 30;                           % degrees
gs = groundStation(sc,latitude,longitude);
```

Add a transmitter to the ground station.

```
tx = transmitter(gs,"MountingAngles",[0; 180; 0]);
```

Create an uplink.

```
lnk = link(tx,rx);
```

Calculate the link percentage of the uplink.

```
linkpercent = linkPercentage(lnk)
```

```
linkpercent = 0
```

Input Arguments

`lnk` — Link analysis
`Link` object vector | `Link` object scalar

Link analysis object, specified as a `Link` object vector or scalar.

Outputs Arguments

`lp` — Link percentage
vector of positive numbers | scalar

Link percentage, returned as a vector of positive numbers or a scalar.

Note When the `AutoSimulate` property of the satellite scenario is `true`, the link percentage corresponds to the duration between `StartTime` and `StopTime`. When the property is `false`, the link percentage corresponds to the duration between `StartTime` and `SimulationTime`.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer` | `Link`

Functions
`show` | `play` | `ebno` | `linkStatus` | `linkIntervals` | `groundStation`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

linkStatus

Package: `satcom.satellitescenario`

Status of link closure between first and last node

Syntax

```
linkstat = linkStatus(lnk)
linkstat = linkStatus(lnk,timeIn)
[linkstat,timeOut] = linkStatus( ___ )
```

Description

`linkstat = linkStatus(lnk)` returns a matrix of logicals representing the link closure status history `linkstat` of each link in the vector `link`. The rows of the matrix correspond to the link objects in `link` and the columns correspond to the time sample.

`linkstat = linkStatus(lnk,timeIn)` returns a column vector of status `linkstat` of each link in the vector `link` at the specified datetime `timeIn`. Each element of `linkstat` corresponds to a link in `link`. If no time zone is specified in `timeIn`, the time zone is assumed to be Coordinated Universal Time (UTC).

`[linkstat,timeOut] = linkStatus( ___ )` returns the link closure status and the corresponding times in UTC.

Examples

Obtain Closed Downlink Status History

Create a satellite scenario object.

```
startTime = datetime(2020,10,13,5,30,0);
stopTime = datetime(2020,10,13,5,45,0);
sampleTime = 60;                                       % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                               % meters
eccentricity = 0;
inclination = 0;                                        % degrees
rightAscensionOfAscendingNode = 0;                      % degrees
argumentOfPeriapsis = 0;                                % degrees
trueAnomaly = 210;                                      % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a transmitter to the satellite.

```
tx = transmitter(sat);
```

Add a ground station to the scenario.

```
latitude = 0;                                  % degrees
longitude = 30;                                % degrees
gs = groundStation(sc,latitude,longitude);
```

Add a receiver to the ground station.

```
rx = receiver(gs,"MountingAngles",[0; 180; 0]);
```

Create a downlink.

```
lnk = link(tx,rx);
```

Obtain the link status history of the closed downlink.

```
s = linkStatus(lnk)
```

```
s = 1x16 logical array

   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Input Arguments

lnk — Link analysis
`Link` object vector | `Link` object scalar

Link analysis object, specified as a `Link` object vector or scalar.

timeIn — Time at which output is calculated
`datetime` scalar

Time at which the output is calculated, specified as a `datetime` scalar. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

Output Arguments

linkstat — Link closure status
matrix of logical values

Link closure status, returned as a matrix of logical values representing the link closure status history `linkstat` of each link in the vector `link`. The rows of the matrix correspond to the link objects in `link` and the columns correspond to the time sample. The status at a given instant is `1` (`true`) if the link between the first and last node is closed. The link between the first and last node is closed when the link between each individual pair of intermediate adjacent nodes in the Sequence property of the link is closed.

- When both nodes of a pair belong to the same satellite or ground station, the link is considered closed.
- Otherwise, the link between the pair is closed if the directionality is from a transmitter to a receiver and the energy per bit to noise power spectral density ratio (Eb/No) at the receiver is greater than its `RequiredEbNo` property.

If a given node is attached to a ground station directly or via a gimbal, the elevation angle of the adjacent node with respect to the ground station must be greater than or equal to its `MinElevationAngle` property.

`timeOut` — Time samples of output link status
scalar | vector

Time samples of the output link status, returned as a scalar or a vector. If the time history of the link status is returned, `timeOut` is a row vector.

Note When the `AutoSimulate` property of the satellite scenario is `true`, the link status history from `StartTime` to `StopTime` is returned. When the property is `false`, the link status history from `StartTime` to `SimulationTime` is returned.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `groundStation` | `satelliteScenarioViewer` | `Link`

Functions
`show` | `play` | `ebno` | `linkPercentage` | `linkIntervals`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

linkIntervals

Package: `satcom.satellitescenario`

Intervals during which link is closed

Syntax

```
interval = linkIntervals(lnk)
```

Description

`interval = linkIntervals(lnk)` returns a table of intervals during which the link between the first node and last node in each link object input vector is closed.

Examples

Obtain Downlink Closed Intervals Between Satellite and Ground Station

Create a satellite scenario object.

```
startTime = datetime(2020,10,13,7,25,0);
stopTime = startTime + days(1);
sampleTime = 60;                                       % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                               % meters
eccentricity = 0;
inclination = 10;                                       % degrees
rightAscensionOfAscendingNode = 0;                      % degrees
argumentOfPeriapsis = 0;                                % degrees
trueAnomaly = 210;                                      % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a transmitter to the satellite.

```
tx = transmitter(sat);
pattern(tx);
```

Add a ground station to the scenario.

```
latitude = 0;                         % degrees
longitude = 30;                       % degrees
gs = groundStation(sc,latitude,longitude);
gb = gimbal(gs);
```

Add a receiver to the ground station.

```
rx = receiver(gb);
pattern(rx,tx.Frequency);
```

Create a downlink.

```
pointAt(gb,sat);
pointAt(sat,gs);
lnk = link(tx,rx);
```

Obtain the intervals table of the closed downlink.

```
intervals = linkIntervals(lnk)
```

```
intervals=8×8 table
        Source            Target       IntervalNumber        StartTime                EndTime
    ________________    _____________    ______________    ____________________    ________________

    "Transmitter 2"    "Receiver 5"          1           13-Oct-2020 07:25:00    13-Oct-2020 07:2
    "Transmitter 2"    "Receiver 5"          2           13-Oct-2020 09:42:00    13-Oct-2020 10:3
    "Transmitter 2"    "Receiver 5"          3           13-Oct-2020 12:50:00    13-Oct-2020 13:4
    "Transmitter 2"    "Receiver 5"          4           13-Oct-2020 15:58:00    13-Oct-2020 16:4
    "Transmitter 2"    "Receiver 5"          5           13-Oct-2020 19:05:00    13-Oct-2020 19:5
    "Transmitter 2"    "Receiver 5"          6           13-Oct-2020 22:12:00    13-Oct-2020 23:0
    "Transmitter 2"    "Receiver 5"          7           14-Oct-2020 01:20:00    14-Oct-2020 02:1
    "Transmitter 2"    "Receiver 5"          8           14-Oct-2020 04:27:00    14-Oct-2020 05:1
```

Input Arguments

`lnk` — Link analysis
`Link` object vector | `Link` object scalar

Link analysis object, specified as a `Link` object vector or scalar.

Output Arguments

`interval` — Link closed intervals
table

Intervals during which the link is closed, returned as a table.

Each row of the table denotes a specific interval, and the columns of the table are named as `Source`, `Target`, `IntervalNumber`, `StartTime`, `EndTime`, `Duration` (in seconds), `StartOrbit`, and `EndOrbit`. `Source` and `Target` are the names of the first and last node, respectively, which define the link analysis.

- If `Source` is directly or indirectly attached to a satellite, then `StartOrbit` and `EndOrbit` correspond to the satellite associated with `Source`.
- If `Target` is directly or indirectly attached to a satellite, then `StartOrbit` and `EndOrbit` correspond to the satellite associated with `Target`. Otherwise, `StartOrbit` and `EndOrbit` are `NaN` because they are associated with ground stations.

Note When the `AutoSimulate` property of satellite scenario is `true`, the link intervals between `StartTime` and `StopTime` are returned. When the property is `false`, the link intervals between `StartTime` and `SimulationTime` are returned.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `groundStation` | `satelliteScenarioViewer` | `Link`

Functions
`show` | `play` | `linkPercentage` | `linkStatus` | `ebno`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

aer

Package: `matlabshared.satellitescenario`

Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame

Syntax

```
az = aer(asset,target)
[az,el] = aer(asset,target)
[az,el,range] = aer(asset,target)
[az,el,range,timeOut] = aer(asset,target)
[ ___ ] = aer(asset,target,timeIn)
[ ___ ] = aer( ___ ,coordinateFrame='ned')
```

Description

`az = aer(asset,target)` returns a 2-D array of the history of azimuth angles `az`, between `asset` and `target` belonging to a given `satelliteScenario` object.

`[az,el] = aer(asset,target)` returns the history of elevation angles, `el`, between satellite or ground station `asset` and another satellite or ground station `target`.

`[az,el,range] = aer(asset,target)` returns row vectors of the history of the `range` of `Satellite` or `GroundStation` in `target` with respect to those in `asset`.

`[az,el,range,timeOut] = aer(asset,target)` returns the corresponding time in `timeOut`.

`[ ___ ] = aer(asset,target,timeIn)` returns the outputs at the specified datetime `timeIn`. `az`, `el`, and `range` are structured the same way as described in syntaxes with an exception that the size of the second dimension is fixed at 1, representing the values at the specified time `timeIn`.

`[ ___ ] = aer( ___ ,coordinateFrame='ned')` returns the `az`, `el`, `range`, and `timeOut` based on the specified output arguments and the coordinate frame defined by the name-value argument.

Note By default, the `aer` function uses the north-east-down (NED) system to perform all the calculations.

Examples

Determine AER of Ground Station

Create a satellite scenario object.

```
startTime = datetime(2021,4,25);                    % April 25, 2021, 12:00 AM UTC
stopTime = datetime(2021,4,26);                     % April 26, 2021, 12:00 AM UTC
sampleTime = 60;                                    % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
tleFile = "eccentricOrbitSatellite.tle";
sat = satellite(sc,tleFile);
```

Add a ground station to the scenario using default properties.

```
gs = groundStation(sc);
```

Determine the azimuth angle, elevation angle, and range of the ground station with respect to the satellite at April 25, 2021, 1:26 AM UTC.

```
time = datetime(2021,4,25,1,26,0);
[azimuth,elevation,range] = aer(sat,gs,time)
```

```
azimuth = 15.2962

elevation = -70.3858

range = 1.3442e+07
```

Input Arguments

`asset` — First scenario component
scalar | vector

First scenario component, specified as a `Satellite`, `GroundStation`, `ConicalSensor`, `Gimbal`, `Transmitter`, or a `Receiver` object.

`target` — Second scenario component
scalar | vector

Second scenario component, specified as a `Satellite`, `GroundStation`, `ConicalSensor`, `Gimbal`, `Transmitter`, or a `Receiver` object.

`timeIn` — Time at which output is calculated
datetime

Time at which output is calculated, specified as a datetime. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

`coordinateFrame` — Coordinate frame
`'ned'` (default) | `'body'`

Coordinate frame, specified as either `'ned'` or `'body'`.

- When `coordinateFrame` is `'ned'` — The azimuth angle is defined in the North-East-Down (NED) frame of (and centered at) `asset` such that 0 degrees is North, 90 degrees is East, 180 degrees is South, and 270 degrees is West. The elevation angle is defined in the NED frame of (and centered at) `asset` such that 0 degrees implies `target` is on the North East (NE) plane, 90 degrees implies `target` is directly above `asset`, and -90 degrees implies `target` is directly below `asset`.
- When `coordinateFrame` is `'body'` — The azimuth angle is the angle between the projection of the relative position vector of `target` on the *x*-*y* plane of the body frame of `asset`, and the *x*-axis of `asset`. The angle is positive for positive (clockwise) rotation about the *z*-axis of `asset`. The elevation angle is the angle between the relative position vector of `target` on the *x*-*y* plane of the

body frame of `asset`. The angle is positive when the *z* component of the relative position of `target` defined in the body frame of `asset` is negative.

Output Arguments

`az` — Azimuth angles

vector | 2-D array | scalar

Azimuth angles of the target in the local azimuth, elevation, and range (AER) system in degrees, returned as a vector, 2-D array, or scalar in the range [0,360). Azimuths are measured clockwise from North. If the `timeIn` argument is not specified, the vector elements correspond to the time samples specified by the `SampleTime` property from the satellite scenario `StartTime` to `StopTime`.

- If both `asset` and `target` are scalars, `az` is a row vector where each element represents the azimuth angle of `target` with respect to `asset` in the NED frame of `asset` at a specified time sample.
- If `asset` is a scalar and `target` is a vector, `az` is a 2-D array, where each row represents the azimuth angle of each element in `target` with respect to `asset` in the NED frame of `asset` and the columns represent the time samples.
- If `asset` is a vector and `target` is a scalar, `az` is a 2-D array, where each row represents the azimuth angle of `target` with respect to each element in `asset` in the NED frame of the element in `asset` and the columns represent the time samples.
- If both `asset` and `target` are vectors, the length of `asset` must equal the length of `target`. The `az` is a 2-D array, where each row index corresponds to the index in `asset` and `target`, and represents the azimuth angle of the element at the index in `target` with respect to the element at the index in `asset` in the NED frame of that element in `asset`. The columns represent the time samples.

If the `timeIn` argument is not specified and when the `AutoSimulate` property of the satellite scenario is `true`, aer function returns the `az` history from `StartTime` to `StopTime`. Otherwise, it returns the `az` history from `StartTime` to `SimulationTime`.

`el` — Elevation angles

vector | 2-D array | scalar

Elevation angles of target in the local AER system in degrees, returned as a vector, 2-D array, or scalar in the range [-90 90]. Elevations are measured with respect to a plane that is perpendicular to the normal of the surface of the earth. If `asset` is on the surface of the Earth, then the plane is tangential to the Earth. If the `timeIn` argument is not specified, the vector elements correspond to the time samples specified by the `SampleTime` property from the satellite scenario `StartTime` to `StopTime`.

If the `timeIn` argument is not specified and when the `AutoSimulate` property of the satellite scenario is `true`, aer function returns the `el` history from `StartTime` to `StopTime`. Otherwise, it returns the `el` history from `StartTime` to `SimulationTime`.

`range` — Distances from local origin

vector | 2-D array | scalar

Distances from the local origin in meters, returned as a vector, 2-D array, or a scalar. The `range` array is structured the same way as the `az` and `el`, described in the above syntaxes.

If the `timeIn` argument is not specified and when the `AutoSimulate` property of the satellite scenario is `true`, aer function returns the `range` history from `StartTime` to `StopTime`. Otherwise, it returns the `range` history from `StartTime` to `SimulationTime`.

`timeOut` — Time samples between start and stop time of scenario
row vector | scalar

Time samples corresponding to `az`, `el`, and `range` in UTC, returned as a row vector, or a scalar.

If the `timeIn` argument is not specified and when the `AutoSimulate` property of the satellite scenario is `true`, aer function returns the time sample history from `StartTime` to `StopTime`. Otherwise, it returns the time sample history from `StartTime` to `SimulationTime`.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `access` | `groundStation` | `conicalSensor` | `transmitter` | `receiver` | `hide`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

accessIntervals

Package: `matlabshared.satellitescenario`

Intervals during which access status is true

Syntax

```
acinterval = accessIntervals(ac)
```

Description

`acinterval = accessIntervals(ac)` returns a table of intervals during which the access status corresponding to each `access` object in the input vector, `ac`, is `true`.

Examples

Add Ground Stations to Scenario and Visualize Access Intervals

Create a satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = 10;
lon = -30;
gs = groundStation(sc,lat,lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

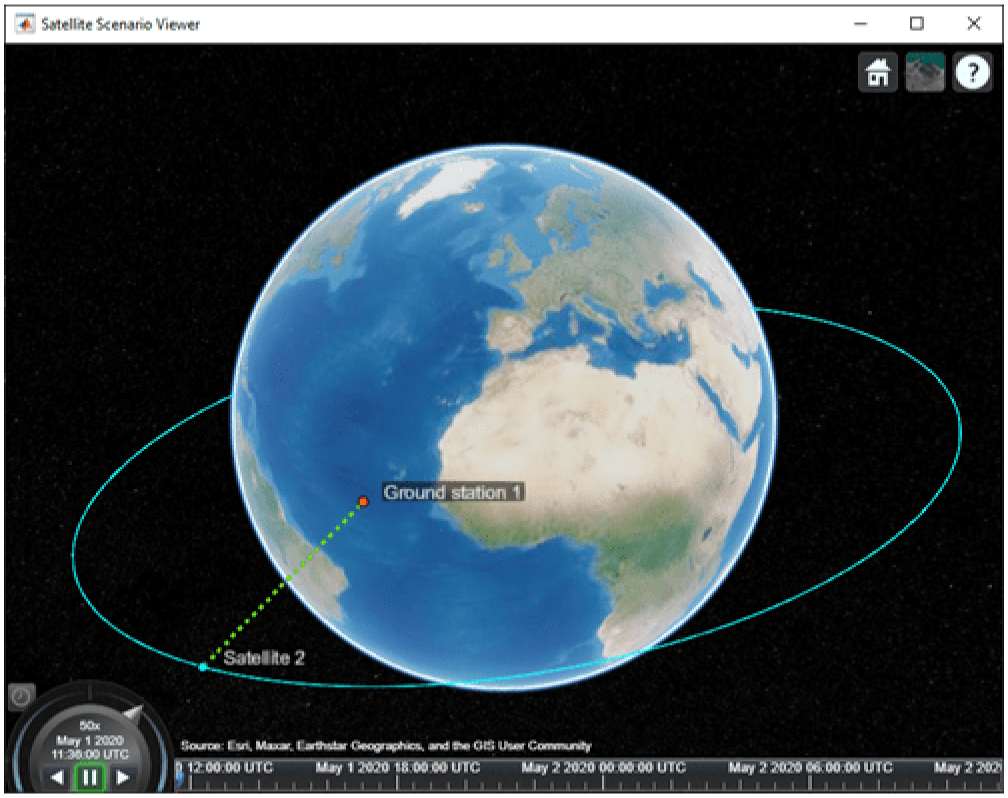
Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat,gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"           1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"           2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"          3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"          4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"          5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"          6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"          7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"          8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Input Arguments

ac — Access analysis
row vector of `Access` objects

Access analysis, specified as a row vector of `Access` objects.

Outputs Arguments

`acinterval` — Intervals during which access is true
table

Intervals during which access is `true`, returned as a table.

Each row of the table denotes a specific interval. The columns of the table are named `Source`, `Target`, `IntervalNumber`, `StartTime`, `EndTime`, `Duration` (in seconds), `StartOrbit`, and

EndOrbit. Source and Target are the names of the first and last node, respectively, which define the access analysis.

- If Source is a satellite or an object that is directly or indirectly attached to a satellite, then StartOrbit and EndOrbit correspond to the satellite associated with Source.
- If Target is a satellite or an object that is directly or indirectly attached to a satellite, then StartOrbit and EndOrbit correspond to the satellite associated with Target. Otherwise, StartOrbit and EndOrbit are NaN because they are associated with ground stations.

Note When the AutoSimulate property of satellite scenario is true, the access intervals between StartTime and StopTime are returned. When the property is false, the access intervals between StartTime and SimulationTime are returned.

Version History

Introduced in R2021a

See Also

Objects
satelliteScenario | satelliteScenarioViewer

Functions
show | play | hide | groundStation | accessPercentage | accessStatus | access

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

orbitalElements

Package: `matlabshared.satellitescenario`

Orbital elements of satellites in scenario

Syntax

```
elements = orbitalElements(sat)
```

Description

`elements = orbitalElements(sat)` returns the orbital elements of the specified satellite `sat`.

Examples

Retrieve Orbital Elements of Satellite

Create a satellite scenario object.

```
sc = satelliteScenario;
```

Add a satellite to the scenario.

```
tleFile = "eccentricOrbitSatellite.tle";
sat1 = satellite(sc,tleFile);
```

Retrieve the orbital elements of `sat1`.

```
elements1 = orbitalElements(sat1)
```

```
elements1 = struct with fields:
                      MeanMotion: 0.0083
                    Eccentricity: 0.7415
                     Inclination: 60.0000
    RightAscensionOfAscendingNode: 30.0000
              ArgumentOfPeriapsis: 280
                     MeanAnomaly: 289.4697
                          Period: 43200
                           Epoch: 05-May-2020 13:51:55
                           BStar: 0
```

Add a satellite from Keplerian elements to the scenario.

```
semiMajorAxis = 6878137;                               % meters
eccentricity = 0;
inclination = 20;                                      % degrees
rightAscensionOfAscendingNode = 0;                     % degrees
argumentOfPeriapsis = 0;                               % degrees
trueAnomaly = 0;                                       % degrees
sat2 = satellite(sc,semiMajorAxis,eccentricity, ...
       inclination,rightAscensionOfAscendingNode, ...
```

```
        argumentOfPeriapsis,trueAnomaly, ...
        "OrbitPropagator","two-body-keplerian", ...
        "Name","Sat2");
```

Retrieve the orbital elements of `sat2`.

```
elements2 = orbitalElements(sat2)
```

```
elements2 = struct with fields:
                      SemiMajorAxis: 6878137
                       Eccentricity: 0
                        Inclination: 20
    RightAscensionOfAscendingNode: 0
                ArgumentOfPeriapsis: 0
                        TrueAnomaly: 0
                             Period: 5.6770e+03
```

Input Arguments

sat — Satellite
scalar

Satellite object, specified as a scalar.

Output Arguments

elements — Orbital elements
structure

Orbital elements of the input `sat`, returned as a structure. The fields of the structure depend on the value of the `OrbitPropagator` property of the `satelliteScenario` object.

For more information on orbital elements, see “Orbital Elements”.

Two-Body Keplerian — Two-Body Keplerian orbit propagator fields
structure

The orbital elements are defined in the Geocentric Celestial Reference Frame (GCRF).

Field	Description
`SemiMajorAxis`	Semimajor axis, in meters
`Eccentricity`	Eccentricity
`Inclination`	Inclination angle, in degrees
`RightAscensionOfAscendingNode`	Right ascension of Ascending node, in degrees
`ArgumentOfPeriapsis`	Argument of periapsis, in degrees
`TrueAnomaly`	True anomaly at reference time, in degrees
`Period`	Orbital period, in seconds

Data Types: `struct`

`SGP4` and `SDP4` — SGP4 and SDP4 orbit propagator fields
structure

The orbital elements represent general perturbation mean elements.

Field	Description
`MeanMotion`	General perturbation mean motion, in degrees per second
`Eccentricity`	Mean eccentricity
`Inclination`	Mean inclination angle, in degrees
`RightAscensionOfAscendingNode`	Mean right ascension of Ascending node, in degrees
`ArgumentOfPeriapsis`	Mean argument of periapsis, in degrees
`MeanAnomaly`	Mean mean anomaly at the reference time, in degrees
`Epoch`	Epoch
`BStar`	Drag term, in per EarthRadius
`Period`	Mean orbital period, in seconds

Data Types: `struct`

`Ephemeris` — Ephemeris orbit propagator fields
structure

Field	Description
`EphemerisStartTime`	Ephemeris start time
`EphemerisStopTime`	Ephemeris stop time
`PositionTimeTable`	Position timetable
`VelocityTimeTable`	Velocity timetable

Data Types: `struct`

`GPS` — GPS orbit propagator fields
structure

The orbital elements are derived from the SEM almanac file or RINEX GPS navigation message, and defined in the Earth-Centered-Earth-Fixed (ECEF) frame.

Field	Description
`PRN`	Pseudorandom noise number
`SatelliteHealth`	Satellite health
`GPSWeekNumber`	GPS week number, with time of ephemeris.
`GPSTimeOfApplicability`	Time of Applicability, in seconds
`SemiMajorAxis`	Semimajor axis, in meters
`Eccentricity`	Eccentricity

Field	Description
`Inclination`	Inclination angle at reference time, in degrees
`GeographicLongitudeOfOrbitalPlane`	Longitude of ascending node of orbit plane at weekly epoch, in degrees
`RateOfRightAscension`	Reference rate of right ascension, in degrees per second
`ArgumentOfPerigee`	Argument of perigee, in degrees
`MeanAnomaly`	Mean anomaly at the reference time, in degrees
`Period`	Orbital period, in seconds

Data Types: `struct`

`Galileo` — Galileo orbit propagator fields
structure

The orbital elements are derived from the RINEX Galileo navigation message, and defined in the Earth-Centered-Earth-Fixed (ECEF) frame.

Field	Description
`SatelliteID`	Satellite system number
`GALWeekNumber`	GAL week number
`TimeofEphemeris`	Time of ephemeris, in seconds
`SatelliteHealth`	Satellite health
`SemiMajorAxis`	Semimajor axis, in meters
`Eccentricity`	Eccentricity
`Inclination`	Inclination angle at reference time, in degrees
`GeographicLongitudeOfOrbitalPlane`	Longitude of ascending node of orbit plane at weekly epoch, in degrees
`RateOfRightAscension`	Reference rate of right ascension, in degrees per second
`ArgumentOfPerigee`	Argument of perigee, in degrees
`MeanAnomaly`	Mean anomaly at the reference time, in degrees
`Period`	Orbital period, in seconds

Data Types: `struct`

Data Types: `struct`

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`groundStation` | `conicalSensor` | `transmitter` | `receiver` | `satellite`

Topics
"Model, Visualize, and Analyze Satellite Scenario"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

accessStatus

Package: `matlabshared.satellitescenario`

Status of access between first and last node defining access analysis

Syntax

```
s = accessStatus(ac)
s = accessStatus(ac,timeIn)
[s,timeOut] = accessStatus( ___ )
```

Description

`s = accessStatus(ac)` returns a matrix `s` of the access status history between the first and last node corresponding to each `access` object in the input vector `ac`.

`s = accessStatus(ac,timeIn)` returns the status of each access analysis object at the specified datetime in `timeIn`. Each element of `s` corresponds to an access object in `ac`.

`[s,timeOut] = accessStatus( ___ )` returns the status of each access analysis object and the corresponding datetime in Coordinated Universal Time (UTC).

Examples

Obtain Access Status Between Satellite and Ground Station

Create a satellite scenario object.

```
startTime = datetime(2021,4,30);                     % 30 April 2021, 12:00 AM UTC
stopTime = datetime(2021,5,1);                       % 1 May 2021, 12:00 AM UTC
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                            % meters
eccentricity = 0;
inclination = 10;                                    % degrees
rightAscensionOfAscendingNode = 0;                   % degrees
argumentOfPeriapsis = 0;                             % degrees
trueAnomaly = 0;                                     % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Add access analysis between the satellite and the ground station.

```
ac = access(sat,gs);
```

Obtain the access status at 30 April 2021, 5:34 PM UTC.

```
time = datetime(2021,4,30,17,34,0);
s = accessStatus(ac,time)
```

```
s = logical
   0
```

Input Arguments

ac — Access analysis
row vector of `Access` objects

Access analysis, specified as a row vector of `Access` objects.

`timeIn` — Time at which output is calculated
`datetime` scalar

Time at which the output is calculated, specified as a `datetime` scalar. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

Outputs Arguments

s — Access analysis status
column vector | matrix

Access analysis status, returned as a column vector or a matrix. If `timeIn` is specified, `s` is a column vector. Otherwise, the output is a matrix. The rows of the matrix correspond to the access object in `ac`, and the columns correspond to the time sample. The status at a given instant is `1` (`true`) if access exists between each pair of adjacent nodes defined by `Sequence`. For example, in a given pair, defined by node 1 and node 2, node 1 has access to node 2 and node 2 has access to node 1:

- If a node is a satellite, then the satellite has access to the adjacent node when both nodes are in line of sight of each other.
- If a node is a ground station, then the ground station has access to the adjacent node when the elevation angle of the node with respect to the ground station is greater than or equal to the `MinElevationAngle` property of `GroundStation`.
- If a node is a conical sensor, then the conical sensor has access to the adjacent node when the latter is in the field of view of the former. If the conical sensor is attached to a ground station directly or via a gimbal, then the elevation angle of the adjacent node with respect to the ground station must be greater than or equal to the `MinElevationAngle` property of `GroundStation`.

`timeOut` — Time samples of output access status
scalar | vector

Time samples of the output access status, returned as a scalar or vector. If the time history of the access status is returned, `timeOut` is a row vector.

Note When the `AutoSimulate` property of the satellite scenario is `true`, the access status history from `StartTime` to `StopTime` is returned. When the property is `false`, the access status history from `StartTime` to `SimulationTime` is returned.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

states

Package: `matlabshared.satellitescenario`

Obtain position and velocity of satellite

Syntax

```
pos = states(sat)
[pos,velocity] = states(sat)
[___] = states(sat,timeIn)
[___] = states(___,CoordinateFrame=C)
[pos,velocity,timeOut] = states(___)
```

Description

`pos = states(sat)` returns a 3-by-*n*-by-*m* array of the position history `pos` of each satellite in the vector `sat`, where *n* is the number of time samples and *m* is the number of satellites. The rows represent the *x*, *y*, and *z* coordinates of the satellite in the Geocentric Celestial Reference Frame (GCRF).

`[pos,velocity] = states(sat)` returns a 3-by-*n*-by-*m* array of the inertial velocity `velocity` of each satellite in the vector `sat` in the GCRF.

`[___] = states(sat,timeIn)` returns one or both of the outputs as 3-by-1-by-*m* arrays in addition to the position at the specified datetime `timeIn`. If no time zone is specified in `timeIn`, the time zone is assumed to be Coordinated Universal Time (UTC).

`[___] = states(___,CoordinateFrame=C)` returns the outputs in the coordinates specified by `C`.

`[pos,velocity,timeOut] = states(___)` returns the position and velocity history of the satellites and the corresponding datetime in UTC.

Examples

Obtain States of Satellite in ECEF Frame

Create a satellite scenario object.

```
startTime = datetime(2021,5,25);                    % May 25, 2021, 12:00 AM UTC
stopTime = datetime(2021,5,26);                     % May 26, 2021, 12:00 AM UTC
sampleTime = 60;                                    % In seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
tleFile = "eccentricOrbitSatellite.tle";
sat = satellite(sc,tleFile);
```

Obtain the position and velocity of the satellite in the Earth-centered Earth-fixed (ECEF) frame corresponding to May 25, 2021, 10:30 PM UTC.

```
time = datetime(2021,5,25,22,30,0);
[position,velocity] = states(sat,time,"CoordinateFrame","ecef")
```

```
position = 3×1
10^7 ×

   -0.9431
   -3.0675
    2.7404


velocity = 3×1
10^3 ×

   -1.2166
    0.4198
   -1.6730
```

Input Arguments

sat — Satellite
row vector of `Satellite` objects

Satellite, specified as a row vector of `Satellite` objects.

`timeIn` — Time at which output is calculated
`datetime` scalar

Time at which the output is calculated, specified as a `datetime` scalar. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

`C` — Coordinate frame
`'inertial'` | `'ecef'` | `'geographic'`

Coordinate frame in which the outputs are returned, specified as `'inertial'`, `'ecef'`, or `'geographic'`.

- The `'inertial'` option returns the position and velocity coordinates in the GCRF.
- The `'ecef'` option returns the position and velocity coordinates in the ECEF frame. For more information on ECEF frames, see “Earth-Centered Earth-Fixed Coordinates”.
- The `'geographic'` option returns the position as [*lat*; *lon*; *altitude*], where *lat* and *lon* are latitude and longitude in degrees and altitude is the height above the surface of the Earth in meters. The velocity returned is in the North-East-Down (NED) frame.

Output Arguments

pos — Position history
3-by-*n*-by-*m* array

Position history of the satellites in meters, returned as a 3-by-*n*-by-*m* array in the GCRF. *n* is the number of time samples and *m* is the number of satellites.

When the `AutoSimulate` property of the satellite scenario is `true`, the position history from `StartTime` to `StopTime` is returned. Otherwise, the position history from `StartTime` to `SimulationTime` is returned.

`velocity` — Velocity history
3-by-*n*-by-*m* array

Velocity history of the satellites in meters per second, returned as a 3-by-*n*-by-*m* array in the GCRF. *n* is the number of time samples and *m* is the number of satellites.

`timeOut` — Time samples of position and velocity
scalar | vector

Time samples of the position and velocity of the satellites, returned as a scalar or vector. If time histories of the position and velocity of the satellite are returned, `timeOut` is a row vector.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`satellite` | `groundStation`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

gimbalAngles

Steering angles of gimbal

Syntax

```
az = gimbalAngles(gimbal)
[az,el] = gimbalAngles(gimbal)
[ ___ ] = gimbalAngles(gimbal,timeIn)
[az,el,timeOut] = gimbalAngles(gimbal)
```

Description

`az = gimbalAngles(gimbal)` returns an array of gimbal azimuth `az` histories of the gimbals defined in the vector `gimbal`.

`[az,el] = gimbalAngles(gimbal)` returns an array of gimbal azimuth `az`and gimbal elevation `el` in the vector `gimbal`.

`[ ___ ] = gimbalAngles(gimbal,timeIn)` returns column vectors of gimbal azimuth and gimbal elevation of gimbals defined in the vector `gimbal` at the specified time `timeIn`, depending on the specified output arguments.

`[az,el,timeOut] = gimbalAngles(gimbal)` returns gimbal azimuth, gimbal elevation, and corresponding time in UTC.

Examples

Retrieve Gimbal Angles at Specific Time

Create a satellite scenario object.

```
startTime = datetime(2020,10,10);                   % 10 October 2020, 12:00 AM UTC
stopTime = datetime(2020,10,11);                    % 11 October 2020, 12:00 AM UTC
sampleTime = 60;                                    % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                           % meters
eccentricity = 0;
inclination = 10;                                   % degrees
rightAscensionOfAscendingNode = 0;                  % degrees
argumentOfPeriapsis = 0;                            % degrees
trueAnomaly = 0;                                    % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a gimbal to the satellite.

```
g = gimbal(sat);
```

Point the gimbal at 0 degree latitude and longitude.

```
pointAt(g,[0; 0; 0]);
```

Get the gimbal azimuth and gimbal elevation corresponding to October 10, 2020, 20:54 PM UTC.

```
time = datetime(2020,10,10,20,54,0);
[az,el] = gimbalAngles(g,time)
```

```
az = -5.4268
```

```
el = 19.0368
```

Input Arguments

gimbal — Gimbal
scalar | vector

Gimbal object whose steering angle is being calculated, specified as either a scalar or a vector.

timeIn — Time at which output is calculated
`datetime` scalar

Time at which the output is calculated, specified as a `datetime` scalar. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

Output Arguments

az — Gimbal azimuth
array

Gimbal azimuth histories of gimbals in degrees, returned as an array in the range [-180,180]. Each row corresponds to a gimbal in `gimbal`, and each column corresponds to a time sample. This represents the angle of rotation of the gimbal about its *y*-axis.

If `AutoSimulate` of the satellite scenario is `true`, `az` returns the gimbal azimuth history from `StartTime` to `StopTime`. Otherwise the gimbal azimuth history is returned from `StartTime` to `SimulationStatus`.

el — Gimbal elevation
array

Gimbal elevation histories of gimbals in degree, returned as an array in the range [0,180]. This represents the angle of rotation of the gimbal about its *y*-axis. Each row corresponds to a gimbal in `gimbal`, and each column corresponds to a time sample. This represents the angle of rotation of the gimbal about its x-axis.

If `AutoSimulate` of the satellite scenario is `true`, `el` returns the gimbal elevation history from `StartTime` to `StopTime`. Otherwise the gimbal elevation history is returned from `StartTime` to `SimulationStatus`.

timeOut — Time samples between start and stop time of scenario
scalar | vector

Time samples between start and stop time of the scenario, returned as a scalar or vector. If `az` and `el` histories are returned, `timeOut` is a row vector.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

show

Package: `matlabshared.satellitescenario`

Show object in satellite scenario viewer

Syntax

```
show(item)
show(item,viewer)
```

Description

`show(item)` shows the item on all open Satellite Scenario Viewers.

`show(item,viewer)` shows the graphic on the Satellite Scenario Viewer specified by `v`.

Examples

Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000;15000000];
eccentricity = [0.01;0.02];
inclination = [0;10];
rightAscensionOfAscendingNode = [0;15];
argumentOfPeriapsis = [0;30];
trueAnomaly = [0;20];

sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  1x2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

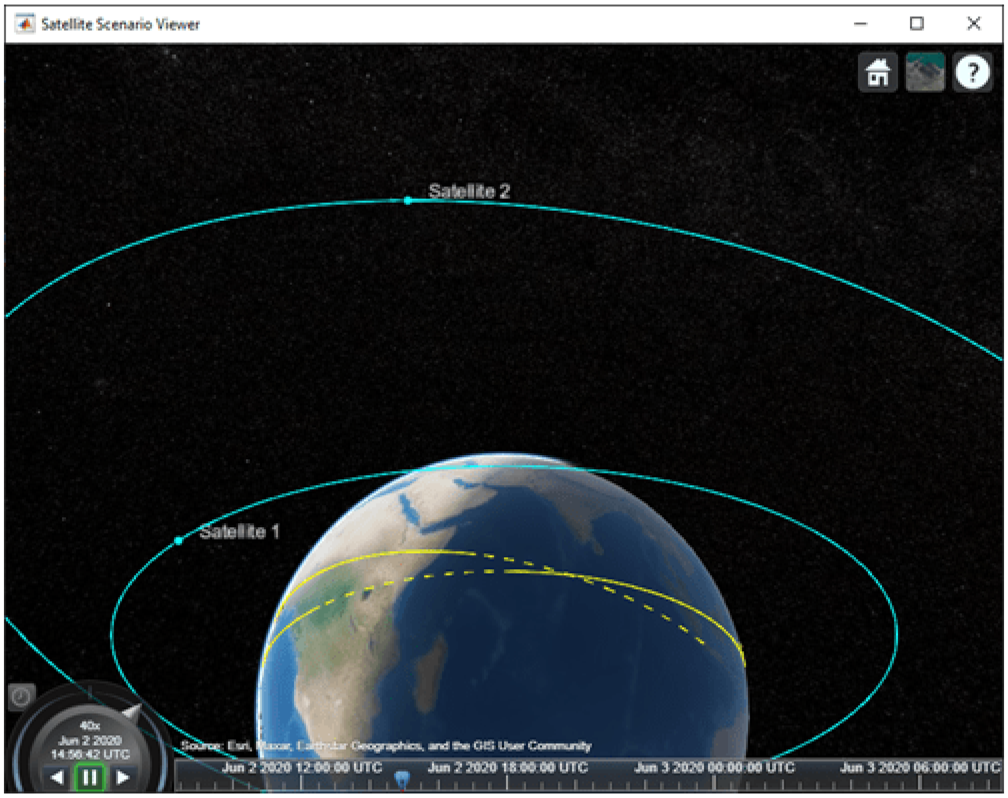
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,LeadTime=3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    LeadLineColor
    TrailLineColor
    VisibilityMode
```

Play the scenario and set the animation speed of the simulation to 40.

```
play(sc,PlaybackSpeedMultiplier=40)
```



Input Arguments

item — Item
Satellite object | GroundStation object | ConicalSensor object | GroundTrack object | FieldofView object | Access object | Link object

The item, specified as a Satellite, GroundStation, ConicalSensors, GroundTrack, FieldOfView, Access, or Link object. These objects must belong to the same satelliteScenario object.

Note If item is a satellite or a ground station, then the associated transmitters, receivers and gimbals are also displayed on the viewer.

viewer — Satellite scenario viewer
row vector of all satelliteScenarioViewer objects (default) | scalar satelliteScenarioViewer object | array of satelliteScenarioViewer objects

Satellite scenario viewer, specified as a scalar, vector, or array of satelliteScenarioViewer objects.

Version History

Introduced in R2021a

See Also

Objects
satelliteScenario | satelliteScenarioViewer

Functions
play | hide | access | groundStation | conicalSensor | transmitter | receiver

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

hide

Package: `matlabshared.satellitescenario`

Hide satellite scenario entity from viewer

Syntax

```
hide(item)
hide(item,viewer)
```

Description

`hide(item)` hides the specified satellite scenario entity from all open instances of the Satellite Scenario Viewer.

`hide(item,viewer)` hides the specified satellite scenario entity on the Satellite Scenario Viewer specified by `viewer`.

Examples

Hide Satellite from Satellite Scenario Viewer

Create a satellite scenario object.

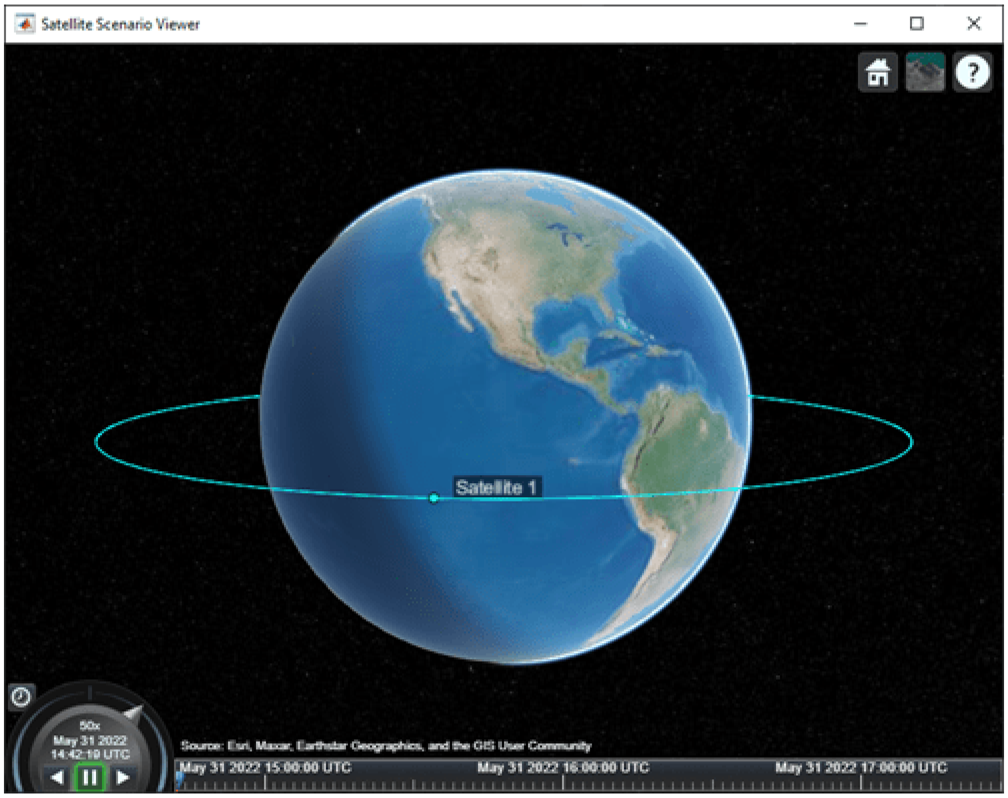
```
sc = satelliteScenario;
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                          % meters
eccentricity = 0;
inclination = 0;                                   % degrees
rightAscensionOfAscendingNode = 0;                 % degrees
argumentOfPeriapsis = 0;                           % degrees
trueAnomaly = 0;                                   % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
      inclination,rightAscensionOfAscendingNode, ...
      argumentOfPeriapsis,trueAnomaly);
```

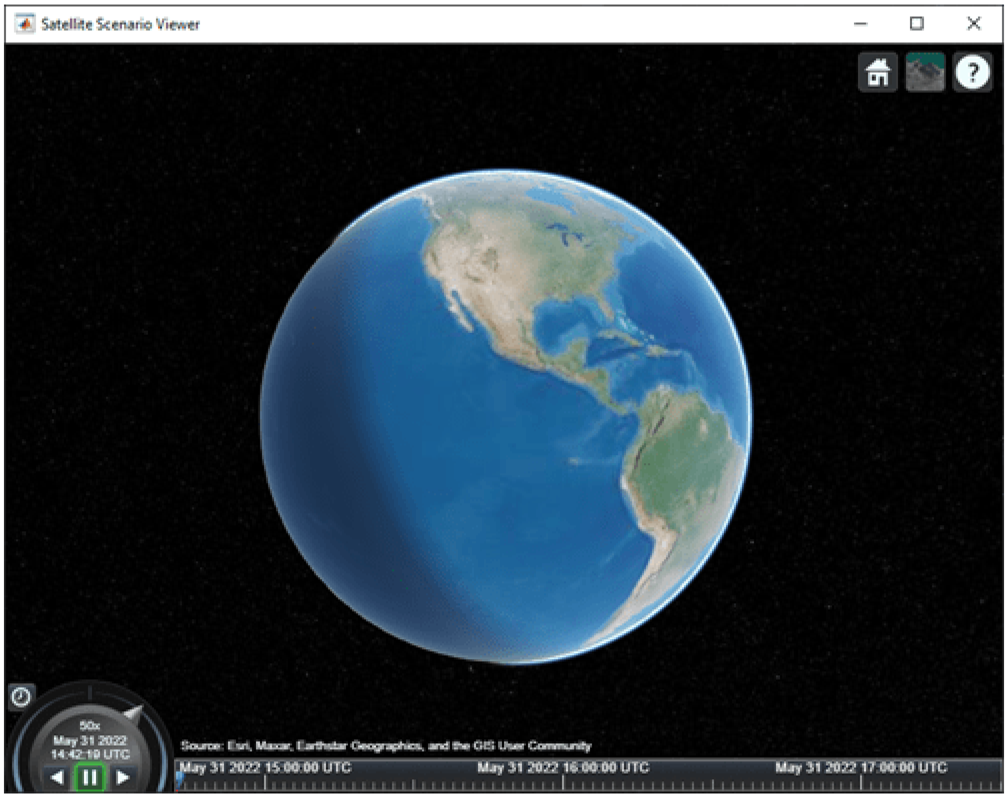
Visualize the satellite by using the Satellite Scenario Viewer.

```
viewer = satelliteScenarioViewer(sc);
```



Hide the satellite from the viewer.

```
hide(sat,viewer);
```



Input Arguments

`item` — Item
`Satellite` object | `GroundStation` object | `ConicalSensor` object | `GroundTrack` object | `FieldofView` object | `Access` object | `Link` object

The item, specified as a `Satellite`, `GroundStation`, `ConicalSensors`, `GroundTrack`, `FieldOfView`, `Access`, or `Link` object. These objects must belong to the same `satelliteScenario` object.

`viewer` — Satellite Scenario Viewer
row vector of all `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite Scenario Viewer, specified as a scalar `satelliteScenarioViewer` object or a vector or array of `satelliteScenarioViewer` objects.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `hideAll` | `showAll`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

ebno

Package: `satcom.satellitescenario`

Eb/No at final node of link

Syntax

```
e = ebno(lnk)
e = ebno(lnk,timeIn)
[e,timeOut] = ebno( ___ )
```

Description

`e = ebno(lnk)` returns a matrix `e` of the history of received values for energy per bit to noise power spectral density (Eb/No) in dB at the final node in each possible multihop link in the vector `lnk`. The rows of the matrix correspond to the link object in `lnk` and the columns correspond to the time sample.

`e = ebno(lnk,timeIn)` returns a column vector of Eb/No `e` in dB at the final node in each link defined in the vector `lnk` at the specified datetime `timein`. Each element of `e` corresponds to a link in `lnk`. If no time zone is specified in `timeIn`, the time zone is assumed to be Coordinated Universal Time(UTC).

`[e,timeOut] = ebno( ___ )` returns the received Eb/No values and the corresponding times in UTC.

Examples

Retrieve Time Samples and Eb/No of Reciever

Create a satellite scenario object.

```
startTime = datetime(2021,12,12,10,42,0);
stopTime = startTime + days(1);
sampleTime = 5000;                                      % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                              % meters
eccentricity = 0;
inclination = 0;                                       % degrees
rightAscensionOfAscendingNode = 0;                     % degrees
argumentOfPeriapsis = 0;                               % degrees
trueAnomaly = 210;                                     % degrees
sat = satellite(sc,semiMajorAxis,eccentricity, ...
    inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly);
```

Add a transmitter to the satellite.

```
tx = transmitter(sat);
```

Add a ground station to the scenario.

```
latitude = 0;                                % degrees
longitude = 30;                              % degrees
gs = groundStation(sc,latitude,longitude);
```

Add a receiver to the ground station.

```
rx = receiver(gs,MountingAngles=[0; 180; 0]);
```

Add link analysis to the transmitter.

```
lnk = link(tx,rx);
```

Get the Eb/No history at the receiver and the time samples.

```
[e,t] = ebno(lnk)
```

```
e = 1×19

  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf  -Inf


t = 1x19 datetime
   12-Dec-2021 10:42:00   12-Dec-2021 12:05:20   12-Dec-2021 13:28:40   12-Dec-2021 14:52:00   12
```

Compute Pointing Loss of Satellite and Ground Station

This example demonstrates a simple technique for incorporating random pointing loss at the transmit and receive antennas in a space-to-ground radio link. Caluclate the link margin without pointing loss, and then you add the random sequences of pointing loss to the result.

Define parameters for pointing error simulation.

```
freq = 8;                   % Carrier frequency, GHz
satAntDiam = 0.4;           % Satellite antenna diameter, m
satAntEff = 0.43;           % Satellite antenna efficiency, (0,1]
satAntPtErr = 0.5;          % Satellite antenna std. dev. pointing error, deg
gsAntDiam = 2.7;            % Ground antenna diameter, m
gsAntEff = 0.63;            % Ground antenna efficiency, (0,1]
gsAntPtErr = 0.2;           % Ground antenna std. dev. pointing error, deg
gsNoiseTemp = 150;          % Ground system noise temperature, Kelvin
reqEbNo = 13.0;             % Required Eb/No for 8PSK BER=1e-5, decibels
```

Create a satellite scenario object.

```
startTime = datetime(2020,8,19,22,35,0);
stopTime = datetime(2020,8,19,23,15,0);
sampleTime = 1;                                        % Seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add satellite orbital elements for a sun-synchronous satellite.

```
sat = satellite(sc,7*1e6,0.0002178,97.8356,359.1795,135.7406,224.3991);
satTx = transmitter(sat,Frequency=freq*1e9,BitRate=60,Power=3);
satAnt = gaussianAntenna(satTx,DishDiameter=satAntDiam, ...
    ApertureEfficiency=satAntEff);

gs = groundStation(sc,Latitude=69.649208,Longitude=18.955324, ...
    Altitude=0.35,MinElevationAngle=5.0);
gsGim = gimbal(gs,MountingAngles=[0;180;0], ...
    MountingLocation=[0;0;-5]);
waveLen = physconst("lightspeed")/(freq*1e9);
pkG = 10*log10(gsAntEff*((pi*gsAntDiam)/waveLen)^2);                % Peak antenna gain, dBi
gsRx = receiver(gsGim,MountingLocation=[0;0;1], ...
    GainToNoiseTemperatureRatio=pkG-10*log10(gsNoiseTemp), ...
    RequiredEbNo=reqEbNo);

pointAt(sat,gs);
pointAt(gsGim,sat);
lnk = link(satTx,gsRx);
[e, time] = ebno(lnk);
```

Generate a random sequence of pointing losses for transmit and receiver antennas.

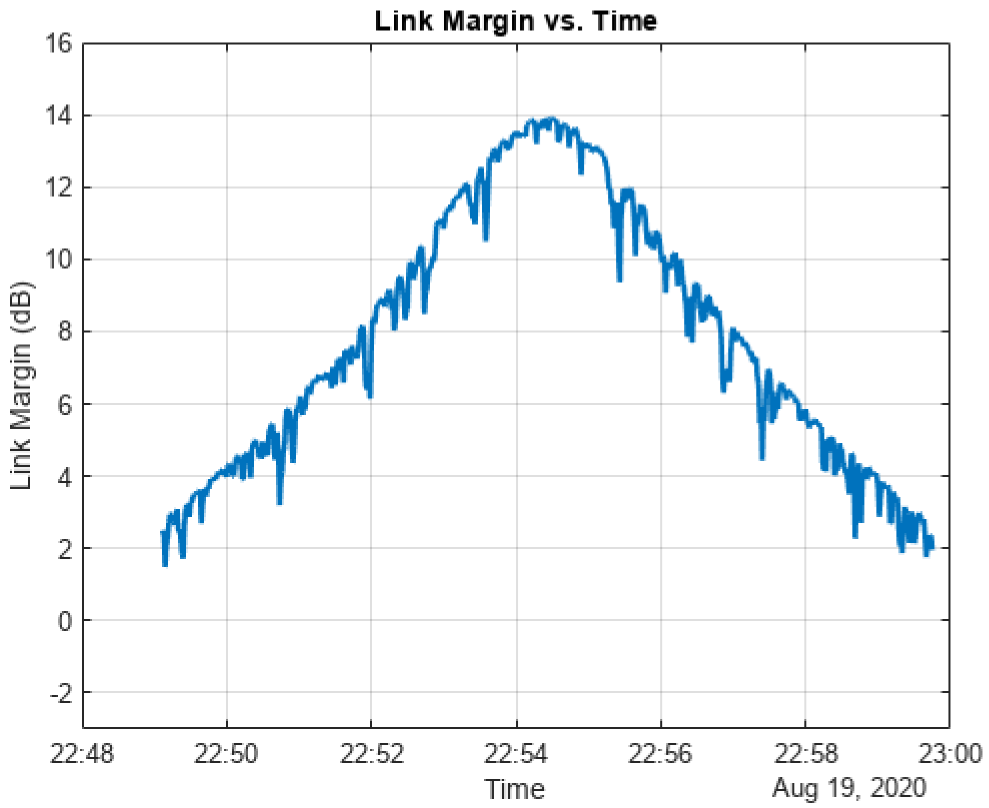
```
satAntPtErrRad = deg2rad(satAntPtErr);
satRanAngle = (satAntPtErrRad)*randn(1,length(time));          % Random pointing angle error, ra
satRanAngle = filter(ones(1,10)/sqrt(10),1,satRanAngle);       % Model control loop filtering
satBeamWidth = (waveLen/(satAntDiam*sqrt(satAntEff)));         % 3 dB beamwidth, radians
satPeakGain = 10*log10(satAntEff*((pi*satAntDiam)/waveLen)^2);  % Peak gain, dBi
satRanAntGain = 10*log10(satAntEff*(((pi.*satAntDiam)./ ...
    waveLen).^2).*exp(-2.76*((satRanAngle./satBeamWidth).^2))); % Antenna gain at off-boresight a
                                                                % from Satellite Communications,
satRanPtLoss = satPeakGain - satRanAntGain;                     % Random pointing loss, dB

gsAntPtErrRad = deg2rad(gsAntPtErr);
gsRanAngle = (gsAntPtErrRad)*randn(1,length(time));             % Random pointing angle error, ra
gsRanAngle = filter(ones(1,10)/sqrt(10),1,gsRanAngle);          % Model control loop filtering
gsBeamWidth = waveLen/(gsAntDiam*sqrt(gsAntEff));               % 3 dB beamwidth, radians
gsPeakGain = 10*log10(gsAntEff*((pi*gsAntDiam)/waveLen)^2);     % Peak gain, dBi
gsRanAntGain = 10*log10(gsAntEff*(((pi.*gsAntDiam)./ ...
    waveLen).^2).*exp(-2.76*((gsRanAngle./gsBeamWidth).^2)));   % Antenna gain at off-boresight a
                                                                % from Satellite Communications,
gsRanPtLoss = gsPeakGain - gsRanAntGain;                        % Random pointing loss, dB
```

Compute composite pointing losses.

```
totRanPtLoss = satRanPtLoss + gsRanPtLoss;
```

Compute the link margin.

```
margin = e-totRanPtLoss-gsRx.RequiredEbNo;
plot(time,margin,LineWidth=2);
title("Link Margin vs. Time");
ylim([-3 16]);
xlabel("Time");
ylabel("Link Margin (dB)");
grid on;
```



Input Arguments

`lnk` — Link analysis
`Link` object vector | `Link` object scalar

Link analysis object, specified as a `Link` object vector or scalar.

`timeIn` — Time at which output is calculated
`datetime` scalar

Time at which the output is calculated, specified as a `datetime` scalar. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

Output Arguments

`e` — Eb/No
scalar | vector | matrix

Energy per bit to noise power spectral density (Eb/No), returned as a scalar, vector, or matrix. If `timeIn` is not specified, `e` is a row vector or a matrix.

`timeOut` — Time samples of output Eb/No
scalar | vector

Time samples of the output Eb/No, returned as a scalar or vector. If time history of Eb/No is returned, `timeOut` is a row vector.

Note When the `AutoSimulate` property of the satellite scenario is `true`, the Eb/No history from `StartTime` to `StopTime` is returned. When the property is `false`, the Eb/No history from `StartTime` to `SimulationTime` is returned.

Algorithms

The Eb/No in dB is calculated as

EbNo = *txPower* + *txAntennaGain* - *txSystemLoss* - *pathloss* + *rxAntGaintoNoiseTempRatio* - 10*log10(*K*) - *rxSystemLoss* - 10*log10(*bitRate*) - 60

where:

- *txPower* is the transmitter power in dBW.
- *txAntennaGain* is the transmitter antenna gain in dB.
- *txSystemLoss* is the transmitter system loss in dB.
- *pathloss* is the path loss in dB.
- *rxAntGaintoNoiseTempRatio* is the receiver antenna gain to noise temperature ratio.
- *K* is the Boltzmann constant.
- *rxSystemLoss* is the receiver system loss in dB.
- *bitRate* is the bit rate in Mbps.

Version History

Introduced in R2021a

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer` | `Link`

Functions

`show` | `play` | `hide`

Topics

"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

access

Package: `matlabshared.satellitescenario`

Add access analysis objects to satellite scenario

Syntax

```
access(asset1,asset2,...)
ac = access( ___ ,'Viewer',Viewer)
ac = access( ___ )
```

Description

`access(asset1,asset2,...)` adds `Access` analysis objects defined by nodes `asset1`, `asset2`, and so on.

`ac = access( ___ ,'Viewer',Viewer)` sets the viewer in addition to any input argument combination from previous syntaxes. For example, `'Viewer',v1` picks the viewer `v1`.

`ac = access( ___ )` returns added access analysis objects in the row vector `ac`.

Examples

Add Ground Stations to Scenario and Visualize Access Intervals

Create a satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = 10;
lon = -30;
gs = groundStation(sc,lat,lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

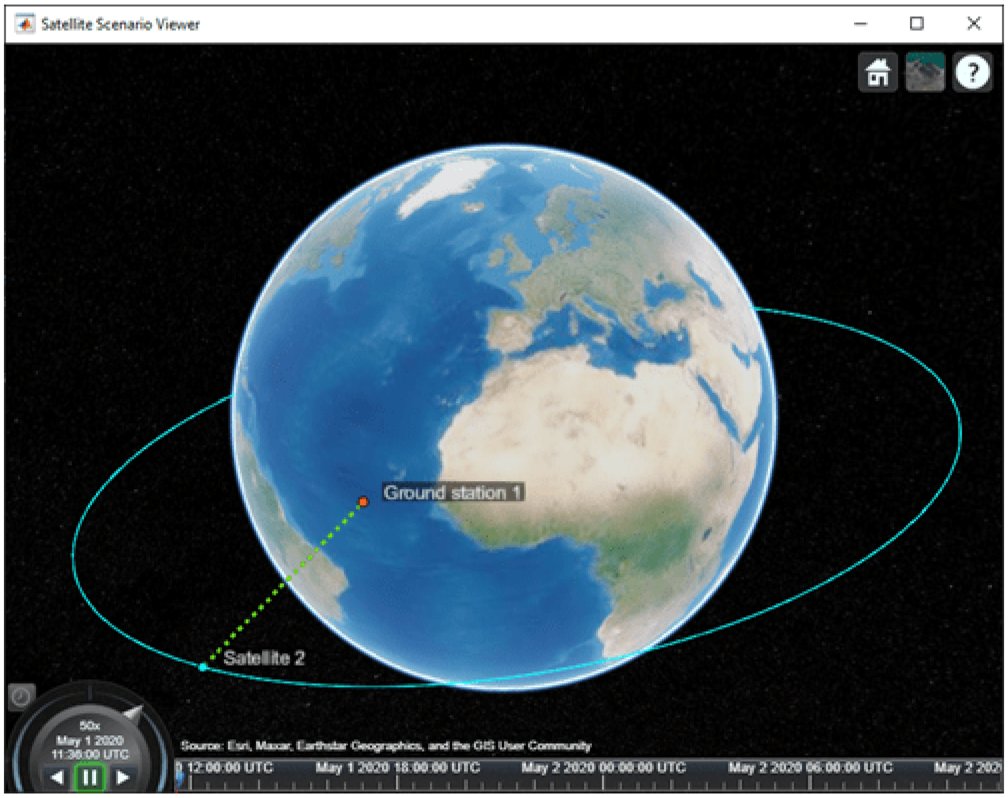
Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat,gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
        Source              Target            IntervalNumber         StartTime                EndTi
    ______________    ____________________    _______________    ______________________    ____________

    "Satellite 2"    "Ground station 1"           1            01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"           2            01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"           3            01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"           4            01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"           5            01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"           6            02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"           7            02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"           8            02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Input Arguments

`asset1,asset2,...` — Satellite, ground station, or conical sensor
scalar | vector

`Satellite`, `GroundStation`, or `ConicalSensors` object, specified as a scalar or vector. These objects must belong to the same `satelliteScenario` object. The function adds the access analysis object to the `Accesses` property of the corresponding asset in `asset1`.

- If the asset in a given node is a scalar, every analysis object uses the same asset for that node position.
- If the asset in a given node is a vector, its length must equal the number of access analysis objects. Each access analysis object uses the corresponding element of the asset vector for that node location.

`Viewer` — Satellite scenario viewer
vector of `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a scalar, vector, or array of `satelliteScenarioViewer` objects. If the `AutoSimulate` property of the scenario is `false`, adding a satellite to the scenario disables any previously available timeline and playback widgets.

Output Arguments

ac — Access analysis
scalar | vector

Access analysis between input objects, returned as either a scalar or vector. For more information about access analysis, see the `Access` object.

Data Types: `matlabshared.satellitescenario.Access`

Note When the `AutoSimulate` property is set to `false`, `SimulationStatus` must be `NotStarted` to call `access` function. Otherwise, use the `restart` function to reset the `SimulationStatus` to `NotStarted`. Note that `restart` also erases the simulation data.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

groundStation

Package: `matlabshared.satellitescenario`

Add ground station to satellite scenario

Syntax

```
groundStation(scenario)
groundStation(scenario,lat,lon)
groundStation(___,Name,Value)
gs = groundStation(___)
```

Description

`groundStation(scenario)` adds a default `GroundStation` object to the specified satellite scenario.

`groundStation(scenario,lat,lon)` sets the Latitude and Longitude properties of the ground station to `lat` and `lon`, respectively. `lat` and `lon` must be of the same length. This length specifies the number of ground stations that the function adds to the input scenario. Together, `lat` and `lon` indicate the locations of the ground stations.

`groundStation(___,Name,Value)` sets options using one or more name-value arguments in addition to any input argument combination from previous syntaxes. For example, `'MinElevationAngle',10` specifies a minimum elevation angle of 10 degrees.

`gs = groundStation(___)` returns a vector of handles to the added ground stations. Specify any input argument combination from previous syntaxes.

Examples

Add Ground Stations to Scenario and Visualize Access Intervals

Create a satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = 10;
lon = -30;
gs = groundStation(sc,lat,lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
```

```
trueAnomaly = 0;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

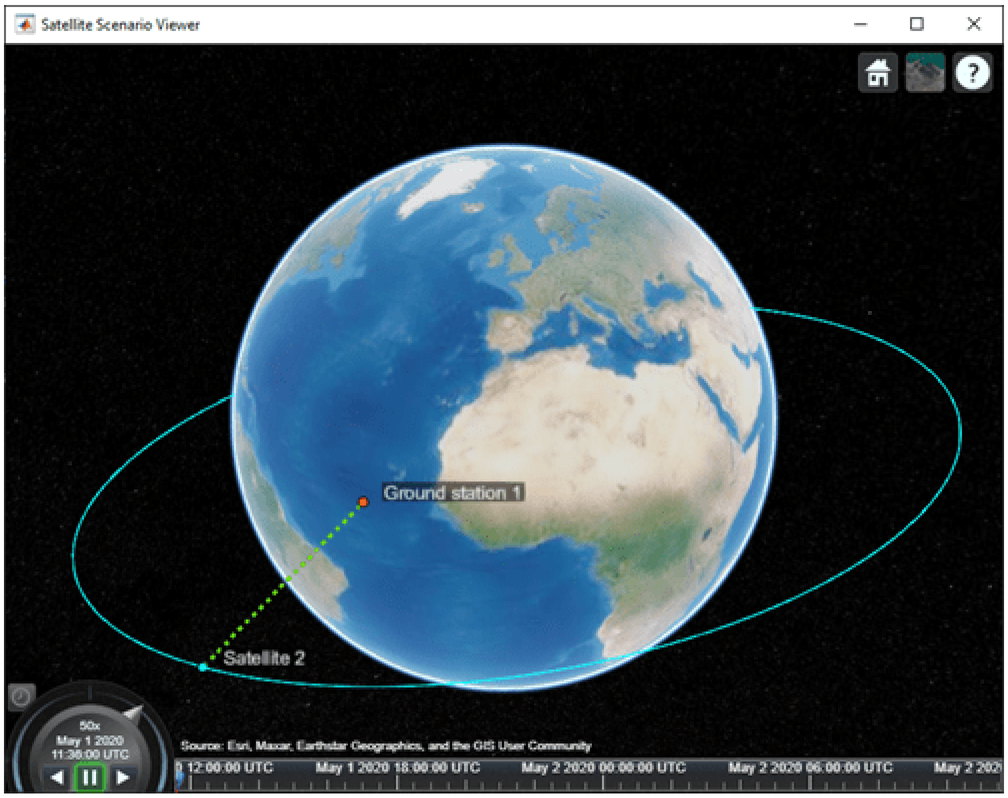
Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat,gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source              Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    __________

    "Satellite 2"    "Ground station 1"           1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"           2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"           3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"           4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"           5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"           6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"           7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"           8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Input Arguments

`scenario` — Satellite scenario
`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object.

`lat`, `lon` — Latitude and longitude
real-valued scalar | real-valued vector

Latitude and longitude of the ground station, specified as a real-valued scalar or real-valued vector.

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: `'MinElevationAngle',10` specifies a minimum elevation angle of 10 degrees.

`Viewer` — Satellite scenario viewer
vector of `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a scalar, vector, or array of `satelliteScenarioViewer` objects. If the `AutoSimulate` property of the scenario is `false`, adding a satellite to the scenario disables any previously available timeline and playback widgets.

`Name` — groundStation name
`"groundStation *idx*"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `groundStation` function. After you call `groundStation` function, this property is read-only.

groundStation name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one groundStation is added, specify `Name` as a string scalar or a character vector.
- If multiple groundStations are added, specify `Name` as a string scalar, character vector, string vector or a cell array of character vectors. All groundStations added as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vector must equal the number of groundStations being added. Each groundStation is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by satellite scenario.

Data Types: `char` | `string`

`Latitude` — Geodetic latitude of ground stations
`42.3001` (default) | scalar | row vector

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

Geodetic latitude of ground stations, specified as a scalar. Values must be in the range [-90, 90].

- If you add only one ground station, specify Latitude as a scalar double.
- If you add multiple ground stations, specify Latitude as a vector double whose length is equal to the number of ground stations being added.

When latitude and longitude are specified as `lat, lon` inputs to groundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`Longitude` — Geodetic longitude of ground stations
`-71.3504` (default) | scalar | row vector

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

Geodetic longitude of ground stations, specified as a scalar or a vector. Values must be in the range [-180, 180].

- If you add only one ground station, specify longitude as a scalar.
- If you add multiple ground stations, specify longitude as a vector whose length is equal to the number of ground stations being added.

When longitude and longitude are specified as `lat, lon` inputs to groundStation, longitude specified as a name-value argument takes precedence.

Data Types: `double`

`Altitude` — Altitude of ground station
`0` m (default) | scalar | vector

You can set this property only when calling groundStation. After you call groundStation, this property is read-only.

Altitude of ground stations, specified as a scalar or a vector.

- If you specify `Altitude` as a scalar, the value is assigned to each ground station in the groundStation.
- If you specify `Altitude` as a vector, the vector length must be equal to the number of ground stations in the groundStation.

When latitude and longitude are specified as `lat, lon` inputs to groundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`MinElevationAngle` — Minimum elevation angle
`0` (default) | scalar | vector

Minimum elevation angle of a satellite for the satellite to be visible from the ground station, and for the ground station to be visible from the satellite in degrees, specified as a scalar or row vector. Values must be in the range [–90, 90]. For access and link closure to be possible, the elevation angle must be at least equal to the value specified in `MinElevationAngle`.

- If you specify `MinElevationAngle` as a scalar, the value is assigned to each ground station in the groundStation.
- If you specify `MinElevationAngle` as a vector, the vector length must be equal to the number of ground stations in the groundStation.

When the `AutoSimulate` property of the satellite scenario is `false`, `MinElevationAngle` property can be modified while the `SimulationStatus` is `NotStarted` or `InProgress`.

Data Types: `double`

Output Arguments

gs — Ground station in scenario
`GroundStation` object

Ground station in the scenario, returned as a `GroundStation` object belonging to the satellite scenario specified by the input `scenario`.

You can modify the `GroundStation` object by changing its property values. The name-value arguments used when calling this function correspond to property names.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `satellite` | `access` | `transmitter` | `receiver`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Constellation Access to Ground Station”
“Comparison of Orbit Propagators”
“Modeling Satellite Constellations Using Ephemeris Data”
“Estimate GNSS Receiver Position with Simulated Satellite Constellations”
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

transmitter

Package: `matlabshared.satellitescenario`

Add transmitter to satellite scenario

Syntax

```
transmitter(parent)
transmitter(parent,Name=Value)
tx = transmitter( ___ )
```

Description

`transmitter(parent)` adds a `Transmitter` object to the `parent` which can be a `Satellites`, `GroundStations`, or `Gimbals`.

`transmitter(parent,Name=Value)` adds transmitters to parents in `parent` using additional parameters specified by optional name-value arguments. For example, `MountingAngle=[20; 35; 10]` sets the yaw, pitch, and roll angles of the transmitter to 20, 35, and 10 degrees, respectively.

`tx = transmitter( ___ )` returns added transmitters as a row vector `tx`. Specify any input argument combination from previous syntaxes.

Note When the `AutoSimulate` property of the satellite scenario is `false`, you can call the `transmitter` function only when `SimulationStatus` is `NotStarted`. Otherwise, you must call the `restart` function to erase the simulation data and reset `SimulationStatus` to `NotStarted`.

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                   % mete
eccentricity = 0;
inclination = 60;                                                           % degr
rightAscensionOfAscendingNode = 0;                                          % degr
argumentOfPeriapsis = 0;                                                    % degr
trueAnomaly = 0;                                                            % degr
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                          % dB/K
systemLoss = 3;                                                           % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)
```

```
rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                  RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                       % Hz
power = 20;                                                             % dBW
bitRate = 20;                                                           % Mbps
systemLoss = 3;                                                         % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)
```

```
txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
```

```
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                             % meters
apertureEfficiency = 0.5;
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                                         % degrees
longitude = 0.1487094;                                         % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                   % Hz
power = 40;                                                                         % dBW
bitRate = 20;                                                                       % Mbp
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                              % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                               % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
lnk =
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

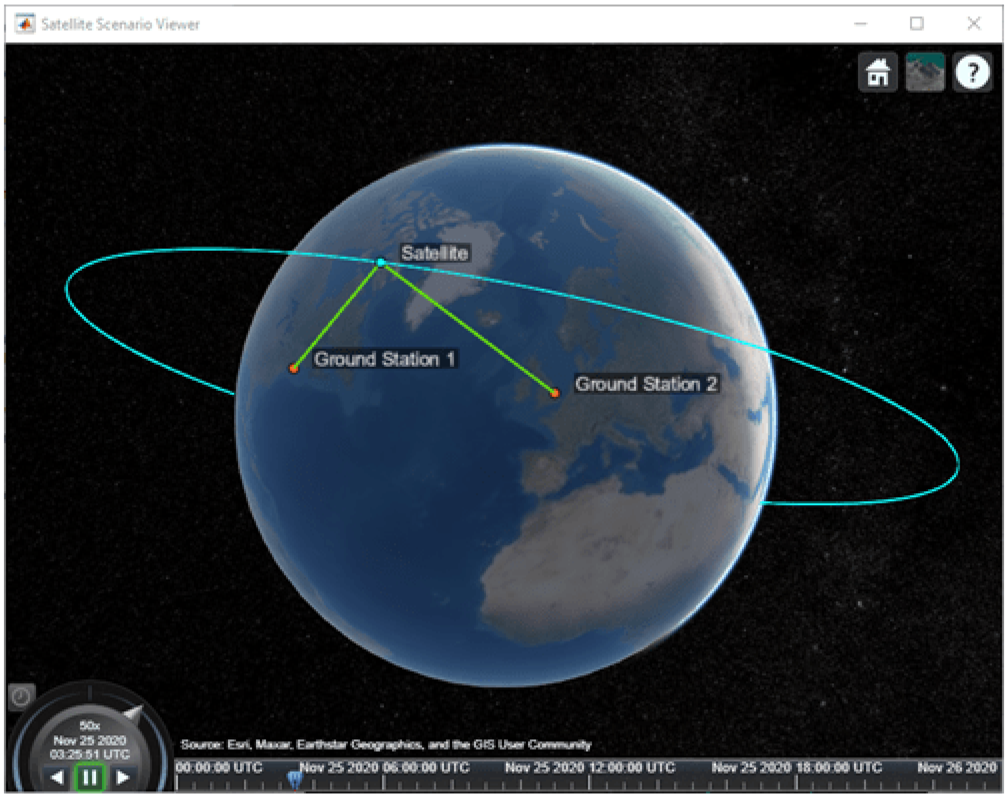
Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)
```

```
ans=4×8 table
               Source                              Target                  IntervalNumber        Star
    ______________________________    ____________________________    ______________    _________

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           1           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           2           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           3           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           4           25-Nov-202
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Input Arguments

`parent` — Element of scenario to which transmitter is added
scalar | vector

Element of scenario to which the transmitter is added, specified as a scalar or vector of satellites, ground stations or gimbals. The number of transmitters specified is determined by the size of the inputs.

- If `parent` is a scalar, all transmitters are added to the parent.
- If `parent` is a vector and the number of transmitters specified is one, that transmitter is added to each parent.
- If `parent` is a vector and the number of transmitters specified is more than one, the number of transmitters must equal the number of `parents` and each `parent` gets one transmitter.

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.

Example: `MountingAngle=[20; 35; 10]` sets the yaw, pitch, and roll angles of the transmitter to 20, 35, and 10 degrees, respectively.

`Name` — Transmitter name
`"Transmitter *idx*"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `transmitter` function. After you call the `transmitter` function, this property is read-only.

Transmitter name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one transmitter, specify `Name` as a string scalar or a character vector.
- If you are adding multiple transmitters, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All transmitters that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of transmitters that you are adding. Each transmitter is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one transmitter, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of transmitter.

- If you are adding multiple transmitters, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified transmitters. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the transmitters. The columns correspond to the mounting location of each specified transmitter and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one transmitter, the `MountingAngles` property is a three-element vector.
- If you are adding multiple transmitters the `MountingAngles` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified transmitters. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the transmitters. The columns correspond to the mounting angles of each specified transmitter and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

`Antenna` — Antenna object associated with transmitter
scalar | vector

`Antenna` object associated with the transmitter, specified as either a scalar or a vector. This object can be the default `gaussianAntenna` object, or one from the Antenna Toolbox or Phased Array System Toolbox. The default Gaussian antenna has a dish diameter of 1 meter and an aperture efficiency of 0.65.

Antenna can be specified in transmitter as a name-value argument consisting of `'Antenna'` and a scalar, antenna or phased array objects.

- If you are adding only one transmitter, `Antenna` must be a scalar.
- If you are adding multiple transmitters, `Antenna` is a vector. The same antenna is assigned to all transmitters.

`SystemLoss` — Total system loss in the transmitter
5 (default) | scalar | vector

Total system loss in the transmitter in dB, specified as a scalar or a vector.

System loss can be specified in transmitter as a name-value argument consisting of `'SystemLoss'` and a scalar, or vector.

- If you are adding only one transmitter, specify `SystemLoss` as a scalar.
- If you are adding multiple transmitters are added, specify `SystemLoss` as a scalar or a vector. When `SystemLoss` is a scalar, the same `SystemLoss` is assigned to all transmitters. When `SystemLoss` is a vector, its length must equal the number of transmitter and each element of `SystemLoss` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `SystemLoss` value while `SimulationStatus` is `NotStarted` or `InProgress`.

`Frequency` — Transmitter frequency
`14e9` (default) | scalar | vector

Transmitter frequency in Hz, specified as a name-value argument consisting of 'Frequency' and a scalar or a vector.

- If you are adding only one transmitter, the `Frequency` must be a scalar.
- If you are adding multiple transmitters are added, the frequency value can be a scalar or a vector. All transmitters added as a scalar are assigned the same specified `Frequency`. The length of the vector must equal the number of transmitters added and each element of `Frequency` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is false, you can modify the `Frequency` value while `SimulationStatus` is `NotStarted` or `InProgress`.

`BitRate` — Bit rate of transmitter
`10` (default) | scalar | vector

Bit rate of the transmitter in Mbps, specified as a name-value pair consisting of 'BitRate' and a scalar or a vector.

- If you are adding only one transmitter, the bit rate value must be a scalar.
- If you are adding multiple transmitters, the bit rate value can be a scalar or a vector. All transmitters added as a scalar are assigned the same specified `BitRate`. The length of the vector must equal the number of transmitters added and each element of `BitRate` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `BitRate` value while `SimulationStatus` is `NotStarted` or `InProgress`.

`Power` — Power of high power amplifier
12 (default) | scalar | vector

Power of the high power amplifier in dbW, specified as a name-value pair consisting of 'Power' and a scalar or a vector.

- If you are adding only one transmitter, the power value must be a scalar.

- If you are adding multiple transmitters, the power value can be a scalar or a vector. All transmitters added as a scalar are assigned the same specified `Power`. The length of the vector must equal the number of transmitters added and each element of `Power` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is false, you can modify the `Power` value while `SimulationStatus` is `NotStarted` or `InProgress`.

Output Arguments

`tx` — Transmitter
row vector

Transmitters attached to `parent`, returned as a row vector.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`play` | `show` | `groundStation` | `access` | `link` | `receiver` | `hide` | `pattern` | `pointAt` | `pointAt`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

receiver

Package: `matlabshared.satellitescenario`

Add receiver to satellite scenario

Syntax

```
receiver(parent)
receiver(parent,Name=Value)
rx = receiver(___)
```

Description

`receiver(parent)` adds a `Receiver` object to the `parent` using default parameters. `parent` can be a `Satellites`, `GroundStations`, or `Gimbals`.

`receiver(parent,Name=Value)` adds receivers to parents in `parent` using additional parameters specified by optional name-value arguments. For example, `MountingAngle=[20; 35; 10]` sets the yaw, pitch, and roll angles of the transmitter to 20, 35, and 10 degrees, respectively.

`rx = receiver(___)` returns added receivers as a row vector `rx`. Specify any input argument combination from previous syntaxes.

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                        % mete
eccentricity = 0;
```

```
inclination = 60;                                                                     % deg
rightAscensionOfAscendingNode = 0;                                                    % deg
argumentOfPeriapsis = 0;                                                              % deg
trueAnomaly = 0;                                                                      % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                                    % dB/K
systemLoss = 3;                                                                     % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)
```

```
rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                    RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                                 % Hz
power = 20;                                                                       % dBW
bitRate = 20;                                                                     % Mbps
systemLoss = 3;                                                                   % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)
```

```
txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                                        % meters
apertureEfficiency = 0.5;
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                                                 % degrees
longitude = 0.1487094;                                                 % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                            % Hz
power = 40;                                                                                  % dBW
bitRate = 20;                                                                                % Mbp
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                                         % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                                    % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
```

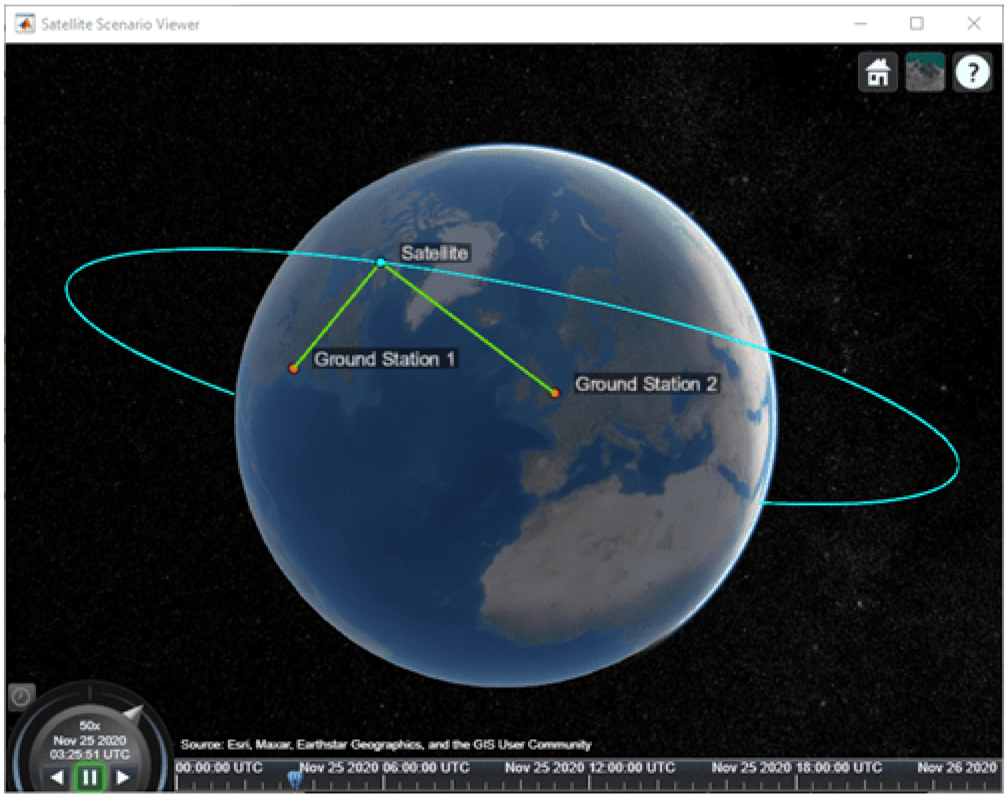
```
lnk = 
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)

ans=4×8 table
                  Source                             Target                 IntervalNumber          Star
    ________________________________    ______________________________    _______________    ________

    "Ground Station 1 Transmitter"      "Ground Station 2 Receiver"              1            25-Nov-202
    "Ground Station 1 Transmitter"      "Ground Station 2 Receiver"              2            25-Nov-202
    "Ground Station 1 Transmitter"      "Ground Station 2 Receiver"              3            25-Nov-202
    "Ground Station 1 Transmitter"      "Ground Station 2 Receiver"              4            25-Nov-202
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Input Arguments

parent — Element of scenario to which receiver is added
scalar | vector

Element of scenario to which the receiver is added, specified as a scalar or vector of satellites, ground stations or gimbals. The number of receivers specified is determined by the size of the inputs.

- If `parent` is a scalar, all receivers are added to the parent.
- If `parent` is a vector and the number of receivers specified is one, that receiver is added to each parent.

- If `parent` is a vector and the number of receivers specified is more than one, the number of receivers must equal the number of `parent`s and each `parent` gets one receiver.

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: `MountingAngle=[20; 35; 10]` sets the yaw, pitch, and roll angles of the receiver to 20, 35, and 10 degrees, respectively.

Name — Receiver name
`"Receiver` *`idx`*`"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `receiver` function. After you call the `receiver` function, this property is read-only.

Receiver name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one receiver, specify `Name` as a string scalar or a character vector.
- If you are adding multiple receivers, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All receivers that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of receivers that you are adding. Each receiver is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one receiver, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of receiver.
- If you are adding multiple receivers, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified receivers. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the receivers. The columns correspond to the mounting location of each specified receiver and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one receiver, the `MountingAngles` property is a three-element vector.
- If you are adding multiple receivers the `MountingAngles` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified receivers. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the receivers. The columns correspond to the mounting angles of each specified receiver and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

`Antenna` — Antenna object associated with receiver
scalar | vector

`Antenna` object associated with the receiver, specified as either a scalar or a vector. This object can be the default `gaussianAntenna` object, or one from the Antenna Toolbox or Phased Array System Toolbox. The default Gaussian antenna has a dish diameter of 1 meter and an aperture efficiency of 0.65.

Antenna can be specified in receiver as a name-value argument consisting of `'Antenna'` and a scalar, antenna or phased array objects.

- If you are adding only one receiver, `Antenna` must be a scalar.
- If you are adding multiple receivers, `Antenna` is a vector. The same antenna is assigned to all receivers.

`SystemLoss` — System loss in receiver
`5` (default) | scalar | vector

System loss in dB, specified as a scalar or a vector. `SystemLoss` must be greater than or equal to `PreReceiverLoss`.

System loss can be specified in `receiver` function as a name-value argument consisting of `'SystemLoss'` and a scalar, or a vector.

- If you are adding only one receiver, `SystemLoss` is a scalar.
- If you are adding multiple receivers, `SystemLoss` is a scalar or a vector. When `SystemLoss` is a scalar, the same `SystemLoss` is assigned to all receivers. When `SystemLoss` is a vector, its length must equal the number of receivers and each element of `SystemLoss` is assigned to the corresponding receivers in the parent.

If you specify PreReceiverLoss property as a name value argument in receiver function , the default value is the greater of 5 dB and the specified PreReceiverLoss value.

When the AutoSimulate property of the satellite scenario is false, you can modify the SystemLoss value while SimulationStatus is NotStarted or InProgress.

PreReceiverLoss — Pre-receiver loss
3 (default) | scalar | vector

Pre-receiver loss in dB, specified as a scalar or a vector. This is the total loss before the receiver input in the receiver system, such as feeder loss, radome loss, and loss due to polarization mismatch. PreReceiverLoss must be less than or equal to SystemLoss.

Pre-receiver loss can be specified in receiver function as a name-value pair consisting of 'PreReceiverLoss' and a scalar, or a vector.

- If you are adding only one receiver, PreReceiverLoss is a scalar.
- If you are adding multiple receivers, PreReceiverLoss is a scalar or a vector. When PreReceiverLoss is a scalar, the same PreReceiverLoss is assigned to all receivers. When PreReceiverLoss is a vector, its length must equal the number of receivers and each element of PreReceiverLoss is assigned to the corresponding receivers in the parent.

If you specify SystemLoss property as a name value argument in receiver function , the default value is the lesser of 3 dB and the specified SystemLoss value.

When the AutoSimulate property of the satellite scenario is false, you can modify the PreReceiverLoss value while SimulationStatus is NotStarted or InProgress.

GainToNoiseTemperatureRatio — Gain to noise temperature ratio
3 (default) | scalar | vector

Gain to noise temperature ratio of the antenna in dB per Kelvin, specified as the name-value argument consisting of 'GainToNoiseTemperatureRatio' and a scalar or a vector.

- If you are adding only one receiver, GainToNoiseTemperatureRatio is a scalar.
- If you are adding multiple receivers, GainToNoiseTemperatureRatio is a scalar, or a vector. When GainToNoiseTemperatureRatio is a scalar, the same GainToNoiseTemperatureRatio is assigned to all receivers. When GainToNoiseTemperatureRatio is a vector, its length must equal the number of receivers and each element of GainToNoiseTemperatureRatio is assigned to the corresponding receiver in the parent.

When the AutoSimulate property of the satellite scenario is false, you can modify the GainToNoiseTemperatureRatio value while SimulationStatus is NotStarted or InProgress.

RequiredEbNo — Minimum Eb/No necessary for link closure
10 (default) | scalar | vector

Minimum energy per bit to noise power spectral density ratio (Eb/No) necessary for link closure in dB, specified as the name-value pair consisting of 'RequiredEbNo' and a scalar or a vector.

- If you are adding only one receiver, RequiredEbNo is a scalar.
- If you are adding multiple receivers, RequiredEbNo is a scalar or a vector. When RequiredEbNo is a scalar, the same RequiredEbNo is assigned to all receivers. When RequiredEbNo is a vector,

its length must equal the number of receivers and each element of `RequiredEbNo` is assigned to the corresponding receiver in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, the `RequiredEbNo` property can be modified while `SimulationStatus` is `NotStarted` or `InProgress`.

Output Arguments

`rx` — Receiver
row vector

Receivers attached to `parent`, returned as a row vector.

Version History

Introduced in R2021a

R2022b: PreReceiverLoss argument added to the function

You can now specify the pre-receiver loss in dB using the `PreReceiverLoss` name value argument.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer` | `Receiver` | `Transmitter`

Functions
`play` | `groundStation` | `transmitter` | `link` | `pattern` | `pointAt` | `pointAt`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

gimbal

Add gimbal to satellite or ground station

Syntax

```
gimbal(parent)
gimbal(parent,Name=Value)
gimbal( ___ )
```

Description

`gimbal(parent)` adds a default `Gimbal` object to each parent in the `parent` vector, which can be a satellite or a ground station. A gimbal can be added to satellites and ground stations, and dynamically change orientation independent of the parent. Transmitters, receivers, and conical sensors can be mounted on the gimbals.

`gimbal(parent,Name=Value)` adds gimbals to parents in `parent` using additional parameters specified by optional name-value pairs.

`gim = gimbal( ___ )` returns the added gimbals in the row vector `gim`.

Examples

Calculate Maximum Revisit Time of Satellite

Create a satellite scenario with a start time of 15-June-2021 8:55:00 AM UTC and a stop time of five days later. Set the simulation sample time to `60` seconds.

```
startTime = datetime(2021,6,21,8,55,0);
stopTime = startTime + days(5);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc =
  satelliteScenario with properties:

         StartTime: 21-Jun-2021 08:55:00
          StopTime: 26-Jun-2021 08:55:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario using Keplerian orbital elements.

```
semiMajorAxis = 7878137;                                                                    % me
eccentricity = 0;
inclination = 50;                                                                           % de
```

```
rightAscensionOfAscendingNode = 0;                                                    % deg
argumentOfPeriapsis = 0;                                                              % deg
trueAnomaly = 50;                                                                     % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly)

sat = 
  Satellite with properties:

                 Name:  Satellite 1
                   ID:  1
       ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
              Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
         Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
            Receivers:  [1x0 satcom.satellitescenario.Receiver]
             Accesses:  [1x0 matlabshared.satellitescenario.Access]
           GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
                Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
      OrbitPropagator:  sgp4
          MarkerColor:  [0.059 1 1]
           MarkerSize:  6
            ShowLabel:  true
       LabelFontColor:  [1 1 1]
        LabelFontSize:  15
```

Add a ground station, which represents the location to be photographed, to the scenario.

```
gs = groundStation(sc,Name="Location to Photograph", ...
    Latitude=42.3001,Longitude=-71.3504)                % degrees

gs = 
  GroundStation with properties:

                   Name:  Location to Photograph
                     ID:  2
               Latitude:  42.3 degrees
              Longitude:  -71.35 degrees
               Altitude:  0 meters
      MinElevationAngle:  0 degrees
         ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
                Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
           Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
              Receivers:  [1x0 satcom.satellitescenario.Receiver]
               Accesses:  [1x0 matlabshared.satellitescenario.Access]
            MarkerColor:  [1 0.4118 0.1608]
             MarkerSize:  6
              ShowLabel:  true
         LabelFontColor:  [1 1 1]
          LabelFontSize:  15
```

Add a gimbal to the satellite. You can steer this gimbal independently of the satellite.

```
g = gimbal(sat)

g = 
  Gimbal with properties:
```

```
              Name:  Gimbal 3
                ID:  3
  MountingLocation:  [0; 0; 0] meters
    MountingAngles:  [0; 0; 0] degrees
    ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
      Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
         Receivers:  [1x0 satcom.satellitescenario.Receiver]
```

Track the location to be photographed using the gimbal.

```
pointAt(g,gs);
```

Add a conical sensor to the gimbal. This sensor represents the camera. Set the field of view to 60 degrees.

```
camSensor = conicalSensor(g,MaxViewAngle=60)
```

```
camSensor = 
  ConicalSensor with properties:

              Name:  Conical sensor 4
                ID:  4
  MountingLocation:  [0; 0; 0] meters
    MountingAngles:  [0; 0; 0] degrees
      MaxViewAngle:  60 degrees
          Accesses:  [1x0 matlabshared.satellitescenario.Access]
       FieldOfView:  [0x0 matlabshared.satellitescenario.FieldOfView]
```

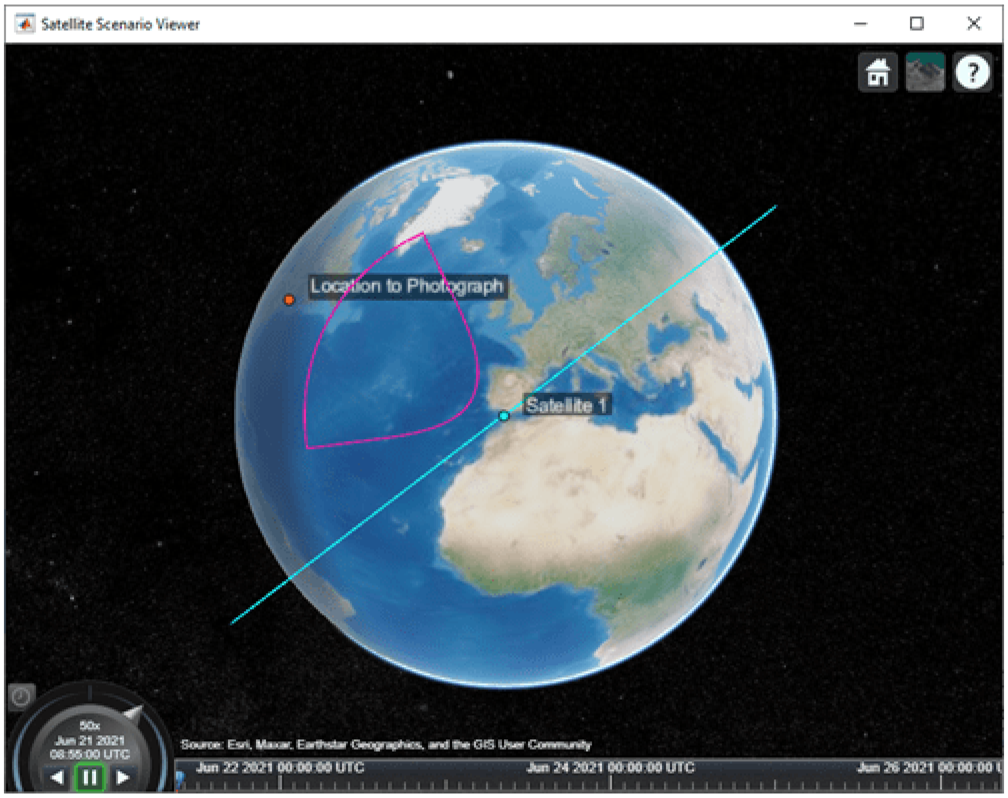
Add access analysis to the conical sensor between the camera and the location to be photographed.

```
ac = access(camSensor,gs)
```

```
ac = 
  Access with properties:

    Sequence:  [4 2]
    LineWidth:  3
    LineColor:  [0.3922 0.8314 0.0745]
```

Visualize the field of view of the camera by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
fieldOfView(camSensor);
```



Determine the intervals during which the camera can see the geographical site.

```
t = accessIntervals(ac)
```

```
t=35×8 table
           Source                    Target                IntervalNumber         StartTime
    _________________    _______________________    ______________    ____________________

    "Conical sensor 4"    "Location to Photograph"          1          21-Jun-2021 10:38:00
    "Conical sensor 4"    "Location to Photograph"          2          21-Jun-2021 12:36:00
    "Conical sensor 4"    "Location to Photograph"          3          21-Jun-2021 14:37:00
    "Conical sensor 4"    "Location to Photograph"          4          21-Jun-2021 16:41:00
    "Conical sensor 4"    "Location to Photograph"          5          21-Jun-2021 18:44:00
    "Conical sensor 4"    "Location to Photograph"          6          21-Jun-2021 20:46:00
    "Conical sensor 4"    "Location to Photograph"          7          21-Jun-2021 22:50:00
    "Conical sensor 4"    "Location to Photograph"          8          22-Jun-2021 09:51:00
    "Conical sensor 4"    "Location to Photograph"          9          22-Jun-2021 11:46:00
    "Conical sensor 4"    "Location to Photograph"         10          22-Jun-2021 13:46:00
    "Conical sensor 4"    "Location to Photograph"         11          22-Jun-2021 15:50:00
    "Conical sensor 4"    "Location to Photograph"         12          22-Jun-2021 17:53:00
    "Conical sensor 4"    "Location to Photograph"         13          22-Jun-2021 19:55:00
    "Conical sensor 4"    "Location to Photograph"         14          22-Jun-2021 21:58:00
    "Conical sensor 4"    "Location to Photograph"         15          23-Jun-2021 10:56:00
    "Conical sensor 4"    "Location to Photograph"         16          23-Jun-2021 12:56:00
      ⋮
```

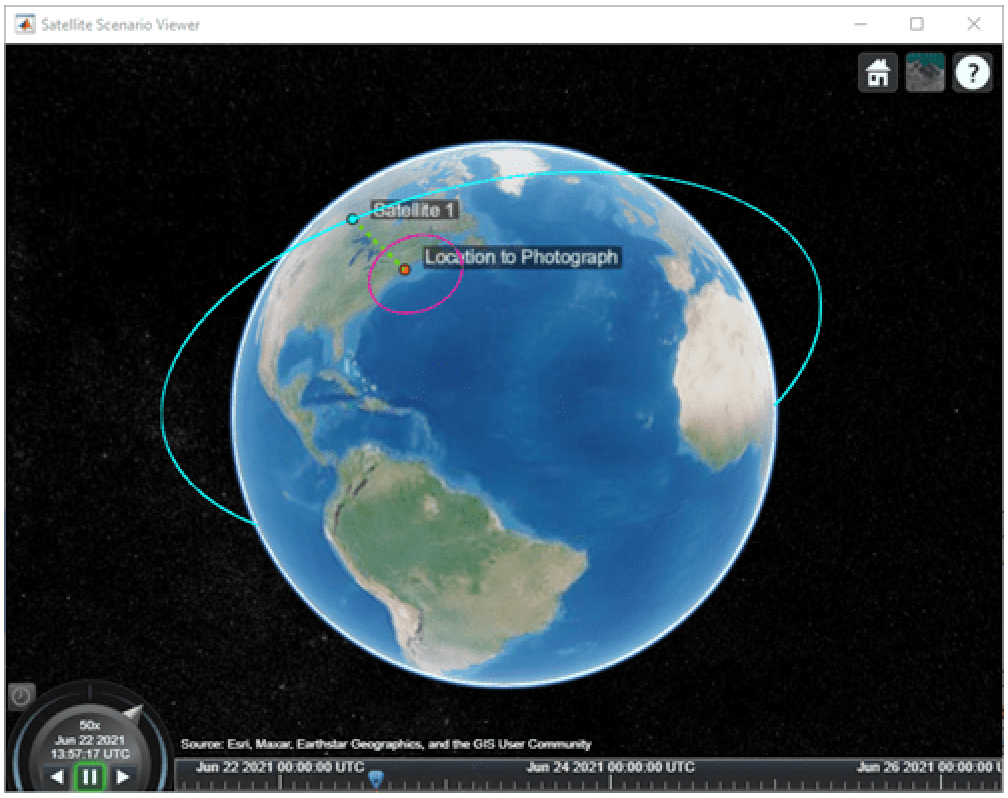
Calculate the maximum revisit time in hours.

```
startTimes = t.StartTime;
endTimes = t.EndTime;
revisitTimes = hours(startTimes(2:end) - endTimes(1:end-1));
maxRevisitTime = max(revisitTimes)                              % hours
```

```
maxRevisitTime = 12.6667
```

Visualize the revisit times that the camera photographs of the location.

```
play(sc);
```



Input Arguments

parent — Element of scenario to which you add gimbal

scalar | vector

Element of scenario to which you add the gimbal, specified as a scalar or vector of satellites or ground stations. The number of gimbals specified is determined by the size of the inputs.

- If `parent` is a scalar, all gimbals are added to the parent.
- If `parent` is a vector and the number of gimbals specified is one, that gimbal is added to each parent.
- If `parent` is a vector and the number of gimbals specified is more than one, the number of gimbals must equal the number of `parent`s and each `parent` gets one gimbal.

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: `'MountingAngle'=[20;35;10]` sets the yaw, pitch, and roll angles of the gimbal to 20, 35, and 10 degrees, respectively.

Note The size of the name-value arguments defines the number of gimbals that you can specify. To understand how to specify multiple gimbals, refer to these properties.

`Name` — Gimbal name

`"Gimbal` *`idx`*`"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `gimbal` function. After you call the `gimbal` function, this property is read-only.

Gimbal name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one gimbal, specify `Name` as a string scalar or a character vector.
- If you are adding multiple gimbals, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All gimbals that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of gimbals that you are adding. Each gimbal is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`MountingLocation` — Mounting location with respect to parent

`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one gimbal, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of gimbal.
- If you are adding multiple gimbals, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified gimbals. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the gimbals. The columns correspond to the mounting location of each specified gimbal and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one gimbal, the `MountingAngles` property is a three-element vector.
- If you are adding multiple gimbals the `MountingAngles` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified gimbals. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the gimbals. The columns correspond to the mounting angles of each specified gimbal and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

Output Arguments

`gim` — Gimbal
scalar | vector

Gimbal object attached to `parent`, returned as either a scalar or a vector.

When the `AutoSimulate` property of the satellite scenario is `false`, you can call the `gimbal` function only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Version History
Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `access` | `groundStation` | `satellite` | `conicalSensor` | `hide`

Topics
"Satellite Scenario Key Concepts"

"Satellite Scenario Basics"

fieldOfView

Package: `matlabshared.satellitescenario`

Visualize field of view of conical sensor

Syntax

```
fieldOfView(sensor)
fieldOfView(sensor,Name,Value)
fov = fieldOfView( ___ )
```

Description

`fieldOfView(sensor)` adds a `FieldOfView` object to the specified conical sensor, and draws contours on the Earth. Each contour represents the field of view of a conical sensor in `sensor` based on the current state of the scenario.

Locations inside the contour are inside the field of view. The field of view contours are drawn on all open satellite scenario viewers. The contours are the lines of intersection of the surface of the earth and the field of view cone. The half angle of the field of view cone equals the `MaxViewAngle` property of the conical sensor, and the axis of the cone is the *z*-axis (or boresight) of the conical sensor. The vertex of the cone is located at the position of the conical sensor. The cone becomes wider along the positive body *z*-axis of the conical sensor.

`fieldOfView(sensor,Name,Value)` specifies options by using one or more name-value arguments.

`fov = fieldOfView( ___ )` returns a vector of handles to the added field of view graphic objects. Specify any input combination from previous syntaxes.

Examples

Calculate Maximum Revisit Time of Satellite

Create a satellite scenario with a start time of 15-June-2021 8:55:00 AM UTC and a stop time of five days later. Set the simulation sample time to `60` seconds.

```
startTime = datetime(2021,6,21,8,55,0);
stopTime = startTime + days(5);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 21-Jun-2021 08:55:00
          StopTime: 26-Jun-2021 08:55:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
```

```
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario using Keplerian orbital elements.

```
semiMajorAxis = 7878137;                                                    % me
eccentricity = 0;
inclination = 50;                                                           % deg
rightAscensionOfAscendingNode = 0;                                          % deg
argumentOfPeriapsis = 0;                                                    % deg
trueAnomaly = 50;                                                           % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  Satellite with properties:

               Name:  Satellite 1
                 ID:  1
     ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
            Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
       Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
          Receivers:  [1x0 satcom.satellitescenario.Receiver]
           Accesses:  [1x0 matlabshared.satellitescenario.Access]
        GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
              Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator:  sgp4
        MarkerColor:  [0.059 1 1]
         MarkerSize:  6
          ShowLabel:  true
     LabelFontColor:  [1 1 1]
      LabelFontSize:  15
```

Add a ground station, which represents the location to be photographed, to the scenario.

```
gs = groundStation(sc,Name="Location to Photograph", ...
    Latitude=42.3001,Longitude=-71.3504)                 % degrees
```

```
gs = 
  GroundStation with properties:

                 Name:  Location to Photograph
                   ID:  2
             Latitude:  42.3 degrees
            Longitude:  -71.35 degrees
             Altitude:  0 meters
    MinElevationAngle:  0 degrees
       ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
              Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
         Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
            Receivers:  [1x0 satcom.satellitescenario.Receiver]
             Accesses:  [1x0 matlabshared.satellitescenario.Access]
          MarkerColor:  [1 0.4118 0.1608]
           MarkerSize:  6
            ShowLabel:  true
       LabelFontColor:  [1 1 1]
```

```
        LabelFontSize:  15
```

Add a gimbal to the satellite. You can steer this gimbal independently of the satellite.

```
g = gimbal(sat)
```

```
g = 
  Gimbal with properties:

                Name:  Gimbal 3
                  ID:  3
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
      ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
        Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
           Receivers:  [1x0 satcom.satellitescenario.Receiver]
```

Track the location to be photographed using the gimbal.

```
pointAt(g,gs);
```

Add a conical sensor to the gimbal. This sensor represents the camera. Set the field of view to 60 degrees.

```
camSensor = conicalSensor(g,MaxViewAngle=60)
```

```
camSensor = 
  ConicalSensor with properties:

                Name:  Conical sensor 4
                  ID:  4
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
        MaxViewAngle:  60 degrees
            Accesses:  [1x0 matlabshared.satellitescenario.Access]
         FieldOfView:  [0x0 matlabshared.satellitescenario.FieldOfView]
```

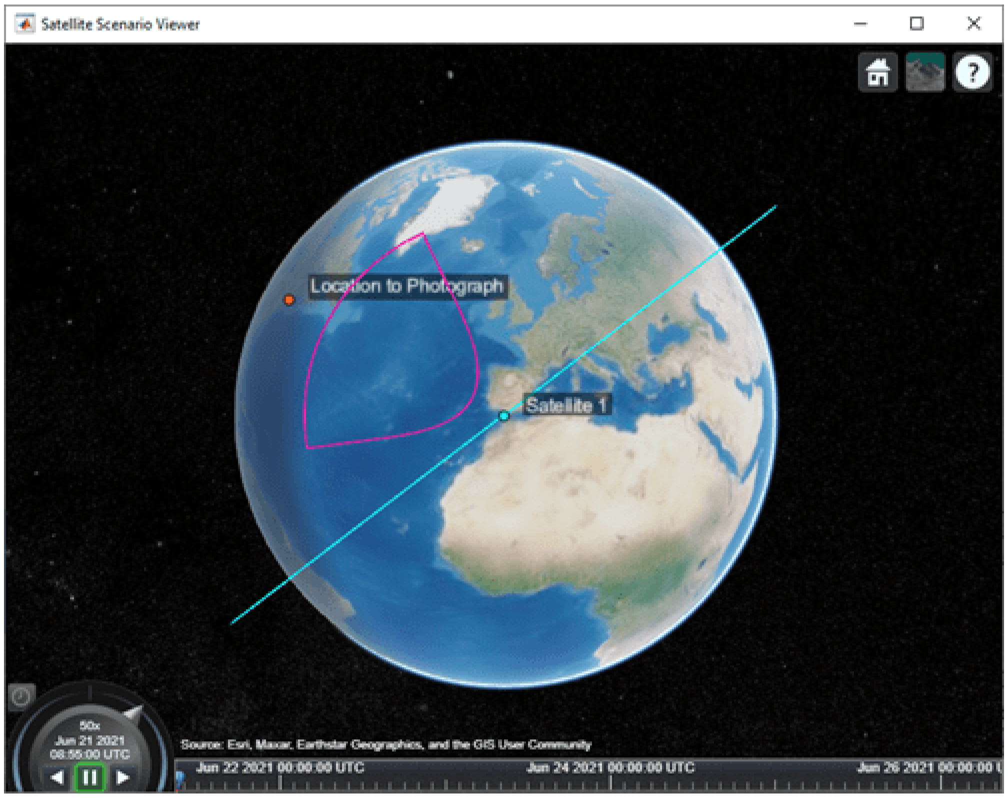
Add access analysis to the conical sensor between the camera and the location to be photographed.

```
ac = access(camSensor,gs)
```

```
ac = 
  Access with properties:

    Sequence:  [4 2]
    LineWidth:  3
    LineColor:  [0.3922 0.8314 0.0745]
```

Visualize the field of view of the camera by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
fieldOfView(camSensor);
```



Determine the intervals during which the camera can see the geographical site.

```
t = accessIntervals(ac)
```

```
t=35×8 table
          Source                  Target              IntervalNumber         StartTime
    ___________________    ________________________    ______________    ____________________

    "Conical sensor 4"    "Location to Photograph"          1           21-Jun-2021 10:38:00
    "Conical sensor 4"    "Location to Photograph"          2           21-Jun-2021 12:36:00
    "Conical sensor 4"    "Location to Photograph"          3           21-Jun-2021 14:37:00
    "Conical sensor 4"    "Location to Photograph"          4           21-Jun-2021 16:41:00
    "Conical sensor 4"    "Location to Photograph"          5           21-Jun-2021 18:44:00
    "Conical sensor 4"    "Location to Photograph"          6           21-Jun-2021 20:46:00
    "Conical sensor 4"    "Location to Photograph"          7           21-Jun-2021 22:50:00
    "Conical sensor 4"    "Location to Photograph"          8           22-Jun-2021 09:51:00
    "Conical sensor 4"    "Location to Photograph"          9           22-Jun-2021 11:46:00
    "Conical sensor 4"    "Location to Photograph"         10           22-Jun-2021 13:46:00
    "Conical sensor 4"    "Location to Photograph"         11           22-Jun-2021 15:50:00
    "Conical sensor 4"    "Location to Photograph"         12           22-Jun-2021 17:53:00
    "Conical sensor 4"    "Location to Photograph"         13           22-Jun-2021 19:55:00
    "Conical sensor 4"    "Location to Photograph"         14           22-Jun-2021 21:58:00
    "Conical sensor 4"    "Location to Photograph"         15           23-Jun-2021 10:56:00
    "Conical sensor 4"    "Location to Photograph"         16           23-Jun-2021 12:56:00
      ⋮
```

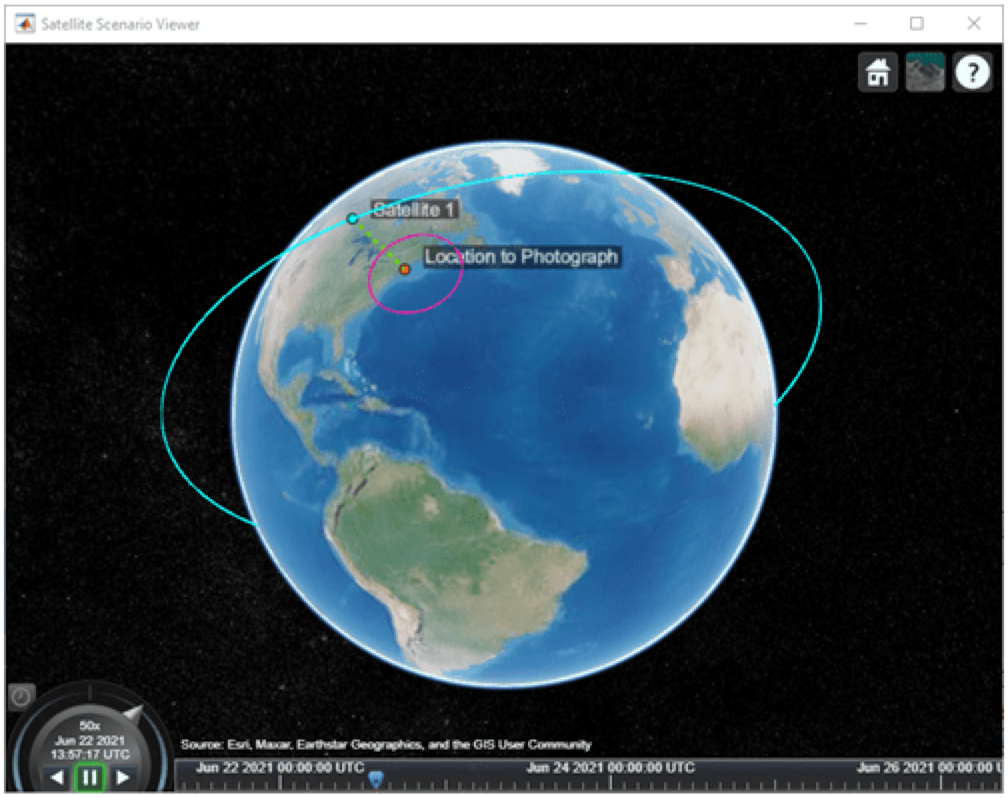
Calculate the maximum revisit time in hours.

```
startTimes = t.StartTime;
endTimes = t.EndTime;
revisitTimes = hours(startTimes(2:end) - endTimes(1:end-1));
maxRevisitTime = max(revisitTimes)                               % hours
```

```
maxRevisitTime = 12.6667
```

Visualize the revisit times that the camera photographs of the location.

```
play(sc);
```



Input Arguments

sensor — Conical sensor
ConicalSensor object

Conical sensor, specified as a `ConicalSensor` object.

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: `'LineWidth',2.5` sets the line width of the field of view to 2.5 pixels.

`Viewer` — Satellite scenario viewer
vector of `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a scalar, vector, or array of `satelliteScenarioViewer` objects. If the `AutoSimulate` property of the scenario is `false`, adding a satellite to the scenario disables any previously available timeline and playback widgets.

`NumContourPoints` — Number of contour points
`40` (default) | integer greater than or equal to 4

Number of contour points used to draw the contour of the field of view, specified as an integer greater than or equal to 4.

Data Types: `double`

`LineWidth` — Visual width of field of view contour
`1` (default) | scalar in the range (0 10]

Visual width of the field of view contour in pixels, specified as a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of field of view contour
`[0 1 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of field of view contour, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	🟪

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
"yellow"	"y"	[1 1 0]	"#FFFF00"	🟨
"black"	"k"	[0 0 0]	"#000000"	⬛
"white"	"w"	[1 1 1]	"#FFFFFF"	⬜
"none"	Not applicable	Not applicable	Not applicable	No color

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"	🟥

Example: 'blue'

Example: [0 0 1]

Example: '#0000FF'

Output Arguments

fov — Field of view of conical sensor
row vector of FieldOfView objects

Field of view of conical sensor, returned as a row vector of FieldOfView objects.

Note When the AutoSimulate property is set to false, the SimulationStatus must equal NotStarted to call the fieldOfView function. Otherwise, use the restart function to reset the SimulationStatus to NotStarted, which erases the simulation data.

Version History

Introduced in R2021a

See Also

Objects
satelliteScenario | satelliteScenarioViewer

Functions
show | play | hide | access | groundStation | conicalSensor | transmitter | receiver

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

link

Package: `satcom.satellitescenario`

Add link analysis objects to transmitter

Syntax

```
link(asset1,asset2,...,assetN)
lnk = link( ___ ,Name=Value)
lnk = link( ___ )
```

Description

`link(asset1,asset2,...,assetN)` adds `Link` analysis objects defined by nodes `asset1`, `asset2`, and so on.

`lnk = link( ___ ,Name=Value)` adds link analysis objects using additional parameters specified as name-value pairs.

`lnk = link( ___ )` adds link analysis objects and returns the vector link

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                                   % mete
eccentricity = 0;
inclination = 60;                                                                           % deg
rightAscensionOfAscendingNode = 0;                                                          % deg
```

```
argumentOfPeriapsis = 0;                                                                                 % deg
trueAnomaly = 0;                                                                                         % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                                      % dB/K
systemLoss = 3;                                                                       % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)
```

```
rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                    RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                                   % Hz
power = 20;                                                                         % dBW
bitRate = 20;                                                                       % Mbps
systemLoss = 3;                                                                     % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)
```

```
txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                                   % meters
apertureEfficiency = 0.5;
```

```
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                                              % degrees
longitude = 0.1487094;                                              % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                        % Hz
power = 40;                                                                              % dBW
bitRate = 20;                                                                            % Mbp
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                                   % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                                 % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
```

```
lnk = 
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

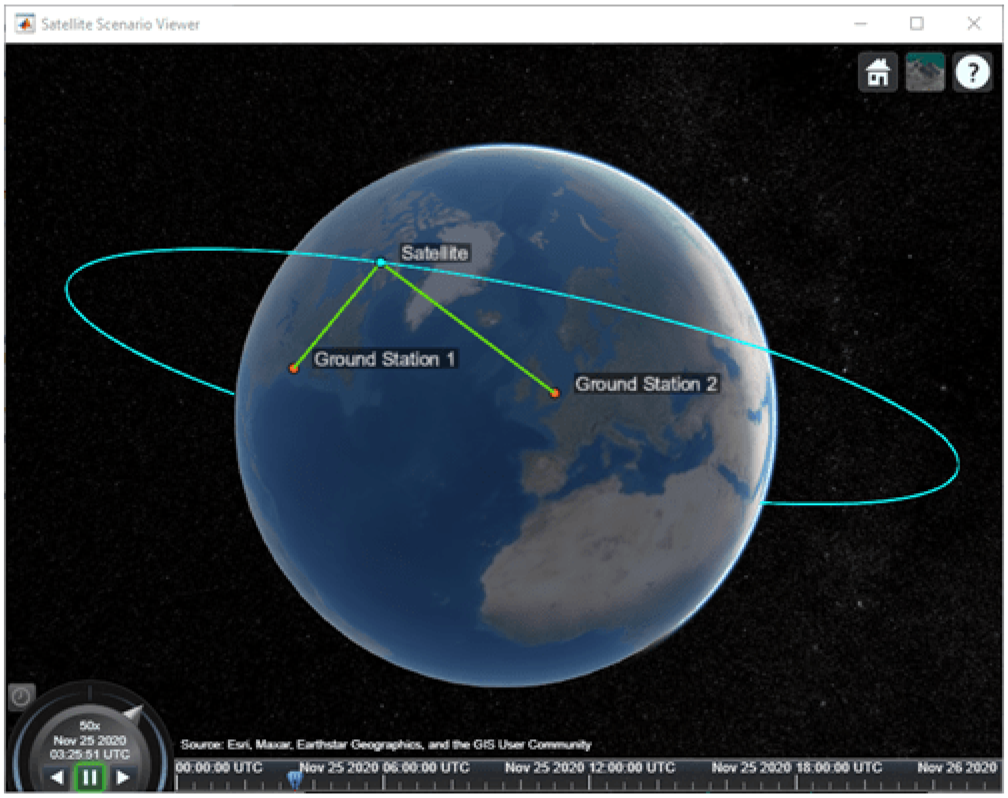
Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)
```

```
ans=4×8 table
                Source                            Target                IntervalNumber        Star
    ______________________________    ___________________________    ______________    _______

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          1           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          2           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          3           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          4           25-Nov-202
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Input Arguments

`asset1,asset2,...,assetN` — Adds link analysis objects
scalar | vector

Adds link analysis objects defined by nodes `asset1`, specified as a scalar or vector of transmitters, `asset2`, and so on, specified as a scalar or a vector of transmitters or receivers.

- If the asset in a given node is scalar, every link analysis object uses the same asset for that node position.
- If the asset in a given node is vector, the asset length must equal the number of link analysis objects.

Each link analysis object uses the corresponding element of the asset vector for that node location. The IDs of ASSET1, ASSET2, ASSET3, and so on, specify the Sequence of the link. These objects must belong to the same `satelliteScenario` object. Each link analysis object is added to the Link property of the corresponding transmitter in ASSET1.

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: `LineWidth=2.5` sets the line width of the field of view to 2.5 pixels.

`Viewer` — Satellite scenario viewer
vector of `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a scalar, vector, or array of `satelliteScenarioViewer` objects. If the `AutoSimulate` property of the scenario is `false`, adding a satellite to the scenario disables any previously available timeline and playback widgets.

Output Arguments

`lnk` — Link analysis
scalar | row vector

Link analysis object between input objects, returned as either a scalar or a row vector.

Note When `AutoSimulate` of the satellite scenario is `false`, you can call `link` only when the `SimulationStatus` is `NotStarted`. Otherwise, you must call the `restart` function to erase the simulation data and reset the `SimulationStatus` to `NotStarted`.

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`transmitter` | `receiver` | `sigstrength`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

gaussianAntenna

Package: `satcom.satellitescenario`

Add Gaussian antennas

Syntax

```
gaussianAntenna(trx)
gaussianAntenna(trx,Name=Value)
ant = gaussianAntenna( ___ )
```

Description

`gaussianAntenna(trx)` adds a Gaussian antenna to each transmitter or receiver in the vector `trx` using default parameters. The existing antennas in the transmitters or receivers are overwritten. For more information about antenna gain, see “Algorithm” on page 2-290.

`gaussianAntenna(trx,Name=Value)` adds a Gaussian antenna to each transmitter or receiver in the vector `trx` and specifies options using name-value arguments. For example, `DishDiameter=`1.7 defines the diameter of the antenna dish.

`ant = gaussianAntenna( ___ )` adds a Gaussian antenna to the transmitters or receivers and returns them in the vector `ant`.

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                    % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                                  % mete
eccentricity = 0;
inclination = 60;                                                                          % degr
rightAscensionOfAscendingNode = 0;                                                         % degr
argumentOfPeriapsis = 0;                                                                   % degr
trueAnomaly = 0;                                                                           % degr
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                                        % dB/K
systemLoss = 3;                                                                         % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)
```

```
rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                  RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                                     % Hz
power = 20;                                                                           % dBW
bitRate = 20;                                                                         % Mbps
systemLoss = 3;                                                                       % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)
```

```
txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                             % meters
apertureEfficiency = 0.5;
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                                            % degrees
longitude = 0.1487094;                                            % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                 % Hz
power = 40;                                                                       % dBW
bitRate = 20;                                                                     % Mbps
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                              % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                                 % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
```

```
lnk = 
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

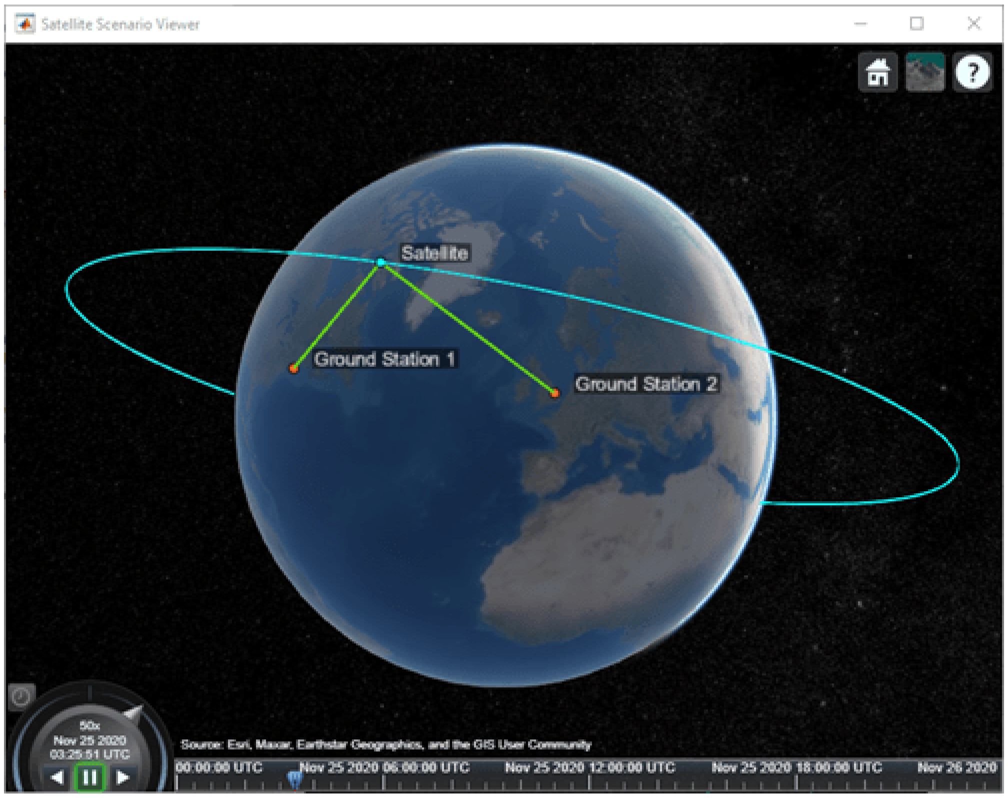
Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)
```

```
ans=4×8 table
               Source                              Target                  IntervalNumber          Star
    _________________________________    ______________________________    ______________    __________

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          1              25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          2              25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          3              25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          4              25-Nov-202
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Input Arguments

`trx` — Transmitter or receiver

scalar | vector

Transmitter or receiver object to which the Gaussian antenna is added, specified as either a scalar or a vector.

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: `'DishDiameter'=1.7` sets the dish diameter of the antenna to 1.7 meters upon creation.

`DishDiameter` — Diameter of the antenna dish

`1` (default) | scalar | vector

This property is read-only.

You can set this property only when calling `gaussianAntenna`. After you call gaussianAntenna, this property is read-only.

Diameter of the Gaussian antenna dish in meters, specified as a scalar or a vector.

- If `DishDiameter` is a scalar, the same value is assigned to all transmitters or receivers in `trx`.
- If `DishDiameter` is a vector, the length of the vector must equal that of `trx`, and each transmitter or receiver in `trx` is assigned the corresponding element in the `DishDiameter` vector.

`ApertureEfficiency` — Aperture efficiency of Gaussian antenna

`0.65` (default) | scalar in the range (0,1] | vector

This property is read-only.

You can set this property only when calling gaussianAntenna. After you call gaussianAntenna, this property is read-only.

Aperture efficiency of the Gaussian antenna, specified as a scalar in the range (0,1].

- If `ApertureEfficiency` is a scalar, the same value is assigned to all transmitters or receivers in `trx`.
- If `ApertureEfficiency` is a vector, the length of the vector must equal that of `trx`, and each transmitter or receiver in `trx` is assigned the corresponding element in the `ApertureEfficiency` vector.

Output Arguments

`ant` — Gaussian antenna

scalar | vector

Gaussian antenna object added to the specified transmitter or receiver, returned as either a scalar or a vector.

Note When the `AutoSimulate` property of satellite scenario is `false`, you can call the `gaussianAntenna` function only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset the `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Algorithms

The Gaussian antenna approximates a parabolic reflector using a Gaussian curve as
  gain(*theta*) = *boresightGain**exp(-(4*log(2)*((*theta*/*3dBbeamwidth*)^2)))
Note that log(2) is a natural log, where:

- *boresightGain* = *rho_a**((pi**d*/*lambda*)^2)
- *3dBbeamwidth* = 70**lambda*/*d*
  (in degrees)
- *theta* is the angle between the direction in which the gain is computed and the boresight direction is in degrees.
- *rho_a* is the aperture efficiency.
- *d* is the dish diameter in meters.
- *lambda* is the wavelength in meters.

The gain in dB is
  gain_dB(*theta*) = 10*log10(*gain*(*theta*)

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario`

Functions
`hide` | `show` | `play` | `satellite` | `access` | `groundStation` | `receiver` | `transmitter`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

groundTrack

Package: `matlabshared.satellitescenario`

Add ground track object to satellite in scenario

Syntax

```
groundTrack(sat)
groundTrack( ___ ,Name=Value)
```

Description

`groundTrack(sat)` adds ground track visualization for each satellite in `sat` based on their current positions. The ground track begins at the scenario StartTime, and ends at the StopTime. The spacing between samples that make up the ground track visualization is determined by the scenario `SampleTime`. If no viewer is open, a new viewer is launched, and the ground track is displayed. If a viewer is already open, the ground track is added to that viewer. By default, ground tracks will be displayed in 2-D.

`groundTrack( ___ ,Name=Value)` adds a `groundTrack` object by using one or more name-value pairs. Enclose each property name in quotes.

Examples

Add Ground Track to Satellite in Geosynchronous Orbit

Create a satellite scenario object.

```
startTime = datetime(2020,5,10);
stopTime = startTime + days(5);
sampleTime = 60;                                      % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Calculate the semimajor axis of the geosynchronous satellite.

```
earthAngularVelocity = 0.0000729211585530;                                                          % rad/s
orbitalPeriod = 2*pi/earthAngularVelocity;                                                          % seconds
earthStandardGravitationalParameter = 398600.4418e9;                                                % m^3/s^2
semiMajorAxis = (earthStandardGravitationalParameter*((orbitalPeriod/(2*pi))^2))^(1/3);
```

Define the remaining orbital elements of the geosynchronous satellite.

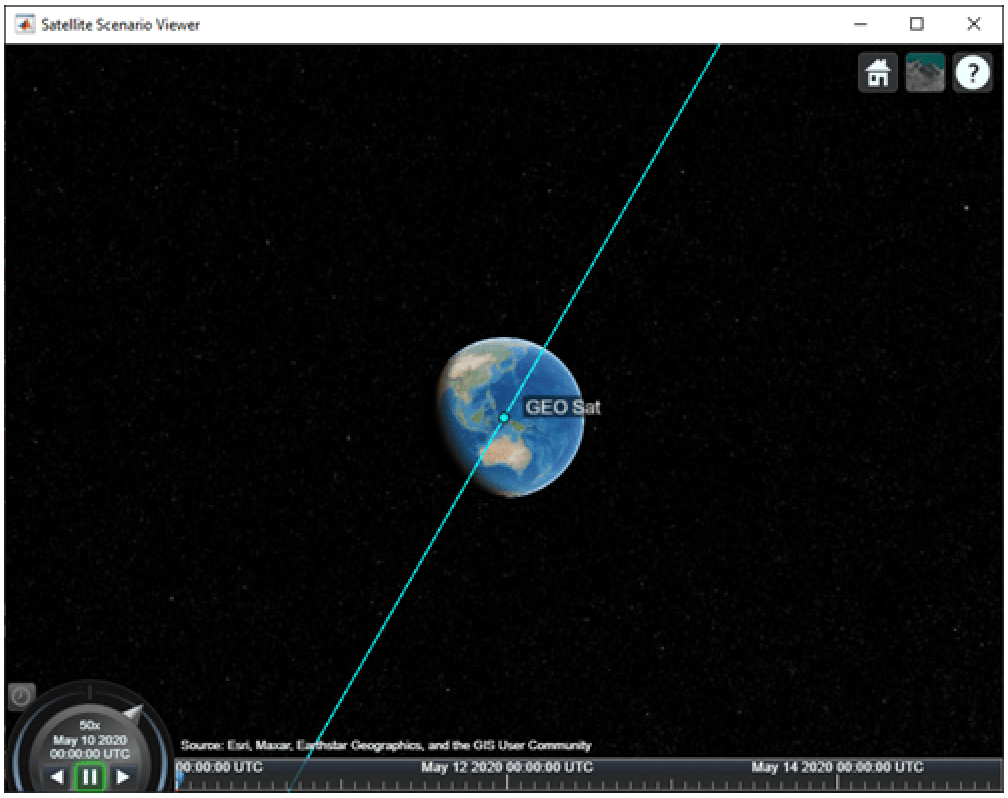
```
eccentricity = 0;
inclination = 60;                   % degrees
rightAscensionOfAscendingNode = 0; % degrees
argumentOfPeriapsis = 0;            % degrees
trueAnomaly = 0;                    % degrees
```

Add the geosynchronous satellite to the scenario.

```
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode,...
        argumentOfPeriapsis,trueAnomaly,"OrbitPropagator","two-body-keplerian","Name","GEO Sat")
```

Visualize the scenario using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```


Satellite Scenario Viewer
GEO Sat
50x
May 10 2020
00:00:00 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
00:00:00 UTC
May 12 2020 00:00:00 UTC
May 14 2020 00:00:00 UTC

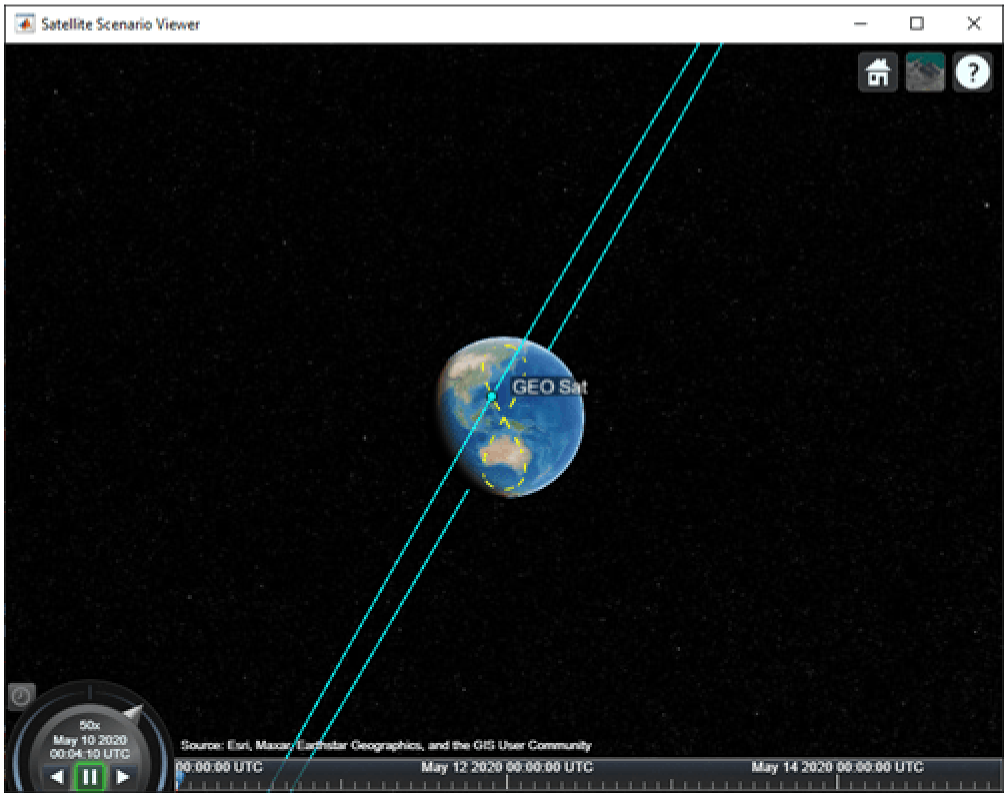
Add a ground track of the satellite to the visualization and adjust how much of the future and history of the ground track to display.

```
leadTime = 2*24*3600;                                           % seconds
trailTime = leadTime;
gt = groundTrack(sat,"LeadTime",leadTime,"TrailTime",trailTime)
```

```
gt = 
  GroundTrack with properties:

          LeadTime: 172800
         TrailTime: 172800
         LineWidth: 1
     LeadLineColor: [1 1 0.0670]
    TrailLineColor: [1 1 0.0670]
    VisibilityMode: 'inherit'
```

Visualize the satellite movement and its trace on the ground. The satellite covers the area around Japan during one half of the day and Australia during the other half.

```
play(sc);
```


Satellite Scenario Viewer
GEO Sat
50x
May 10 2020
00:04:10 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
00:00:00 UTC
May 12 2020 00:00:00 UTC
May 14 2020 00:00:00 UTC

Input Arguments

sat — Satellite
row vector of `Satellite` objects

Satellite, specified as a row vector of `Satellite` objects.

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.

Example: `LeadTime=3600` sets the lead time of the ground track to 3600 seconds upon creation.

Viewer — Satellite scenario viewer
vector of `satelliteScenarioViewer` objects (default) | scalar `satelliteScenarioViewer` object | array of `satelliteScenarioViewer` objects

Satellite scenario viewer, specified as a scalar, vector, or array of `satelliteScenarioViewer` objects. If the `AutoSimulate` property of the scenario is `false`, adding a satellite to the scenario disables any previously available timeline and playback widgets.

LeadTime — Period of ground track to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track to be visualized in the satellite scenario viewer, specified as `'LeadTime'` and a positive scalar in seconds.

`TrailTime` — Period of ground track history to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track history to be visualized in `Viewer`, specified as `'TrailTime'` and a positive scalar in seconds.

`LineWidth` — Visual width of ground track
1 (default) | scalar in the range (0 10]

Visual width of the ground track in pixels, specified as `'LineWidth'` and a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LeadLineColor` — Color of future ground track line
`[1 0 1]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the future ground track line, specified as `'LeadLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`TrailLineColor` — Color of ground track line history
`[1 0.5 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the ground track line history, specified as `'TrailLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `groundStation` | `access` | `hide` | `satellite`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

pattern

Package: `satcom.satellitescenario`

Plot 3-D radiation pattern of antenna

Syntax

```
pat = pattern(tx)
pat = pattern(rx,freq)
pat = pattern( ___ ,Name,Value)
```

Description

`pat = pattern(tx)` plots the 3-D radiation pattern of the antenna for each transmitter in vector `tx`. The signal gain value (in dBi) in a particular direction determines the color of the pattern. The function scales the pattern on the plot according to the `Size` name-value argument. The function plots the pattern for the transmitter frequency as specified by the `Frequency` property of `tx`.

`pat = pattern(rx,freq)` plots the 3-D radiation pattern of the antenna for each receiver in vector, `rx` with frequency `freq`.

`pat = pattern( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to any of the input argument combinations in previous syntaxes. For example, `'ColorMap','jet'` specifies the jet colormap for coloring the pattern plot.

Examples

Visualize Radiation Pattern of Transmitter Antenna on Satellite

Set up the satellite scenario.

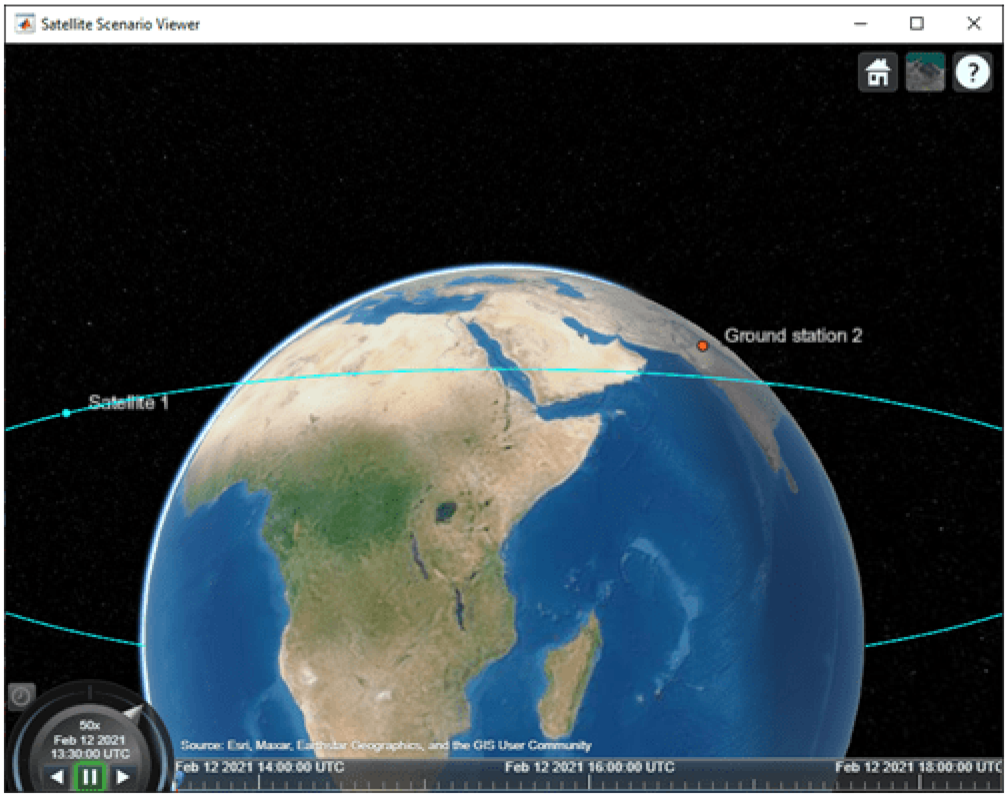
```
startTime = datetime(2021,2,12,13,30,0);
stopTime = startTime + hours(5);
sampleTime = 60;                                     %seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Create a satellite, ground station, transmitter, and receiver.

```
sat = satellite(sc,1e7,0,0,0,0,0);
gs = groundStation(sc,"Latitude",30,"Longitude",74);
tx = transmitter(sat,"Frequency",3e9);
rx = receiver(gs);
```

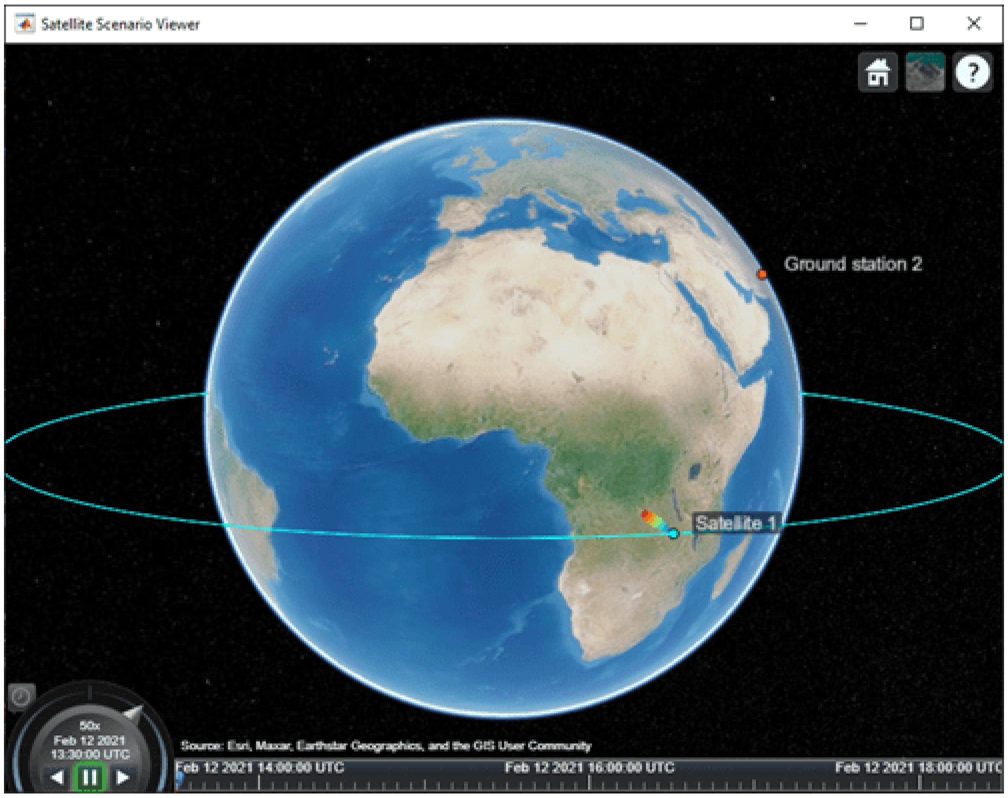
Visualize the scenario in the satellite scenario viewer.

```
viewer = satelliteScenarioViewer(sc);
```



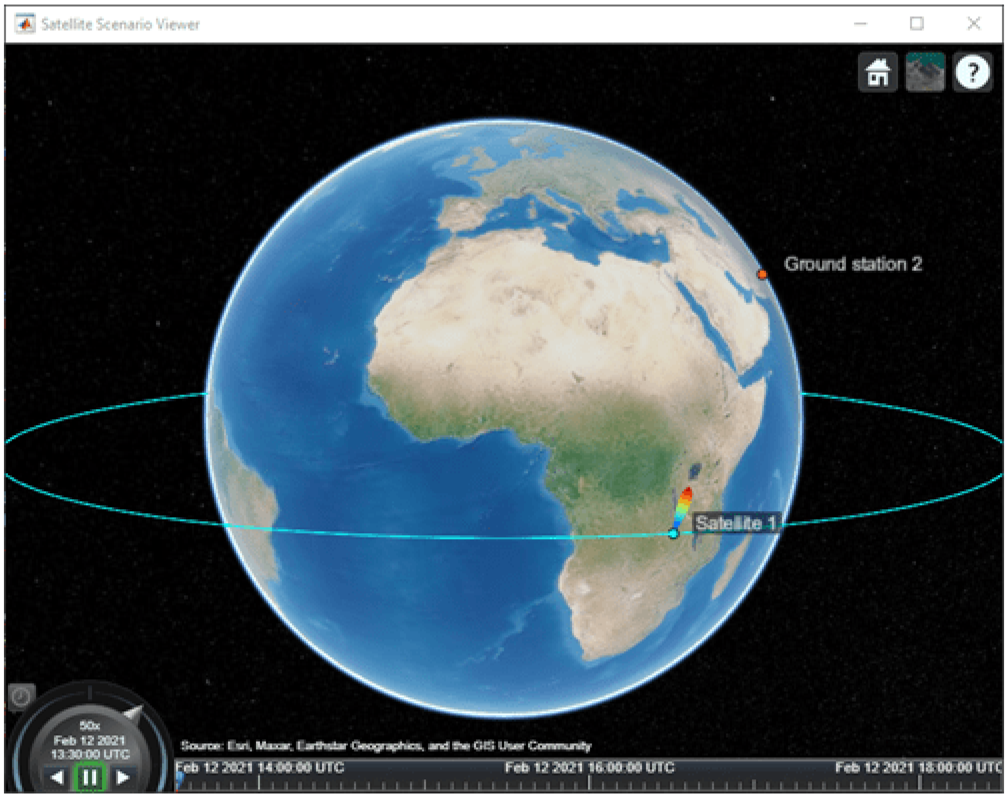
Plot the radiation pattern of the transmitter antenna.

```
pat = pattern(tx);
```



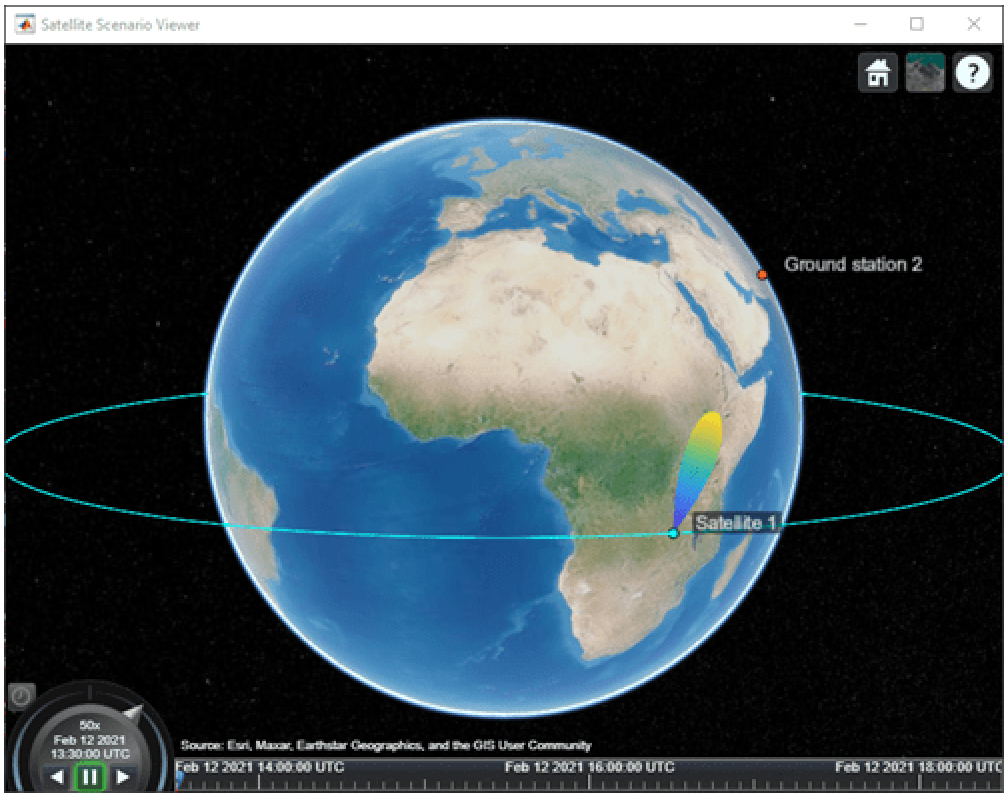
Point the satellite at the ground station. The pattern rotates to reflect the new orientation of the antenna.

```
pointAt(sat,gs);
```



Increase the visual size of the radiation pattern.

```
pat.Size = 3000000;
pat.Colormap = "parula";
```



Visualize Radiation Pattern of Receiver Antenna on Satellite

Set up the satellite scenario.

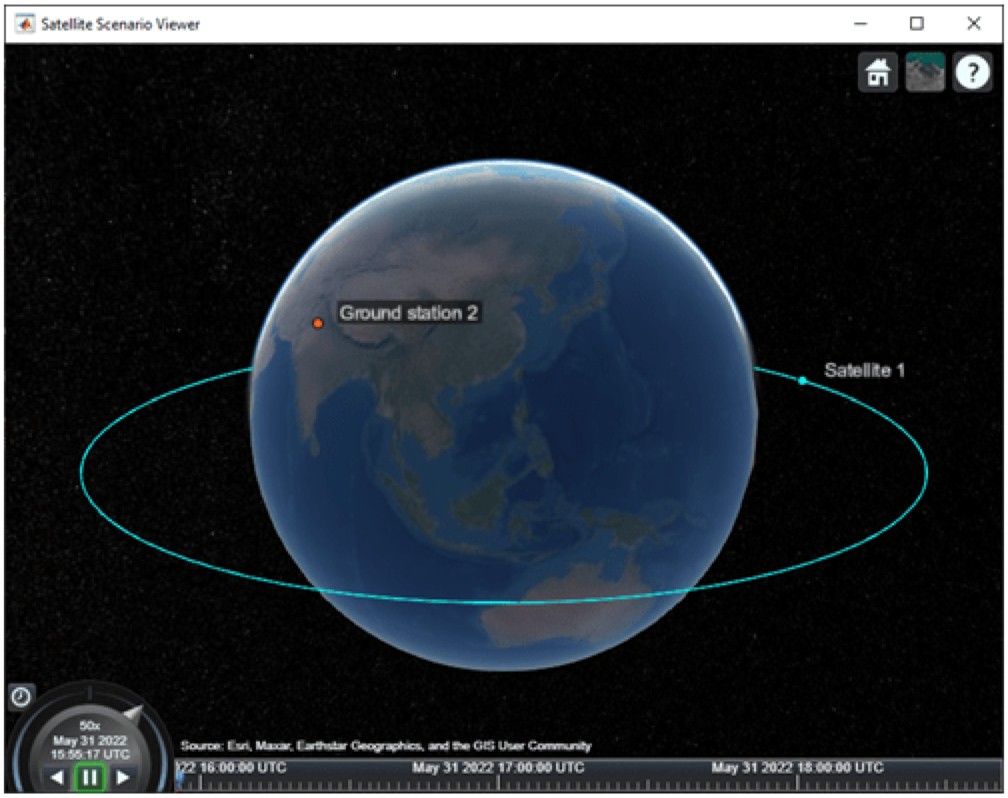
```
sc = satelliteScenario;
```

Create a satellite, ground station, transmitter, and receiver.

```
sat = satellite(sc,1e7,0,0,0,0,0);
gs = groundStation(sc,"Latitude",30,"Longitude",74);
tx = transmitter(sat,"Frequency",1e9);
rx = receiver(gs);
```

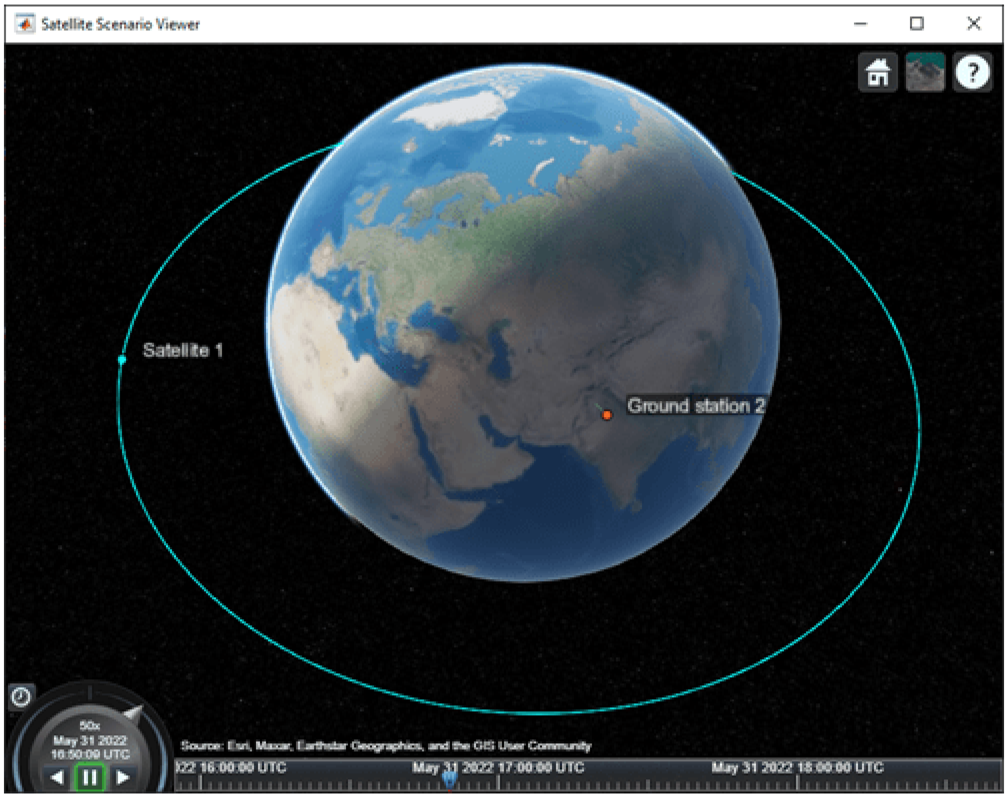
Visualize the scenario in the satellite scenario viewer.

```
viewer = satelliteScenarioViewer(sc);
```



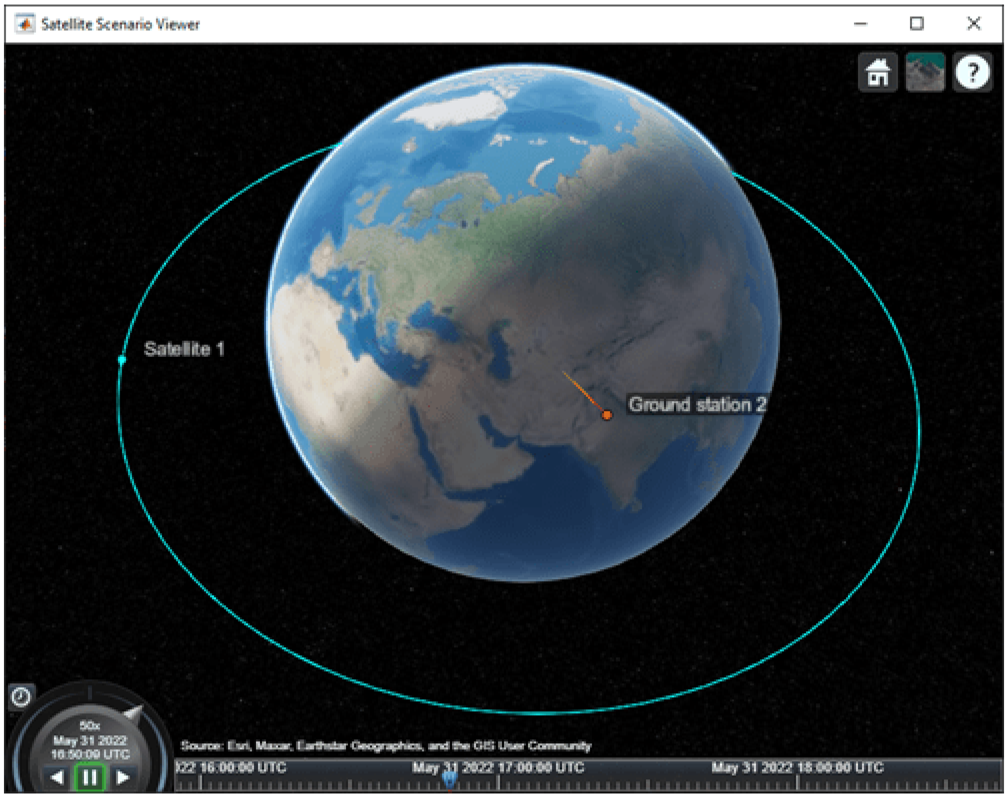
Plot the radiation pattern of the receiver antenna.

```
freq = 30e9;
pat = pattern(rx,freq);
```



Increase the visual size and specify the transparency of the radiation pattern.

```
pat.Size = 3000000;
pat.Colormap = 'autumn';
```



Input Arguments

tx — Transmitter
scalar | vector

Transmitter object, specified as either a scalar or vector.

rx — Receiver
scalar | vector

Receiver object, specified as either a scalar or vector.

freq — Frequency to calculate radiation pattern
scalar | vector

Frequency to calculate radiation pattern, specified as a scalar or a vector.

- If `freq` is scalar, its value is applied to the pattern of all receivers in `rx`.
- If `freq` is vector, its length must equal to that of `rx`.

Each element in `freq` corresponds to the pattern of the antenna of the receiver in `rx`.

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: `'Size',1000` sets the size of the radiation pattern plot to 1,000 meters.

`Size` — Size of radiation pattern plot

`1000000` (default) | numeric scalar

Size of the radiation pattern plot, specified as a numeric scalar in meters. This value represents the distance between the antenna position and the point on the plot with the highest gain.

Data Types: `double`

`Colormap` — Colormap for coloring pattern plot

`'jet'` (default) | predefined colormap name | *M*-by-3 matrix

Colormap for coloring the pattern plot, specified as a predefined colormap name or an *M*-by-3 matrix of red, green, blue (RGB) triplets that define *M* individual colors. For more information on the colormap names, see “map”.

Data Types: `double` | `char` | `string`

`Transparency` — Transparency of the pattern plot

`0.4` (default) | scalar in the range [0, 1]

Transparency of the pattern plot, specified as a scalar in the range [0, 1]. A value of `0` means the plot is completely transparent, and a value of `1` means the plot is opaque.

Data Types: `double`

`Resolution` — Resolution of 3-D pattern

`'high'` (default) | `'medium'` | `'low'`

Resolution of the 3-D pattern, specified as `'low'`, `'medium'`, or `'high'`. Use this argument to control the visual quality of the pattern and time the function takes to plot the pattern. `'low'` corresponds to the fastest and least-detailed pattern.

Data Types: `char` | `string`

`Viewer` — Satellite Scenario Viewer to visualize satellite

row vector (default) | scalar | matrix

Satellite Scenario Viewer to visualize the satellite, specified as a scalar, row vector, or matrix of `satelliteScenarioViewer` objects that are associated with the satellite scenario.

Output Arguments

`pat` — Radiation pattern visualization for transmitter or receiver

scalar | vector

Radiation pattern visualization for transmitter or receiver object, returned as either a scalar or vector.

Version History

Introduced in R2021b

See Also

Objects
`Receiver` | `Transmitter` | `satelliteScenarioViewer` | `satelliteScenario`

Functions
`show` | `hide` | `receiver` | `transmitter`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

advance

Move simulation forward by one sample time

Syntax

```
isrunning = advance(sc)
```

Description

`isrunning = advance(sc)` moves the simulation forward by the amount of time specified by the `SampleTime` property of the scenario `sc`.

Examples

Manual Simulation of Satellite Scenario

Create a satellite scenario object and set the `AutoSimulate` property to `false` to enable manual simulation of the satellite scenario.

```
startTime = datetime(2022,1,12);
stopTime = startTime + days(0.5);
sampleTime = 60;                             % Seconds
sc = satelliteScenario('AutoSimulate',false);
```

Add a GPS satellite constellation to the scenario.

```
sat = satellite(sc,"gpsAlmanac.txt");
```

Simulate the scenario using the `advance` function.

```
while advance(sc)
end
```

Obtain the satellite position histories.

```
p = states(sat);
```

`AutoSimulate` is `false`, so restart the scenario before adding a ground station.

```
restart(sc);
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Add access analysis between each satellite and ground station.

```
ac = access(sat,gs);
```

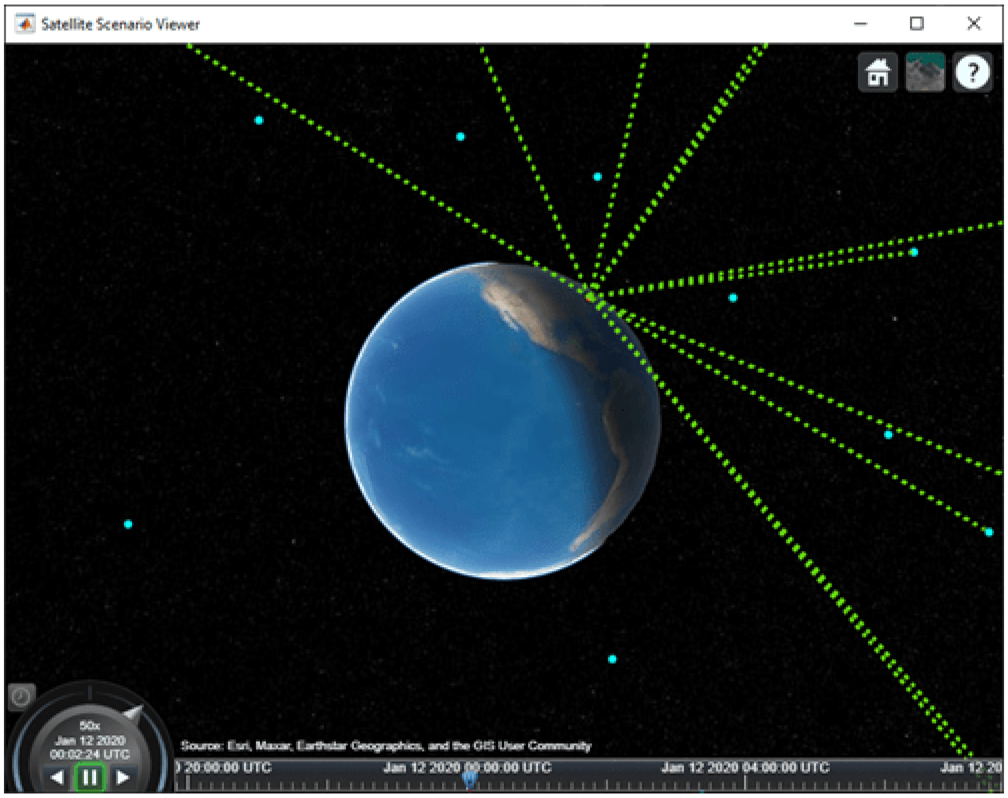
Simulate the scenario and determine the access intervals.

```
while advance(sc)
end
intvls1 = accessIntervals(ac)
```

```
intvls1=35×8 table
     Source            Target            IntervalNumber         StartTime               EndTime
    _________    _____________________    ______________    ______________________    ________________

    "PRN:1"      "Ground station 32"            1           11-Jan-2020 23:20:25    12-Jan-2020 05:
    "PRN:2"      "Ground station 32"            1           12-Jan-2020 04:03:16    12-Jan-2020 07:
    "PRN:3"      "Ground station 32"            1           11-Jan-2020 19:50:06    11-Jan-2020 21:
    "PRN:3"      "Ground station 32"            2           12-Jan-2020 01:52:43    12-Jan-2020 06:
    "PRN:4"      "Ground station 32"            1           11-Jan-2020 19:50:06    12-Jan-2020 00:
    "PRN:4"      "Ground station 32"            2           12-Jan-2020 04:54:02    12-Jan-2020 07:
    "PRN:5"      "Ground station 32"            1           12-Jan-2020 05:52:03    12-Jan-2020 07:
    "PRN:6"      "Ground station 32"            1           12-Jan-2020 02:43:29    12-Jan-2020 07:
    "PRN:7"      "Ground station 32"            1           11-Jan-2020 21:09:52    12-Jan-2020 03:
    "PRN:8"      "Ground station 32"            1           11-Jan-2020 20:33:36    12-Jan-2020 03:
    "PRN:9"      "Ground station 32"            1           11-Jan-2020 19:50:06    12-Jan-2020 00:
    "PRN:9"      "Ground station 32"            2           12-Jan-2020 05:08:32    12-Jan-2020 07:
    "PRN:10"     "Ground station 32"            1           12-Jan-2020 00:32:56    12-Jan-2020 01:
    "PRN:11"     "Ground station 32"            1           11-Jan-2020 22:15:09    12-Jan-2020 04:
    "PRN:12"     "Ground station 32"            1           12-Jan-2020 04:32:16    12-Jan-2020 07:
    "PRN:13"     "Ground station 32"            1           12-Jan-2020 00:03:56    12-Jan-2020 02:
      ⋮
```

Visualize the simulation results.

```
v = satelliteScenarioViewer(sc,'ShowDetails',false);
play(sc);
```


Satellite Scenario Viewer
50x
Jan 12 2020
00:02:24 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
20:00:00 UTC
Jan 12 2020 00:00:00 UTC
Jan 12 2020 04:00:00 UTC
Jan 12 20

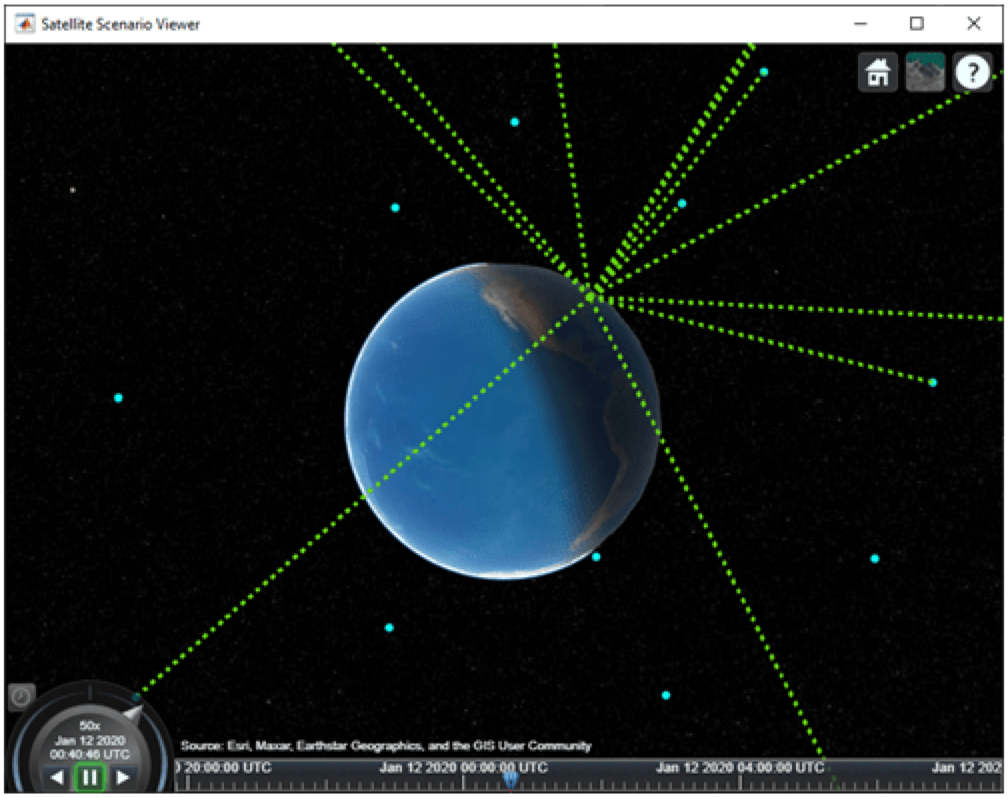
Verify that the access intervals are the same when you set the `AutoSimulate` property to `true`.

```
sc.AutoSimulate = true;
intvls2 = accessIntervals(ac)
```

```
intvls2=35×8 table
     Source           Target          IntervalNumber         StartTime                EndTime
    _________    __________________    ______________    ____________________    _____________

    "PRN:1"      "Ground station 32"         1           11-Jan-2020 23:20:25    12-Jan-2020 05:
    "PRN:2"      "Ground station 32"         1           12-Jan-2020 04:03:16    12-Jan-2020 07:
    "PRN:3"      "Ground station 32"         1           11-Jan-2020 19:50:06    11-Jan-2020 21:
    "PRN:3"      "Ground station 32"         2           12-Jan-2020 01:52:43    12-Jan-2020 06:
    "PRN:4"      "Ground station 32"         1           11-Jan-2020 19:50:06    12-Jan-2020 00:
    "PRN:4"      "Ground station 32"         2           12-Jan-2020 04:54:02    12-Jan-2020 07:
    "PRN:5"      "Ground station 32"         1           12-Jan-2020 05:52:03    12-Jan-2020 07:
    "PRN:6"      "Ground station 32"         1           12-Jan-2020 02:43:29    12-Jan-2020 07:
    "PRN:7"      "Ground station 32"         1           11-Jan-2020 21:09:52    12-Jan-2020 03:
    "PRN:8"      "Ground station 32"         1           11-Jan-2020 20:33:36    12-Jan-2020 03:
    "PRN:9"      "Ground station 32"         1           11-Jan-2020 19:50:06    12-Jan-2020 00:
    "PRN:9"      "Ground station 32"         2           12-Jan-2020 05:08:32    12-Jan-2020 07:
    "PRN:10"     "Ground station 32"         1           12-Jan-2020 00:32:56    12-Jan-2020 01:
    "PRN:11"     "Ground station 32"         1           11-Jan-2020 22:15:09    12-Jan-2020 04:
    "PRN:12"     "Ground station 32"         1           12-Jan-2020 04:32:16    12-Jan-2020 07:
    "PRN:13"     "Ground station 32"         1           12-Jan-2020 00:03:56    12-Jan-2020 02:
       ⋮
```

Visualize the scenario.

```
play(sc);
```



Input Arguments

sc — Satellite scenario
satelliteScenario object

Satellite scenario, specified as a `satelliteScenario` object. The argument applies only if the `AutoSimulate` property of the `sc` object is `false`.

Output Arguments

isrunning — Running status of satellite scenario simulation
true or 1 | false or 0

Running status of the satellite scenario simulation, returned as a logical 1 (`true`) or 0 (`false`). The `isrunning` value is `true` until the scenario reaches the specified `StopTime` value.

Version History

Introduced in R2022a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`play` | `satellite` | `groundStation` | `restart`

restart

Restart simulation from beginning

Syntax

```
restart(sc)
```

Description

`restart(sc)` resets the satellite scenario `sc` to the initial start time.

Examples

Manual Simulation of Satellite Scenario

Create a satellite scenario object and set the `AutoSimulate` property to `false` to enable manual simulation of the satellite scenario.

```
startTime = datetime(2022,1,12);
stopTime = startTime + days(0.5);
sampleTime = 60;                             % Seconds
sc = satelliteScenario('AutoSimulate',false);
```

Add a GPS satellite constellation to the scenario.

```
sat = satellite(sc,"gpsAlmanac.txt");
```

Simulate the scenario using the `advance` function.

```
while advance(sc)
end
```

Obtain the satellite position histories.

```
p = states(sat);
```

`AutoSimulate` is `false`, so restart the scenario before adding a ground station.

```
restart(sc);
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Add access analysis between each satellite and ground station.

```
ac = access(sat,gs);
```

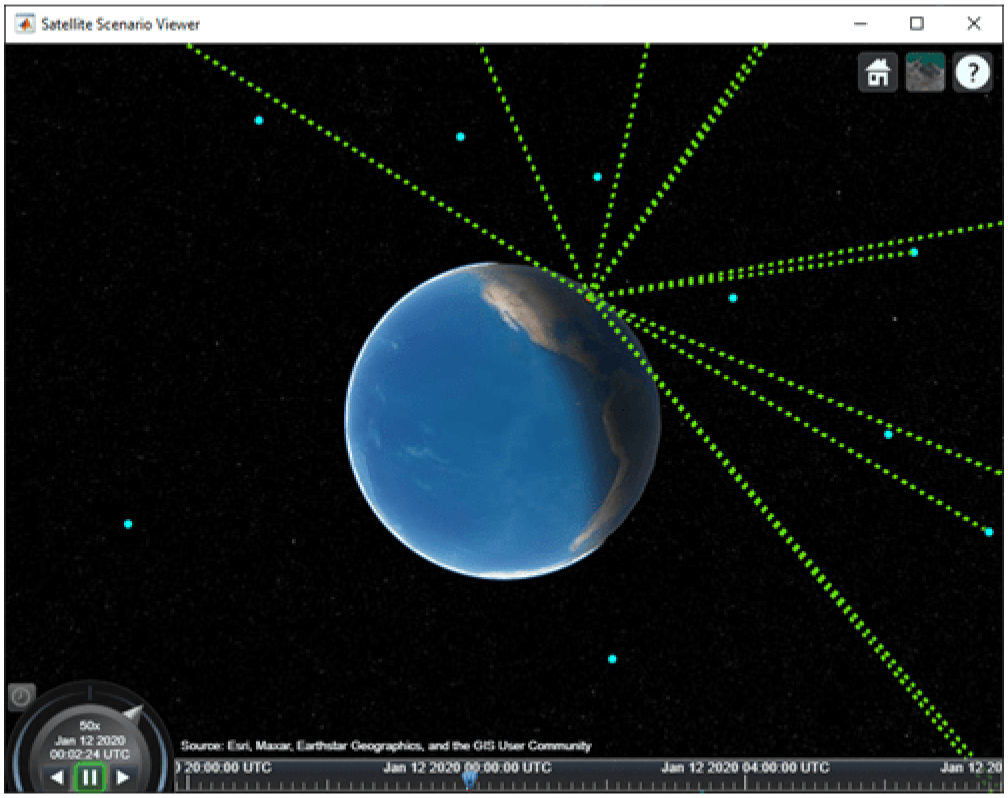
Simulate the scenario and determine the access intervals.

```
while advance(sc)
end
intvls1 = accessIntervals(ac)
```

```
intvls1=35×8 table
      Source              Target             IntervalNumber          StartTime                  EndTime
    _________    _____________________    ______________    ______________________    _________________

    "PRN:1"      "Ground station 32"            1           11-Jan-2020 23:20:25      12-Jan-2020 05:1
    "PRN:2"      "Ground station 32"            1           12-Jan-2020 04:03:16      12-Jan-2020 07:4
    "PRN:3"      "Ground station 32"            1           11-Jan-2020 19:50:06      11-Jan-2020 21:3
    "PRN:3"      "Ground station 32"            2           12-Jan-2020 01:52:43      12-Jan-2020 06:4
    "PRN:4"      "Ground station 32"            1           11-Jan-2020 19:50:06      12-Jan-2020 00:1
    "PRN:4"      "Ground station 32"            2           12-Jan-2020 04:54:02      12-Jan-2020 07:4
    "PRN:5"      "Ground station 32"            1           12-Jan-2020 05:52:03      12-Jan-2020 07:4
    "PRN:6"      "Ground station 32"            1           12-Jan-2020 02:43:29      12-Jan-2020 07:4
    "PRN:7"      "Ground station 32"            1           11-Jan-2020 21:09:52      12-Jan-2020 03:2
    "PRN:8"      "Ground station 32"            1           11-Jan-2020 20:33:36      12-Jan-2020 03:1
    "PRN:9"      "Ground station 32"            1           11-Jan-2020 19:50:06      12-Jan-2020 00:4
    "PRN:9"      "Ground station 32"            2           12-Jan-2020 05:08:32      12-Jan-2020 07:4
    "PRN:10"     "Ground station 32"            1           12-Jan-2020 00:32:56      12-Jan-2020 01:5
    "PRN:11"     "Ground station 32"            1           11-Jan-2020 22:15:09      12-Jan-2020 04:3
    "PRN:12"     "Ground station 32"            1           12-Jan-2020 04:32:16      12-Jan-2020 07:4
    "PRN:13"     "Ground station 32"            1           12-Jan-2020 00:03:56      12-Jan-2020 02:5
       ⋮
```

Visualize the simulation results.

```
v = satelliteScenarioViewer(sc,'ShowDetails',false);
play(sc);
```


Satellite Scenario Viewer
50x
Jan 12 2020
00:02:24 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
20:00:00 UTC
Jan 12 2020 00:00:00 UTC
Jan 12 2020 04:00:00 UTC
Jan 12 20

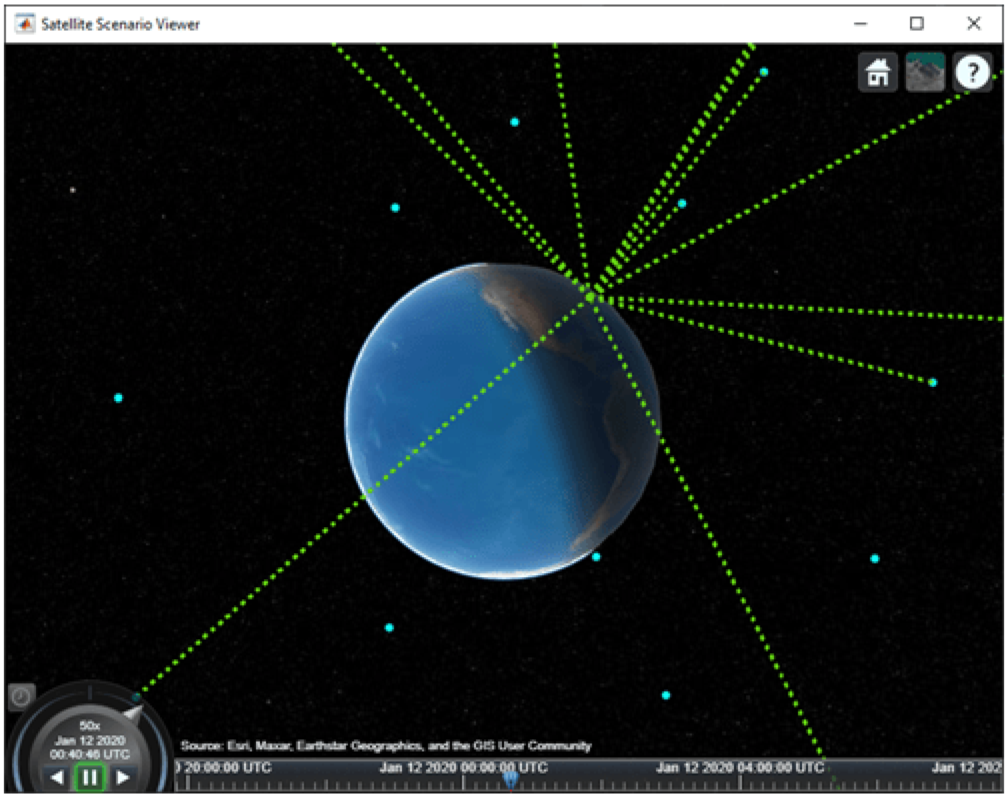
Verify that the access intervals are the same when you set the `AutoSimulate` property to `true`.

```
sc.AutoSimulate = true;
intvls2 = accessIntervals(ac)
```

```
intvls2=35×8 table
     Source           Target          IntervalNumber         StartTime                EndTime
    ________    ____________________    ______________    ____________________    ________________

    "PRN:1"     "Ground station 32"           1           11-Jan-2020 23:20:25    12-Jan-2020 05:1
    "PRN:2"     "Ground station 32"           1           12-Jan-2020 04:03:16    12-Jan-2020 07:4
    "PRN:3"     "Ground station 32"           1           11-Jan-2020 19:50:06    11-Jan-2020 21:3
    "PRN:3"     "Ground station 32"           2           12-Jan-2020 01:52:43    12-Jan-2020 06:4
    "PRN:4"     "Ground station 32"           1           11-Jan-2020 19:50:06    12-Jan-2020 00:1
    "PRN:4"     "Ground station 32"           2           12-Jan-2020 04:54:02    12-Jan-2020 07:4
    "PRN:5"     "Ground station 32"           1           12-Jan-2020 05:52:03    12-Jan-2020 07:4
    "PRN:6"     "Ground station 32"           1           12-Jan-2020 02:43:29    12-Jan-2020 07:4
    "PRN:7"     "Ground station 32"           1           11-Jan-2020 21:09:52    12-Jan-2020 03:2
    "PRN:8"     "Ground station 32"           1           11-Jan-2020 20:33:36    12-Jan-2020 03:1
    "PRN:9"     "Ground station 32"           1           11-Jan-2020 19:50:06    12-Jan-2020 00:4
    "PRN:9"     "Ground station 32"           2           12-Jan-2020 05:08:32    12-Jan-2020 07:4
    "PRN:10"    "Ground station 32"           1           12-Jan-2020 00:32:56    12-Jan-2020 01:5
    "PRN:11"    "Ground station 32"           1           11-Jan-2020 22:15:09    12-Jan-2020 04:3
    "PRN:12"    "Ground station 32"           1           12-Jan-2020 04:32:16    12-Jan-2020 07:4
    "PRN:13"    "Ground station 32"           1           12-Jan-2020 00:03:56    12-Jan-2020 02:5
      ⋮
```

Visualize the scenario.

```
play(sc);
```



Input Arguments

`sc` — Satellite scenario
`satelliteScenario` object

Satellite scenario, specified as a `satelliteScenario` object. The argument applies only if the `AutoSimulate` property of the `sc` object is `false`.

The timeline and playback widgets on the open satellite scenario viewers that were previously made available after calling the `play` function become unavailable for interaction.

Version History

Introduced in R2022a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`play` | `satellite` | `groundStation` | `advance`

sigstrength

Package: `satcom.satellitescenario`

Calculate received signal strength at last node of link

Syntax

```
PISO = sigstrength(lnk)
[PISO,PRI] = sigstrength(lnk)
[ ___ ] = sigstrength(lnk,timeIn)
[PISO,PRI,timeOut] = sigstrength( ___ )
```

Description

`PISO = sigstrength(lnk)` returns a history matrix of received isotropic power in dBW at the final node in each link defined in the vector, `lnk`. The rows of the matrix correspond to the link object in `lnk` and the columns correspond to the time sample.

`[PISO,PRI] = sigstrength(lnk)` returns received isotropic power, `PISO`, and history matrix of the power at receiver input, `PRI`, in dBW at the final node in each link defined in the vector, `lnk`. The rows of the matrix correspond to the link object in `lnk` and the columns correspond to the time sample.

`[ ___ ] = sigstrength(lnk,timeIn)` returns one or both of the outputs as column vectors at the specified datetime `timeIn`. The elements in the vectors correspond to the links in `lnk`. If no time zone is specified in `timeIn`, the time zone is assumed to be Coordinated Universal Time (UTC).

`[PISO,PRI,timeOut] = sigstrength( ___ )` returns the received isotropic power, power at the receiver input, and the corresponding datetime in UTC.

Examples

Obtain Signal Strength of Satellite Link

This example computes the history of received isotropic power, power at receiver input, and the corresponding time samples of the satellite link.

Create a satellite scenario object.

```
startTime = datetime(2020,10,13,10,42,0);
stopTime = datetime(2020,10,13,10,50,0);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                         % meters
eccentricity = 0;
inclination = 0;                                                  % degrees
rightAscensionOfAscendingNode = 0;                                % degrees
```

```
argumentOfPeriapsis = 0;                                        % degrees
trueAnomaly = 210;                                              % degrees
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

Add a transmitter to the satellite.

```
tx = transmitter(sat);
```

Add a ground station to the scenario.

```
latitude = 0;                            % degrees
longitude = 30;                          % degrees
gs = groundStation(sc,latitude,longitude);
```

Add a receiver to the ground station.

```
rx = receiver(gs,MountingAngles=[0; 180; 0]);
```

Add a link analysis to the transmitter.

```
lnk = link(tx,rx);
```

Obtain the history of received isotropic power, power at receiver input, and time samples.

```
[PISO,PRI,timeOut] = sigstrength(lnk)
```

```
PISO = 1×9
10^3 ×

   -0.2770   -0.1509   -0.1461   -0.2630   -0.4950   -0.8299   -1.2512   -1.7398   -2.2756


PRI = 1×9
10^3 ×

   -0.5934   -0.1562   -0.1397   -0.5446   -1.3544   -2.5375   -4.0507      -Inf      -Inf


timeOut = 1x9 datetime
   13-Oct-2020 10:42:00   13-Oct-2020 10:43:00   13-Oct-2020 10:44:00   13-Oct-2020 10:45:00   13
```

Obtain Signal Strength of Satellite Link at Specified Time

This example computes the received isotropic power and the power at receiver input of the satellite link at a specified time.

Create a satellite scenario object.

```
startTime = datetime(2021,10,13,10,5,0);
stopTime = startTime + days(1);
sampleTime = 60;                                    % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                       % meters
eccentricity = 0;
inclination = 0;                                                % degrees
rightAscensionOfAscendingNode = 0;                              % degrees
argumentOfPeriapsis = 0;                                        % degrees
trueAnomaly = 210;                                              % degrees
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

Add a receiver to the satellite.

```
rx = receiver(sat);
```

Add a ground station to the scenario.

```
latitude = 0;                           % degrees
longitude = 30;                         % degrees
gs = groundStation(sc,latitude,longitude);
```

Add a transmitter to the ground station.

```
tx = transmitter(gs,MountingAngles=[0; 180; 0]);
```

Add a link analysis to the transmitter.

```
lnk = link(tx,rx);
```

Obtain the received isotropic power and power at receriver input on 13 October 2021 at 10:06 AM UTC.

```
time = datetime(2021,10,13,10,6,0);
[PISO,PRI] = sigstrength(lnk,time)
```

```
PISO = -2.9142e+03

PRI = -3.9716e+03
```

Input Arguments

lnk — Link analysis object
vector

Link analysis object, specified as a vector.

timeIn — Time at which output is calculated
`datetime` scalar

Time at which the output is calculated, specified as a `datetime` scalar. If no time zone is specified in `timeIn`, the time zone is assumed to be UTC.

Output Arguments

PISO — Received isotropic power
matrix

Received isotropic power in dBW, returned as a matrix. This output is the power level just before the receiver antenna, after accounting for the transmitter effective isotropic radiated power and path loss. The rows of the matrix correspond to the link object in `lnk` and the columns correspond to the time sample.

When the `AutoSimulate` property of the satellite scenario is `true`, the received isotropic power history from `StartTime` to `StopTime` is returned. Otherwise, the received isotropic power history from `StartTime` to `SimulationTime` is returned.

`PRI` — Power at receiver input
matrix

Power at receiver input in dBW, returned as a matrix. This output is the power level after adding the receiver antenna gain in the direction of the received signal to `PISO` and subtracting `PreReceiverLoss` of the receiver. The rows of the matrix correspond to the link object in `lnk` and the columns correspond to the time sample.

When the `AutoSimulate` property of the satellite scenario is `true`, the history of power at the receiver input from `StartTime` to `StopTime` is returned. Otherwise, the history of power at the receiver input from `StartTime` to `SimulationTime` is returned.

`timeOut` — Time samples of received isotropic power and power at receiver input
datetime scalar | datetime row vector

Time samples of received isotropic power, `PISO`, and power at receiver input, `PRI`, returned as a datetime scalar. If time histories of `PISO` and `PRI` are returned, `timeOut` is a datetime row vector.

Version History

Introduced in R2022b

See Also

Objects
`satelliteScenario`

Functions
`link` | `ebno`

Topics
“Satellite Scenario Key Concepts”

coordinateAxes

Package: `matlabshared.satellitescenario`

Visualize coordinate axes triad of satellite scenario assets

Syntax

```
coordinateAxes(asset)
coordinateAxes(asset,Name=Value)
ax = coordinateAxes( ___ )
```

Description

`coordinateAxes(asset)` adds a coordinate axes triad graphic visualization (`CoordinateAxes` object) for an asset. In the visualization, the *x*-axis (roll) is red in color, the *y*-axis (pitch) is green, and *z*-axis (yaw) is blue.

`coordinateAxes(asset,Name=Value)` specifies additional arguments using one or more name-value.

`ax = coordinateAxes( ___ )` returns the `CoordinateAxes` object.

Examples

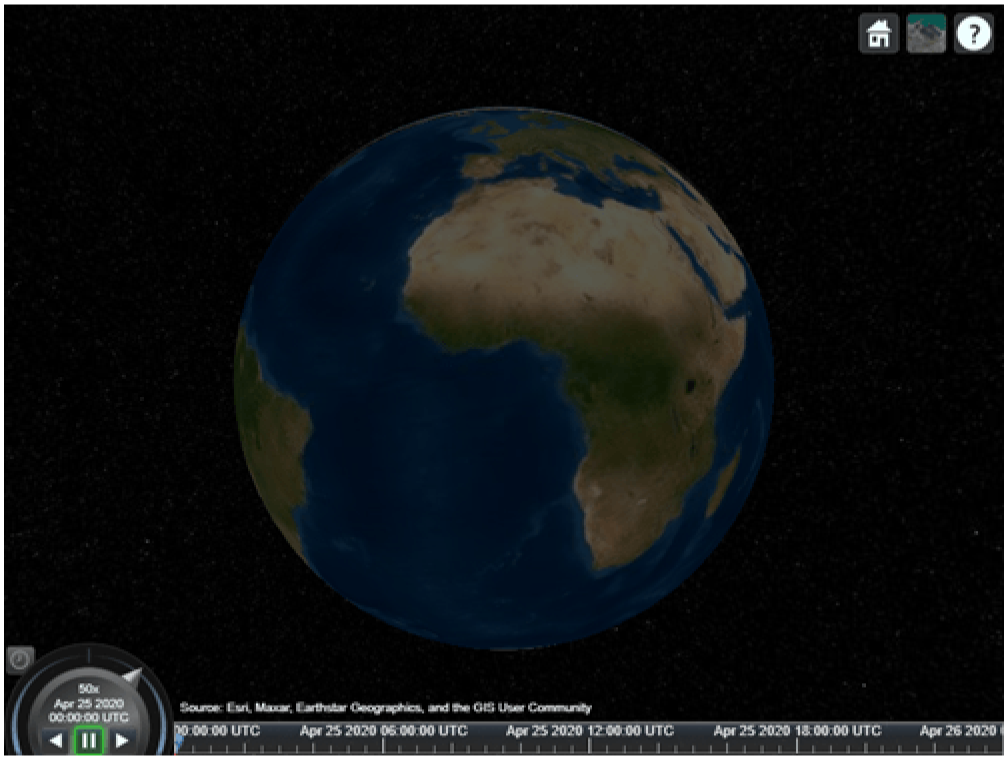
Visualize Coordinate Axes Triad of Satellite

Create a `satelliteScenario` object.

```
startTime = datetime(2020,4,25); % April 25, 2020, 12:00 AM UTC
stopTime = datetime(2020,4,26);  % April 26, 2020, 12:00 AM UTC
sampleTime = 60;                 % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

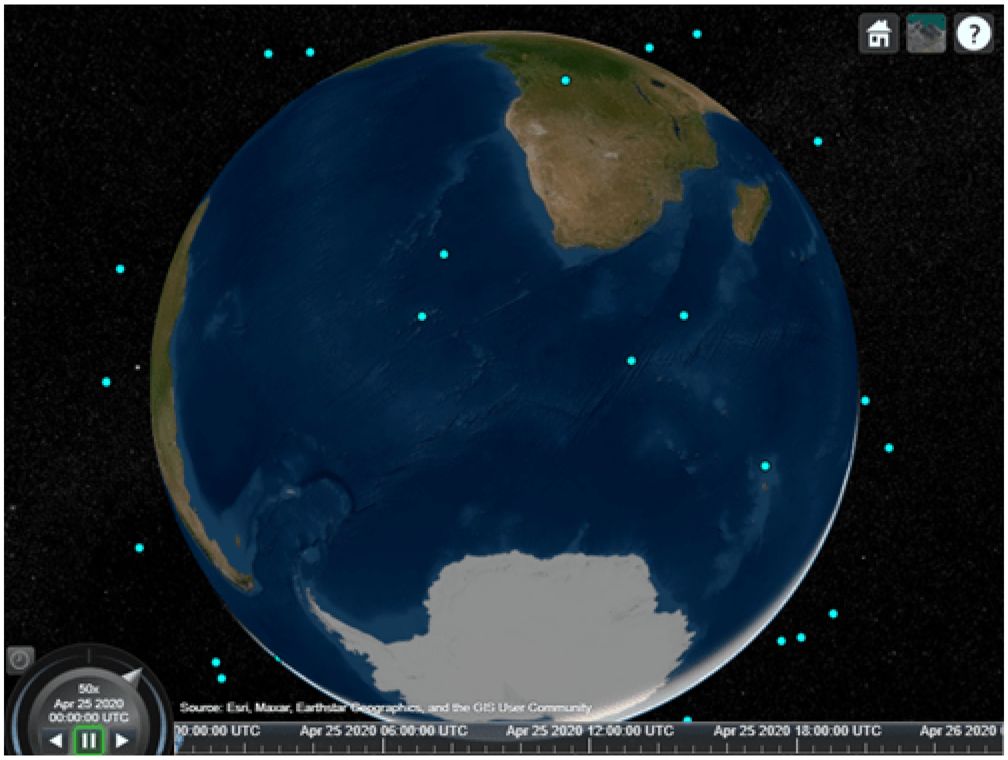
Launch a satellite scenario viewer with `ShowDetails` set to false.

```
v = satelliteScenarioViewer(sc,ShowDetails=false);
```



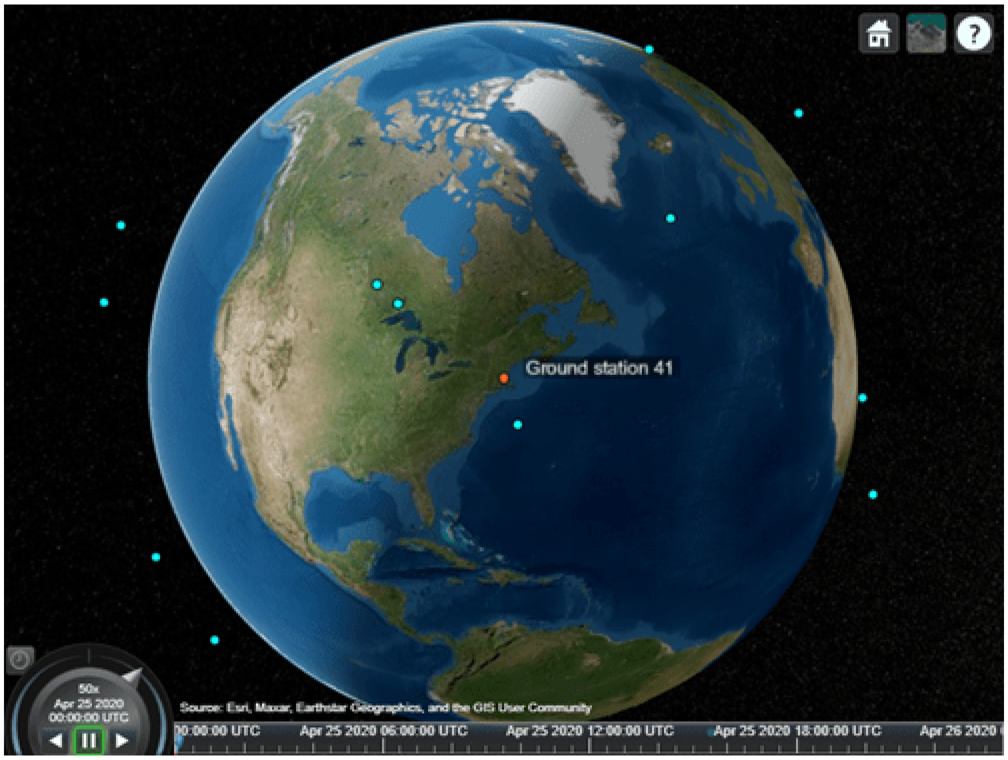
Add a low-earth orbit satellite constellation to the scenario from a TLE file.

```
tleFile = "leoSatelliteConstellation.tle";
sat = satellite(sc,tleFile);
```



Add a ground station to the scenario.

```
gs = groundStation(sc);
gs.ShowLabel = true;
```



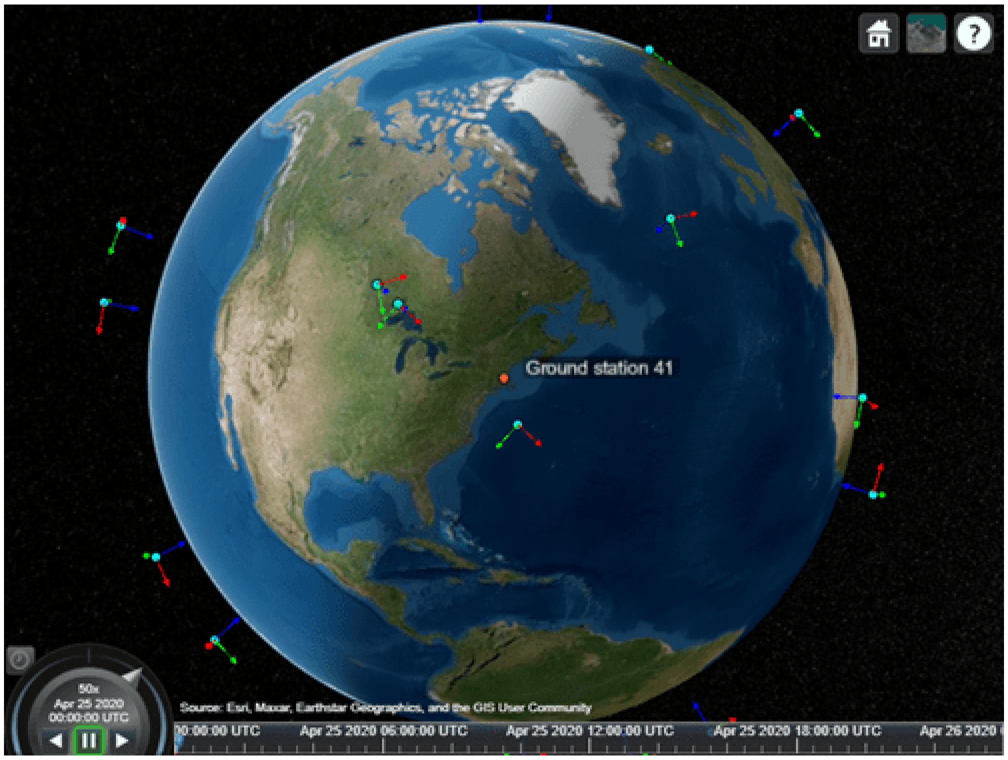
Visualize the coordinate axes triads of the satellites.

```
ax = coordinateAxes(sat)
```

```
ax=1×40 object
  1×40 CoordinateAxes array with properties:

    Scale
    VisibilityMode
```

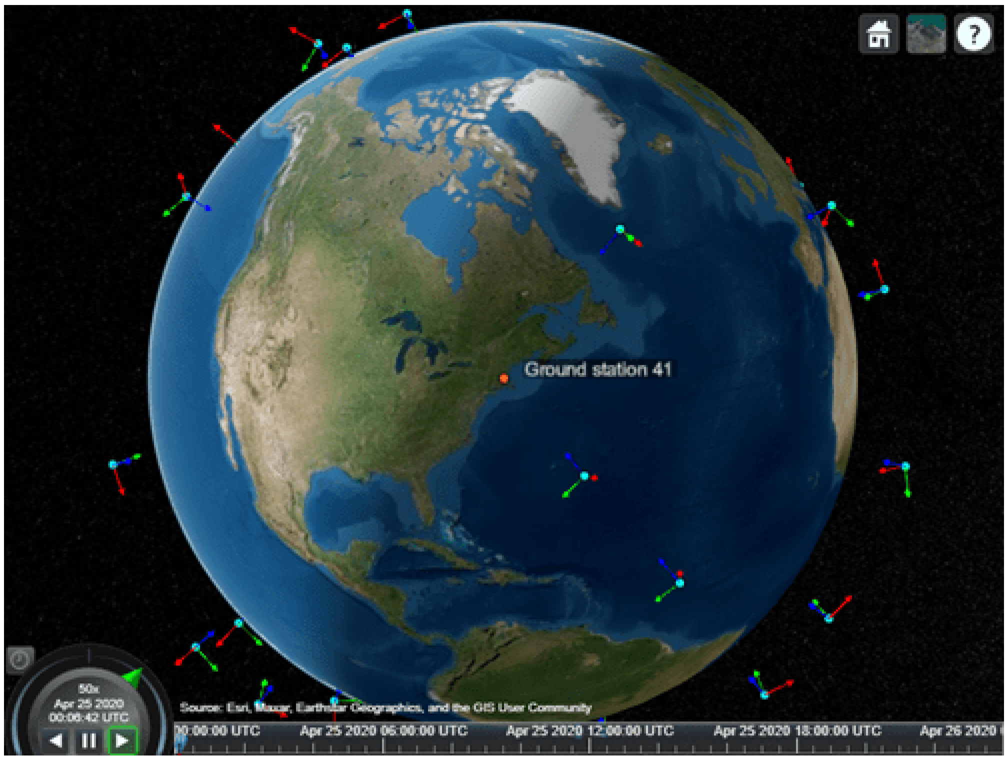
Point the satellites towards the ground station. The coordinate axes triad object updates to represent the new direction of the satellites.

```
pointAt(sat,gs);
```



Play the scenario.

```
play(sc);
```



Input Arguments

`asset` — Satellites, ground stations, gimbals, conical sensors, transmitters, or receivers
vector

Satellites, ground stations, gimbals, conical sensors, transmitters, or receivers, specified as a vector.

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.

Example: `Scale =2` scales the size of the coordinate axes triad visualization by 2.

`Viewer` — Satellite scenario viewer
vector

Satellite scenario viewer, specified as a vector. You view the coordinate axes triad in the viewer specified in this argument.

`Scale` — Scale of coordinate axes triad graphic
`1` (default) | positive scalar

Scale of the coordinate axes triad graphic, specified as a positive scalar.

Data Types: `double`

Output Arguments

ax — Coordinate axes triad
scalar | vector

Coordinate axes triad, returned as an object.

Version History

Introduced in R2023a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`play` | `show` | `groundStation` | `access` | `link` | `receiver` | `hide` | `pattern`

Topics
“Model, Visualize, and Analyze Satellite Scenario”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

dopplershift

Package: `matlabshared.satellitescenario`

Calculate Doppler shift at target asset in satellite scenario

Syntax

```
shift = dopplershift(source,target)
[shift,timeOut] = dopplershift(source,target)
[shift,timeOut,info] = dopplershift(source,target)
[ ___ ] = dopplershift( ___ ,timeIn)
[ ___ ] = dopplershift( ___ ,Frequency=freq)
```

Description

`shift = dopplershift(source,target)` calculates the history of Doppler shifts at the `target` asset corresponding to the `source` asset. Both `source` and `target` must belong to the same `satelliteScenario` object. If the value of `AutoSimulate` property of the `satelliteScenario` object is `true`, the `dopplershift` function returns the Doppler shift history from the value of `StartTime` to the value of `StopTime`. Otherwise, it returns the Doppler shift history from the value of `StartTime` to the value of `SimulationTime`.

`[shift,timeOut] = dopplershift(source,target)` returns the sample time history `timeOut`.

`[shift,timeOut,info] = dopplershift(source,target)` returns a structure, `info`, containing the Doppler rate and relative velocity information.

`[ ___ ] = dopplershift( ___ ,timeIn)` calculates Doppler shift at the specified datetime `timeIn`. When using this syntax, `dopplershift` sets the second dimension of `shift`, `timeOut`, and the fields of `info` to 1. In this case, the `DopplerRate` field of `info` is empty.

`[ ___ ] = dopplershift( ___ ,Frequency=freq)` additionally specifies one or more carrier frequencies at which to calculate Doppler shift.

Note The `dopplershift` function uses the Geocentric Celestial Reference Frame (GCRF) coordinate system to perform all calculations.

Examples

Determine Doppler Shift, Doppler Rate, Relative Velocity, and Time History

Determine the Doppler shift, the Doppler rate, the relative velocity, and the corresponding time history at a ground station. The source is a satellite that transmits on a carrier frequency of 14 GHz.

Specify the start time and the stop time of the satellite scenario.

```
startTime = datetime(2021,4,25);                   % April 25, 2021, 12:00 AM UTC
stopTime = datetime(2021,4,26);                    % April 26, 2021, 12:00 AM UTC
```

Set the sample time in seconds.

```
sampleTime = 60;
```

Create a `satelliteScenario` object.

```
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;               % In meters
eccentricity = 0;
inclination = 0;                        % In degrees
rightAscensionOfAscendingNode = 0;      % In degrees
argumentOfPeriapsis = 0;                % In degrees
trueAnomaly = 0;                        % In degrees
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

Add a default ground station to the scenario.

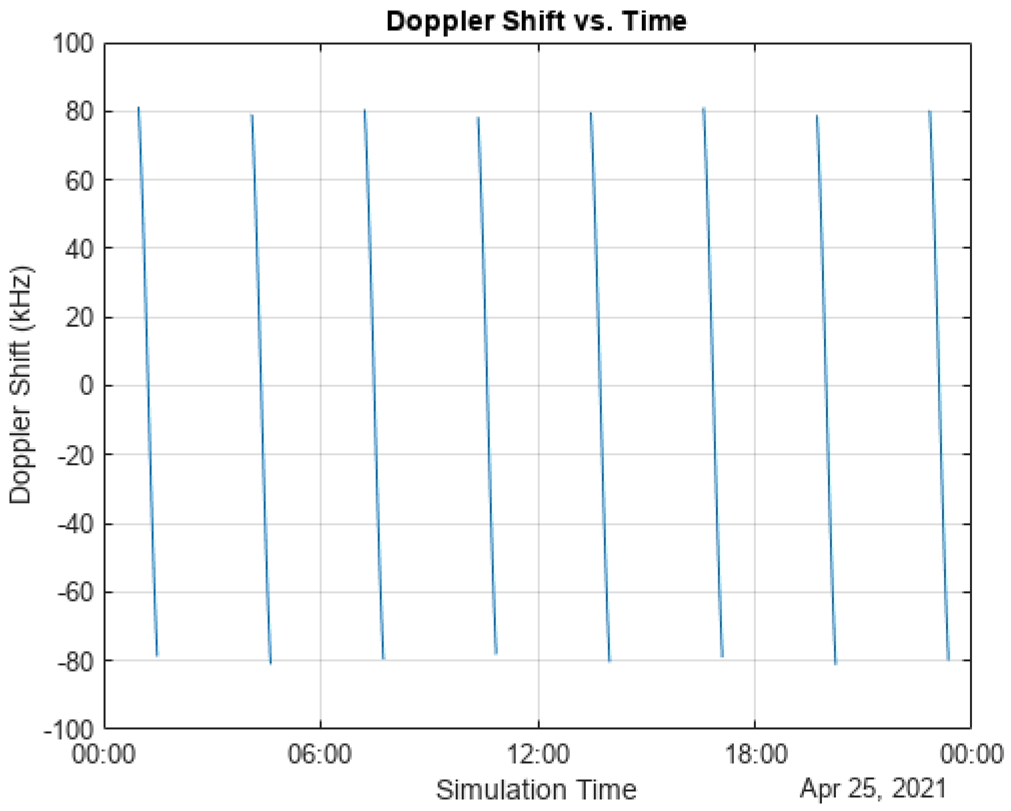
```
gs = groundStation(sc);
```

Determine the Doppler shift, the Doppler rate, and the relative velocity, as well as the corresponding time history.

```
carrierFrequency=14e9;
[frequencyShift,timeOut,dopplerInfo] = dopplershift(sat,gs,Frequency=carrierFrequency);
frequencyRate = dopplerInfo.DopplerRate;
relativeVelocity = dopplerInfo.RelativeVelocity;
```

Plot the Doppler shift at the ground station.

```
plot(timeOut,frequencyShift(1,:)/1e3)       % Doppler shift in kilohertz (kHz)
xlim([timeOut(1) timeOut(end)])
title("Doppler Shift vs. Time")
xlabel("Simulation Time")
ylabel("Doppler Shift (kHz)")
grid on
```



Input Arguments

`source` — Source asset from scenario
`Satellite` object | `GroundStation` object | `Transmitter` object | array of `Satellite` objects | array of `GroundStation` objects | array of `Transmitter` objects

Source asset from scenario, specified as a `Satellite` object, `GroundStation` object, `Transmitter` object, array of `Satellite` objects, array of `GroundStation` objects, or array of `Transmitter` objects.

`target` — Target asset from scenario
`Satellite` object | `GroundStation` object | `Receiver` object | array of `Satellite` objects | array of `GroundStation` objects | array of `Receiver` objects

Target asset from scenario, specified as a `Satellite` object, `GroundStation` object, `Receiver` object, array of `Satellite` objects, array of `GroundStation` objects, or array of `Receiver` objects.

`timeIn` — Time at which function calculates Doppler shift
datetime scalar

Time at which the function calculates the Doppler shift, specified as a datetime scalar. If `timeIn` does not specify a time zone, the function uses Coordinated Universal Time (UTC).

Data Types: `datetime`

`freq` — Carrier frequency

-1 (default) | scalar | vector

Carrier frequency at which to calculate Doppler shift, specified as a scalar or vector of nonnegative real numbers, in hertz or `-1`. Specify `-1` to use the default carrier frequency value for each source asset.

If the `freq` value is `-1` and `source` is a `Satellite` or `GroundStation`, the carrier frequency is 14 gigahertz (GHz) . If the `freq` value is `-1` and `source` is a `Transmitter`, the carrier frequency is the `Frequency` property value of the `Transmitter` object.

Data Types: `double`

Output Arguments

`shift` — History of Doppler shift

scalar | vector | matrix | 3-D array

History of Doppler shift, in hertz, returned as a scalar, vector, matrix, or 3-D array. When the `source` and `target` cannot access each other, the corresponding `shift` value is `NaN`. These tables show how this output changes based on the different input argument sizes and combinations.

- `timeIn` not specified

`source`	`target`	`shift`	Description
scalar	scalar	row vector	Doppler shift history, in which each element represents the Doppler shift between `source` and `target` at a specific time sample.
scalar	vector	matrix	Each row represents the Doppler shift history between source and an asset in `target`. The row corresponds to the position of the asset in `target`.
vector	scalar	matrix	Each row represents the Doppler shift history between `target` and an asset in `source`. The row corresponds to the position of the asset in `source`.

vector	vector	matrix	Each row represents the Doppler shift history between a pair of `source` and `target` assets. The row corresponds to the position of the assets in `source` and `target`.

- `timeIn` specified

`source`	`target`	`shift`	Description
scalar	scalar	scalar	Doppler shift at the specified datetime `timeIn`.
scalar	vector	column vector	Each element represents the Doppler shift between `source` and the corresponding asset in `target` at the specified datetime `timeIn`.
vector	scalar	column vector	Each element represents the Doppler shift from the corresponding asset in `source` to `target` at the specified datetime `timeIn`.
vector	vector	column vector	Each element represents the Doppler shift between the corresponding pair of `source` and `target` assets at the specified datetime `timeIn`.

If you specify freq as a vector, shift has a third dimension in which each page is the output of the corresponding combination of `source`, `target`, and `timeIn` from the previous tables for the corresponding carrier frequency in `freq`.

Data Types: `double`

`timeOut` — Time samples between start and stop time of scenario
datetime vector | datetime scalar

Time samples between the start and stop time of the scenario, returned as a scalar or vector.

If `timeIn` does not specify a time zone and the `AutoSimulate` property the `satelliteScenario` object is `true`, the `dopplershift` function returns the time sample history from `StartTime` to `StopTime`. Otherwise, it returns the time sample history from the value of `StartTime` to the value of `SimulationTime`.

Data Types: `datetime`

`info` — History of Doppler rate and relative velocity
structure

History of Doppler rate and relative velocity, returned as a structure that contains these fields.

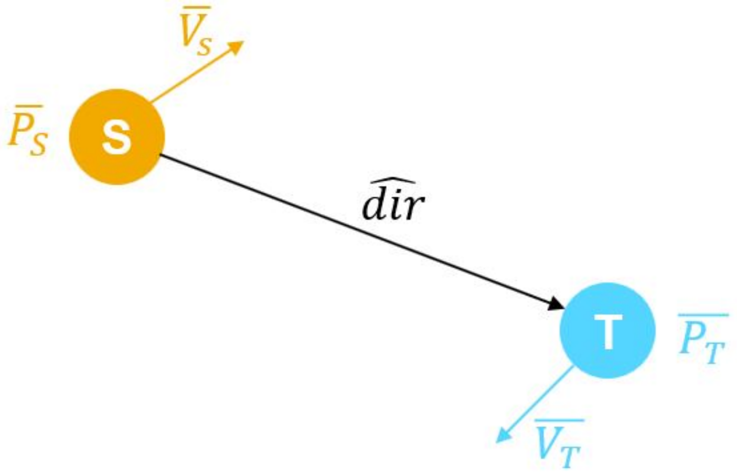
- `DopplerRate` — The Doppler rate history, returned as a vector, matrix, or 3-D array. The format of `DopplerRate` is similar to `shift` except that the number of columns equals the number of time samples minus 1. When `source` and `target` cannot see each other, the corresponding `DopplerRate` value is `NaN`.
- `RelativeVelocity` — The history of relative velocity of `source` and `target` in the direction from `source` to `target`, returned as a scalar, vector, or matrix. The number of rows in the matrix is the greater number between the `source` and `target` assets. When `source` and `target` cannot access each other, the `RelativeVelocity` value is `NaN`.

Data Types: `struct`

More About

Doppler Shift

If you know the position of the source ($\overline{P_S}$) , the position of the target ($\overline{P_T}$), the velocity of the source ($\overline{V_S}$), the velocity of the target ($\overline{V_T}$), and the carrier frequency (f_c), use these equations to calculate Doppler shift.



1 The unit vector from the source to the target is $\widehat{dir} = \dfrac{\overline{P_T} - \overline{P_S}}{\|\overline{P_T} - \overline{P_S}\|}$, where $\|\|$ indicates the Euclidean norm.

2 The velocity of the source in the unit direction is

$$V_{\mathrm{S,dir}} = \overline{V_{\mathrm{S}}} \cdot \widehat{dir}.$$

If the angle between $\overline{V_{\mathrm{S}}}$ and $\widehat{dir}$, in degrees, is in the range [0, 90], $V_{\mathrm{S,dir}}$ is positive. A positive value of $V_{\mathrm{S,dir}}$ indicates the source is moving towards the target. A negative value of $V_{\mathrm{S,dir}}$ indicates the source is moving away from the target.

3 The velocity of the target in the unit direction is

$$V_{\mathrm{T,dir}} = \overline{V_{\mathrm{T}}} \cdot \widehat{dir}.$$

If the angle between $\overline{V_{\mathrm{T}}}$ and $\widehat{dir}$, in degrees, is in the range [0, 90], $V_{\mathrm{T,dir}}$ is positive. A positive value of $V_{\mathrm{T,dir}}$indicates the target is moving away from the source. A negative value of $V_{\mathrm{T,dir}}$ indicates the target is moving towards the source.

4 The relative velocity of source and target in the direction from the source to the target is

$$V_{\mathrm{rel}} = V_{\mathrm{S,dir}} - V_{\mathrm{T,dir}}.$$

5 The Doppler frequency observed at the target is

$$f_{\mathrm{d}} = \left(\frac{c - V_{\mathrm{T,dir}}}{c - V_{\mathrm{S,dir}}}\right) f_{\mathrm{c}}.$$

In this equation, c is the speed of light in meters per second.

6 The Doppler shift at the target is

$$f_{\mathrm{shift}} = f_{\mathrm{d}} - f_{\mathrm{c}} = \left(\frac{V_{\mathrm{rel}}}{c - V_{\mathrm{S,dir}}}\right) f_{\mathrm{c}}.$$

Note All position and velocity vectors are in the GCRF coordinate frame.

Version History

Introduced in R2023a

See Also

Objects

`satelliteScenario` | `Satellite` | `GroundStation` | `Receiver` | `Transmitter`

Functions

`satellite` | `groundStation` | `transmitter` | `receiver`

Topics

"Calculate Latency and Doppler in a Satellite Scenario"

latency

Package: `matlabshared.satellitescenario`

Calculate propagation delay from one asset to another asset

Syntax

```
delay = latency(source,target)
[delay,timeOut] = latency(source,target)
[ ___ ] = latency(source,target,timeIn)
```

Description

`delay = latency(source,target)` calculates the history of propagation delays from the `source` asset to the `target` asset. The `source` and `target` assets must belong to the same `satelliteScenario` object. If the value of the `AutoSimulate` property of the `satelliteScenario` object is `true`, the `latency` function returns the propagation delay history from the value of `StartTime` to the value of `StopTime`. Otherwise, it returns the propagation delay history from `StartTime` to `SimulationTime`.

`[delay,timeOut] = latency(source,target)` returns the history of time samples `timeOut`.

`[ ___ ] = latency(source,target,timeIn)` calculates latency at the specified datetime `timeIn`. When using this syntax, the `latency` function sets the second dimension of `delay` and `timeOut` to `1`.

Note When the `target` asset is a `Satellite` object, the `latency` function uses a numerical iterative solution to compute the propagation delay. When the `target` asset is a `GroundStation` object, the `latency` function applies Sagnac correction to compute the propagation delay.

Examples

Determine Propagation Delay

Determine the history of propagation delay from a satellite to a ground station. Also, determine the corresponding time samples.

Create a `satelliteScenario` object.

```
startTime = datetime(2022,8,21);                      % August 21, 2022, 12:00 AM UTC
stopTime = datetime(2022,8,23);                       % August 23, 2022, 12:00 AM UTC
sampleTime = 60;                                      % Units are in seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;             % In meters
eccentricity = 0;
```

```
inclination = 0;                           % In degrees
rightAscensionOfAscendingNode = 0;    % In degrees
argumentOfPeriapsis = 0;              % In degrees
trueAnomaly = 0;                           % In degrees
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode,argumentO
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Determine the propagation delay history between two assets and the corresponding time samples.

```
[delay,timeOut] = latency(sat,gs);
```

Input Arguments

source — Source asset from scenario
Satellite object | GroundStation object | array of Satellite objects | array of GroundStation objects

Source asset from scenario, specified as a `Satellite` object, `GroundStation` object, array of `Satellite` objects, or array of `GroundStation` objects.

target — Target asset from scenario
Satellite object | GroundStation object | array of Satellite objects | array of GroundStation objects

Target asset from scenario, specified as a `Satellite` object, `GroundStation` object, array of `Satellite` objects, or array of `GroundStation` objects.

timeIn — Time instance at which function calculates latency
datetime scalar

Time instance at which the function calculates latency, specified as a datetime scalar. If `timeIn` does not specify a time zone, the function uses Coordinated Universal Time (UTC).

Data Types: `datetime`

Output Arguments

delay — History of propagation delay
scalar | vector | matrix

Delay, in seconds, returned as a scalar, vector, or matrix. When `source` and `target` cannot access each other, the corresponding latency value is `NaN`. These tables show how this output changes based on the different input argument sizes and combinations.

- `timeIn` not specified

source	target	delay	Description
scalar	scalar	row vector	Latency history, in which each element represents the propagation delay from `source` to `target` at a specific time.
scalar	vector	matrix	Each row represents the propagation delay from `source` to `target`. The row index indicates the position of the asset in `target`.
vector	scalar	matrix	Each row represents the propagation delay from `source` to `target`. The row index indicates the position of the asset in `source`.
vector	vector	matrix	Each row represents the propagation delay from `source` to `target`. The row index indicates the position of the assets in `source` and `target`.

- `timeIn` specified

source	target	delay	Description
scalar	scalar	scalar	Latency at the specified datetime `timeIn`.
scalar	vector	column vector	Each element represents the propagation delay from `source` to the corresponding asset in `target` at the specified datetime `timeIn`.

`source`	`target`	`delay`	Description
vector	scalar	column vector	Each element represents the propagation delay from the corresponding asset in `source` to `target` at the specified datetime `timeIn`.
vector	vector	column vector	Each element represents the propagation delay between the corresponding pair of `source` and `target` assets at the specified datetime `timeIn`.

Data Types: `double`

`timeOut` — Time samples between start and stop time of scenario
datetime vector | datetime scalar

Time samples between the start and stop time of the scenario, returned as a scalar or vector.

If `timeIn` does not specify a time zone and the `AutoSimulate` property the `satelliteScenario` object is `true`, the `latency` function returns the time sample history from `StartTime` to `StopTime`. Otherwise, it returns the time sample history from the value of `StartTime` to the value of `SimulationTime`.

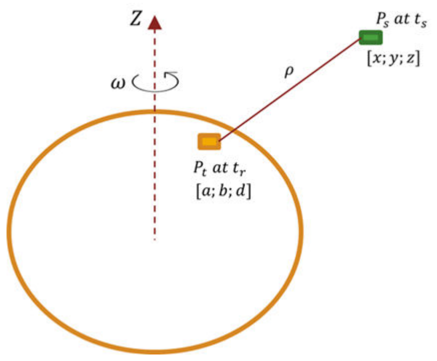
Data Types: `datetime`

More About

Latency When Ground Station Is Target

Consider a satellite scenario in which a satellite is the source and a ground station on Earth is the target. Because the satellite scenario currently supports only static ground stations, assume that the velocity of the ground station is the same as the angular velocity of Earth.

Given the instant at which the source transmits the signal t_{s} and the instant at which the target receives the signal t_{r}, the latency or propagation delay is the difference $t_{\mathrm{r}} - t_{\mathrm{s}}$. Assume that at every instant, the source transmits the signal. The unknown is the instant at which the signal reaches the target.



Consider that the positions of the source and the target are in International Terrestrial Reference Frame (ITRF).

- Source position P_s at t_s— $[x\ y\ z]'$
- Target position P_t at t_r — $[a, b, d]'$

The distance between the target position P_t and the source position P_s in the t_r time frame is the distance travelled by the signal ρ, where $\rho = \left\| P_t^{t_r} - P_s^{t_r} \right\|$.

To obtain the source position in the t_r time frame, apply this rotation matrix to the source position.

$$R_{t_s}^{t_r} = \begin{bmatrix} \cos(\omega(t_r - t_s)) & \sin(\omega(t_r - t_s)) & 0 \\ -\sin(\omega(t_r - t_s)) & \cos(\omega(t_r - t_s)) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In this equation, the time difference $t_r - t_s$ equals the time taken by the signal to cover the distance ρ.

$$\therefore t_r - t_s = \frac{\rho}{c}.$$

Substitute this value in the rotation matrix to obtain

$$R_{t_s}^{t_r} = \begin{bmatrix} \cos(\omega\frac{\rho}{c}) & \sin(\omega\frac{\rho}{c}) & 0 \\ -\sin(\omega\frac{\rho}{c}) & \cos(\omega\frac{\rho}{c}) & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

The position of source in the time frame t_r is

$$P_s^{t_r} = R_{t_s}^{t_r} P_s^{t_s}.$$

$$\therefore P_s^{t_r} = \begin{bmatrix} \cos(\omega\frac{\rho}{c}) & \sin(\omega\frac{\rho}{c}) & 0 \\ -\sin(\omega\frac{\rho}{c}) & \cos(\omega\frac{\rho}{c}) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

In this equation,

- ω is to the order of `1e-5`.
- ρ is to the order of `1e7`.
- c is to the order of `1e-8`.
- $\omega\frac{\rho}{c}$ is to the order of `1e-5`.

As $\omega\frac{\rho}{c}$ is very small, apply small angle approximation ($\sin(\theta) \approx \theta$ and $\cos(\theta) \approx 1$) to obtain

$$P_{\mathrm{s}}^{\mathrm{tr}} = \begin{bmatrix} 1 & \omega\frac{\rho}{c} & 0 \\ -\omega\frac{\rho}{c} & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

Substitute this value in $\rho = \left\| P_{\mathrm{t}}^{\mathrm{tr}} - P_{\mathrm{s}}^{\mathrm{tr}} \right\|$ to obtain

$$\rho = \sqrt{\left((x-a) + y\omega\frac{\rho}{c}\right)^2 + \left(-x\omega\frac{\rho}{c} + (y-b)\right)^2 + (z-d)^2}\,.$$

Square both sides of this equation to obtain

$$\rho^2 = \left((x-a)^2 + (y\omega\frac{\rho}{c})^2 + 2(x-a)(y\omega\frac{\rho}{c})\right) + \left((y-b)^2 + (x\omega\frac{\rho}{c})^2 - 2(y-b)(x\omega\frac{\rho}{c})\right) + (z-d)^2\,.$$

Rearrange the terms and simplify the equation to obtain

$$\rho^2 = (x-a)^2 + (y-b)^2 + (z-d)^2 + \left((\omega\frac{\rho}{c})^2(x^2+y^2) + 2\omega\frac{\rho}{c}(bx-ay)\right).$$

The true geometric range for the position of source and target in the t_{r} time frame is

$$r^2 = \left((a-x)^2 + (b-y)^2 + (d-z)^2\right).$$

Let $R_{\mathrm{corr}} = \frac{\omega}{c}(bx-ay)$ and $e_1{}' = \left(\frac{\omega}{c}\right)^2(x^2+y^2)$. Substitute these values in $\rho^2 = (x-a)^2 + (y-b)^2 + (z-d)^2 + \left((\omega\frac{\rho}{c})^2(x^2+y^2) + 2\omega\frac{\rho}{c}(bx-ay)\right)$ to obtain

$$\rho^2 = r^2 + (e_1{}'\rho^2 + 2\rho R_{\mathrm{corr}})\,.$$

$$\therefore (1 - e_1{}')\rho^2 - 2\rho R_{\mathrm{corr}} - r^2 = 0.$$

The discriminant of this quadratic equation is

$$discriminant = 4R_{\mathrm{corr}}^2 + 4(1 - e_1{}')r^2\,.$$

$$\therefore \rho = \frac{2R_{\mathrm{corr}} + \sqrt{discriminant}}{2(1 - e_1{}')}\,.$$

If $R_{\mathrm{corr}} \ll r\sqrt{(1 - e_1{}')}$, then you can express ρ as

$$\rho = \frac{2R_{\text{corr}} + 2r\sqrt{1 - e_1{}'}}{2(1 - e_1{}')} = \frac{R_{\text{corr}}}{(1 - e_1{}')} + \frac{r}{\sqrt{1 - e_1{}'}}\,.$$

Latency When Satellite Is Target

When the target is a satellite, you must consider the movement of a satellite to know when the satellite receives the signal.

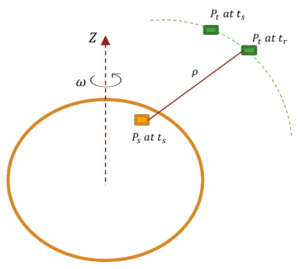
Let f be the function that defines the orbital motion of the satellite, which is nonlinear with no sharp peaks. Also, let P_{S} be the position of the source in Geocentric Celestial Reference Frame (GCRF). The time taken by the signal to reach the target equals the time the target takes to move from its position at the transmission time to its position at the reception time.

You can find the solution for the objective function iteratively by using the Newton-Raphson method.

$ObjectiveFunction(t) = \left(\frac{\|f(t_{\text{S}} + t) - P_{\text{S}}\|}{c}\right) - t$, where t_S is the source transmit time.

Assuming the satellite does not move, the initial guess to the function is the time the signal takes to reach the satellite.

The root of the objective function is latency. Obtaining the exact root is not always possible because of the numerical precision errors. Therefore, consider the value for which the objective function returns a value less than `2.2204e-16` as the root.



This methodology is applicable for any moving target, regardless of source movement.

Version History

Introduced in R2023a

See Also

Objects
`satelliteScenario`

Functions
`satellite` | `groundStation`

Topics
"Calculate Latency and Doppler in a Satellite Scenario"

pointAt

Package: `matlabshared.satellitescenario`

Point transmitter or receiver at target

Syntax

```
pointAt
pointAt(asset,"none")
pointAt(asset,coordinates)
pointAt(asset,target)
pointAt(asset,"nadir")
pointAt(asset,weights)
```

Description

`pointAt` specify the target at which the phased array antenna on a transmitter or a receiver must point. The antenna is electronically steered towards the desired target by computing the appropriate weights and assigning them to the array elements. The antenna is continuously steered electronically for the duration of the simulation to point at the desired target.

Note For regular antennas and default transmitter receivers, use `pointAt`.

`pointAt(asset,"none")` disables pointing for the transmitters or receivers specified by asset. All elements in asset must contain a phased array antenna. The phased array weights are set to the original values as they were at the time of creation of the transmitter or receiver. "none" is the default pointing mode.

`pointAt(asset,coordinates)` steers the phased array antennas in transmitters or receivers towards the geographical coordinates.

`pointAt(asset,target)` steers phased array antennas in transmitters or receivers towards the specified target.

`pointAt(asset,"nadir")` steers the phased array antennas in transmitters or receivers towards the nadir direction of their respective parent satellite, ground stations, or gimbals. The nadir direction is specified by latitude and longitude of the parent, and 0 m altitude.

`pointAt(asset,weights)` explicitly assigns the weights for the phased array antenna elements of the transmitters or receivers.

Input Arguments

`asset` — Transmitter or receivers
scalar | vector

Transmitter or receiver objects, specified as either a scalar or a vector.

`coordinates` — Geographical coordinates of transmitter or receiver
three-element vector | 2-D array

Geographical coordinates of the transmitter or receiver, specified as a three-element vector or a 2-D array.

- When the coordinates are a three-element vector, the elements of the vector correspond to the latitude, longitude, and altitude, in that order, and all satellites or gimbals are steered to point at this location.
- When the coordinates are a 2-D array, the number of rows must equal 3 and the number of columns must equal the number of transmitter or receivers in `asset`. The rows correspond to the latitude, longitude, and altitude, in that order, and each column represents the pointing coordinates of the corresponding transmitter or receiver in the vector `asset`. The latitudes and longitudes are specified in degrees and the altitudes are specified in meters, which represent the height above the surface of the Earth.

`target` — Target
scalar | vector

Target at which input `asset` is pointed, specified as a scalar or a vector. The input `target` can be another satellite or a ground station.

- When `target` is a scalar, all assets point to the specified `target`.
- When `target` is a vector, the length of `target` must equal the number of transmitter or receivers in `asset`. Each element in `target` represents the pointing target of an asset.

`weights` — Weights for phased array antenna elements
scalar | vector

Weights for phased array antenna elements, specified as a scalar or a vector of complex numbers whose length is equal to the number of phased array elements.

Version History

Introduced in R2023a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `access` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Objects

ccsdsTCConfig

CCSDS TC configuration parameters

Description

The `ccsdsTCConfig` object creates a configuration object for Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) using default and specified values. `ccsdsTCConfig` object is configurable by using applicable “Properties” on page 3-2.

Creation

Syntax

```
cfg = ccsdsTCConfig
cfg = ccsdsTCConfig(Name,Value)
```

Description

`cfg = ccsdsTCConfig` creates a CCSDS TC configuration object using default properties.

`cfg = ccsdsTCConfig(Name,Value)` sets “Properties” on page 3-2 using one or more name-value pairs. Enclose each property name in quotes. For example, `ccsdsTCConfig('DataFormat','CLTU','Modulation','BPSK')` configures the CSSDS TC configuration object with a communications link transmission unit data format and binary phase shift keying (BPSK) modulation scheme.

Properties

`DataFormat` — Data formats used by PLOPs
`"CLTU"` (default) | `"acquisition sequence"` | `"idle sequence"`

Data formats used by physical layer operation procedures (PLOPs), specified as one of these options.

- `"CLTU"` — Communications link transmission unit (CLTU)
- `"acquisition sequence"`
- `"idle sequence"`

Data Types: `char` | `string`

`ChannelCoding` — Forward error correction coding
`"BCH"` (default) | `"LDPC"`

Forward error correction coding, specified as one of these options.

- `"BCH"` — Bose Chaudhuri Hocquenghem (BCH)
- `"LDPC"` — Low-density parity–check (LDPC)

Dependencies

To enable this property, set the `DataFormat` property to `"CLTU"`.

Data Types: `char` | `string`

`LDPCCodewordLength` — LDPC codeword length

`128` (default) | `512`

LDPC codeword length, specified as `128` or `512`.

Dependencies

To enable this property, set the `ChannelCoding` property to `"LDPC"`.

Data Types: `double`

`HasRandomizer` — Flag to indicate randomization

`1` or `true` (default) | `0` or `false`

Flag to indicate randomization on the bits in CLTU and on the fill data added prior to randomization, specified as a logical value of `1` (`true`) or `0` (`false`). To indicate the presence of a randomizer in the waveform, set this value to `1` (`true`).

Dependencies

To enable this property, set the `ChannelCoding` property to `"BCH"`.

Data Types: `logical`

`HasTailSequence` — Flag to indicate tail sequence in CLTU

`1` or `true` (default) | `0` or `false`

Flag to indicate the tail sequence in CLTU, specified as a logical value of `1` (`true`) or `0` (`false`). To indicate the presence of the tail sequence to delimit the end of a CLTU, set this value to `1` (`true`).

Dependencies

To enable this property, set the `ChannelCoding` property to `"LDPC"` and the `LDPCCodewordLength` property to `128`.

Data Types: `logical`

`Modulation` — Modulation scheme

`"PCM/PSK/PM"` (default) | `"PCM/PM/biphase-L"` | `"BPSK"`

Modulation scheme used to generate the CCSDS TC waveform, in the form of baseband in-phase quadrature (IQ) samples, specified as one of these options.

- `"PCM/PSK/PM"` — The line coded signal as per the pulse code modulation (PCM) format is phase shift keying (PSK) modulated on a sine wave subcarrier and then phase modulated (PM) on a residual carrier.
- `"PCM/PM/biphase-L"` — The biphase-L (Manchester) encoded data is phase modulated on a residual carrier.
- `"BPSK"` — Suppressed carrier modulation by using non-return-to-zero (NRZ) data on the carrier.

For more details on these modulation schemes, see [3].

Data Types: `char` | `string`

`PCMFormat` — PCM format
`"NRZ-L"` (default) | `"NRZ-M"`

Pulse code modulation (PCM) format, specified as one of these options. This property specifies the PCM coding in the CCSDS TC waveform.

- `"NRZ-L"` — NRZ-level
- `"NRZ-M"` — NRZ-mark

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `char` | `string`

`ModulationIndex` — Modulation index in residual carrier phase modulation
`0.4` (default) | scalar in the range [0.2, 2]

Modulation index in the residual carrier phase modulation, specified as a scalar in the range [0.2, 2]. Units are in radians.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"` or `"PCM/PM/biphase-L"`.

Data Types: `double`

`SubcarrierFrequency` — Sine wave subcarrier frequency
`16000` (default) | `8000`

Sine wave subcarrier frequency in Hertz, specified as `16000` or `8000`. The subcarrier waveform is used to PSK-modulate the NRZ data on the residual RF carrier.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `double`

`SymbolRate` — Symbol rate
`4000` (default) | `2000` | `1000` | `500` | `250` | `125` | `62.5` | `31.25` | `15.625` | `7.8125`

Symbol rate in coded symbols per second, specified as one of these options.

- `4000`
- `2000`
- `1000`
- `500`
- `250`
- `125`
- `62.5`
- `31.25`

- `15.625`
- `7.8125`

Note If you set `SymbolRate` to `4000` coded symbols per second, you must set the `SubcarrierFrequency` property to `16000`.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `double`

`SamplesPerSymbol` — Number of samples per symbol

`10` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"` or `"PCM/PM/biphase-L"`.

Data Types: `double`

`SubcarrierWaveform` — Waveform used to PSK-modulate NRZ data

`"sine"`

This property is read-only.

Waveform used to PSK-modulate the NRZ data, returned as `"sine"`. CCSDS TC supports only sine-wave subcarriers.

Dependencies

To enable this property, set the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `char` | `string`

Object Functions

Specific to This Object

ccsdsTCWaveform	Generate CCSDS TC waveform

Examples

Create CCSDS TC Object

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) configuration object. Specify the properties of the object.

```
cfg = ccsdsTCConfig;
cfg.ChannelCoding = "LDPC";
cfg.HasTailSequence = false;
cfg.PCMFormat = "NRZ-M";
```

Display the properties of the CCSDS TC object.

```
disp(cfg)
```

```
  ccsdsTCConfig with properties:

             DataFormat: "CLTU"
          ChannelCoding: "LDPC"
     LDPCCodewordLength: 128
        HasTailSequence: 0
             Modulation: "PCM/PSK/PM"
              PCMFormat: "NRZ-M"
        ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
             SymbolRate: 4000
       SamplesPerSymbol: 10

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Create CCSDS TC Waveform for Multiple CLTUs

Create a Consultative Committee for Space Data Systems (CCSDS) Telecommand (TC) time-domain waveform for multiple communications link transmission units (CLTUs).

Create a default CCSDS TC configuration object.

```
cfg = ccsdsTCConfig;
disp(cfg)
```

```
  ccsdsTCConfig with properties:

             DataFormat: "CLTU"
          ChannelCoding: "BCH"
          HasRandomizer: 1
             Modulation: "PCM/PSK/PM"
              PCMFormat: "NRZ-L"
        ModulationIndex: 0.4000
    SubcarrierFrequency: 16000
             SymbolRate: 4000
       SamplesPerSymbol: 10

   Read-only properties:
     SubcarrierWaveform: "sine"
```

Specify the number of CLTUs and the transfer frame length.

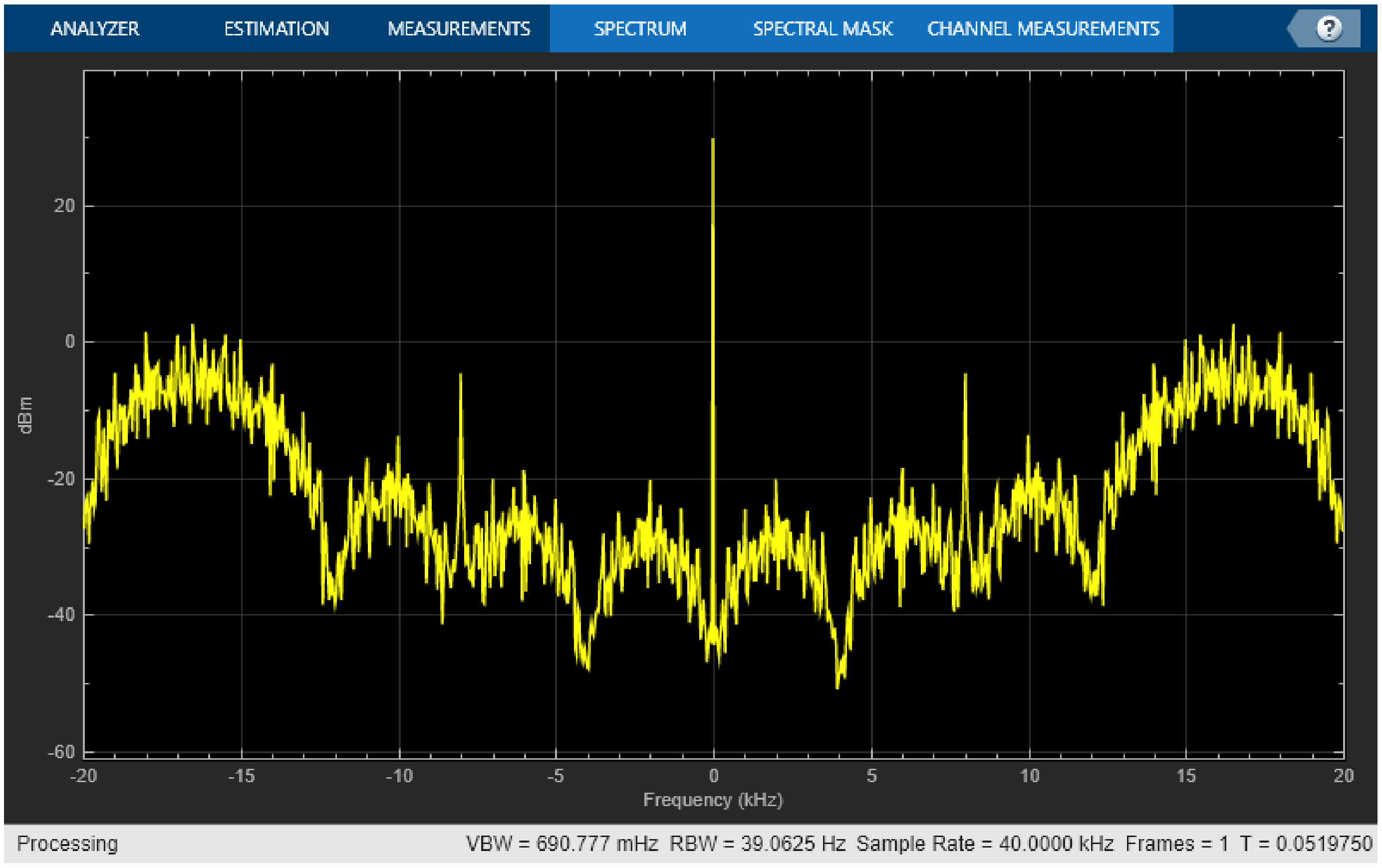
```
numCLTUs = 10;
transferFramesLength = 8; % Number of octets in each transfer frame
```

Generate the CCSDS TC time-domain waveform for the transfer frames.

```
c = cell(1,numCLTUs); % Cell array to store the generated waveform for all CLTUs
for k=1:numCLTUs
    bits = randi([0 1],8*transferFramesLength,1); % Bits in the TC transfer frame
    waveform = ccsdsTCWaveform(bits,cfg);
    c{1,k} = waveform; % Waveform for each CLTU
end
```

Create a spectrum analyzer System object to display the frequency spectrum of the generated CCSDS TC time-domain waveform from the last CLTU.

```
scope = spectrumAnalyzer;
scope.SampleRate = cfg.SamplesPerSymbol*cfg.SymbolRate;
scope(waveform)  % Last CLTU spectrum display
```



VBW = 690.777 mHz RBW = 39.0625 Hz Sample Rate = 40.0000 kHz Frames = 1 T = 0.0519750

Version History

Introduced in R2021a

References

[1] CCSDS 231.0-B-3. Blue Book. Issue 3. "TC Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-29. Blue Book. Issue 29. "Radio Frequency and Modulation Systems - Part 1". *Earth Stations and Spacecraft*. Washington, D.C.: CCSDS, September 2019.

[3] Nguyen, T.M., W.L. Martin, and Hen-Geul Yeh. "Required Bandwidth, Unwanted Emission, and Data Power Efficiency for Residual and Suppressed Carrier Systems - a Comparative Study."

IEEE transactions on electromagnetic compatibility 37, no. 1 (February 1995): 34-50. https://doi.org/10.1109/15.350238.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Properties `LDPCCodewordLength` and `ChannelCoding` must be provided as compile-time constant inputs in code generation. Use `coder.Constant` (MATLAB Coder) object to convert the input variable to a constant during code generation.

See Also

Functions
`ccsdsTCWaveform` | `ccsdsTCIdealReceiver`

Objects
`ccsdsTMWaveformGenerator`

dvbrcs2RecoveryConfig

Receiver configuration parameters for DVB-RCS2

Description

The `dvbrcs2RecoveryConfig` object creates a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) recovery configuration object. Recover the frame protocol data unit (PDU) from the received DVB-RCS2 waveform by using object properties.

Creation

Syntax

```
cfgrcs2 = dvbrcs2RecoveryConfig
cfgrcs2 = dvbrcs2RecoveryConfig(Name,Value)
```

Description

`cfgrcs2 = dvbrcs2RecoveryConfig` creates a default DVB-RCS2 recovery configuration object.

`cfgrcs2 = dvbrcs2RecoveryConfig(Name,Value)` sets “Properties” on page 3-9 using one or more name-value pairs. Enclose each property name in quotes. For example, `dvbrcs2RecoveryConfig('IsCustomWaveform',true)` recovers a custom DVB-RCS2 waveform with the specified property values.

Properties

`TransmissionFormat` — Transmission format
`"TC-LM"` (default) | `"SS-TC-LM"`

Transmission format, specified as one of these values.

- `"TC-LM"` — Turbo codes with linear modulation (TC-LM)
- `"SS-TC-LM"` — Spread spectrum turbo codes with linear modulation (SS-TC-LM)

Data Types: `char` | `string`

`ContentType` — Frame PDU burst content type
`"traffic"` (default) | `"logon"` | `"control"`

Frame protocol data unit (PDU) burst content type, specified as `"traffic"`, `"logon"`, or `"control"`.

Data Types: `char` | `string`

`IsCustomWaveform` — Custom waveform indicator
`0` or `false` (default) | `1` or `true`

Custom waveform indicator, specified as one of these values.

- `0` (`false`) — Use this option to demodulate the complex in-phase quadrature (IQ) samples from a standard-defined reference waveform.
- `1` (`true`) — Use this option to demodulate the complex IQ samples from a custom waveform.

Data Types: `logical`

`WaveformID` — Reference waveform ID
`1` (default) | positive integer

Reference waveform ID, specified as one of these options.

- Integer in the range [1, 22] or [32, 49] — Use this option when you set the `TransmissionFormat` property to `"TC-LM"`.
- Integer in the range [1, 19] — Use this option when you set the `TransmissionFormat` property to `"SS-TC-LM"`.

Based on the values set for `TransmissionFormat` and `WaveformID` properties, this object considers the receiver parameters according to ETSI EN 301 545-2 Annex A Table A-1 and A-2 [1].

Dependencies

To enable this property, set the `IsCustomWaveform` property to `false`.

Data Types: `double` | `unit8`

`SpreadingFactor` — Spreading factor
`2` (default) | integer in the range [2, 16]

Spreading factor, specified as an integer in the range [2, 16].

Dependencies

To enable this property, set the `TransmissionFormat` property to `"SS-TC-LM"` and the `IsCustomWaveform` property to `true`.

Data Types: `double`

`BurstLength` — Burst length
`256` (default) | integer in the range [7, 25,233,405]

Burst length, specified as an integer in the range [7, 25,233,405]. This length includes the preamble, postamble, and pilot sum, in addition to the payload symbols.

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of burst length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of burst length is chips.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`MappingScheme` — Mapping scheme
`"pi/2-BPSK"` (default) | `"QPSK"` | `"8PSK"` | `"16QAM"`

Mapping scheme, specified as one of these values.

- `"pi/2-BPSK"`
- `"QPSK"`
- `"8PSK"`
- `"16QAM"`

Dependencies

To enable this property, set the `TransmissionFormat` property to `"TC-LM"` and the `IsCustomWaveform` property to `true`.

Note When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the only valid value of `MappingScheme` is `"pi/2-BPSK"`.

Data Types: `char` | `string`

`CodeRate` — Code rate

`"1/3"` (default) | `"1/2"` | `"2/3"` | `"3/4"` | `"4/5"` | `"5/6"` | `"6/7"` | `"7/8"`

Code rate, specified as one of these values.

- `"2/3"`, `"3/4"`, `"4/5"`, `"5/6"`, `"6/7"`, or `"7/8"` — Use one of these values when you set the `MappingScheme` property to `"8PSK"`.
- `"3/4"`, `"4/5"`, `"5/6"`, `"6/7"`, or `"7/8"` — Use one of these values when you set the `MappingScheme` property to `"16QAM"`.

All code rates are applicable if `MappingScheme` property is set to `"pi/2-BPSK"` or `"QPSK"`.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `char` | `string`

`PermutationParameters` — Permutation control parameters

`[9 0 0 0 0]` (default) | vector

Permutation control parameters that the dvbrcs2RecoveryConfig uses to generate turbo encoder interleaver indices, specified as a five-element vector in order: P, Q_0, Q_1, Q_2, and Q_3. P must be in the range [9, 255], and Q_0, Q_1, Q_2, and Q_3 must be in the range [0, 15].

To generate unique interleaver indices, the value of P must be co-prime to `PayloadLengthInBytes`*4.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PreambleLength` — Preamble length

8 (default) | integer in the range [0, 255]

Preamble length, specified as an integer in the range [0, 255].

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of preamble length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of preamble length is chips.

A preamble of this specified length is prefixed to the payload symbols.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PostambleLength` — Postamble length
`8` (default) | integer in the range [0, 255]

Postamble length, specified as an integer in the range [0, 255].

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of postamble length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of postamble length is chips.

A postamble of this specified length is suffixed to the payload symbols in the burst sequence.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PilotPeriod` — Pilot period
`0` (default) | integer in the range [0, 4095]

Pilot period, specified as an integer in the range [0, 4095]. A value of `0` indicates no pilots are inserted.

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of pilot period is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of pilot period is chips.

The pilot period represents the length of the sequence from first symbol of a pilot block to the first symbol of the next pilot block in symbols or chips.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PilotBlockLength` — Pilot block length
`1` (default) | integer in the range [1, 255]

Pilot block length, specified as an integer in the range [1, 255].

After every `PilotPeriod` symbols or chips, a pilot block of this specified length is detected, which must be removed to recover the payload symbols.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true` and `PilotPeriod` property to a positive integer.

Data Types: `double`

`PilotSum` — Total pilot symbols or chips in received waveform
`0` (default) | nonnegative integer

Total pilot symbols or chips in the received waveform, specified as one of these options.

- Integer in the range [0, 255] — Use this option when you set the `TransmissionFormat` property to `"TC-LM"`.
- Integer in the range [0, 65,535] — Use this option when you set the `TransmissionFormat` property to `"SS-TC-LM"`.

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of pilot sum is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of pilot sum is chips.

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true` and `PilotPeriod` property to a positive integer.

Data Types: `double`

`ScramblingPolynomial` — Scrambling polynomial
16-bit zero vector (default) | 16-bit vector of binary values | numeric vector

Scrambling polynomial, specified as one of these options.

- 16-bit vector of binary values from the most significant bit (MSB), z^{16}, to least significant bit (LSB), z^{1}. Each element of this vector corresponds to the coefficient of z and its exponent, specified from MSB to LSB. For details on the binary representation, see ETSI EN 301 545-2 Section 7.3.7.1.5.
- Numeric vector containing the exponents of z for nonzero terms of the polynomial in descending order.

The scrambling polynomial determines the shift register feedback connection to generate the spreading sequence.

The coefficient of z^{0} is always 1.

The default value of this scrambling polynomial indicates the default scrambling sequence provided in the standard. When you set the `TransmissionFormat` property to `"SS-TC-LM"` and the `IsCustomWaveform` property to `false`, the default scrambling sequence is used to descramble the received reference waveform.

Dependencies

To enable this property, set the `TransmissionFormat` property to `"SS-TC-LM"` and the `IsCustomWaveform` property to `true`.

Data Types: `double` | `logical`

`ScramblingInitialConditions` — Scrambling initial conditions
`[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]` (default) | `1` | 16-bit vector of binary values

Scrambling initial conditions of the shift register, specified as one of these options.

- `1` — Use this option to set the initial condition of each cell of the shift register to this value.
- 16-bit vector of binary values from the MSB (z^{16}) to LSB (z^1) — Use this option to set the initial condition of each cell of the shift register to the corresponding element in this vector.

Dependencies

To enable this property, set the `TransmissionFormat` property to `"SS-TC-LM"` and the `ScramblingPolynomial` property to a value other than the default value.

Data Types: `double` | `logical`

`NumDecodingIterations` — Number of decoding iterations
8 (default) | positive integer

Number of decoding iterations of the DVB-RCS2 turbo decoder, specified as a positive integer.

Data Types: `double`

`PayloadLengthInBytes` — Payload length in bytes
`10` (default) | positive integer

This property is read-only.

Payload length in bytes, retuned as a positive integer. This length represents the DVB-RCS2 turbo decoder output length.

Use this property output to choose a valid value for the first element of `PermutationParameters` property (that is, *P*).

`PayloadLengthInBytes`*4 and *P* must be co-primes.

Data Types: `double`

Object Functions

Specific to This Object

dvbrcs2BitRecover	Recover bits for DVB-RCS2 waveform

Examples

Create DVB-RCS2 Receiver Object

Create a DVB-RCS2 recovery configuration object.

Create and then set the properties of the object.

```
cfgrcs2 = dvbrcs2RecoveryConfig;
cfgrcs2.TransmissionFormat = "SS-TC-LM";
cfgrcs2.ContentType = "control";
cfgrcs2.WaveformID = 20;
cfgrcs2.NumDecodingIterations = 6;
```

Display the properties of the DVB-RCS2 object.

```
disp(cfgrcs2)

  dvbrcs2RecoveryConfig with properties:

       TransmissionFormat: "SS-TC-LM"
              ContentType: "control"
         IsCustomWaveform: 0
               WaveformID: 20

   Coding and Modulation:
    NumDecodingIterations: 6
```

Recover PDU from DVB-RCS2 Reference Waveform

Recover the frame PDU for a DVB-RCS2 reference waveform.

Set the properties of a DVB-RCS2 waveform generator System object™.

```
wg = dvbrcs2WaveformGenerator;
wg.TransmissionFormat = "SS-TC-LM";
wg.WaveformID = 7;
wg.SamplesPerSymbol = 2;
```

Generate a frame PDU.

```
framePDU = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2-based burst symbols.

```
txWaveform = wg(framePDU);
```

Add additive white Gaussian noise (AWGN) to the generated waveform.

```
sps = wg.SamplesPerSymbol;
EsNodB = 1;
snrdB = EsNodB - 10*log10(sps);
rxIn = awgn(txWaveform,snrdB,"measured");
```

Create and then configure the DVB-RCS2 recovery configuration object.

```
cfg = dvbrcs2RecoveryConfig;
cfg.TransmissionFormat = wg.TransmissionFormat;
cfg.WaveformID = wg.WaveformID;
```

Create a raised cosine receiver filter.

```
rxFilter = comm.RaisedCosineReceiveFilter( ...
                 'RolloffFactor',0.2, ...
                 'InputSamplesPerSymbol',sps, ...
                 'DecimationFactor',sps);
span = rxFilter.FilterSpanInSymbols;
```

Apply matched filtering and remove the filter delay.

```
filtOut = rxFilter([rxIn; ...
               complex(zeros(span/2*sps,1))]);
rxSymb = filtOut(span+1:end);
```

Recover user packets. Display the frame PDU cyclic redundancy check (CRC) status and the numbers of bit errors.

```
[rxOut,pduErr] = dvbrcs2BitRecover(rxSymb,cfg,10^(-EsNodB/10));
fprintf("Erroneous frame PDU = %d\n", pduErr)
```

```
Erroneous frame PDU = 0
```

```
fprintf("Number of bit errors = %d\n", sum(framePDU~=rxOut))
```

```
Number of bit errors = 0
```

Version History

Introduced in R2021b

References

[1] ETSI Standard EN 301 545-2 V1.2.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Interactive Satellite Systems (DVB-RCS2); Part 2: Lower Layers for Satellite Standard.*

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dvbrcs2BitRecover` | `dvbrcs2TurboDecode`

Objects
`dvbrcs2WaveformGenerator`

Pattern

Radiation pattern visualization

Description

The `Pattern` object defines a radiation pattern visualization for a transmitter or receiver.

Creation

You can create `Pattern` objects by using the `pattern` object function of the `Transmitter` or `Receiver` object.

Properties

`Size` — Size of radiation pattern plot
`1000000` (default) | numeric scalar

Size of the radiation pattern plot, specified as a numeric scalar in meters. This value represents the distance between the antenna position and the point on the plot with the highest gain.

Data Types: `double`

`Colormap` — Colormap for coloring pattern plot
`'jet'` (default) | predefined colormap name | *M*-by-3 matrix

Colormap for coloring the pattern plot, specified as a predefined colormap name or an *M*-by-3 matrix of red, green, blue (RGB) triplets that define *M* individual colors. For more information on the colormap names, see “map”.

Data Types: `double` | `string` | `char`

`Transparency` — Transparency of pattern plot
`0.4` (default) | scalar in the range [0, 1]

Transparency of the pattern plot, specified as a scalar in the range [0, 1]. A value of `0` means the plot is completely transparent, and a value of `1` means the plot is opaque.

Data Types: `double`

`VisibilityMode` — Visibility of graphic relative to its parent
`'inherit'` (default) | `'manual'`

Visibility of the graphic relative to its parent, specified as `'inherit'` or `'manual'`. This visibility mode determines the visibility of this graphic in the `satelliteScenarioViewer` object relative to its parent graphic. The parent graphic of the `Pattern` object is its corresponding satellite.

- `'inherit'`— Inherit visibility from the parent graphic. The visibility of the graphic matches the parent visibility.
- `'manual'`— Do not inherit visibility from the parent. The visibility of the graphic is independent of the parent visibility.

Data Types: char | string

Object Functions

show	Show object in satellite scenario viewer
hide	Hide satellite scenario entity from viewer

Examples

Visualize Radiation Pattern of Transmitter Antenna on Satellite

Set up the satellite scenario.

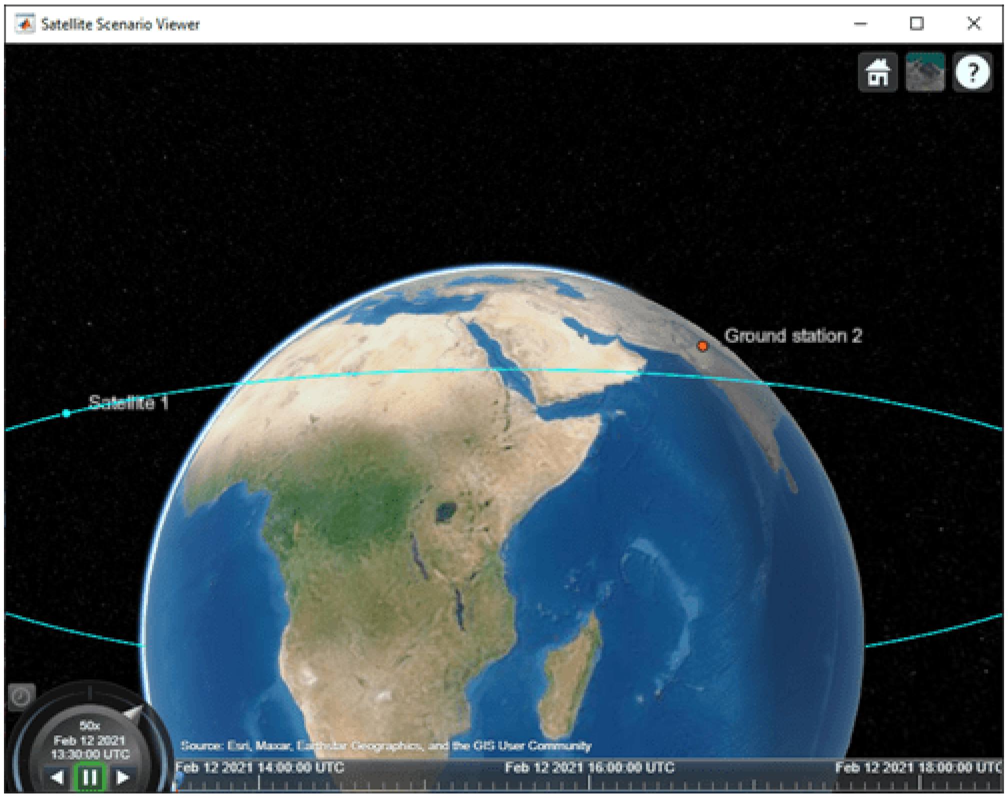
```
startTime = datetime(2021,2,12,13,30,0);
stopTime = startTime + hours(5);
sampleTime = 60;                                      %seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Create a satellite, ground station, transmitter, and receiver.

```
sat = satellite(sc,1e7,0,0,0,0,0);
gs = groundStation(sc,"Latitude",30,"Longitude",74);
tx = transmitter(sat,"Frequency",3e9);
rx = receiver(gs);
```

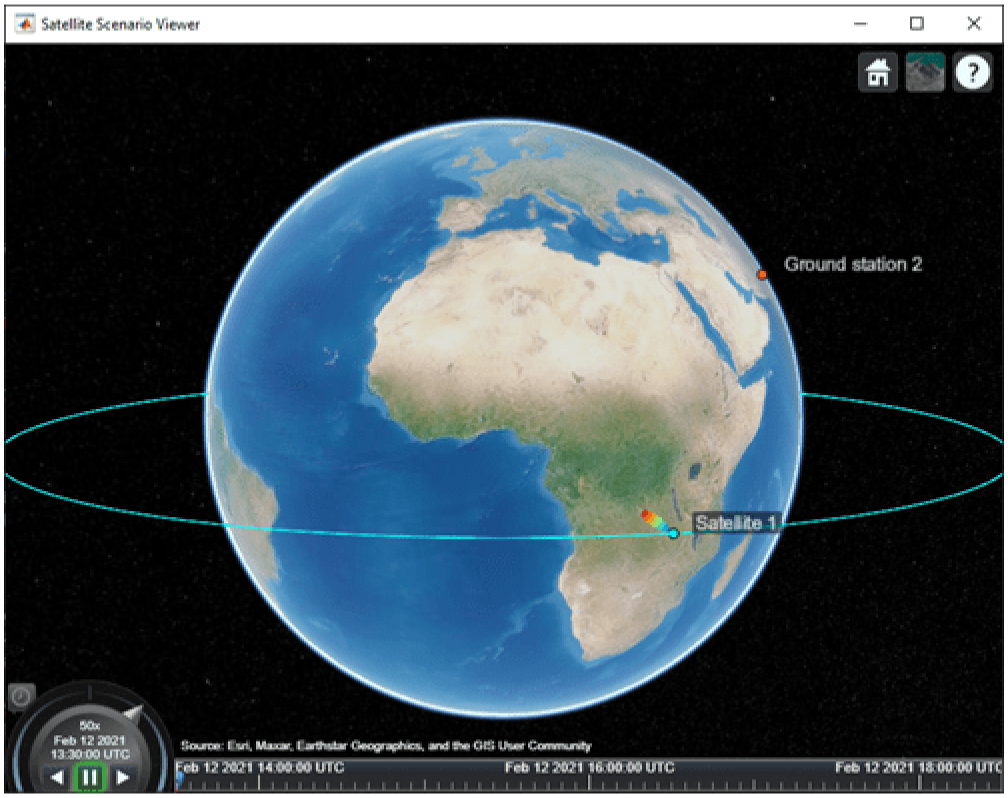
Visualize the scenario in the satellite scenario viewer.

```
viewer = satelliteScenarioViewer(sc);
```



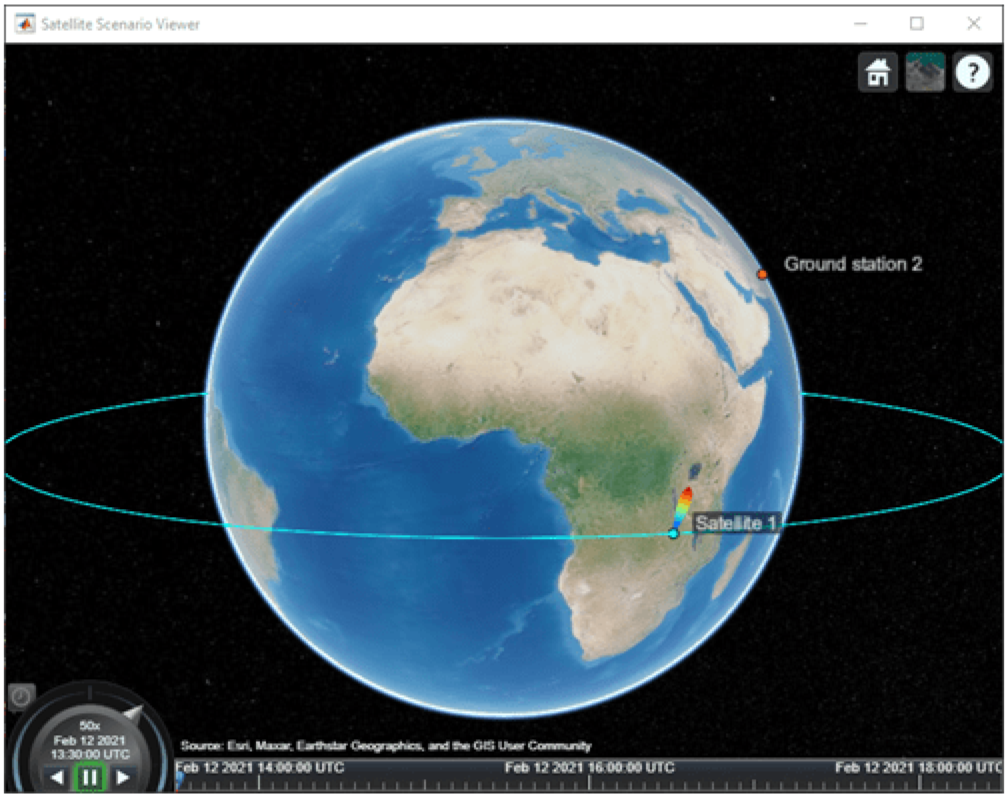
Plot the radiation pattern of the transmitter antenna.

```
pat = pattern(tx);
```



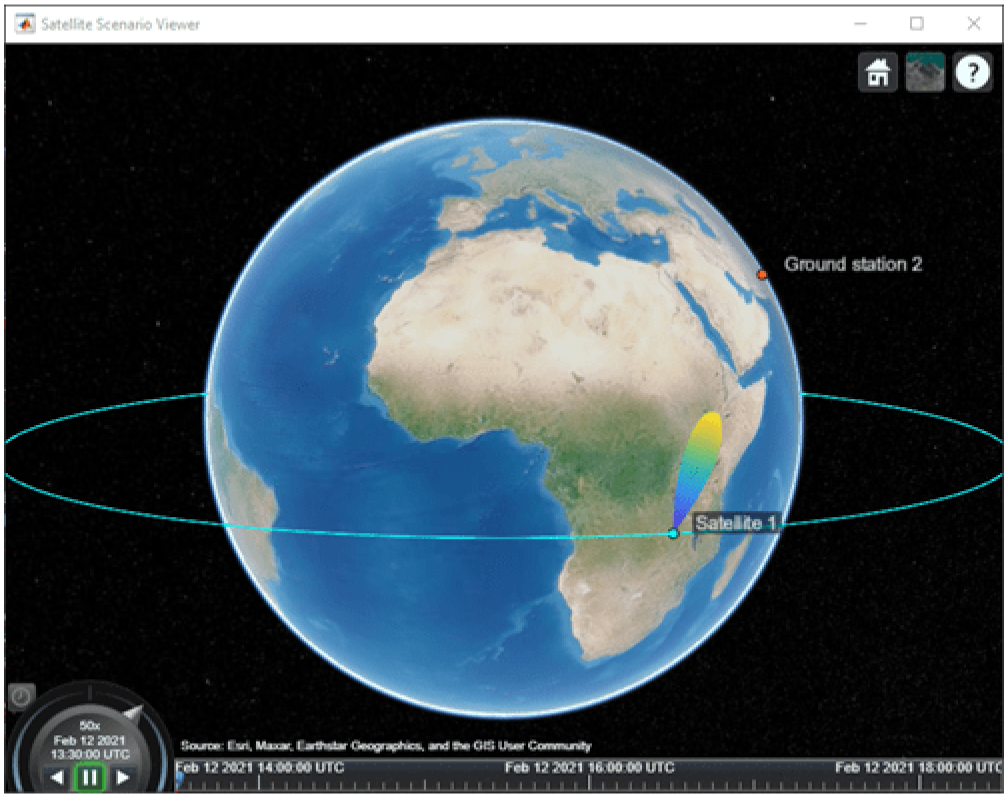
Point the satellite at the ground station. The pattern rotates to reflect the new orientation of the antenna.

```
pointAt(sat,gs);
```



Increase the visual size of the radiation pattern.

```
pat.Size = 3000000;
pat.Colormap = "parula";
```



Version History

Introduced in R2021b

See Also

Objects
`Receiver` | `Transmitter` | `satelliteScenarioViewer` | `satelliteScenario`

Functions
`show` | `hide` | `receiver` | `transmitter`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

p618Config

Create P.618 configuration object

Description

The `p618Config` object sets the P.618 configuration parameters required for the calculation of the Earth-space propagation losses, cross-polarization discrimination, and sky noise temperature, as defined in the ITU-R P.618 recommendation [1].

Creation

Syntax

```
cfgP618 = p618Config
cfgP618 = p618Config(Name,Value)
```

Description

`cfgP618 = p618Config` creates a P.618 configuration object with default property values.

`cfgP618 = p618Config(Name,Value)` specifies “Properties” on page 3-22 using one or more name-value pair arguments. Enclose each property name in quotes. For example, p618Config(`'GasAnnualExceedance',10,'AntennaEfficiency',0.65`) configures a P.618 configuration object with 10% average annual time percentage of excess for gaseous attenuation and 0.65 antenna efficiency.

Properties

`Frequency` — Signal frequency
`14.25e9` (default) | scalar in the range [1e9, 55e9]

Signal frequency in Hz, specified as a scalar in the range [1e9, 55e9].

Data Types: `double` | `single`

`ElevationAngle` — Elevation angle
`31.0769` (default) | scalar in the range [5, 90]

Elevation angle in degrees, specified as a scalar in the range [5, 90].

Data Types: `double` | `single`

`Latitude` — Earth station latitude
`51.5000` (default) | scalar in the range [-90, 90]

Earth station latitude in degrees, specified as a scalar in the range [-90, 90]. A positive value corresponds to a North latitude, and a negative value corresponds to a South latitude.

Data Types: `double` | `single`

`Longitude` — Earth station longitude
`-0.1400` (default) | scalar in the range [-180, 180]

Earth station longitude in degrees, specified as a scalar in the range [-180, 180]. A positive value corresponds to East longitude, and a negative value corresponds to West longitude.

Data Types: `double` | `single`

`GasAnnualExceedance` — Average annual time percentage of excess for gaseous attenuation
`1` (default) | scalar in the range [0.1, 99]

Average annual time percentage of excess for the gaseous attenuation, specified as a scalar in the range [0.1, 99]. This property calculates the gaseous attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Note The fraction of time during which a preselected threshold is exceeded in an average year is referred to as the annual time percentage of excess.

Data Types: `double` | `single`

`CloudAnnualExceedance` — Average annual time percentage of excess for cloud attenuation
`1` (default) | scalar in the range [0.1, 99]

Average annual time percentage of excess for the cloud attenuation, specified as a scalar in the range [0.1, 99]. This property calculates the cloud attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`RainAnnualExceedance` — Average annual time percentage of excess for rain attenuation
`1` (default) | scalar in the range [0.001, 5]

Average annual time percentage of excess for the rain attenuation, specified as a scalar in the range [0.001, 5]. This property calculates the rain attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`ScintillationAnnualExceedance` — Average annual time percentage of excess for tropospheric scintillation
`1` (default) | scalar in the range [0.01, 50]

Average annual time percentage of excess for the tropospheric scintillation, specified as a scalar in the range [0.01, 50]. This property calculates the tropospheric scintillation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`TotalAnnualExceedance` — Average annual time percentage of excess for total attenuation
`1` (default) | scalar in the range [0.001, 50]

Average annual time percentage of excess for the total attenuation, specified as a scalar in the range [0.001, 50]. This property calculates the total attenuation, which satisfies the exceedance condition, in terms of the percentage of an average year.

Data Types: `double` | `single`

`PolarizationTiltAngle` — Polarization tilt angle
`0` (default) | scalar in the range [-90, 90]

Polarization tilt angle in degrees, specified as a scalar in the range [-90, 90].

Data Types: `double` | `single`

`AntennaDiameter` — Physical diameter of earth station antenna
`1` (default) | positive scalar

Physical diameter of the earth station antenna in meters, specified as a positive scalar.

Data Types: `double` | `single`

`AntennaEfficiency` — Antenna efficiency of earth station antenna
`0.5` (default) | positive scalar

Antenna efficiency of the earth station antenna, specified as a positive scalar.

Data Types: `double` | `single`

Object Functions

Specific to This Object

p618PropagationLosses	Calculate Earth-space propagation losses, cross-polarization discrimination, and sky noise temperature

Examples

Create P.618 Configuration Object

Create a default P.618 configuration object.

```
cfg = p618Config;
```

Specify the signal frequency as 25 GHz, elevation angle as 45 degrees, and antenna efficiency as 0.65. Set the time percentage of excess for the total attenuation per annum as 0.001.

```
cfg.Frequency = 25e9;
cfg.ElevationAngle = 45;
cfg.AntennaEfficiency = 0.65;
cfg.TotalAnnualExceedance = 0.001;
```

Set the earth station direction.

```
cfg.Latitude = 30;   % North direction
cfg.Longitude = 120; % East direction
```

Display the properties of the configuration object.

```
disp(cfg)

  p618Config with properties:

                      Frequency: 2.5000e+10
                 ElevationAngle: 45
                       Latitude: 30
                      Longitude: 120
            GasAnnualExceedance: 1
          CloudAnnualExceedance: 1
           RainAnnualExceedance: 1
  ScintillationAnnualExceedance: 1
          TotalAnnualExceedance: 1.0000e-03
          PolarizationTiltAngle: 0
                AntennaDiameter: 1
              AntennaEfficiency: 0.6500
```

Calculate Propagation Losses, Cross-Polarization Discrimination, and Sky Noise Temperature

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a default P.618 configuration object.

```
cfg = p618Config;
```

Specify the time percentage of excess for the rain attenuation per annum as 0.01 and the time percentage of excess for the total attenuation per annum as 0.001.

```
cfg.RainAnnualExceedance = 0.01;
cfg.TotalAnnualExceedance = 0.001;
```

Calculate the propagation losses, cross-polarization discrimination, and sky noise temperature.

```
[pl,xpd,tsky] = p618PropagationLosses(cfg)
```

```
pl = struct with fields:
    Ag: 0.2269
    Ac: 0.4552
    Ar: 6.7981
    As: 0.2633
    At: 15.6091

xpd = 32.8876

tsky = 267.4689
```

Calculate Propagation Losses in Light Rainfall

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a P.618 configuration object that occupies a signal frequency of 20 GHz.

```
cfg = p618Config('Frequency',20e9);
```

Calculate the propagation losses in a light rainfall of 1 mm/hr with an earth station height of 0.75 km.

```
pl =  p618PropagationLosses(cfg,'RainRate',1,'StationHeight',0.75)
```

```
pl = struct with fields:
    Ag: 0.7996
    Ac: 0.8793
    Ar: 0.0177
    As: 0.3187
    At: 1.7514
```

Version History

Introduced in R2021a

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`p618SiteDiversityConfig`

Functions
`p618PropagationLosses` | `p618SiteDiversityOutage`

p618SiteDiversityConfig

Create P.618 site diversity configuration object

Description

The `p618SiteDiversityConfig` object sets P.618 site diversity configuration parameters required for the calculation of outage probability due to rain attenuation, as defined in the ITU-R P.618 recommendation [1].

Creation

Syntax

```
cfgSD = p618SiteDiversityConfig
cfgSD = p618SiteDiversityConfig(Name,Value)
```

Description

`cfgSD = p618SiteDiversityConfig` creates a P.618 site diversity configuration object with default property values.

`cfgSD = p618SiteDiversityConfig(Name,Value)` specifies “Properties” on page 3-28 using one or more name-value pair arguments. Enclose each property name in quotes. For example, p618SiteDiversityConfig(`'Frequency',14.25e9,'ElevationAngle',[52.4099 52.4852]`) configures a P.618 site diversity configuration object with a 14.25 GHz signal frequency and an elevation angle for two sites as [52.4099 52.4852].

Properties

`Frequency` — Signal frequency
`14.25e9` (default) | scalar in the range [1e9, 55e9]

Signal frequency in Hz, specified as a scalar in the range [1e9, 55e9].

Data Types: `double` | `single`

`ElevationAngle` — Elevation angle of two sites
`[52.4099 52.4852]` (default) | two-element vector of values in the range [0, 90]

Elevation angle of the two sites in degrees, specified as a two-element vector of values in the range [0, 90].

Data Types: `double` | `single`

`Latitude` — Latitude of two sites
`[25.768 25.463]` (default) | two-element vector of values in the range [-90, 90]

Latitude of the two sites in degrees, specified as a two-element vector of values in the range [-90, 90]. A positive value corresponds to a North latitude, and a negative value corresponds to a South latitude.

Data Types: `double` | `single`

`Longitude` — Longitude of two sites
`[-80.205 -80.486]` (default) | two-element vector of values in the range [-180, 180]

Longitude of the two sites in degrees, specified as a two-element vector of values in the range [-180, 180]. A positive value corresponds to East longitude, and a negative value corresponds to West longitude.

Data Types: `double` | `single`

`PolarizationTiltAngle` — Polarization tilt angle for two sites
`[0 0]` (default) | two-element vector of values in the range [-90, 90]

Polarization tilt angle for the two sites in degrees, specified as a two-element vector of values in the range [-90, 90].

Data Types: `double` | `single`

`SiteDistance` — Separation between two sites
`44.0256` (default) | positive scalar

Separation between the two sites in km, specified as a positive scalar.

Data Types: `double` | `single`

`AttenuationThreshold` — Attenuation threshold on two links
`[9 3]` (default) | two-element vector

Attenuation threshold on the two links in dB, specified as a two-element vector. The attenuation threshold on an earth space link is the maximum allowed attenuation on the path. Any attenuation value above this property value is considered an outage in the link.

Data Types: `double` | `single`

Object Functions

Specific to This Object

p618SiteDiversityOutage	Calculate outage probability due to rain attenuation with site diversity

Examples

Create P.618 Site Diversity Configuration Object

Create a default P.618 site diversity configuration object.

```
cfg = p618SiteDiversityConfig;
```

Specify the polarization tilt angles for two sites as [-90 90] degrees, separation between the two sites as 50 km, and attenuation threshold on the two links as [9 9] dB.

```
cfg.PolarizationTiltAngle = [-90 90];
cfg.SiteDistance = 50;
cfg.AttenuationThreshold = [9 9];
```

Set the direction of each earth station.

```
cfg.Latitude = [30 60];    % North direction
cfg.Longitude = [120 150]; % East direction
```

Display the properties of the configuration object.

```
disp(cfg);
```

```
  p618SiteDiversityConfig with properties:

                Frequency: 1.4500e+10
           ElevationAngle: [52.4099 52.4852]
                 Latitude: [30 60]
                Longitude: [120 150]
    PolarizationTiltAngle: [-90 90]
             SiteDistance: 50
     AttenuationThreshold: [9 9]
```

Calculate Outage Probability due to Rain Attenuation with Site Diversity

This example requires MAT-files with digital maps from ITU documents. If they are not available on the path, execute the following commands to download and untar the MAT-files.

```
maps = exist('maps.mat','file');
p836 = exist('p836.mat','file');
p837 = exist('p837.mat','file');
p840 = exist('p840.mat','file');
matFiles = [maps p836 p837 p840];
if ~all(matFiles)
    if ~exist('ITURDigitalMaps.tar.gz','file')
        url = 'https://www.mathworks.com/supportfiles/spc/P618/ITURDigitalMaps.tar.gz';
        websave('ITURDigitalMaps.tar.gz',url);
        untar('ITURDigitalMaps.tar.gz');
    else
        untar('ITURDigitalMaps.tar.gz');
    end
    addpath(cd);
end
```

Create a P.618 site diversity configuration object with a signal frequency of 25 GHz.

```
cfgsd = p618SiteDiversityConfig;
cfgsd.Frequency = 25e9;
```

Specify the polarization tilt angles for two sites as [-90 90] degrees, separation between the two sites as 50 km, and attenuation threshold on the two links as [9 9] dB.

```
cfgsd.PolarizationTiltAngle = [-90 90];
cfgsd.SiteDistance = 50;
cfgsd.AttenuationThreshold = [9 9];
```

Calculate the outage probability due to rain attenuation with site diversity.

```
outage = p618SiteDiversityOutage(cfgsd)
```

```
outage = 0.0338
```

Version History

Introduced in R2021a

References

[1] International Telecommunication Union, ITU-R Recommendation P.618 (12/2017).

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Objects
`p618Config`

Functions
`p618PropagationLosses` | `p618SiteDiversityOutage`

satelliteCNRConfig

Carrier-to-noise ratio configuration parameters

Description

Use the `satelliteCNRConfig` object to create a carrier-to-noise ratio (CNR) configuration object with default or specified values.

Creation

Syntax

```
cfg = satelliteCNRConfig
cfg = satelliteCNRConfig(Name=Value)
```

Description

`cfg = satelliteCNRConfig` creates a default CNR configuration object.

This object contains the parameters required to calculate a satellite link budget CNR.

`cfg = satelliteCNRConfig(Name=Value)` sets properties on page 3-32 using one or more name-value arguments. For example, `TransmitterPower=20` sets the transmitter power to 20 dBW.

Properties

`TransmitterPower` — Transmitter power
`10` (default) | real scalar

Transmitter power in dBW, specified as a real scalar.

Data Types: `double`

`TransmitterSystemLoss` — Total system loss of the transmitter
`0` (default) | nonnegative real scalar

Total system loss of the transmitter in dB, specified as a nonnegative real scalar.

Data Types: `double`

`TransmitterAntennaGain` — Transmitter antenna gain
`10` (default) | nonnegative real scalar

Transmitter antenna gain in dBi, specified as a nonnegative real scalar.

Data Types: `double`

`Distance` — Distance between transmitter and receiver antenna
`3786` (default) | nonnegative real scalar

Distance between the transmitter and the receiver antenna in km, specified as a nonnegative real scalar.

Data Types: `double`

`Frequency` — Signal frequency
`14` (default) | positive real scalar

Signal frequency in GHz, specified as a positive real scalar.

Data Types: `double`

`MiscellaneousLoss` — Miscellaneous losses
`0` (default) | nonnegative real scalar

Miscellaneous losses in dB, specified as a nonnegative real scalar. Major miscellaneous losses include polarization loss, interference loss, and atmospheric attenuation.

Data Types: `double`

`GainToNoiseTemperatureRatio` — Receiver gain-to-noise temperature ratio
`3` (default) | real scalar

Receiver gain-to-noise temperature ratio in the direction of the transmitting antenna, specified as a real scalar. Value is in dB/K.

Data Types: `double`

`ReceiverSystemLoss` — Total receiver system loss
`0` (default) | nonnegative real scalar

Total receiver system loss in dB, specified as a nonnegative real scalar.

Data Types: `double`

`BitRate` — Bit rate of link
`10` (default) | positive real scalar

Bit rate of the link in Mbps, specified as a positive real scalar.

Data Types: `double`

`SymbolRate` — Symbol rate of link
`10` (default) | positive real scalar

Symbol rate of the link in Msym/s, specified as a positive real scalar.

Data Types: `double`

`Bandwidth` — Signal bandwidth
`6` (default) | positive real scalar

Signal bandwidth in MHz, specified as a positive real scalar.

Data Types: `double`

Object Functions

satelliteCNR	Carrier-to-noise ratio for configured satellite link budget parameters

Examples

Create Satellite CNR Object

Create a default satellite carrier-to-noise ratio (CNR) object.

Set the bandwidth to 15 MHz and gain-to-noise temperature to 5 dB/K.

```
cfg = satelliteCNRConfig;
cfg.Bandwidth = 15;
cfg.GainToNoiseTemperatureRatio = 5;
```

Display the CNR configuration object properties.

```
disp(cfg)
```

```
  satelliteCNRConfig with properties:

             TransmitterPower: 10
        TransmitterSystemLoss: 0
       TransmitterAntennaGain: 10
                     Distance: 3786
                    Frequency: 14
            MiscellaneousLoss: 0
  GainToNoiseTemperatureRatio: 5
           ReceiverSystemLoss: 0
                      BitRate: 10
                   SymbolRate: 10
                    Bandwidth: 15
```

Calculate CNR and Link Margin

Calculate the CNR and the received link margin for the specified link budget parameters.

Create a default CNR configuration object, and then set its properties.

```
cfg = satelliteCNRConfig;
cfg.TransmitterPower = 17;              % in dBW
cfg.TransmitterSystemLoss = 9;          % in dB
cfg.TransmitterAntennaGain = 38;        % in dBi
cfg.Distance = 40215;                   % in km
cfg.Frequency = 11;                     % in GHz

% Here, miscellaneous losses include polarization loss, interference
% loss, and antenna mispointing loss, respectively.
polLoss = 3.0103;
intLoss = 2;
antLoss = 1;
cfg.MiscellaneousLoss =  polLoss + intLoss + antLoss; % in dB
cfg.GainToNoiseTemperatureRatio = 25;                 % in dB/K
cfg.ReceiverSystemLoss = 2;                          % in dB
cfg.BitRate = 10;                                    % in Mbps
```

Display the CNR configuration object properties.

```
disp(cfg)

  satelliteCNRConfig with properties:

               TransmitterPower: 17
          TransmitterSystemLoss: 9
         TransmitterAntennaGain: 38
                       Distance: 40215
                      Frequency: 11
              MiscellaneousLoss: 6.0103
    GainToNoiseTemperatureRatio: 25
             ReceiverSystemLoss: 2
                        BitRate: 10
                     SymbolRate: 10
                      Bandwidth: 6
```

Calculate the CNR.

```
[cn,info] = satelliteCNR(cfg)

cn = 18.4440

info = struct with fields:
               TransmitterEIRP: 46
                          FSPL: 205.3634
        ReceivedIsotropicPower: -165.3737
    CarrierToNoiseDensityRatio: 86.2255
                  ReceivedEbNo: 16.2255
                  ReceivedEsNo: 16.2255
```

Compute the link margin. Assume a required energy per bit to noise power density ratio (Eb/No) of 10 dB and an implementation loss of 2 dB in the receiver.

```
reqEbNo = 10;
implLoss = 2;
margin = info.ReceivedEbNo - reqEbNo - implLoss

margin = 4.2255
```

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`satelliteCNR`

vita49Reader

VITA 49 file reader

Description

The `vita49Reader` object reads and decodes the VMEbus International Trade Association (VITA) 49 packets from the input VITA 49 file. The object reads both signal data packets and context packets.

Creation

Syntax

```
vita49ReaderObj = vita49Reader(filename)
vita49ReaderObj = vita49Reader(filename,OutputTimestampFormat="datetime")
```

Description

`vita49ReaderObj = vita49Reader(filename)` creates a VITA 49 file reader object to read the packets from the input VITA 49 file.

`vita49ReaderObj = vita49Reader(filename,OutputTimestampFormat="datetime")` sets the `OutputTimestampFormat` property to `"datetime"`.

Input Arguments

`filename` — Name of VITA 49 file
character vector | string scalar

Name of a VITA 49 file, including the extension, specified as a character vector or a string scalar.

Data Types: `char` | `string`

Properties

`OutputTimestampFormat` — Output format for packet timestamp
`"seconds"` (default) | `"datetime"`

Output format for the packet timestamp, specified as `"seconds"` or `"datetime"`. This value specifies the timestamp of the decoded VITA 49 packet.

Data Types: `char` | `string`

`PacketsRead` — Total number of packets read
integer scalar

This property is read-only.

Total number of packets read, stored as an integer scalar.

Data Types: `double`

Object Functions

read	Read next VITA 49 packet from file
reset	Reset VITA 49 file reader to first VITA 49 packet of file

Examples

Read Specified Number of Packets from File

Create a VITA 49 file reader object, specifying the name of a VITA 49 file and an output format for the packet timestamp.

```
vita49ReaderObj = vita49Reader("VITA49SampleData.bin");
vita49ReaderObj.OutputTimestampFormat = "seconds";
```

Specify the number of packets to be read from the file.

```
numpkt = 11;
```

Read the specified number of packets from the VITA 49 file to the MATLAB® workspace.

```
[signalDataPacket,contextPacket,contextPacketChangeIndex] = read(vita49ReaderObj,NumPackets=numpk
```

```
signalDataPacket = struct with fields:
                  PacketType: 1
                    StreamID: 0
                     ClassID: "7C386C0000"
                 PadBitCount: 0
        IntegerTimestampType: "GPS"
       IntegerTimestampValue: 1625215654
     FractionalTimestampType: "real time"
    FractionalTimestampValue: 900000344000
                    RawBytes: [1472x1 uint8]
                   IQSamples: [722x1 double]
                     Trailer: [1x1 struct]


contextPacket=1×10 struct array with fields:
    PacketType
    StreamID
    ClassID
    IntegerTimestampType
    IntegerTimestampValue
    FractionalTimestampType
    FractionalTimestampValue
    RawBytes
    ContextFieldChangeIndicator
    ReferencePointIdentifier
    Bandwidth
    IFReferenceFrequency
    RFFrequency
    RFFrequencyOffset
    IFBandOffset
    ReferenceLevel
    Gain
    OverRangeCount
    SampleRate
    TimestampAdjustment
    TimestampCalibrationTime
    StateAndEventIndicator
    SignalDataPayloadFormat
      ⋮

contextPacketChangeIndex = 1×10

     0     0     0     0     0     0     0     0     0     1
```

Version History

Introduced in R2022b

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

Code generation is available only when the `OutputTimestampFormat` property is set to the default value of `"seconds"`.

See Also

`read` | `reset`

Topics
“VITA 49 File Reader”

Access

Access analysis object belonging to scenario

Description

The Access object defines an access analysis object belonging to a `Satellite`, `GroundStation`, or `ConicalSensor` object. Use this object to determine if the line-of-sight (LOS) between two objects exists.

- If one of the objects belongs to `GroundStation`, the MinElevationAngle property of the `GroundStation` object limits the LOS visibility.
- If one of the objects belongs to `ConicalSensor`, the MaxViewAngle property of the `ConicalSensor` object limits the LOS visibility.

Creation

You can create an `Access` object using the `access` object function of `GroundStation` or `Satellite`.

Properties

Sequence — IDs of satellites, ground stations, or conical sensors
vector of positive numbers

IDs of the satellites, ground stations, and conical sensors defining access analysis, specified as a vector of positive numbers.

`LineWidth` — Visual width of access analysis object
`1` (default) | scalar

Visual width of access analysis object in pixels, specified as a scalar in the range (0, 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of analysis line
`[0.5 0 1]` (default) | RGB triplet | hexadecimal color code | color name | short name

Color of access analysis line, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

Object Functions

show	Show object in satellite scenario viewer
accessStatus	Status of access between first and last node defining access analysis
accessIntervals	Intervals during which access status is true
accessPercentage	Percentage of time when access exists between first and last node in access analysis
hide	Hide satellite scenario entity from viewer

Examples

Add Ground Stations to Scenario and Visualize Access Intervals

Create a satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = 10;
lon = -30;
gs = groundStation(sc,lat,lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

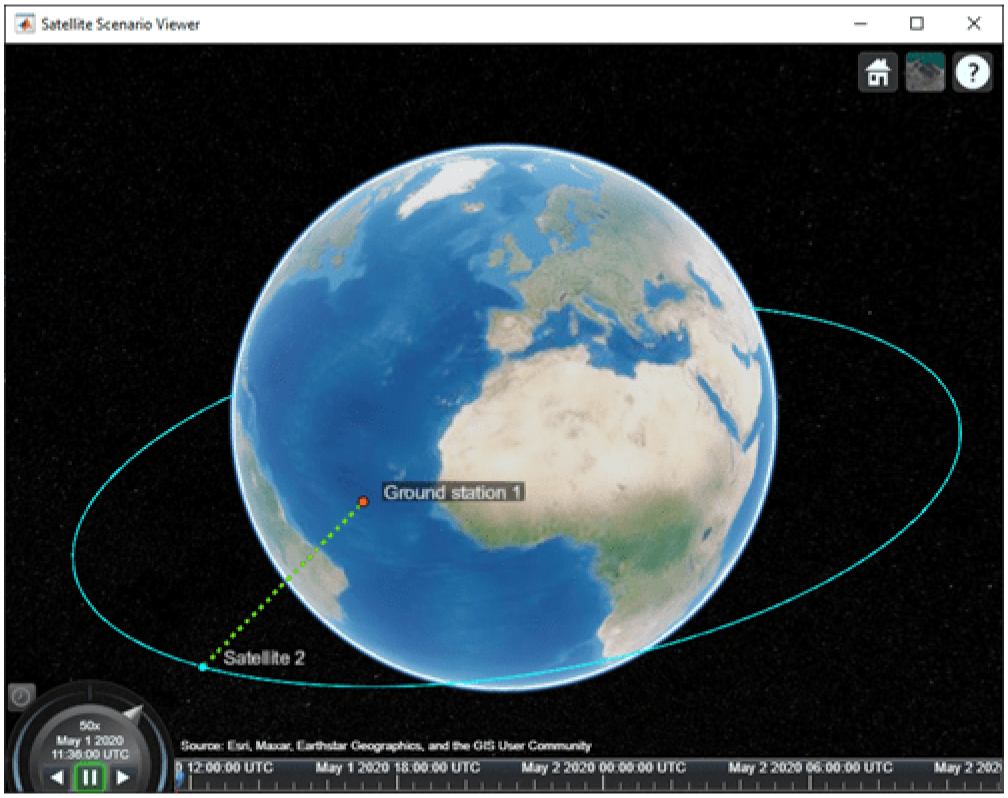
Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat,gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source             Target          IntervalNumber         StartTime                EndTi
    _____________    ___________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"           1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"           2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"           3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"           4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"           5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"           6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"           7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"           8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Version History

Introduced in R2021a

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`show` | `play` | `hide` | `groundStation` | `conicalSensor` | `transmitter` | `receiver` | `satellite`

Topics

"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

ConicalSensor

Conical sensor object belonging to satellite scenario

Description

`ConicalSensor` defines a conical sensor object belonging to a satellite scenario.

Creation

You can create the `ConicalSensor` object using the `conicalSensor` object function of the `Satellite`, `GroundStation`, or `Gimbal` objects.

Properties

`Name` — Conical sensor name
`"Conical sensor *idx*"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `conicalSensor` function. After you call the `conicalSensor` function, this property is read-only.

Conical sensor name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one conical sensor, specify `Name` as a string scalar or a character vector.
- If you are adding multiple conical sensors, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All conical sensors that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of conical sensors that you are adding. Each conical sensor is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`ID` — ConicalSensor ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

ConicalSensor ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one conical sensor, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of conical sensor.
- If you are adding multiple conical sensors, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified conical sensors. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the conical sensors. The columns correspond to the mounting location of each specified conical sensor and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one conical sensor, `MountingAngles` is a three-element vector.
- If you are adding multiple conical sensors, `MountingAngles` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified conical sensors. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the conical sensors. The columns correspond to the mounting angles of each specified conical sensor and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

`MaxViewAngle` — Field of view angle
`30` (default) | scalar in the range [0, 180] | vector

Field of view angle in degrees, specified as a scalar in the range [0, 180] or a vector.

- If you add one conical sensor, `MaxViewAngle` must be a scalar.
- If you add multiple conical sensors, `MaxViewAngle` can be a scalar or a vector. When `MaxViewAngle` is a scalar, the same field of view angle is assigned to all conical sensors that you are adding. When `MaxViewAngle` is a vector, the length of `MaxViewAngle` must equal the number of conical sensors in the parent. Each element of `MaxViewAngle` is assigned to the specified corresponding conical sensor.

When the AutoSimulate property of the satellite scenario is false, you can modify MaxViewAngle while the SimulationStatus is NotStarted or InProgress.

Data Types: double

Accesses — Access analysis objects
row vector of Access objects

You can set this property only when calling ConicalSensor. After you call ConicalSensor, this property is read-only.

Access analysis objects, specified as a row vector of Access objects.

CoordinateAxes — Coordinate axes triad graphic object
CoordinateAxes object (default)

You can set this property only when calling coordinateAxes. After you call coordinateAxes, this property is read-only.

Coordinate axes triad graphic object, specified as CoordinateAxes object.

FieldOfView — Field of view objects
row vector of FieldOfView objects

You can set this property only when calling ConicalSensor. After you call ConicalSensor, this property is read-only.

Field of view objects, specified as a scalar of FieldOfView objects.

Note The properties Name, MountingLocation , MountingAngles, and MaxViewAngle can be specified as name-value arguments in the conicalSensor object function. The size of specified name-value pairs determines the number of conical sensors that you can specify. Refer to these properties to understand how they must be defined when specifying multiple conical sensors.

Object Functions

aer	Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame
access	Add access analysis objects to satellite scenario
fieldOfView	Visualize field of view of conical sensor
coordinateAxes	Visualize coordinate axes triad of satellite scenario assets

Examples

Calculate Maximum Revisit Time of Satellite

Create a satellite scenario with a start time of 15-June-2021 8:55:00 AM UTC and a stop time of five days later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2021,6,21,8,55,0);
stopTime = startTime + days(5);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
sc = 
  satelliteScenario with properties:

         StartTime: 21-Jun-2021 08:55:00
          StopTime: 26-Jun-2021 08:55:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario using Keplerian orbital elements.

```
semiMajorAxis = 7878137;                                                    % me
eccentricity = 0;
inclination = 50;                                                           % de
rightAscensionOfAscendingNode = 0;                                          % de
argumentOfPeriapsis = 0;                                                    % de
trueAnomaly = 50;                                                           % de
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  Satellite with properties:

               Name:  Satellite 1
                 ID:  1
     ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
            Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
       Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
          Receivers:  [1x0 satcom.satellitescenario.Receiver]
           Accesses:  [1x0 matlabshared.satellitescenario.Access]
        GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
              Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator:  sgp4
        MarkerColor:  [0.059 1 1]
         MarkerSize:  6
          ShowLabel:  true
     LabelFontColor:  [1 1 1]
      LabelFontSize:  15
```

Add a ground station, which represents the location to be photographed, to the scenario.

```
gs = groundStation(sc,Name="Location to Photograph", ...
    Latitude=42.3001,Longitude=-71.3504)                 % degrees
```

```
gs = 
  GroundStation with properties:

                 Name:  Location to Photograph
                   ID:  2
             Latitude:  42.3 degrees
            Longitude:  -71.35 degrees
             Altitude:  0 meters
    MinElevationAngle:  0 degrees
       ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
```

```
             Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
        Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
           Receivers:  [1x0 satcom.satellitescenario.Receiver]
            Accesses:  [1x0 matlabshared.satellitescenario.Access]
         MarkerColor:  [1 0.4118 0.1608]
          MarkerSize:  6
           ShowLabel:  true
      LabelFontColor:  [1 1 1]
       LabelFontSize:  15
```

Add a gimbal to the satellite. You can steer this gimbal independently of the satellite.

```
g = gimbal(sat)
```

```
g = 
  Gimbal with properties:

                Name:  Gimbal 3
                  ID:  3
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
      ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
        Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
           Receivers:  [1x0 satcom.satellitescenario.Receiver]
```

Track the location to be photographed using the gimbal.

```
pointAt(g,gs);
```

Add a conical sensor to the gimbal. This sensor represents the camera. Set the field of view to 60 degrees.

```
camSensor = conicalSensor(g,MaxViewAngle=60)
```

```
camSensor = 
  ConicalSensor with properties:

                Name:  Conical sensor 4
                  ID:  4
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
        MaxViewAngle:  60 degrees
            Accesses:  [1x0 matlabshared.satellitescenario.Access]
         FieldOfView:  [0x0 matlabshared.satellitescenario.FieldOfView]
```

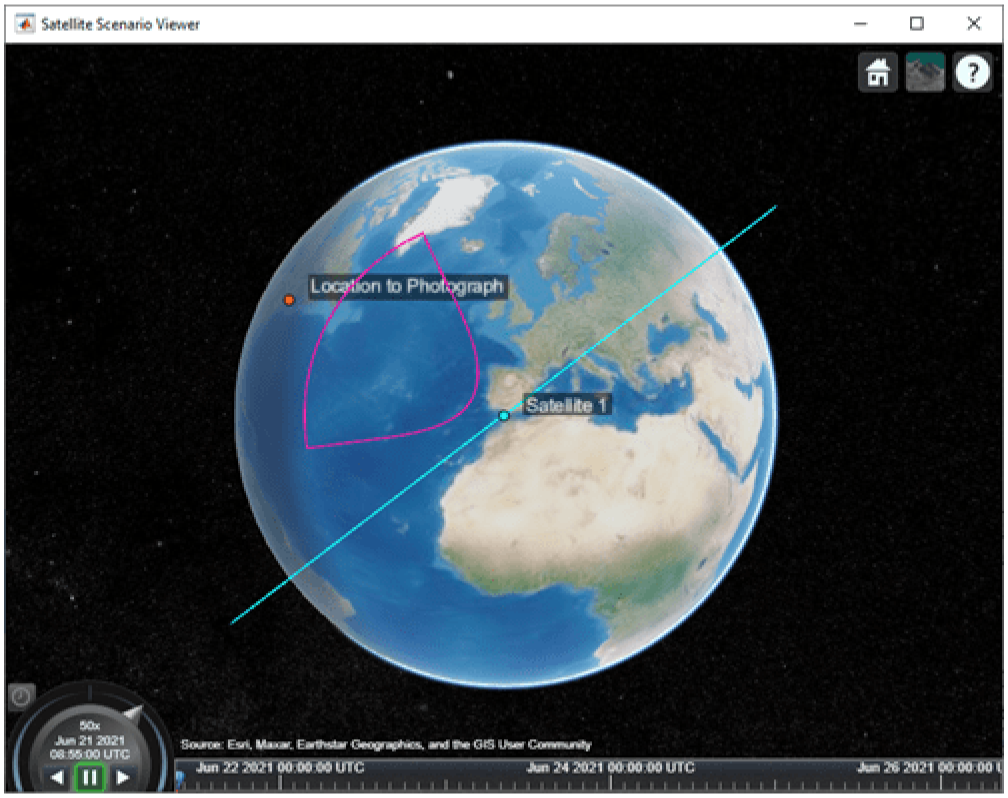
Add access analysis to the conical sensor between the camera and the location to be photographed.

```
ac = access(camSensor,gs)
```

```
ac = 
  Access with properties:

    Sequence:  [4 2]
    LineWidth:  3
    LineColor:  [0.3922 0.8314 0.0745]
```

Visualize the field of view of the camera by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
fieldOfView(camSensor);
```



Determine the intervals during which the camera can see the geographical site.

```
t = accessIntervals(ac)
```

```
t=35×8 table
            Source                      Target               IntervalNumber          StartTime
     _________________     _________________________     ______________     _____________________

    "Conical sensor 4"     "Location to Photograph"             1           21-Jun-2021 10:38:00
    "Conical sensor 4"     "Location to Photograph"             2           21-Jun-2021 12:36:00
    "Conical sensor 4"     "Location to Photograph"             3           21-Jun-2021 14:37:00
    "Conical sensor 4"     "Location to Photograph"             4           21-Jun-2021 16:41:00
    "Conical sensor 4"     "Location to Photograph"             5           21-Jun-2021 18:44:00
    "Conical sensor 4"     "Location to Photograph"             6           21-Jun-2021 20:46:00
    "Conical sensor 4"     "Location to Photograph"             7           21-Jun-2021 22:50:00
    "Conical sensor 4"     "Location to Photograph"             8           22-Jun-2021 09:51:00
    "Conical sensor 4"     "Location to Photograph"             9           22-Jun-2021 11:46:00
    "Conical sensor 4"     "Location to Photograph"            10           22-Jun-2021 13:46:00
    "Conical sensor 4"     "Location to Photograph"            11           22-Jun-2021 15:50:00
    "Conical sensor 4"     "Location to Photograph"            12           22-Jun-2021 17:53:00
    "Conical sensor 4"     "Location to Photograph"            13           22-Jun-2021 19:55:00
    "Conical sensor 4"     "Location to Photograph"            14           22-Jun-2021 21:58:00
    "Conical sensor 4"     "Location to Photograph"            15           23-Jun-2021 10:56:00
    "Conical sensor 4"     "Location to Photograph"            16           23-Jun-2021 12:56:00
```

⋮

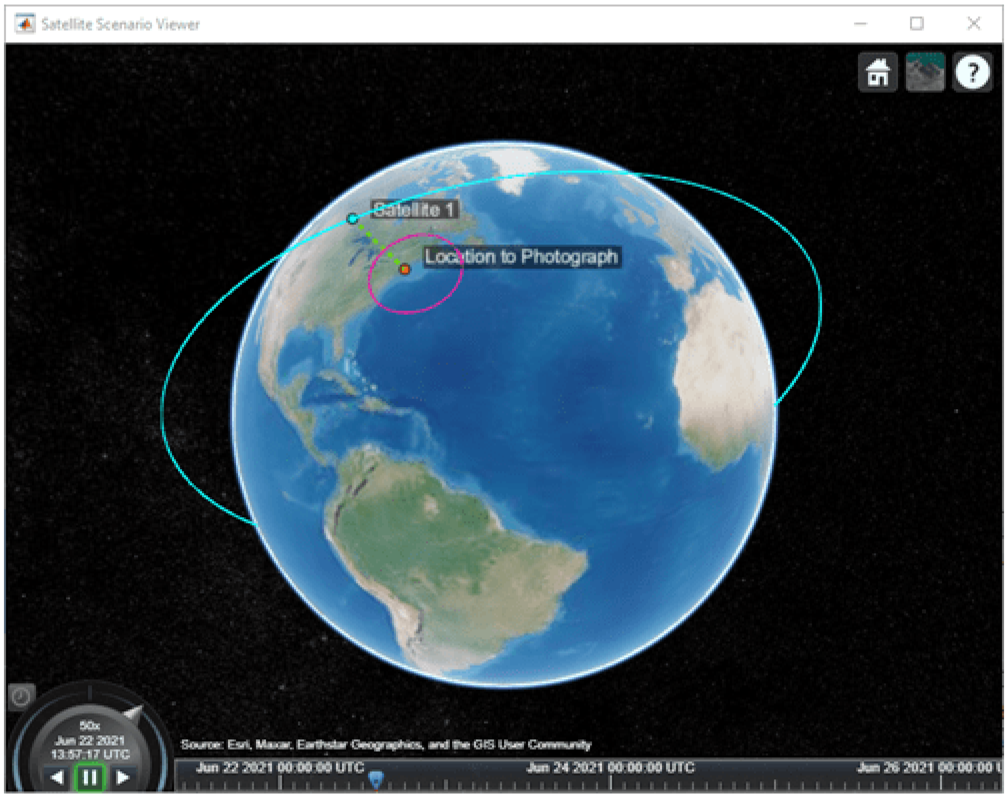
Calculate the maximum revisit time in hours.

```
startTimes = t.StartTime;
endTimes = t.EndTime;
revisitTimes = hours(startTimes(2:end) - endTimes(1:end-1));
maxRevisitTime = max(revisitTimes)                              % hours
```

```
maxRevisitTime = 12.6667
```

Visualize the revisit times that the camera photographs of the location.

```
play(sc);
```



Version History

Introduced in R2021a

See Also

Objects
satelliteScenario | satelliteScenarioViewer

Functions
show | play | hide | access | groundStation | transmitter | receiver

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

FieldOfView

Field of view object belonging to satellite scenario

Description

The FieldOfView object defines a field of view object belonging to a satellite scenario.

Creation

You can create a `FieldOfView` object using the `fieldOfView` object function of the `ConicalSensor` object.

Properties

`LineWidth` — Visual width of field of view contour
`1` (default) | scalar in the range (0 10]

Visual width of the field of view contour in pixels, specified as a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of field of view contour
`[0 1 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of field of view contour, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	🟩

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`"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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`VisibilityMode` — Visibility mode of field of view contour
`'inherit'` (default) | `'manual'`

Visibility mode of the field of view contour, specified as one of these values:

- `'inherit'` — Visibility of the graphic matches that of the parent
- `'manual'` — Visibility of the graphic is not inherited and is independent of that of the parent

Object Functions

show	Show object in satellite scenario viewer
hide	Hide satellite scenario entity from viewer

Examples

Calculate Maximum Revisit Time of Satellite

Create a satellite scenario with a start time of 15-June-2021 8:55:00 AM UTC and a stop time of five days later. Set the simulation sample time to `60` seconds.

```
startTime = datetime(2021,6,21,8,55,0);
stopTime = startTime + days(5);
sampleTime = 60;                                      % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 21-Jun-2021 08:55:00
          StopTime: 26-Jun-2021 08:55:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario using Keplerian orbital elements.

```
semiMajorAxis = 7878137;                                                                % me
eccentricity = 0;
inclination = 50;                                                                       % deg
rightAscensionOfAscendingNode = 0;                                                      % deg
argumentOfPeriapsis = 0;                                                                % deg
trueAnomaly = 50;                                                                       % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  Satellite with properties:

               Name:  Satellite 1
                 ID:  1
     ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
            Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
       Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
          Receivers:  [1x0 satcom.satellitescenario.Receiver]
           Accesses:  [1x0 matlabshared.satellitescenario.Access]
        GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
              Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator:  sgp4
        MarkerColor:  [0.059 1 1]
         MarkerSize:  6
          ShowLabel:  true
     LabelFontColor:  [1 1 1]
      LabelFontSize:  15
```

Add a ground station, which represents the location to be photographed, to the scenario.

```
gs = groundStation(sc,Name="Location to Photograph", ...
    Latitude=42.3001,Longitude=-71.3504)                % degrees
```

```
gs = 
  GroundStation with properties:

                 Name:  Location to Photograph
                   ID:  2
```

```
             Latitude:  42.3 degrees
            Longitude:  -71.35 degrees
             Altitude:  0 meters
    MinElevationAngle:  0 degrees
       ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
              Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
         Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
            Receivers:  [1x0 satcom.satellitescenario.Receiver]
             Accesses:  [1x0 matlabshared.satellitescenario.Access]
          MarkerColor:  [1 0.4118 0.1608]
           MarkerSize:  6
            ShowLabel:  true
       LabelFontColor:  [1 1 1]
        LabelFontSize:  15
```

Add a gimbal to the satellite. You can steer this gimbal independently of the satellite.

```
g = gimbal(sat)
```

```
g = 
  Gimbal with properties:

                Name:  Gimbal 3
                  ID:  3
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
      ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
        Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
           Receivers:  [1x0 satcom.satellitescenario.Receiver]
```

Track the location to be photographed using the gimbal.

```
pointAt(g,gs);
```

Add a conical sensor to the gimbal. This sensor represents the camera. Set the field of view to 60 degrees.

```
camSensor = conicalSensor(g,MaxViewAngle=60)
```

```
camSensor = 
  ConicalSensor with properties:

                Name:  Conical sensor 4
                  ID:  4
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
        MaxViewAngle:  60 degrees
            Accesses:  [1x0 matlabshared.satellitescenario.Access]
         FieldOfView:  [0x0 matlabshared.satellitescenario.FieldOfView]
```

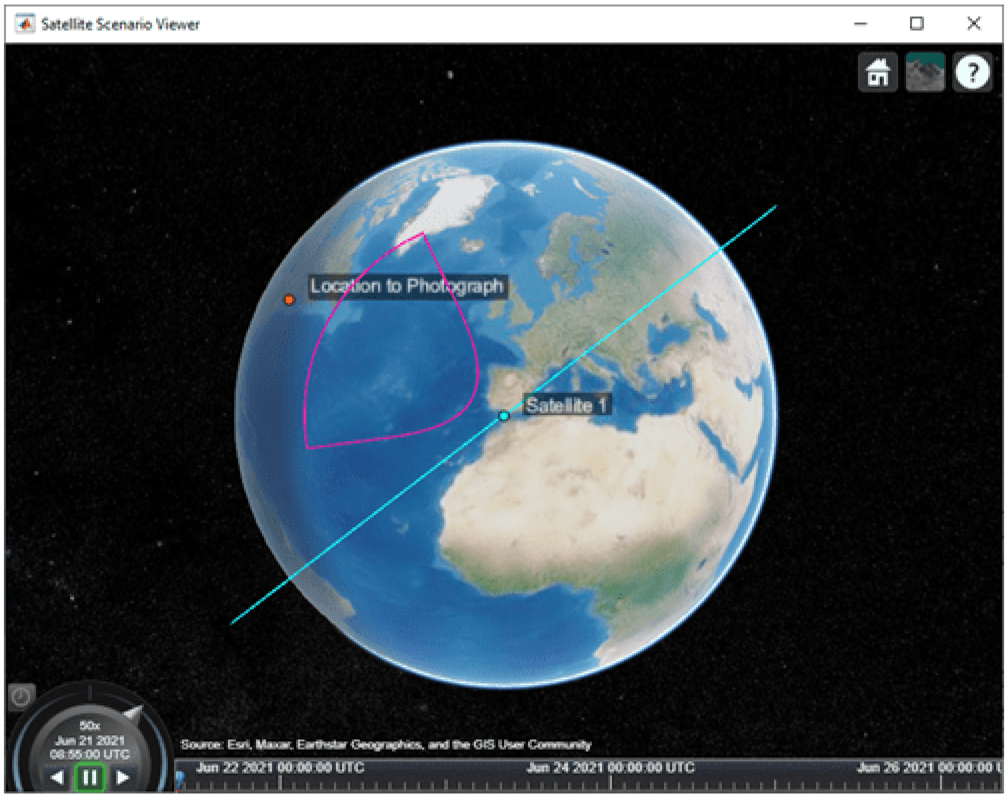
Add access analysis to the conical sensor between the camera and the location to be photographed.

```
ac = access(camSensor,gs)
```

```
ac = 
  Access with properties:
    Sequence:  [4 2]
    LineWidth:  3
    LineColor:  [0.3922 0.8314 0.0745]
```

Visualize the field of view of the camera by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
fieldOfView(camSensor);
```



Determine the intervals during which the camera can see the geographical site.

```
t = accessIntervals(ac)
```

```
t=35×8 table
          Source                    Target              IntervalNumber         StartTime
    __________________    ________________________    ______________    ____________________

    "Conical sensor 4"    "Location to Photograph"           1          21-Jun-2021 10:38:00
    "Conical sensor 4"    "Location to Photograph"           2          21-Jun-2021 12:36:00
    "Conical sensor 4"    "Location to Photograph"           3          21-Jun-2021 14:37:00
    "Conical sensor 4"    "Location to Photograph"           4          21-Jun-2021 16:41:00
    "Conical sensor 4"    "Location to Photograph"           5          21-Jun-2021 18:44:00
    "Conical sensor 4"    "Location to Photograph"           6          21-Jun-2021 20:46:00
    "Conical sensor 4"    "Location to Photograph"           7          21-Jun-2021 22:50:00
    "Conical sensor 4"    "Location to Photograph"           8          22-Jun-2021 09:51:00
    "Conical sensor 4"    "Location to Photograph"           9          22-Jun-2021 11:46:00
    "Conical sensor 4"    "Location to Photograph"          10          22-Jun-2021 13:46:00
    "Conical sensor 4"    "Location to Photograph"          11          22-Jun-2021 15:50:00
    "Conical sensor 4"    "Location to Photograph"         12          22-Jun-2021 17:53:00    2
    "Conical sensor 4"    "Location to Photograph"         13          22-Jun-2021 19:55:00    2
    "Conical sensor 4"    "Location to Photograph"         14          22-Jun-2021 21:58:00    2
    "Conical sensor 4"    "Location to Photograph"         15          23-Jun-2021 10:56:00    2
    "Conical sensor 4"    "Location to Photograph"         16          23-Jun-2021 12:56:00    2
      ⋮
```

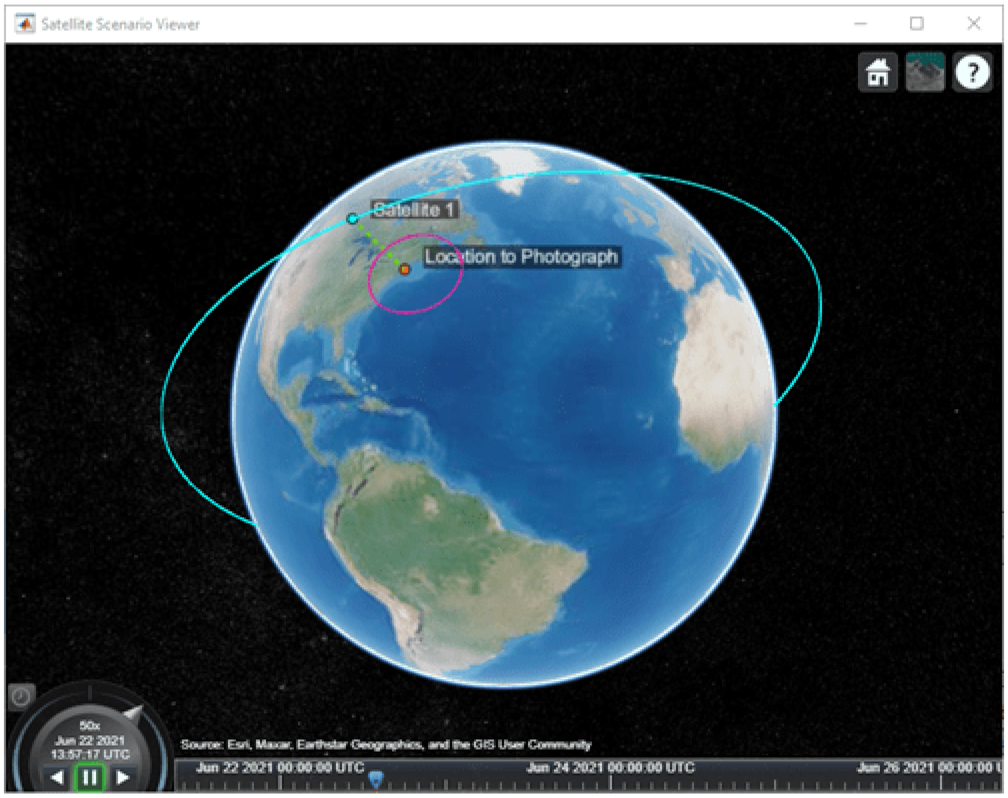
Calculate the maximum revisit time in hours.

```
startTimes = t.StartTime;
endTimes = t.EndTime;
revisitTimes = hours(startTimes(2:end) - endTimes(1:end-1));
maxRevisitTime = max(revisitTimes)                             % hours
```

```
maxRevisitTime = 12.6667
```

Visualize the revisit times that the camera photographs of the location.

```
play(sc);
```



Version History

Introduced in R2021a

See Also

Objects
satelliteScenario | satelliteScenarioViewer

Functions
show | play | hide | groundStation | access

Topics
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

Gimbal

Gimbal object belonging to satellite scenario

Description

The `Gimbal` defines a gimbal object belonging to a satellite scenario.

Creation

You can create a `Gimbal` object using the `gimbal` object function of the `Satellite` or `GroundStation`.

Properties

`Name` — Gimbal name
`"Gimbal` *`idx`*`"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `Gimbal` function. After you call the `gimbal` function, this property is read-only.

Gimbal name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one Gimbal, specify `Name` as a string scalar or a character vector.
- If you are adding multiple gimbals, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All gimbals that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of gimbals that you are adding. Each gimbal is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`ID` — Gimbal ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Gimbal ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one gimbal, the `MountingLocation` property is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of gimbal.
- If you are adding multiple gimbals, the `MountingLocation` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified gimbals. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the gimbals. The columns correspond to the mounting location of each specified gimbal and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one gimbal, the `MountingAngles` property is a three-element vector.
- If you are adding multiple gimbals the `MountingAngles` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified gimbals. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the gimbals. The columns correspond to the mounting angles of each specified gimbal and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

`ConicalSensors` — Conical sensors
row vector of conical sensors

You can set this property only when calling the `conicalSensor`. After you call the `conicalSensor` function, this property is read-only.

Conical sensors attached to the Gimbal, specified as a row vector of conical sensors.

`Transmitters` — Transmitters attached to Gimbal
row vector of `Transmitter` objects

You can set this property only when calling `transmitter` function. After you call the `transmitter` function, this property is read-only.

Transmitters attached to the Gimbal, specified as a row vector of `Transmitter` objects.

`Receivers` — Receivers attached to the satellite
row vector of `Receiver` objects

You can set this property only when calling the `receiver`. After you call the `receiver` function, this property is read-only.

Receivers attached to the satellite, specified as a row vector of `Receiver` objects.

`CoordinateAxes` — Coordinate axes triad graphic object
`CoordinateAxes` object (default)

You can set this property only when calling `coordinateAxes`. After you call `coordinateAxes`, this property is read-only.

Coordinate axes triad graphic object, specified as `CoordinateAxes` object.

Object Functions

aer	Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame
conicalSensor	Add conical sensor to satellite scenario
gimbalAngles	Steering angles of gimbal
pointAt	Point satellite at target
CoordinateAxes	Coordinate axes triad graphic object
receiver	Add receiver to satellite scenario
transmitter	Add transmitter to satellite scenario

Examples

Calculate Maximum Revisit Time of Satellite

Create a satellite scenario with a start time of 15-June-2021 8:55:00 AM UTC and a stop time of five days later. Set the simulation sample time to `60` seconds.

```
startTime = datetime(2021,6,21,8,55,0);
stopTime = startTime + days(5);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 21-Jun-2021 08:55:00
          StopTime: 26-Jun-2021 08:55:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario using Keplerian orbital elements.

```
semiMajorAxis = 7878137;                                                             % me
eccentricity = 0;
inclination = 50;                                                                    % deg
rightAscensionOfAscendingNode = 0;                                                   % deg
argumentOfPeriapsis = 0;                                                             % deg
trueAnomaly = 50;                                                                    % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  Satellite with properties:

               Name:  Satellite 1
                 ID:  1
     ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
            Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
       Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
          Receivers:  [1x0 satcom.satellitescenario.Receiver]
           Accesses:  [1x0 matlabshared.satellitescenario.Access]
        GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
              Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator:  sgp4
        MarkerColor:  [0.059 1 1]
         MarkerSize:  6
          ShowLabel:  true
     LabelFontColor:  [1 1 1]
      LabelFontSize:  15
```

Add a ground station, which represents the location to be photographed, to the scenario.

```
gs = groundStation(sc,Name="Location to Photograph", ...
    Latitude=42.3001,Longitude=-71.3504)                    % degrees
```

```
gs = 
  GroundStation with properties:

                 Name:  Location to Photograph
                   ID:  2
             Latitude:  42.3 degrees
            Longitude:  -71.35 degrees
             Altitude:  0 meters
    MinElevationAngle:  0 degrees
       ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
              Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
         Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
            Receivers:  [1x0 satcom.satellitescenario.Receiver]
             Accesses:  [1x0 matlabshared.satellitescenario.Access]
          MarkerColor:  [1 0.4118 0.1608]
           MarkerSize:  6
            ShowLabel:  true
       LabelFontColor:  [1 1 1]
        LabelFontSize:  15
```

Add a gimbal to the satellite. You can steer this gimbal independently of the satellite.

```
g = gimbal(sat)
```

```
g = 
  Gimbal with properties:

                Name:  Gimbal 3
                  ID:  3
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
      ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
        Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
           Receivers:  [1x0 satcom.satellitescenario.Receiver]
```

Track the location to be photographed using the gimbal.

```
pointAt(g,gs);
```

Add a conical sensor to the gimbal. This sensor represents the camera. Set the field of view to 60 degrees.

```
camSensor = conicalSensor(g,MaxViewAngle=60)
```

```
camSensor = 
  ConicalSensor with properties:

                Name:  Conical sensor 4
                  ID:  4
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
        MaxViewAngle:  60 degrees
            Accesses:  [1x0 matlabshared.satellitescenario.Access]
         FieldOfView:  [0x0 matlabshared.satellitescenario.FieldOfView]
```

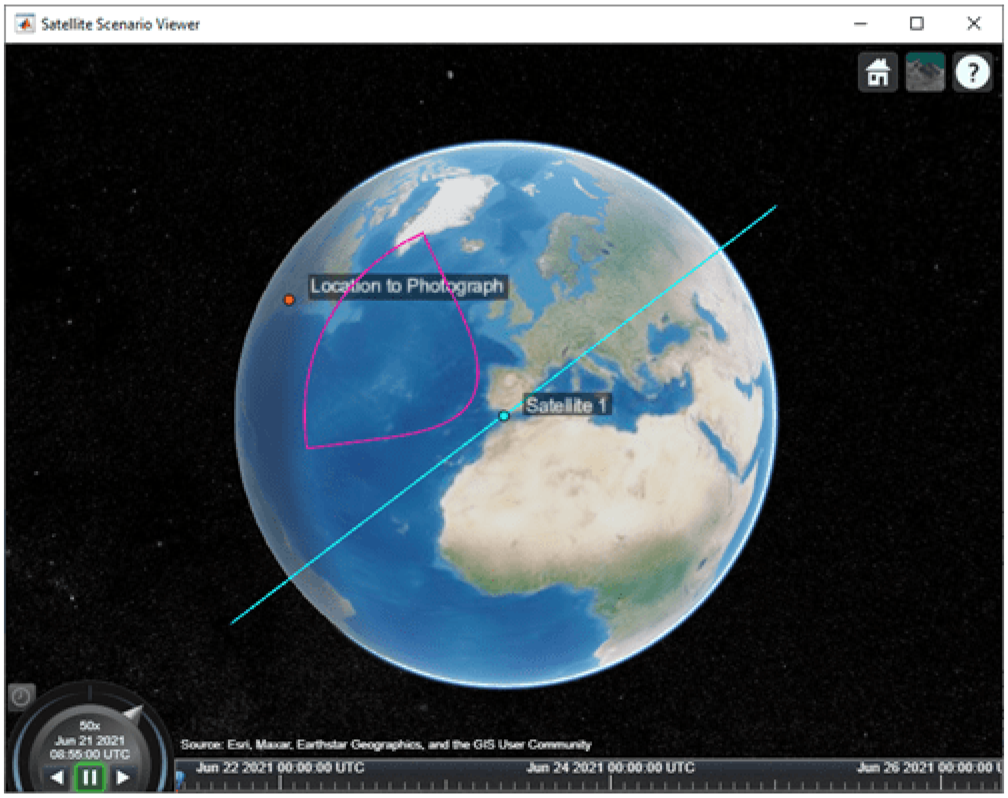
Add access analysis to the conical sensor between the camera and the location to be photographed.

```
ac = access(camSensor,gs)
```

```
ac = 
  Access with properties:

    Sequence:  [4 2]
    LineWidth:  3
    LineColor:  [0.3922 0.8314 0.0745]
```

Visualize the field of view of the camera by using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
fieldOfView(camSensor);
```



Determine the intervals during which the camera can see the geographical site.

```
t = accessIntervals(ac)
```

```
t=35×8 table
          Source                    Target              IntervalNumber         StartTime
    ________________     _______________________     ______________     ____________________

    "Conical sensor 4"    "Location to Photograph"           1          21-Jun-2021 10:38:00
    "Conical sensor 4"    "Location to Photograph"           2          21-Jun-2021 12:36:00
    "Conical sensor 4"    "Location to Photograph"           3          21-Jun-2021 14:37:00
    "Conical sensor 4"    "Location to Photograph"           4          21-Jun-2021 16:41:00
    "Conical sensor 4"    "Location to Photograph"           5          21-Jun-2021 18:44:00
    "Conical sensor 4"    "Location to Photograph"           6          21-Jun-2021 20:46:00
    "Conical sensor 4"    "Location to Photograph"           7          21-Jun-2021 22:50:00
    "Conical sensor 4"    "Location to Photograph"           8          22-Jun-2021 09:51:00
    "Conical sensor 4"    "Location to Photograph"           9          22-Jun-2021 11:46:00
    "Conical sensor 4"    "Location to Photograph"          10          22-Jun-2021 13:46:00
    "Conical sensor 4"    "Location to Photograph"          11          22-Jun-2021 15:50:00
    "Conical sensor 4"    "Location to Photograph"          12          22-Jun-2021 17:53:00
    "Conical sensor 4"    "Location to Photograph"          13          22-Jun-2021 19:55:00
    "Conical sensor 4"    "Location to Photograph"          14          22-Jun-2021 21:58:00
    "Conical sensor 4"    "Location to Photograph"          15          23-Jun-2021 10:56:00
    "Conical sensor 4"    "Location to Photograph"          16          23-Jun-2021 12:56:00
      ⋮
```

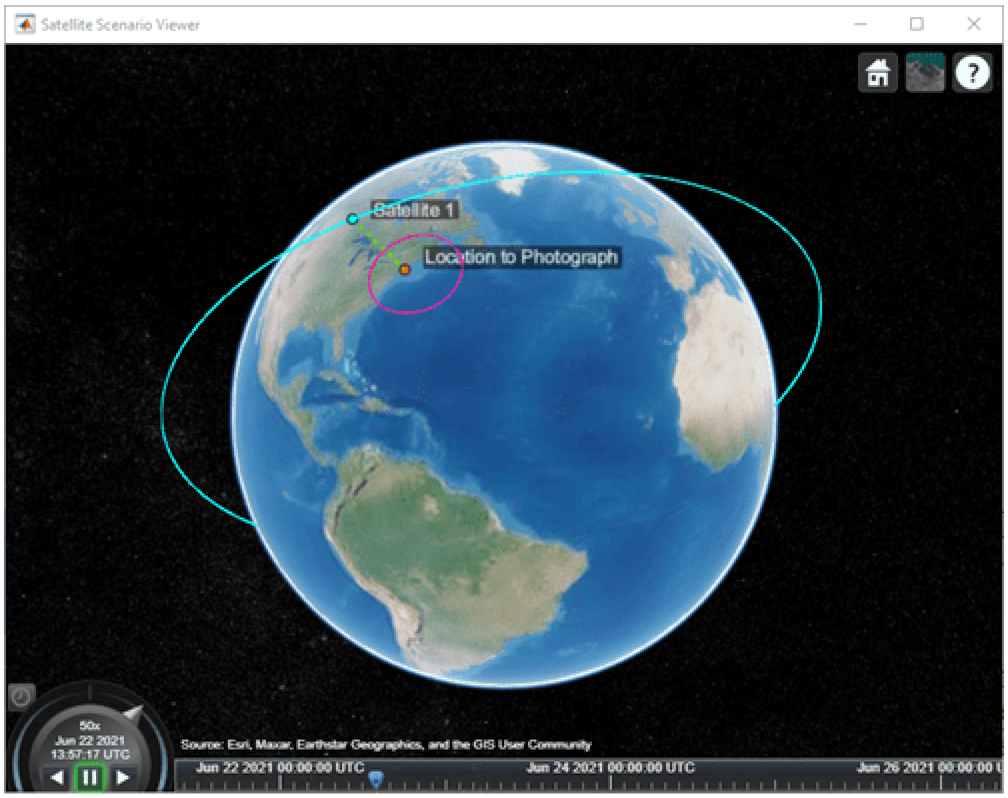
Calculate the maximum revisit time in hours.

```
startTimes = t.StartTime;
endTimes = t.EndTime;
revisitTimes = hours(startTimes(2:end) - endTimes(1:end-1));
maxRevisitTime = max(revisitTimes)                             % hours
```

```
maxRevisitTime = 12.6667
```

Visualize the revisit times that the camera photographs of the location.

```
play(sc);
```



Version History

Introduced in R2021a

See Also

Objects

satelliteScenario | satelliteScenarioViewer

Functions

show | play | hide | satellite | access | groundStation | conicalSensor | transmitter | receiver

Topics

“Satellite Scenario Key Concepts”

“Satellite Scenario Basics”

GroundTrack

Ground track object belonging to satellite in scenario

Description

`GroundTrack` defines a ground track object belonging to a satellite in a scenario.

Creation

You can create a `GroundTrack` object using the `groundTrack` object function of the `Satellite` object.

Properties

`LeadTime` — Period of ground track to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track to be visualized in the satellite scenario viewer, specified as `'LeadTime'` and a positive scalar in seconds.

`TrailTime` — Period of ground track history to be visualized
`StartTime` to `StopTime` (default) | positive scalar

Period of the ground track history to be visualized in `Viewer`, specified as `'TrailTime'` and a positive scalar in seconds.

`LineWidth` — Visual width of ground track
`1` (default) | scalar in the range (0 10]

Visual width of the ground track in pixels, specified as `'LineWidth'` and a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LeadLineColor` — Color of future ground track line
`[1 0 1]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the future ground track line, specified as `'LeadLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`TrailLineColor` — Color of ground track line history
`[1 0.5 0]` (default) | RGB triplet | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the ground track line history, specified as `'TrailLineColor'` and an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`VisibilityMode` — Visibility mode of ground track
`'inherit'` (default) | `'manual'`

Visibility mode of the ground track, specified as either one of these values:

- `'inherit'` — Visibility of the graphic matches that of the parent.
- `'manual'` — Visibility of the graphic is not inherited and is independent of that of the parent.

Object Functions

show	Show object in satellite scenario viewer
hide	Hide satellite scenario entity from viewer

Examples

Add Ground Track to Satellite in Geosynchronous Orbit

Create a satellite scenario object.

```
startTime = datetime(2020,5,10);
stopTime = startTime + days(5);
sampleTime = 60;                                       % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Calculate the semimajor axis of the geosynchronous satellite.

```
earthAngularVelocity = 0.0000729211585530;                                                  % rad/s
orbitalPeriod = 2*pi/earthAngularVelocity;                                                  % seconds
earthStandardGravitationalParameter = 398600.4418e9;                                        % m^3/s^2
semiMajorAxis = (earthStandardGravitationalParameter*((orbitalPeriod/(2*pi))^2))^(1/3);
```

Define the remaining orbital elements of the geosynchronous satellite.

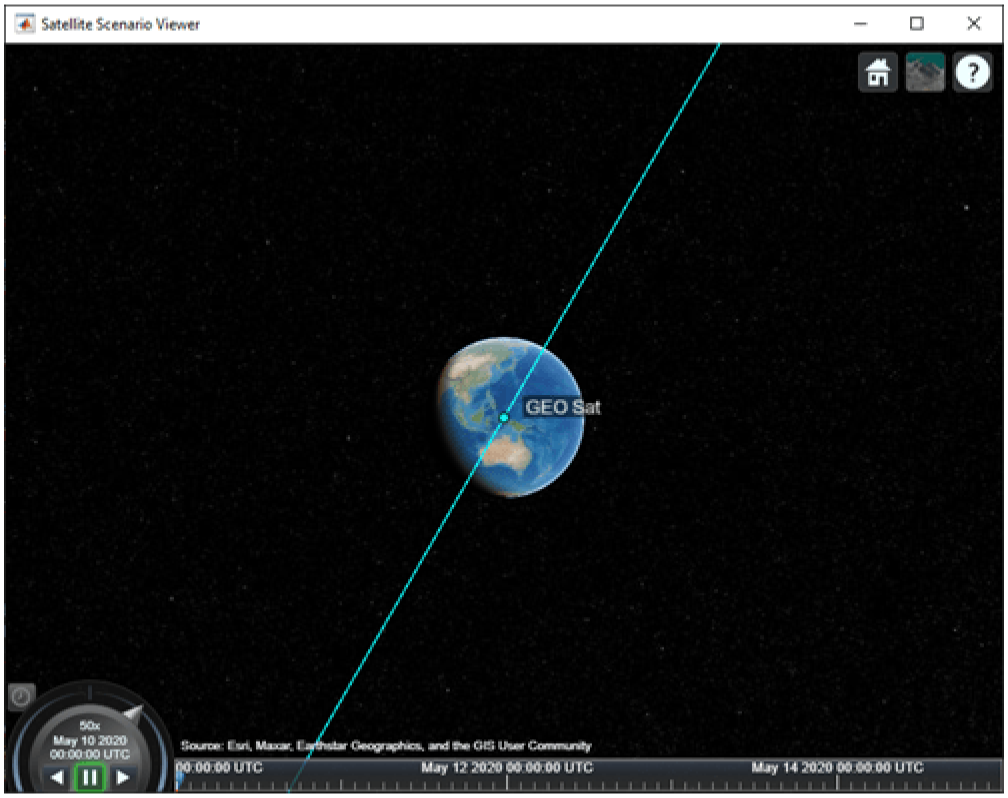
```
eccentricity = 0;
inclination = 60;                   % degrees
rightAscensionOfAscendingNode = 0; % degrees
argumentOfPeriapsis = 0;            % degrees
trueAnomaly = 0;                    % degrees
```

Add the geosynchronous satellite to the scenario.

```
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode,...
        argumentOfPeriapsis,trueAnomaly,"OrbitPropagator","two-body-keplerian","Name","GEO Sat")
```

Visualize the scenario using the Satellite Scenario Viewer.

```
v = satelliteScenarioViewer(sc);
```


Satellite Scenario Viewer
GEO Sat
50x
May 10 2020
00:00:00 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
00:00:00 UTC
May 12 2020 00:00:00 UTC
May 14 2020 00:00:00 UTC

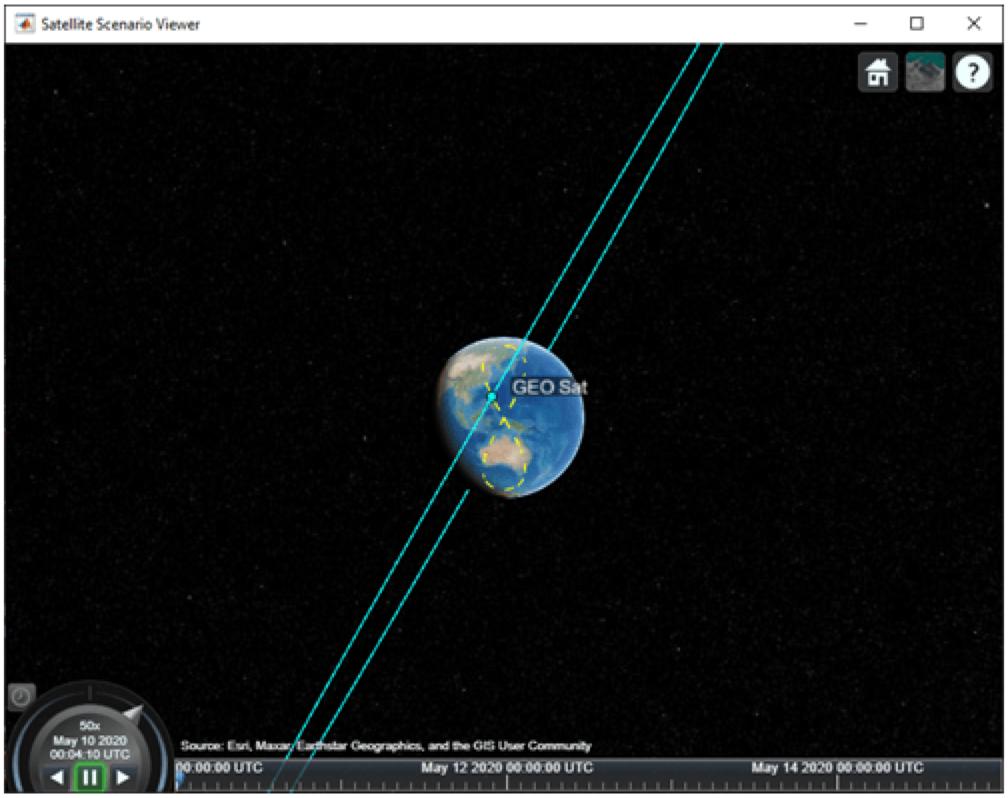
Add a ground track of the satellite to the visualization and adjust how much of the future and history of the ground track to display.

```
leadTime = 2*24*3600;                                          % seconds
trailTime = leadTime;
gt = groundTrack(sat,"LeadTime",leadTime,"TrailTime",trailTime)
```

```
gt = 
  GroundTrack with properties:

          LeadTime: 172800
         TrailTime: 172800
         LineWidth: 1
     LeadLineColor: [1 1 0.0670]
    TrailLineColor: [1 1 0.0670]
    VisibilityMode: 'inherit'
```

Visualize the satellite movement and its trace on the ground. The satellite covers the area around Japan during one half of the day and Australia during the other half.

```
play(sc);
```


Satellite Scenario Viewer
GEO Sat
50x
May 10 2020
00:04:10 UTC
Source: Esri, Maxar, Earthstar Geographics, and the GIS User Community
00:00:00 UTC
May 12 2020 00:00:00 UTC
May 14 2020 00:00:00 UTC

Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `groundStation` | `access` | `hide` | `satellite`

Topics
"Model, Visualize, and Analyze Satellite Scenario"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

GroundStation

Ground station object belonging to satellite scenario

Description

The `GroundStation` object defines a ground station object belonging to a satellite scenario.

Creation

You can create `GroundStation` object using the `groundStation` object function of the `satelliteScenario` object.

Properties

`Name` — GroundStation name
`"GroundStation *idx*"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `groundStation` function. After you call `groundStation` function, this property is read-only.

GroundStation name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one GroundStation is added, specify `Name` as a string scalar or a character vector.
- If multiple GroundStations are added, specify `Name` as a string scalar, character vector, string vector or a cell array of character vectors. All GroundStations added as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vector must equal the number of GroundStations being added. Each GroundStation is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by satellite scenario.

Data Types: `char` | `string`

`ID` — GroundStation ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

GroundStation ID assigned by the simulator, specified as a positive scalar.

`Latitude` — Geodetic latitude of ground stations
`42.3001` (default) | scalar | row vector

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Geodetic latitude of ground stations, specified as a scalar. Values must be in the range [-90, 90].

- If you add only one ground station, specify Latitude as a scalar double.
- If you add multiple ground stations, specify Latitude as a vector double whose length is equal to the number of ground stations being added.

When latitude and longitude are specified as `lat, lon` inputs to GroundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`Longitude` — Geodetic longitude of ground stations
`-71.3504` (default) | scalar | row vector

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Geodetic longitude of ground stations, specified as a scalar or a vector. Values must be in the range [-180, 180].

- If you add only one ground station, specify longitude as a scalar.
- If you add multiple ground stations, specify longitude as a vector whose length is equal to the number of ground stations being added.

When longitude and longitude are specified as `lat, lon` inputs to GroundStation, longitude specified as a name-value argument takes precedence.

Data Types: `double`

`Altitude` — Altitude of ground station
`0` m (default) | scalar | vector

You can set this property only when calling GroundStation. After you call GroundStation, this property is read-only.

Altitude of ground stations, specified as a scalar or a vector.

- If you specify `Altitude` as a scalar, the value is assigned to each ground station in the GroundStation.
- If you specify `Altitude` as a vector, the vector length must be equal to the number of ground stations in the GroundStation.

When latitude and longitude are specified as `lat, lon` inputs to GroundStation, Latitude specified as a name-value argument takes precedence.

Data Types: `double`

`CoordinateAxes` — Coordinate axes triad graphic object
`CoordinateAxes` object (default)

You can set this property only when calling `coordinateAxes`. After you call `coordinateAxes`, this property is read-only.

Coordinate axes triad graphic object, specified as `CoordinateAxes` object.

`MinElevationAngle` — Minimum elevation angle
`0` (default) | scalar | vector

Minimum elevation angle of a satellite for the satellite to be visible from the ground station, and for the ground station to be visible from the satellite in degrees, specified as a scalar or row vector. Values must be in the range [–90, 90]. For access and link closure to be possible, the elevation angle must be at least equal to the value specified in `MinElevationAngle`.

- If you specify `MinElevationAngle` as a scalar, the value is assigned to each ground station in the GroundStation.
- If you specify `MinElevationAngle` as a vector, the vector length must be equal to the number of ground stations in the GroundStation.

When the `AutoSimulate` property of the satellite scenario is `false`, `MinElevationAngle` property can be modified while the `SimulationStatus` is `NotStarted` or `InProgress`.

Data Types: `double`

`Accesses` — Access analysis objects
row vector of `Access` objects

You can set this property only when calling `GroundStation`. After you call `GroundStation`, this property is read-only.

Access analysis objects, specified as a row vector of `Access` objects.

`ConicalSensors` — Conical sensors
row vector of conical sensors

You can set this property only when calling the `conicalSensor`. After you call the `conicalSensor` function, this property is read-only.

Conical sensors attached to the GroundStation, specified as a row vector of conical sensors.

`Gimbals` — Gimbals
row vector of `Gimbal` objects

You can set this property only when calling `gimbal`. After you call `gimbal`, this property is read-only.

Gimbals attached to the GroundStation, specified as the comma-separated pair consisting of `'Gimbals'` and a row vector of `Gimbal` objects.

`Transmitters` — Transmitters attached to GroundStation
row vector of `Transmitter` objects

You can set this property only when calling `transmitter` function. After you call the `transmitter` function, this property is read-only.

Transmitters attached to the GroundStation, specified as a row vector of `Transmitter` objects.

`Receivers` — Receivers attached to the satellite
row vector of `Receiver` objects

You can set this property only when calling the `receiver`. After you call the `receiver` function, this property is read-only.

Receivers attached to the satellite, specified as a row vector of `Receiver` objects.

`MarkerColor` — Color of marker

`[1 0 0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the marker, specified as a comma-separated pair consisting of `'MarkerColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

`MarkerSize` — Size of marker

`10` (default) | positive scalar less than 30

Size of the marker, specified as a comma-separated pair consisting of `'MarkerSize'` and a real positive scalar less than 30. The unit is in pixels.

`ShowLabel` — State of GroundStation label visibility
`true` or `1` (default) | `false` or `0`

State of GroundStation label visibility, specified as a comma-separated pair consisting of `'ShowLabel'` and numerical or logical value of `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LabelFontSize` — Font size of GroundStation label
`15` (default) | positive scalar in the range [6 30]

Font size of the GroundStation label, specified as a comma-separated pair consisting of `'LabelFontSize'` and a positive scalar in the range [6 30].

`LabelFontColor` — Font color of GroundStation label
`[1,0,0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Font color of the GroundStationlabel, specified as a comma-separated pair consisting of `'LabelFontColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Object Functions

access	Add access analysis objects to satellite scenario
conicalSensor	Add conical sensor to satellite scenario
transmitter	Add transmitter to satellite scenario
receiver	Add receiver to satellite scenario
gimbal	Add gimbal to satellite or ground station
coordinateAxes	Visualize coordinate axes triad of satellite scenario assets
show	Show object in satellite scenario viewer
aer	Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame
hide	Hide satellite scenario entity from viewer
dopplershift	Calculate Doppler shift at target asset in satellite scenario
latency	Calculate propagation delay from one asset to another asset

Examples

Add Ground Stations to Scenario and Visualize Access Intervals

Create a satellite scenario and add ground stations from latitudes and longitudes.

```
startTime = datetime(2020,5,1,11,36,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
lat = 10;
lon = -30;
gs = groundStation(sc,lat,lon);
```

Add satellites using Keplerian elements.

```
semiMajorAxis = 10000000;
eccentricity = 0;
inclination = 10;
rightAscensionOfAscendingNode = 0;
argumentOfPeriapsis = 0;
trueAnomaly = 0;
sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
        rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly);
```

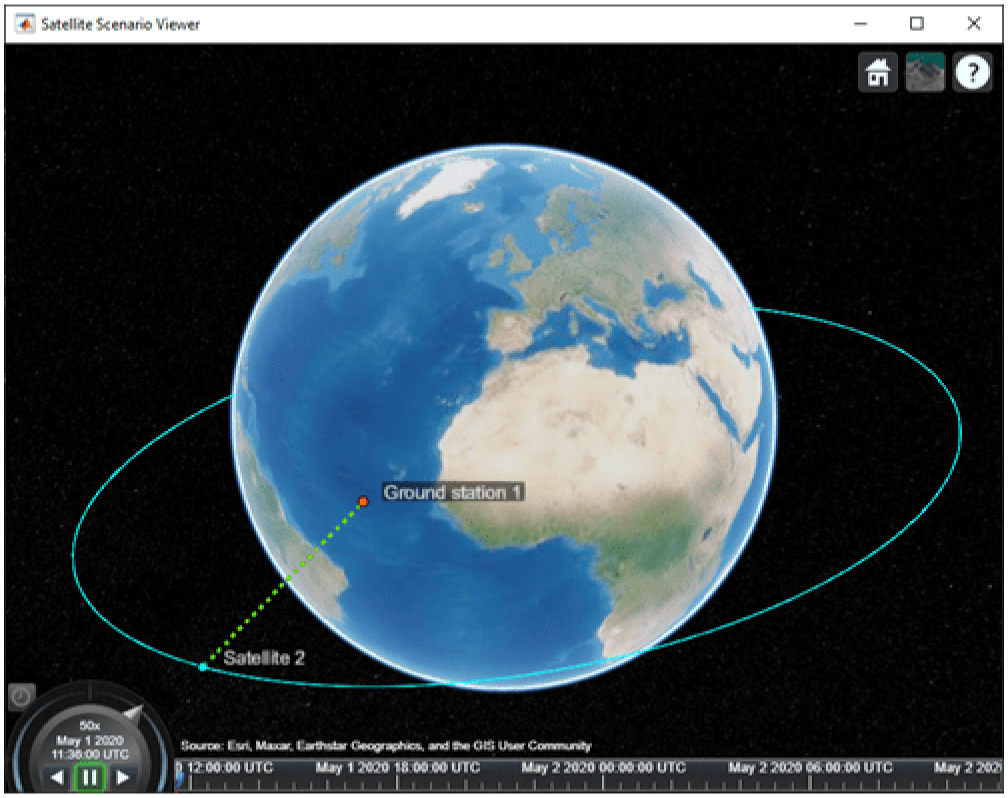
Add access analysis to the scenario and obtain the table of intervals of access between the satellite and the ground station.

```
ac = access(sat,gs);
intvls = accessIntervals(ac)
```

```
intvls=8×8 table
       Source                Target           IntervalNumber         StartTime               EndTi
    _____________    ____________________    ______________    ____________________    ___________

    "Satellite 2"    "Ground station 1"            1           01-May-2020 11:36:00    01-May-2020
    "Satellite 2"    "Ground station 1"            2           01-May-2020 14:20:00    01-May-2020
    "Satellite 2"    "Ground station 1"            3           01-May-2020 17:27:00    01-May-2020
    "Satellite 2"    "Ground station 1"            4           01-May-2020 20:34:00    01-May-2020
    "Satellite 2"    "Ground station 1"            5           01-May-2020 23:41:00    02-May-2020
    "Satellite 2"    "Ground station 1"            6           02-May-2020 02:50:00    02-May-2020
    "Satellite 2"    "Ground station 1"            7           02-May-2020 05:59:00    02-May-2020
    "Satellite 2"    "Ground station 1"            8           02-May-2020 09:06:00    02-May-2020
```

Play the scenario to visualize the ground stations.

```
play(sc)
```



Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide` | `satellite` | `access` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics
"Multi-Hop Satellite Communications Link Between Two Ground Stations"
"Satellite Constellation Access to Ground Station"
"Comparison of Orbit Propagators"
"Modeling Satellite Constellations Using Ephemeris Data"
"Estimate GNSS Receiver Position with Simulated Satellite Constellations"
"Model, Visualize, and Analyze Satellite Scenario"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

Link

Link analysis object belonging to Transmitter

Description

The `Link` object defines a link analysis object belonging to `Transmitter`.

Creation

You can create a `Link` object using the `link` object function of the `Transmitter` or `Receiver` objects.

Properties

Sequence — Transmitter or receiver ID
vector of positive numbers

You can set this property only when calling Link. After you call Link, this property is read-only.

Transmitter or receiver ID, specified as a vector of positive numbers.

`LineWidth` — Visual width of link line
`1` (default) | scalar in the range (0 10]

Visual width of link line in pixels, specified as a scalar in the range (0 10].

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`LineColor` — Color of link line
`[0 1 0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the link line, specified as an RGB triplet, a hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

Object Functions

ebno	Eb/No at final node of link
linkPercentage	Percentage of time when link between first and last node in link analysis is closed
linkIntervals	Intervals during which link is closed
linkStatus	Status of link closure between first and last node
show	Show object in satellite scenario viewer
sigstrength	Calculate received signal strength at last node of link
hide	Hide satellite scenario entity from viewer

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                                  % mete
eccentricity = 0;
inclination = 60;                                                                          % deg
rightAscensionOfAscendingNode = 0;                                                         % deg
argumentOfPeriapsis = 0;                                                                   % deg
trueAnomaly = 0;                                                                           % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                                        % dB/K
systemLoss = 3;                                                                         % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)
```

```
rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                   RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                                     % Hz
power = 20;                                                                           % dBW
```

```
bitRate = 20;                                                                   % Mbps
systemLoss = 3;                                                                 % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)
```

```
txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                              % meters
apertureEfficiency = 0.5;
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                                              % degrees
longitude = 0.1487094;                                              % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                   % Hz
power = 40;                                                                         % dBW
bitRate = 20;                                                                       % Mbp
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                               % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                                 % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
```

```
lnk = 
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

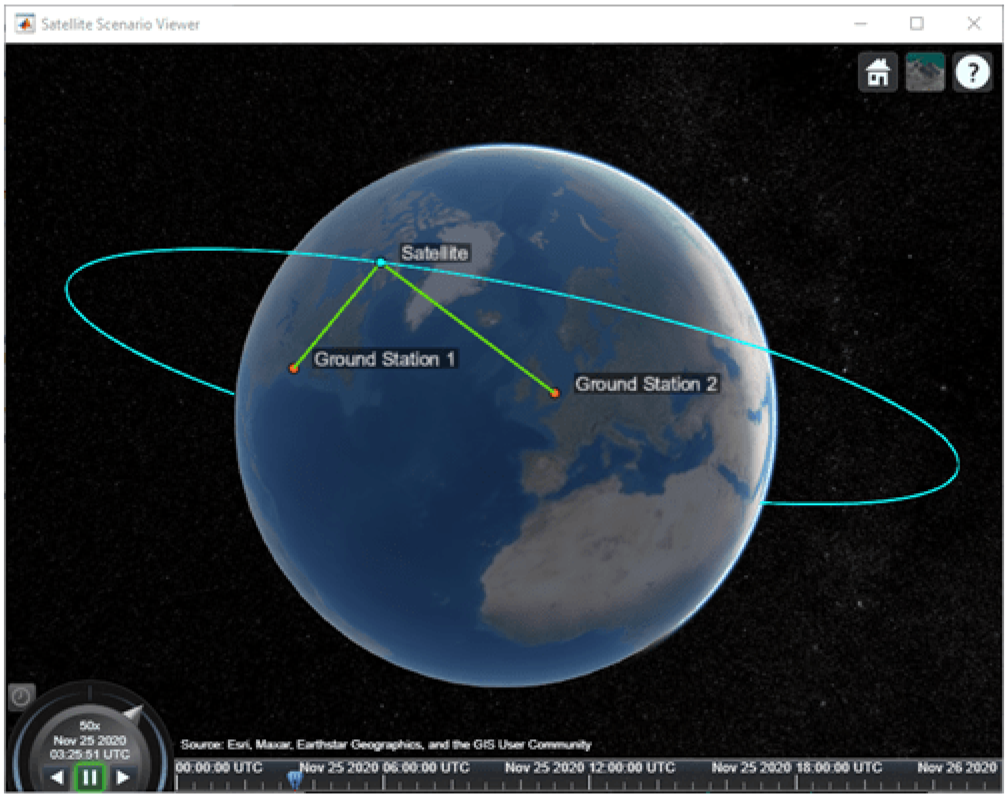
Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)
```

```
ans=4×8 table
                Source                          Target               IntervalNumber         Star
    ______________________________    ___________________________    ______________    _________

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          1           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          2           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          3           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          4           25-Nov-202
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Version History

Introduced in R2021a

R2022b: sigstrength object function added to the object

You can now compute the received signal strength at the last node of the link using `sigstrength` object function.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`play` | `groundStation` | `transmitter` | `receiver`

Topics
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

Receiver

Receiver object belonging to satellite scenario

Description

The `Receiver` object defines a receiver object function belonging to the satellite scenario.

Creation

You can create `Receiver` object using the `receiver` object function of the `Satellite`, `GroundStation`, or `Gimbal` object.

Properties

`Name` — Receiver name
`"Receiver *idx*"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `receiver` function. After you call the `receiver` function, this property is read-only.

Receiver name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one receiver, specify `Name` as a string scalar or a character vector.
- If you are adding multiple receivers, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All receivers that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of receivers that you are adding. Each receiver is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`ID` — Receiver ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Receiver ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one receiver, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of receiver.
- If you are adding multiple receivers, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified receivers. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the receivers. The columns correspond to the mounting location of each specified receiver and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one receiver, the `MountingAngles` property is a three-element vector.
- If you are adding multiple receivers the `MountingAngles` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified receivers. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the receivers. The columns correspond to the mounting angles of each specified receiver and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

Antenna — Antenna object associated with Receiver
scalar | vector

`Antenna` object associated with the receiver, specified as either a scalar or a vector. This object can be the default `gaussianAntenna` object, or one from the Antenna Toolbox or Phased Array System Toolbox. The default Gaussian antenna has a dish diameter of 1 meter and an aperture efficiency of 0.65.

Antenna can be specified in receiver as a name-value argument consisting of `'Antenna'` and a scalar, antenna or phased array objects.

- If you are adding only one receiver, `Antenna` must be a scalar.
- If you are adding multiple receivers, `Antenna` as a vector. The same antenna is assigned to all receivers.

SystemLoss — System loss in receiver
5 (default) | scalar | vector

System loss in dB, specified as a scalar or a vector. SystemLoss must be greater than or equal to PreReceiverLoss.

System loss can be specified in receiver function as a name-value argument consisting of 'SystemLoss' and a scalar, or a vector.

- If you are adding only one receiver, SystemLoss is a scalar.
- If you are adding multiple receivers, SystemLoss is a scalar or a vector. When SystemLoss is a scalar, the same SystemLoss is assigned to all receivers. When SystemLoss is a vector, its length must equal the number of receivers and each element of SystemLoss is assigned to the corresponding receivers in the parent.

If you specify PreReceiverLoss property as a name value argument in receiver function , the default value is the greater of 5 dB and the specified PreReceiverLoss value.

When the AutoSimulate property of the satellite scenario is false, you can modify the SystemLoss value while SimulationStatus is NotStarted or InProgress.

PreReceiverLoss — Pre-receiver loss
3 (default) | scalar | vector

Pre-receiver loss in dB, specified as a scalar or a vector. This is the total loss before the receiver input in the receiver system, such as feeder loss, radome loss, and loss due to polarization mismatch. PreReceiverLoss must be less than or equal to SystemLoss.

Pre-receiver loss can be specified in receiver function as a name-value pair consisting of 'PreReceiverLoss' and a scalar, or a vector.

- If you are adding only one receiver, PreReceiverLoss is a scalar.
- If you are adding multiple receivers, PreReceiverLoss is a scalar or a vector. When PreReceiverLoss is a scalar, the same PreReceiverLoss is assigned to all receivers. When PreReceiverLoss is a vector, its length must equal the number of receivers and each element of PreReceiverLoss is assigned to the corresponding receivers in the parent.

If you specify SystemLoss property as a name value argument in receiver function , the default value is the lesser of 3 dB and the specified SystemLoss value.

When the AutoSimulate property of the satellite scenario is false, you can modify the PreReceiverLoss value while SimulationStatus is NotStarted or InProgress.

GainToNoiseTemperatureRatio — Gain to noise temperature ratio
3 (default) | scalar | vector

Gain to noise temperature ratio of the antenna in dB per Kelvin, specified as the name-value argument consisting of 'GainToNoiseTemperatureRatio' and a scalar or a vector.

- If you are adding only one receiver, GainToNoiseTemperatureRatio is a scalar.
- If you are adding multiple receivers, GainToNoiseTemperatureRatio is a scalar, or a vector. When GainToNoiseTemperatureRatio is a scalar, the same GainToNoiseTemperatureRatio is assigned to all receivers. When GainToNoiseTemperatureRatio is a vector, its length must

equal the number of receivers and each element of `GainToNoiseTemperatureRatio` is assigned to the corresponding receiver in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `GainToNoiseTemperatureRatio` value while `SimulationStatus` is `NotStarted` or `InProgress`.

`CoordinateAxes` — Coordinate axes triad graphic object
`CoordinateAxes` object (default)

You can set this property only when calling `coordinateAxes`. After you call `coordinateAxes`, this property is read-only.

Coordinate axes triad graphic object, specified as `CoordinateAxes` object.

`RequiredEbNo` — Minimum Eb/No necessary for link closure
`10` (default) | scalar | vector

Minimum energy per bit to noise power spectral density ratio (Eb/No) necessary for link closure in dB, specified as the name-value pair consisting of `'RequiredEbNo'` and a scalar or a vector.

- If you are adding only one receiver, `RequiredEbNo` is a scalar.
- If you are adding multiple receivers, `RequiredEbNo` is a scalar or a vector. When `RequiredEbNo` is a scalar, the same `RequiredEbNo` is assigned to all receivers. When `RequiredEbNo` is a vector, its length must equal the number of receivers and each element of `RequiredEbNo` is assigned to the corresponding receiver in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, the `RequiredEbNo` property can be modified while `SimulationStatus` is `NotStarted` or `InProgress`.

Note The above properties except `ID` can be specified as name-value arguments in `receiver`. The size of specified name-value pairs determines the number of receivers specified. Refer to these properties to understand how they must be defined when specifying multiple receivers.

Object Functions

aer	Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame
gaussianAntenna	Add Gaussian antennas
pattern	Plot 3-D radiation pattern of antenna
pointAt	Point transmitter or receiver at target
pointAt	Point satellite at target
coordinateAxes	Visualize coordinate axes triad of satellite scenario assets

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
```

```
sampleTime = 60;                                    % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                   % mete
eccentricity = 0;
inclination = 60;                                                           % deg
rightAscensionOfAscendingNode = 0;                                          % deg
argumentOfPeriapsis = 0;                                                    % deg
trueAnomaly = 0;                                                            % deg
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                          % dB/K
systemLoss = 3;                                                           % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)
```

```
rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                    RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                        % Hz
power = 20;                                                              % dBW
bitRate = 20;                                                            % Mbps
```

```
systemLoss = 3;                                                                       % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)

txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                                    % meters
apertureEfficiency = 0.5;
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                              % degrees
longitude = 0.1487094;                              % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                          % Hz
power = 40;                                                                                % dBW
bitRate = 20;                                                                              % Mbp
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                                     % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                                % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
```

```
lnk = 
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

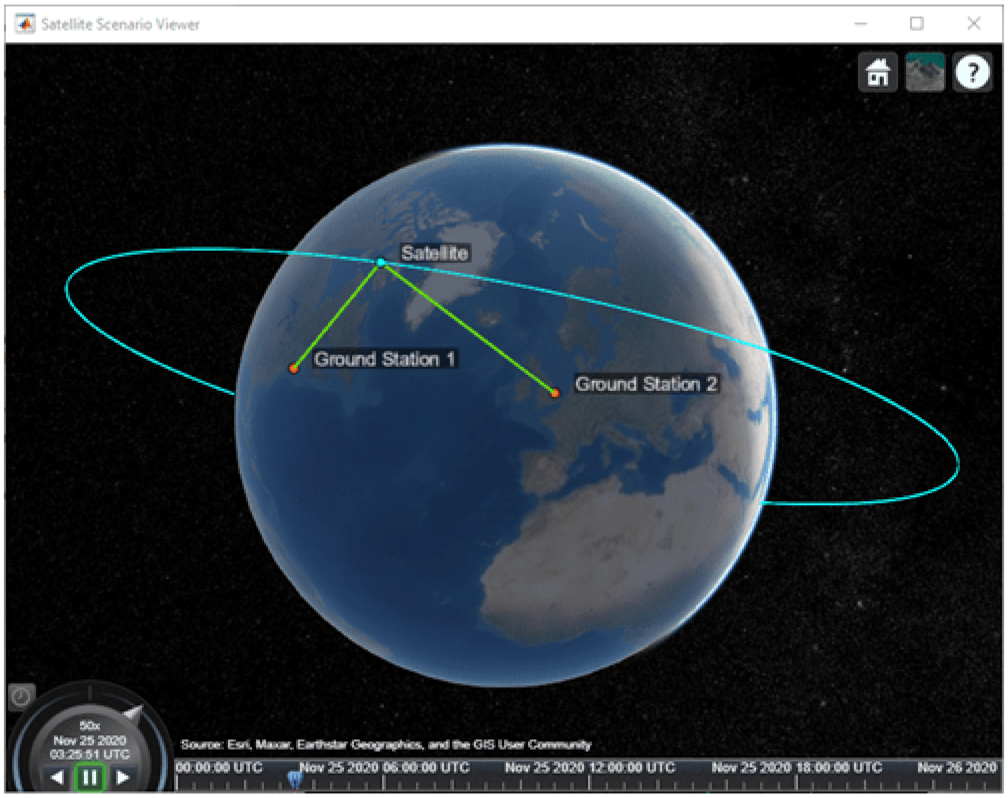
Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)
```

```
ans=4×8 table
                Source                            Target              IntervalNumber       Star
    ______________________________    ___________________________    ______________    _________

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          1           25-Nov-20
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          2           25-Nov-20
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          3           25-Nov-20
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          4           25-Nov-20
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Version History

Introduced in R2021a

R2022b: PreReceiverLoss property added to the object

You can now specify the pre-receiver loss in dB using the `PreReceiverLoss` property.

See Also

Objects
`satelliteScenario` | `satelliteScenarioViewer`

Functions
`groundStation` | `link` | `play` | `transmitter`

Topics
“Multi-Hop Satellite Communications Link Between Two Ground Stations”
“Satellite Scenario Key Concepts”
“Satellite Scenario Basics”

satelliteScenario

Satellite scenario

Description

The `satelliteScenario` object represents a 3D arena consisting of satellites, ground stations, and the interactions between them. Use this object to model satellite constellations, model ground station networks, perform access analyses between the satellites and the ground stations, and visualize the results.

Creation

Syntax

```
sc = satelliteScenario
sc = satelliteScenario(startTime,stopTime,sampleTime)
sc = satelliteScenario( ___ ,AutoSimulate=false)
```

Description

`sc = satelliteScenario` creates a default satellite scenario object.

`sc = satelliteScenario(startTime,stopTime,sampleTime)` sets the `StartTime`, `StopTime`, and `SampleTime` properties to the values of `startTime`, `stopTime`, and `sampleTime`, respectively.

`sc = satelliteScenario( ___ ,AutoSimulate=false)` sets the `AutoSimulate` property to a specified value.

Properties

`StartTime` — Start time of satellite scenario simulation in UTC
`datetime` scalar

Start time of the satellite scenario simulation in UTC, specified as a `datetime` scalar.

The default `StartTime` is the current UTC time if no satellites are present in the scenario. Otherwise, it is the earliest value among

- the current UTC time.
- the epoch defined in the TLE files.
- the reference time deduced from the GPS week number defined in SEM files.
- the earliest time deduced from RINEX navigation data for GPS and Galileo.
- the initial time in the timetable and timeseries.

If the `StartTime`, `StopTime`, or `SampleTime` properties are explicitly specified, the `StartTime` property no longer updates with further additions of satellites.

When the AutoSimulate property is false, you can modify the StartTime property only when the SimulationStatus is NotStarted. You can use the restart function to reset SimulationStatus to NotStarted, but doing so erases the simulation data.

Data Types: datetime

StopTime — Stop time of satellite scenario simulation in UTC
datetime scalar

Stop time of the satellite scenario simulation in UTC, specified as a datetime scalar. The default StopTime is StartTime + longest orbital period among the satellites in the scenario. If no satellites are added to the scenario, the default StopTime is the same as the default StartTime. If the StartTime, StopTime, or SampleTime properties are explicitly specified, the StopTime property no longer updates with further additions of satellites.

When the AutoSimulate property is false, you can modify the StopTime property only when the SimulationStatus is NotStarted. You can use the restart function to reset SimulationStatus to NotStarted, but doing so erases the simulation data.

Data Types: datetime

SampleTime — Sample time of satellite scenario simulation
scalar

Sample time of the satellite scenario simulation, specified as a real-valued scalar. The default sample time is set such that there are 100 samples between StartTime and StopTime. If the default StartTime and StopTime are the same, which is the case when no satellites are added to the scenario, the default SampleTime is 60 seconds. If the StartTime, StopTime, or SampleTime properties are explicitly specified, the SampleTime property no longer updates with further additions of satellites.

When the AutoSimulate property is false, you can modify the SampleTime property only when the SimulationStatus is NotStarted. You can use the restart function to reset SimulationStatus to NotStarted, but doing so erases the simulation data.

Data Types: double

SimulationTime — Simulation time of satellite scenario in UTC
current UTC time (default) | datetime scalar

This property is read-only.

Current simulation time of the satellite scenario simulation in UTC, specified as a datetime scalar. Call the restart function to reset the time to StartTime.

Dependencies

To enable this property, set AutoSimulate to false.

Data Types: datetime

SimulationStatus — Simulation status
'NotStarted' (default) | 'InProgress' | 'Completed'

This property is read-only.

Simulation status of the satellite scenario, specified as one of the following:

- 'NotStarted' — No call to the advance function has been made
- 'InProgress' — Simulation is running
- 'Completed' — Simulation is finished

The simulation starts when the first call to the advance function is made. The simulation continues until one of the following occurs:

- The simulation reaches the StopTime.
- A new asset is added to the satellite scenario.
- Certain properties of the asset (satellites, ground stations, gimbals, conical sensors, and so on) have been modified, such as MountingLocation or MountingAngles. Refer to the properties to determine if modifying them can stop the simulation.

Call the restart function to restart the simulation, erase the simulation data, and set SimulationStatus to NotStarted.

Dependencies

To enable this property, set AutoSimulate to false.

AutoSimulate — Option to simulate satellite scenario automatically
true or 1 (default) | false or 0

Option to simulate the satellite scenario automatically, specified as one of these numeric or logical values.

- 1 (true) — Simulate the satellite scenario automatically on any call to an analysis function, such as states or accessIntervals.
- 0 (false)— Simulate the satellite scenario only by calling the advance function.

Changing the AutoSimulate value erases the previous simulation data.

Data Types: double

Satellites — Satellites in the scenario
row vector of Satellite objects

Satellites in the scenario, returned as a row vector of Satellite objects. To create a Satellite object and add it to the satellite scenario, use the satellite object function. After adding satellites to the scenario, this property is read-only.

GroundStations — Ground stations in scenario
row vector of GroundStation objects

Ground stations in the scenario, returned as a row vector of GroundStation objects. To create a GroundStation object and add it to the satellite scenario, use the groundStation object function. After adding ground stations to the scenario, this property is read-only.

AutoShow — Option to automatically show graphics
true or 1 (default) | false or 0

Option to automatically show graphics, specified as a logical 1 (true) or 0 (false). This property determines if entities added to the scenario are automatically shown in an open satelliteScenarioViewer window.

Object Functions

groundStation	Add ground station to satellite scenario
satellite	Add satellites to satellite scenario
satelliteScenarioViewer	Create viewer for satellite scenario
advance	Move simulation forward by one sample time
restart	Restart simulation from beginning
play	Play satellite scenario simulation results on viewer

Examples

Create Satellite Scenario with Custom Start and Stop Times

Specify the start time in the current time zone as yesterday. The simulation lasts for half a day.

```
startTime = datetime("yesterday","TimeZone","local");
stopTime = startTime + days(0.5);
```

Specify the sample time as 60 seconds. Create a satellite scenario object, specifying the start time, stop time, and sample time.

```
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 02-Mar-2023 05:00:00
          StopTime: 02-Mar-2023 17:00:00
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1x0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1x0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0x0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add Satellites to Scenario Using Keplerian Elements

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC, and the stop time set to one day later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + days(1);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add two satellites to the scenario using their Keplerian elements.

```
semiMajorAxis = [10000000;15000000];
eccentricity = [0.01;0.02];
inclination = [0;10];
rightAscensionOfAscendingNode = [0;15];
argumentOfPeriapsis = [0;30];
```

```
trueAnomaly = [0;20];

sat = satellite(sc,semiMajorAxis,eccentricity,inclination, ...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly)
```

```
sat = 
  1x2 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

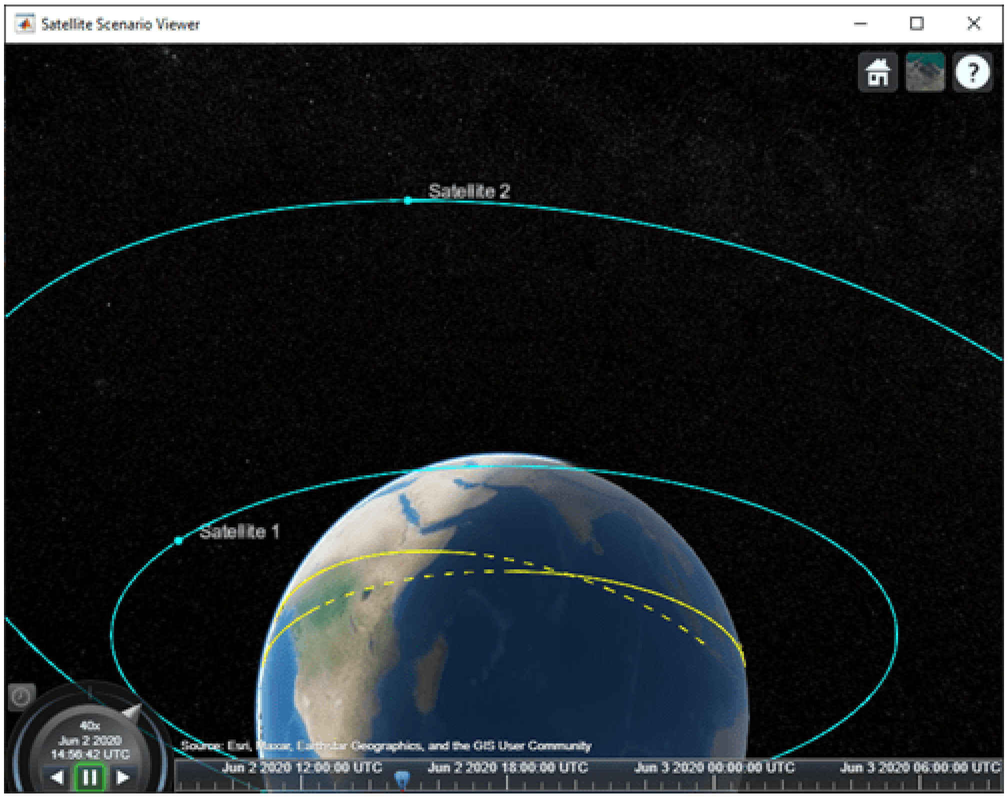
View the satellites in orbit and the ground tracks over one hour.

```
show(sat)
groundTrack(sat,LeadTime=3600)
```

```
ans=1×2 object
  1×2 GroundTrack array with properties:

    LeadTime
    TrailTime
    LineWidth
    LeadLineColor
    TrailLineColor
    VisibilityMode
```

Play the scenario and set the animation speed of the simulation to 40.

```
play(sc,PlaybackSpeedMultiplier=40)
```



Manual Simulation of Satellite Scenario

Create a satellite scenario object and set the `AutoSimulate` property to `false` to enable manual simulation of the satellite scenario.

```
startTime = datetime(2022,1,12);
stopTime = startTime + days(0.5);
sampleTime = 60;                             % Seconds
sc = satelliteScenario('AutoSimulate',false);
```

Add a GPS satellite constellation to the scenario.

```
sat = satellite(sc,"gpsAlmanac.txt");
```

Simulate the scenario using the `advance` function.

```
while advance(sc)
end
```

Obtain the satellite position histories.

```
p = states(sat);
```

`AutoSimulate` is `false`, so restart the scenario before adding a ground station.

```
restart(sc);
```

Add a ground station to the scenario.

```
gs = groundStation(sc);
```

Add access analysis between each satellite and ground station.

```
ac = access(sat,gs);
```

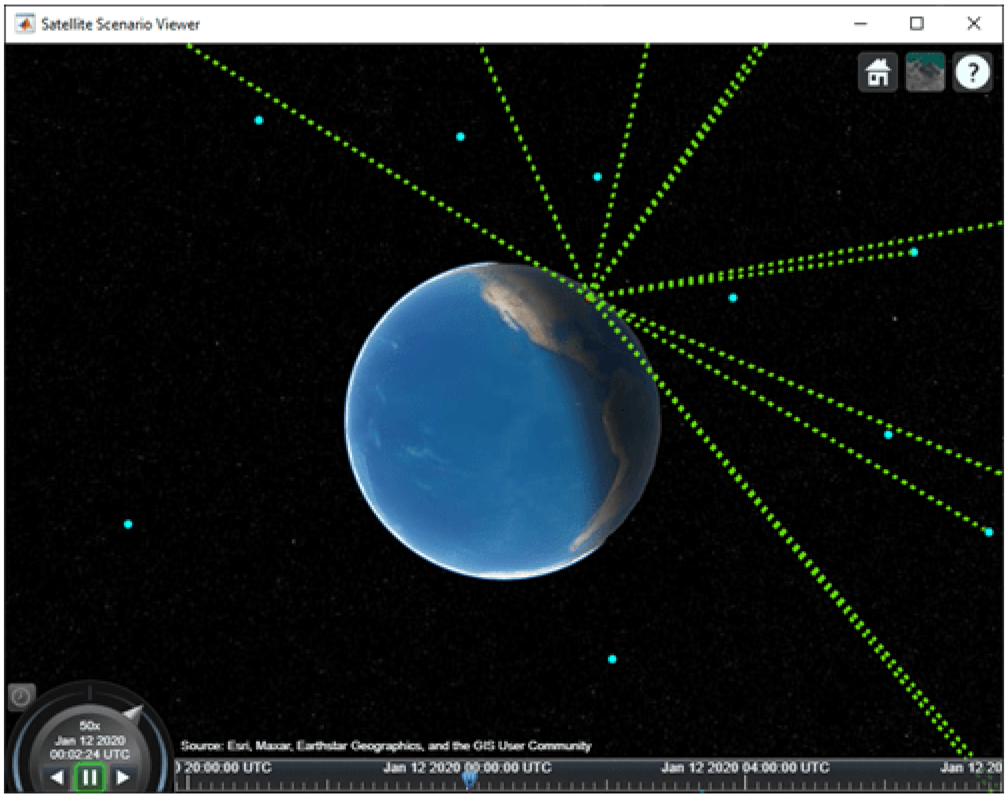
Simulate the scenario and determine the access intervals.

```
while advance(sc)
end
intvls1 = accessIntervals(ac)
```

```
intvls1=35×8 table
     Source            Target            IntervalNumber         StartTime                EndTime
    ________    ____________________    ______________    ____________________    ________________

    "PRN:1"     "Ground station 32"           1           11-Jan-2020 23:20:25    12-Jan-2020 05:1
    "PRN:2"     "Ground station 32"           1           12-Jan-2020 04:03:16    12-Jan-2020 07:4
    "PRN:3"     "Ground station 32"           1           11-Jan-2020 19:50:06    11-Jan-2020 21:3
    "PRN:3"     "Ground station 32"           2           12-Jan-2020 01:52:43    12-Jan-2020 06:4
    "PRN:4"     "Ground station 32"           1           11-Jan-2020 19:50:06    12-Jan-2020 00:1
    "PRN:4"     "Ground station 32"           2           12-Jan-2020 04:54:02    12-Jan-2020 07:4
    "PRN:5"     "Ground station 32"           1           12-Jan-2020 05:52:03    12-Jan-2020 07:4
    "PRN:6"     "Ground station 32"           1           12-Jan-2020 02:43:29    12-Jan-2020 07:4
    "PRN:7"     "Ground station 32"           1           11-Jan-2020 21:09:52    12-Jan-2020 03:2
    "PRN:8"     "Ground station 32"           1           11-Jan-2020 20:33:36    12-Jan-2020 03:1
    "PRN:9"     "Ground station 32"           1           11-Jan-2020 19:50:06    12-Jan-2020 00:4
    "PRN:9"     "Ground station 32"           2           12-Jan-2020 05:08:32    12-Jan-2020 07:4
    "PRN:10"    "Ground station 32"           1           12-Jan-2020 00:32:56    12-Jan-2020 01:5
    "PRN:11"    "Ground station 32"           1           11-Jan-2020 22:15:09    12-Jan-2020 04:3
    "PRN:12"    "Ground station 32"           1           12-Jan-2020 04:32:16    12-Jan-2020 07:4
    "PRN:13"    "Ground station 32"           1           12-Jan-2020 00:03:56    12-Jan-2020 02:5
      ⋮
```

Visualize the simulation results.

```
v = satelliteScenarioViewer(sc,'ShowDetails',false);
play(sc);
```



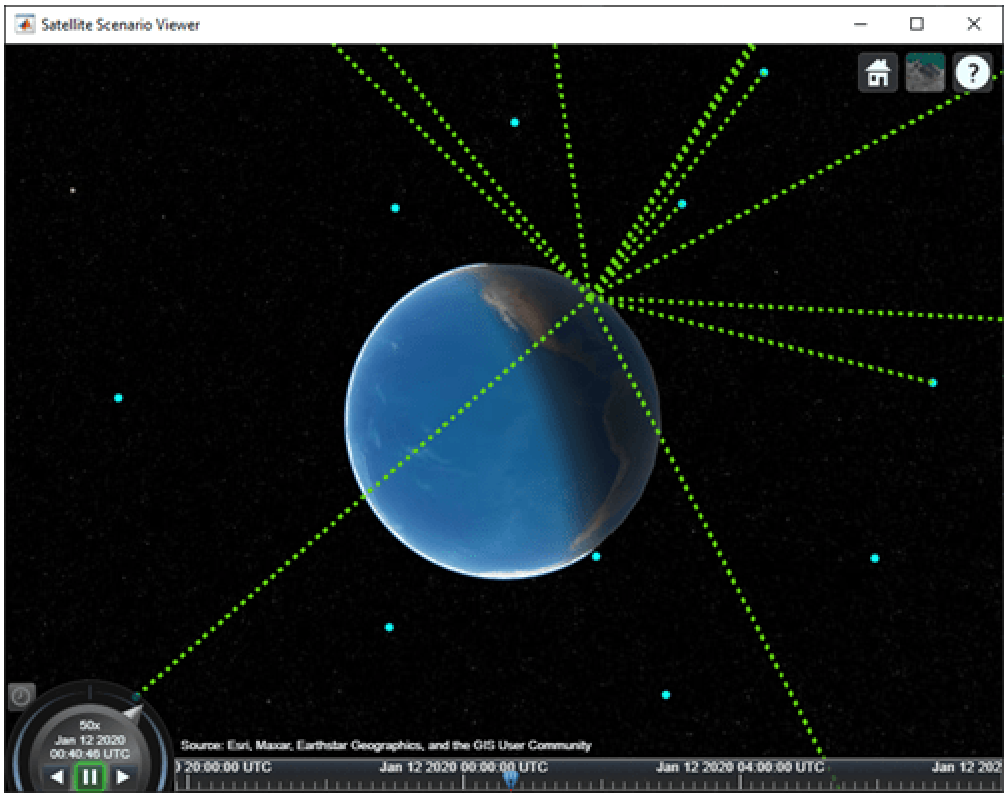
Verify that the access intervals are the same when you set the `AutoSimulate` property to `true`.

```
sc.AutoSimulate = true;
intvls2 = accessIntervals(ac)
```

```
intvls2=35×8 table
     Source            Target            IntervalNumber         StartTime                EndTime
    _________    ____________________    ______________    ____________________    ____________

    "PRN:1"      "Ground station 32"           1           11-Jan-2020 23:20:25    12-Jan-2020 05:
    "PRN:2"      "Ground station 32"           1           12-Jan-2020 04:03:16    12-Jan-2020 07:
    "PRN:3"      "Ground station 32"           1           11-Jan-2020 19:50:06    11-Jan-2020 21:
    "PRN:3"      "Ground station 32"           2           12-Jan-2020 01:52:43    12-Jan-2020 06:
    "PRN:4"      "Ground station 32"           1           11-Jan-2020 19:50:06    12-Jan-2020 00:
    "PRN:4"      "Ground station 32"           2           12-Jan-2020 04:54:02    12-Jan-2020 07:
    "PRN:5"      "Ground station 32"           1           12-Jan-2020 05:52:03    12-Jan-2020 07:
    "PRN:6"      "Ground station 32"           1           12-Jan-2020 02:43:29    12-Jan-2020 07:
    "PRN:7"      "Ground station 32"           1           11-Jan-2020 21:09:52    12-Jan-2020 03:
    "PRN:8"      "Ground station 32"           1           11-Jan-2020 20:33:36    12-Jan-2020 03:
    "PRN:9"      "Ground station 32"           1           11-Jan-2020 19:50:06    12-Jan-2020 00:
    "PRN:9"      "Ground station 32"           2           12-Jan-2020 05:08:32    12-Jan-2020 07:
    "PRN:10"     "Ground station 32"           1           12-Jan-2020 00:32:56    12-Jan-2020 01:
    "PRN:11"     "Ground station 32"           1           11-Jan-2020 22:15:09    12-Jan-2020 04:
    "PRN:12"     "Ground station 32"           1           12-Jan-2020 04:32:16    12-Jan-2020 07:
    "PRN:13"     "Ground station 32"           1           12-Jan-2020 00:03:56    12-Jan-2020 02:
       ⋮
```

Visualize the scenario.

```
play(sc);
```



Tips

- When saving the satellite scenario, either save the entire workspace containing the scenario object or save the scenario object itself.

Version History

Introduced in R2021a

See Also

Objects
`satellite` | `satelliteScenarioViewer`

Functions
`play` | `show` | `hide` | `advance` | `restart` | `access` | `groundStation`

Topics
"Model, Visualize, and Analyze Satellite Scenario"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

satelliteScenarioViewer

Create viewer for satellite scenario

Description

The `satelliteScenarioViewer` object represents a 3D visualization of the `satelliteScenario` object. Use this object to focus the camera onto the satellite or the ground station by double-clicking the satellite or ground station. Once focused, you can rotate the camera and it will orbit around the selected satellite/ground station and not be allowed to pan out. To unfocus, you must double-click elsewhere in the viewer or press the home button.

Creation

Syntax

```
satelliteScenarioViewer(scenario)
v = satelliteScenarioViewer(scenario)
satelliteScenarioViewer(scenario,Name=Value)
```

Description

`satelliteScenarioViewer(scenario)` creates a 3-D or 2-D satellite scenario viewer for the specified satellite scenario. Satellite Scenario Viewer is a 3-D map display and requires hardware graphics support for WebGL™.

`v = satelliteScenarioViewer(scenario)` returns the handle to the satellite scenario viewer.

`satelliteScenarioViewer(scenario,Name=Value)` sets properties on page 3-104 using one or more optional name-value arguments. For example, `satelliteScenarioViewer(scenario,Position = [1 1 500 500])` sets the position to `[1 1 500 500]`.

Properties

`Name` — Name of viewer window
`'Satellite Scenario Viewer'` (default) | string scalar | character vector

Name of the viewer window, specified as a comma-separated pair consisting of `'Name'` and either a string scalar or a character vector.

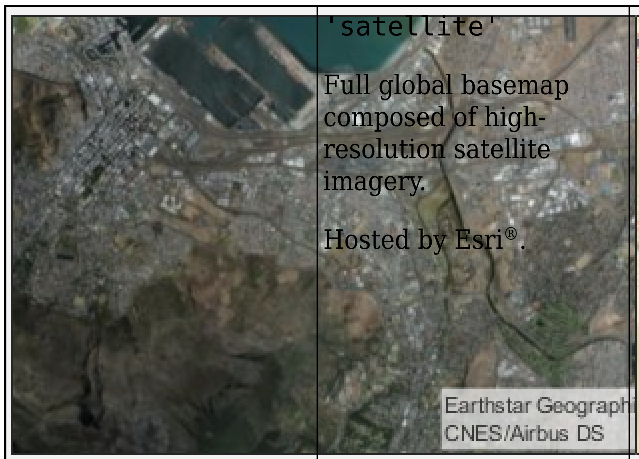
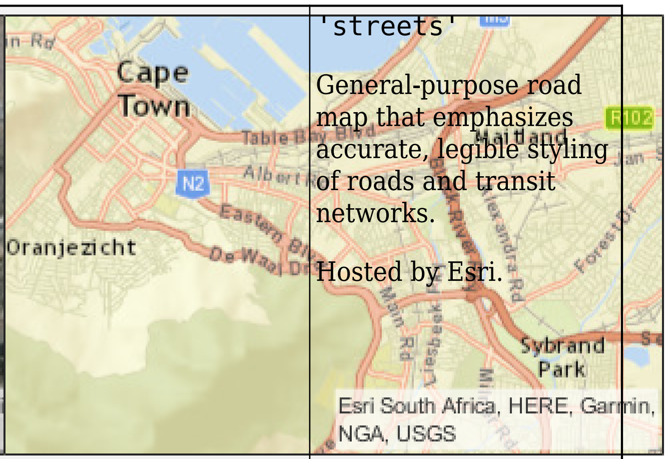
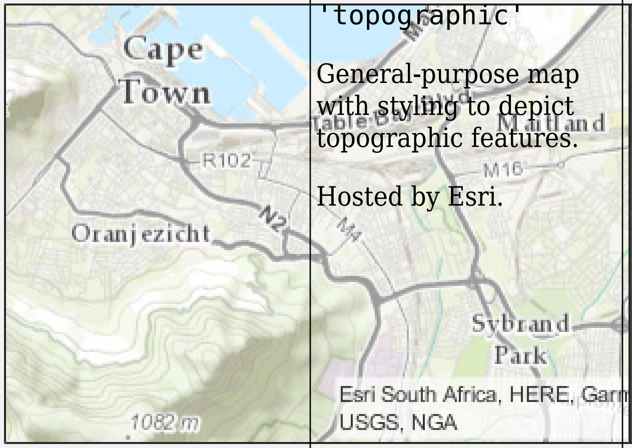
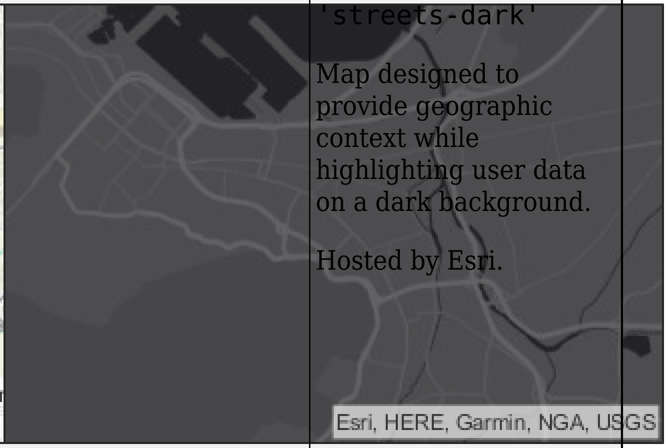
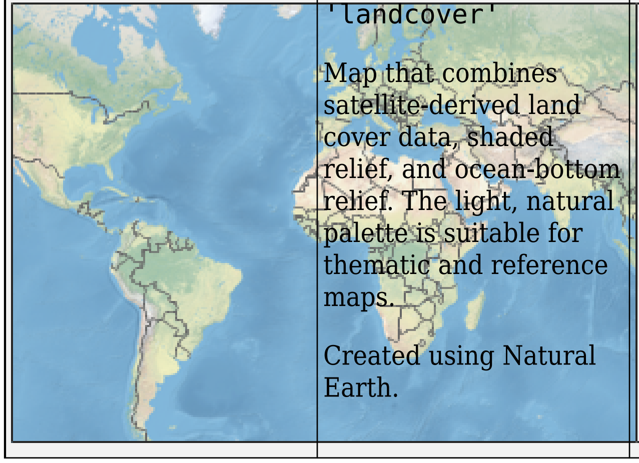
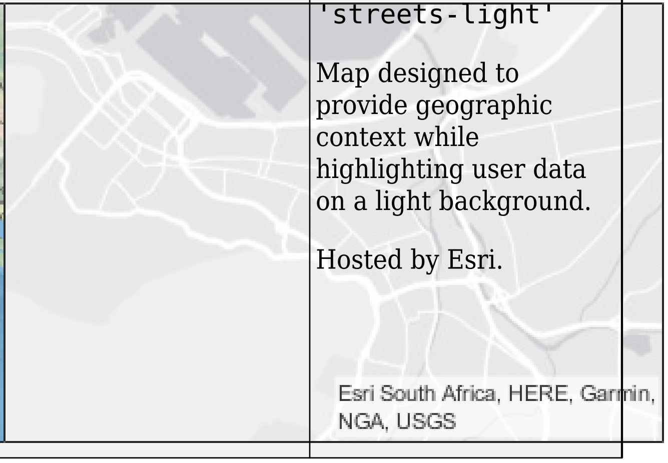
Data Types: `char` | `string`

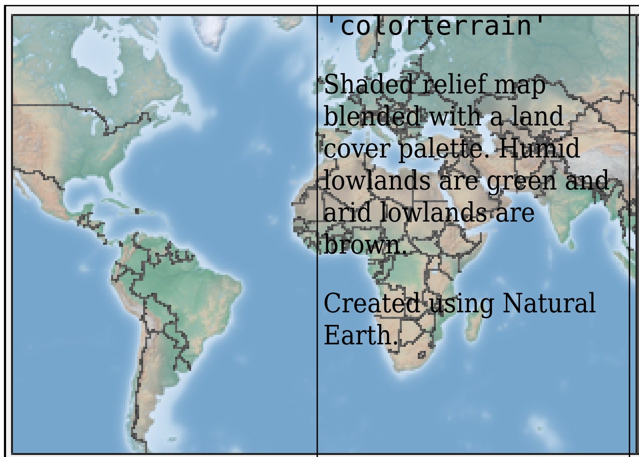
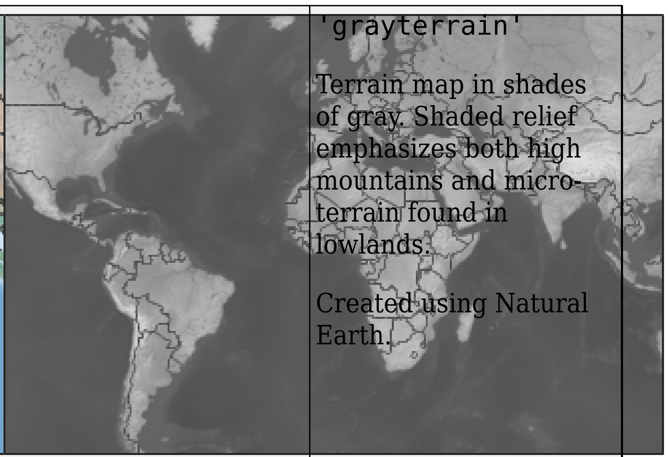
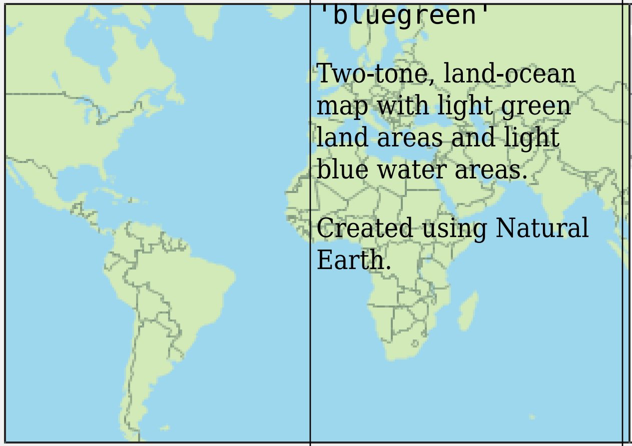
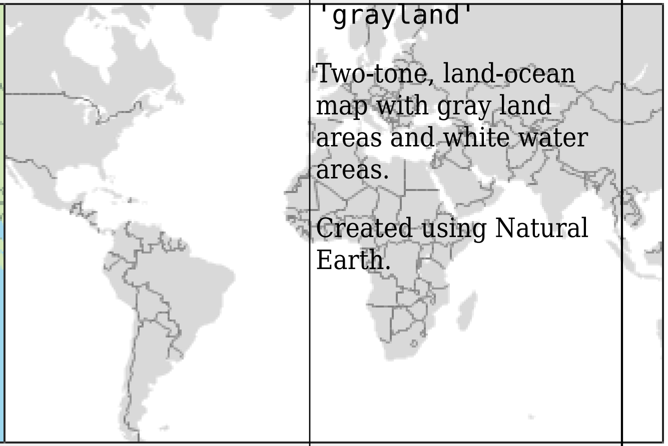
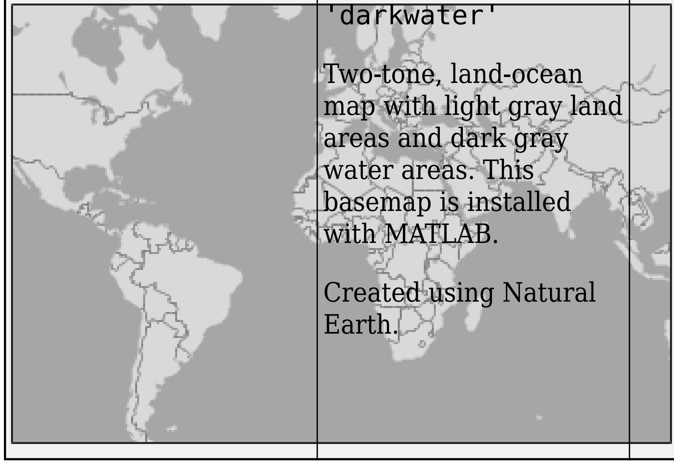
`Position` — Viewer window position
center of the screen (default) | row vector of four elements

Size and location of the satellite scenario window in pixels, specified as a row vector of four elements. The elements of the vector are `[left bottom width height]`. In the default case, width and height are 800 and 600 pixels, respectively.

Basemap — Map on which scenario is visualized
`'satellite'` (default) | `'topographic'` | `'streets'` | `'streets-light'` | `'streets-dark'` | `'darkwater'` | `'grayland'` | `'bluegreen'` | `'colorterrain'` | `'grayterrain'` | `'landcover'`

Map on which scenario is visualized, specified as a comma-separated pair consisting of `'Basemap'` and one of the values specified in this table:

	`'satellite'` Full global basemap composed of high-resolution satellite imagery. Hosted by Esri®.	 Cape Town Oranjezicht N2 R102 Maitland Sybrand Park Esri South Africa, HERE, Garmin, NGA, USGS	`'streets'` General-purpose road map that emphasizes accurate, legible styling of roads and transit networks. Hosted by Esri.
 Cape Town R102 Oranjezicht N2 M16 M4 Maitland Sybrand Park 1082 m Esri South Africa, HERE, Garm USGS, NGA	`'topographic'` General-purpose map with styling to depict topographic features. Hosted by Esri.		`'streets-dark'` Map designed to provide geographic context while highlighting user data on a dark background. Hosted by Esri.
	`'landcover'` Map that combines satellite-derived land cover data, shaded relief, and ocean-bottom relief. The light, natural palette is suitable for thematic and reference maps. Created using Natural Earth.		`'streets-light'` Map designed to provide geographic context while highlighting user data on a light background. Hosted by Esri.

	`'colorterrain'` Shaded relief map blended with a land cover palette. Humid lowlands are green and arid lowlands are brown. Created using Natural Earth.		`'grayterrain'` Terrain map in shades of gray. Shaded relief emphasizes both high mountains and micro-terrain found in lowlands. Created using Natural Earth.
	`'bluegreen'` Two-tone, land-ocean map with light green land areas and light blue water areas. Created using Natural Earth.		`'grayland'` Two-tone, land-ocean map with gray land areas and white water areas. Created using Natural Earth.
	`'darkwater'` Two-tone, land-ocean map with light gray land areas and dark gray water areas. This basemap is installed with MATLAB. Created using Natural Earth.		

All basemaps except `'darkwater'` require Internet access. The `'darkwater'` basemap is included with MATLAB and Satellite Communications Toolbox.

If you do not have consistent access to the Internet, you can download the basemaps created using Natural Earth onto your local system by using the Add-On Explorer. The basemaps hosted by Esri are not available for download.

Alignment of boundaries and region labels are a presentation of the feature provided by the data vendors and do not imply endorsement by The MathWorks®.

Data Types: `char` | `string`

`PlaybackSpeedMultiplier` — Speed of animation
`50` (default) | positive scalar

Speed of the animation for the input `scenario` used by the `play` function, specified as a comma-separated pair consisting of `'PlaybackSpeedMultiplier'` and a positive scalar.

`CameraReferenceFrame` — Reference frame of camera
`'ECEF'` (default) | `'Inertial'`

Reference frame of the camera, specified as a comma-separated pair consisting of `'CameraReferenceFrame'` and one of these values:

- `'ECEF'` — Earth-Centered Earth-Fixed camera.
- `'Inertial'` — Inertially fixed camera.

When you specify `'Inertial'`, the globe rotates with respect to the camera. When you specify `'ECEF'`, the camera rotates with the globe.

Dependencies

To enable this name-value argument, set to `Dimension` to `'3-D'`.

`CurrentTime` — Current simulation time
`StartTime` of `satelliteScenario` (default) | `datetime` array

Current simulation time of the viewer, specified as a `datetime` array. This value changes over time when the animation is playing.

Dependencies

To enable this name-value argument, set `AutoSimulate` to `true`.

Data Types: `datetime`

`Dimension` — Dimension of viewer
`'3D'` (default) | `'2D'`

Dimension of the viewer, specified as a comma-separated pair consisting of `'Dimension'` and either `'3D'` or `'2D'`.

`ShowDetails` — Flag to show graphical details
`true` or `1` (default) | `false` or `0`

Flag to show the graphical details for Satellite Scenario Viewer, specified as one of these numeric or logical values1.

- `1` (`true`) — Show all graphical details of satellites and ground stations except those explicitly hidden.
- `0` (`false`) — Hide all graphical details of satellites and the ground stations, including orbits, fields of view, labels, and the ground track. Even when `ShowDetails` is `false`, clicking or pausing on satellites and ground stations reveals hidden graphical details or labels, respectively.

Data Types: `logical`

Object Functions

campos	Set or get position of camera for Satellite Scenario Viewer
camheight	Set or get height of camera for Satellite Scenario Viewer
camheading	Set or get heading angle of camera for Satellite Scenario Viewer
camroll	Set or get roll angle of camera for Satellite Scenario Viewer
campitch	Set or get pitch angle of camera for Satellite Scenario Viewer
camtarget	Set camera target for Satellite Scenario Viewer
play	Play satellite scenario simulation results on viewer
showAll	Show all graphics in viewer
hideAll	Hide all graphics in satellite scenario viewer

Examples

Create and Visualize Satellite Scenario

Create a satellite scenario object.

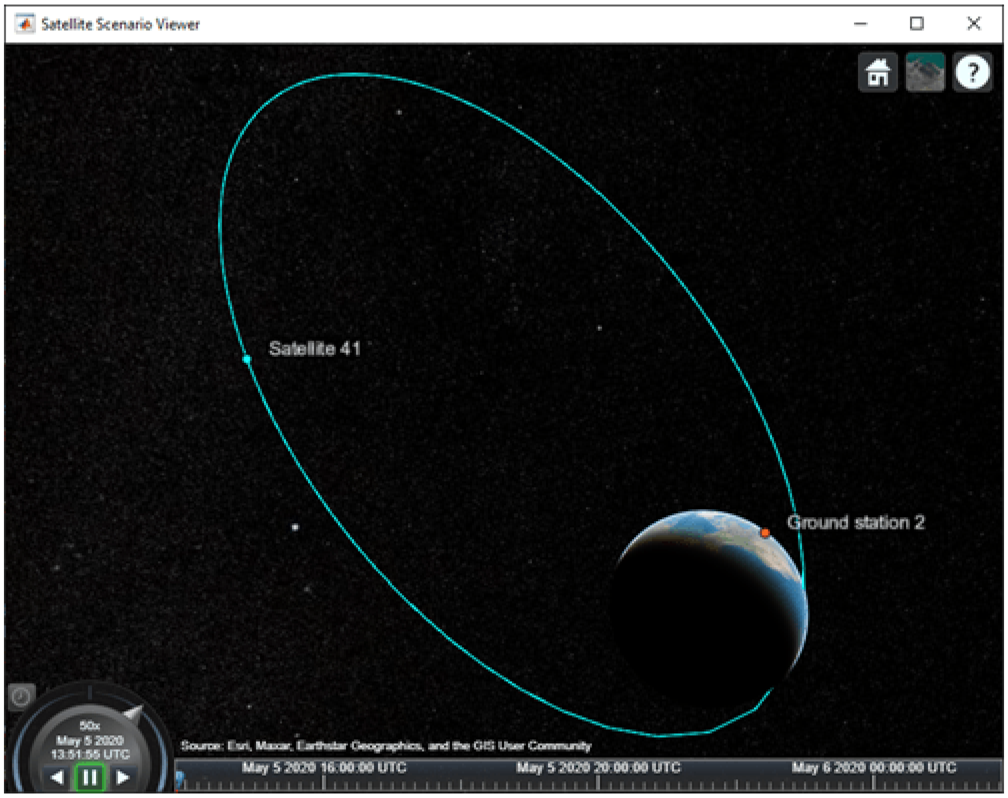
```
sc = satelliteScenario;
```

Add a satellite and ground station to the scenario. Additionally, add an access between the satellite and the ground station.

```
sat = satellite(sc,"eccentricOrbitSatellite.tle");
gs = groundStation(sc);
access(sat,gs);
```

Visualize the scenario at the start time defined in the TLE file by using the Satellite Scenario Viewer.

```
satelliteScenarioViewer(sc);
```



Version History

Introduced in R2021a

See Also

Objects
`satelliteScenario`

Functions
`show` | `hide` | `access` | `groundStation` | `satellite`

Topics
"Multi-Hop Satellite Communications Link Between Two Ground Stations"
"Comparison of Orbit Propagators"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

Satellite

Satellite in satellite scenario

Description

Satellite defines a satellite in satellite scenario object.

Creation

You can create Satellite objects using the `satellite` function of `satelliteScenario` object.

Properties

`Name` — Satellite name
string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `satellite` function. After you call `satellite` function, this property is read-only.

Satellite name, specified as a comma-separated pair consisting of `'Name'` and a string scalar, string vector, character vector or a cell array of character vectors.

- If only one satellite is added, specify `Name` as a string scalar or a character vector.
- If multiple satellites are added, specify `Name` as a string scalar, character vector, string vector or a cell array of character vectors. All satellites added as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vector must equal the number of satellites being added. Each satellite is assigned the corresponding name from the vector or cell array.

The default value when satellite is added to the satellite scenario using

- Keplerian orbital elements, TLE file, timeseries, or timetable — "Satellite ID", where `ID` is assigned by the satellite scenario.
- SEM almanac file or RINEX GPS navigation data — "PRN:prnValue", where prnValue is an integer denoting the pseudorandom noise code of the satellite as specified in the SEM almanac file.
- RINEX Galileo navigation data — "GAL Sat IF: id", where "id" is the satellite ID of the Galileo satellite defined in the RINEX navigation data.

Data Types: `string`

`ID` — Satellite ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Satellite ID assigned by the simulator, specified as a positive scalar.

ConicalSensors — Conical sensors
row vector of conical sensors

You can set this property only when calling the `conicalSensor`. After you call the `conicalSensor` function, this property is read-only.

Conical sensors attached to the Satellite, specified as a row vector of conical sensors.

Gimbals — Gimbals
row vector of `Gimbal` objects

You can set this property only when calling `gimbal`. After you call `gimbal`, this property is read-only.

Gimbals attached to the Satellite, specified as the comma-separated pair consisting of `'Gimbals'` and a row vector of `Gimbal` objects.

Transmitters — Transmitters attached to Satellite
row vector of `Transmitter` objects

You can set this property only when calling `transmitter` function. After you call the `transmitter` function, this property is read-only.

Transmitters attached to the Satellite, specified as a row vector of `Transmitter` objects.

Receivers — Receivers attached to the satellite
row vector of `Receiver` objects

You can set this property only when calling the `receiver`. After you call the `receiver` function, this property is read-only.

Receivers attached to the satellite, specified as a row vector of `Receiver` objects.

Accesses — Access analysis objects
row vector of `Access` objects

You can set this property only when calling `Satellite`. After you call `Satellite`, this property is read-only.

Access analysis objects, specified as a row vector of `Access` objects.

GroundTrack — Ground track of the Satellite
row vector of `GroundTrack` objects

You can set this property only when calling `groundTrack`. After you call `groundTrack`, this property is read-only.

Ground track of the Satellite, specified as a row vector of `GroundTrack` objects.

CoordinateAxes — Coordinate axes triad graphic object
`CoordinateAxes` object (default)

You can set this property only when calling `coordinateAxes`. After you call `coordinateAxes`, this property is read-only.

Coordinate axes triad graphic object, specified as `CoordinateAxes` object.

Orbit — Orbit graphic

Orbit object

Orbit object parameters for a satellite, specified as an `orbit` object. Only these object properties are relevant for this function.

LineColor — Color of orbit

`[1,0,0]` (default) | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'`

Color of the orbit, specified as an RGB triplet, hexadecimal color code, a color name, or a short name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟦

RGB Triplet	Hexadecimal Color Code	Appearance
`[0.6350 0.0780 0.1840]`	`"#A2142F"`	🟥

Example: `'blue'`

Example: `[0 0 1]`

Example: `'#0000FF'`

`LineWidth` — Visual width of orbit
`1` (default) | scalar in the range (0, 10)

Visual width of orbit in pixels, specified as a scalar in the range (0, 10).

The line width cannot be thinner than the width of a pixel. If you set the line width to a value that is less than the width of a pixel on your system, the line displays as one pixel wide.

`VisibilityMode` — Visibility mode of orbit graphic
`'inherit'` (default) | `'manual'`

Visibility mode of orbit graphic, specified as one of these values:

- `'inherit'` — Visibility of the graphic matches that of the parent
- `'manual'` — Visibility of the graphic is not inherited and is independent of that of the parent

Data Types: `char` | `string`

`OrbitPropagator` — Name of orbit propagator
`"sgp4"` | `"sdp4"` | `"two-body-keplerian"` | `"ephemeris"` | `"gps"` | `"galileo"`

You can set this property on `satellite` object creation and then this property becomes read-only.

Name of the orbit propagator used for propagating the satellite position and velocity, specified as one of these options.

- If you specify the satellite using timetable, table, `timeseries`, or `tscollection`, the `OrbitPropagator` value is `"ephemeris"`.
- If you specify the satellite using a SEM almanac file or RINEX data containing a GPS navigation message, the `OrbitPropagator` value can take one of these options.
  - `"gps"` (default)
  - `"sgp4"`
  - `"sdp4"`
  - `"two-body-keplerian"`
- If you specify the satellite using the RINEX data containing a Galileo navigation message, the `OrbitPropagator` value can take one of these options.
  - `"galileo"` (default)
  - `"sgp4"`
  - `"sdp4"`
  - `"two-body-keplerian"`

- If you specify the satellite using a TLE file, the `OrbitPropagator` value can take one of these options.

  - `"two-body-keplerian"`
  - `"sgp4"`
  - `"sdp4"`

  If the orbital period is less than 225 minutes, the default `OrbitPropagator` value is `"sgp4"`. Otherwise, the default `OrbitPropagator` value is `"sdp4"`.
- If you specify the satellite using `Keplerian` elements, the `OrbitPropagator` value can take one of these options.

  - `"two-body-keplerian"`
  - `"sgp4"`
  - `"sdp4"`

If the RINEX data contains both valid GPS and Galileo navigation messages, you cannot specify `OrbitPropagator` as `"gps"` or `"galileo"` using a name-value argument. However, you can still specify it as `"two-body-keplerian"`, `"sgp4"`, or `"sdp4"`.

`MarkerColor` — Color of marker
`[1 0 0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Color of the marker, specified as a comma-separated pair consisting of `'MarkerColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

`MarkerSize` — Size of marker

`10` (default) | positive scalar less than 30

Size of the marker, specified as a comma-separated pair consisting of `'MarkerSize'` and a real positive scalar less than 30. The unit is in pixels.

`ShowLabel` — State of Satellite label visibility

`true` or `1` (default) | `false` or `0`

State of Satellite label visibility, specified as a comma-separated pair consisting of `'ShowLabel'` and numerical or logical value of `1` (`true`) or `0` (`false`).

Data Types: `logical`

`LabelFontColor` — Font color of Satellite label

`[1,0,0]` (default) | `RGB triplet` | `string scalar of color name` | `character vector of color name`

Font color of the Satellitelabel, specified as a comma-separated pair consisting of `'LabelFontColor'` and either an RGB triplet or a string or character vector of a color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 string scalar or character vector 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. Therefore, 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"`	⬜
`"none"`	Not applicable	Not applicable	Not applicable	No color

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"`	🟥

`LabelFontSize` — Font size of Satellite label
15 (default) | positive scalar in the range [6 30]

Font size of the Satellite label, specified as a comma-separated pair consisting of `'LabelFontSize'` and a positive scalar in the range [6 30].

Object Functions

access	Add access analysis objects to satellite scenario
aer	Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame
dopplershift	Calculate Doppler shift at target asset in satellite scenario
latency	Calculate propagation delay from one asset to another asset
conicalSensor	Add conical sensor to satellite scenario
gimbal	Add gimbal to satellite or ground station
groundTrack	Add ground track object to satellite in scenario
orbitalElements	Orbital elements of satellites in scenario
coordinateAxes	Visualize coordinate axes triad of satellite scenario assets
pointAt	Point satellite at target
receiver	Add receiver to satellite scenario
transmitter	Add transmitter to satellite scenario

states	Obtain position and velocity of satellite
show	Show object in satellite scenario viewer
hide	Hide satellite scenario entity from viewer

Examples

Visualize Line of Sight Between Two Satellites

Create a satellite scenario object.

```
startTime = datetime(2020,5,5,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                      %seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a satellite from a TLE file to the scenario.

```
tleFile = "eccentricOrbitSatellite.tle";
sat1 = satellite(sc,tleFile,"Name","Sat1")
```

```
sat1 = 
  Satellite with properties:

               Name:  Sat1
                 ID:  1
     ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
            Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
       Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
          Receivers:  [1x0 satcom.satellitescenario.Receiver]
           Accesses:  [1x0 matlabshared.satellitescenario.Access]
        GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
              Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator:  sdp4
        MarkerColor:  [0.059 1 1]
         MarkerSize:  6
          ShowLabel:  true
     LabelFontColor:  [1 1 1]
      LabelFontSize:  15
```

Add a satellite from Keplerian elements to the scenario and specify its orbit propagator to be "two-body-keplerian".

```
semiMajorAxis = 6878137;                                                                    %m
eccentricity = 0;
inclination = 20;                                                                           %deg
rightAscensionOfAscendingNode = 0;                                                          %deg
argumentOfPeriapsis = 0;                                                                    %deg
trueAnomaly = 0;                                                                            %deg
sat2 = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode,...
    argumentOfPeriapsis,trueAnomaly,"OrbitPropagator","two-body-keplerian","Name","Sat2")
```

```
sat2 = 
  Satellite with properties:

               Name:  Sat2
```

```
                ID:  2
    ConicalSensors:  [1x0 matlabshared.satellitescenario.ConicalSensor]
           Gimbals:  [1x0 matlabshared.satellitescenario.Gimbal]
      Transmitters:  [1x0 satcom.satellitescenario.Transmitter]
         Receivers:  [1x0 satcom.satellitescenario.Receiver]
          Accesses:  [1x0 matlabshared.satellitescenario.Access]
       GroundTrack:  [1x1 matlabshared.satellitescenario.GroundTrack]
             Orbit:  [1x1 matlabshared.satellitescenario.Orbit]
   OrbitPropagator:  two-body-keplerian
       MarkerColor:  [0.059 1 1]
        MarkerSize:  6
         ShowLabel:  true
    LabelFontColor:  [1 1 1]
     LabelFontSize:  15
```

Add access analysis between the two satellites.

```
ac = access(sat1,sat2);
```

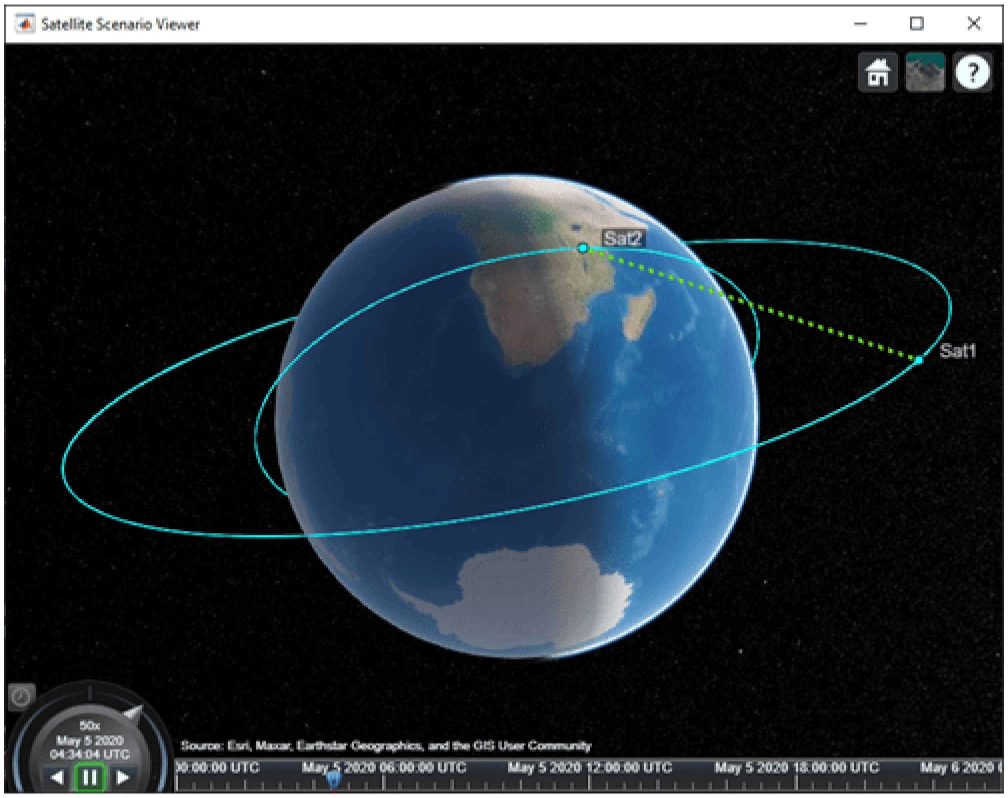
Determine the times when there is line of sight between the two satellites.

```
accessIntervals(ac)
```

```
ans=15×8 table
    Source    Target    IntervalNumber         StartTime                EndTime           Duratio
    ______    ______    ______________    ____________________    ____________________    _______

    "Sat1"    "Sat2"           1          05-May-2020 00:09:00    05-May-2020 01:08:00     3540
    "Sat1"    "Sat2"           2          05-May-2020 01:50:00    05-May-2020 02:47:00     3420
    "Sat1"    "Sat2"           3          05-May-2020 03:45:00    05-May-2020 04:05:00     1200
    "Sat1"    "Sat2"           4          05-May-2020 04:32:00    05-May-2020 05:26:00     3240
    "Sat1"    "Sat2"           5          05-May-2020 06:13:00    05-May-2020 07:10:00     3420
    "Sat1"    "Sat2"           6          05-May-2020 07:52:00    05-May-2020 08:50:00     3480
    "Sat1"    "Sat2"           7          05-May-2020 09:30:00    05-May-2020 10:29:00     3540
    "Sat1"    "Sat2"           8          05-May-2020 11:09:00    05-May-2020 12:07:00     3480
    "Sat1"    "Sat2"           9          05-May-2020 12:48:00    05-May-2020 13:46:00     3480
    "Sat1"    "Sat2"          10          05-May-2020 14:31:00    05-May-2020 15:27:00     3360
    "Sat1"    "Sat2"          11          05-May-2020 17:12:00    05-May-2020 18:08:00     3360
    "Sat1"    "Sat2"          12          05-May-2020 18:52:00    05-May-2020 19:49:00     3420
    "Sat1"    "Sat2"          13          05-May-2020 20:30:00    05-May-2020 21:29:00     3540
    "Sat1"    "Sat2"          14          05-May-2020 22:08:00    05-May-2020 23:07:00     3540
    "Sat1"    "Sat2"          15          05-May-2020 23:47:00    06-May-2020 00:00:00      780
```

Visualize the line of sight between the satellites.

```
play(sc);
```



Visualize GPS Constellation

Set up the satellite scenario.

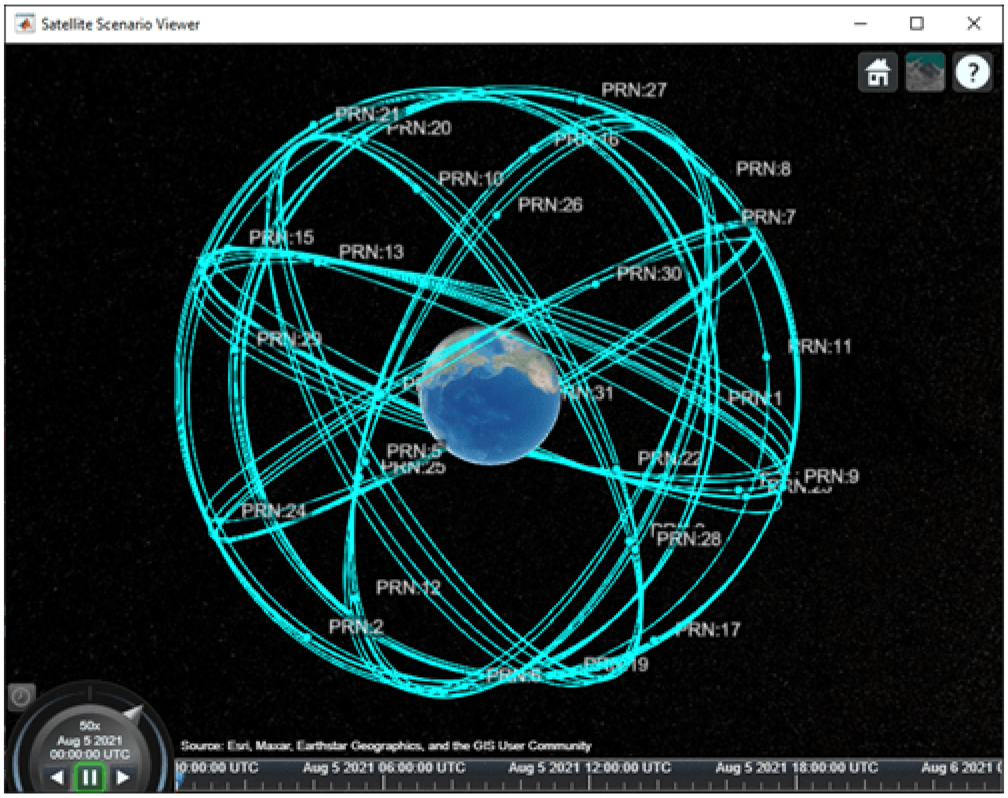
```
startTime = datetime(2021,8,5);
stopTime = startTime + days(1);
sampleTime = 60;                                        % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add satellites to the scenario from a SEM almanac file.

```
sat = satellite(sc,"gpsAlmanac.txt","OrbitPropagator","gps");
```

Visualize the GPS constellation.

```
v = satelliteScenarioViewer(sc);
```



Version History

Introduced in R2021a

References

[1] Hoots, Felix R., and Ronald L. Roehrich. *Models for propagation of NORAD element sets.* Aerospace Defense Command Peterson AFB CO Office of Astrodynamics, 1980.

See Also

Objects
`satelliteScenario` | `groundStation` | `access` | `satelliteScenarioViewer`

Functions
`show` | `play` | `hide`

Topics
"Comparison of Orbit Propagators"
"Model, Visualize, and Analyze Satellite Scenario"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

Transmitter

Transmitter object belonging to satellite scenario

Description

Transmitter defines a transmitter object belonging to a satellite scenario.

Creation

You can create Transmitter objects using the `transmitter` function of `satellite`, `groundStation`, or `gimbal`.

Properties

`Name` — Transmitter name
`"Transmitter` *`idx`*`"` (default) | string scalar | string vector | character vector | cell array of character vectors

You can set this property only when calling the `transmitter` function. After you call the `transmitter` function, this property is read-only.

Transmitter name, specified as a name-value argument consisting of `'Name'` and a string scalar, string vector, character vector, or a cell array of character vectors.

- If you are adding only one transmitter, specify `Name` as a string scalar or a character vector.
- If you are adding multiple transmitters, specify `Name` as a string scalar, character vector, string vector, or a cell array of character vectors. All transmitters that you add as a string scalar or a character vector are assigned the same specified name. The number of elements in the string vector or cell array of character vectors must equal the number of transmitters that you are adding. Each transmitter is assigned the corresponding name from the vector or cell array.

In the default value, *idx* is the ID assigned by the satellite scenario.

Data Types: `char` | `string`

`ID` — Transmitter ID assigned by simulator
real positive scalar

This property is set internally by the simulator and is read-only.

Transmitter ID assigned by the simulator, specified as a positive scalar.

`MountingLocation` — Mounting location with respect to parent
`[0; 0; 0]` (default) | three-element vector | matrix

Mounting location with respect to the parent object in meters, specified as a three-element vector or a matrix. The position vector is specified in the body frame of the input `parent`.

- If you are adding one transmitter, `MountingLocation` is a three-element vector. The elements specify the *x*, *y*, and *z* components of the Cartesian coordinates in the body frame of transmitter.
- If you are adding multiple transmitters, `MountingLocation` can be a three-element vector or a matrix. When specified as a vector, the same set of mounting locations are assigned to all specified transmitters. When specified as a matrix, `MountingLocation` must contain three rows and the same number of columns as the transmitters. The columns correspond to the mounting location of each specified transmitter and the rows correspond to the mounting location coordinates in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingLocation` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Data Types: `double`

`MountingAngles` — Mounting orientation with respect to parent object
`[0; 0; 0]` (default) | three-element row vector of positive numbers | matrix

Mounting orientation with respect to parent object in degrees, specified as a three-element row vector of positive numbers. The elements of the vector correspond to yaw, pitch, and roll, in that order. Yaw, pitch, and roll are positive rotations about the *z*-axis, intermediate *y*-axis, and intermediate *x*-axis of the parent.

- If you are adding one transmitter, the `MountingAngles` property is a three-element vector.
- If you are adding multiple transmitters the `MountingAngles` property can be a three-element vector or a matrix. When specified as a vector, the same set of mounting angles are assigned to all specified transmitters. When specified as a matrix, `MountingAngles` must contain three rows and the same number of columns as the transmitters. The columns correspond to the mounting angles of each specified transmitter and the rows correspond to the yaw, pitch, and roll angles in the parent body frame.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `MountingAngles` property only when `SimulationStatus` is `NotStarted`. You can use the `restart` function to reset `SimulationStatus` to `NotStarted`, but doing so erases the simulation data.

Example: `[0; 30; 60]`

Data Types: `double`

`Antenna` — Antenna object associated with transmitter
scalar | vector

`Antenna` object associated with the transmitter, specified as either a scalar or a vector. This object can be the default `gaussianAntenna` object, or one from the Antenna Toolbox or Phased Array System Toolbox. The default Gaussian antenna has a dish diameter of 1 meter and an aperture efficiency of 0.65.

Antenna can be specified in transmitter as a name-value argument consisting of `'Antenna'` and a scalar, antenna or phased array objects.

- If you are adding only one transmitter, `Antenna` must be a scalar.
- If you are adding multiple transmitters, `Antenna` is a vector. The same antenna is assigned to all transmitters.

SystemLoss — Total system loss in the transmitter
5 (default) | scalar | vector

Total system loss in the transmitter in dB, specified as a scalar or a vector.

System loss can be specified in transmitter as a name-value argument consisting of `'SystemLoss'` and a scalar, or vector.

- If you are adding only one transmitter, specify `SystemLoss` as a scalar.
- If you are adding multiple transmitters are added, specify `SystemLoss` as a scalar or a vector. When `SystemLoss` is a scalar, the same `SystemLoss` is assigned to all transmitters. When `SystemLoss` is a vector, its length must equal the number of transmitter and each element of `SystemLoss` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `SystemLoss` value while `SimulationStatus` is `NotStarted` or `InProgress`.

Frequency — Transmitter frequency
14e9 (default) | scalar | vector

Transmitter frequency in Hz, specified as a name-value argument consisting of 'Frequency' and a scalar or a vector.

- If you are adding only one transmitter, the `Frequency` must be a scalar.
- If you are adding multiple transmitters are added, the frequency value can be a scalar or a vector. All transmitters added as a scalar are assigned the same specified `Frequency`. The length of the vector must equal the number of transmitters added and each element of `Frequency` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is false, you can modify the `Frequency` value while `SimulationStatus` is `NotStarted` or `InProgress`.

BitRate — Bit rate of transmitter
10 (default) | scalar | vector

Bit rate of the transmitter in Mbps, specified as a name-value pair consisting of 'BitRate' and a scalar or a vector.

- If you are adding only one transmitter, the bit rate value must be a scalar.
- If you are adding multiple transmitters, the bit rate value can be a scalar or a vector. All transmitters added as a scalar are assigned the same specified `BitRate`. The length of the vector must equal the number of transmitters added and each element of `BitRate` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is `false`, you can modify the `BitRate` value while `SimulationStatus` is `NotStarted` or `InProgress`.

Power — Power of high power amplifier
12 (default) | scalar | vector

Power of the high power amplifier in dbW, specified as a name-value pair consisting of 'Power' and a scalar or a vector.

- If you are adding only one transmitter, the power value must be a scalar.

- If you are adding multiple transmitters, the power value can be a scalar or a vector. All transmitters added as a scalar are assigned the same specified `Power`. The length of the vector must equal the number of transmitters added and each element of `Power` is assigned to the corresponding transmitter in the parent.

When the `AutoSimulate` property of the satellite scenario is false, you can modify the `Power` value while `SimulationStatus` is `NotStarted` or `InProgress`.

`CoordinateAxes` — Coordinate axes triad graphic object
`CoordinateAxes` object (default)

You can set this property only when calling `coordinateAxes`. After you call `coordinateAxes`, this property is read-only.

Coordinate axes triad graphic object, specified as `CoordinateAxes` object.

`Links` — Link analysis objects
row vector of `Link` objects

This property is read-only.

Link analysis objects, specified as a row vector `Link` objects.

Object Functions

aer	Calculate azimuth angle, elevation angle, and range of another satellite or ground station in NED frame
gaussianAntenna	Add Gaussian antennas
link	Add link analysis objects to transmitter
pattern	Plot 3-D radiation pattern of antenna
pointAt	Point transmitter or receiver at target
pointAt	Point satellite at target
coordinateAxes	Visualize coordinate axes triad of satellite scenario assets
dopplershift	Calculate Doppler shift at target asset in satellite scenario

Examples

Determine Times of Availability for Satellite Link Between Two Ground Stations

Create a satellite scenario object.

```
startTime = datetime(2020,11,25,0,0,0);
stopTime = startTime + days(1);
sampleTime = 60;                                     % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc = 
  satelliteScenario with properties:

         StartTime: 25-Nov-2020
          StopTime: 26-Nov-2020
        SampleTime: 60
      AutoSimulate: 1
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
```

```
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
          AutoShow: 1
```

Add a satellite to the scenario.

```
semiMajorAxis = 10000000;                                                       % mete
eccentricity = 0;
inclination = 60;                                                               % degr
rightAscensionOfAscendingNode = 0;                                              % degr
argumentOfPeriapsis = 0;                                                        % degr
trueAnomaly = 0;                                                                % degr
sat = satellite(sc,semiMajorAxis,eccentricity,inclination,rightAscensionOfAscendingNode, ...
        argumentOfPeriapsis,trueAnomaly,Name="Satellite");
```

Add gimbals to the satellite. These gimbals enable the satellite receiver antenna to steer to the first ground station, and its transmitter antenna to steer to the second ground station.

```
gimbalrxSat = gimbal(sat);
gimbaltxSat = gimbal(sat);
```

Add a receiver to the first gimbal of the satellite.

```
gainToNoiseTemperatureRatio = 5;                                              % dB/K
systemLoss = 3;                                                               % dB
rxSat = receiver(gimbalrxSat,Name="Satellite Receiver",GainToNoiseTemperatureRatio= ...
    gainToNoiseTemperatureRatio,SystemLoss=systemLoss)

rxSat = 
  Receiver with properties:

                           Name:  Satellite Receiver
                             ID:  4
               MountingLocation:  [0; 0; 0] meters
                 MountingAngles:  [0; 0; 0] degrees
                        Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
                     SystemLoss:  3 decibels
                PreReceiverLoss:  3 decibels
    GainToNoiseTemperatureRatio:  5 decibels/Kelvin
                    RequiredEbNo:  10 decibels
```

Add a transmitter to the second gimbal of the satellite.

```
frequency = 27e9;                                                            % Hz
power = 20;                                                                  % dBW
bitRate = 20;                                                                % Mbps
systemLoss = 3;                                                              % dB
txSat = transmitter(gimbaltxSat,Name="Satellite Transmitter",Frequency=frequency, ...
    power=power,BitRate=bitRate,SystemLoss=systemLoss)

txSat = 
  Transmitter with properties:

                Name:  Satellite Transmitter
                  ID:  5
    MountingLocation:  [0; 0; 0] meters
      MountingAngles:  [0; 0; 0] degrees
```

```
             Antenna:  [1x1 satcom.satellitescenario.GaussianAntenna]
          SystemLoss:  3 decibels
           Frequency:  2.7e+10 Hertz
             BitRate:  20 Mbps
               Power:  20 decibel-watts
               Links:  [1x0 satcom.satellitescenario.Link]
```

Specify the antenna specifications of the repeater.

```
dishDiameter = 0.5;                                                                    % meters
apertureEfficiency = 0.5;
gaussianAntenna(txSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
gaussianAntenna(rxSat,DishDiameter=dishDiameter,ApertureEfficiency=apertureEfficiency);
```

Add two ground stations to the scenario.

```
gs1 = groundStation(sc,Name="Ground Station 1");
latitude = 52.2294963;                                              % degrees
longitude = 0.1487094;                                              % degrees
gs2 = groundStation(sc,latitude,longitude,Name="Ground Station 2");
```

Point gimbals of the satellite towards the two ground stations for the simulation duration.

```
pointAt(gimbaltxSat,gs2);
pointAt(gimbalrxSat,gs1);
```

Add gimbals to the ground stations. These gimbals enable the ground station antennas to steer towards the satellite.

```
gimbalgs1 = gimbal(gs1);
gimbalgs2 = gimbal(gs2);
```

Add a transmitter to ground station `gs1`.

```
frequency = 30e9;                                                                          % Hz
power = 40;                                                                                % dBW
bitRate = 20;                                                                              % Mbp
txGs1 = transmitter(gimbalgs1,Name="Ground Station 1 Transmitter",Frequency=frequency, ...
        Power=power,BitRate=bitRate);
```

Add a receiver to ground station `gs2`.

```
requiredEbNo = 14;                                                                     % dB
rxGs2 = receiver(gimbalgs2,Name="Ground Station 2 Receiver",RequiredEbNo=requiredEbNo);
```

Define the antenna specifications of the ground stations.

```
dishDiameter = 5;                                % meters
gaussianAntenna(txGs1,DishDiameter=dishDiameter);
gaussianAntenna(rxGs2,DishDiameter=dishDiameter);
```

Point gimbals of the ground stations towards the satellite for the simulation duration.

```
pointAt(gimbalgs1,sat);
pointAt(gimbalgs2,sat);
```

Add link analysis to transmitter `txGs1`.

```
lnk = link(txGs1,rxSat,txSat,rxGs2)
```

```
lnk = 
  Link with properties:

    Sequence:  [10 4 5 11]
    LineWidth:  2
    LineColor:  [0.3922 0.8314 0.0745]
```

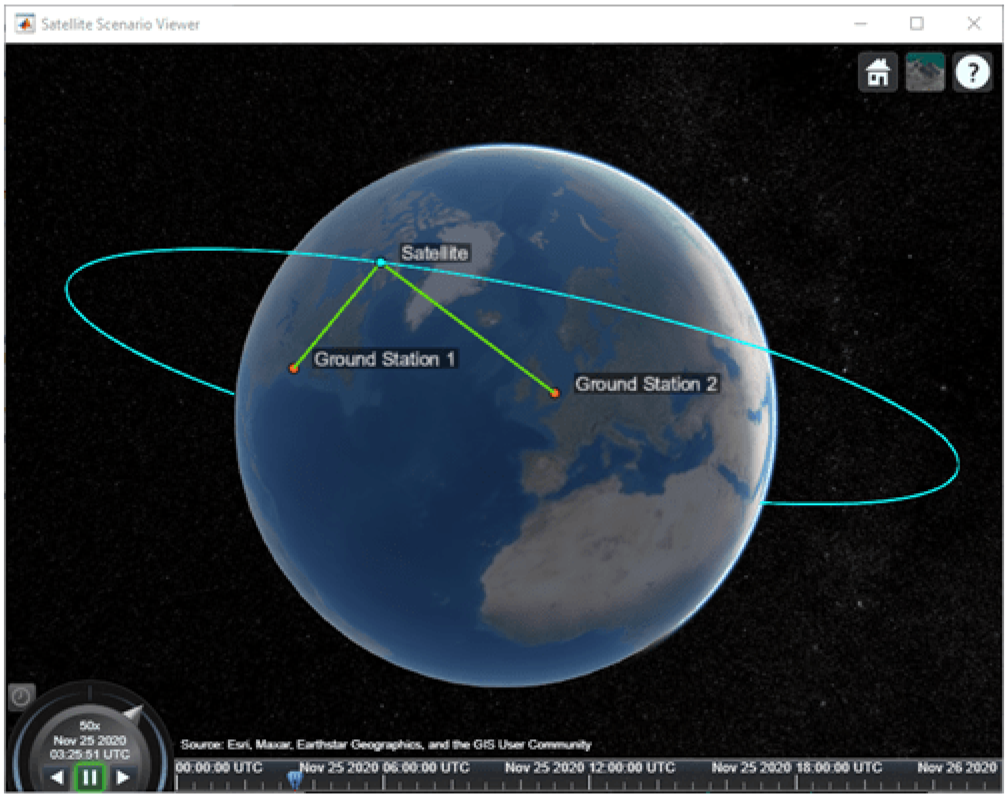
Determine the times when ground station `gs1` can send data to ground station `gs2` via the satellite.

```
linkIntervals(lnk)
```

```
ans=4×8 table
                Source                            Target                 IntervalNumber         Star
    _______________________________    ____________________________    ______________    __________

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           1           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           2           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           3           25-Nov-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"           4           25-Nov-202
```

Visualize the link by using the Satellite Scenario Viewer.

```
play(sc);
```



Version History

Introduced in R2021a

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer` | `Receiver`

Functions

`play` | `show` | `hide` | `groundStation` | `access` | `receiver` | `transmitter` | `pointAt`

Topics

"Model, Visualize, and Analyze Satellite Scenario"
"Multi-Hop Satellite Communications Link Between Two Ground Stations"
"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

skyplot

Plot satellite azimuth and elevation data

Syntax

```
skyplot(azdata,eldata)
skyplot(azdata,eldata,labeldata)
skyplot(status)
skyplot( ___ ,Name,Value)

skyplot(parent, ___ )
h = skyplot( ___ )
```

Description

`skyplot(azdata,eldata)` creates a sky plot using the azimuth and elevation data specified as matrices in degrees. Azimuth angles are measured in degrees, clockwise-positive from the North direction. Elevation angles are measured from the horizon line with 90 degrees being directly up. For details about the sky plot figure elements, see “Main Sky Plot Elements” on page 3-136.

`skyplot(azdata,eldata,labeldata)` specifies data labels as a string array with elements corresponding to each data point in the `azdata` and `eldata` inputs.

`skyplot(status)` specifies the azimuth and elevation data in a structure with fields `SatelliteAzimuth` and `SatelliteElevation`.

`skyplot( ___ ,Name,Value)` specifies options using one or more name-value arguments in addition to the input arguments in previous syntaxes. The name-value arguments are properties of the `SkyPlotChart` object. For a list of properties, see SkyPlotChart Properties.

`skyplot(parent, ___ )` creates the sky plot in the figure, panel, or tab specified by `parent`.

`h = skyplot( ___ )` returns the sky plot as a `SkyPlotChart` object, `h`. Use `h` to modify the properties of the chart after creating it. For a list of properties, see SkyPlotChart Properties.

Examples

View Satellite Positions from GNSS Sensor

Create a GNSS sensor model as a `gnssSensor` (Navigation Toolbox) System object™.

```
gnss = gnssSensor;
```

Specify the position and velocity of the sensor. Simulate the sensor readings and get status from visible satellites. Store the azimuth and elevation angles as vectors.

```
pos = [0 0 0];
vel = [0 0 0];
[~, ~, status] = gnss(pos, vel);
```

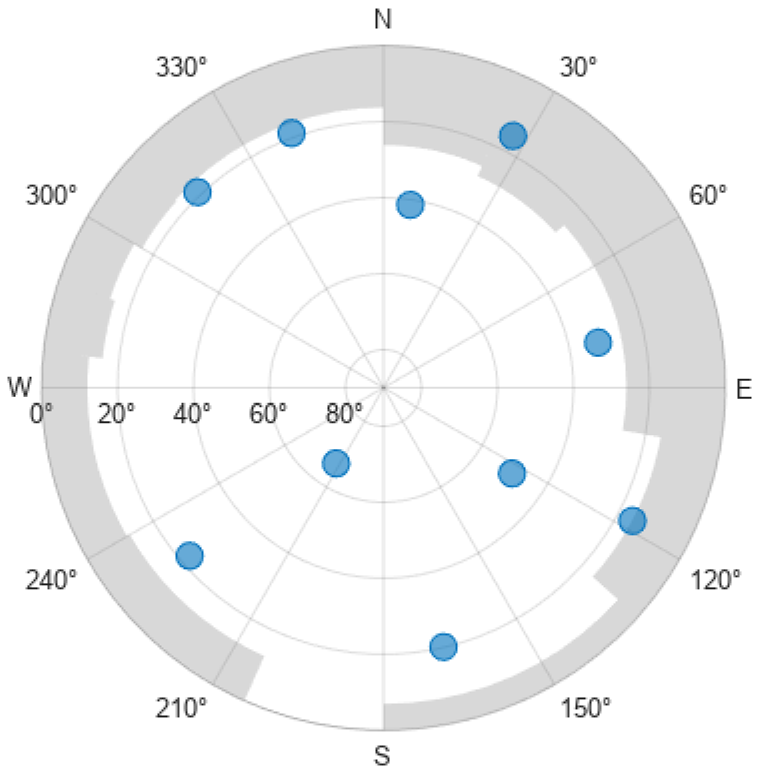
```
satAz = status.SatelliteAzimuth;
satEl = status.SatelliteElevation;
```

Create random local elevation masks, with a maximum elevation of 30 degrees, to act as the local environment.

```
rng(8)
terrainMaskElevations = 30*rand(1,12); % elevations (degrees)
terrainMaskEdges = [0 24 48 100 132 180 204 240 276 288 300 312 360]; % azimuth edges (degrees)
```

Plot the satellite positions with the elevation masks.

```
skyplot(satAz,satEl,MaskElevation=terrainMaskElevations,MaskAzimuthEdges=terrainMaskEdges);
```

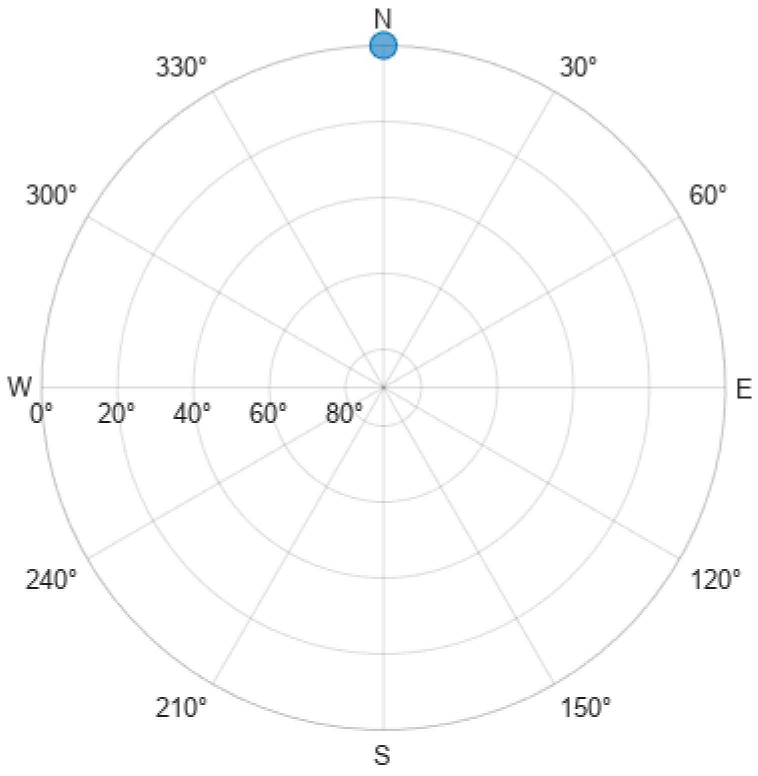


Plot Series of Satellite Positions Over Time

Animate the trajectory of satellite positions over time from a GNSS sensor.

Initialize the sky plot figure. Specify the relevant time-stepping information.

```
skyplotHandle = skyplot(0,0);
```



```
numHours = 12;
dt = 100;
numSeconds = numHours * 60 * 60;
numSimSteps = numSeconds/dt;
```

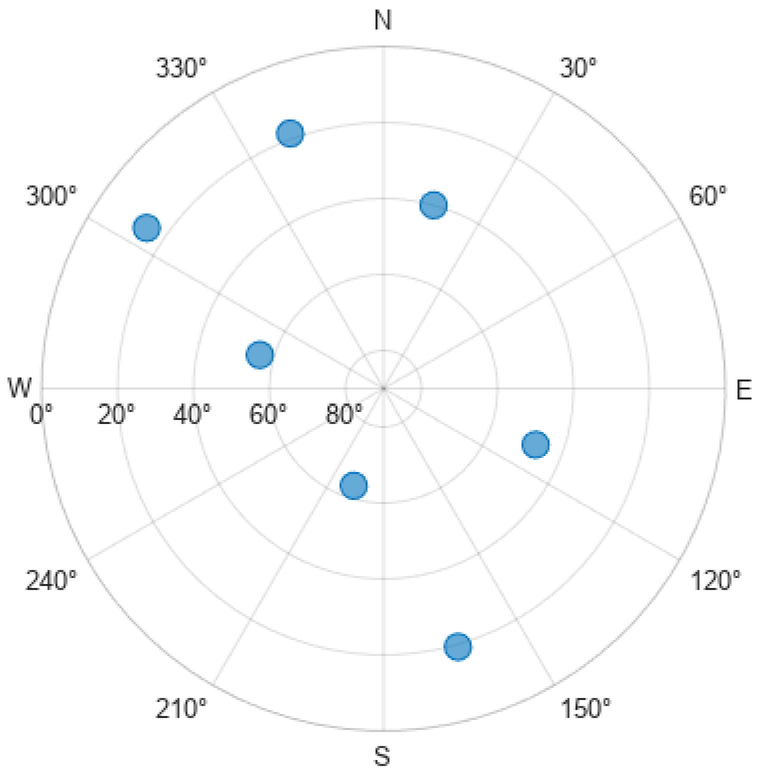
Create a GNSS sensor model as a `gnssSensor` (Navigation Toolbox) System Object™.

```
gnss = gnssSensor('SampleRate', 1/dt);
```

Iterate through the time steps and do the following:

- Simulate the sensor readings. Specify the zero postion and velocity for the stationary sensor.
- Store the azimuth and elevation angles as vectors.
- Set the `AzimuthData` and `ElevationData` properties of the `SkyPlotChart` handle directly.

```
for i = 1:numSimSteps

    [~, ~, status] = gnss([0 0 0],[0 0 0]);

    satAz = status.SatelliteAzimuth;
    satEl = status.SatelliteElevation;

    set(skyplotHandle,'AzimuthData',satAz,'ElevationData',satEl);

    drawnow
end
```



View Satellite Positions For Different Groups

Load the azimuth and elevation data from a logfile generated by an Adafruit® GPS satellite sensor. The data provided in this example contains the azimuth and elevation of each satellite and the pseudorandom noise (PRN) codes. Store these values as vectors.

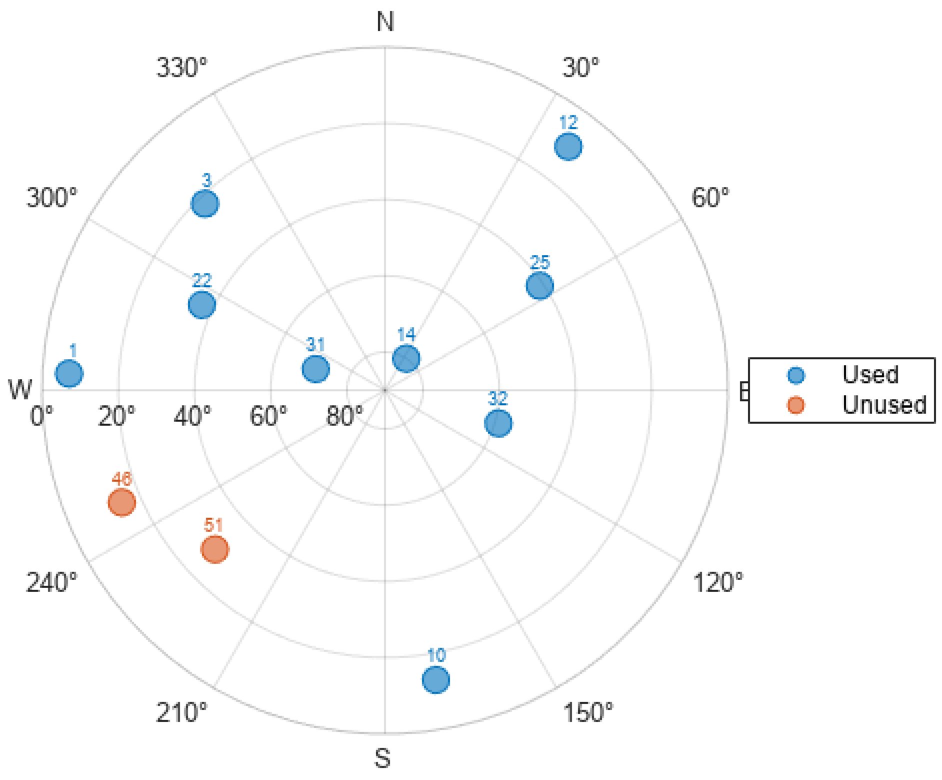
```
load('gpsHWInfo','hwInfo')
satAz = hwInfo.SatelliteAzimuths;
satEl = hwInfo.SatelliteElevations;
prn = hwInfo.SatellitePRNs;
```

Separate the satellites based on the PRN codes. To correlate each position with a group, create a `categorical` array. For this set of satellites, only the ones with PRNs less than 32 are used in the positioning solution.

```
isUnused = (prn > 32);
group = categorical(isUnused,[false true],["Used in Positioning Solution" "Unused"]);
```

Visualize the satellites and specify the categorical groups in the `GroupData` name-value argument. Specify the PRN as the label for each point. Show the legend.

```
skyplot(satAz,satEl,prn,GroupData=group)
legend('Used','Unused')
```



Visualize Satellite Trajectories in Skyplot

Specify the receiver position, RINEX navigation file, mask angle, time step size, and number of hours of data to sample from the RINEX file.

```
recPos = [42 -71 50];
navfile = "GODS00USA_R_20211750000_01D_GN.rnx";
maskAngle = 25;
dt = 60; % seconds
numHours = 4;
```

Read the navigation file, and get the GPS data of all satellites captured in the file.

```
data = rinexread(navfile);
[~,satIdx] = unique(data.GPS.SatelliteID);
navmsg = data.GPS(satIdx,:);
```

Set the starting time to the initial time of the navigation message. Then, create the time vector `t`.

```
startTime = navmsg.Time(1);
secondsPerHour = 3600;
timeElapsed = 0:dt:(secondsPerHour*numHours);
t = startTime + seconds(timeElapsed);
```

Initialize vectors for azimuth and elevation. Then, collect azimuth and elevation data at times `t` for all satellites.

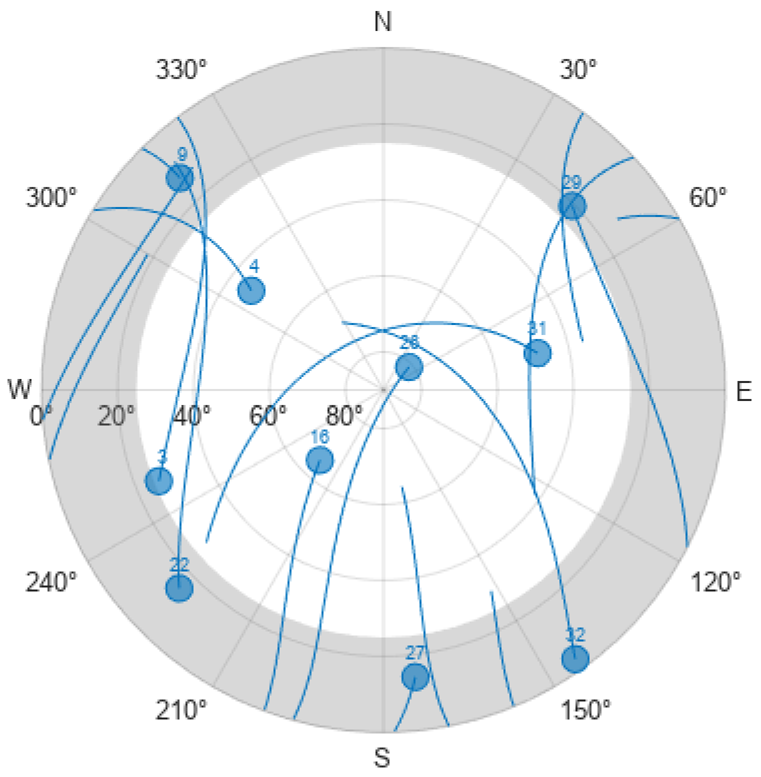
```
numSats = numel(navmsg.SatelliteID);
allAz = NaN(numel(t),numSats);
allEl = allAz;
for idx = 1:numel(t)
    [satPos,~,satID] = gnssconstellation(t(idx),RINEXData=navmsg);
    [az,el,vis] = lookangles(recPos,satPos,maskAngle);
    allAz(idx,:) = az;
    allEl(idx,:) = el;
end
```

Mark all satellites below the horizon with an elevation less than `0` as missing.

```
allEl(allEl < 0) = missing;
```

Display the satellite trajectories as an animation by creating a skyplot and updating the `AzimuthData` and `ElevationData` properties.

```
figure
sp = skyplot(allAz(1,:),allEl(1,:),satID,MaskElevation=maskAngle);
for idx = 1:size(allAz, 1)
    set(sp,AzimuthData=allAz(1:idx,:),ElevationData=allEl(1:idx,:));
    drawnow limitrate
end
```



Input Arguments

`azdata` — Azimuth angles for visible satellite positions
n-element vector of angles | *t*-by-*n* matrix of angles

Azimuth angles for visible satellite positions, specified as an *n*-element vector of angles or *t*-by-*n* matrix of angles. *n* is the number of visible satellite positions in the plot, and *t* is the number of time steps of the satellites. Azimuth angles are measured in degrees, clockwise-positive from the north direction.

Example: `[25 45 182 356]` specifies the azimuth angles for four satellites at one time step.

Data Types: `double`

`eldata` — Elevation angles for visible satellite positions
n-element vector of angles | *t*-by-*n* matrix of angles

Elevation angles for visible satellite positions, specified as an *n*-element vector of angles or *t*-by-*n* matrix of angles. *n* is the number of visible satellite positions in the plot, and *t* is the number of time steps of the satellites. Elevation angles are measured from the horizon line with 90 degrees being directly up.

Example: `[45 90 27 74]` specifies the elevation angles for four satellites at one time step.

Data Types: `double`

`labeldata` — Labels for visible satellite positions
n-element string array

Labels for visible satellite positions, specified as an *n*-element string array. *n* is the number of visible satellite positions in the plot.

Example: `["G1" "G11" "G7" "G3"]`

Data Types: `string`

`status` — Satellite status
structure array

Satellite status, specified as a structure array with fields `SatelliteAzimuth` and `SatelliteElevation`. Typically, this status structure comes from a `gnssSensor` object, which simulates satellite positions and velocities.

Example: `gnss = gnssSensor; [~,~,status] = gnss(position,velocity)`

Data Types: `struct`

`parent` — Parent container
`Figure` object | `Panel` object | `Tab` object | `TiledChartLayout` object | `GridLayout` object

Parent container, specified as a `Figure`, `Panel`, `Tab`, `TiledChartLayout`, or `GridLayout` object.

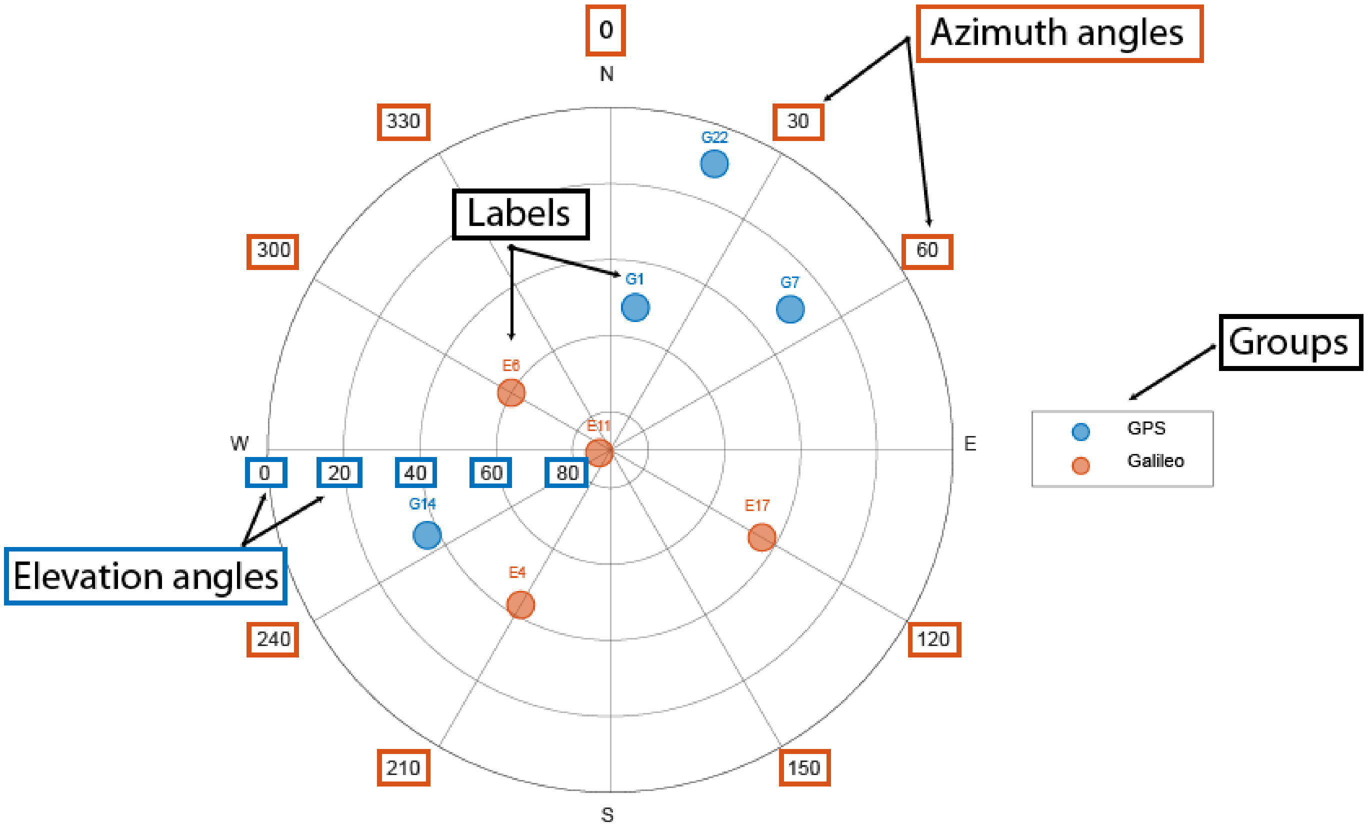
Output Arguments

`h` — Sky plot chart
`SkyPlotChart` object

Sky plot chart, returned as a `SkyPlotChart` object, which is a standalone visualization on page 3-136. Use `h` to set properties on the sky plot chart. For more information, see SkyPlotChart Properties (Navigation Toolbox).

More About

Main Sky Plot Elements



The main elements of the figure are:

- Azimuth axes — Specified by the `azdata` input argument, azimuth angle positions are measured clockwise-positive from the North direction.
- Elevation axes —Specified by the `eldata` input argument, elevation angle positions are measured from the horizon line with 90 degrees being directly up.
- Labels — Specified by the `labeldata` input argument as a string array with an element for each point in the `azdata` and `eldata` vectors.
- Groups — Specified by the `GroupData` property, a `categorical` array defines the group for each satellite position.

Standalone Visualization

A standalone visualization is a chart designed for a special purpose that works independently from other charts. Unlike other charts such as `plot` and `surf`, a standalone visualization has a preconfigured axes object built into it, and some customizations are not available. A standalone visualization also has these characteristics:

- It cannot be combined with other graphics elements, such as lines, patches, or surfaces. Thus, the `hold` command is not supported.
- The `gca` function can return the chart object as the current axes.
- You can pass the chart object to many MATLAB functions that accept an axes object as an input argument. For example, you can pass the chart object to the `title` function.

Version History

Introduced in R2021a

R2022b: `skyplot` supports azimuth and elevation trajectories and elevation masks

- The `azdata` and `eldata` arguments now accept matrices, enabling you to represent trajectories by adding azimuth and elevation data for satellites at multiple time steps.
- Elevation angle masks are now supported using these new `SkyPlotChart` properties:
  - `MaskElevation`
  - `MaskAlpha`
  - `MaskColor`
  - `MaskAzimuthEdges`
  - `MaskAzimuthEdgesMode`

See Also

Functions
`polarscatter`

Properties
SkyPlotChart Properties (Navigation Toolbox)

Objects
`gnssSensor` | `nmeaParser`

SkyPlotChart Properties

Sky plot chart appearance and behavior

Description

The `SkyPlotChart` properties control the appearance of a sky plot chart generated using the `skyplot` function. To modify the chart appearance, use dot notation on the `SkyPlotChart` object:

```
h = skyplot;
h.AzimuthData = [45 120 295];
h.ElevationData = [10 45 60];
h.Labels = ["G1" "G4" "G11"];
```

Properties

Sky Plot Properties

`AzimuthData` — Azimuth angles for visible satellite positions
n-element vector of angles | *t*-by-*n* matrix of angles

Azimuth angles for visible satellite positions, specified as an *n*-element vector of angles or *t*-by-*n* matrix of angles. *n* is the number of visible satellite positions in the plot, and *t* is the number of time steps of the satellites. Azimuth angles are measured in degrees, clockwise-positive from the north direction.

If you specify `AzimuthData` as a matrix, the last row indicates the current azimuth angles of the satellites.

Example: `[25 45 182 356]` specifies azimuth angles for four satellites at one time step

Data Types: `double`

`ElevationData` — Elevation angles for visible satellite positions
n-element vector of angles | *t*-by-*n* matrix of angles

Elevation angles for visible satellite positions, specified as an *n*-element vector of angles or *t*-by-*n* matrix of angles. *n* is the number of visible satellite positions in the plot, and *t* is the number of time steps of the satellites. Elevation angles are measured from the horizon line with 90 degrees being directly up.

If you specify `ElevationData` as a matrix, the last row indicates the current elevation angles of the satellites.

Example: `[45 90 27 74]` specifies elevation angles for four satellites at one time step

Data Types: `double`

`LabelData` — Labels for visible satellite positions
n-element string array

Labels for visible satellite positions, specified as an *n*-element string array. *n* is the number of visible satellite positions in the plot.

Example: `["G1" "G11" "G7" "G3"]`

Data Types: `string`

`GroupData` — Group for each satellite position
`categorical` array

Group for each satellite position, specified as a `categorical` array. Each group has a different color label defined by the ColorOrder property.

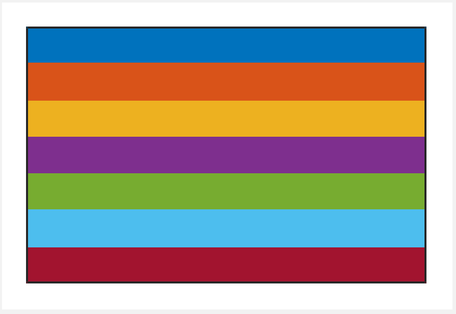
Example: `[GPS GPS Galileo Galileo]`

Data Types: `double`

`ColorOrder` — Color order
seven predefined colors (default) | three-column matrix of RGB triplets

Color order, specified as a three-column matrix of RGB triplets. This property defines the palette of colors MATLAB uses to create plot objects such as `Line`, `Scatter`, and `Bar` objects. Each row of the array is an RGB triplet. An RGB triplet is a three-element vector whose elements specify the intensities of the red, green, and blue components of a color. The intensities must be in the range [0, 1]. This table lists the default colors.

Colors	`ColorOrder` Matrix

	`[ 0 0.4470 0.7410`
	`0.8500 0.3250 0.0980`
	`0.9290 0.6940 0.1250`
	`0.4940 0.1840 0.5560`
	`0.4660 0.6740 0.1880`
	`0.3010 0.7450 0.9330`
	`0.6350 0.0780 0.1840]`

MATLAB assigns colors to objects according to their order of creation. For example, when plotting lines, the first line uses the first color, the second line uses the second color, and so on. If there are more lines than colors, then the cycle repeats.

You can also set the color order using the `colororder` function.

Label Properties

`LabelFontSize` — Font size of labels
scalar numeric value

Font size of labels, specified as a scalar numeric value. The default font depends on the specific operating system and locale.

Example: `h = skyplot(__,'LabelFontSize',12)`

Example: `h.LabelFontSize = 12`

`LabelFontSizeMode` — Selection mode for font size of labels
`'auto'` (default) | `'manual'`

Selection mode for the font size of labels, specified as one of these values:

- `'auto'` — Font size specified by MATLAB. If you resize the axes to be smaller than the default size, the font size can scale down to improve readability and layout.
- `'manual'` — Font size specified manually. MATLAB does not scale the font size as the axes size changes. To specify the font size, set the `LabelFontSize` property.

Mask Properties

`MaskElevation` — Mask elevation angle

`0` (default) | nonnegative scalar | *N*-element vector

Elevation angle of mask, specified as a nonnegative scalar or *N*-element vector of nonnegative values, in degrees. *N* is *m* + 1, where *m* is the number of elements in `MaskAzimuthEdges`..

Example: `h = skyplot(__,MaskElevation=25)`

Data Types: `double`

`MaskAlpha` — Mask transparency

`0.3` (default) | scalar in range [0, 1]

Mask transparency, specified as a scalar in the range [0, 1]. A transparency value of `1` is opaque, `0` is completely transparent, and values between `0` and `1` are semitransparent.

Example: `h = skyplot(__,MaskAlpha=0.1)`

Data Types: `double`

`MaskColor` — Mask color

`[0.4902 0.4902 0.4902]` (default) | RGB triplet | hexadecimal color code | color name | short color name

Mask color, specified as an RGB triplet, a hexadecimal color code, a color name, or a short color name.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 the 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'`	🟪

Color Name	Short Name	RGB Triplet	Hexadecimal Color Code	Appearance
`'yellow'`	`'y'`	`[1 1 0]`	`'#FFFF00'`	🟨
`'black'`	`'k'`	`[0 0 0]`	`'#000000'`	⬛
`'white'`	`'w'`	`[1 1 1]`	`'#FFFFFF'`	⬜
`'none'`	Not applicable	Not applicable	Not applicable	No color

This table shows 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'`	🟥

Example: `h = skyplot(__,MaskColor="r")`

Data Types: `double` | `string` | `char`

`MaskAzimuthEdges` — Mask angle azimuth edges
`[0 360]` (default) | *m*-element row vector

Mask angle azimuth edges, specified as an *m*-element row vector, where *m* is the total number of azimuth edges.

Example: `h = skyplot(__,MaskAzimuthEdges=0:45:360)`

Data Types: `double`

`MaskAzimuthEdgesMode` — Mask angle azimuth edges mode
`"auto"` (default) | `"manual"`

Mask angle azimuth edges mode, specified as `"auto"` or `"manual"`.

- `"auto"` — Automatically divide mask angle azimuth edges evenly between `0` and `360` degrees.
- `"manual"` — Specify mask angle azimuth edges manually using the `MaskAzimuthEdges` property.

Example: `h = skyplot(__,MaskAzimuthEdgesMode="manual")`

Data Types: `string` | `char`

Chart Properties

`HandleVisibility` — Visibility of object handle
`'on'` (default) | `'off'` | `'callback'`

Visibility of the `SkyPlotChart` object handle in the `Children` property of the parent, specified as one of these values:

- `'on'` — Object handle is always visible.
- `'off'` — Object handle is invisible at all times. This option is useful for preventing unintended changes to the UI by another function. To temporarily hide the handle during the execution of that function, set the `HandleVisibility` to `'off'`.
- `'callback'` — Object handle is visible from within callbacks or functions invoked by callbacks, but not from within functions invoked from the command line. This option blocks access to the object at the command line, but allows callback functions to access it.

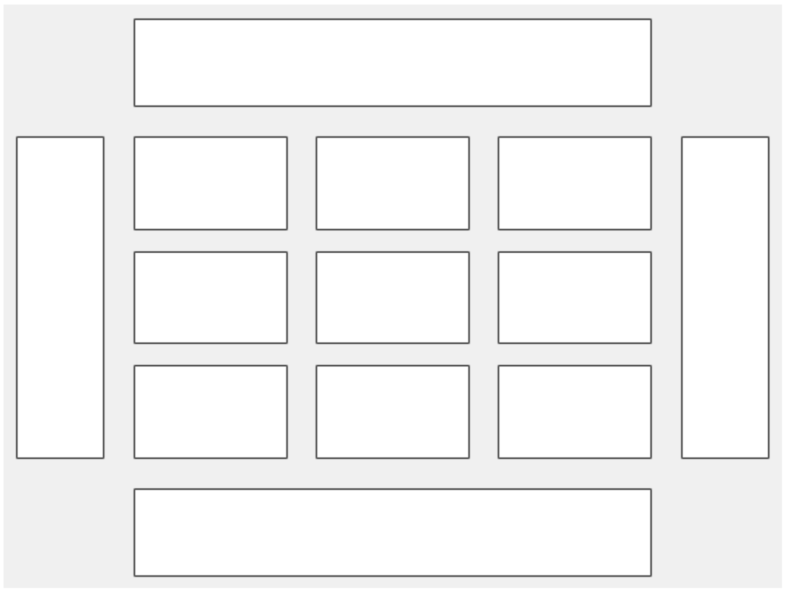
If the object is not listed in the `Children` property of the parent, then functions that obtain object handles by searching the object hierarchy or querying handle properties cannot return it. This includes `get`, `findobj`, `gca`, `gcf`, `gco`, `newplot`, `cla`, `clf`, and `close`.

Hidden object handles are still valid. Set the root `ShowHiddenHandles` property to `'on'` to list all object handles, regardless of their `HandleVisibility` property setting.

`Layout` — Layout options
empty `LayoutOptions` array (default) | `TiledChartLayoutOptions` object | `GridLayoutOptions` object

Layout options, specified as a `TiledChartLayoutOptions` or `GridLayoutOptions` object. This property is useful when the chart is either in a tiled chart layout or a grid layout.

To position the chart within the grid of a tiled chart layout, set the `Tile` and `TileSpan` properties on the `TiledChartLayoutOptions` object. For example, consider a 3-by-3 tiled chart layout. The layout has a grid of tiles in the center, and four tiles along the outer edges. In practice, the grid is invisible and the outer tiles do not take up space until you populate them with axes or charts.



This code places the chart `c` in the third tile of the grid.

```
c.Layout.Tile = 3;
```

To make the chart span multiple tiles, specify the `TileSpan` property as a two-element vector. For example, this chart spans 2 rows and 3 columns of tiles.

```
c.Layout.TileSpan = [2 3];
```

To place the chart in one of the surrounding tiles, specify the `Tile` property as `"north"`, `"south"`, `"east"`, or `"west"`. For example, setting the value to `"east"` places the chart in the tile to the right of the grid.

```
c.Layout.Tile = "east";
```

To place the chart into a layout within an app, specify this property as a `GridLayoutOptions` object. For more information about working with grid layouts in apps, see `uigridlayout`.

If the chart is not a child of either a tiled chart layout or a grid layout (for example, if it is a child of a figure or panel) then this property is empty and has no effect.

`Parent` — Parent container
`Figure` object | `Panel` object | `Tab` object | `TiledChartLayout` object | `GridLayout` object

Parent container, specified as a `Figure`, `Panel`, `Tab`, `TiledChartLayout`, or `GridLayout` object.

Marker Properties

`MarkerEdgeAlpha` — Marker edge transparency
`1` (default) | scalar in range `[0,1]` | `'flat'`

Marker edge transparency, specified as a scalar in the range `[0,1]` or `'flat'`. A value of 1 is opaque and 0 is completely transparent. Values between 0 and 1 are semitransparent.

To set the edge transparency to a different value for each point in the plot, set the `AlphaData` property to a vector the same size as the `XData` property, and set the `MarkerEdgeAlpha` property to `'flat'`.

`MarkerEdgeColor` — Marker outline color
`'flat'` (default) | `'auto'` | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...

Marker outline color, specified as `'auto'`, an RGB triplet, a hexadecimal color code, a color name, or a short name. The value of `'auto'` uses the same color as the `Color` property.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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 the 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'`	⬜
`'none'`	Not applicable	Not applicable	Not applicable	No color

This table shows 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'`	🟥

`MarkerFaceAlpha` — Marker face transparency
`0.6` (default) | scalar in range `[0,1]` | `'flat'`

Marker face transparency, specified as a scalar in the range `[0,1]` or `'flat'`. A value of 1 is opaque and 0 is completely transparent. Values between 0 and 1 are partially transparent.

To set the marker face transparency to a different value for each point, set the `AlphaData` property to a vector the same size as the `XData` property, and set the `MarkerFaceAlpha` property to `'flat'`.

`MarkerFaceColor` — Marker fill color
`'flat'` (default) | `'auto'` | `'none'` | RGB triplet | hexadecimal color code | `'r'` | `'g'` | `'b'` | ...

Marker fill color, specified as `'flat'`, `'auto'`, an RGB triplet, a hexadecimal color code, a color name, or a short name. The `'flat'` option uses the `CData` values. The `'auto'` option uses the same color as the `Color` property for the axes.

For a custom color, specify an RGB triplet or a hexadecimal color code.

- 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'`	⬜
`'none'`	Not applicable	Not applicable	Not applicable	No color

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'`	🟥

Example: `[0.3 0.2 0.1]`

Example: `'green'`

Example: `'#D2F9A7'`

`MarkerSizeData` — Marker size

`100` (default) | positive scalar | vector of positive values

Marker size, specified as a positive scalar or vector of positive values in points, where one point = 1/72 of an inch. If specified as a vector, the vector must be of the same length as `AzimuthData`.

Position

`PositionConstraint` — Position to hold constant

`"outerposition"` | `"innerposition"`

Position property to hold constant when adding, removing, or changing decorations, specified as one of the following values:

- `"outerposition"` — The `OuterPosition` property remains constant when you add, remove, or change decorations such as a title or an axis label. If any positional adjustments are needed, MATLAB adjusts the `InnerPosition` property.
- `"innerposition"` — The `InnerPosition` property remains constant when you add, remove, or change decorations such as a title or an axis label. If any positional adjustments are needed, MATLAB adjusts the `OuterPosition` property.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`OuterPosition` — Outer size and location

`[0 0 1 1]` (default) | four-element vector

Outer size and location of the skyplot within the parent container (typically a figure, panel, or tab), specified as a four-element vector of the form `[left bottom width height]`. The outer position includes the colorbar, title, and axis labels.

- The `left` and `bottom` elements define the distance from the lower-left corner of the container to the lower-left corner of the skyplot.
- The `width` and `height` elements are the skyplot dimensions, which include the skyplot cells, plus a margin for the surrounding text and colorbar.

The default value of `[0 0 1 1]` covers the whole interior of the container. The units are normalized relative to the size of the container. To change the units, set the `Units` property.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`InnerPosition` — Inner size and location

`[0.1300 0.1100 0.7750 0.8114]` (default) | four-element vector

Inner size and location of the skyplot within the parent container (typically a figure, panel, or tab), specified as a four-element vector of the form `[left bottom width height]`. The inner position does not include the colorbar, title, or axis labels.

- The `left` and `bottom` elements define the distance from the lower-left corner of the container to the lower-left corner of the skyplot.
- The `width` and `height` elements are the skyplot dimensions, which include only the skyplot cells.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`Position` — Inner size and location

four-element vector

Inner size and location of the skyplot within the parent container (typically a figure, panel, or tab), specified as a four-element vector of the form `[left bottom width height]`. This property is equivalent to the `InnerPosition` property.

Note Setting this property has no effect when the parent container is a `TiledChartLayout`.

`Units` — Position units

`'normalized'` (default) | `'inches'` | `'centimeters'` | `'points'` | `'pixels'` | `'characters'`

Position units, specified as one of these values.

Units	Description
`'normalized'` (default)	Normalized with respect to the container, which is typically the figure or a panel. The lower left corner of the container maps to `(0,0)`, and the upper right corner maps to `(1,1)`.
`'inches'`	Inches.
`'centimeters'`	Centimeters.
`'characters'`	Based on the default `uicontrol` font of the graphics root object: • Character width = width of letter `x`. • Character height = distance between the baselines of two lines of text.
`'points'`	Typography points. One point equals 1/72 inch.
`'pixels'`	Pixels. Starting in R2015b, distances in pixels are independent of your system resolution on Windows® and Macintosh systems: • On Windows systems, a pixel is 1/96th of an inch. • On Macintosh systems, a pixel is 1/72nd of an inch. On Linux® systems, the size of a pixel is determined by your system resolution.

When specifying the units as a name-value argument during object creation, you must set the `Units` property before specifying the properties that you want to use these units, such as `OuterPosition`.

`Visible` — State of visibility

`'on'` (default) | on/off logical value

State of visibility, specified as `'on'` or `'off'`, or as numeric or logical `1` (`true`) or `0` (`false`). A value of `'on'` is equivalent to `true`, and `'off'` is equivalent to `false`. Thus, you can use the value of this property as a logical value. The value is stored as an on/off logical value of type `matlab.lang.OnOffSwitchState`.

- `'on'` — Display the skyplot.
- `'off'` — Hide the skyplot without deleting it. You can still access the properties of an invisible `SkyPlotChart` object.

Version History

Introduced in R2021a

R2022b: `SkyPlotChart` supports azimuth and elevation trajectories and elevation masks

- The `AzimuthData` and `ElevationData` properties now accept matrices, enabling you to represent trajectories by adding azimuth and elevation data for satellites at multiple time steps.
- Elevation angle masks are now supported using these new properties:
  - `MaskElevation`
  - `MaskAlpha`
  - `MaskColor`
  - `MaskAzimuthEdges`
  - `MaskAzimuthEdgesMode`

See Also

Functions
`skyplot` | `polarscatter`

Objects
`gnssSensor` | `nmeaParser`

CoordinateAxes

Coordinate axes triad graphic object

Description

`CoordinateAxes` object defines a coordinate axes triad graphic that corresponds to the body frame of a satellite scenario asset.

Creation

You can create a `CoordinateAxes` object using the `coordinateAxes` object function of the `Satellite`, `GroundStation`, `Gimbal`, `ConicalSensor`, `Transmitter`, or `Receiver`.

Properties

`Scale` — Scale size of coordinate axes triad graphic
`1` (default) | positive scalar

Scale size of coordinate axes triad, specified as a positive scalar.

Data Types: `double`

`VisibilityMode` — Visibility of body coordinate axes triad
`'auto'` (default) | `'inherit'` | `'manual'`

Visibility of coordinate axes triad graphic relative to the parent in the Satellite Scenario Viewer, specified as:

- `'auto'` — Hide coordinate axes triad.
- `'inherit'` — Inherit visibility from the asset.
- `'manual'` — Manually specify visibility. In this mode, the visibility of the coordinate axes triad is independent of the visibility of the asset.

Data Types: `string`

Object Functions

show	Show object in satellite scenario viewer
hide	Hide satellite scenario entity from viewer

Examples

Visualize Coordinate Axes Triad of Satellite

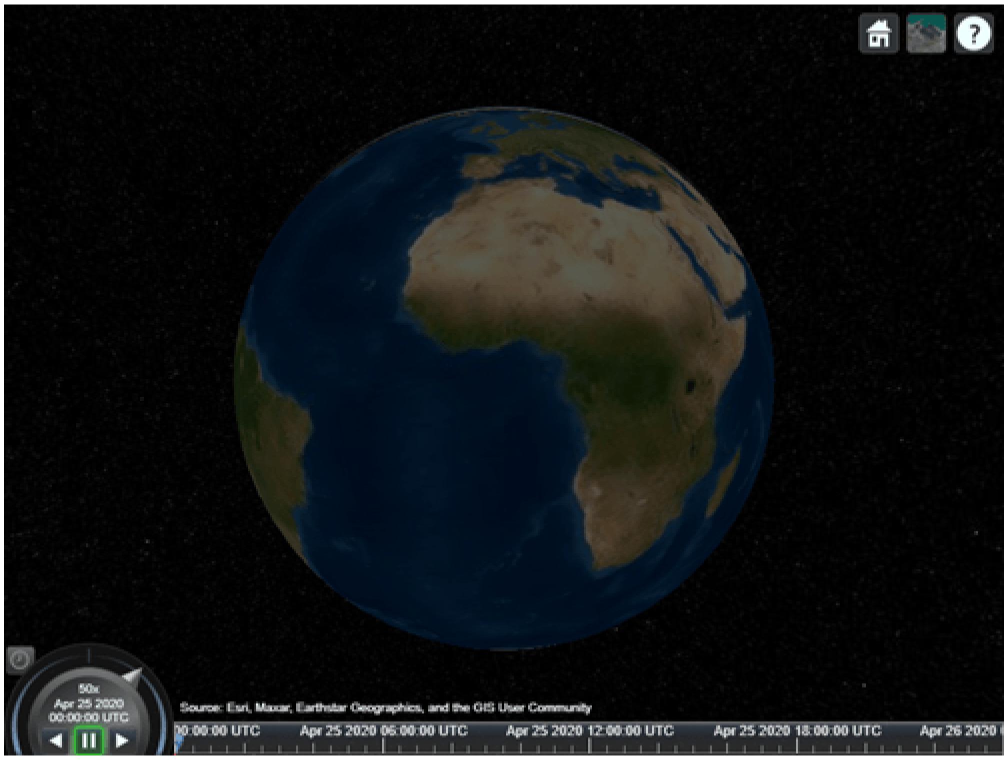
Create a `satelliteScenario` object.

```
startTime = datetime(2020,4,25); % April 25, 2020, 12:00 AM UTC
stopTime = datetime(2020,4,26);  % April 26, 2020, 12:00 AM UTC
```

```
sampleTime = 60;                  % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

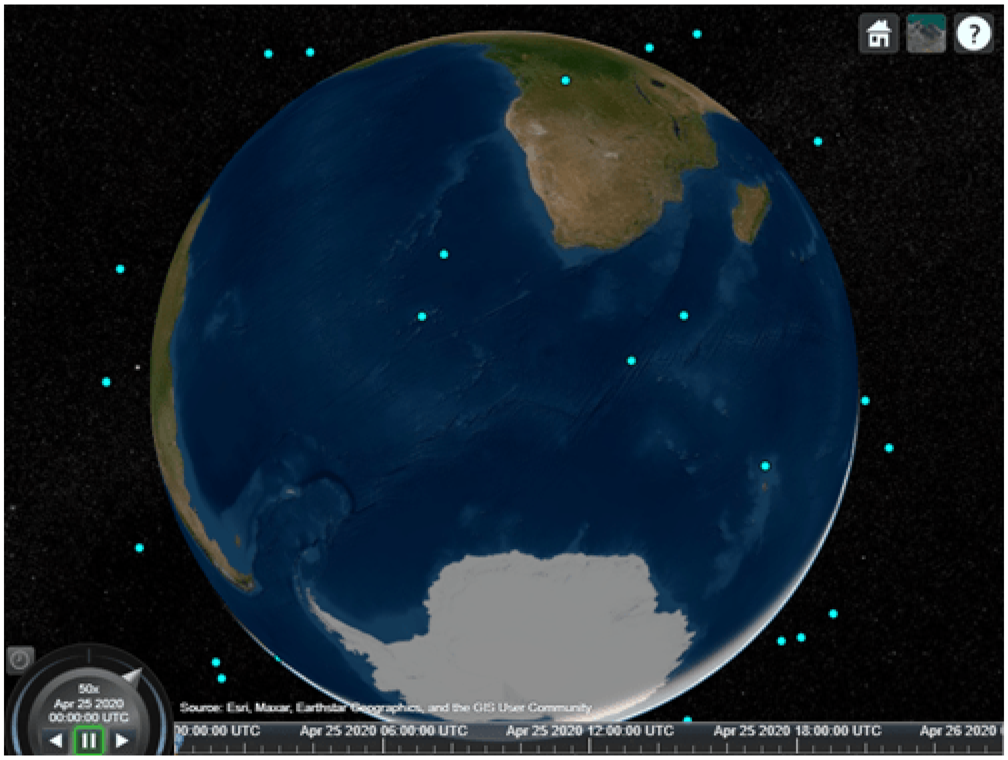
Launch a satellite scenario viewer with `ShowDetails` set to false.

```
v = satelliteScenarioViewer(sc,ShowDetails=false);
```



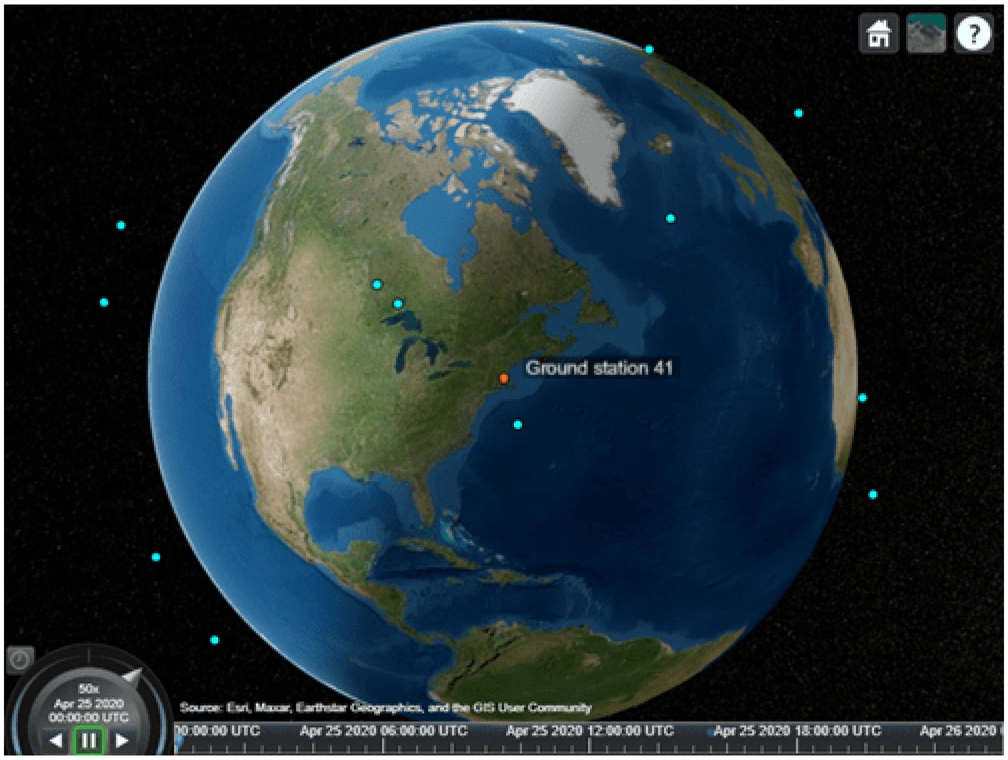
Add a low-earth orbit satellite constellation to the scenario from a TLE file.

```
tleFile = "leoSatelliteConstellation.tle";
sat = satellite(sc,tleFile);
```



Add a ground station to the scenario.

```
gs = groundStation(sc);
gs.ShowLabel = true;
```



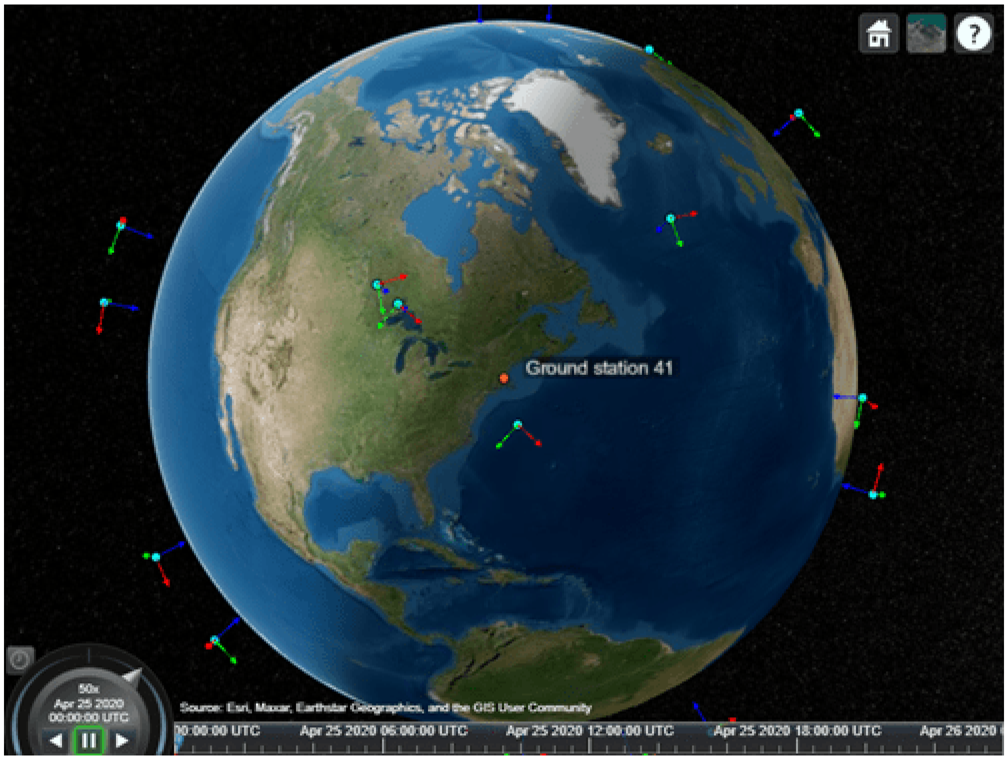
Visualize the coordinate axes triads of the satellites.

```
ax = coordinateAxes(sat)
```

```
ax=1×40 object
  1×40 CoordinateAxes array with properties:

    Scale
    VisibilityMode
```

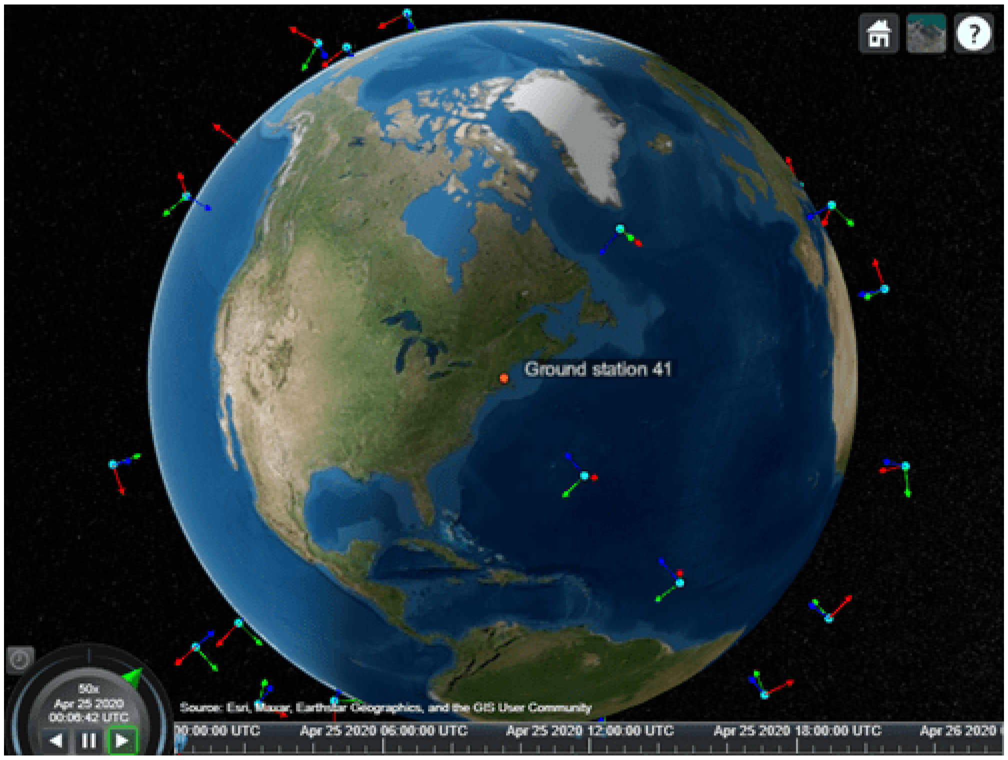
Point the satellites towards the ground station. The coordinate axes triad object updates to represent the new direction of the satellites.

```
pointAt(sat,gs);
```



Play the scenario.

```
play(sc);
```



Version History

Introduced in R2023a

See Also

Objects

`satelliteScenario` | `satelliteScenarioViewer`

Functions

`coordinateAxes` | `show` | `play` | `hide` | `satellite` | `access` | `groundStation` | `conicalSensor` | `transmitter` | `receiver`

Topics

"Satellite Scenario Key Concepts"
"Satellite Scenario Basics"

System Objects

ccsdsTMWaveformGenerator

Generate CCSDS TM waveform

Description

The `ccsdsTMWaveformGenerator` System object generates a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) time-domain waveform. The object implements the waveform generation aspects of CCSDS standard blue books:

- CCSDS 131.0-B-3 — TM synchronization and channel coding [1]
- CCSDS 401.0-B-30 — Radio frequency and modulation systems [2]
- CCSDS 131.2-B-1 — Flexible advanced coding and modulation scheme for high rate TM applications [3]

Note The object supports waveform generation specified by the CCSDS TM synchronization and channel coding standard [1] and CCSDS flexible advanced coding and modulation scheme for high rate TM standard [3]. To obtain the waveform for either of the desired standard, set the `WaveformSource` property.

To generate a CCSDS TM waveform:

1. Create the `ccsdsTMWaveformGenerator` 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

```
tmWaveGen = ccsdsTMWaveformGenerator
tmWaveGen = ccsdsTMWaveformGenerator(Name,Value)
```

Description

`tmWaveGen = ccsdsTMWaveformGenerator` creates a default CCSDS TM waveform generator System object.

`tmWaveGen = ccsdsTMWaveformGenerator(Name,Value)` sets "Properties" on page 4-3 using one or more name-value pairs. For example, `ccsdsTMWaveformGenerator("WaveformSource","flexible advanced coding and modulation","ACMFormat",20)` specifies the CSSDS TM waveform source as flexible advanced coding and modulation standard with ACM format as 20 for the generated waveform.

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.

General

`WaveformSource` — CCSDS TM waveform source
`"synchronization and channel coding"` (default) | `"flexible advanced coding and modulation"`

CCSDS TM waveform source, specified as one of these values.

- `"synchronization and channel coding"` — Use this option to set the waveform to CCSDS TM synchronization and channel coding, as specified in CCSDS 131.0-B-3 [1].
- `"flexible advanced coding and modulation"` — Use this option to set the waveform to CCSDS flexible advanced coding and modulation for high rate TM applications, as specified in CCSDS 131.2-B-1 [3].

Data Types: `char` | `string`

`ACMFormat` — ACM format
`1` (default) | integer in the range [1, 27]

Adaptive coding and modulation (ACM) format, specified as an integer in the range [1, 27], as specified in CCSDS 131.2-B-1 Section 5.2.4 Table 5-2 [3].

Tunable: Yes

Dependencies

To enable this property, set the `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `uint8`

`NumBytesInTransferFrame` — Number of bytes in one transfer frame
`223` (default) | integer in the range [1, 2048]

Number of bytes in one transfer frame, specified as an integer in the range [1, 2048].

Dependencies

To enable this property, one of these conditions should be satisfied:

- Set `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"none"`, `"convolutional"`, or `"LDPC"` on stream of sync marked transfer frame (SMTF).
- Set `WaveformSource` property to `"flexible advanced coding and modulation"`. In this case, the minimum number of `NumBytesInTransferFrame` is 223.

For other values of `ChannelCoding`, this `NumBytesInTransferFrame` property is calculated internally based on other properties.

Data Types: `double` | `uint16`

`HasRandomizer` — Option for randomizing data
`1` or `true` (default) | `0` or `false`

Option for randomizing the data, specified as a numeric or `logical` value of `1` (`true`) or `0` (`false`). Set this value to `1` (`true`) to randomize the data present in the channel access data unit (CADU).

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`.

When you set the `ChannelCoding` property to `"LDPC"` and `IsLDPCOnSMTF` property to `1` (`true`), this property is not applicable, and is set to `1` (`true`).

Data Types: `double` | `logical`

`HasASM` — Option for inserting ASM
`1` or `true` (default) | `0` or `false`

Option for inserting attached sync marker (ASM), specified as a numeric or `logical` value of `1` (`true`) or `0` (`false`). Set this value to `1` (`true`) to indicate the data in CADU is attached with ASM.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`.

When you set the `ChannelCoding` property to `"LDPC"` and `IsLDPCOnSMTF` property to `1` (`true`), this property is not applicable, and is set to `1` (`true`).

Data Types: `double` | `logical`

`PCMFormat` — PCM format
`"NRZ-L"` (default) | `"NRZ-M"`

Pulse code modulation (PCM) format to select the PCM coding in the CCSDS TM waveform, specified as one of these values.

- `"NRZ-L"` — NRZ-level
- `"NRZ-M"` — NRZ-mark

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"BPSK"`, `"QPSK"`, `"8PSK"`, `"OPSK"`, or `"PCM/PSK/PM"`.

Data Types: `char` | `string`

Channel Coding

`ChannelCoding` — Forward error correction coding scheme
`"RS"` (default) | `"none"` | `"convolutional"` | `"concatenated"` | `"turbo"` | `"LDPC"`

Forward error correction coding scheme, specified as one of these values.

- `"none"`
- `"RS"`
- `"convolutional"`
- `"concatenated"`
- `"turbo"`
- `"LDPC"`

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`.

Data Types: `char` | `string`

`NumBitsInInformationBlock` — Number of bits in turbo or LDPC message

`7136` (default) | `1784` | `3568` | `8920` | `1024` | `4096` | `16384`

Number of bits in the turbo or lower density parity check (LDPC) message, specified as one of these values.

- `1784`, `3568`, `7136`, or `8920` — Use one of these values when you set the `ChannelCoding` property to `"turbo"`.
- `1024`, `4096`, `16384`, or `7136` — Use one of these values when you set the `ChannelCoding` property to `"LDPC"`.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to either `"turbo"` or `"LDPC"`.

Data Types: `double` | `uint8`

`ConvolutionalCodeRate` — Code rate of convolutional code

`"1/2"` (default) | `"2/3"` | `"3/4"` | `"5/6"` | `"7/8"`

Code rate of convolutional code, specified as a one of these values.

- `"1/2"`
- `"2/3"`
- `"3/4"`
- `"5/6"`
- `"7/8"`

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to either `"convolutional"` or `"concatenated"`.

When you set the `ChannelCoding` property to `"concatenated"`, the numeric value of the code rate also depends on the constituent Reed-Solomon (RS) code. You can obtain the actual numeric value for any code from the output field `ActualCodeRate` of the `info` object function.

Data Types: char | string

CodeRate — Code rate of turbo or LDPC code
"1/2" (for turbo code) (default) | "7/8" (for LDPC code) (default) | "2/3" | "1/3" | "1/4" | "1/6" | "4/5"

Code rate of turbo or LDPC code, specified as one of these values.

- "1/2", "1/3", "1/4", or "1/6" — Use one of these values when you set the ChannelCoding property to "turbo".
- "1/2", "2/3", "4/5", or "7/8" — Use one of these values when you set the ChannelCoding property to "LDPC".

Note When you set the ChannelCoding property to "LDPC" and the NumBitsInInformationBlock property to 7136, the CodeRate must be "7/8".

For an LDPC code, setting CodeRate to 7/8 implies an actual code rate numeric value of 223/255. You can obtain the actual numeric value for any code from the output field ActualCodeRate of the info object function.

Dependencies

To enable this property, set the WaveformSource property to "synchronization and channel coding" and the ChannelCoding property to either "turbo" or "LDPC".

Data Types: char | string

RSMessageLength — Number of bytes in one RS message block
223 (default) | 239

Number of bytes in one RS message block, specified as 223 or 239.

Dependencies

To enable this property, set the WaveformSource property to "synchronization and channel coding" and the ChannelCoding property to "RS" or "concatenated".

Data Types: double | uint8

RSInterleavingDepth — Interleaving depth of RS code
1 (default) | 2 | 3 | 4 | 5 | 8

Interleaving depth of the RS code, specified as 1, 2, 3, 4, 5, or 8. The interleaving depth is the number of RS codewords in one code block.

Dependencies

To enable this property, set the WaveformSource property to "synchronization and channel coding" and the ChannelCoding property to "RS" or "concatenated".

Data Types: double | uint8

IsRSMessageShortened — Option to shorten RS code
0 or false (default) | 1 or true

Option to shorten the RS code, specified as a numeric or `logical` value of `0` (`false`) or `1` (`true`). Set this value to `1` (`true`) to shorten the RS code.

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"RS"` or `"concatenated"`.

Data Types: `double` | `logical`

`RSShortenedMessageLength` — Number of bytes in RS shortened message block
`223` (default) | integer in the range [1, `RSMessageLength`]

Number of bytes in the RS shortened message block, specified as an integer in the range [1, `RSMessageLength`].

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`, the `ChannelCoding` property to `"RS"` or `"concatenated"`, and the `IsRSMessageShortened` property to `1` (`true`).

Data Types: `double` | `uint8`

`IsLDPCOnSMTF` — Option for using LDPC on stream of SMTF
`0` or `false` (default) | `1` or `true`

Option for using LDPC on the stream of a sync marked transfer frame (SMTF), specified as a numeric or `logical` value of `0` (`false`) or `1` (`true`). Set this value to `1` (`true`) to indicate LDPC on the stream of SMTF as specified in CCSDS 131.0-B-3 Section 8 of the TM synchronization and channel coding standard [1]. To indicate LDPC on the transfer frame, set this value to `0` (`false`).

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"` and the `ChannelCoding` property to `"LDPC"`.

Data Types: `double` | `logical`

`LDPCCodeBlockSize` — Number of LDPC codewords in LDPC code block of stream of SMTF
`1` (default) | integer in the range [1, 8]

Number of LDPC codewords in the LDPC code block of the stream of SMTF, specified as an integer in the range [1, 8].

Dependencies

To enable this property, set the `WaveformSource` property to `"synchronization and channel coding"`, the `ChannelCoding` property to `"LDPC"`, and the `IsLDPCOnSMTF` property to `true`.

Data Types: `double` | `uint8`

Digital Modulation and Filter

`Modulation` — Modulation scheme
`"QPSK"` (default) | `"BPSK"` | `"8PSK"` | `"OQPSK"` | `"GMSK"` | `"PCM/PSK/PM"` | `"PCM/PM/biphase-L"` | `"4D-8PSK-TCM"`

Modulation scheme used in CCSDS TC waveform, specified as one of these values.

- "QPSK"
- "BPSK"
- "8PSK"
- "OQPSK"
- "GMSK"
- "PCM/PSK/PM"
- "PCM/PM/biphase-L"
- "4D-8PSK-TCM"

Dependencies

To enable this property, set the WaveformSource property to "synchronization and channel coding".

Data Types: char | string

PulseShapingFilter — Pulse shaping filter
"root raised cosine" (default) | "none"

Pulse shaping filter, specified as "root raised cosine" or "none".

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set WaveformSource property to "synchronization and channel coding" and the Modulation property to "BPSK", "QPSK", "8PSK", or "4D-8PSK-TCM".
- Set WaveformSource property to"flexible advanced coding and modulation".

Data Types: char | string

RolloffFactor — Roll-off factor of SRRC baseband filter
0.35 (default) | scalar in the range [0, 1]

Roll-off factor of the square root raised cosine (SRRC) baseband filter, specified as a scalar in the range [0, 1].

Note This property is not applicable when you set the PulseShapingFilter property to "none" for either value of the WaveformSource property.

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set WaveformSource property to "synchronization and channel coding" and the Modulation property to either "BPSK", "QPSK", "8PSK", "OQPSK", or "4D-8PSK-TCM".
- Set WaveformSource property to "flexible advanced coding and modulation".

Data Types: double

FilterSpanInSymbols — Filter span in number of symbols
10 (default) | positive integer

Filter span in number of symbols, specified as a positive integer.

The `ccsdsTMWaveformGenerator` System object truncates the infinite impulse response of the ideal root raised cosine filter to this value.

Note This property is not applicable when you set the `PulseShapingFilter` property to `"none"` for either value of the `WaveformSource` property.

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to either `"BPSK"`, `"QPSK"`, `"8PSK"`, `"OQPSK"`, or `"4D-8PSK-TCM"`.
- Set `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `uint32`

`BandwidthTimeProduct` — Bandwidth time product for GMSK modulator
`0.25` (default) | `0.5`

Bandwidth time product for the Gaussian minimum shift keying (GMSK) modulator, specified as `0.25` or `0.5`.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"GMSK"`.

Data Types: `double`

`ModulationEfficiency` — Modulation efficiency of 4D-8PSK-TCM
`2` (default) | `2.25` | `2.5` | `2.75`

Modulation efficiency of 4D-8PSK trellis coded modulator (TCM), specified as `2`, `2.25`, `2.5`, or `2.75`. This property indicates the number of bits for each complex baseband symbol.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"4D-8PSK-TCM"`.

Data Types: `double`

`SubcarrierWaveform` — Type of waveform to PSK-modulate NRZ data
`"sine"` (default) | `"square"`

Type of waveform to PSK-modulate the non-return-to-zero (NRZ) data, specified as `"sine"` or `"square"`.

Dependencies

To enable this property, set `WaveformSource` property to `"synchronization and channel coding"` and the `Modulation` property to `"PCM/PSK/PM"`.

Data Types: `char` | `string`

ModulationIndex — Modulation index in residual carrier phase modulation
0.4 (default) | scalar in the range [0.2, 2]

Modulation index in the residual carrier phase modulation, specified as a scalar in the range [0.2, 2]. Units are in radians.

Dependencies

To enable this property, set WaveformSource property to "synchronization and channel coding" and the Modulation property to "PCM/PSK/PM" or "PCM/PM/biphase-L".

Data Types: double

SymbolRate — Coded symbol rate
2000 (default) | positive scalar

Coded symbol rate in Hz, specified as a positive scalar.

Dependencies

To enable this property, set WaveformSource property to "synchronization and channel coding" and the Modulation property to "PCM/PSK/PM".

Data Types: double

SubcarrierToSymbolRateRatio — Ratio of subcarrier frequency to symbol rate
4 (default) | integer in the range [1, 50]

Ratio of the subcarrier frequency to the symbol rate, specified as an integer in the range [1, 50].

Dependencies

To enable this property, set WaveformSource property to "synchronization and channel coding" and the Modulation property to "PCM/PSK/PM".

Data Types: double | uint8

SamplesPerSymbol — Number of samples per symbol
10 (default) | positive integer

Number of samples per symbol, specified as a positive integer.

This property is applicable for either input value of the WaveformSource property.

Dependencies

To enable this property, one of these conditions must be satisfied:

- Set the Modulation property to "OQPSK", "PCM/PSK/PM", or "GMSK".
- Set the PulseShapingFilter to "root raised cosine".

Data Types: double | uint8

HasPilots — Option for inserting pilot symbols
0 or false (default) | 1 or true

Option for inserting pilot symbols within data, specified as a numeric or logical value of 0 (false) or 1 (true). Set this value to 1 (true) to indicate pilots are inserted, as described in CCSDS flexible advanced coding and modulation scheme for high rate TM standard [3].

Dependencies

To enable this property, set the `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `logical`

`ScramblingCodeNumber` — Scrambling code number
`0` (default) | integer in the range [0, (2^{18} – 2)]

Scrambling code number for flexible advanced coding and modulation for high rate TM applications standard [3], specified as an integer in the range [0, (2^{18} – 2)].

`ScramblingCodeNumber` is used to randomize the complex baseband symbols.

Dependencies

To enable this property, set the `WaveformSource` property to `"flexible advanced coding and modulation"`.

Data Types: `double` | `uint32`

Read-Only

`NumInputBits` — Minimum number of bits required to generate waveform
integer

This property is read-only.

Minimum number of input bits to generate a waveform, returned as an integer.

The number of input bits must be an integer multiple of `NumInputBits`.

Data Types: `double`

`MinNumTransferFrames` — Minimum number of transfer frames for nonempty output
integer

This property is read-only.

Minimum number of transfer frames for a nonempty System object output, returned as an integer.

When you set the `WaveformSource` property to `"flexible advanced coding and modulation"`, or to `"synchronization and channel coding"` with the `IsLDPCOnSMTF` property set to `1` (`true`), System object output is empty until it has sufficient input to process through channel coding and modulation.

Data Types: `double`

Usage

Syntax

```
txWaveform = tmWaveGen(bits)
[txWaveform,encodedBits] = tmWaveGen(bits)
```

Description

`txWaveform = tmWaveGen(bits)` generates a CCSDS TM time-domain waveform for the corresponding input bits.

`[txWaveform,encodedBits] = tmWaveGen(bits)` also returns the bits obtained after TM synchronization and channel coding sublayer operations.

Input Arguments

`bits` — Information bits
binary-valued column vector

Information bits, in the form of transfer frames, specified as a binary-valued column vector. The length of this vector must be an integer multiple of the number of bits in one transfer frame. The `NumInputBits` property indicates the number of bits in one transfer frame.

Data Types: `double` | `int8` | `logical`

Output Arguments

`txWaveform` — Generated CCSDS TM time-domain waveform
column vector

Generated CCSDS TM time-domain waveform, returned as a column vector. This output is generated in the form of complex in-phase quadrature (IQ) samples.

Data Types: `double`

`encodedBits` — Output bits obtained after TM synchronization and channel coding sublayer operations
binary-valued column vector

Output bits obtained after TM synchronization and channel coding sublayer operations, returned as a binary-valued column vector.

Data Types: `double` | `int8` | `logical`

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 `ccsdsTMWaveformGenerator`

info	Characteristic information about object
flushFilter	Flush transmit filter

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object

isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate CCSDS TM Waveform for Synchronization and Channel Coding Scheme

Generate a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) waveform for the synchronization and channel coding standard, for multiple transfer frames. Visualize the waveform by using a spectrum plot.

Create a CCSDS TM System object. Set the waveform type as `synchronization and channel coding` with GMSK-modulated concatenated codes.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "concatenated";
tmWaveGen.Modulation = "GMSK";
tmWaveGen.RSMessageLength = 239;
tmWaveGen.RSInterleavingDepth = 2;
tmWaveGen.BandwidthTimeProduct = 0.5;
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

           WaveformSource: "synchronization and channel coding"
            HasRandomizer: true
                   HasASM: true

   Channel coding
            ChannelCoding: "concatenated"
     ConvolutionalCodeRate: "1/2"
          RSMessageLength: 239
      RSInterleavingDepth: 2
     IsRSMessageShortened: false

   Digital modulation and filter
               Modulation: "GMSK"
     BandwidthTimeProduct: 0.5000
         SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

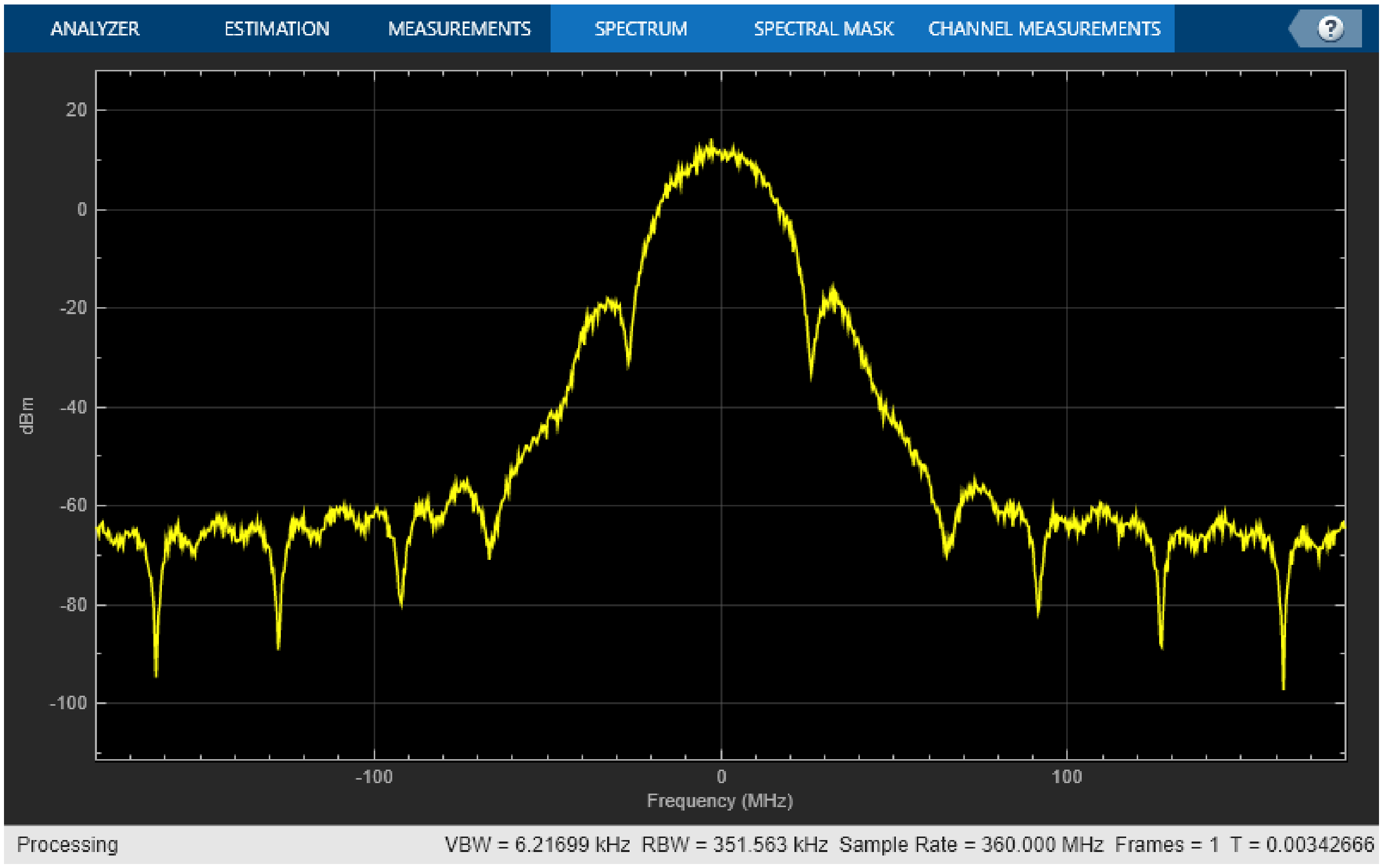
```
numTF = 15;
waveform = [];  % Initialize waveform as null
```

Generate the CCSDS TM waveform for the synchronization and channel coding standard by using multiple System object calls.

```
rng default    % For reproducible results
for iTF = 1:numTF
     bits = randi([0 1],tmWaveGen.NumInputBits,1);
     waveform = [waveform; tmWaveGen(bits)];
end
```

Create a spectrum analyzer System object to display the frequency spectrum of the generated CCSDS TM time-domain waveform.

```
BW = 36e6;     % Typical satellite channel bandwidth
Fsamp = tmWaveGen.SamplesPerSymbol*BW;
scope = spectrumAnalyzer('SampleRate',Fsamp,...
                          'AveragingMethod','Exponential');
scope(waveform)
```



Processing VBW = 6.21699 kHz RBW = 351.563 kHz Sample Rate = 360.000 MHz Frames = 1 T = 0.00342666

Generate CCSDS TM Waveform for Flexible Advanced Coding and Modulation Scheme

Generate a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) waveform for the flexible advanced coding and modulation scheme for high rate TM applications standard, for one physical layer (PL) frame. Visualize the waveform by using a scatter plot.

Create a CCSDS TM System object, and then specify its properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "flexible advanced coding and modulation";
tmWaveGen.ACMFormat = 17;   % 16APSK
tmWaveGen.PulseShapingFilter = "none";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:
```

```
            WaveformSource: "flexible advanced coding and modulation"
                 ACMFormat: 17
    NumBytesInTransferFrame: 223

   Channel coding
    No properties.

   Digital modulation and filter
         PulseShapingFilter: "none"
                  HasPilots: false
       ScramblingCodeNumber: 0

  Use get to show all properties
```

Calculate the number of transfer frames in one PL frame.

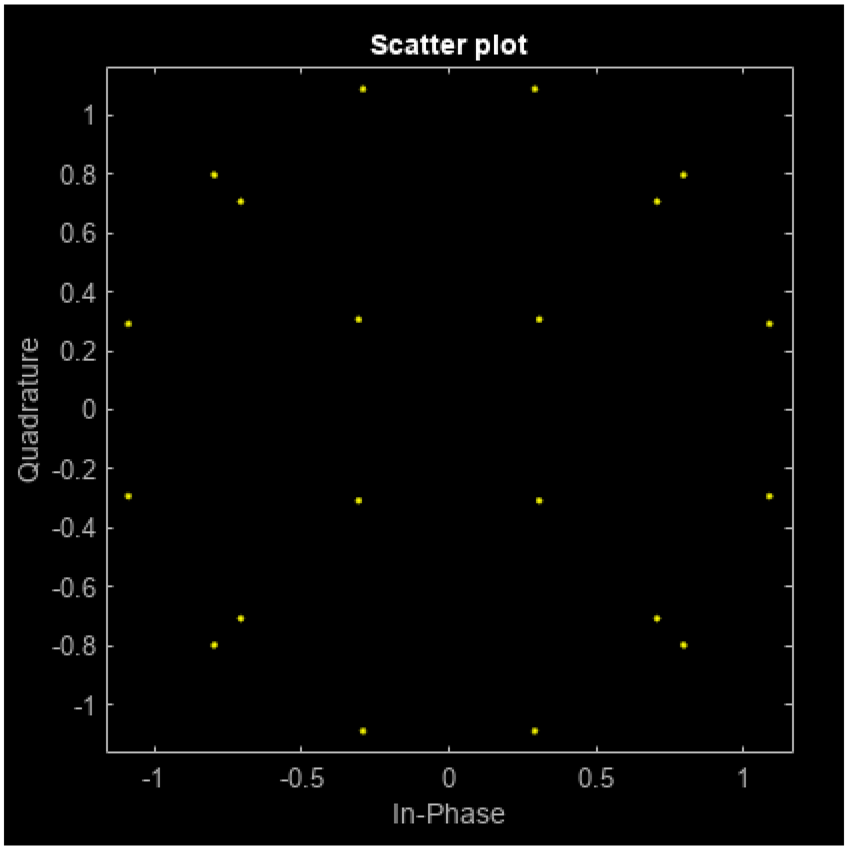
```
NumTFInOnePL = tmWaveGen.MinNumTransferFrames*16; % One PL frame consists of 16 codewords, as spe
waveform = [];  % Initialize waveform as null
```

Generate the CCSDS TM waveform for the flexible advanced coding and modulation scheme for high rate TM applications standard.

```
rng default    % For reproducible results
for iTF = 1:NumTFInOnePL
   bits = randi([0 1],tmWaveGen.NumInputBits,1);
   waveform = [waveform; tmWaveGen(bits)];
end
```

Display the scatter plot of the constellation for the generated waveform.

```
scatterplot(waveform);
legend off;
```



Get CCSDS TM Waveform Generator Information and Check Transmit Filter Delay

Get information from a `ccsdsTMWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

Create a Consultative Committee for Space Data Systems (CCSDS) Telemetry (TM) System object. Set the waveform type as `synchronization and channel coding` with low-density parity-check (LDPC) channel coding. Display the properties.

```
tmWaveGen = ccsdsTMWaveformGenerator;
tmWaveGen.WaveformSource = "synchronization and channel coding";
tmWaveGen.ChannelCoding = "LDPC";
tmWaveGen.NumBitsInInformationBlock = 1024;
tmWaveGen.Modulation = "QPSK";
tmWaveGen.CodeRate = "1/2";
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

               WaveformSource: "synchronization and channel coding"
                HasRandomizer: true
                       HasASM: true
                    PCMFormat: "NRZ-L"

   Channel coding
```

```
                ChannelCoding: "LDPC"
    NumBitsInInformationBlock: 1024
                     CodeRate: "1/2"
                 IsLDPCOnSMTF: false

   Digital modulation and filter
                   Modulation: "QPSK"
           PulseShapingFilter: "root raised cosine"
                RolloffFactor: 0.3500
          FilterSpanInSymbols: 10
             SamplesPerSymbol: 10

  Use get to show all properties
```

Specify the number of transfer frames.

```
numTF = 20;
```

Get the characteristic information about the CCSDS TM waveform generator.

```
info(tmWaveGen)
```

```
ans = struct with fields:
         ActualCodeRate: 0.5000
       NumBitsPerSymbol: 2
    SubcarrierFrequency: []
```

Generate the input bits for the CCSDS TM waveform generator, and then generate the waveform.

```
bits = randi([0 1], tmWaveGen.NumInputBits*numTF,1);
waveform = tmWaveGen(bits);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(tmWaveGen)
```

```
ans = 100×1 complex

  -0.0772 - 0.0867i
  -0.0751 - 0.0859i
  -0.0673 - 0.0788i
  -0.0549 - 0.0654i
  -0.0388 - 0.0469i
  -0.0200 - 0.0250i
   0.0002 - 0.0012i
   0.0208 + 0.0227i
   0.0405 + 0.0453i
   0.0587 + 0.0653i
      ⋮
```

Version History

Introduced in R2021a

References

[1] CCSDS 131.0-B-3. Blue Book. Issue 3. "TM Synchronization and Channel Coding." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, September 2017.

[2] CCSDS 401.0-B-30. Blue Book. Issue 30. "Radio Frequency and Modulation Systems - Part 1: Earth Stations and Spacecraft." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, February 2020.

[3] CCSDS 131.2-B-1. Blue Book. Issue 1. "Flexible Advanced Coding and Modulation Scheme for High Rate Telemetry Applications." *Recommendation for Space Data System Standards*. Washington, D.C.: CCSDS, March 2012.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`ccsdsTCWaveform` | `ccsdsTCIdealReceiver`

Objects
`ccsdsTCConfig`

dvbrcs2WaveformGenerator

Generate DVB-RCS2 waveform

Description

The `dvbrcs2WaveformGenerator` System object generates a Digital Video Broadcasting Second Generation Return Channel over Satellite (DVB-RCS2) time-domain reference or a custom waveform. The object implements the waveform generation aspects of ETSI EN 301 545-2 V1.2.1 (2014-11) [1].

To generate a DVB-RCS2 waveform:

1. Create the `dvbrcs2WaveformGenerator` 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

```
rcs2WaveGen = dvbrcs2WaveformGenerator
rcs2WaveGen = dvbrcs2WaveformGenerator(Name,Value)
```

Description

`rcs2WaveGen = dvbrcs2WaveformGenerator` creates a default DVB-RCS2 waveform generator System object.

`rcs2WaveGen = dvbrcs2WaveformGenerator(Name,Value)` sets properties on page 4-19 using one or more name-value arguments. For example, `'TransmissionFormat',"SS-TC-LM"` specifies to generate a reference DVB-RCS2 waveform of spread spectrum turbo codes with linear modulation (SS-TC-LM) format.

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.

`TransmissionFormat` — Transmission format
`"TC-LM"` (default) | `"SS-TC-LM"`

Transmission format, specified as one of these values.

- `"TC-LM"` — Turbo codes with linear modulation (TC-LM)
- `"SS-TC-LM"` — Spread spectrum turbo codes with linear modulation (SS-TC-LM)

Tunable: Yes

Data Types: `char` | `string`

`ContentType` — Frame PDU burst content type
`"traffic"` (default) | `"logon"` | `"control"`

Frame protocol data unit (PDU) burst content type, specified as `"traffic"`, `"logon"`, or `"control"`.

Data Types: `char` | `string`

`IsCustomWaveform` — Custom waveform indicator
`false` or `0` (default) | `true` or `1`

Custom waveform indicator, specified as one of these numeric or logical values.

- `0` (`false`) — Generate a standard-defined reference waveform. For details, refer to ETSI EN 301 545-2 V1.2.1 (2014-11) Annex A Tables A-1 and A-2 [1].
- `1` (`true`) — Generate a custom waveform.

Tunable: Yes

Data Types: `logical`

`WaveformID` — Reference waveform ID
`1` (default) | positive integer

Reference waveform ID, specified as one of these options.

- Integer in the range [1, 22] or [32, 49] — Use this option when you set the `TransmissionFormat` property to `"TC-LM"`.
- Integer in the range [1, 19] — Use this option when you set the `TransmissionFormat` property to `"SS-TC-LM"`.

Based on the `TransmissionFormat` and `WaveformID` properties, the System object considers the transmission parameters according to ETSI EN 301 545-2 Annex A Table A-1 and A-2 [1].

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `false`.

Data Types: `double` | `unit8`

`PreBurstGuardLength` — Preburst guard length
`0` (default) | nonnegative integer

Preburst guard length, specified as a nonnegative integer. This length represents the number of zero-valued symbols in the guard time that are prefixed to the burst symbols, prior to the preamble.

A value of `0` indicates no guard symbols are prefixed.

Tunable: Yes

Data Types: `double`

`PostBurstGuardLength` — Postburst guard length
`0` (default) | nonnegative integer

Postburst guard length, specified as a nonnegative integer. This length represents the number of zero-valued symbols in the guard time that are suffixed to the burst symbols, after the postamble.

In absence of the postamble, these symbols are suffixed directly after the payload symbols.

Tunable: Yes

Data Types: `double`

`FilterSpanInSymbols` — Filter span in symbols
`10` (default) | positive integer

Filter span in symbols, specified as a positive integer.

The ideal impulse response of the raised cosine filter is truncated to a length that spans the number of symbols specified in this property.

Data Types: `double`

`SamplesPerSymbol` — Number of samples per symbol
`4` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Data Types: `double`

`PayloadLengthInBytes` — Payload length
`10` (default) | positive integer

Payload length in bytes, specified as one of these options.

- Integer in the range [3, 65,535] — Use this option when you set the `ContentType`property to `"control"` or `"logon"`.
- Integer in the range [5, 65,535] — Use this option when you set the `ContentType` property to `"traffic"`.

This length represents the size of the input data to the turbo encoder of this System object. Input data includes the frame PDU and the cyclic redundancy check (CRC) bits.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`MappingScheme` — Mapping scheme
`"pi/2-BPSK"` (default) | `"QPSK"` | `"8PSK"` | `"16QAM"`

Mapping scheme, specified as one of these values.

- `"pi/2-BPSK"`
- `"QPSK"`
- `"8PSK"`
- `"16QAM"`

Dependencies

To enable this property, set the `TransmissionFormat` property to `"TC-LM"` and the `IsCustomWaveform` property to `true`.

Note When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the only valid value of `MappingScheme` is `"pi/2-BPSK"`.

Data Types: `char` | `string`

`CodeRate` — Code rate
`"1/3"` (default) | `"1/2"` | `"2/3"` | `"3/4"` | `"4/5"` | `"5/6"` | `"6/7"` | `"7/8"`

Code rate, specified as one of these values.

- `"2/3"`, `"3/4"`, `"4/5"`, `"5/6"`, `"6/7"`, or `"7/8"` — Use one of these values when you set the `MappingScheme` property to `"8PSK"`.
- `"3/4"`, `"4/5"`, `"5/6"`, `"6/7"`, or `"7/8"` — Use one of these values when you set the `MappingScheme` property to `"16QAM"`.

All code rates are applicable if `MappingScheme` property is set to `"pi/2-BPSK"` or `"QPSK"`.

This code rate is passed as an input to the turbo encoder function, that is, `dvbrcs2TurboEncode`, of this System object.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `char` | `string`

`PreambleLength` — Preamble length
8 (default) | integer in the range [0, 255]

Preamble length, specified as an integer in the range [0, 255].

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of preamble length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of preamble length is chips.

A preamble of this specified length is prefixed to the burst sequence, prior to the modulation.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PostambleLength` — Postamble length
`8` (default) | integer in the range [0, 255]

Postamble length, specified as an integer in the range [0, 255].

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of postamble length is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of postamble length is chips.

A postamble of this specified length is suffixed to the burst sequence, prior to the modulation.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PilotPeriod` — Pilot period
`0` (default) | integer in the range [0, 4095]

Pilot period, specified as an integer in the range [0, 4095]. A value of `0` indicates no pilots are inserted.

When you set the `TransmissionFormat` property to `"TC-LM"`, the unit of pilot period is symbols. When you set the `TransmissionFormat` property to `"SS-TC-LM"`, the unit of pilot period is chips.

The pilot period represents the length of the sequence from first symbol of a pilot block to the first symbol of the next pilot block in symbols or chips.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`PilotBlockLength` — Pilot block length
`1` (default) | integer in the range [1, 255]

Pilot block length, specified as an integer in the range [1, 255].

After every `PilotPeriod` symbols or chips, a pilot block of this specified length is added in the burst sequence.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true` and `PilotPeriod` property to a positive integer.

Data Types: `double`

`PermutationParameters` — Permutation control parameters

`[9 0 0 0 0]` (default) | vector

Permutation control parameters that the dvbrcs2WaveformGenerator uses to generate turbo encoder interleaver indices, specified as a five-element vector in order: P, Q_0, Q_1, Q_2, and Q_3. P must be in the range [9, 255], and Q_0, Q_1, Q_2, and Q_3 must be in the range [0, 15].

To generate unique interleaver indices, the value of P must be co-prime to `PayloadLengthInBytes`*4.

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `double`

`UniqueWord` — Unique word

`"FFFF"` (default) | character array | string scalar

Unique word, specified as a character array or string scalar.

A unique word is a string of hexadecimal values that include the combination of the preamble, one pilot block, and the postamble sequence. Pilots are included only when you set the `PilotPeriod` property as nonzero.

To know the minimum required length of the unique word, use this formula.

`ceil`((`PreambleLength` + `PostambleLength` + `PilotBlockLength`)**bps*/4); where *bps* is the bits per seconds, determined by the `MappingScheme` specified.

For example, if `PreambleLength` = `9`, `PostambleLength` = `8`, `PilotBlockLength` = `1`, and `MappingScheme` = `"QPSK"` (*bps* = 2) then the minimum required length of the unique word by using this formula:

`ceil`((9 + 8 + 1)*2/4) = 9 (hexadecimal values)

Tunable: Yes

Dependencies

To enable this property, set the `IsCustomWaveform` property to `true`.

Data Types: `char` | `string`

`SpreadingFactor` — Spreading factor

2 (default) | integer in the range [2, 16]

Spreading factor, specified as an integer in the range [2, 16].

Tunable: Yes

Dependencies

To enable this property, set the `TransmissionFormat` property to `"SS-TC-LM"` and the `IsCustomWaveform` property to `true`.

Data Types: `double`

`ScramblingPolynomial` — Scrambling polynomial
16-bit zero vector (default) | 16-bit vector of binary values | numeric vector

Scrambling polynomial, specified as one of these options.

- 16-bit vector of binary values from the most significant bit (MSB), z^{16}, to least significant bit (LSB), z^1. Each element of this vector corresponds to the coefficient of z and its exponent, specified from MSB to LSB. For details on the binary representation, see ETSI EN 301 545-2 Section 7.3.7.1.5.
- Numeric vector containing the exponents of z for nonzero terms of the polynomial in descending order.

The scrambling polynomial determines the shift register feedback connection to generate the spreading sequence.

The coefficient of z^0 is always 1.

The default value of this scrambling polynomial indicates the default scrambling sequence provided in the standard. When you set the `TransmissionFormat` property to `"SS-TC-LM"` and the `IsCustomWaveform` property to `false`, all of the reference waveforms use this default scrambling sequence.

Tunable: Yes

Dependencies

To enable this property, set the `TransmissionFormat` property to `"SS-TC-LM"` and the `IsCustomWaveform` property to `true`.

Data Types: `double` | `logical`

`ScramblingInitialConditions` — Scrambling initial conditions
`[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]` (default) | `1` | 16-bit vector of binary values

Scrambling initial conditions of the shift register, specified as one of these options.

- `1` — Use this option to set the initial condition of each cell of the shift register to this value.
- 16-bit vector of binary values from the MSB (z^{16}) to LSB (z^1) — Use this option to set the initial condition of each cell of the shift register to the corresponding element in this vector.

For this System object to generate a nonzero sequence, you must specify at least one nonzero element in this vector.

Tunable: Yes

Dependencies

To enable this property, set the `TransmissionFormat` property to `"SS-TC-LM"` and the `ScramblingPolynomial` property to a value other than the default value.

Data Types: `double` | `logical`

`FramePDULength` — Frame PDU length
`48` (default) | positive integer

This property is read-only.

Frame PDU length, returned as a positive integer.

The frame PDU length indicates the length in bits of the input data to this System object. This length is calculated by subtracting the length of the CRC sequence from the payload length in bits.

Data Types: `double`

Usage

Syntax

`burst = rcs2WaveGen(pdu)`

Description

`burst = rcs2WaveGen(pdu)` generates a DVB-RCS2-based burst symbols for the corresponding input binary sequence.

Input Arguments

pdu — Frame PDU
binary-valued column vector

Frame PDU, specified as a binary-valued column vector.

Data Types: `double` | `logical`

Output Arguments

`burst` — DVB-RCS2-based burst samples
column vector

DVB-RCS2-based burst samples, returned as a column vector.

The System object outputs these burst symbols (including the guard symbols) post modulation and pulse shaping.

Data Types: `double`

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 `dvbrcs2WaveformGenerator`

info	Characteristic information about object

Common to All System Objects

step	Run System object algorithm

release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate Reference DVB-RCS2 Waveform

Generate a reference DVB-RCS2 time-domain waveform with SS-TC-LM format.

Create and then set the properties of a DVB-RCS2 waveform generator System object™.

```
wg = dvbrcs2WaveformGenerator;
wg.TransmissionFormat = "SS-TC-LM";
wg.ContentType = "logon";
wg.WaveformID = 10;
wg.SamplesPerSymbol = 6;
```

Display the properties of the waveform generator.

```
disp(wg)
```

```
  dvbrcs2WaveformGenerator with properties:

      TransmissionFormat: "SS-TC-LM"
             ContentType: "logon"
        IsCustomWaveform: false
              WaveformID: 10
     PreBurstGuardLength: 0
    PostBurstGuardLength: 0
     FilterSpanInSymbols: 10
        SamplesPerSymbol: 6

   Read-only:
          FramePDULength: 1344
```

Generate a frame PDU.

```
framePDU = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2-based burst samples.

```
txWaveform = wg(framePDU);
```

Generate Custom DVB-RCS2 Waveform

Generate a custom DVB-RCS2 time-domain waveform having TC-LM format.

Create and then set the properties of the DVB-RCS2 waveform generator System object™.

```
wg = dvbrcs2WaveformGenerator;
wg.IsCustomWaveform = true;
wg.ContentType = "control";
wg.MappingScheme = "QPSK";
wg.CodeRate = "2/3";
wg.PreambleLength = 10;
wg.PostambleLength = 8;
wg.PermutationParameters = [13 4 2 1 2];
wg.UniqueWord = "FFFFFFFFF";
```

Display the properties of the waveform generator.

```
disp(wg)
```

```
  dvbrcs2WaveformGenerator with properties:

       TransmissionFormat: "TC-LM"
              ContentType: "control"
         IsCustomWaveform: true
      PreBurstGuardLength: 0
     PostBurstGuardLength: 0
      FilterSpanInSymbols: 10
         SamplesPerSymbol: 4
     PayloadLengthInBytes: 10

   Coding and Modulation:
            MappingScheme: "QPSK"
                 CodeRate: "2/3"
    PermutationParameters: [13 4 2 1 2]

   Unique Word:
           PreambleLength: 10
          PostambleLength: 8
              PilotPeriod: 0
         PilotBlockLength: 1
               UniqueWord: "FFFFFFFFF"

   Read-only:
           FramePDULength: 64
```

Generate a frame PDU.

```
framePDU = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2-based burst samples.

```
txWaveform = wg(framePDU);
```

Generate Multiple Content Type DVB-RCS2 Bursts

Generate multiple `ContentType` DVB-RCS2 bursts.

Set the `ContentType` of the DVB-RCS2 waveform generator System Object™ as `logon`.

```
wg = dvbrcs2WaveformGenerator;
wg.ContentType = "logon";
```

Generate a frame PDU.

```
framePDU1 = randi([0 1],wg.FramePDULength,1);
```

Generate the DVB-RCS2 logon burst samples.

```
txWaveform1 = wg(framePDU1);
```

Now, generate the DVB-RCS2 traffic burst samples.

```
% ContentType property is tunable
wg.ContentType = "traffic";
framePDU2 = randi([0 1],wg.FramePDULength,1);
txWaveform2 = wg(framePDU2);
```

Version History

Introduced in R2021b

References

[1] ETSI Standard EN 301 545-2 V1.2.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Interactive Satellite Systems (DVB-RCS2)*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`dvbrcs2TurboEncode` | `dvbrcs2BitRecover`

Objects
`dvbrcs2RecoveryConfig`

dvbs2WaveformGenerator

Generate DVB-S2 waveform

Description

The `dvbs2WaveformGenerator` System object generates a Digital Video Broadcasting Satellite Second Generation (DVB-S2) time-domain waveform consisting of a single or multiple physical layer frames. The object implements the waveform generation aspects of ETSI EN 302 307-1 V1.4.1 (2014-11) [1].

To generate a DVB-S2 waveform:

1. Create the `dvbs2WaveformGenerator` 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

```
s2waveGen = dvbs2WaveformGenerator
s2waveGen = dvbs2WaveformGenerator(Name,Value)
```

Description

`s2waveGen = dvbs2WaveformGenerator` creates a default DVB-S2 waveform generator System object.

`s2waveGen = dvbs2WaveformGenerator(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `dvbs2WaveformGenerator('NumInputStreams',4,'UPL',100)` specifies four input streams, each with a user packet length of 100 bits.

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.

`StreamFormat` — Input stream format
`"TS"` (default) | `"GS"`

Input stream format, specified as one of these values.

- `"TS"` — For transport stream format
- `"GS"` — For generic stream format

Data Types: `char` | `string`

`NumInputStreams` — Number of input streams
`1` (default) | integer in the range [1, 256]

Number of input streams, specified as an integer in the range [1, 256].

Data Types: `double`

`UPL` — User packet length
`0` (default) | nonnegative integer | vector of nonnegative integers

User packet (UP) length in bits, specified as one of these options.

- Nonnegative integer — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be equal to the integer value of the `UPL` property.
- Vector of nonnegative integers — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Note When you specify `UPL` as a multi-input stream, all UPs must be either packetized or in a continuous stream. Mixing stream types is not supported.

The maximum value of `UPL` as an integer scalar or an integer element in the row vector must be less than or equal to the corresponding `DFL` property value.

For a generic continuous stream, set `UPL` to `0`.

Dependencies

To enable this property, set the `StreamFormat` property to `"GS"`. If you set the `StreamFormat` property to `"TS"`, the UPL is fixed to 1504 bits.

Data Types: `double`

`FECFrame` — FEC frame format
`"normal"` (default) | `"short"`

Forward error correction (FEC) frame format, specified as one of these two options.

- `"normal"` — Sets the low density parity-check (LDPC) codeword length to 64,800 bits
- `"short"` — Sets the LDPC codeword length to 16,200 bits

Tunable: Yes

Data Types: `char` | `string`

`MODCOD` — Modulation scheme and FEC rate
`1` (default) | integer in the range [1, 28] | vector of integers in the range [1, 28]

Modulation scheme and FEC rate for input transmission, specified as one of these options, as defined in ETSI EN 302 307-1 Section 5.5.2.2 Table 12 [1].

- Integer in the range [1, 28] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, each stream has the same modulation scheme and coding rate.
- Vector of integers in the range [1, 28] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, each input stream has a modulation scheme and coding rate equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Note `MODCOD` values `11`, `17`, `23`, and `28` are not valid when you set the `FECFrame` property to `"short"` (as specified in ETSI EN 302 307-1 Section 5.3 Table 5b [1]).

Tunable: Yes

Data Types: `double`

`DFL` — Data field length

`15,928` (default) | integer in the range [1, (K_{BCH}−80)] | vector of integers in the range [1, (K_{BCH}−80)]

Data field (DF) length in bits, specified as one of these options. K_{BCH} is the uncoded BCH block length, as specified in ETSI EN 302 307-1 Section 5.3 Table 5a and 5b [1].

- Integer in the range [1, (K_{BCH}−80)] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the length of the DF in baseband frame of each stream is the same value.
- Vector of integers in the range [1, (K_{BCH}−80)] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the length of the data field in the baseband frame of each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Tunable: Yes

Data Types: `double`

`ScalingMethod` — Constellation amplitude scaling method

`"outer radius as 1"` (default) | `"unit average power"`

Constellation amplitude scaling method, specified as `"outer radius as 1"` or `"unit average power"`.

Dependencies

To enable this property, set the `MODCOD` property to a value in the range [18, 28], which indicates only 16APSK and 32APSK modulation schemes.

Data Types: `char` | `string`

`HasPilots` — Pilot block indication

`0` or `false` (default) | `1` or `true` | vector of `logical` values

Pilot block indication, specified as a logical value of `0` (`false`), `1` (`true`), or a vector of `logical` values. Set this value to `1` (`true`) to indicate pilots are inserted in the physical layer frame.

If you set the NumInputStreams property to a value greater than 1, you can configure pilots for each stream by specifying this property as a vector. The length of this vector must be equal to NumInputStreams.

Tunable: Yes

Data Types: logical

RolloffFactor — Transmit filter roll-off factor
0.35 (default) | 0.25 | 0.2

Transmit filter roll-off factor for baseband pulse shaping, specified as 0.35, 0.25, or 0.2.

Data Types: double

FilterSpanInSymbols — Filter span in symbols
10 (default) | positive integer

Filter span in symbols, specified as a positive integer.

The ideal impulse response of the raised cosine filter is truncated to a length that spans the number of symbols specified in this property.

Data Types: double

SamplesPerSymbol — Number of samples per symbol
4 (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Data Types: double

ISSYI — Input stream synchronization indicator
0 or false (default) | 1 or true

Input stream synchronization (ISSY) indicator, specified as a logical value of 0 (false) or 1 (true). To indicate that input stream synchronization is used, set this value to 1 (true).

Dependencies

To enable this property, set the NumInputStreams property to a value greater than 1 and set the UPL property to a nonzero value.

Data Types: logical

ISCRFormat — Input stream clock reference format
"short" (default) | "long"

Input stream clock reference format, specified as one of these options.

- "short" — Indicates the length of ISSY as 2 bytes
- "long" — Indicates the length of ISSY as 3 bytes

When you set the StreamFormat property to "GS", NumInputStreams property to a value greater than 1, UPL property to a nonzero value, and ISSYI to 1 (true), only the "short" option of this ISCRFormat property is applicable.

Dependencies

To enable this property, set the `StreamFormat` property to `"TS"`, the `NumInputStreams` property to a value greater than 1, and the `ISSYI` property to `1` (`true`).

Data Types: `char` | `string`

`MinNumPackets` — Minimum number of packets required to create DF
integer in the range [1, 58,112] | row vector of integers

This property is read-only.

Minimum number of packets required to create a DF, returned as one of these options.

- Integer in the range [1, 58,112] — This option applies with single-input streams only.
- Row vector of integers in the range [1, 58,112] — This option applies with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the minimum number of packets required for each stream is equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

The value of `MinNumPackets` is computed based of values of `DFL` and `UPL` properties.

Dependencies

To enable this property, set the `UPL` property to a nonzero value.

Data Types: `double`

Usage

Syntax

`txWaveform = s2waveGen(data)`

Description

`txWaveform = s2waveGen(data)` generates a DVB-S2 time-domain waveform from the input information bits.

Input Arguments

`data` — Input information bits
`[ ]` | binary-valued column vector | cell array of binary-valued column vectors

Input information bits, specified as one of these options. Each element of the column vector or cell array can be of data type `double`, `int8`, or `logical`.

- Binary-valued column vector — Use this option with single-input streams.

  To generate a dummy physical layer (PL) frame, specify `data` as an empty column vector.
- Cell array of binary-valued column vectors — Use this option with multi-input streams. Each element of the array represents the corresponding input stream. The length of the cell array must be equal to the value of the `NumInputStreams` property.

To generate a dummy frame for a particular input stream, specify the corresponding element of the `data` cell array as an empty column vector.

Input `data`, either as a single-input or multi-input stream, must be input in one of these forms.

- Packetized stream — The number of packets in each stream must be an integer multiple of the `MinNumPackets` property.

  For a packetized stream, an 8-bit sync field must be included at the beginning of each packet. The combined length of a packet and its sync bits must be equal to the corresponding value of the `UPL` property.
- Continuous stream — The number of bits for each stream must be an integer multiple of the `DFL` property.

Note When you set the `StreamFormat` property to `"TS"`, the sync byte is fixed as 47 hex.

Data Types: `double` | `int8` | `logical` | `cell`

Output Arguments

`txWaveform` — Generated time-domain DVB-S2 waveform
column vector

Generated time-domain DVB-S2 waveform, returned as a column vector. The waveform is generated in the form of complex in-phase quadrature (IQ) samples and can consist of a single physical layer frame or multiple physical layer frames.

When you set the `NumInputStreams` property to a value greater than 1, the data fields generated from different input streams are merged using the round-robin technique.

Data Types: `double`

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 `dvbs2WaveformGenerator`

info	Characteristic information about object
flushFilter	Flush transmit filter

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate DVB-S2 Waveform for Single-Input Stream

Generate a Digital Video Broadcasting Satellite Second Generation (DVB-S2) time-domain waveform for a single-input transport stream (TS) with a single physical layer (PL) frame per stream. Visualize the waveform using constellation plots and signal spectrum.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 1;
```

Create a DVB-S2 System object. Specify the modulation scheme and FEC rate (MODCOD) and data field length (DFL).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.MODCOD = 21;              % 16APSK 5/6
s2WaveGen.DFL = 39690;
s2WaveGen.HasPilots = true;         % Pilot insertion indication
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

           StreamFormat: "TS"
        NumInputStreams: 1
               FECFrame: "normal"
                 MODCOD: 21
                    DFL: 39690
          ScalingMethod: "outer radius as 1"
              HasPilots: 1
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4

  Show all properties
```

Create a bit vector of information bits, `data`, of concatenated TS user packets.

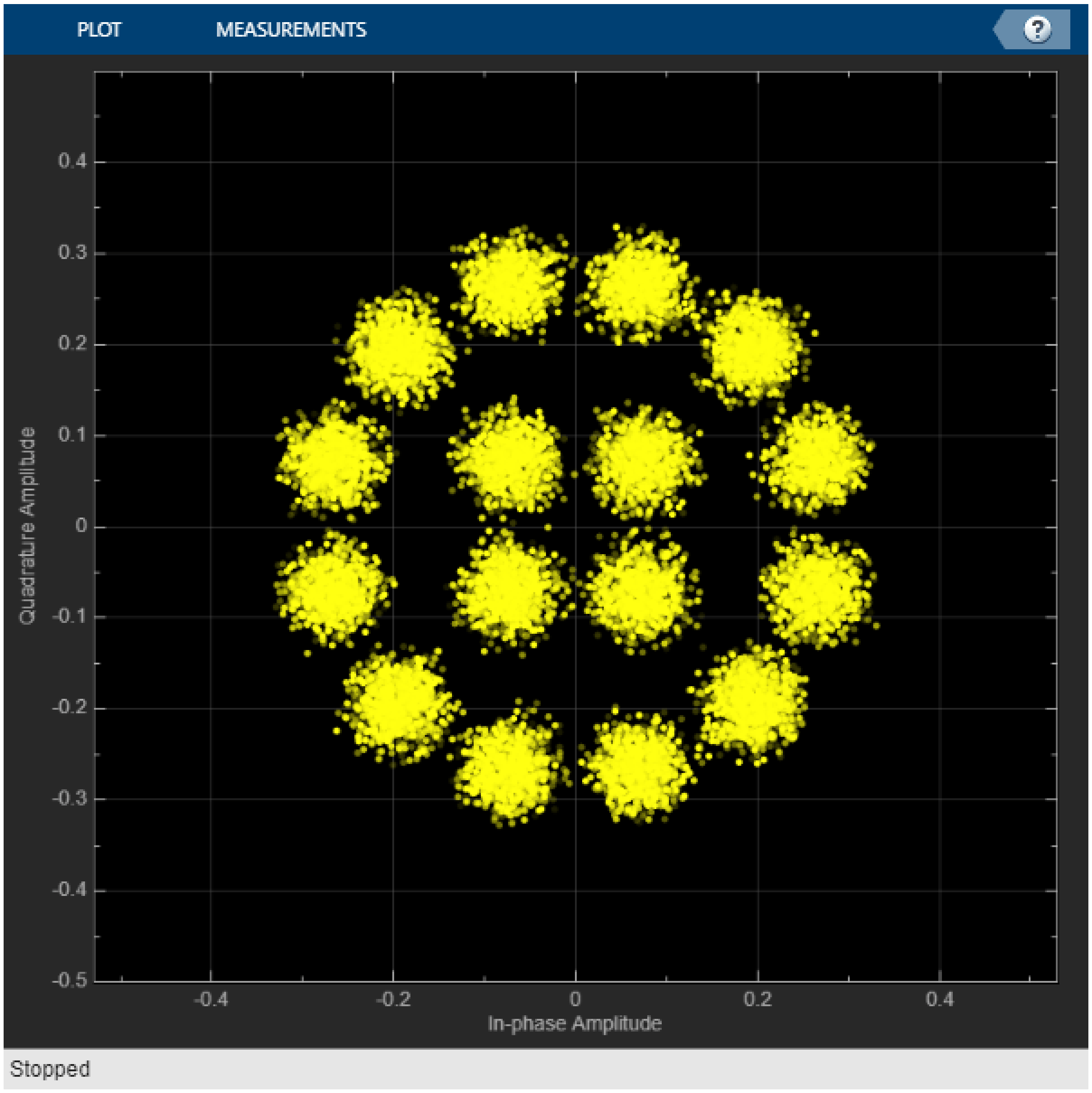
```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
numPkts = s2WaveGen.MinNumPackets*numFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate a DVB-S2 time-domain waveform using the information bits, `data`.

```
txWaveform = s2WaveGen(data);
```

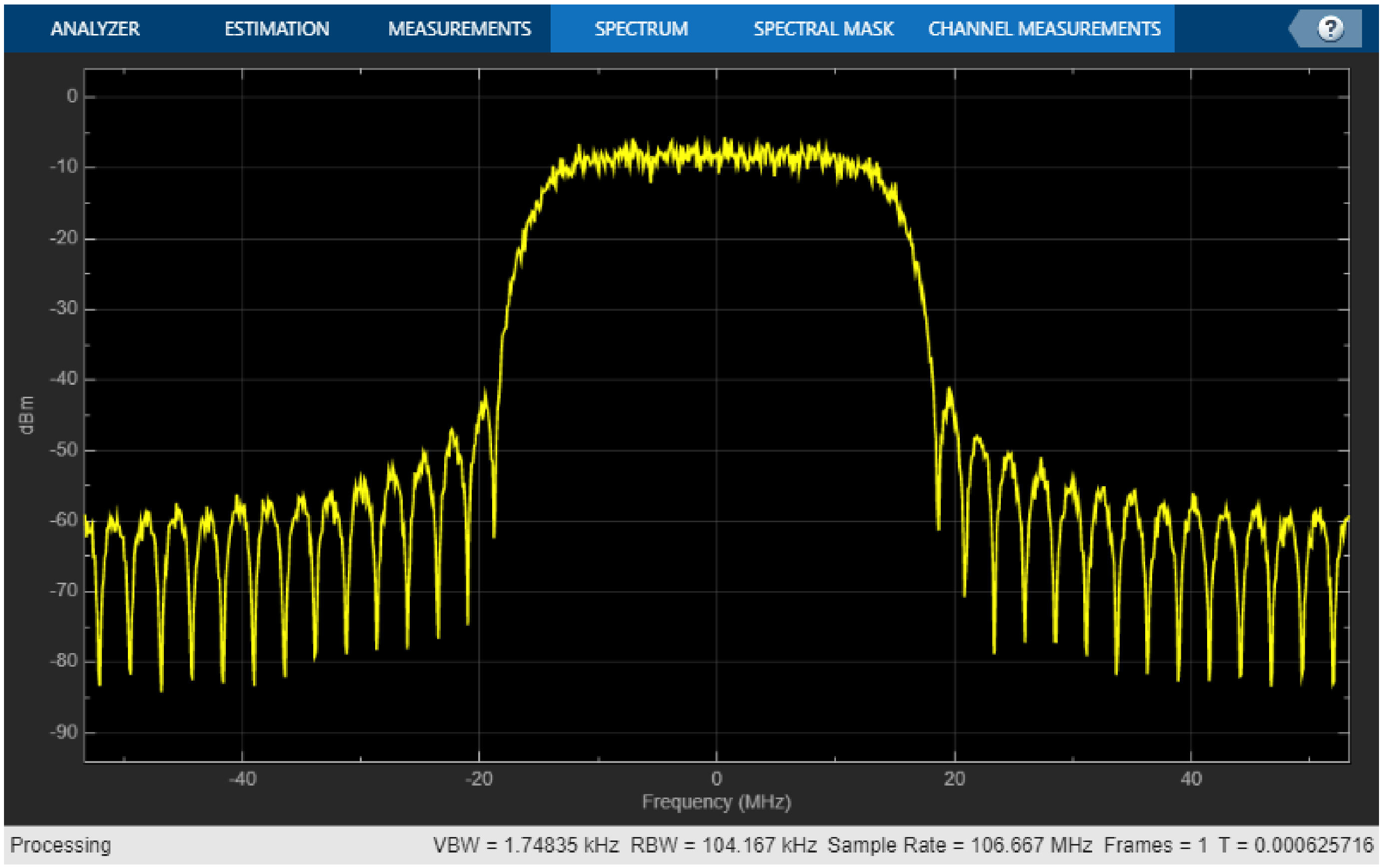
Visualize the constellation plot for the generated DVB-S2 time-domain waveform by creating a constellation diagram System object.

```
sps = s2WaveGen.SamplesPerSymbol;
constel = comm.ConstellationDiagram('ColorFading',true, ...
    'ShowTrajectory',0, ...
    'SamplesPerSymbol',sps, ...
    'ShowReferenceConstellation',false, ...
    'XLimits',[-0.5 0.5], 'YLimits',[-0.5 0.5]);
plHeaderLen = 90*sps;           % PL header length
constel(txWaveform(plHeaderLen+1:end));
release(constel);
```



Display the frequency spectrum of the generated DVB-S2 time-domain waveform by creating a spectrum analyzer System object.

```
BW = 36e6;                 % Typical satellite channel bandwidth
Fsym = BW/(1+s2WaveGen.RolloffFactor);
Fsamp = Fsym*sps;
scope = spectrumAnalyzer('SampleRate',Fsamp);
scope(txWaveform)
```



VBW = 1.74835 kHz RBW = 104.167 kHz Sample Rate = 106.667 MHz Frames = 1 T = 0.000625716

Generate DVB-S2 Waveform for Multi-Input Stream

Generate a Digital Video Broadcasting Satellite Second Generation (DVB-S2) time-domain waveform for a multi-input generic stream (GS) with multiple physical layer (PL) frames per stream.

This example requires MAT-files with LDPC parity matrices. If they are not available on the path, execute the following commands to download and unzip the MAT-files.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip', 'file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 3;
```

Create a DVB-S2 System object with variable coding and modulation configuration for a multi-input GS. Specify the modulation scheme and FEC rate (MODCOD) and data field length (DFL).

```
s2WaveGen = dvbs2WaveformGenerator;
s2WaveGen.StreamFormat = "GS";
s2WaveGen.NumInputStreams = 2;
s2WaveGen.MODCOD = [6 24];          % QPSK 2/3 and 32APSK 3/4
s2WaveGen.DFL = [42960 48328];
s2WaveGen.HasPilots = true;
s2WaveGen.SamplesPerSymbol = 10;
disp(s2WaveGen)
```

```
  dvbs2WaveformGenerator with properties:

           StreamFormat: "GS"
        NumInputStreams: 2
                    UPL: 0
               FECFrame: "normal"
                 MODCOD: [6 24]
                    DFL: [42960 48328]
          ScalingMethod: "outer radius as 1"
              HasPilots: 1
          RolloffFactor: 0.3500
      FilterSpanInSymbols: 10
       SamplesPerSymbol: 10
```

Create a bit vector of input information bits for each GS user packet.

```
data = cell(s2WaveGen.NumInputStreams,1);
   for i = 1:s2WaveGen.NumInputStreams
           data{i} = randi([0 1],s2WaveGen.DFL(i)*numFrames,1,'int8');
   end
```

Generate the DVB-S2 time-domain waveform with the input information bits.

```
txWaveform = s2WaveGen(data);
```

Generate DVB-S2 Dummy PL Frame

Generate a DVB-S2 dummy PL frame for a single-input transport stream.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Create a default DVB-S2 System object.

```
s2WaveGen = dvbs2WaveformGenerator;
```

Generate a PL dummy frame.

```
data = zeros(0,1);
```

Generate a DVB-S2 waveform.

```
txWaveform = s2WaveGen(data);
```

Version History

Introduced in R2021a

References

[1] ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For all properties that support string and cell array input, code generation is only supported with cell array input.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Functions
`dvbs2BitRecover`

Objects
`dvbs2xWaveformGenerator`

dvbs2xWaveformGenerator

Generate DVB-S2X waveform

Description

The `dvbs2xWaveformGenerator` System object generates a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) time-domain waveform consisting of a single or multiple physical layer (PL) frames. The object implements the waveform generation aspects of ETSI EN 302 307-2 V1.1.1 (2015-11) [2].

To generate a DVB-S2X waveform:

1. Create the `dvbs2xWaveformGenerator` 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

```
s2xWaveGen = dvbs2xWaveformGenerator
s2xWaveGen = dvbs2xWaveformGenerator(Name,Value)
```

Description

`s2xWaveGen = dvbs2xWaveformGenerator` creates a default DVB-S2X waveform generator System object.

`s2xWaveGen = dvbs2xWaveformGenerator(Name,Value)` sets properties using one or more name-value pairs. Enclose each property name in quotes. For example, `dvbs2xWaveformGenerator('NumInputStreams',4,'UPL',100)` specifies four input streams, each with a user packet length of 100 bits.

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.

`StreamFormat` — Input stream format
`"TS"` (default) | `"GS"`

Input stream format, specified as one of these values.

- `"TS"` — For transport stream format
- `"GS"` — For generic stream format

Data Types: `char` | `string`

`HasTimeSlicing` — Time slicing indicator
`0` or `false` (default) | `1` or `true`

Time slicing indicator, specified as a logical value of `0` (`false`) or `1` (`true`). To indicate that time slicing transmission format is used, set this value to `1` (`true`).

If you set this property to `1` (`true`), you can set the `NumInputStreams` property to a maximum value of `8`.

Data Types: `logical`

`NumInputStreams` — Number of input streams
`1` (default) | integer in the range [1, 256]

Number of input streams, specified as an integer in the range [1, 256].

When you set the `HasTImeSlicing` property to `true`, `NumInputStreams` property can be specified to a maximum value of `8`.

Data Types: `double`

`UPL` — User packet length
`0` (default) | nonnegative integer | vector of nonnegative integers

User packet (UP) length in bits, specified as one of these options.

- Nonnegative integer — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be equal to the integer value of the `UPL` property.
- Vector of nonnegative integers — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the UP in each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Note When you specify `UPL` as a multi-input stream, all UPs must be either packetized or in a continuous stream. Mixing stream types is not supported.

The maximum value of `UPL` as an integer scalar or an integer element in the row vector must be less than or equal to the corresponding `DFL` property value.

For a generic continuous stream, set `UPL` to `0`.

Dependencies

To enable this property, set the `StreamFormat` property to `"GS"`. If you set the `StreamFormat` property to `"TS"`, the UPL is fixed to 1504 bits.

Data Types: `double`

`PLSDecimalCode` — PL signalling code information
`132` (default) | integer in the range [4, 249] | vector of integers in the range [4, 249]

PL signalling code information, in decimal format, specified as one of these options (as described in ETSI EN 302 307-1 Section 5.5.2.2 [1] and ETSI EN 302 307-2 Section 5.5.2.2 Table 17a [2]).

- Integer in the range [4, 249] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, each stream has the same modulation scheme and coding rate.
- Vector of integers in the range [4, 249] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, each stream has a modulation scheme and coding rate equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

All odd integer values are considered to include pilots in PL frames.

Note Few `PLSDecimalCode` values are invalid in this specified value range. Invalid values include {46, 47, 70, 71, 94, 95, 114, 128, 130, 176, 177, 188, 189, 192, 193, 196, and 197}.

To calculate the `PLSDecimalCode` property value for a DVB-S2X system configuration, use this formula.

`MODCOD`*4 + (0 - for normal `FECFrame`/1 - for short `FECFrame`)*2 + (0 - if `HasPilots` property value is `false`/1 - if `HasPilots` property value is `true`)

For example, if `MODCOD` = 18 (16APSK 2/3) with short FEC frame and pilots disabled, the value of `PLSDecimalCode` calculated by using this formula is:

`PLSDecimalCode` = 18*4 + 1*2 + 0 = 74

Note For very low signal to noise ratio (VL-SNR) frames, you must set the `PLSDecimalCode` property to either `129` or `131`, which indicates the VL-SNR set 1 or 2, respectively.

VL-SNR frames must not be combined with regular frames.

Tunable: Yes

Data Types: `double`

`CanonicalMODCODName` — Canonical modulation scheme and code rate name
`"QPSK 2/9"` (default) | character vector | string scalar | cell array | string array

Canonical modulation scheme and code rate name for VL-SNR frame transmission, specified as one of these options (as described in ETSI EN 302 307-2 Section 5.5.2.2 Table 18a [2]).

- Character vector or string scalar — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, each stream has the same modulation scheme and coding rate.
- Cell array or string array — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, each input stream has a modulation scheme and coding rate equal to the corresponding value in this array. The length of this array must be equal to `NumInputStreams`.

Valid `CanonicalMODCODName` values include these options.

- `"QPSK 2/9"`, `"BPSK 1/5"`, `"BPSK 11/45"`, `"BPSK-S 1/5"`, `"BPSK-S 11/45"`, and `"BPSK 1/3"` — Applicable for VL-SNR set 1
- `"BPSK 1/5"`, `"BPSK 4/15"`, and `"BPSK 1/3"` — Applicable for VL-SNR set 2

Tunable: Yes

Dependencies

To enable this property, set the `PLSDecimalCode` property to either `129` (for VL-SNR set 1) or `131` (for VL-SNR set 2). This property applies for only VL-SNR frame transmissions.

Data Types: `char` | `string`

`DFL` — Data field length

`18,448` (default) | integer in the range [1, (K_{BCH}−80)] | vector of integers in the range [1, (K_{BCH}−80)]

Data field (DF) length in bits, specified as one of these options. K_{BCH} is the uncoded BCH block length, as specified in ETSI EN 302 307-1 Section 5.3 Table 5a and 5b [1].

- Integer in the range [1, (K_{BCH}−80)] — Use this option with single-input and multi-input streams. If you set the `NumInputStreams` property to a value greater than 1, the length of the DF in baseband frame of each stream is the same value.
- Vector of integers in the range [1, (K_{BCH}−80)] — Use this option with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the length of the data field in the baseband frame of each stream must be the size of the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

Tunable: Yes

Data Types: `double`

`ScalingMethod` — Constellation amplitude scaling method

`"outer radius as 1"` (default) | `"unit average power"`

Constellation amplitude scaling method, specified as `"outer radius as 1"` or `"unit average power"`.

Dependencies

To enable this property, set the `PLSDecimalCode` property to a value corresponding to APSK modulation, with the following as exception: {164, 165, 158, 159, 206, 207, 212, and 213}. The other exceptions are QPSK and 8 PSK values: {4 to 69, inclusive; 129; 131; 132 to 137, inclusive; 142 to 147, inclusive; 216 to 235, inclusive}.

Data Types: `char` | `string`

`PLScramblingIndex` — PL scrambling sequence index

integer in the range [0, 7] | vector of integers in the range [0, 7]

PL scrambling sequence index, specified as one of these options (as described in ETSI EN 302 307-2 Section 5.5.4 Table 19e [2]).

- Integer in the range [0, 7] — Use this option with single-input and multi-input streams.

  If you set the `NumInputStreams` property to a value greater than 1, each stream has the same value of PL scrambling index.

- Vector of integers in the range [0, 7] — Use this option when you set the `HasTimeSlicing` property to `true` for multi-input streams.

  If you set the `NumInputStreams` property to a value greater than 1, the PL scrambling index value of each stream must be equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

To generate the PL scrambling sequence, the actual index used is `PLScramblingIndex`*10949.

Data Types: `double`

`RolloffFactor` — Transmit filter roll-off factor
`0.35` (default) | `0.05` | `0.1` | `0.15` | `0.2` | `0.25`

Transmit filter roll-off factor for baseband pulse shaping, specified as `0.35`, `0.05`, `0.1`, `0.15`, `0.2`, or `0.25`.

Data Types: `double`

`FilterSpanInSymbols` — Filter span in symbols
`10` (default) | positive integer

Filter span in symbols, specified as a positive integer.

The ideal impulse response of the raised cosine filter is truncated to a length that spans the number of symbols specified in this property.

Data Types: `double`

`SamplesPerSymbol` — Number of samples per symbol
`4` (default) | positive integer

Number of samples per symbol, specified as a positive integer.

Data Types: `double`

`ISSYI` — Input stream synchronization indicator
`0` or `false` (default) | `1` or `true`

Input stream synchronization (ISSY) indicator, specified as a logical value of `0` (`false`) or `1` (`true`). To indicate that input stream synchronization is used, set this value to `1` (`true`).

Dependencies

To enable this property, set the `NumInputStreams` property to a value greater than 1 and set the `UPL` property to a nonzero value.

Data Types: `logical`

`ISCRFormat` — Input stream clock reference format
`"short"` (default) | `"long"`

Input stream clock reference format, specified as one of these options.

- `"short"` — Indicates the length of ISSY as 2 bytes
- `"long"` — Indicates the length of ISSY as 3 bytes

When you set the `StreamFormat` property to `"GS"`, `NumInputStreams` property to a value greater than 1, `UPL` property to a nonzero value, and `ISSYI` to `1` (`true`), only the `"short"` option of this `ISCRFormat` property is applicable.

Dependencies

To enable this property, set the `StreamFormat` property to `"TS"`, the `NumInputStreams` property to a value greater than 1, and the `ISSYI` property to `1` (`true`).

Data Types: `char` | `string`

`MinNumPackets` — Minimum number of packets required to create DF
integer in the range [1, 58,112] | row vector of integers

This property is read-only.

Minimum number of packets required to create a DF, returned as one of these options.

- Integer in the range [1, 58,112] — This option applies with single-input streams only.
- Row vector of integers in the range [1, 58,112] — This option applies with multi-input streams only. If you set the `NumInputStreams` property to a value greater than 1, the minimum number of packets required for each stream is equal to the corresponding element in this vector. The length of this vector must be equal to `NumInputStreams`.

The value of `MinNumPackets` is computed based of values of `DFL` and `UPL` properties.

Dependencies

To enable this property, set the `UPL` property to a nonzero value.

Data Types: `double`

Usage

Syntax

`txWaveform = s2xWaveGen(data)`

Description

`txWaveform = s2xWaveGen(data)` generates a DVB-S2X time-domain waveform from the input information bits.

Input Arguments

`data` — Input information bits
[ ] | binary-valued column vector | cell array of binary-valued column vectors

Input information bits, specified as one of these options. Each element of the column vector or cell array can be of the data type `double`, `int8`, or `logical`.

- Binary-valued column vector – Use this option with single-input stream.

  To generate a dummy physical layer (PL) frame, specify `data` as an empty column vector.

- Cell array of binary-valued column vectors – Use this option with multi-input streams. Each element of the array represents the corresponding input stream. The length of the cell array must be equal to the value of the `NumInputStreams` property.

  To generate a dummy frame for a particular input stream, specify the corresponding element of the `data` cell array as an empty column vector.

`data`, either single stream or multi-stream, can be input in one of these forms.

- Packetized stream – The number of packets in each stream must be an integer multiple of the `MinNumPackets` property.

  For a packetized stream, an 8-bit sync field must be included at the beginning of each packet. The combined length of a packet and its sync bits must be equal to the corresponding value of the `UPL` property.
- Continuous Stream – The number of bits for each stream must be an integer multiple of the `DFL` property.

Note When you set the `StreamFormat` property to `"TS"`, the sync byte is fixed as 47 hex.

Data Types: `double` | `int8` | `logical` | `cell`

Output Arguments

`txWaveform` — Generated time-domain DVB-S2X waveform
column vector

Generated time-domain DVB-S2X waveform, returned as a column vector. The waveform is generated in the form of complex in-phase quadrature (IQ) samples and can consist of a single physical layer frame or multiple physical layer frames.

When you set the `NumInputStreams` property to a value greater than 1, the data fields generated from different input streams are merged using the round-robin technique.

Data Types: `double`

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 `dvbs2xWaveformGenerator`

info	Characteristic information about object
flushFilter	Flush transmit filter

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics

clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate DVB-S2X Waveform for Single-Input Stream

Generate a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) time-domain waveform for a single-input transport stream (TS) with a single physical layer (PL) frame per stream.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 1;
```

Create a DVB-S2X System object with pilot-aided PL.

```
s2xWaveGen = dvbs2xWaveformGenerator;
s2xWaveGen.PLSDecimalCode = 133;      % QPSK 13/45
                                      % All odd PLSDecimalCode values are pilot aided
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

           StreamFormat: "TS"
         HasTimeSlicing: false
        NumInputStreams: 1
         PLSDecimalCode: 133
                    DFL: 18448
       PLScramblingIndex: 0
          RolloffFactor: 0.3500
     FilterSpanInSymbols: 10
       SamplesPerSymbol: 4

  Show all properties
```

Create the bit vector of information bits, `data`, of concatenated TS user packets.

```
syncBits = [0 1 0 0 0 1 1 1]';    % Sync byte for TS packet is 47 Hex
pktLen = 1496;                    % UP length without sync bits is 1496
numPkts = s2xWaveGen.MinNumPackets*numFrames;
txRawPkts = randi([0 1],pktLen,numPkts);
txPkts = [repmat(syncBits,1,numPkts); txRawPkts];
data = txPkts(:);
```

Generate a DVB-S2X time-domain waveform using the information bits, `data`.

```
txWaveform = s2xWaveGen(data);
```

Generate DVB-S2X Waveform Consisting of VL-SNR Frame

Generate a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) time-domain waveform for a single-input generic stream (GS) with multiple physical layer (PL) frames per stream.

The DVB-S2X waveform generated in this example consists of a very low signal to noise ratio (VL-SNR) frame of set 2.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of PL frames per stream.

```
numFrames = 2;
```

Create a DVB-S2X System object and specify its properties.

```
s2xWaveGen = dvbs2xWaveformGenerator;
s2xWaveGen.StreamFormat = "GS";
s2xWaveGen.PLSDecimalCode = 131;    % VL-SNR set 2
s2xWaveGen.CanonicalMODCODName = "BPSK 1/3";
s2xWaveGen.DFL = 5080;
s2xWaveGen.SamplesPerSymbol = 6;
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

           StreamFormat: "GS"
         HasTimeSlicing: false
        NumInputStreams: 1
                    UPL: 0
         PLSDecimalCode: 131
    CanonicalMODCODName: "BPSK 1/3"
                    DFL: 5080
      PLScramblingIndex: 0
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 6
```

Create a bit vector of information bits for each stream.

```
data = randi([0 1],s2xWaveGen.DFL*numFrames,1,'int8');
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Get DVB-S2X Waveform Generator Information and Check Transmit Filter Delay

Get information from a `dvbs2xWaveformGenerator` System object by using the `info` function. Then retrieve the filter residual samples by using the `flushFilter` object function.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
addpath('s2xLDPCParityMatrices');
end
```

Specify the number of physical layer (PL) frames per stream.

```
numFrames = 2;
```

Create a Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) System object and specify its properties. Use time slicing technique and variable coding and modulation configuration mode.

```
s2xWaveGen = dvbs2xWaveformGenerator();
s2xWaveGen.HasTimeSlicing = true;
s2xWaveGen.NumInputStreams = 2;
s2xWaveGen.PLSDecimalCode = [135 145];   % QPSK 9/20 and 8PSK 25/36
s2xWaveGen.DFL = [18048 44656];
s2xWaveGen.PLScramblingIndex = [0 1];
disp(s2xWaveGen)
```

```
  dvbs2xWaveformGenerator with properties:

           StreamFormat: "TS"
         HasTimeSlicing: true
        NumInputStreams: 2
         PLSDecimalCode: [135 145]
                    DFL: [18048 44656]
      PLScramblingIndex: [0 1]
          RolloffFactor: 0.3500
    FilterSpanInSymbols: 10
       SamplesPerSymbol: 4
                  ISSYI: false

  Show all properties
```

Get the characteristic information about the DVB-S2X waveform generator.

```
info(s2xWaveGen)
```

```
ans = struct with fields:
             FECFrame: {'normal'  'normal'}
     ModulationScheme: {'QPSK'  '8PSK'}
    LDPCCodeIdentifier: {'9/20'  '25/36'}
```

Create the bit vector of input information bits, `data`, of concatenated TS user packets for each input stream.

```
syncBits = [0 1 0 0 0 1 1 1]';       % Sync byte for TS packet is 47 Hex
pktLen = 1496;                       % UP length without sync bits is 1496
data =  cell(1, s2xWaveGen.NumInputStreams);
for i = 1:s2xWaveGen.NumInputStreams
    numPkts = s2xWaveGen.MinNumPackets(i)*numFrames;
    txRawPkts = randi([0 1], pktLen, numPkts);
    txPkts = [repmat(syncBits, 1, numPkts); txRawPkts];
    data{i} = txPkts(:);
end
```

Generate a DVB-S2X time-domain waveform using the information bits.

```
txWaveform = s2xWaveGen(data);
```

Check the filter residual data samples that remain in the filter delay.

```
flushFilter(s2xWaveGen)
```

```
ans = 40×1 complex

  -0.2412 - 0.0143i
  -0.2619 - 0.0861i
  -0.2726 - 0.1337i
  -0.2511 - 0.1597i
  -0.1851 - 0.1680i
  -0.0780 - 0.1602i
   0.0448 - 0.1288i
   0.1598 - 0.0751i
   0.2482 - 0.0049i
   0.3026 + 0.0702i
      ⋮
```

Generate DVB-S2X Dummy VL-SNR Frame

Generate a DVB-S2X dummy VL-SNR frame for a single-input transport stream.

This example uses MAT-files with LDPC parity matrices. If the MAT-files are not available on the path, download and unzip the MAT-files by entering this code at the MATLAB command prompt.

```
if ~exist('dvbs2xLDPCParityMatrices.mat','file')
    if ~exist('s2xLDPCParityMatrices.zip','file')
        url = 'https://ssd.mathworks.com/supportfiles/spc/satcom/DVB/s2xLDPCParityMatrices.zip';
        websave('s2xLDPCParityMatrices.zip',url);
        unzip('s2xLDPCParityMatrices.zip');
    end
```

```
addpath('s2xLDPCParityMatrices');
end
```

Create a default DVB-S2X System object.

```
s2xWaveGen = dvbs2xWaveformGenerator;
```

Specify the PLS decimal code value to indicate VL-SNR frame, and set the DFL value.

```
s2xWaveGen.PLSDecimalCode = 129;  % VL-SNR set 1
s2xWaveGen.DFL = 14128;
```

Generate a PL dummy frame.

```
data = zeros(0,1);
```

Generate a DVB-S2X waveform.

```
txWaveform = s2xWaveGen(data);
```

Version History

Introduced in R2021a

References

[1] ETSI Standard EN 302 307-1 V1.4.1(2014-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.

[2] ETSI Standard EN 302 307-2 V1.1.1(2015-11). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: DVB-S2 Extensions (DVB-S2X)*.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- For all properties that support string and cell array input, code generation is only supported with cell array input.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects

`dvbs2WaveformGenerator`

Functions
`dvbs2BitRecover`

dsocPoissonChannel

Create a deep space optical communications Poisson channel

Description

The `dsocPoissonChannel` System object adds Poisson distributed noise to a signal through a deep space optical communications (DSOC) Poisson channel. The object uses the Poisson channel implementation described in section IV of the Interplanetary Network Progress report [1].

To add Poisson distributed noise to the input signal through a DSOC Poisson channel:

1. Create the `dsocPoissonChannel` 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

```
chan = dsocPoissonChannel
chan = dsocPoissonChannel(Name=Value)
```

Description

`chan = dsocPoissonChannel` creates a default DSOC Poisson channel System object.

`chan = dsocPoissonChannel(Name=Value)` sets properties using one or more optional name-value arguments. For example, `dsocPoissonChannel(NumSignalPhotons=6)` sets the average number of signal photons per pulsed slot to `6`.

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.

`NumSignalPhotons` — Average signal photons per pulsed slot
`2.5` (default) | nonnegative finite real scalar

Average signal photons per pulsed slot, specified as a nonnegative finite real scalar.

Data Types: `double`

`NumNoisePhotons` — Average background noise photons per slot
`0.2` (default) | nonnegative finite real scalar

Average background noise photons per slot, specified as nonnegative finite real scalar.

Data Types: `double`

`TimingOffset` — Normalized timing offset
`0` (default) | real number in the range [0, 1)

Normalized timing offset, specified as a real number in the range [0, 1)

Data Types: `double`

`RandomStream` — Source of random number stream
`"Global stream"` (default) | `"mt19937ar with seed"`

Source of the random number stream, specified as `"Global stream"` or `"mt19937ar with seed"` .

- `"Global stream"`— The object uses the current global random number stream for uniformly and normally distributed random number generation.
- `"mt19937ar with seed"`— The object uses the mt19937ar algorithm for uniformly and normally distributed random number generation. In this case, the `reset` object function reinitializes the random number stream to the `Seed` value.

Data Types: `string` | `char`

`Seed` — Initial seed of mt19937ar random number stream generator algorithm
`73` (default) | nonnegative integer

Initial seed of the mt19937ar random number stream generator algorithm, specified as a nonnegative integer. When you call the `reset` object function, it reinitializes the mt19937ar random number stream to the `Seed` value.

Data Types: `double`

Usage

Syntax

`y = chan(x)`

Description

`y = chan(x)` adds Poisson distributed noise to the input signal `x` through a DSOC Poisson channel, and returns the output signal `y`.

Input Arguments

`x` — Input signal
N-element column vector

Input signal, specified as an *N*-element column vector in which each value is a numeric `1` or `0`. *N* is the number of samples. The vector contains the modulated data, with `1` representing a laser pulse and `0` representing the absence of a laser pulse.

Data Types: `double`

Output Arguments

y — Output signal
N-by-1 vector

Output signal, returned as an *N*-element column vector. *N* is the number of samples. The vector contains the estimate of the photon count within each slot. `y` is of the same length as `x`.

Data Types: `double`

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)
```

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Estimate Photon Count per Slot at Receiver by Configuring DSOC Poisson Channel

To estimate photon count per slot at the receiver, configure a DSOC Poisson channel and pass the input signal through the channel.

Generate data.

```
rng("default")
symLen = 1000;
m = 6;
M = 2^m;                              % M-ary PPM
data = randi([0 M-1],symLen,1);
```

Create a default channel object.

```
chanObj = dsocPoissonChannel;
```

Modulate the data using the pulse position modulation (PPM) technique.

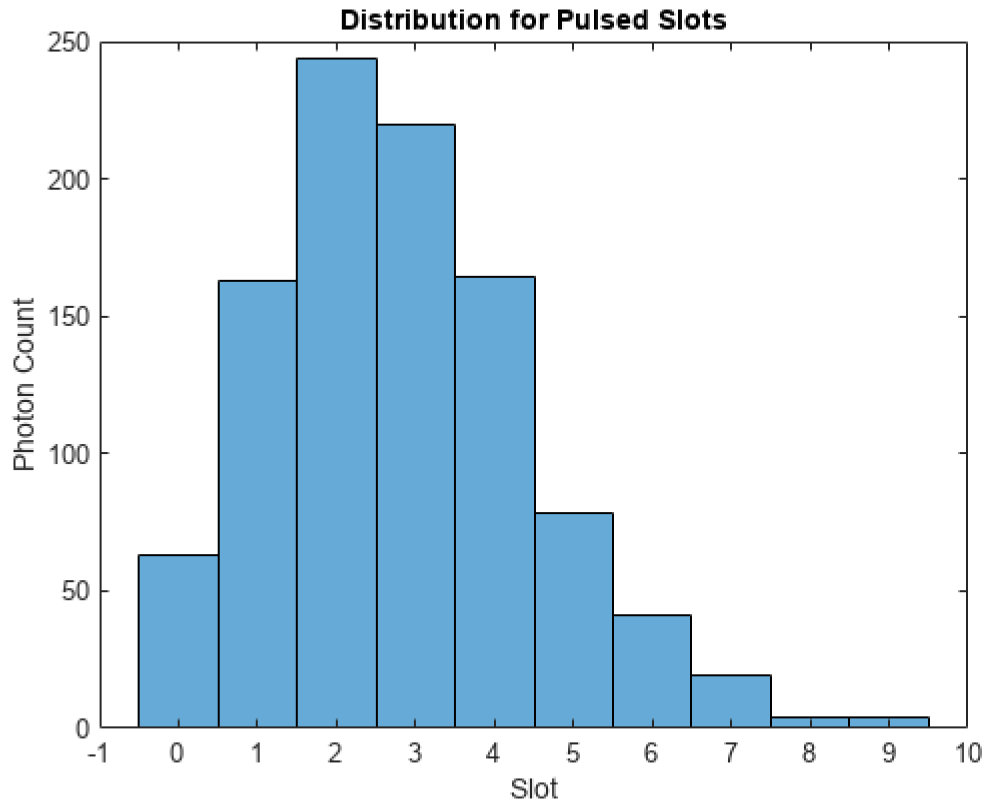
```
modOut = zeros(symLen*M,1);
mapIndex = (0:symLen-1)'*M + data + 1; % Mapping Index
modOut(mapIndex) = 1;
```

Pass the modulated data through a Poisson channel.

```
receivedData = chanObj(modOut);
```

Plot the distribution for pulsed slots.

```
histogram(receivedData(modOut==1))
xlabel("Slot")
ylabel("Photon Count")
title("Distribution for Pulsed Slots")
```



Add Timing Offset to Signal and Estimate Photon Counts per Slot at Receiver

Estimate photon counts per slot by configuring a DSOC Poisson channel and adding a timing offset to the signal.

Generate data.

```
rng("default")
symLen = 1000;
m = 6;
```

```
M = 2^m;                              % M-ary PPM
data = randi([0 M-1],symLen,1);
```

Create a DSOC Poisson channel object with these properties.

- Average signal photons per pulsed slot — `6`
- Average noise photons per slot — `0.01`
- Normalized timing offset — `0.11`
- Source of random number stream — `"mt19937ar with seed"`
- Initial seed of mt19937ar random number stream— `64`

```
chanObj = dsocPoissonChannel(NumSignalPhotons=6,NumNoisePhotons=0.01,TimingOffset=0.11, ...
                       RandomStream="mt19937ar with seed",Seed=64);
```

Modulate the data using PPM.

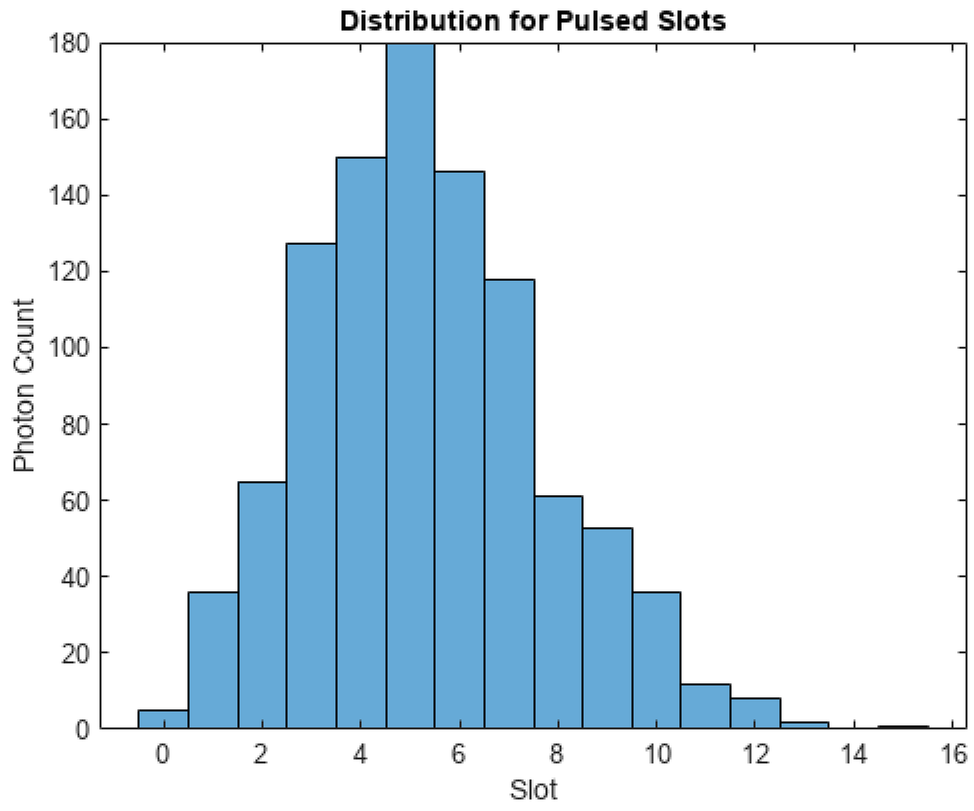
```
modOut = zeros(symLen*M,1);
mapIndex = (0:symLen-1)'*M + data + 1; % Mapping Index
modOut(mapIndex) = 1;
```

Pass the modulated data through a Poisson channel.

```
receivedData = chanObj(modOut);
```

Plot the distribution for pulsed slots.

```
histogram(receivedData(modOut==1))
xlabel("Slot")
ylabel("Photon Count")
title("Distribution for Pulsed Slots")
```



Version History

Introduced in R2023a

References

[1] Moision, B., and J. Hamkins. "Coded Modulation for the Deep-Space Optical Channel: Serially Concatenated Pulse-Position Modulation." *The Interplanetary Network Progress Report,* vol. 42-161 (May 15, 2005): 1–25. https://ipnpr.jpl.nasa.gov/progress_report/42-161/161T.pdf.

[2] Quirk, Kevin J., Jonathan W. Gin, and Meera Srinivasan. "Optical PPM Synchronization for Photon Counting Receivers." In *2008 IEEE Military Communications Conference (MILCOM 2008)*, 1–7. San Diego, CA: IEEE, 2008. https://doi.org/10.1109/MILCOM.2008.4753054.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions

`ccsdsSCPPMEncode` | `ccsdsSCPPMDecode`

Topics

"End-to-End CCSDS SCPPM Simulation Using Deep Space Poisson Channel"
"CCSDS Optical High Photon Efficiency Telemetry Waveform Generation"

etsiRicianChannel

Filter input signal through multipath ETSI frequency-flat Rician fading channel

Description

The `etsiRicianChannel` System object filters an input signal through a multipath European Telecommunication Standards Institute (ETSI) frequency-flat Rician fading channel. For more information on the `etsiRicianChannel` fading model, see "Channel Model Block Diagram" on page 4-67.

To filter an input signal using a multipath ETSI Rician fading channel:

1. Create the `etsiRicianChannel` 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

```
chan = etsiRicianChannel
chan = etsiRicianChannel(Name,Value)
```

Description

`chan = etsiRicianChannel` creates a multipath ETSI frequency-flat Rician fading channel System object. This object filters a real or complex input signal through the multipath channel to obtain the channel-impaired signal.

`chan = etsiRicianChannel(Name,Value)` sets properties on page 4-62 using one or more name-value pairs. Enclose each property name in quotes. For example, `etsiRicianChannel("SampleRate",2)` sets the input signal sample rate to 2.

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.

`SampleRate` — Input signal sample rate
`1` (default) | positive scalar

Input signal sample rate in Hz, specified as a positive scalar.

Data Types: `double`

`KFactor` — Rician *K*-factor
`3` (default) | nonnegative nonzero scalar

Rician *K*-factor in dB, specified as a nonnegative nonzero scalar.

`KFactor` is the ratio of direct signal power to the total multipath power. For details, see “Channel Model Block Diagram” on page 4-67.

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift for channel path
`0.001` (default) | nonnegative scalar

Maximum Doppler shift for the channel path, specified as a nonnegative scalar. Units are in hertz.

When you set this property to `0`, the channel remains static for the entire input. You can use the `reset` object function to generate a new channel realization. The `MaximumDopplerShift` property value must be smaller than `SampleRate`/10.

Data Types: `double`

`NumSinusoids` — Number of sinusoids used
`48` (default) | positive integer

Number of sinusoids used to model the fading process, specified as a positive integer.

Data Types: `double`

`RandomStream` — Source of random number stream
`"Global stream"` (default) | `"mt19937ar with seed"`

Source of random number stream, specified as one of these options.

- `"Global stream"` — The current global random number stream is used for normally distributed random number generation. In this case, the `reset` object function resets the channel filters only.
- `"mt19937ar with seed"` — The mt19937ar algorithm is used for normally distributed random number generation. In this case, the `reset` object function resets the channel filters and reinitializes the random number stream to the value of the `seed` property.

Data Types: `char` | `string`

Seed — Initial seed of mt19937ar random number stream
`73` (default) | nonnegative integer

Initial seed of the mt19937ar random number stream generator algorithm, specified as a nonnegative integer.

Dependencies

To enable this property, set the `RandomStream` property to `"mt19937ar with seed"`.

Data Types: `double`

`Visualization` — Channel visualization
`"Off"` (default) | `"Impulse response"` | `"Frequency response"` | `"Impulse and frequency responses"` | `"Doppler spectrum"`

Channel visualization, specified as `"Off"`, `"Impulse response"`, `"Frequency response"`, `"Impulse and frequency responses"`, or `"Doppler spectrum"`.

When you set this property to `"Doppler spectrum"`, the values plotted are in dB.

Data Types: `char` | `string`

Usage

Syntax

```
y = chan(x)
[y,pathgains] = chan(x)
```

Description

`y = chan(x)` filters input signal `x` through a multipath ETSI frequency-flat Rician fading channel and returns the output signal in `y`.

`[y,pathgains] = chan(x)` returns the channel path gains of the underlying multipath ETSI frequency-flat Rician fading process in `pathgains`.

Input Arguments

x — Input signal
N_S-by-1 vector

Input signal, specified as an N_S-by-1 vector, where N_S is the number of samples.

Data Types: `double`
Complex Number Support: Yes

Output Arguments

y — Output signal
N_S-by-1 vector

Output signal, returned as an N_S-by-1 vector of complex values with the same data precision as the input signal x on page 4-0 . N_S is the number of samples.

Data Types: `double`
Complex Number Support: Yes

`pathgains` — Path gains
N_S-by-1 vector

Path gains, returned as an N_S-by-1 vector of complex values with the same data precision as the input signal x on page 4-0 . N_S is the number of samples.

Data Types: `double`
Complex Number Support: Yes

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 `etsiRicianChannel`

info	Characteristic information about object

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Transmit Input Signal Through ETSI Rician Channel

Transmit an input signal through a European Telecommunication Standards Institute (ETSI) Rician channel model.

Define the channel configuration using an `etsiRicianChannel` System object and specify its properties.

```
chan = etsiRicianChannel;
chan.SampleRate = 2.9e6;
chan.KFactor = 4;
chan.MaximumDopplerShift = 30;
chan.NumSinusoids = 45;
disp(chan)
```

```
  etsiRicianChannel with properties:

             SampleRate: 2900000
                KFactor: 4
    MaximumDopplerShift: 30

  Use get to show all properties
```

Generate a QPSK-modulated input signal to pass through the channel.

```
txWaveform = pskmod(randi([0 3],chan.SampleRate,1),4);
```

Filter the signal through the Rician channel.

```
rxWaveform = chan(txWaveform);
```

Verify ETSI Rician Channel Outputs Using Two Random Number Generation Methods

Produce the same multipath European Telecommunication Standards Institute (ETSI) Rician fading channel response by using two different methods for random number generation. The multipath ETSI Rician fading channel System object includes two methods for random number generation. You can use the current global stream or the mt19937ar algorithm with a specified seed. By interacting with the global stream, the System object can produce the same outputs from the two methods.

Create `etsiRicianChannel` System object, and then specify its properties. Set the random number generation method as the mt19937ar algorithm.

```
chan = etsiRicianChannel;
chan.SampleRate = 150000;
chan.KFactor = 2;
chan.MaximumDopplerShift = 10;
chan.RandomStream = "mt19937ar with seed";
chan.Seed = 80;
```

Modulate randomly generated data.

```
txWaveform = pskmod(randi([0 3],512,1),4);
```

Filter the modulated data by using the multipath Rician fading channel System object.

```
[ChanOut1,PathGains1] = chan(txWaveform);
```

Set the System object to use the global stream for random number generation.

```
release(chan);
chan.RandomStream = "Global stream";
```

Set the global stream to have the same seed that was specified when creating the multipath Rician fading channel System object.

```
rng(80)
```

Filter the modulated data by using the multipath Rician fading channel System object again.

```
[ChanOut2,PathGains2] = chan(txWaveform);
```

Verify that the channel and path gain outputs are the same for each of the two random number generation methods.

```
isequal(ChanOut1,ChanOut2)
```

```
ans = logical
   1
```

```
isequal(PathGains1,PathGains2)
```

```
ans = logical
   1
```

Plot Doppler Spectrum for ETSI Rician Fading Channel

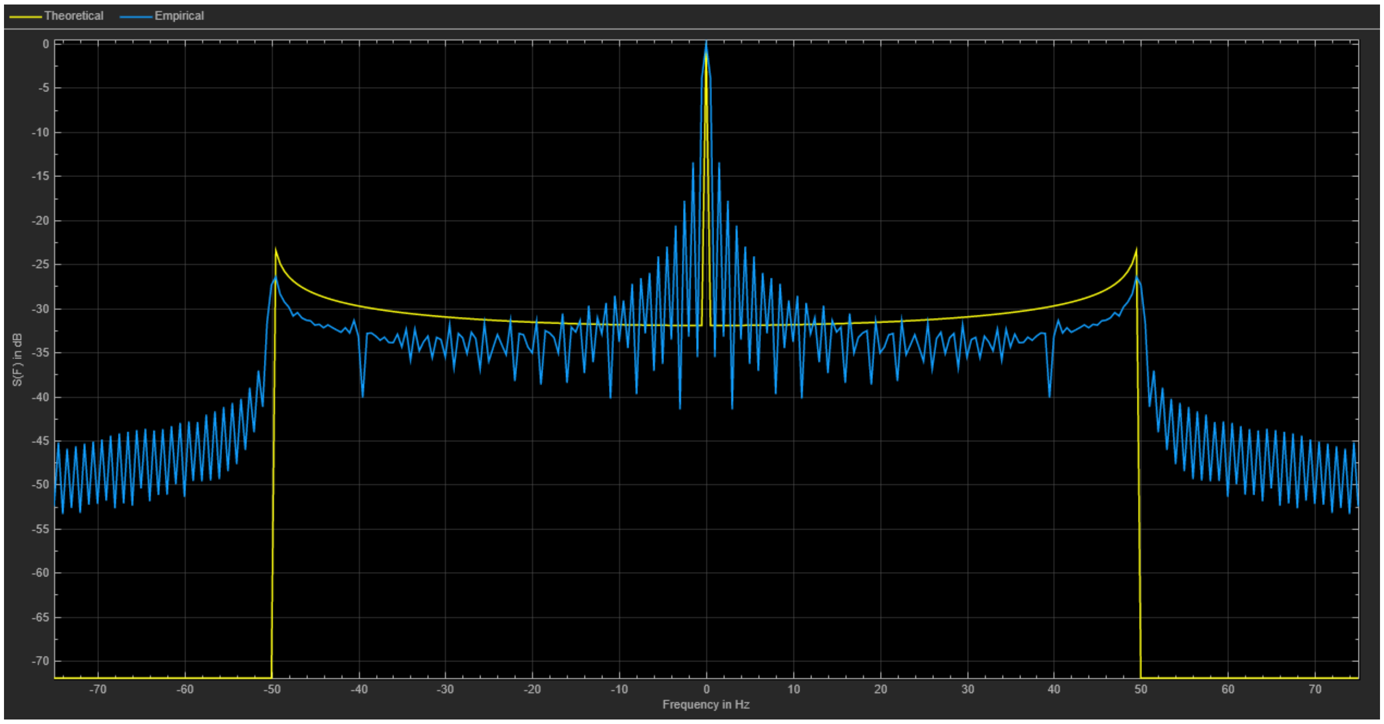
Create a multipath European Telecommunication Standards Institute (ETSI) Rician fading channel and display its Doppler spectrum.

Create `etsiRicianChannel` System object, and then specify its properties.

```
chan = etsiRicianChannel;
chan.SampleRate = 3.6e6;
chan.KFactor = 10;
chan.MaximumDopplerShift = 50;
chan.Visualization = "Doppler Spectrum";  % Jake's Doppler spectrum
```

Generate random binary data for `n` consecutive frames and pass the data through the multipath Rician fading channel.

```
n = 50;
for i = 1:n
  x = randi([0 1],3.6e6,1);
  y = chan(x);  % Spectrum visualization is updated only when the buffer is filled
                % Required samples to fill the buffer is mentioned in the scope
end
```



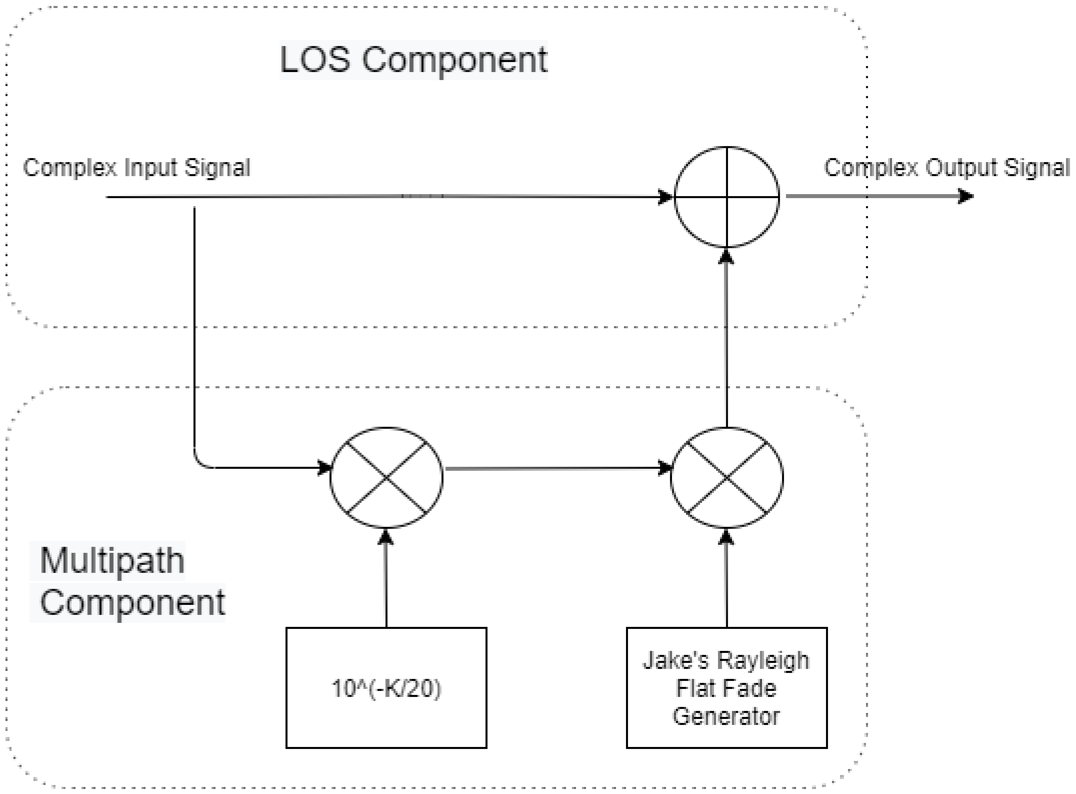
More About

Channel Model Block Diagram

The channel model block diagram provides an overview of the `etsiRicianChannel` System object, as specified in ETSI TS 101 376-5-5 V1.3.1 (2005-02) [1].

- The complex input signal is multiplied by a fixed gain and then by a complex Rayleigh fading gain. These actions form the multipath portion of the signal path. *K* is the Rician fade factor in dB.
- The multipath portion is then added to the direct signal component to form the Rician fading signal. This action forms the line-of-sight (LOS) component of the signal path.

  The coherent summation of many multipath components yield a classical Doppler spectrum for Rayleigh fading process, which when added to the direct path signal, forms the Rician fading signal.
- Noise samples can be subsequently added to the sum of the LOS component and multipath components.



The Rician spectrum for linear Rician factor K_f is given by the following equation:

$$S(f) = \frac{1}{Kf * \pi * fd\sqrt{1 - \left(\frac{f}{fd}\right)^2}} + \delta(f)$$

where:

- K_f is the Rician K-factor, “KFactor” on page 4-0 .
- f_d is the maximum Doppler shift for all multipath signals.
- f is in the range $-f_d < f < +f_d$.
- δ(*f*) is the contribution due to the LOS component.

Note The power of the complex output faded signal is $(1+1/K_f)$.

Version History

Introduced in R2021a

References

[1] ETSI TS 101 376-5-5 V1.3.1 (2005-02). *GEO-Mobile Radio Interface Specifications (Release 1); Part 5: Radio interface physical layer specifications; Sub-part 5: Radio Transmission and Reception; GMR-1 05.005.*

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation is available only when the `Visualization` property is `"Off"`.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

Objects
`comm.RicianChannel` | `comm.RayleighChannel` | `comm.AWGNChannel` | `comm.RayTracingChannel`

Functions
`doppler`

gpsPCode

Generate P-code for GPS satellites

Description

The `gpsPCode` System object generates a precision code (P-code) for a Global Positioning System (GPS) satellite, as defined in IS-GPS-200L Section 3.3.2.2 [1].

To generate a P-code for a GPS satellite:

1. Create the `gpsPCode` 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

```
pgenerator = gpsPCode
pgenerator = gpsPCode(Name,Value)
```

Description

`pgenerator = gpsPCode` creates a default P-code generator System object.

`pgenerator = gpsPCode(Name,Value)` sets “Properties” on page 4-70 using one or more name-value pairs. For example, `'PRNID',10` specifies a pseudo-random noise (PRN) ID of 10.

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.

`PRNID` — GPS satellite PRN index
1 (default) | integer in the range [1, 210] | vector of integer elements in the range [1, 210]

GPS satellite PRN index, specified as one of these options.

- Integer in the range [1, 210] — Use this option to input a PRN index for a single satellite.
- Vector of integer elements in the range [1, 210] — Use this option to input PRN indices for multiple satellites.

For details on PRN ID values, see IS-GPS-200L Tables 3-Ia, 3-Ib, and 6-I [1].

Data Types: `double` | `uint8`

`OutputCodeLength` — Output code length
`10230` (default) | positive integer

Output code length, specified as a positive integer. This length specifies the number of rows in the output P-code.

The default value of `10230` corresponds to 1 millisecond of P-code, as the P-code chips are at 10.23 MHz.

Tunable: Yes

Data Types: `double` | `uint64`

`InitialStateFormat` — Format of the initial state
`"seconds"` (default) | `"datetime"` | `"chips"`

Format of the initial state, specified as `"seconds"`, `"datetime"`, or `"chips"`.

Data Types: `char` | `string`

`InitialTime` — Initial time within one week
`0` (default) | integer in the range [0, 604,800] | `datetime` object

Initial time within one week, specified as one of these options.

- Integer in the range [0, 604,800] — Use this option when you set the `InitialStateFormat` property to `"seconds"`. In this case, initial time specifies the seconds that have elapsed from the beginning of the week.
- `datetime` object — Use this option when you set the `InitialStateFormat` property to `"datetime"`. In this case, initial time specifies the time elapsed from the beginning of the week to the time specified by `datetime` object.

Note The P-code is one week long.

The default value of `0` assumes that you set the `InitialStateFormat` property to `"seconds"`.

Dependencies

To enable this property, set the `InitialStateFormat` property to `"seconds"` or `"datetime"`.

Data Types: `double`

`InitialNumChipsElapsed` — Initial number of elapsed P-code chips
`0` (default) | integer in the range [0, 604,800x10.23e6]

Initial number of elapsed P-code chips, from the beginning of the week, specified as an integer in the range [0, 604,800x10.23e6].

The maximum input value, 604,800x10.23e6, is the total number of chips elapsed in one week (7×24×60×60×10.23e6).

Note `10.23e6` is the number of chips elapsed in one second.

Dependencies

To enable this property, set the `InitialStateFormat` property to `"chips"`.

Data Types: `double` | `uint64`

Usage

Syntax

`code = pgenerator()`

Description

`code = pgenerator()`

Output Arguments

code — Generated binary-valued P-code
vector | matrix

Generated binary-valued P-code, specified as one of these options.

- Vector — The System object returns this option when you specify the PRNID property as a scalar.
- Matrix — The System object returns this option when you specify the PRNID property as a vector. Each column of this matrix represents the generated P-code corresponding to the element in the `PRNID` vector.

The number of rows is equal to the value of the OutputCodeLength property. The number of columns is equal to the length of the `PRNID` property. Each element of the vector or matrix is of data type `int8`.

Data Types: `int8`

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 `gpsPCode`

info	Characteristic information about object

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object

isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Generate P-code When Initial Format Is Seconds

Create a precision code generator (P-code) System object™, and then set its properties.

```
pgen = gpsPCode;
pgen.PRNID = [10 50];         % 2 satellites
pgen.OutputCodeLength = 1024;
pgen.InitialTime = 1800;      % Seconds (default)
disp(pgen)
```

```
  gpsPCode with properties:

                 PRNID: [10 50]
      OutputCodeLength: 1024
    InitialStateFormat: "seconds"
           InitialTime: 1800
```

Generate the P-code.

```
code = pgen();
```

Generate P-code When Initial Format Is Chips

Create the P-code System object™ and set its properties.

```
pgen = gpsPCode;
pgen.PRNID = 45;
pgen.OutputCodeLength = 102400;
```

Set the initial state format as chips. Generate the P-code for the last 5,000 chips in one week.

```
pgen.InitialStateFormat = "chips";
% 604,800 is the total seconds in one week
% 10.23e6 is the number of P-code chips that elapsed in one second
pgen.InitialNumChipsElapsed = 604800*10.23e6 - 5000;
code = pgen();
```

Generate P-code When Initial Format Is `datetime` Object

Create a P-code System object™ and specify the PRN index and the output code length.

Set the format of the initial state as a `datetime` object. Generate the P-code for the current time.

```
pgen = gpsPCode;
pgen.PRNID = 25;
pgen.OutputCodeLength = 20460;
pgen.InitialStateFormat = "datetime";
pgen.InitialTime = datetime("now");
code = pgen();
```

Display the properties of the P-code generator.

```
disp(pgen)
```

```
  gpsPCode with properties:

                PRNID: 25
     OutputCodeLength: 20460
   InitialStateFormat: "datetime"
          InitialTime: 03-Mar-2023 22:25:46
```

Get P-Code State Information

Get information from a `gpsPCode` System object™ by using the `info` object function. Observe how the precision of initial time impacts the generation of the P-code.

Create a P-code generator System object™, and then specify its properties.

```
format long
pgen = gpsPCode
```

```
pgen = 
  gpsPCode with properties:

                PRNID: 1
     OutputCodeLength: 10230
   InitialStateFormat: "seconds"
          InitialTime: 0
```

```
pgen.InitialStateFormat = "chips";
pgen.InitialNumChipsElapsed = 8388600;
```

Get the characteristic information about the P-code generator.

```
pgen.info
```

```
ans = struct with fields:
    TotalNumChipsElapsed: 8388600
     TotalSecondsElapsed: 0.820000000000000
```

Advance the time by a quarter of a P-code chip time (that is, 0.25/10.23e6).

```
pgen1 = gpsPCode;
pgen1.InitialTime = pgen.info.TotalSecondsElapsed + 0.25/10.23e6
```

```
pgen1 = 
  gpsPCode with properties:

                PRNID: 1
     OutputCodeLength: 10230
```

```
    InitialStateFormat: "seconds"
           InitialTime: 0.820000024437928
```

```
pgen1.info
```

```
ans = struct with fields:
    TotalNumChipsElapsed: 8388600
     TotalSecondsElapsed: 0.820000000000000
```

The `info` function output shows no increment in the `TotalNumChipsElapsed` in this case, because `TotalNumChipsElapsed` is calculated internally using the function `round`.

Advance the time by half of a P-code chip time now (that is, 0.5/10.23e6).

```
pgen2 = gpsPCode;
pgen2.InitialTime = pgen.info.TotalSecondsElapsed + 0.5/10.23e6
```

```
pgen2 = 
  gpsPCode with properties:

                 PRNID: 1
      OutputCodeLength: 10230
    InitialStateFormat: "seconds"
           InitialTime: 0.820000048875855
```

```
pgen2.info
```

```
ans = struct with fields:
    TotalNumChipsElapsed: 8388601
     TotalSecondsElapsed: 0.820000097751711
```

The `info` function output now shows the `TotalNumChipsElapsed` is incremented by one, due to the internal usage of `round()` function.

Compare the output of each System object call.

```
code = pgen();
code1 = pgen1();
code2 = pgen2();
isequal(code, code1) % code and code1 are equal
```

```
ans = logical
   1
```

```
isequal(code1,code2) % code1 and code2 are unequal
```

```
ans = logical
   0
```

Version History
Introduced in R2021b

References

[1] IS-GPS-200L. "*NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*." GPS Enterprise Space & Missile Systems Center (SMC) - LAAFB, May 14, 2020.

Extended Capabilities

C/C++ Code Generation
Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`gnssCACode`

Objects
`comm.GoldSequence` | `comm.PNSequence`

Topics
“GPS Waveform Generation”

lutzLMSChannel

Filter input signal through Lutz LMS frequency-flat fading channel

Description

The `lutzLMSChannel` System object filters a real or complex input signal through a Lutz land mobile-satellite (LMS) frequency-flat fading communication channel, as defined in [1].

To filter an input signal through a Lutz LMS time-varying channel:

1. Create the `lutzLMSChannel` 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

```
chan = lutzLMSChannel
chan = lutzLMSChannel(Name=Value)
```

Description

`chan = lutzLMSChannel` creates a Lutz LMS frequency-flat fading channel System object. Use this channel to simulate an LMS time-varying channel for a single geostationary satellite operating in the L-band.

The default System object parameters are set to a scenario in which the mobile is moving at a constant speed of 3 km/hr on a highway with a conical spiral antenna operating at a 1.54 GHz carrier frequency and a satellite elevation angle of 34 degrees, as specified in [1].

`chan = lutzLMSChannel(Name=Value)` sets properties on page 4-77 using one or more name-value arguments. For example, `SampleRate=20e3` sets the input signal sample rate to `20e3`.

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.

`SampleRate` — Input signal sample rate
`7.68e6` (default) | positive scalar

Input signal sample rate in Hz, specified as a positive scalar.

Data Types: `double`

`InitialState` — Initial state of channel
`"Good"` (default) | `"Bad"`

Initial state of the channel, specified as `"Good"` or `"Bad"`.

Tunable: Yes

Data Types: `char` | `string`

`KFactor` — K-factor of Rician fading channel
`11.7` (default) | real scalar

K-factor of the Rician fading channel in dB, specified as a real scalar. This property applies only when the channel state is good.

Tunable: Yes

Data Types: `double`

`LogNormalFading` — Fading parameters of Rayleigh log-normal fading channel
`[-8.8 3.8]` (default) | two-element vector

Fading parameters of the Rayleigh log-normal fading channel in dB, specified as a two-element vector. The first element corresponds to the mean power level due to shadowing. Nominally, the mean power level due to shadowing is a negative value. The second element corresponds to the standard deviation of power level due to shadowing.

This property applies only when the channel state is bad.

Tunable: Yes

Data Types: `double`

`StateDurationDistribution` — Distribution type used for state duration
`"Exponential"` (default) | `"None"`

Distribution type used for state duration, specified as one of these values.

- `"Exponential"` — The duration of each state follows an exponential distribution. The MeanStateDuration property specifies the mean duration of each state.
- `"None"` — The MeanStateDuration property directly specifies the duration of each state.

Tunable: Yes

Data Types: `char` | `string`

`MeanStateDuration` — Mean duration of each state
`[1800 14.4]` (default) | two-element vector

Mean duration of each state in seconds, specified as a two-element vector. The first element corresponds to the mean duration of the good state, and the second element corresponds to the mean duration of the bad state. Both the elements must be nonzero values.

Tunable: Yes

Data Types: `double`

`MaximumDopplerShift` — Maximum Doppler shift due to mobile movement
`4.2807` (default) | nonnegative scalar

Maximum Doppler shift due to mobile movement in Hz, specified as a nonnegative scalar.

When this property value is 0, the channel is static. The default value of 4.2807 Hz corresponds to a mobile speed of 3 km/hr at a carrier frequency of 1.54 GHz.

Data Types: `double`

`ChannelFiltering` — Channel filtering
`true` or `1` (default) | `false` or `0`

Channel filtering, specified as one of these logical values.

- `1` (`true`) — The object accepts an input signal and produces a filtered output signal, in addition to the channel path gains, sample times, and state series.
- `0` (`false`) — The object does not accept an input signal, produces no filtered output signal, and outputs only channel path gains, sample times, and state series. You must specify the duration of the fading process by using the NumSamples property, and the sampling rate by using the SampleRate property.

Data Types: `logical`

`NumSamples` — Number of time samples
`7680` (default) | nonnegative integer

Number of time samples used to set the duration of the fading process realization, specified as a nonnegative integer.

Tunable: Yes

Dependencies

To enable this property, set ChannelFiltering property to `false`.

Data Types: `double`

`OutputDataType` — Data type of step method outputs
`"double"` (default) | `"single"`

Data type of step method outputs, specified as one of these values.

- `"double"`
- `"single"`

Dependencies

To enable this property, set ChannelFiltering property to `false`.

Data Types: `char` | `string`

`FadingTechnique` — Channel model fading technique
`"Filtered Gaussian noise"` (default) | `"Sum of sinusoids"`

Channel model fading technique, specified as "Filtered Gaussian noise" or "Sum of sinusoids".

Data Types: char | string

NumSinusoids — Number of sinusoids used
48 (default) | positive integer

Number of sinusoids used to generate the Doppler fading samples, specified as a positive integer.

Dependencies

To enable this property, set the FadingTechnique property to "Sum of sinusoids".

Data Types: double | uint16

RandomStream — Source of random number stream
"Global stream" (default) | "mt19937ar with seed"

Source of the random number stream, specified as "Global stream" or "mt19937ar with seed".

- When you specify "Global stream", the object uses the current global random number stream for uniformly and normally distributed random number generation. In this case, the reset object function resets only the filters.
- When you specify "mt19937ar with seed", the object uses the mt19937ar algorithm for uniformly and normally distributed random number generation. In this case, the reset object function resets the filters and reinitializes the random number stream to the value of the Seed property.

Data Types: char | string

Seed — Initial seed
73 (default) | nonnegative integer

Initial seed of the mt19937ar random number stream generator algorithm, specified as a nonnegative integer. When you call the reset object function, it reinitializes the mt19937ar random number stream to the Seed value.

Dependencies

To enable this property, set the RandomStream property to "mt19937ar with seed".

Data Types: double | uint32

Visualization — Channel visualization
"Off" (default) | "Impulse response" | "Frequency response" | "Impulse and frequency responses" | "Doppler spectrum"

Channel visualization, specified as one of these options.

- "Off"
- "Impulse response"
- "Frequency response"
- "Impulse and frequency responses"

- `"Doppler spectrum"`

When you set this property to enable the visualization, selected channel characteristics are animated in separate figures, with each System object call.

For more information, see the "Channel Visualization" on page 4-88 section.

Data Types: `char` | `string`

`TimeShare` — Time share of channel in good and bad states
`[0.9921 0.0079]` (default) | two-element vector

This property is read-only.

Time share of the channel in good and bad states, returned as a two-element vector. The first element is the ratio of the good state mean duration to the sum of good and bad state mean durations. The second element is the ratio of the bad state mean duration to the sum of good and bad state mean durations.

Data Types: `double`

Usage

Syntax

```
[pathgains,sampletimes,stateseries] = chan()
[y,pathgains,sampletimes,stateseries] = chan(x)
```

Description

`[pathgains,sampletimes,stateseries] = chan()` produces path gains, `pathgains`, sample times, `sampletimes`, and state series, `stateseries` for a Lutz LMS frequency-flat fading channel.

In this case, the System object acts as a source of path gains, sample times, and state series.

Specify the duration of the fading process by using the NumSamples property. Specify the data type of `pathgains` and `sampletimes` using the OutputDataType property.

Note This syntax is applicable when you set the ChannelFiltering property to `false`.

`[y,pathgains,sampletimes,stateseries] = chan(x)` filters the input signal, `x`, through a Lutz LMS frequency-flat fading channel, and returns the output channel-impaired signal in `y`, in addition to the outputs in the previous syntax.

Note This syntax is applicable when you set the ChannelFiltering property to `true`.

Input Arguments

x — Input signal
N_S-by-1 vector

Input signal, specified as an N_S-by-1 vector, where N_S is the number of input samples.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

y — Output signal
N_S-by-1 vector

Output signal, returned as an N_S-by-1 vector of complex values with the same data precision and length as the input signal `x`. N_S is the number of input samples.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathgains` — Channel path gains of fading process
N_S-by-1 vector

Channel path gains of fading process, returned as an N_S-by-1 vector of complex values.

- When you set the ChannelFiltering property to `true`, `pathgains` is of the same data precision as the input signal `x`, and N_S is the number of input samples.
- When you set the ChannelFiltering property to `false`, `pathgains` is of the same data precision as the OutputDataType property and N_S is equal to the NumSamples property.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampletimes` — Sample times of channel snapshots
N_S-by-1 vector

Sample times of channel snapshots, returned as an N_S-by-1 vector.

- When you set the ChannelFiltering property to `true`, `sampletimes` is of the same data precision as the input signal `x`, and N_S is the number of input samples.
- When you set the ChannelFiltering property to `false`, `sampletimes` is of the same data precision as the OutputDataType property and N_S is equal to the NumSamples property.

Data Types: `single` | `double`

`stateseries` — State series of channel
N_S-by-1 vector

State series of the channel, returned as an N_S-by-1 vector. Each value of this vector describes the state in which channel is present for that channel snapshot. A value of `0` represents bad state. A value of `1` represents good state. A value between 0 and 1 represents a state transition.

- When you set the ChannelFiltering property to `true`, N_S is the number of input samples.
- When you set the ChannelFiltering property to `false`, N_S is equal to the NumSamples property.

`stateseries` is of the data type logical.

Data Types: `logical`

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 `lutzLMSChannel`

info	Characteristic information about object

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Create and Configure Lutz LMS Channel

Define the channel configuration using a `lutzLMSChannel` System object and set the properties to these values.

- Symbol rate: 4800 symbols/sec (Assume sample rate to be equal to symbol rate)
- Product of the maximum Doppler frequency and symbol duration that indicates a normalized measure of channel state time variations (fd*T): 0.01
- Rice factor (K-factor): 20 dB
- Fading parameters of Rayleigh log-normal fading channel in bad state: mean (mu): -20.8 dB and standard deviation (sigma): -0.09 dB
- Fading signal (Doppler Spectrum): Jakes model with sum of sinusoids technique (10 sinusoids)
- Good state time-share parameter (X): 0.9
- Mean duration of shadowing period (Ct): 10 sec

```
chan = lutzLMSChannel;
chan.SampleRate = 4800;                 % Assuming same as symbol rate
chan.MaximumDopplerShift = 48;          % fd = 0.01/T = 0.01*SymbolRate
chan.KFactor = 20;                      % Rice factor
chan.LogNormalFading = [-20.8 -0.09];   % [mu sigma]
chan.MeanStateDuration = [9 1];         % [X*Ct (1-X)*Ct];
chan.FadingTechnique = "Sum of sinusoids";
chan.NumSinusoids = 10;
```

Display the channel characteristics.

```
disp(chan)
```

```
  lutzLMSChannel with properties:
```

```
                   SampleRate: 4800
                 InitialState: "Good"
                      KFactor: 20
               LogNormalFading: [-20.8000 -0.0900]
    StateDurationDistribution: "Exponential"
            MeanStateDuration: [9 1]
           MaximumDopplerShift: 48
             ChannelFiltering: true

  Use get to show all properties
```

Model Lutz LMS Channel Using Lutz IEEE Paper Configuration

Model the Lutz LMS channel using the configuration entries from the Lutz IEEE paper (Table II), as defined in the IEEE Transactions on Vehicular Technology article (stated in the *References* section).

Define the channel configuration using a `lutzLMSChannel` System object and set the properties to these values.

- Rician K-factor (k): 10.2
- Fading parameters of Rayleigh log-normal fading channel in bad state: mean (mu): -8.9 dB and standard deviation (sig): 5.1 dB
- Mean duration of good state, in meters (Dg): 90
- Mean duration of bad state, in meters (Db): 29
- Assume mobile speed is 10 km/hr at 1.54 GHz carrier frequency

```
v = 10;                        % Mobile speed in km/hr
v_ms = (v*1000/3600);          % Mobile speed in m/s
fc = 1.54e9;                   % Carrier frequency in Hz
c = physconst("lightspeed");   % Speed of light in m/s
fd = v_ms/c*fc;                % Mobile maximum Doppler frequency in Hz

k = 10.2;                      % Rician K-factor in dB
mu = -8.9;                     % Bad state mean
sig = 5.1;                     % Bad state standard deviation

% Get the mean state duration in seconds
Dg = 90;                       % Mean duration of good state in m
Db = 29;                       % Mean duration of bad state in m
Dg_sec = Dg/v_ms;              % Mean duration of good state in sec
Db_sec = Db/v_ms;              % Mean duration of bad state in sec

chan = lutzLMSChannel;
chan.MaximumDopplerShift = fd;
chan.KFactor = k;
chan.LogNormalFading = [mu sig];
chan.MeanStateDuration = [Dg_sec Db_sec];
```

Display the channel characteristics.

```
disp(chan)
```

```
  lutzLMSChannel with properties:
```

```
                  SampleRate: 7680000
                InitialState: "Good"
                     KFactor: 10.2000
              LogNormalFading: [-8.9000 5.1000]
    StateDurationDistribution: "Exponential"
            MeanStateDuration: [32.4000 10.4400]
          MaximumDopplerShift: 14.2691
             ChannelFiltering: true

  Use get to show all properties
```

Plot Space Series and State Series for Lutz LMS Channel

Create and Configure the Channel

Create a Lutz LMS channel using `lutzLMSChannel` System object.

Consider a scenario in which a mobile is moving with variable velocity and a maximum Doppler shift (*fd*) of 25 Hz. This is the behavior of the mobile during the scenario:

- First 5 minutes: Good state with a Rice factor of 20 dB
- Next 1 minutes: Good state with a Rice factor of 8 dB
- Next 2 minutes: Bad state with the mean of fading level (*mu*) at -8 dB and standard deviation of fading level (*sigma*) at 2 dB

```
fd = 25;                              % In Hz
TimeDurInMins = [5 1 2];              % In minutes
TimeDurInSec = TimeDurInMins*60;      % In seconds
riceFactor = [20 8 nan];              % In dB
mu = [nan nan -8];                    % In dB
sigma = [nan nan 2];                  % In dB
sampleRate = 1000;                    % In Hz
state = ["Good" "Good" "Bad"];
```

Create the System object and configure its properties to these values.

- Distribution type used for state duration: None
- Random stream: mt19937ar with seed
- Channel filtering: 0 (Disabled)

```
% Initialize the channel
chan = lutzLMSChannel;
chan.InitialState = "Good";
chan.StateDurationDistribution = "None";
chan.RandomStream = "mt19937ar with seed";
chan.ChannelFiltering = 0;
chan.SampleRate = sampleRate;
chan.MaximumDopplerShift = fd;
```

Display the channel characteristics.

```
disp(chan)
```

```
  lutzLMSChannel with properties:
```

```
                      SampleRate: 1000
                    InitialState: "Good"
                         KFactor: 11.7000
                 LogNormalFading: [-8.8000 3.8000]
       StateDurationDistribution: "None"
               MeanStateDuration: [1800 14.4000]
             MaximumDopplerShift: 25
                ChannelFiltering: false
                      NumSamples: 7680
                  OutputDataType: "double"

  Use get to show all properties
```

Generate Lutz LMS Channel

Simulate the channel for all three cases of the scenario.

```
numChanges = length(TimeDurInSec);
pathGainsT = cell(numChanges,1);
stateSeriesT = cell(numChanges,1);
sampleTimesT = cell(numChanges,1);

% Run the channel across different cases
for loopIdx = 1:length(TimeDurInSec)
    chan.InitialState = state(loopIdx);
    if strcmpi(chan.InitialState,"Good")
        chan.MeanStateDuration = [TimeDurInSec(loopIdx) 0];
        chan.KFactor = riceFactor(loopIdx);
    else
        chan.MeanStateDuration = [0 TimeDurInSec(loopIdx)];
        chan.LogNormalFading = [mu(loopIdx) sigma(loopIdx)];
    end
    chan.NumSamples = round(sum(chan.MeanStateDuration)*chan.SampleRate);
    [pathGainsT{loopIdx},sampleTimesT{loopIdx},stateSeriesT{loopIdx}] = chan();
end

% Convert cell to matrix
pathGains = cell2mat(pathGainsT);
sampleTimes = cell2mat(sampleTimesT);
stateSeries = cell2mat(stateSeriesT);
```

Visualize Space Series and State Series

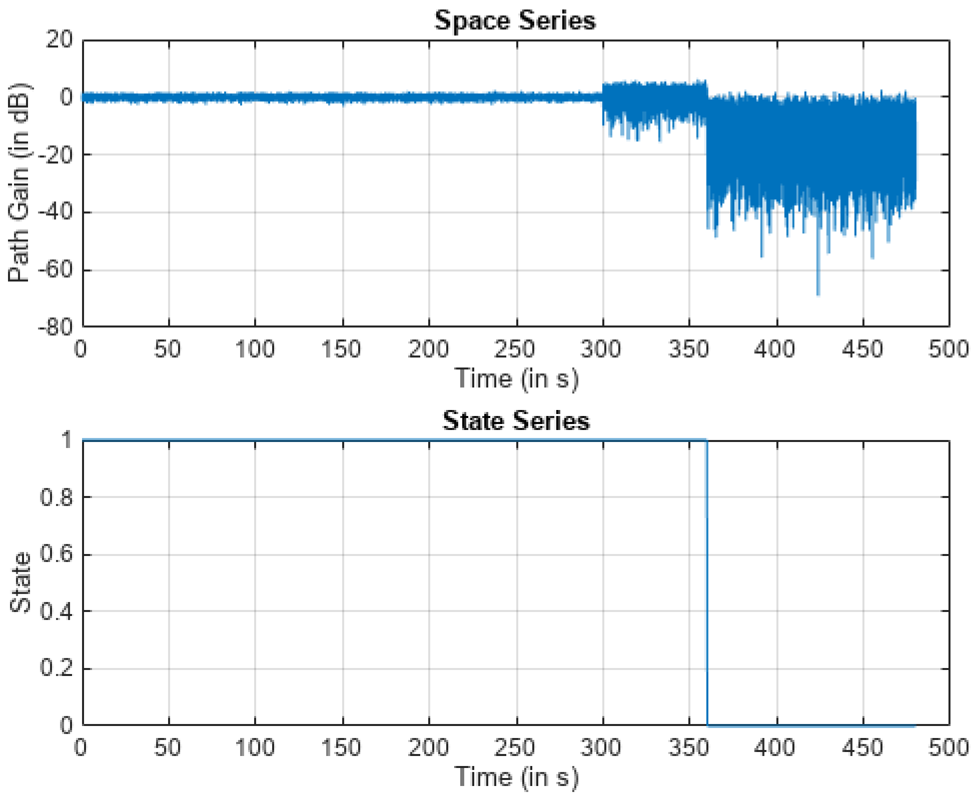
Plot the space series as a function of time.

```
figure
subplot(211)
plot(sampleTimes,20*log10(abs(pathGains)))
title("Space Series")
xlabel("Time (in s)")
ylabel("Path Gain (in dB)")
grid on
```

Plot the state series as a function of time.

```
subplot(212)
plot(sampleTimes,stateSeries)
title("State Series")
xlabel("Time (in s)")
```

```
ylabel("State")
grid on
```



Get Lutz LMS Channel Information

Get channel information from a `LutzLMSChannel` System object by using the `info` object function.

Create a Lutz LMS channel System object and specify its properties.

```
chan = lutzLMSChannel;
chan.SampleRate = 6000;
chan.KFactor = 20;
chan.MeanStateDuration = [8 2];
disp(chan)
```

```
  lutzLMSChannel with properties:

                   SampleRate: 6000
                 InitialState: "Good"
                      KFactor: 20
              LogNormalFading: [-8.8000 3.8000]
    StateDurationDistribution: "Exponential"
            MeanStateDuration: [8 2]
          MaximumDopplerShift: 4.2807
             ChannelFiltering: true
  Show all properties
```

QPSK-modulate a random input signal, and then pass it through the channel.

```
numSamples = 2e4;
txWaveform = pskmod(randi([0 3],numSamples,1),4);
[rxWaveform,pathGains,sampleTimes,stateSeries] = chan(txWaveform);
```

Get the characteristic information about the Lutz LMS channel.

```
info(chan)
```

```
ans = struct with fields:
                   PathDelays: 0
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 20000
```

Transmit another QPSK-modulated random input signal through the channel

```
numSamples2 = 3e4;
txWaveform2 = pskmod(randi([0 3],numSamples2,1),4);
[rxWaveform2,pathGains2,sampleTimes2,stateSeries2] = chan(txWaveform2);
```

Observe the change in number of samples processed.

```
info(chan)
```

```
ans = struct with fields:
                   PathDelays: 0
           ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
          NumSamplesProcessed: 50000
```

More About

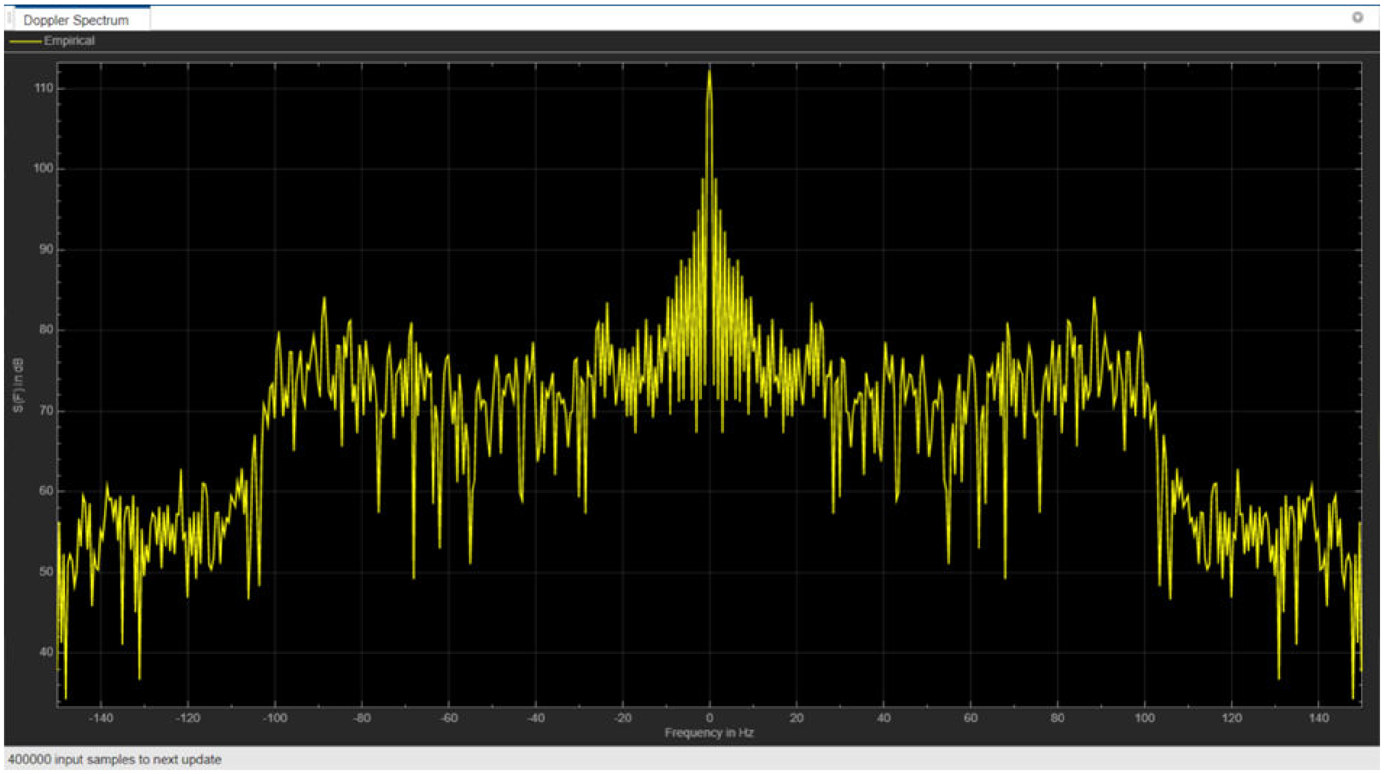
Channel Visualization

The `lutzLMSChannel` System object enables visualization of the channel impulse response, frequency response, and Doppler spectrum.

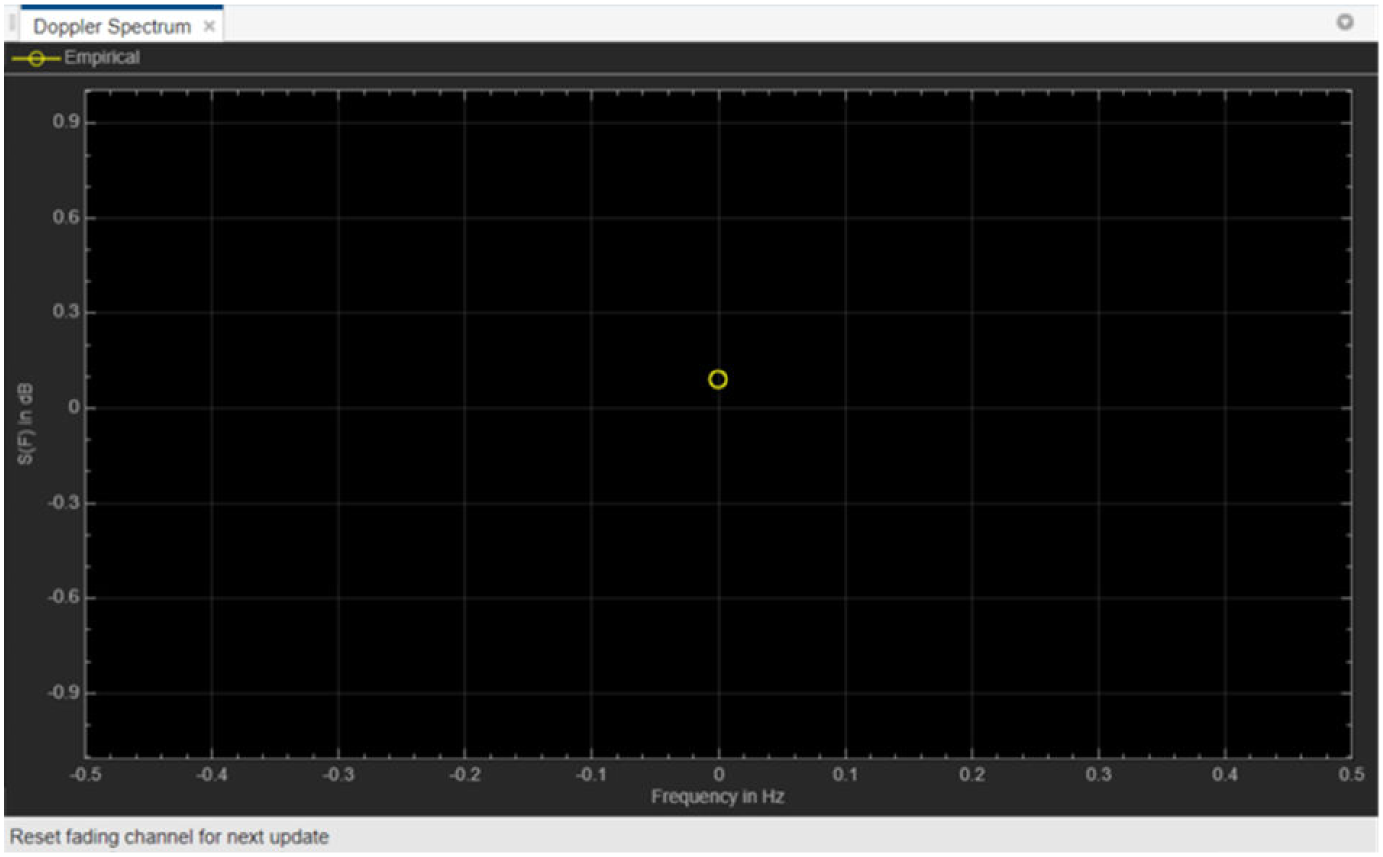
- *Doppler Spectrum*

  The Doppler spectrum plot displays the empirically determined spectrum from the path gains of the channel. The Doppler spectrum values are in dB.

  When there is no mobile movement, the channel is static channel. The empirical data is displayed as a line for nonstatic channels and as a point for static channels. Before the empirical plot is updated, the internal buffer must be completely filled with the required number of channel samples. The empirical plot is the running mean of the spectrum that is calculated from each full buffer. For nonstatic channels, the number of input samples that is needed before the next update is displayed in the status bar located at the bottom of plot. The number of samples that is needed is a function of the sample rate and the maximum Doppler shift depending on mobile movement. For static channels, the text `"Reset fading channel for next update"` is displayed.



Nonstatic Channel



Static Channel

- For channel impulse and frequency visualizations, see “Channel Visualization”.

Version History

Introduced in R2022b

References

[1] Lutz, E., D. Cygan, M. Dippold, F. Dolainsky, and W. Papke. “The Land Mobile Satellite Communication Channel-Recording, Statistics, and Channel Model.” *IEEE Transactions on Vehicular Technology* 40, no. 2 (May 1991): 375–86. https://doi.org/10.1109/25.289418.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation is available only when the `Visualization` property is `"Off"`.
- See “System Objects in MATLAB Code Generation” (MATLAB Coder).

See Also

`p681LMSChannel` | `etsiRicianChannel` | `comm.RayleighChannel`

Topics

"Simulate and Visualize Land Mobile-Satellite Channel"

p681LMSChannel

Filter input signal through ITU-R P.681-11 LMS frequency-flat fading channel

Description

The `p681LMSChannel` System object filters a real or complex input signal through a frequency-flat fading land mobile-satellite (LMS) communication channel, as defined in the ITU-R Recommendation P.681-11 Section 6.2 [1].

To filter an input signal through a P.681-11 LMS time-varying channel:

1. Create the `p681LMSChannel` 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

```
chan = p681LMSChannel
chan = p681LMSChannel(Name=Value)
```

Description

`chan = p681LMSChannel` creates an ITU-R P.681-11 LMS frequency-flat fading channel System object.

The default System object has the environment set to an urban scenario, with carrier frequency of 2.2 GHz and an elevation angle of 45 degrees. This object models a single geostationary satellite.

`chan = p681LMSChannel(Name=Value)` sets properties on page 4-92 using one or more name-value arguments. For example, `SampleRate=20e3` sets the input signal sample rate to `20e3`.

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.

`SampleRate` — Input signal sample rate
`7.68e6` (default) | positive scalar

Input signal sample rate in hertz, specified as a positive scalar.

Data Types: `double`

`InitialState` — Initial state of channel
`"Good"` (default) | `"Bad"`

Initial state of the channel, specified as `"Good"` or `"Bad"`.

Data Types: `char` | `string`

`CarrierFrequency` — Carrier frequency
`2.2e9` (default) | nonnegative scalar

Carrier frequency in hertz, specified as a nonnegative scalar.

Data Types: `double`

`ElevationAngle` — Path elevation angle to geostationary satellite
`45` (default) | scalar

Path elevation angle to a geostationary satellite in degrees, specified as a scalar. The nominal value is in the range [0, 90].

Data Types: `double`

`MobileSpeed` — Speed of mobile terminal
`0.8333` (default) | nonnegative scalar

Speed of mobile terminal in m/s, specified as a nonnegative scalar. The default value of `0.8333` m/s translates to 3 km/h.

This property affects the Doppler spread applied to the multipath component and also the Doppler shift applied to the direct path component.

Setting this property to `0`, results in a static channel. In this case, the Doppler spread is not applicable for the multipath component and the Doppler shift is also not applied in direct path component of the channel.

Data Types: `double`

`AzimuthOrientation` — Azimuth orientation
`0` (default) | scalar

Azimuth orientation in degrees, specified as a scalar. This value specifies the direction of movement of the ground or mobile terminal. The nominal value is in the range [0, 360].

When you set this property to odd multiples of 90, the Doppler shift caused by the mobile movement in the direct path component is nonexistent.

Data Types: `double`

`Environment` — Type of propagation environment
`"Urban"` (default) | `"Suburban"` | `"RuralWooded"` | `"Village"` | `"Residential"` | `"Highway"` | `"Rural"` | `"Train"` | `"Custom"`

Type of propagation environment, specified as one of these values.

- `"Urban"`

- `"Suburban"`
- `"RuralWooded"`
- `"Village"`
- `"Residential"`
- `"Highway"` — Applicable only when you set the value of CarrierFrequency property in the range [10, 20] GHz
- `"Rural"` — Applicable only when you set the value of CarrierFrequency property in the range [10, 20] GHz
- `"Train"` — Applicable only when you set the value of CarrierFrequency property in the range [10, 20] GHz
- `"Custom"`

When you set this property to `"Custom"`, configure the propagation environment using these properties.

- StateDistribution
- MinStateDuration
- DirectPathDistribution
- MultipathPowerCoefficients
- StandardDeviationCoefficients
- DirectPathCorrelationDistance
- TransitionLengthCoefficients
- StateProbabilityRange

Data Types: `char` | `string`

`StateDistribution` — Parameters of state duration distribution
`[3.0639 2.9108; 1.6980 1.2602]` (default) | 2-by-2 matrix

Parameters of state duration distribution in dB, specified as a 2-by-2 matrix. For example, if you specify the input as [*muG*, *muB*; *sigmaG*, *sigmaB*], then:

- *muG* and *sigmaG* represent the mean and standard deviation of good state duration, respectively.
- *muB* and *sigmaB* represent the mean and standard deviation of bad state duration, respectively.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`MinStateDuration` — Minimum duration of each state
`[10 6]` (default) | two-element row vector

Minimum duration of each state in meters, specified as a two-element row vector. The first element corresponds to good state and the second element corresponds to bad state.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`DirectPathDistribution` — Parameters of direct path amplitude distribution
`[-1.8225 -15.4844; 1.1317 3.3245]` (default) | 2-by-2 matrix

Parameters of direct path amplitude distribution in dB, specified as a 2-by-2 matrix. For example, if you specify the input as [*muMaG*, *muMaB*; *sigmaMaG*, *sigmaMaB*], then:

- *muMaG* and *sigmaMaG* represent the mean of the direct path amplitude (*Ma*) and the standard deviation of *Ma* in good state, respectively.
- *muMaB* and *sigmaMaB* represent the mean and standard deviation of *Ma* in bad state, respectively.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`MultipathPowerCoefficients` — Coefficients to compute multipath power
`[-0.0481 0.9434; -14.7450 -1.7555]` (default) | 2-by-2 matrix

Coefficients to compute the multipath power, specified as a 2-by-2 matrix. For example, if you specify the input as [*h1G*, *h1B*; *h2G*, *h2B*], then:

- *h1G* and *h2G* represent the coefficients in good state.
- *h1B* and *h2B* represent the coefficients in bad state.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`StandardDeviationCoefficients` — Coefficients to compute standard deviation of direct path amplitude
`[-0.4643 -0.0798; 0.3334 2.8101]` (default) | 2-by-2 matrix

Coefficients to compute standard deviation of direct path amplitude in all states, specified as a 2-by-2 matrix. For example, if you specify the input as [*g1G*, *g1B*; *g2G*, *g2B*], then:

- *g1G* and *g2G* represent the coefficients in good state.
- *g1B* and *g2B* represent the coefficients in bad state.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`DirectPathCorrelationDistance` — Direct path amplitude correlation distance
`[1.7910 1.7910]` (default) | two-element row vector

Direct path amplitude correlation distance (*Lcorr*) in meters, specified as a two-element row vector. The first element corresponds to good state and the second element corresponds to bad state.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`TransitionLengthCoefficients` — Coefficients to compute transition length

`[0.0744; 2.1423]` (default) | two-element column vector

Coefficients to compute the transition length (*f1;f2*), specified as a two-element column vector.

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`StateProbabilityRange` — Minimum and maximum probability of each state

`[0.05 0.1; 0.95 0.9]` (default) | 2-by-2 matrix

Minimum and maximum probability of each state, specified as a 2-by-2 matrix. For example, if you specify the input as [*pminG*, *pminB*; *pmaxG*, *pmaxB*], then:

- *pminG* and *pmaxG* represent the minimum and maximum values of state probability in good state.
- *pminB* and *pmaxB* represent the minimum and maximum values of state probability in bad state.

The minimum probability must be less than the maximum probability in a state. The value of each element must be in range [0, 1].

Dependencies

To enable this property, set the Environment property to `"Custom"`.

Data Types: `double`

`ChannelFiltering` — Channel filtering

`true` or `1` (default) | `false` or `0`

Channel filtering, specified as one of these logical values.

- `1` (`true`) — The object accepts an input signal and produces a filtered output signal, in addition to the channel path gains, sample times, and state series.
- `0` (`false`) — The object does not accept an input signal, produces no filtered output signal, and outputs only channel path gains, sample times, and state series. You must specify the duration of the fading process by using the NumSamples property, and the sampling rate by using the SampleRate property.

Data Types: `logical`

`NumSamples` — Number of time samples

`7680` (default) | nonnegative integer

Number of time samples used to set the duration of the fading process realization, specified as a nonnegative integer.

Tunable: Yes

Dependencies

To enable this property, set ChannelFiltering property to `false`.

Data Types: `double`

`OutputDataType` — Data type of outputs
`"double"` (default) | `"single"`

Data type of outputs, specified as one of these values.

- `"double"`
- `"single"`

Dependencies

To enable this property, set ChannelFiltering property to `false`.

Data Types: `char` | `string`

`FadingTechnique` — Channel model fading technique
`"Filtered Gaussian noise"` (default) | `"Sum of sinusoids"`

Channel model fading technique, specified as `"Filtered Gaussian noise"` or `"Sum of sinusoids"`.

Data Types: `char` | `string`

`NumSinusoids` — Number of sinusoids used
48 (default) | positive integer

Number of sinusoids used to generate the Doppler fading samples, specified as a positive integer.

Dependencies

To enable this property, set the FadingTechnique property to `"Sum of sinusoids"`.

Data Types: `double` | `uint16`

`RandomStream` — Source of random number stream
`"Global stream"` (default) | `"mt19937ar with seed"`

Source of the random number stream, specified as `"Global stream"` or `"mt19937ar with seed"`.

- When you specify `"Global stream"`, the object uses the current global random number stream for uniformly and normally distributed random number generation. In this case, the `reset` object function resets only the filters.
- When you specify `"mt19937ar with seed"`, the object uses the mt19937ar algorithm for uniformly and normally distributed random number generation. In this case, the `reset` object function resets the filters and reinitializes the random number stream to the value of the Seed property.

Data Types: `char` | `string`

Seed — Initial seed
73 (default) | nonnegative integer

Initial seed of the mt19937ar random number stream generator algorithm, specified as a nonnegative integer. When you call the `reset` object function, it reinitializes the mt19937ar random number stream to the `Seed` value.

Dependencies

To enable this property, set the RandomStream property to `"mt19937ar with seed"`.

Data Types: `double` | `uint32`

`Visualization` — Channel visualization

`"Off"` (default) | `"Impulse response"` | `"Frequency response"` | `"Impulse and frequency responses"` | `"Doppler spectrum"`

Channel visualization, specified as one of these options.

- `"Off"`
- `"Impulse response"`
- `"Frequency response"`
- `"Impulse and frequency responses"`
- `"Doppler spectrum"`

When you set this property to enable the visualization, selected channel characteristics are animated in separate figures, with each System object call.

For more information, see the “Channel Visualization” on page 4-105 section.

Data Types: `char` | `string`

Usage

Syntax

```
[pathgains,sampletimes,stateseries] = chan()
[y,pathgains,sampletimes,stateseries] = chan(x)
```

Description

`[pathgains,sampletimes,stateseries] = chan()` produces path gains, `pathgains`, sample times, `sampletimes`, and state series, `stateseries` for an ITU-R P.681-11 LMS flat fading channel.

In this case, the System object acts as a source of path gains, sample times, and state series.

Specify the duration of the fading process by using the NumSamples property. Specify the datatype of outputs using the OutputDataType property.

Note This syntax is applicable when you set the ChannelFiltering property to `false`.

`[y,pathgains,sampletimes,stateseries] = chan(x)` filters the input signal, `x`, through an ITU-R P.681-11 LMS flat fading channel, and returns the output channel-impaired signal in `y`, in addition to the outputs in the previous syntax.

Note This syntax is applicable when you set the ChannelFiltering property to `true`.

Input Arguments

x — Input signal

N_S-by-1 vector

Input signal, specified as an N_S-by-1 vector, where N_S is the number of input samples.

Data Types: `single` | `double`
Complex Number Support: Yes

Output Arguments

y — Output signal

N_S-by-1 vector

Output signal, returned as an N_S-by-1 vector of complex values with the same data precision and length as the input signal `x`. N_S is the number of input samples.

Data Types: `single` | `double`
Complex Number Support: Yes

`pathgains` — Channel path gains of fading process

N_S-by-1 vector

Channel path gains of fading process, returned as an N_S-by-1 vector of complex values.

- When you set the ChannelFiltering property to `true`, `pathgains` is of the same data precision as the input signal `x`, and N_S is the number of input samples.
- When you set the ChannelFiltering property to `false`, `pathgains` is of the same data precision as the OutputDataType property and N_S is equal to the NumSamples property.

Data Types: `single` | `double`
Complex Number Support: Yes

`sampletimes` — Sample times of channel snapshots

N_S-by-1 vector

Sample times of channel snapshots, returned as an N_S-by-1 vector.

- When you set the ChannelFiltering property to `true`, `sampletimes` is of the same data precision as the input signal `x`, and N_S is the number of input samples.
- When you set the ChannelFiltering property to `false`, `sampletimes` is of the same data precision as the OutputDataType property and N_S is equal to the NumSamples property.

Data Types: `single` | `double`

`stateseries` — State series of channel

N_S-by-1 vector

State series of the channel, returned as an N_S-by-1 vector. Each value of this vector describes the state in which channel is present for that channel snapshot. A value of `0` represents bad state. A value of `1` represents good state. A value between 0 and 1 represents a state transition.

- When you set the ChannelFiltering property to `true`, `stateseries` is of the same data precision as the input signal `x`, and N_S is the number of input samples.
- When you set the ChannelFiltering property to `false`, `stateseries` is of the same data precision as the OutputDataType property and N_S is equal to the NumSamples property.

Data Types: `single` | `double`

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 `p681LMSChannel`

info	Characteristic information about object

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Transmit Signal Through P.681-11 LMS Channel

Create and Configure the Channel

Create an ITU-R P.681-11 LMS channel and configure it for a suburban scenario with a carrier frequency of 20 GHz and an elevation angle of 50 degrees. Set the sample rate to 6000 kHz.

Specify the mobile terminal speed to 50 m/s, with an azimuth orientation of 20 degrees.

```
chan = p681LMSChannel;
chan.SampleRate = 6e6;           % Hz
chan.CarrierFrequency = 20e9;    % Hz
chan.ElevationAngle = 50;        % degrees
chan.Environment = "Suburban";
chan.MobileSpeed = 50;           % m/s
chan.AzimuthOrientation = 20;    % degrees
```

Display the channel characteristics.

```
disp(chan)
```

```
  p681LMSChannel with properties:

            SampleRate: 6000000
          InitialState: "Good"
```

```
        CarrierFrequency: 2.0000e+10
          ElevationAngle: 50
             MobileSpeed: 50
      AzimuthOrientation: 20
             Environment: "Suburban"
        ChannelFiltering: true

    Use get to show all properties
```

Transmit Input Signal Through Channel

Set the random number generation seed as default.

```
rng("default")
```

Generate a random QPSK-modulated input signal.

```
numSamples = 6e6;
txWaveform = pskmod(randi([0 3],numSamples,1),4); % Modulation order = 4
```

Filter the signal through the channel.

```
[rxWaveform,pathGains,sampleTimes,stateSeries] = chan(txWaveform);
```

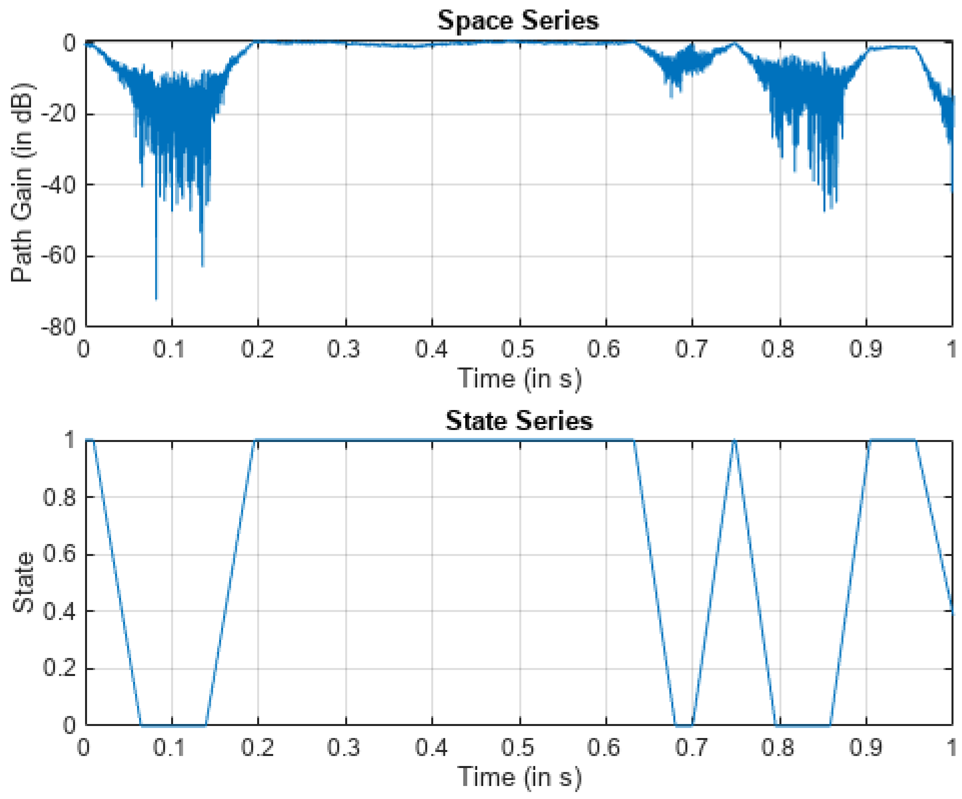
Visualize Space Series and State Series

Plot the space series as a function of time.

```
figure                                      % Create figure window
subplot(2,1,1)
plot(sampleTimes,20*log10(abs(pathGains)))
title('Space Series')
xlabel('Time (in s)')
ylabel('Path Gain (in dB)')
grid on
```

Plot the state series as a function of time.

```
subplot(2,1,2)
plot(sampleTimes,stateSeries)
title('State Series')
xlabel('Time (in s)')
ylabel('State')
grid on
```



Plot Doppler Spectrum for P.681-11 LMS Channel

Define the channel configuration using a `p681LMSChannel` System object and specify its properties.

Set the visualization as Doppler spectrum and disable the channel filtering.

```
chan = p681LMSChannel;
chan.SampleRate = 450000;                % Hz
chan.CarrierFrequency = 11e9;            % Hz
chan.ElevationAngle = 50;                % degrees
chan.MobileSpeed = 20;                   % m/s
chan.Visualization = "Doppler spectrum";
chan.ChannelFiltering = false;
chan.NumSamples = 4e7;
```

Display the channel characteristics.

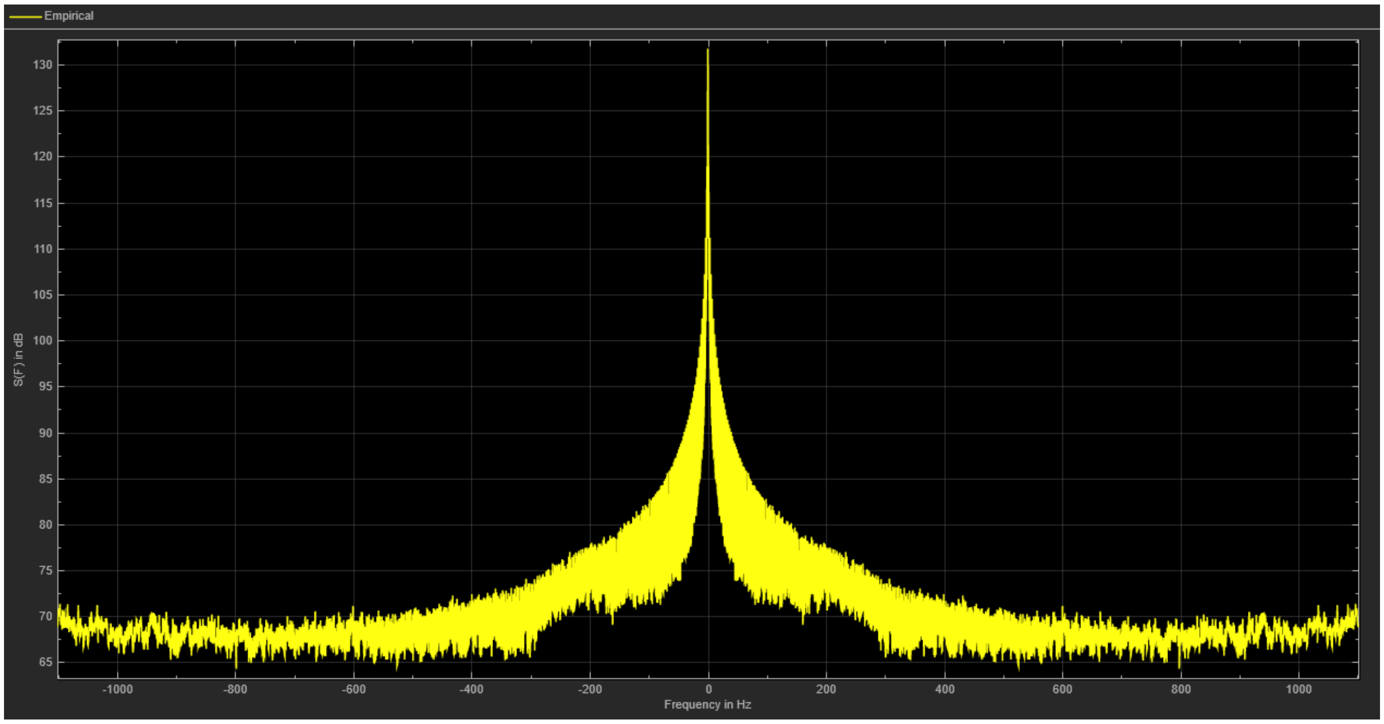
```
disp(chan)
```

```
  p681LMSChannel with properties:

            SampleRate: 450000
          InitialState: "Good"
      CarrierFrequency: 1.1000e+10
        ElevationAngle: 50
           MobileSpeed: 20
```

```
    AzimuthOrientation: 0
           Environment: "Urban"
      ChannelFiltering: false
            NumSamples: 40000000
        OutputDataType: "double"

  Use get to show all properties
```

Get the path gains, sample times, and state series of the channel. Also, observe the Doppler spectrum.

```
[pathGains,sampleTimes,stateSeries] = chan();
```



Get P.681-11 LMS Channel Information

Get channel information from a `p681LMSChannel` System object by using the `info` object function.

Create an ITU-R P.681-11 LMS channel System object and specify its properties.

```
chan = p681LMSChannel;
chan.SampleRate = 10e3;              % Hz
chan.MobileSpeed = 2;                % m/s
chan.Environment = "RuralWooded";
disp(chan)

  p681LMSChannel with properties:

            SampleRate: 10000
          InitialState: "Good"
```

```
      CarrierFrequency: 2.2000e+09
        ElevationAngle: 45
           MobileSpeed: 2
    AzimuthOrientation: 0
           Environment: "RuralWooded"
      ChannelFiltering: true

  Use get to show all properties
```

QPSK-modulate a random input signal, and then pass it through the channel.

```
numSamples = 2e4;
txWaveform = pskmod(randi([0 3],numSamples,1),4);
[rxWaveform,pathGains,sampleTimes,stateSeries] = chan(txWaveform);
```

Get the characteristic information about the P.681-11 LMS channel.

```
info(chan)
```

```
ans = struct with fields:
                  PathDelays: 0
          ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
         NumSamplesProcessed: 20000
```

Transmit another QPSK-modulated random input signal through the channel

```
numSamples2 = 3e4;
txWaveform2 = pskmod(randi([0 3],numSamples2,1),4);
[rxWaveform2,pathGains2,sampleTimes2,stateSeries2] = chan(txWaveform2);
```

Observe the change in number of samples processed.

```
info(chan)
```

```
ans = struct with fields:
                  PathDelays: 0
          ChannelFilterDelay: 0
    ChannelFilterCoefficients: 1
         NumSamplesProcessed: 50000
```

Algorithms

Doppler Phenomena

To calculate the Doppler spread and Doppler shift due to the movement of the mobile on Earth, refer to these formulas.

- The maximum Doppler spread due to mobile movement is given by the following formula:

  $F_{mob\_max\_spread} = (v_{mob} * f_c) / c$

  where:

  - v_{mob} is the speed of the mobile terminal on Earth in m/s, specified as the MobileSpeed property.
  - f_c is the carrier frequency in Hz, specified by the CarrierFrequency property.
  - c is the speed of light in free space in m/s, specified as `physconst('lightspeed')`.
- The Doppler shift due to mobile movement is given by the following formula:

  $fd_{mob} = F_{mob\_max\_spread}$ * `cosd`(θ) * `cosd`(φ)

  where:

  - $F_{mob\_max\_spread}$ is the maximum Doppler spread due to mobile movement.
  - θ is the path elevation angle to the satellite in degrees, specified by the ElevationAngle property.
  - φ is the azimuth orientation in degrees, specified by the AzimuthOrientation property.

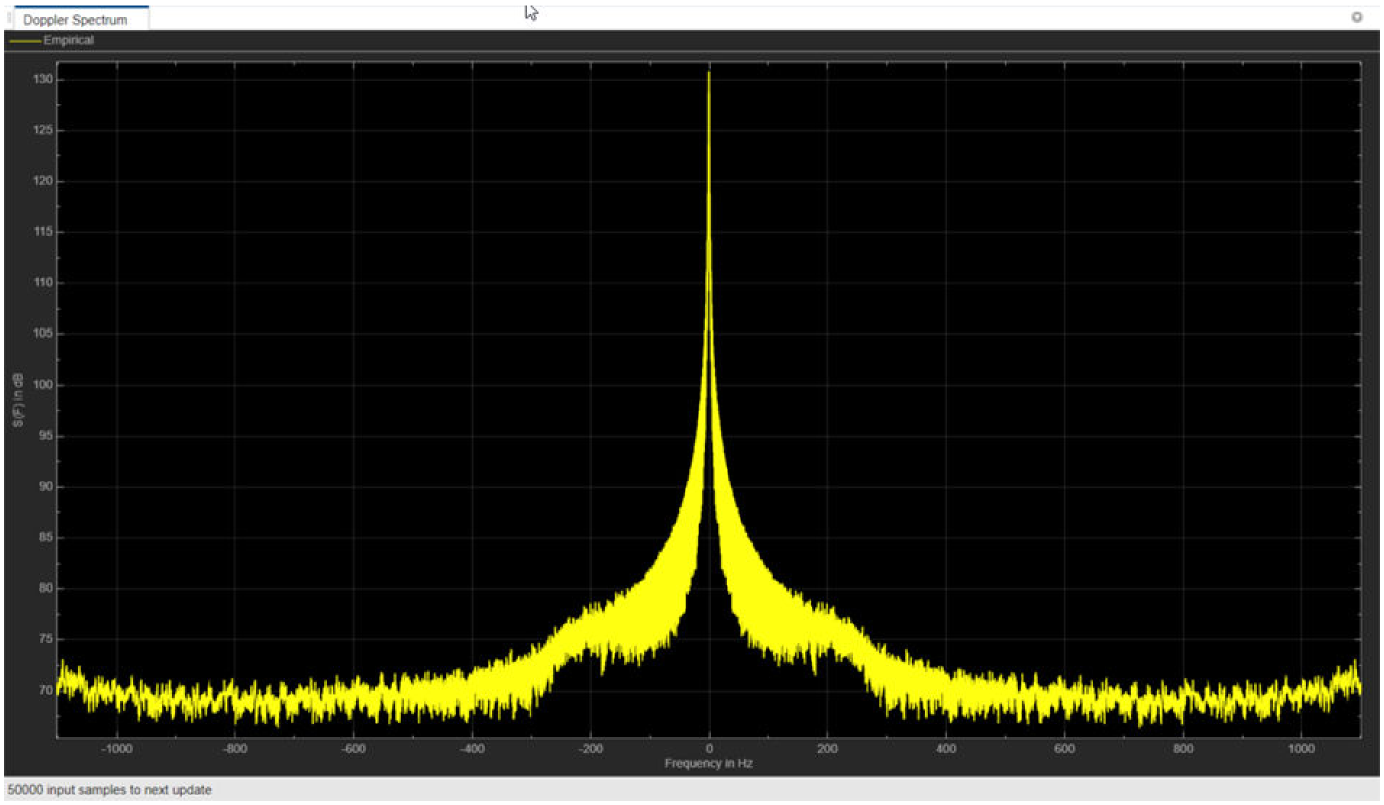
The maximum Doppler shift caused by the movement of the mobile must be less than one-tenth of Sample Rate property.

Channel Visualization

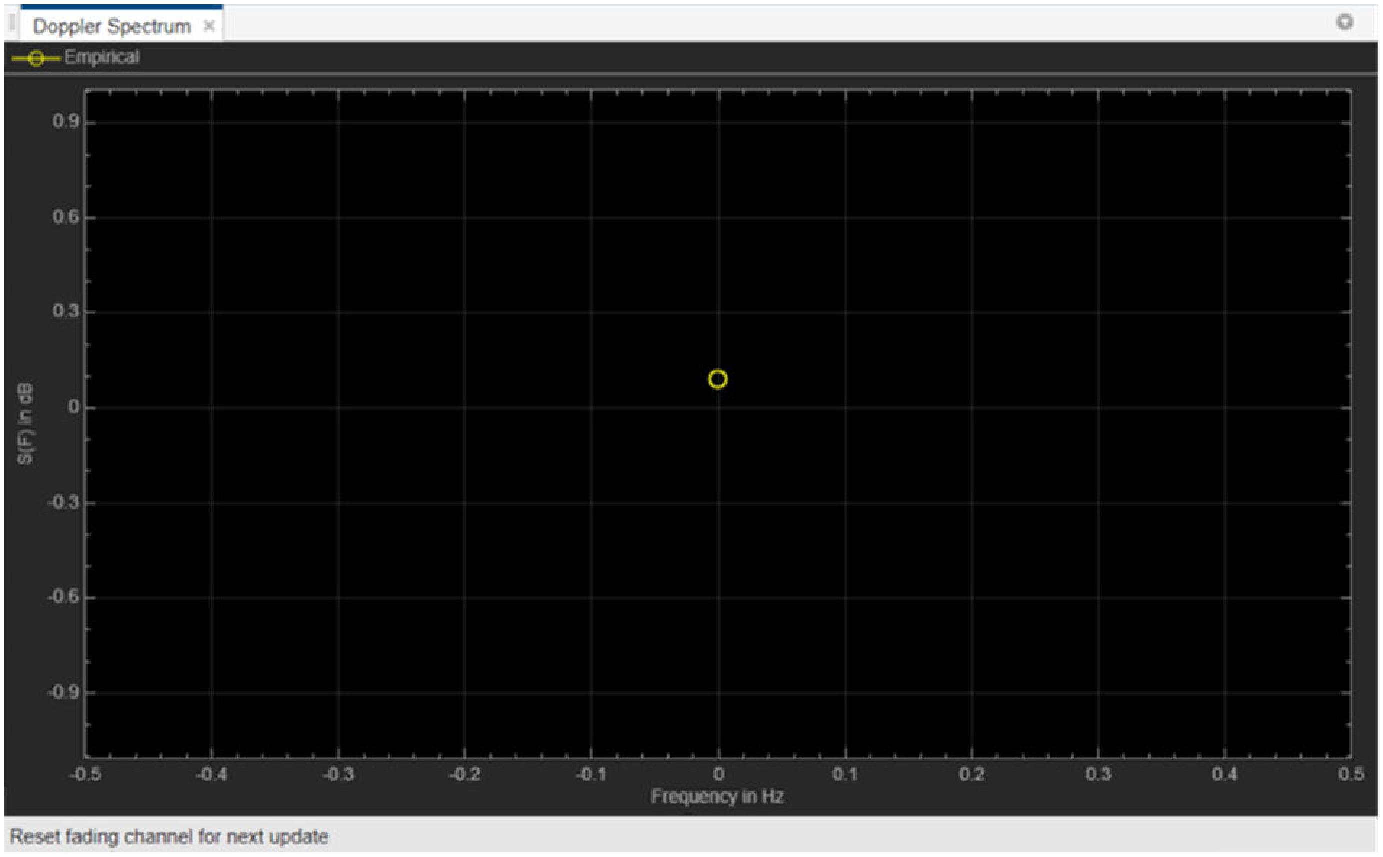
The `p681LMSChannel` System object enables visualization of the channel impulse response, frequency response, and Doppler spectrum.

- The Doppler spectrum plot displays the empirically determined spectrum from the path gains of the channel. The Doppler spectrum values are in dB.

  When there is no mobile movement, the channel is static channel. The empirical data is displayed as a line for the case of nonstatic channels and as a point for static channels. Before the empirical plot is updated, the internal buffer must be completely filled with the required number of channel samples. The empirical plot is the running mean of the spectrum that is calculated from each full buffer. For nonstatic channels, the number of input samples that is needed before the next update is displayed in the status bar located at the bottom of plot. The number of samples that is needed is a function of the sample rate and the maximum Doppler shift depending on mobile movement. For static channels, the text `"Reset fading channel for next update"` is displayed.



Nonstatic Channel



Reset fading channel for next update

Static Channel

- For channel impulse and frequency visualizations, see "Channel Visualization".

Version History

Introduced in R2022a

References

[1] ITU-R Recommendation P.681-11 (08/2019). "Propagation data required for the design systems in the land mobile-satellite service." P Series; Radiowave propagation.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

Usage notes and limitations:

- Code generation is available only when the `Visualization` property is `"Off"`.
- See "System Objects in MATLAB Code Generation" (MATLAB Coder).

See Also

`etsiRicianChannel` | `comm.RayleighChannel`

Topics

"Simulate and Visualize Land Mobile-Satellite Channel"

gnssSignalAcquirer

Acquire GNSS signals

Description

The `gnssSignalAcquirer` System object detects signals emanating from a given global navigation satellite system (GNSS) satellite constellation. The object also estimates the coarse values of the code-phase offset and the frequency offset.

The object supports these satellite systems that use C/A-code for spreading their signals.

- Global Positioning System (GPS)
- Navigation with Indian Constellation (NavIC)
- Quasi-Zenith Satellite System (QZSS)

To acquire a GNSS signal :

1. Create the `gnssSignalAcquirer` 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

```
gsa = gnssSignalAcquirer
gsa = gnssSignalAcquirer(Name=Value)
```

Description

`gsa = gnssSignalAcquirer` creates a default GNSS signal acquirer System object.

`gsa = gnssSignalAcquirer(Name=Value)` sets properties using one or more optional name-value arguments. For example, `gnssSignalAcquirer(GNSSSignalType="QZSS C/A")` sets the type of the GNSS signal to `"QZSS C/A"`.

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.

`GNSSSignalType` — Type of GNSS signal to acquire
`"GPS C/A"` (default) | `"NavIC L5 C/A"` | `"NavIC S C/A"` | `"QZSS C/A"`

Type of GNSS signal to acquire, specified as one of these options.

- `"GPS C/A"`
- `"NavIC L5 C/A"`
- `"NavIC S C/A"`
- `"QZSS C/A"`

Data Types: `char` | `string`

`SampleRate` — Sample rate of input signal
`10.23e6` (default) | positive numeric scalar

Sample rate of the input signal, specified as a positive numeric scalar.

Data Types: `single` | `double`

`IntermediateFrequency` — Intermediate frequency of the input signal
0 (default) | nonnegative numeric scalar

Intermediate frequency of the input signal in hertz, specified as a nonnegative numeric scalar.

Data Types: `single` | `double`

`FrequencyRange` — Range of the frequency search
`[-10e3 10e3]` (default) | two-element numeric vector

Range of the frequency search, specified as a two-element numeric vector.

Data Types: `single` | `double`

`FrequencyResolution` — Resolution of the frequency search
`500` (default) | positive numeric scalar

Resolution of the frequency search, specified as a positive numeric scalar.

Data Types: `single` | `double`

`DetectionThresholdFactor` — Detection threshold factor
`1.9` (default) | numerical scalar greater than or equal to `1`

Detection threshold factor, specified as a numerical scalar greater than or equal to `1`.

Data Types: `single` | `double`

Usage

Syntax

```
y = gsa(x,prn)
[y,c] = gsa(x,prn)
```

Description

`y = gsa(x,prn)` searches the input signal `x` for a satellite with a pseudo-random noise (PRN) index, `prn`, and returns the information for that PRN ID about the frequency offset, code phase offset, and detection status of a satellite with specified PRN ID.

`[y,c] = gsa(x,prn)` additionally returns a 3-D matrix, `c`, containing the correlation values of the searched satellites.

Input Arguments

`x` — Input signal
vector

Input signal, specified as a column vector of length `gsa.SampleRate*10e-3`, where `10e-3` represents one millisecond of the data.

`prn` — Satellite PRN IDs
M-element vector

Satellite PRN IDs, specified as an*M*-element vector. *M* is the number of PRN IDs to search for on the input signal.

Output Arguments

`y` — PRN ID Information
M-by-4 table

PRN ID information, returned as an *M*-by-`4` table. The table contains these columns.

- `PRNID` — The PRN ID corresponding to the information in the other columns.
- `FrequencyOffset` — Coarse estimate of the frequency offset for the PRN ID
- `CodePhaseOffset` — Coarse delay, in number of C/A-code chips, for the PRN ID.
- `IsDetected` — Detection status of the PRN ID, returned as numeric `1` or `0`.

The order of the PRN IDs in the first column of the table differs from the order of the PRN IDs in the input argument `prn`. The `step` function arranges the PRN IDs in the table in decreasing order of maximum correlation value.

`c` — Correlation value of searched satellites
P-by-*Q*-by-*R*

Correlation value of the searched satellites, returned as a *P*-by-*Q*-by-*R* array.

- *P* is the number of code-phase offsets in the search. To calculate *P*, the step function uses this equation:

  `P = floor(gsa.SampleRate*10e-3)`.

  As a result, the `step` function searches the incoming signal for one block of C/A-code.
- *Q* is the number of frequency offset bins in the search. To calculate *Q*, the step function uses this equation.

  `Q = floor((gsa.FrequencyRange(2) - GSA.FrequencyRange(1))/`
  `gsa.FrequencyResolution) + 1`.

- *R* is the number of searched satellites. *R* is equal to the length of *prn*. The `step` function arranges the *R* layers in decreasing order of maximum correlation value within each layer.

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 `gnssSignalAcquirer`

info	Characteristic information about object

Common to All System Objects

step	Run System object algorithm
release	Release resources and allow changes to System object property values and input characteristics
clone	Create duplicate System object
isLocked	Determine if System object is in use
reset	Reset internal states of System object

Examples

Acquire GPS Baseband Signal

Acquire a GPS baseband signal at a sampling rate of `10.23` MHz and visualize the correlation plot.

Load a precomputed GPS waveform.

```
load gnssWaveforms
```

Initialize the GNSS signal acquirer object.

```
gsa = gnssSignalAcquirer
```

```
gsa = 
  gnssSignalAcquirer with properties:

              GNSSSignalType: "GPS C/A"
                  SampleRate: 10230000
       IntermediateFrequency: 0
              FrequencyRange: [-10000 10000]
         FrequencyResolution: 500
    DetectionThresholdFactor: 1.9000
```

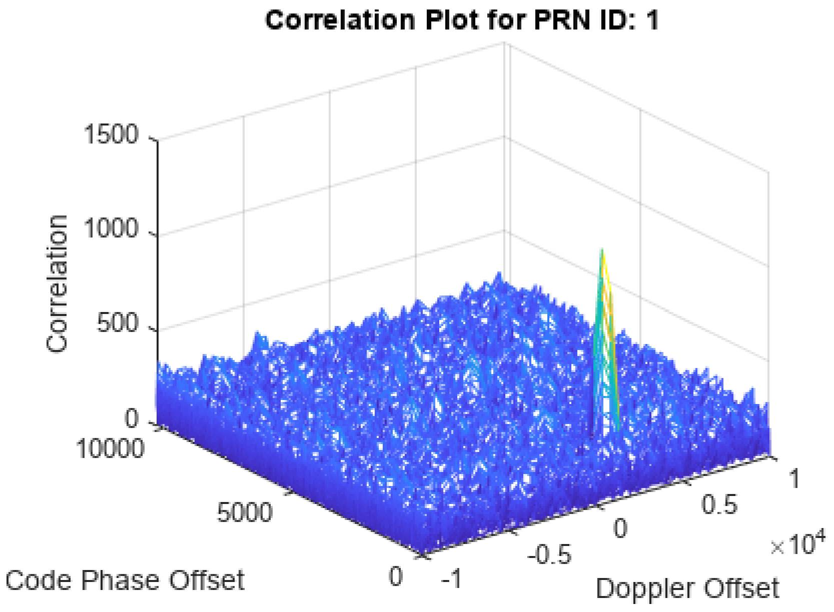
Search for 32 GPS satellite PRN IDs, and display the frequency offset, code-phase offset, and the detection status of a satellite for each PRN ID.

```
[acqInfo,corVal] = gsa(gpsBBWaveform,1:32);
display(acqInfo)
```

```
acqInfo=32×4 table
    PRNID    FrequencyOffset    CodePhaseOffset    IsDetected
    _____      _________________      _________________      ___________

      1              2000                   111               true
      5              9000                   222               true
     19              7000                   444               true
     12              5000                   333               true
     25              4500                   555               true
     30              1000                   666               false
     17             -5500                  25.1               false
     18              -500                   666               false
      6             -4500                   134               false
     21              4500                   979               false
     16             -1000                   677               false
     27             -9500                 292.4               false
      7             -8500                   343               false
     14              9000                   790               false
      2             -4000                    51               false
      3             -9500                   657               false
      ⋮
```

Visualize a surface plot of the 2-D correlation values for the satellite that has the largest correlation magnitude.

```
freqRange= gsa.FrequencyRange;                                            % Range of the 
stepSize = gsa.FrequencyResolution;                                       % Step size of 
satIndex = 1;                                                             % Visualize 1st
mesh(freqRange(1):stepSize:freqRange(2),0:size(corVal,1)-1,corVal(:,:,satIndex)) % Surface plot
xlabel("Doppler Offset")
ylabel("Code Phase Offset")
zlabel("Correlation")
title("Correlation Plot for PRN ID: " + acqInfo.PRNID(satIndex))
```



Acquire GPS IF Signal

Acquire a GPS intermediate frequency (IF) signal at `38.192` MHz.

Load a precomputed GPS waveform.

```
load gnssWaveforms
```

Initialize the GNSS signal acquirer object.

```
gsa = gnssSignalAcquirer(IntermediateFrequency=10e6,SampleRate=38.192e6);
```

Search for 32 GPS satellite PRN IDs.

```
Information = gsa(gpsIFWaveform,1:32); % Information about frequency Offset, code-phase Offset, a
gsaInfo = info(gsa)                   % Characteristic information
```

```
gsaInfo = struct with fields:
    ReferenceNoiseLevel: 1.2586e+03
```

Version History

Introduced in R2023a

References

[1] Science Applications International Corporation (SAIC). *NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*. IS-GPS-200N. El Segundo, CA: SAIC, August 22, 2022

[2] Indian Space Research Organization (ISRO). *Signal in Space ICD for Standard Positioning Service*. ISRO-IRNSS-ICD-SPS-1.1. Bangalore: ISRO, August 2017.

[3] Quasi-Zenith Satellite System Services. *Quasi-Zenith Satellite System Interface Specification Satellite Positioning, Navigation and Timing Service*. IS-QZSS-PNT-004. Cabinet Office, Government of Japan (CAO), January 25, 2021.

[4] Ward, P.W. “GPS Receiver Search Techniques.” In *Proceedings of Position, Location and Navigation Symposium - PLANS* ’96, 604–11. Atlanta, GA, USA: IEEE, 1996. https://doi.org/10.1109/PLANS.1996.509134.

[5] Kaplan, Elliott D., and C. Hegarty, eds. *Understanding GPS/GNSS: Principles and Applications*. Third edition. GNSS Technology and Applications Series. Boston; London: Artech House, 2017.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using MATLAB® Coder™.

See Also

Functions
`gnssCACode`

Objects
`gpsPCode`

Topics
“GPS Receiver Acquisition and Tracking Using C/A-Code”