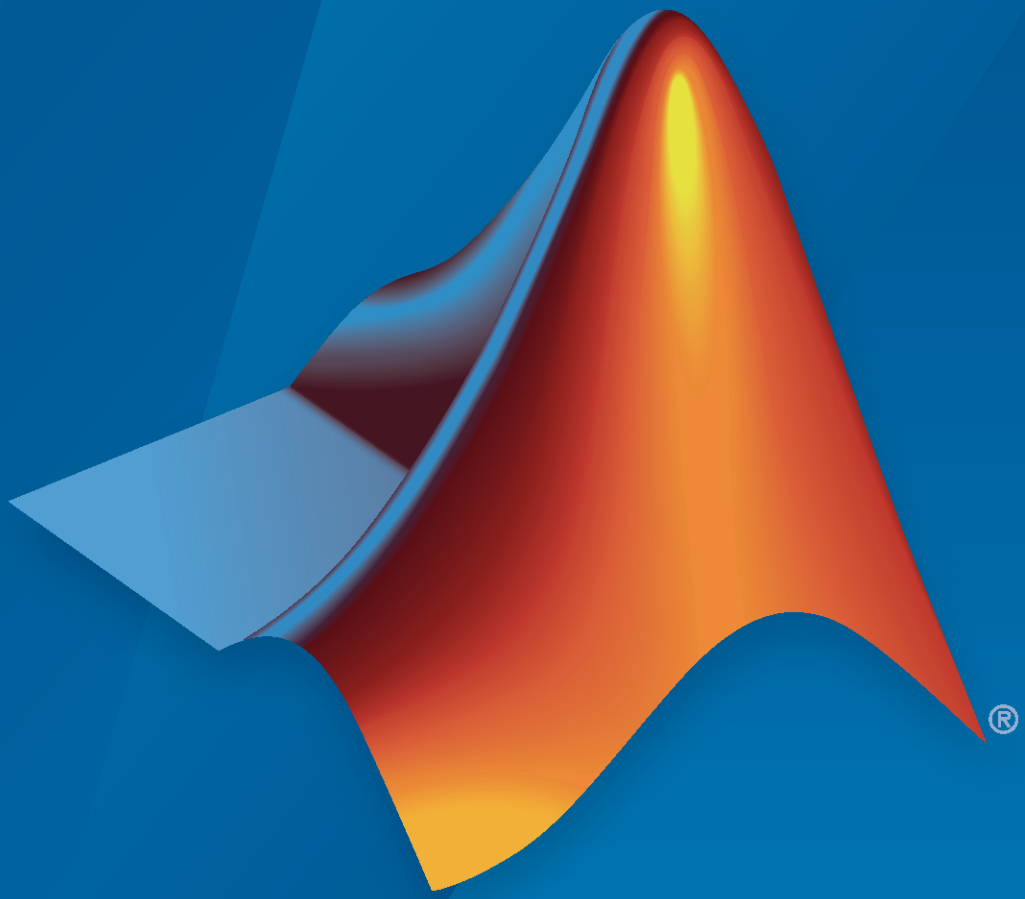
Satellite Communications Toolbox

User's Guide



MATLAB®

R2021b

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 User's Guide

Revision History

March 2021	Online only	New for Version 1.0 (Release 2021a)
September 2021	Online only	Revised for Version 1.1 (Release 2021b)

Contents

1 Satellite Scenario Generation

2 Signal Transmission

3 RF Propagation and Channel Models

4 End-to-End Simulation

5 Code Generation and Deployment

Satellite Scenario Generation

Multi-Hop Satellite Communications Link Between Two Ground Stations

This example demonstrates how to set up a multi-hop satellite communications link between two ground stations. The first ground station is located in India (Ground Station 1), and the second ground station is located in Australia (Ground Station 2). The link is routed via two satellites (Satellite 1 and Satellite 2). Each satellite acts as a regenerative repeater. A regenerative repeater receives an incoming signal, and then demodulates, remodulates, amplifies, and retransmits the received signal. The times over the course of a day during which Ground Station 1 can send data to Ground Station 2 are determined.

Create Satellite Scenario

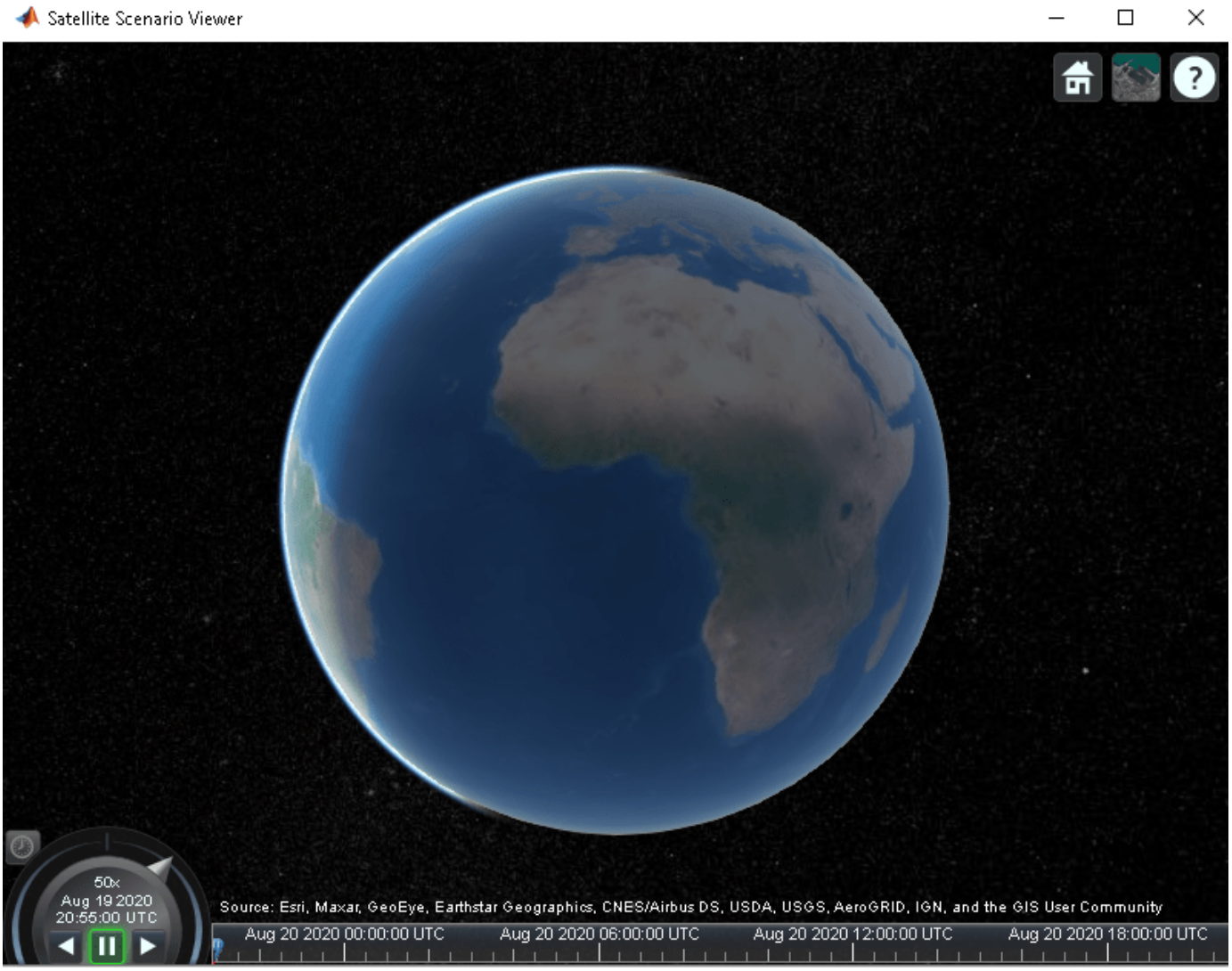
Use `satelliteScenario` to create a satellite scenario. Use `datetime` to define the start time and stop time of the scenario. Set the sample time to 60 seconds.

```
startTime = datetime(2020,8,19,20,55,0); % 19 August 2020 8:55 PM UTC
stopTime = startTime + days(1);          % 20 August 2020 8:55 PM UTC
sampleTime = 60;                         % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Launch Satellite Scenario Viewer

Use `satelliteScenarioViewer` to launch a Satellite Scenario Viewer.

```
satelliteScenarioViewer(sc);
```



Add the Satellites

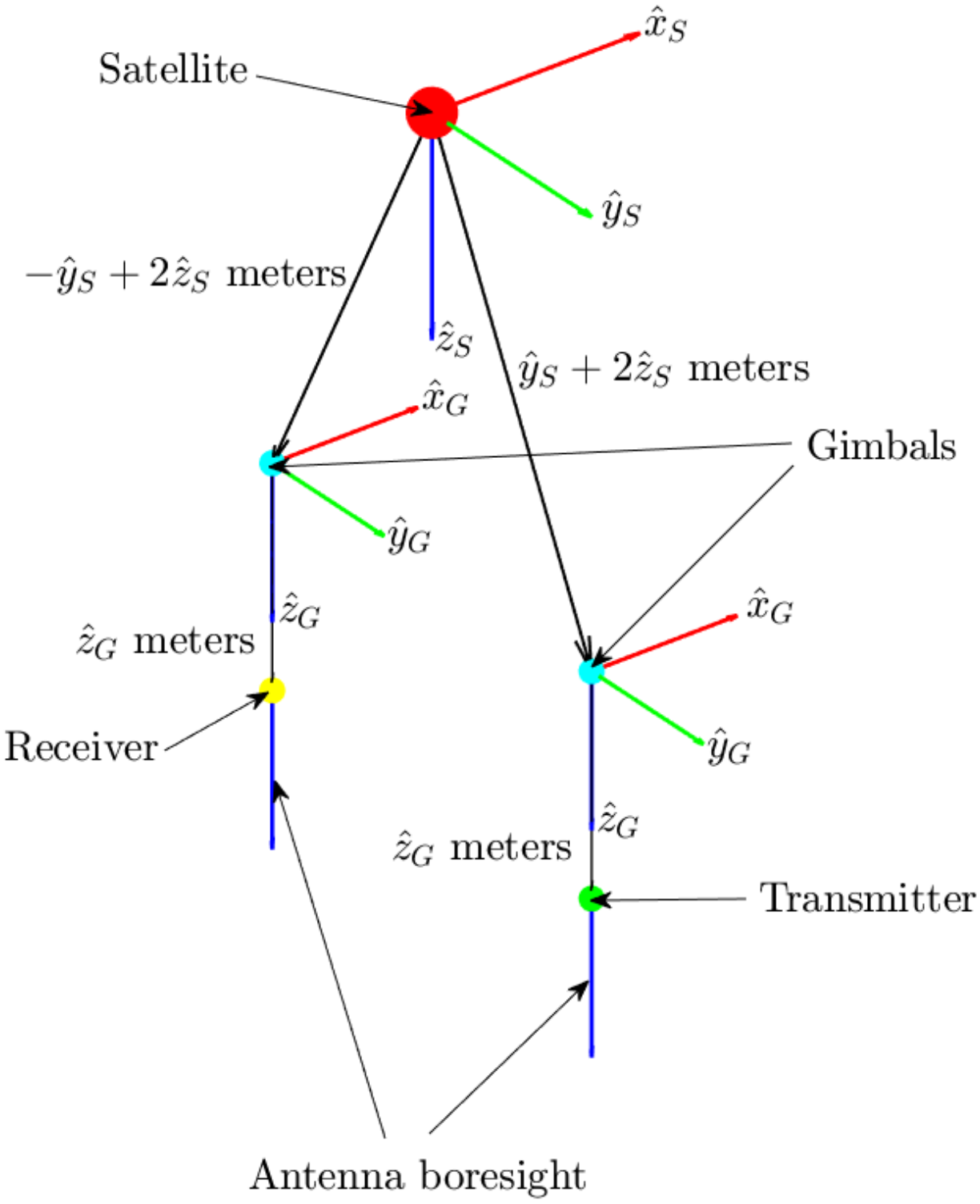
Use `satellite` to add Satellite 1 and Satellite 2 to the scenario by specifying their Keplerian orbital elements corresponding to the scenario start time.

```
semiMajorAxis = 10000000;          % meters
eccentricity = 0;
inclination = 0;                   % degrees
rightAscensionOfAscendingNode = 0; % degrees
argumentOfPeriapsis = 0;           % degrees
trueAnomaly = 0;                   % degrees
sat1 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
```

```
    "Name","Satellite 1", ...
    "OrbitPropagator","two-body-keplerian");

semiMajorAxis = 10000000;            % meters
eccentricity = 0;
inclination = 30;                    % degrees
rightAscensionOfAscendingNode = 120; % degrees
argumentOfPeriapsis = 0;             % degrees
trueAnomaly = 300;                   % degrees
sat2 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
    "Name","Satellite 2", ...
    "OrbitPropagator","two-body-keplerian");
```

Add Gimbals to the Satellites

Use `gimbal` to add gimbals to the satellites. Each satellite consists of two gimbals on opposite sides of the satellite. One gimbal holds the receiver antenna and the other gimbal holds the transmitter antenna. The mounting location is specified in cartesian coordinates in the body frame of the satellite, which is defined by $(\hat{x}_S, \hat{y}_S, \hat{z}_S)$, where $\hat{x}_S$, $\hat{y}_S$ and $\hat{z}_S$ are the roll, pitch and yaw axes respectively, of the satellite. The mounting location of the gimbal that holds the receiver is $-\hat{y}_S + 2\hat{z}_S$ meters and that of the gimbal that holds the transmitter is $\hat{y}_S + 2\hat{z}_S$ meters, as illustrated in the diagram below.


Satellite
$\hat{x}_S$
$\hat{y}_S$
$-\hat{y}_S + 2\hat{z}_S$ meters
$\hat{z}_S$
$\hat{y}_S + 2\hat{z}_S$ meters
$\hat{x}_G$
Gimbals
$\hat{y}_G$
$\hat{z}_G$
$\hat{z}_G$ meters
$\hat{x}_G$
Receiver
$\hat{y}_G$
$\hat{z}_G$
$\hat{z}_G$ meters
Transmitter
Antenna boresight

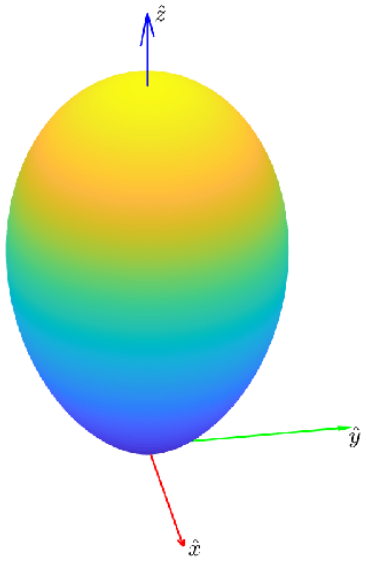
```
gimbalSat1Tx = gimbal(sat1, ...
    "MountingLocation",[0;1;2]);  % meters
gimbalSat2Tx = gimbal(sat2, ...
    "MountingLocation",[0;1;2]);  % meters
gimbalSat1Rx = gimbal(sat1, ...
    "MountingLocation",[0;-1;2]); % meters
gimbalSat2Rx = gimbal(sat2, ...
    "MountingLocation",[0;-1;2]); % meters
```

Add Receivers and Transmitters to the Gimbals

Each satellite consists of a receiver and a transmitter, constituting a regenerative repeater. Use `receiver` to add a receiver to the gimbals `gimbalSat1Rx` and `gimbalSat2Rx`. The mounting location of the receiving antenna with respect to the gimbal is $\hat{z}_G$ meters, as illustrated in the diagram above. The receiver gain to noise temperature ratio is 3dB/K and the required Eb/No is 4 dB.

```
sat1Rx = receiver(gimbalSat1Rx, ...
    "MountingLocation",[0;0;1], ...      % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",4);                   % decibels
sat2Rx = receiver(gimbalSat2Rx, ...
    "MountingLocation",[0;0;1], ...      % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",4);                   % decibels
```

Use `gaussianAntenna` to set the dish diameter of the receiver antennas on the satellites to 0.5 m. A Gaussian antenna has a radiation pattern that peaks at its boresight and decays radial-symmetrically based on a Gaussian distribution while moving away from boresight, as shown in the diagram below. The peak gain is a function of the dish diameter and aperture efficiency.



```
gaussianAntenna(sat1Rx, ...
    "DishDiameter",0.5); % meters
gaussianAntenna(sat2Rx, ...
    "DishDiameter",0.5); % meters
```

Use `transmitter` to add a transmitter to the gimbals `gimbalSat1Tx` and `gimbalSat2Tx`. The mounting location of the transmitting antenna with respect to the gimbal is $\hat{z}_G$ meters, where $(\hat{x}_G, \hat{y}_G, \hat{z}_G)$ define the body frame of the gimbal. The boresight of the antenna is aligned with $\hat{z}_G$. Both satellites transmit with a power of 15 dBW. The transmitter onboard Satellite 1 is used in the crosslink for sending data to Satellite 2 at a frequency of 30 GHz. The transmitter onboard Satellite 2 is used in the downlink to Ground Station 2 at a frequency of 27 GHz.

```
sat1Tx = transmitter(gimbalSat1Tx, ...
    "MountingLocation",[0;0;1], ... % meters
    "Frequency",30e9, ...           % hertz
    "Power",15);                    % decibel watts
sat2Tx = transmitter(gimbalSat2Tx, ...
    "MountingLocation",[0;0;1], ... % meters
    "Frequency",27e9, ...           % hertz
    "Power",15);                    % decibel watts
```

Like the receiver, the transmitter also uses a Gaussian antenna. Set the dish diameter of the transmitter antennas of the satellites to 0.5 m.

```
gaussianAntenna(sat1Tx, ...
    "DishDiameter",0.5); % meters
gaussianAntenna(sat2Tx, ...
    "DishDiameter",0.5); % meters
```

Add the Ground Stations

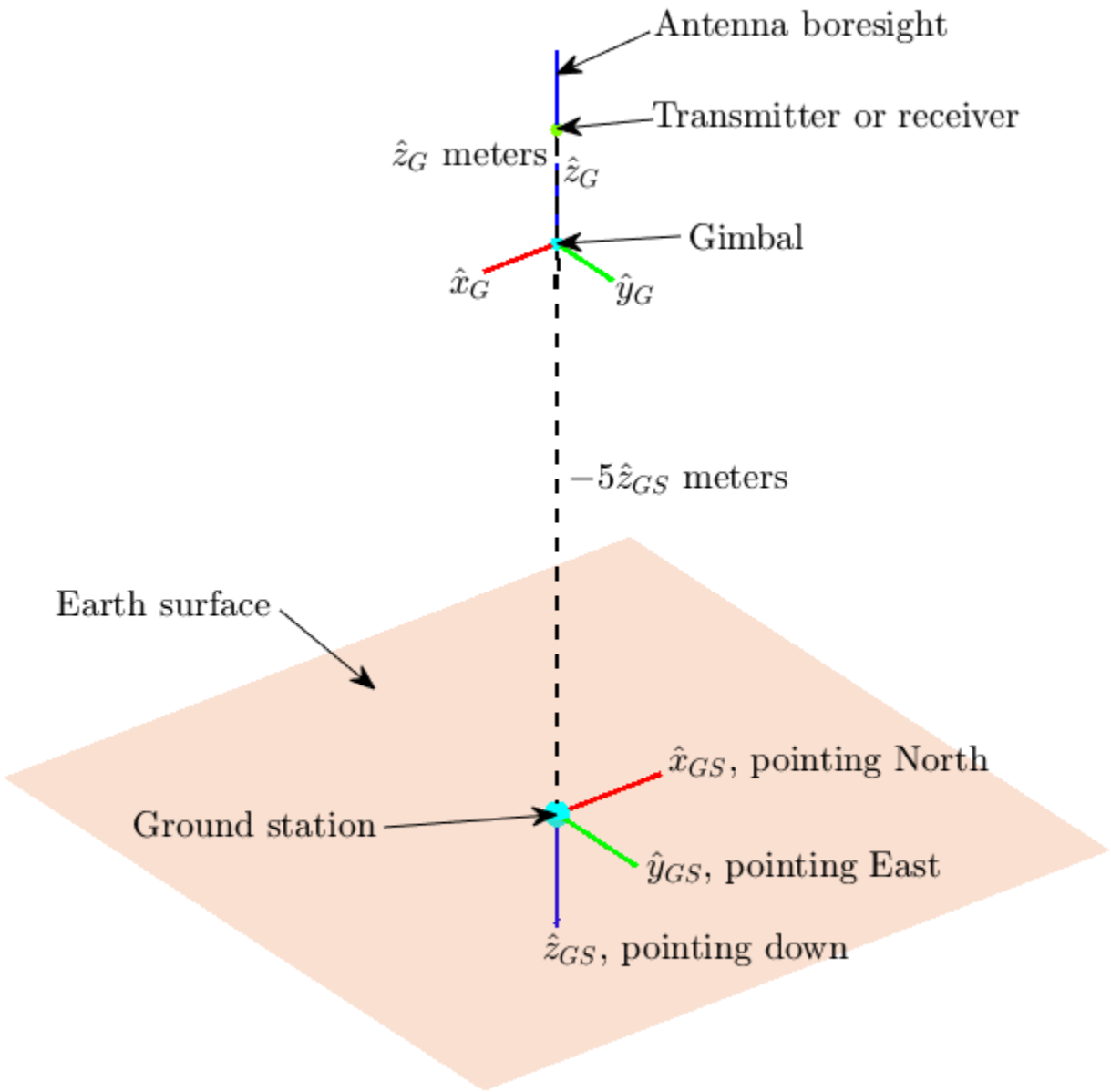
Use `groundStation` to add the ground stations at India (Ground Station 1) and Australia (Ground Station 2).

```
latitude = 12.9436963;  % degrees
longitude = 77.6906568; % degrees
gs1 = groundStation(sc, ...
    latitude, ...
    longitude, ...
    "Name","Ground Station 1");

latitude = -33.7974039;  % degrees
longitude = 151.1768208; % degrees
gs2 = groundStation(sc, ...
    latitude, ...
    longitude, ...
    "Name","Ground Station 2");
```

Add Gimbal to Each Ground Station

Use `gimbal` to add a gimbal to Ground Station 1 and Ground Station 2. The gimbal at Ground Station 1 holds a transmitter, and the gimbal at Ground Station 2 holds a receiver. The gimbals are located 5 meters above their respective ground stations, as illustrated in the diagram below. Consequently, their mounting locations are $-5\hat{z}_{GS}$ meters, where $(\hat{x}_{GS}, \hat{y}_{GS}, \hat{z}_{GS})$ define the body axis of the ground stations. $\hat{x}_{GS}$, $\hat{y}_{GS}$ and $\hat{z}_{GS}$ always point North, East and down respectively. Therefore, the $\hat{z}_{GS}$ component of the gimbals is -5 meters so that they are placed above the ground station and not

below. Furthermore, by default, the mounting angles of the gimbal are such that their body axes $(\hat{x}_G, \hat{y}_G, \hat{z}_G)$ are aligned with the parent (in this case, the ground station) body axes $(\hat{x}_{GS}, \hat{y}_{GS}, \hat{z}_{GS})$. As a result, when the gimbals are not steered, their $\hat{z}_G$ axis points straight down, and so does the antenna attached to it using default mounting angles as well. Therefore, you must set the mounting pitch angle to 180 degrees, so that $\hat{z}_G$ points straight up when the gimbal is not steered.




```
gimbalGs1 = gimbal(gs1, ...
    "MountingAngles",[0;180;0], ... % degrees
    "MountingLocation",[0;0;-5]);   % meters
gimbalGs2 = gimbal(gs2, ...
```

```
    "MountingAngles",[0;180;0], ... % degrees
    "MountingLocation",[0;0;-5]);   % meters
```

Add Transmitters and Receivers to Ground Station Gimbals

Use `transmitter` to add a transmitter to the gimbal at Ground Station 1. The uplink transmitter sends data to Satellite 1 at a frequency of 30 GHz and a power of 30 dBW. The transmitter antenna is mounted at $\hat{z}_G$ meters with respect to the gimbal.

```
gs1Tx = transmitter(gimbalGs1, ...
    "Name","Ground Station 1 Transmitter", ...
    "MountingLocation",[0;0;1], ... % meters
    "Frequency",30e9, ...           % hertz
    "Power",30);                    % decibel watts
```

Use `gaussianAntenna` to set the dish diameter of the transmitter antenna to 2 m.

```
gaussianAntenna(gs1Tx, ...
    "DishDiameter",2); % meters
```

Use `receiver` to add a receiver to the gimbal at Ground Station 2 to receive downlink data from Satellite 2. The receiver gain to noise temperature ratio is 3 dB/K and the required Eb/No is 1 dB. The mounting location of the receiver antenna is $\hat{z}_G$ meters with respect to the gimbal.

```
gs2Rx = receiver(gimbalGs2, ...
    "Name","Ground Station 2 Receiver", ...
    "MountingLocation",[0;0;1], ...      % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",1);                   % decibels
```

Use `gaussianAntenna` to set the dish diameter of the receiver antenna to 2 m.

```
gaussianAntenna(gs2Rx, ...
    "DishDiameter",2); % meters
```

Set Tracking Targets for Gimbals

For the best link quality, the antennas must continuously point at their respective targets. The gimbals can be steered independent of their parents (satellite or ground station), and configured to track other satellites and ground stations. Use `pointAt` to set the tracking target for the gimbals so that:

- The transmitter antenna at Ground Station 1 points at Satellite 1
- The receiver antenna aboard Satellite 1 points at Ground Station 1
- The transmitter antenna aboard Satellite 1 points at Satellite 2
- The receiver antenna aboard Satellite 2 points at Satellite 1
- The transmitter antenna aboard Satellite 2 points at Ground Station 2
- The receiver antenna at Ground Station 2 points at Satellite 2

```
pointAt(gimbalGs1,sat1);
pointAt(gimbalSat1Rx,gs1);
pointAt(gimbalSat1Tx,sat2);
pointAt(gimbalSat2Rx,sat1);
pointAt(gimbalSat2Tx,gs2);
pointAt(gimbalGs2,sat2);
```

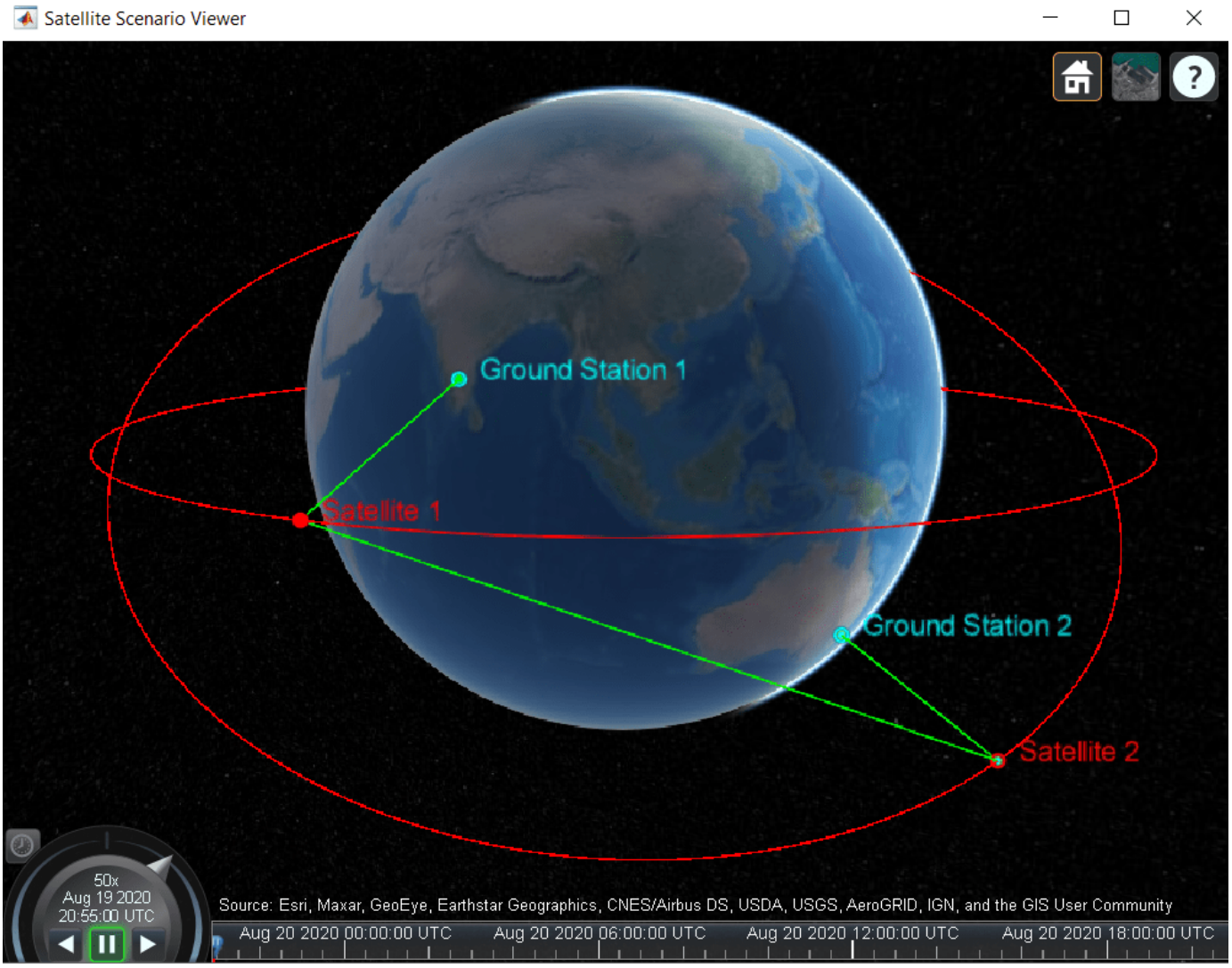
When a target for a gimbal is set, its $\hat{z}_G$ axis will track the target. Since the antenna is on $\hat{z}_G$ and its boresight is aligned with $\hat{z}_G$, the antenna will also track the desired target.

Add Link Analysis and Visualize Scenario

Use `link` to add link analysis to the transmitter at Ground Station 1. The link is of regenerative repeater-type that originates at `gs1Tx` and ends at `gs2Rx`, and is routed via `sat1Rx`, `sat1Tx`, `sat2Rx` and `sat2Tx`.

```
lnk = link(gs1Tx,sat1Rx,sat1Tx,sat2Rx,sat2Tx,gs2Rx);
```

The Satellite Scenario Viewer automatically updates to display the entire scenario. Use the viewer as a visual confirmation that the scenario has been set up correctly. The green lines represent the link and confirm that the link is closed.



Determine Times When Link is Closed and Visualize Link Closures

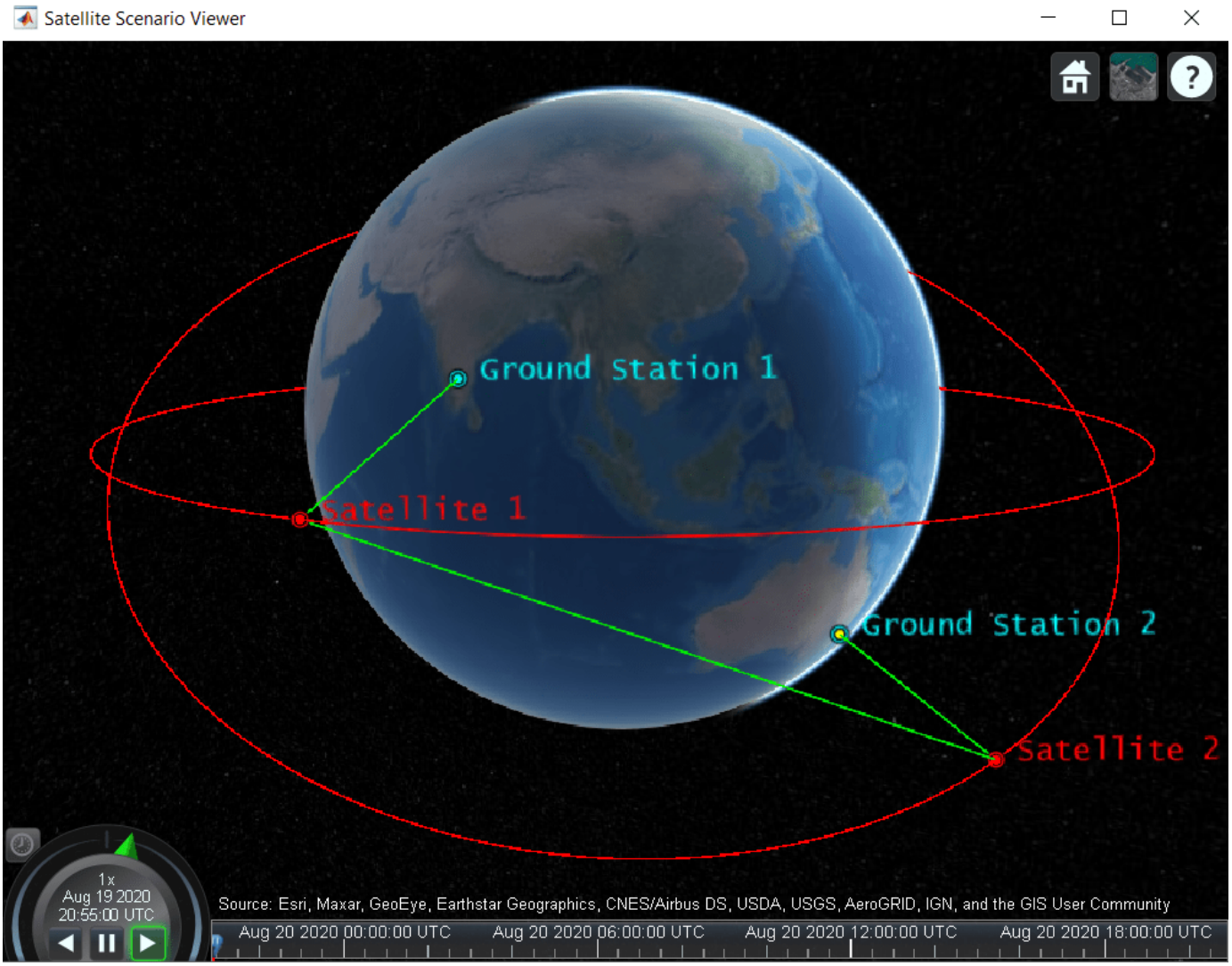
Use the `linkIntervals` method to determine the times when the link is closed. The `linkIntervals` method outputs a table of the start and stop times of link closures that represent the intervals during which Ground Station 1 can send data to Ground Station 2. Source and Target are the first and last nodes in the link. If one of Source or Target is on a satellite, StartOrbit and EndOrbit provide the orbit count of the source or target satellite that they are attached directly or via gimbals, starting from the scenario start time. If both Source and Target are attached to a satellite, StartOrbit and EndOrbit provide the orbit count of the satellite to which Source is attached. Since both Source and Target are attached to ground stations, StartOrbit and EndOrbit are NaN.

```
linkIntervals(lnk)
```

```
ans=6×8 table
                Source                              Target                IntervalNumber         Start
    ____________________________________    ____________________________________    ______________    _____

    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          1           19-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          2           19-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          3           20-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          4           20-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          5           20-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          6           20-Aug-202
```

Use `play` to visualize the scenario simulation from its start time to stop time. The green lines disappear whenever the link cannot be closed.

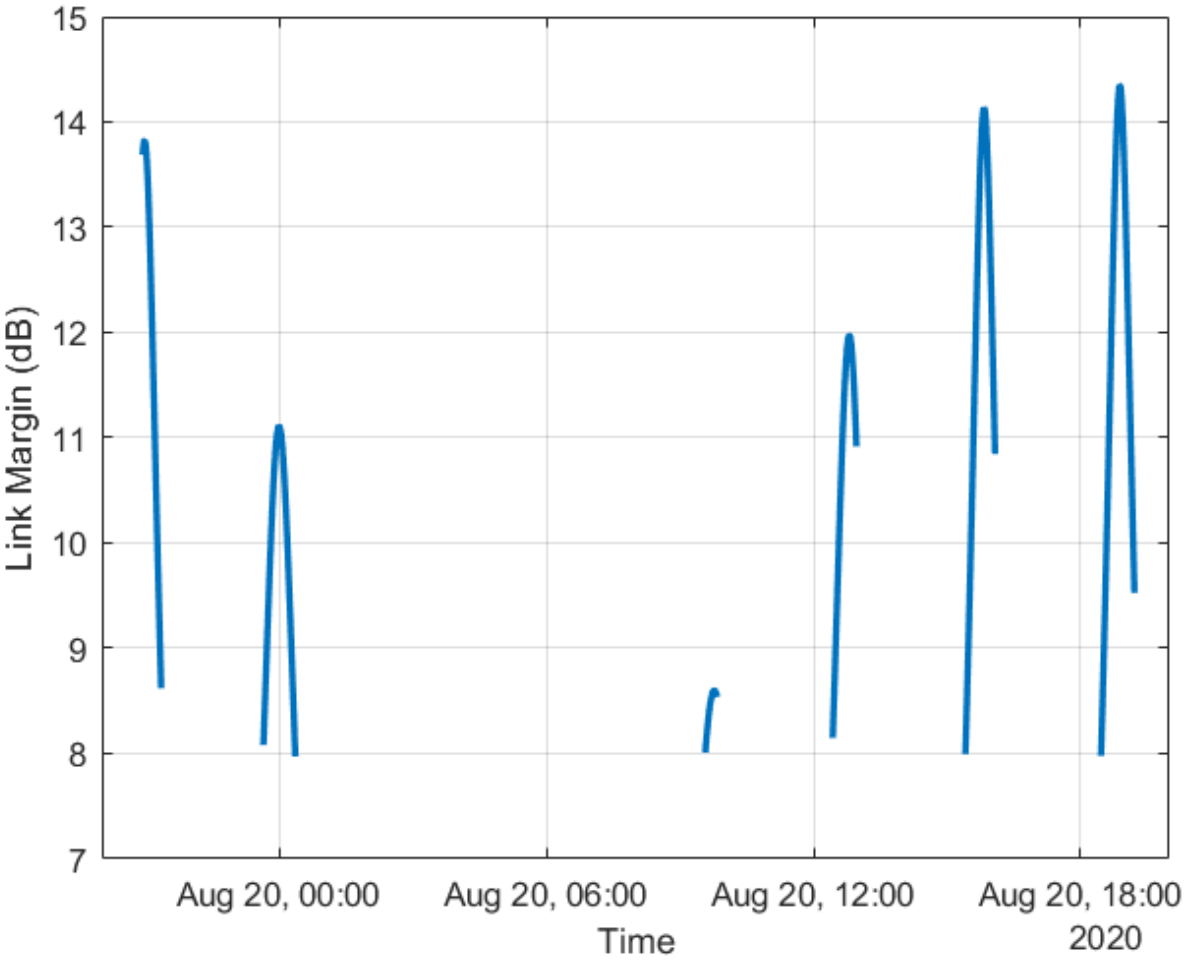
```
play(sc);
```



Plot Link Margin at Ground Station 2

The link margin at a receiver is the difference between the energy per bit to noise power spectral density ratio (Eb/No) at the receiver and its `RequiredEbNo`. For successful link closure, the link margin must be positive at all receiver nodes. Higher the link margin, better the link quality. To calculate the link margin at final node, that is, Ground Station 2 Receiver, use `ebno` to get the Eb/No history at the Ground Station 2 Receiver, and subtract its `RequiredEbNo` from this quantity to get the link margin. Also, use `plot` to plot the calculate link margin.

```
[e, time] = ebno(lnk);
margin = e - gs2Rx.RequiredEbNo;
plot(time,margin,"LineWidth",2);
xlabel("Time");
ylabel("Link Margin (dB)");
grid on;
```



The gaps in the plot imply that the link was broken before reaching the final node in the link, or the line of sight between final node and the node before it, that is, Satellite 2, was broken. At all other times, the link margin is positive. This implies that Satellite 2 Transmitter power and Ground Station 2 Receiver sensitivity are always sufficient. It also implies that the margin is positive at all other hops of the link.

Modify Required Eb/No and Observe Effect on Link Intervals

Increase the `RequiredEbNo` of the receiver at Ground Station 2 from 1 dB to 10 dB and recompute the link intervals. Increasing `RequiredEbNo` essentially reduces the sensitivity of Ground Station 2 Receiver. This negatively impacts the resultant link closure times. The number of closed link intervals drops from six to five, and the duration of the closed link intervals is shorter.

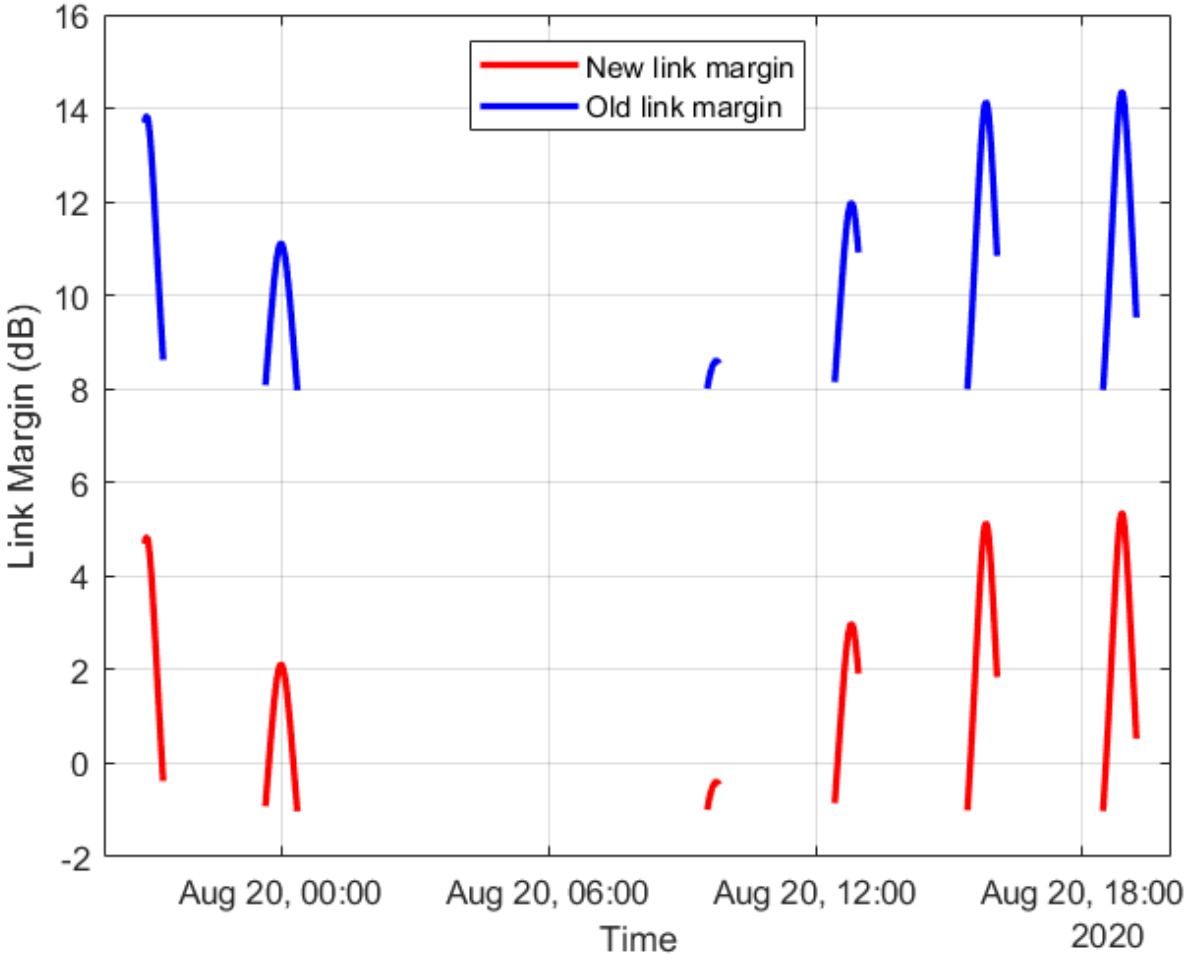
```
gs2Rx.RequiredEbNo = 10; % decibels
linkIntervals(lnk)
```

```
ans=5×8 table
                Source                                Target                IntervalNumber        Start
    ____________________________________    ____________________________    ______________    _________
```

```
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          1           19-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          2           19-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          3           20-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          4           20-Aug-202
    "Ground Station 1 Transmitter"    "Ground Station 2 Receiver"          5           20-Aug-202
```

Additionally, the increase in `RequiredEbNo` negatively impacts the link margin. To observe this, recompute and plot the new link margin, and compare it with the previous plot. The link margin has reduced in general, implying that the link quality has gone down as a result of reducing the sensitivity of the receiver by increasing `RequiredEbNo`. At certain instances, the link margin is negative, signifying that there are times when the link does get broken at Ground Station 2 Receiver, even if it has line of sight to Satellite 2. This implies that the link closure is sometimes limited by the link margin, as opposed to just the line of sight between adjacent nodes.

```
[e, newTime] = ebno(lnk);
newMargin = e - gs2Rx.RequiredEbNo;
plot(newTime,newMargin,"r",time,margin,"b","LineWidth",2);
xlabel("Time");
ylabel("Link Margin (dB)");
legend("New link margin","Old link margin","Location","north");
grid on;
```



Next Steps

This example demonstrated how to set up a multi-hop regenerative repeater-type link and how to determine the times when the link is closed. The link closure times are influenced by the link margin at each receiver in the link. The link margin is the difference between energy per bit to noise power spectral density ratio (Eb/No) at the receiver and the required Eb/No. The Eb/No at a receiver is a function of:

- Orbit and pointing mode of satellites holding the transmitters and receivers
- Position of ground stations holding the transmitters and receivers
- Position, orientation, and pointing mode of the gimbals holding the transmitters and receivers
- Position and orientation of the transmitters and receivers with respect to their parents
- Specifications of the transmitters - power, frequency, bit rate, and system loss
- Specifications of the receivers - gain to noise temperature ratio, required Eb/No, and system loss
- Specifications of the transmitter and receiver antennas, such as dish diameter and aperture efficiency for a Gaussian antenna

Modify the above parameters and observe their impact on the link to perform different types of what-if analyses.

See Also

Objects
`satelliteScenario` | `satellite` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor` | `transmitter` | `receiver`

Functions
`show` | `play` | `hide`

Related Examples

- “Satellite Constellation Access to a Ground Station” on page 1-17
- “Comparison of Orbit Propagators” on page 1-31
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-39
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-49
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Satellite Constellation Access to a Ground Station

This example demonstrates how to set up access analysis between a ground station and conical sensors onboard a constellation of satellites. A ground station and a conical sensor belonging to a satellite are said to have access to one another if the ground station is inside the conical sensor's field of view and the conical sensor's elevation angle with respect to the ground station is greater than or equal to the latter's minimum elevation angle. The scenario involves a constellation of 40 low-Earth orbit satellites and a geographical site. Each satellite has a camera with a field of view of 90 degrees. The entire constellation of satellites is tasked with photographing the geographical site, which is located at 42.3001 degrees North and 71.3504 degrees West. The photographs are required to be taken between 12 May 2020 1:00 PM UTC and 12 May 2020 7:00 PM UTC when the site is adequately illuminated by the sun. In order to capture good quality pictures with minimal atmospheric distortion, the satellite's elevation angle with respect to the site should be at least 30 degrees (please note that 30 degrees was arbitrarily chosen for illustrative purposes). During the 6 hour interval, it is required to determine the times during which each satellite can photograph the site. It is also required to determine the percentage of time during this interval when at least one satellite's camera can see the site. This percentage quantity is termed the system-wide access percentage.

Create a Satellite Scenario

Create a satellite scenario using `satelliteScenario`. Use `datetime` to set the start time to 12-May-2020 1:00:00 PM UTC, and the stop time to 12-May-2020 7:00:00 PM UTC. Set the simulation sample time to 30 seconds.

```
startTime = datetime(2020,5,12,13,0,0);
stopTime = startTime + hours(6);
sampleTime = 30; % seconds
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc =
  satelliteScenario with properties:

         StartTime: 12-May-2020 13:00:00
          StopTime: 12-May-2020 19:00:00
        SampleTime: 30
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
        Satellites: [1×0 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
          AutoShow: 1
```

Add Satellites to the Satellite Scenario

Use `satellite` to add satellites to the scenario from the TLE file `leoSatelliteConstellation.tle`. The TLE file defines the mean orbital parameters of 40 generic satellites in nearly circular low-Earth orbits at an altitude and inclination of approximately 500 km and 55 degrees respectively.

```
tleFile = "leoSatelliteConstellation.tle";
sat = satellite(sc,tleFile)
```

```
sat =
  1x40 Satellite array with properties:

    Name
```

```
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

Add Cameras to the Satellites

Use `conicalSensor` to add a conical sensor to each satellite. These conical sensors represent the cameras. Specify their `MaxViewAngle` to be 90 degrees, which defines the field of view.

```
for idx = 1:numel(sat)
    name = sat(idx).Name + " Camera";
    conicalSensor(sat(idx),"Name",name,"MaxViewAngle",90);
end

% Retrieve the cameras
cam = [sat.ConicalSensors]
```

```
cam =
  1x40 ConicalSensor array with properties:

    Name
    ID
    MountingLocation
    MountingAngles
    MaxViewAngle
    Accesses
    FieldOfView
```

Define the Geographical Site to be Photographed in the Satellite Scenario

Use `groundStation` to add a ground station, which represents the geographical site to be photographed. Specify its `MinElevationAngle` to be 30 degrees. If latitude and longitude are not specified, they default to 42.3001 degrees North and 71.3504 degrees West.

```
name = "Geographical Site";
minElevationAngle = 30; % degrees
geoSite = groundStation(sc, ...
    "Name",name, ...
    "MinElevationAngle",minElevationAngle)
```

```
geoSite =
  GroundStation with properties:

                Name:  Geographical Site
                  ID:  81
```

```
            Latitude:  42.3 degrees
           Longitude:  -71.35 degrees
            Altitude:  0 meters
    MinElevationAngle:  30 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:  [0 1 1]
           MarkerSize:  10
            ShowLabel:  true
       LabelFontColor:  [0 1 1]
        LabelFontSize:  15
```

Add Access Analysis Between the Cameras and the Geographical Site

Use `access` to add access analysis between each camera and the geographical site. The access analyses will be used to determine when each camera can photograph the site.

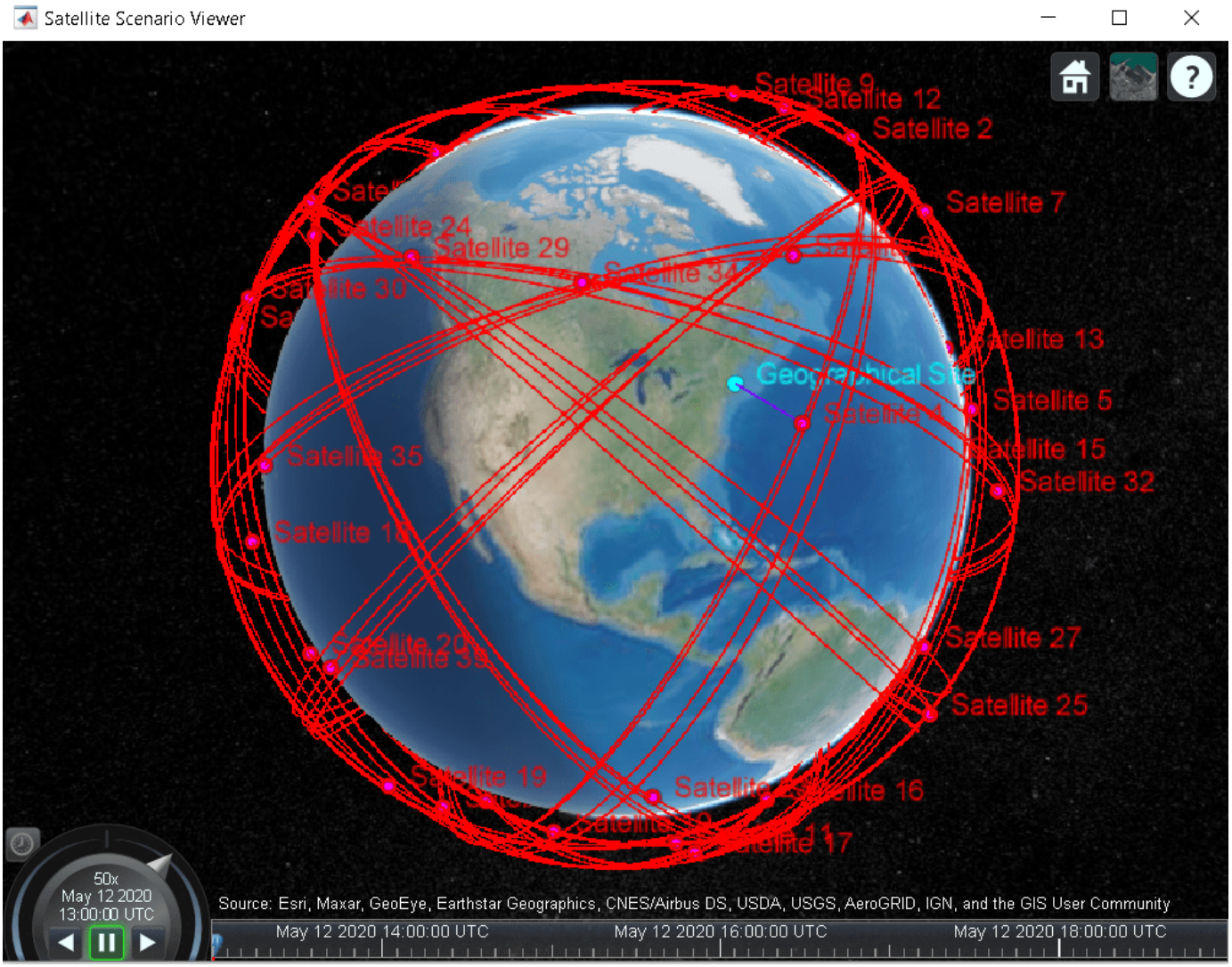
```
for idx = 1:numel(cam)
    access(cam(idx),geoSite);
end

% Retrieve the access analysis objects
ac = [cam.Accesses];

% Properties of access analysis objects
ac(1)
```

```
ans =
  Access with properties:

    Sequence:  [41 81]
    LineWidth:  1
    LineColor:  [0.5 0 1]
```

Visualize the Scenario

Use `satelliteScenarioViewer` to launch a satellite scenario viewer and visualize the scenario.

```
v = satelliteScenarioViewer(sc);
```



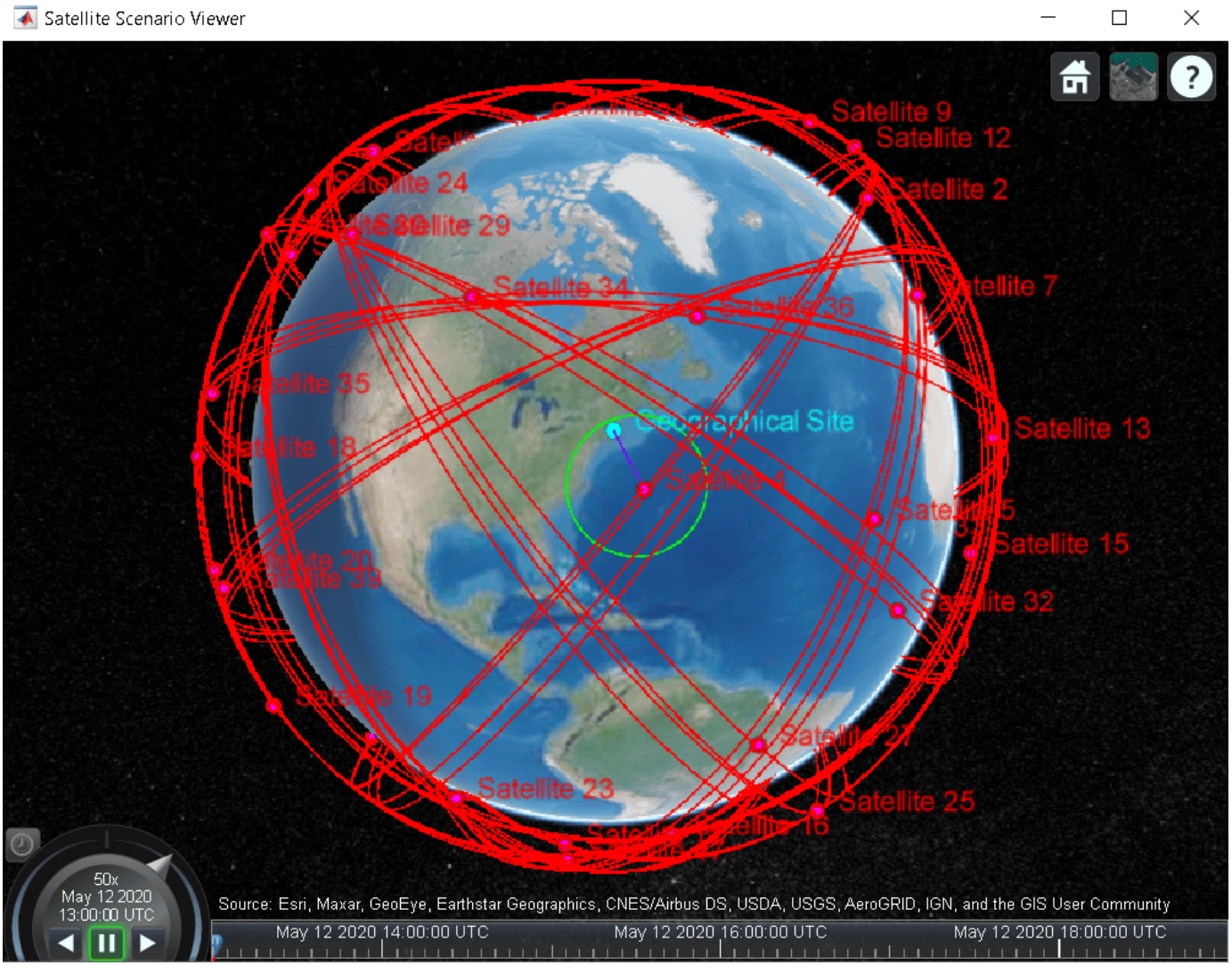
The viewer may be used as a visual confirmation that the scenario has been set up correctly. The violet line indicates that the camera on Satellite 4 and the geographical site have access to one another. This means that the geographical site is inside the camera's field of view and the camera's elevation angle with respect to the site is greater than or equal to 30 degrees. For the purposes of this scenario, this means that the camera can successfully photograph the site.

Visualize the Field Of View of the Camera

Use `fieldOfView` to visualize the field of view of each camera on Satellite 4.

```
fov = fieldOfView(cam([cam.Name] == "Satellite 4 Camera"))
```

```
fov =
  FieldOfView with properties:

         LineWidth: 1
         LineColor: [0 1 0]
    VisibilityMode: 'inherit'
```

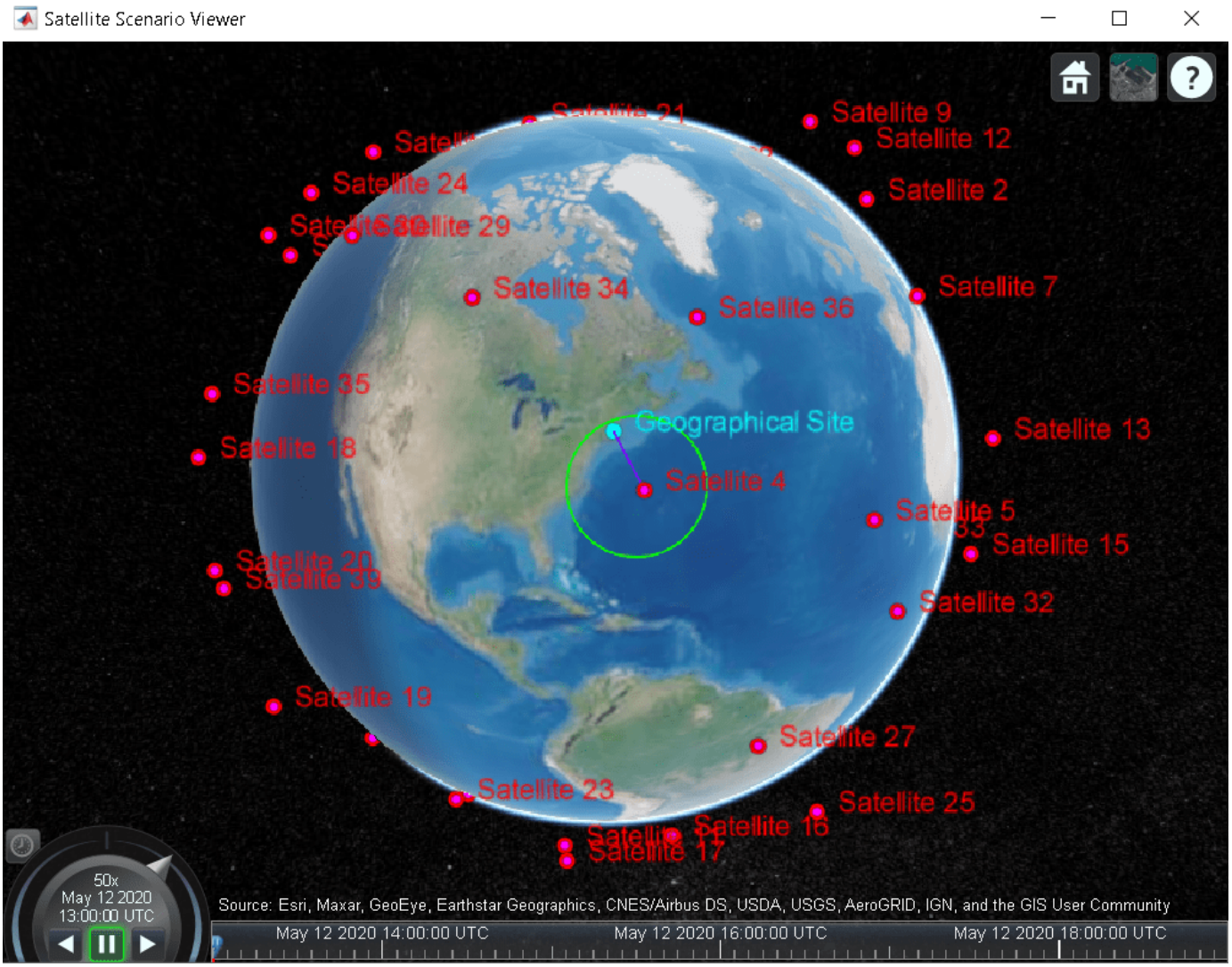


The presence of the geographical site inside the contour is a visual confirmation that it is inside the field of view of the camera onboard Satellite 4.

Customize the Visualizations

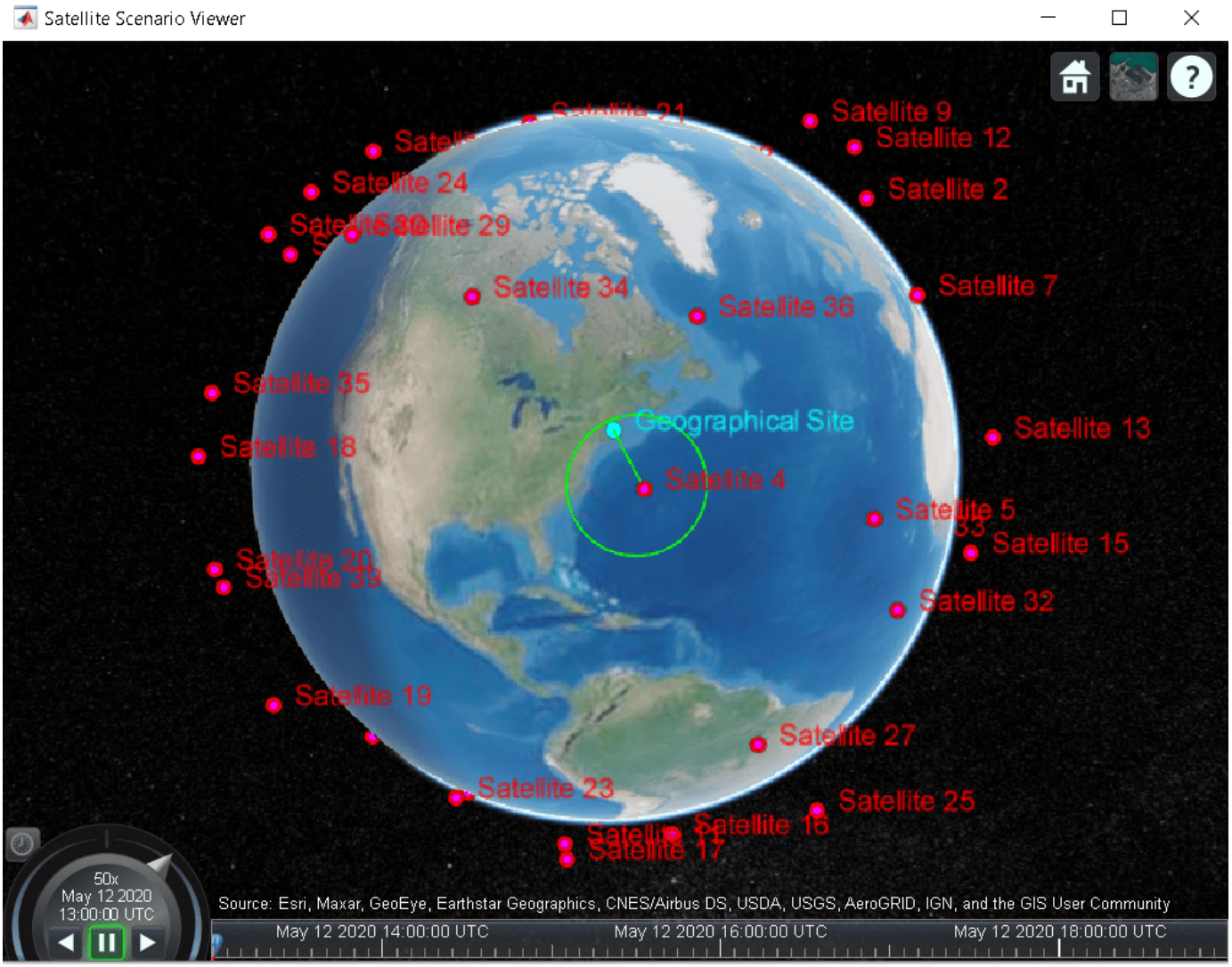
Use `hide` to hide the satellite orbits and declutter the visualization.

```
hide([sat.Orbit]);
```



Change the color of access visualizations to green.

```
for idx = 1:numel(ac)
    ac(idx).LineColor = 'green';
end
```



Determine the Times when the Cameras can Photograph the Geographical Site

Use `accessIntervals` to determine the times when there is access between each camera and the geographical site. These are the times when the camera can photograph the site.

```
accessIntervals(ac)
```

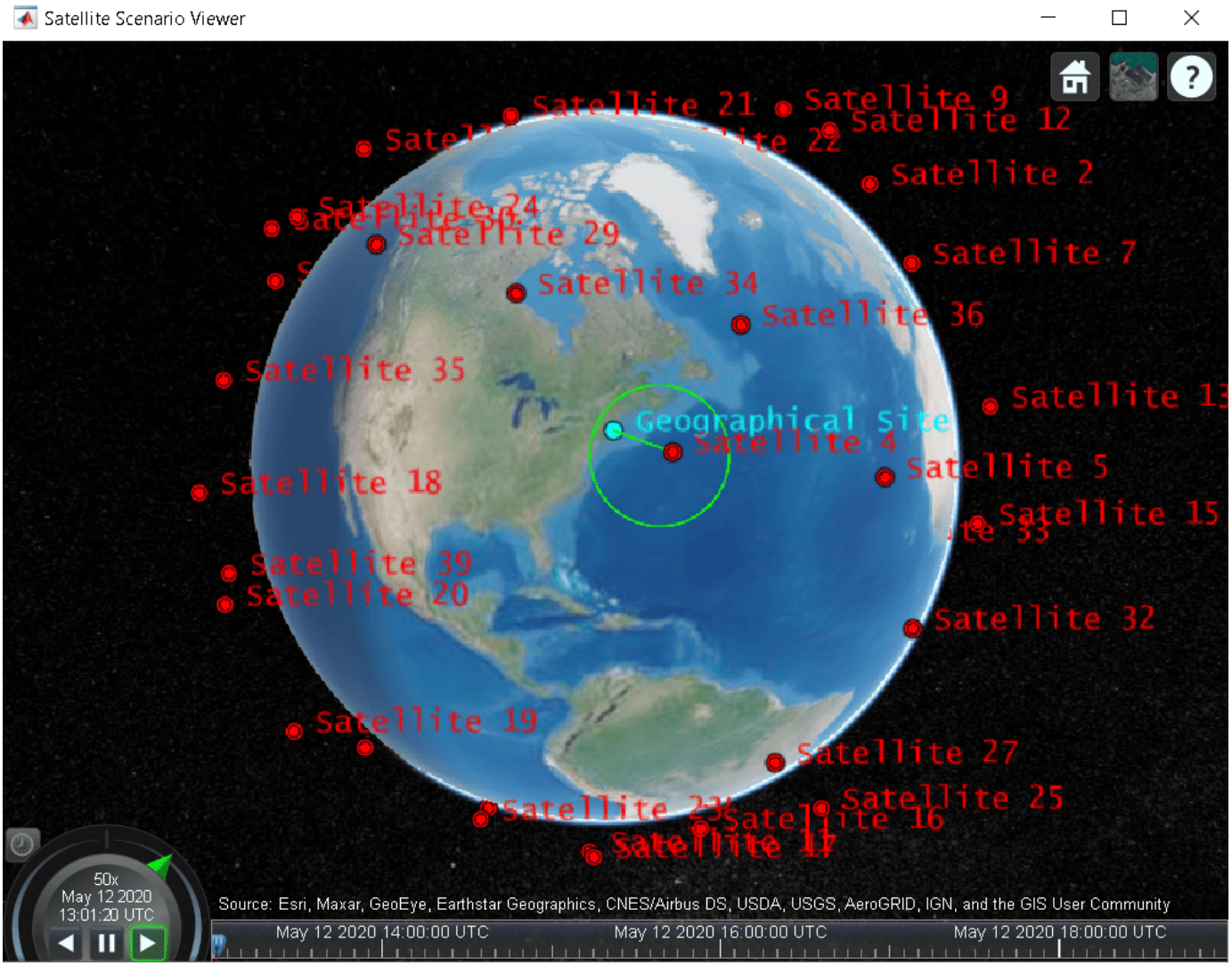
```
ans=30×8 table
          Source                 Target           IntervalNumber        StartTime
    ____________________    ____________________    ______________    ____________________

    "Satellite 1 Camera"    "Geographical Site"          1           12-May-2020 13:36:00
    "Satellite 1 Camera"    "Geographical Site"          2           12-May-2020 15:23:00
    "Satellite 2 Camera"    "Geographical Site"          1           12-May-2020 14:30:30
    "Satellite 3 Camera"    "Geographical Site"          1           12-May-2020 13:28:30
    "Satellite 4 Camera"    "Geographical Site"          1           12-May-2020 13:00:00
    "Satellite 4 Camera"    "Geographical Site"          2           12-May-2020 14:46:00
    "Satellite 5 Camera"    "Geographical Site"          1           12-May-2020 16:28:30
    "Satellite 6 Camera"    "Geographical Site"          1           12-May-2020 17:05:30
```

```
    "Satellite 7 Camera"     "Geographical Site"          1           12-May-2020 16:20:00    12
    "Satellite 8 Camera"     "Geographical Site"          1           12-May-2020 15:18:00    12
    "Satellite 8 Camera"     "Geographical Site"          2           12-May-2020 17:03:30    12
    "Satellite 9 Camera"     "Geographical Site"          1           12-May-2020 17:55:30    12
    "Satellite 10 Camera"    "Geographical Site"          1           12-May-2020 18:44:30    12
    "Satellite 11 Camera"    "Geographical Site"          1           12-May-2020 18:39:30    12
    "Satellite 12 Camera"    "Geographical Site"          1           12-May-2020 17:58:00    12
    "Satellite 29 Camera"    "Geographical Site"          1           12-May-2020 13:09:30    12
      ⋮
```

The above table consists of the start and end times of each interval during which a given camera can photograph the site. The duration of each interval is reported in seconds. StartOrbit and EndOrbit are the orbit counts of the satellite that the camera is attached to when the access begins and ends. The count starts from the scenario start time.

Use `play` to visualize the simulation of the scenario from its start time to stop time. It can be seen that the green lines appear whenever the camera can photograph the geographical site.

```
play(sc);
```



Calculate System-Wide Access Percentage

In addition to determining the times when each camera can photograph the geographical site, it is also required to determine the system-wide access percentage, which is the percentage of time from the scenario start time to stop time when at least one satellite can photograph the site. This is computed as follows:

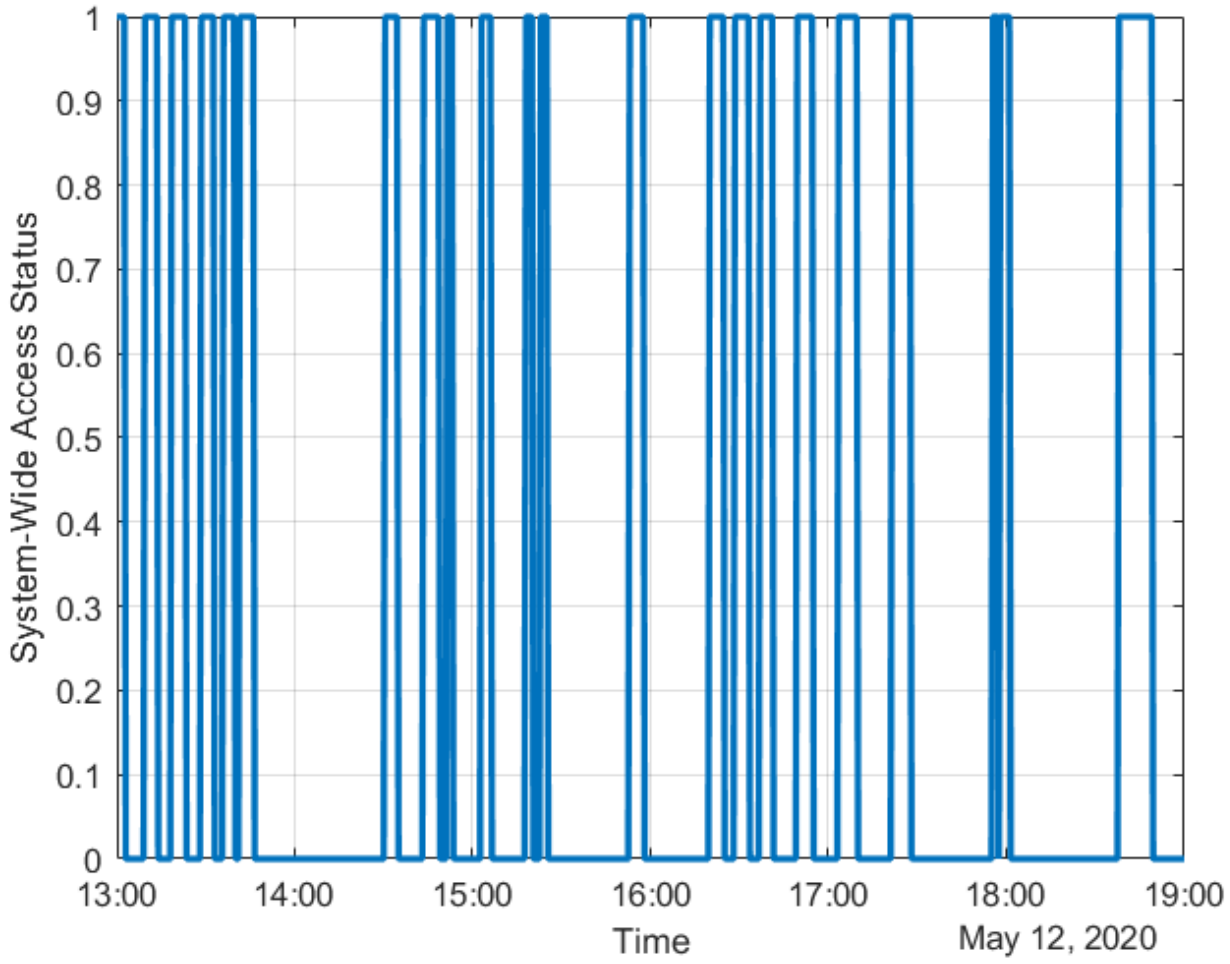
- For each camera, calculate the access status history to the site using `accessStatus`. For a given camera, this is a row vector of logicals, where each element in the vector represents the access status corresponding to a given time sample. A value of `True` indicates that the camera can photograph the site at that specific time sample.
- Perform a logical `OR` on all these row vectors corresponding to access of each camera to the site. This will result in a single row vector of logicals, in which a given element is true if at least one camera can photograph the site at the corresponding time sample for a duration of one scenario sample time of 30 seconds.

- Count the number of elements in the vector whose value is `True`. Multiply this quantity by the sample time of 30 seconds to determine the total time in seconds when at least one camera can photograph the site.
- Divide this quantity by the scenario duration of 6 hours and multiply by 100 to get the system-wide access percentage.

```
for idx = 1:numel(ac)
    [s,time] = accessStatus(ac(idx));

    if idx == 1
        % Initialize system-wide access status vector in the first iteration
        systemWideAccessStatus = s;
    else
        % Update system-wide access status vector by performing a logical OR
        % with access status for the current camera-site access
        % analysis
        systemWideAccessStatus = or(systemWideAccessStatus,s);
    end
end
```

Use `plot` to plot the system-wide access status with respect to time.

```
plot(time,systemWideAccessStatus,"LineWidth",2);
grid on;
xlabel("Time");
ylabel("System-Wide Access Status");
```



Whenever system-wide access status is 1 (`True`), at least one camera can photograph the site.

Use `nnz` to determine the number of elements in `systemWideAccessStatus` whose value is `True`.

```
n = nnz(systemWideAccessStatus)
```

```
n = 203
```

Determine the total time when at least one camera can photograph the site. This is accomplished by multiplying the number of `True` elements by the scenario's sample time.

```
systemWideAccessDuration = n*sc.SampleTime % seconds
```

```
systemWideAccessDuration = 6090
```

Use `seconds` to calculate the total scenario duration.

```
scenarioDuration = seconds(sc.StopTime - sc.StartTime)
```

```
scenarioDuration = 21600
```

Calculate the system-wide access percentage.

```
systemWideAccessPercentage = (systemWideAccessDuration/scenarioDuration)*100
```

```
systemWideAccessPercentage = 28.1944
```

Improve the System-Wide Access Percentage by Making the Cameras Track the Geographical Site

The default attitude configuration of the satellites is such that their yaw axes point straight down towards nadir (the point on Earth directly below the satellite). Since the cameras are aligned with the yaw axis by default, they point straight down as well. As a result, the geographical site goes outside the field of view of the cameras before their elevation angle dips below 30 degrees. Therefore, the cumulative access percentage is limited by the cameras' field of view.

If instead the cameras always point at the geographical site, the latter is always inside the cameras' field of view as long as the Earth is not blocking the line of sight. Consequently, the system-wide access percentage will now be limited by the `MinElevationAngle` of the geographical site, as opposed to the cameras' field of view. In the former case, the access intervals began and ended when the site entered and left the camera's field of view. It entered the field of view some time after the camera's elevation angle went above 30 degrees, and left the field of view before its elevation angle dipped below 30 degrees. However, if the cameras constantly point at the site, the access intervals will begin when the elevation angle rises above 30 degrees and end when it dips below 30 degrees, thereby increasing the duration of the intervals. Therefore, the system-wide access percentage will increase as well.

Since the cameras are rigidly attached to the satellites, each satellite is required to be continuously reoriented along its orbit so that its yaw axis tracks the geographical site. As the cameras are aligned with the yaw axis, they too will point at the site. Use `pointAt` to make each satellite's yaw axis track the geographical site.

```
for idx = 1:numel(sat)
    pointAt(sat(idx),geoSite);
end
```

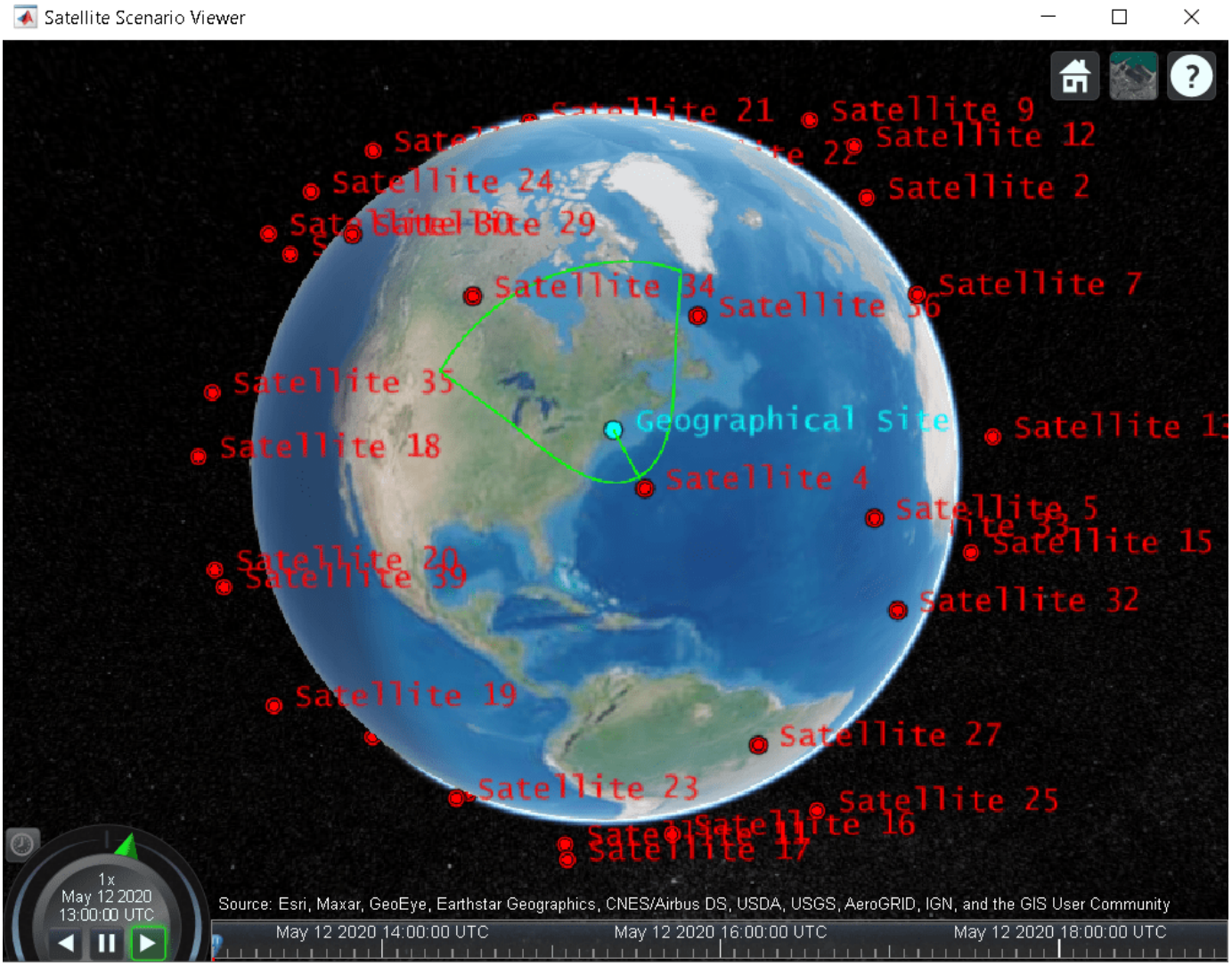
Re-calculate the system-wide access percentage.

```
% Calculate system-wide access status
for idx = 1:numel(ac)
    [s,time] = accessStatus(ac(idx));

    if idx == 1
        % Initialize system-wide access status vector in the first iteration
        systemWideAccessStatus = s;
    else
        % Update system-wide access status vector by performing a logical OR
        % with access status for the current camera-site combination
        systemWideAccessStatus = or(systemWideAccessStatus,s);
    end
end

% Calculate system-wide access percentage
n = nnz(systemWideAccessStatus);
systemWideAccessDuration = n*sc.SampleTime;
systemWideAccessPercentageWithTracking = (systemWideAccessDuration/scenarioDuration)*100
```

```
systemWideAccessPercentageWithTracking = 38.3333
```

The system-wide access percentage has improved by about 36%. This is the result of the cameras continuously pointing at the geographical site. This can be visualized by using `play` again.

```
play(sc)
```



The field of view contour is no longer circular because the camera is not pointing straight down anymore as it is tracking the geographical site.

Exploring the Example

This example demonstrated how to determine the times at which cameras onboard satellites in a constellation can photograph a geographical site. The cameras were modeled using conical sensors and access analysis was used to calculate the times when the cameras can photograph the site. Additionally, system-wide access percentage was computed to determine the percentage of time during a 6 hour period when at least one satellite can photograph the site. It was seen that these results depended on the direction at which the cameras were pointing.

These results are also a function of:

- Orbit of the satellites
- `MinElevationAngle` of the geographical site
- Mounting position and location of the cameras with respect to the satellites
- Field of view (`MaxViewAngle`) of the cameras if they are not continuously pointing at the geographical site

Modify the above parameters to your requirements and observe their influence on the access intervals and system-wide access percentage. The orbit of the satellites can be changed by explicitly specifying their Keplerian orbital elements using `satellite`. Additionally, the cameras can be mounted on `gimbals`, which can be rotated independent of the satellite. This way, the satellites can point straight down (the default behavior), while the gimbals can be configured so that the cameras independently track the geographical site.

See Also

Objects
`satelliteScenario` | `satellite` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor` | `transmitter` | `receiver`

Functions
`show` | `play` | `hide`

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Comparison of Orbit Propagators” on page 1-31
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-39
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-49
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Comparison of Orbit Propagators

This example compares the orbits predicted by the Two-Body-Keplerian, Simplified General Perturbations-4 (SGP4) and Simplified Deep-Space Perturbations-4 (SDP4) orbit propagators. An orbit propagator is a solver that calculates the position and velocity of an object whose motion is predominantly influenced by gravity from celestial bodies. The Two-Body-Keplerian orbit propagator is based on the relative two-body model that assumes a spherical gravity field for the Earth and neglects third body effects and other environmental perturbations, and hence, is the least accurate. The SGP4 orbit propagator accounts for secular and periodic orbital perturbations caused by Earth's geometry and atmospheric drag, and is applicable to near-Earth satellites whose orbital period is less than 225 minutes. The SDP4 orbit propagator builds upon SGP4 by accounting for solar and lunar gravity, and is applicable to satellites whose orbital period is greater than or equal to 225 minutes. The default orbit propagator for `satelliteScenario` is SGP4 for satellites whose orbital period is less than 225 minutes, and SDP4 otherwise.

Create a Satellite Scenario

Create a satellite scenario by using the `satelliteScenario` function. Set the start time to 11-May-2020 12:35:38 PM UTC, and the stop time to 13-May-2020 12:35:38 PM UTC, by using the `datetime` function. Set the sample time to 60 seconds.

```
startTime = datetime(2020,5,11,12,35,38);
stopTime = startTime + days(2);
sampleTime = 60;
sc = satelliteScenario(startTime,stopTime,sampleTime)
```

```
sc =
  satelliteScenario with properties:

         StartTime: 11-May-2020 12:35:38
          StopTime: 13-May-2020 12:35:38
        SampleTime: 60
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
        Satellites: []
    GroundStations: []
          AutoShow: 1
```

Add Satellites to the Satellite Scenario

Add three satellites to the satellite scenario from the two-line element (TLE) file `eccentricOrbitSatellite.tle` by using the `satellite` function. TLE is a data format used for encoding the orbital elements of an Earth-orbiting object defined at a specific time. Assign a Two-Body-Keplerian orbit propagator to the first satellite, SGP4 to the second satellite, and SDP4 to the third satellite.

```
tleFile = "eccentricOrbitSatellite.tle";
satTwoBodyKeplerian = satellite(sc,tleFile, ...
    "Name","satTwoBodyKeplerian", ...
    "OrbitPropagator","two-body-keplerian")
```

```
satTwoBodyKeplerian =
  Satellite with properties:

              Name: "satTwoBodyKeplerian"
                ID: 1
```

```
     ConicalSensors: []
            Gimbals: []
       Transmitters: []
          Receivers: []
           Accesses: []
         GroundTrack: [1×1 matlabshared.satellitescenario.GroundTrack]
               Orbit: [1×1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator: "two-body-keplerian"
        MarkerColor: [1 0 0]
         MarkerSize: 10
          ShowLabel: 1
     LabelFontColor: [1 0 0]
      LabelFontSize: 15
```

```
satSGP4 = satellite(sc,tleFile, ...
    "Name","satSGP4", ...
    "OrbitPropagator","sgp4")
```

```
satSGP4 =
  Satellite with properties:

               Name: "satSGP4"
                 ID: 2
     ConicalSensors: []
            Gimbals: []
       Transmitters: []
          Receivers: []
           Accesses: []
         GroundTrack: [1×1 matlabshared.satellitescenario.GroundTrack]
               Orbit: [1×1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator: "sgp4"
        MarkerColor: [1 0 0]
         MarkerSize: 10
          ShowLabel: 1
     LabelFontColor: [1 0 0]
      LabelFontSize: 15
```

```
satSDP4 = satellite(sc,tleFile, ...
    "Name","satSDP4", ...
    "OrbitPropagator","sdp4")
```

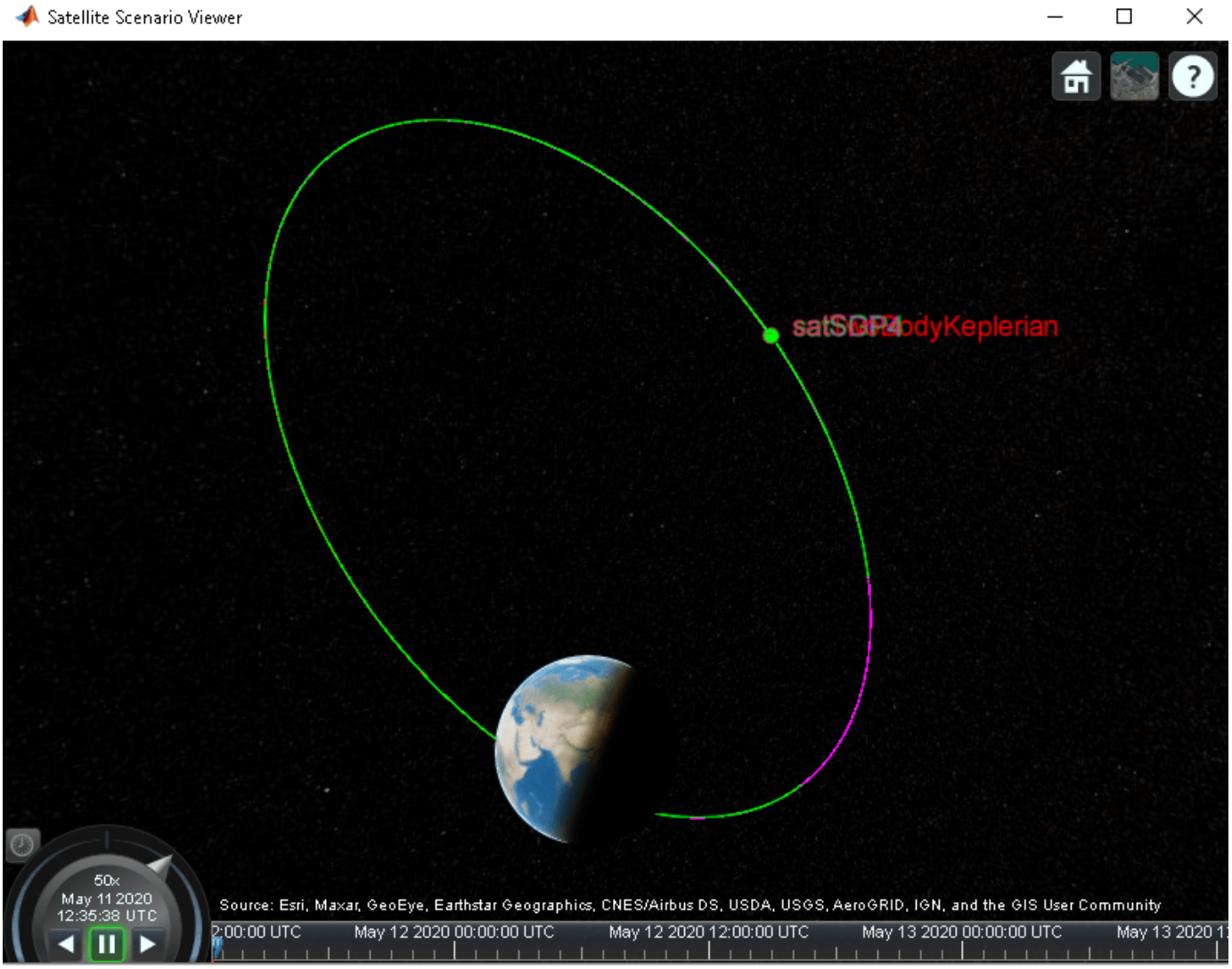
```
satSDP4 =
  Satellite with properties:

               Name: "satSDP4"
                 ID: 3
     ConicalSensors: []
            Gimbals: []
       Transmitters: []
          Receivers: []
           Accesses: []
         GroundTrack: [1×1 matlabshared.satellitescenario.GroundTrack]
               Orbit: [1×1 matlabshared.satellitescenario.Orbit]
    OrbitPropagator: "sdp4"
        MarkerColor: [1 0 0]
         MarkerSize: 10
          ShowLabel: 1
```

```
    LabelFontColor: [1 0 0]
     LabelFontSize: 15
```

Visualize the Satellites and their Orbits

Launch a satellite scenario viewer and visualize the satellite scenario by using the `satelliteScenarioViewer` function. Set the visualizations of `satTwoBodyKeplerian` to red, `satSGP4` to green, and `satSDP4` to magenta.

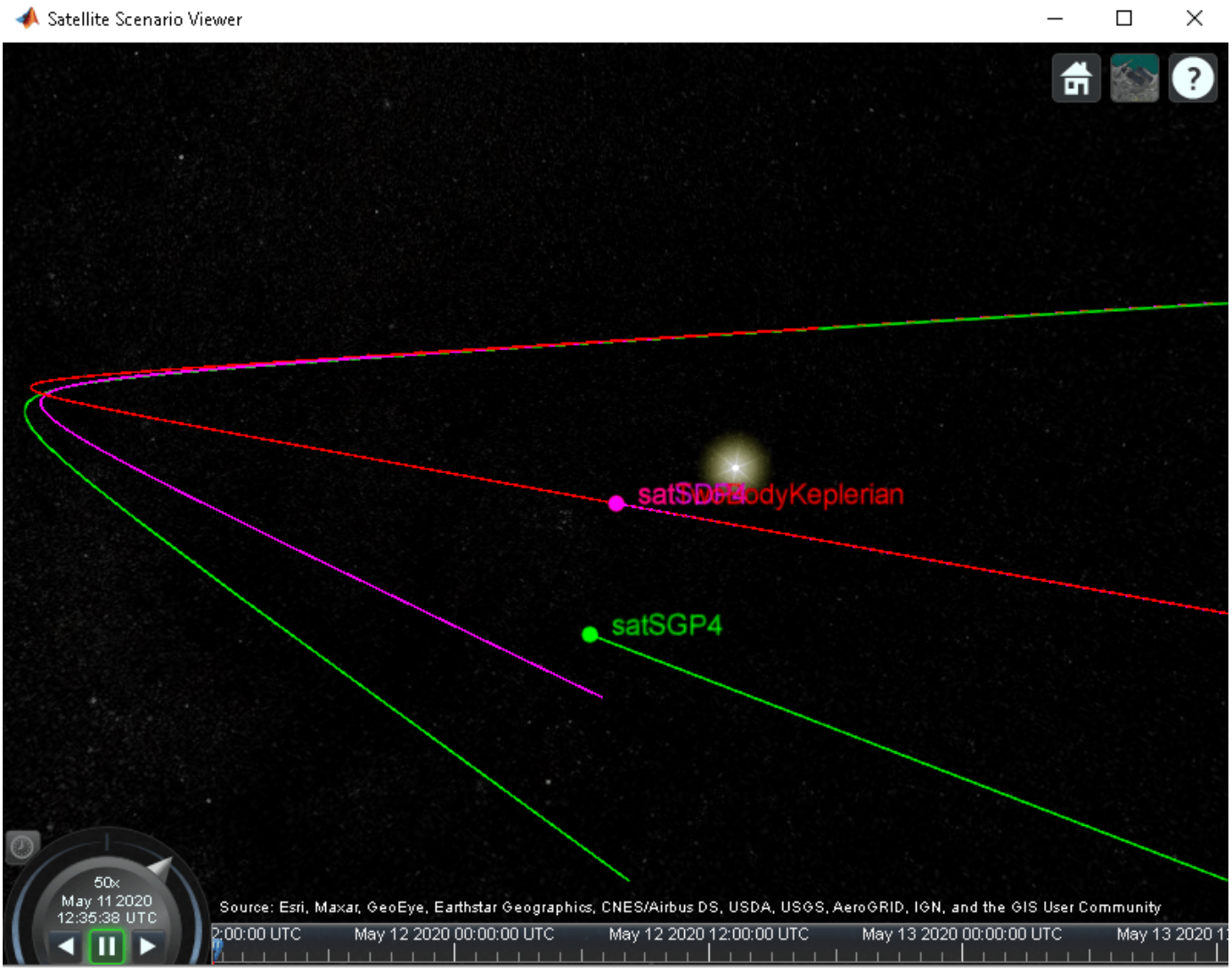
```
v = satelliteScenarioViewer(sc);
satSGP4.MarkerColor = [0 1 0];
satSGP4.Orbit.LineColor = [0 1 0];
satSGP4.LabelFontColor = [0 1 0];
satSDP4.MarkerColor = [1 0 1];
satSDP4.Orbit.LineColor = [1 0 1];
satSDP4.LabelFontColor = [1 0 1];
```



Focus the camera on `satTwoBodyKeplerian` by using the `camtarget` function.

```
camtarget(v,satTwoBodyKeplerian);
```

Left-click anywhere inside the satellite scenario viewer window and move the mouse while holding the click to pan the camera. Adjust the zoom level using the scroll wheel to bring all three satellites into view.



Visualize a Dynamic Animation of the Satellite Movement

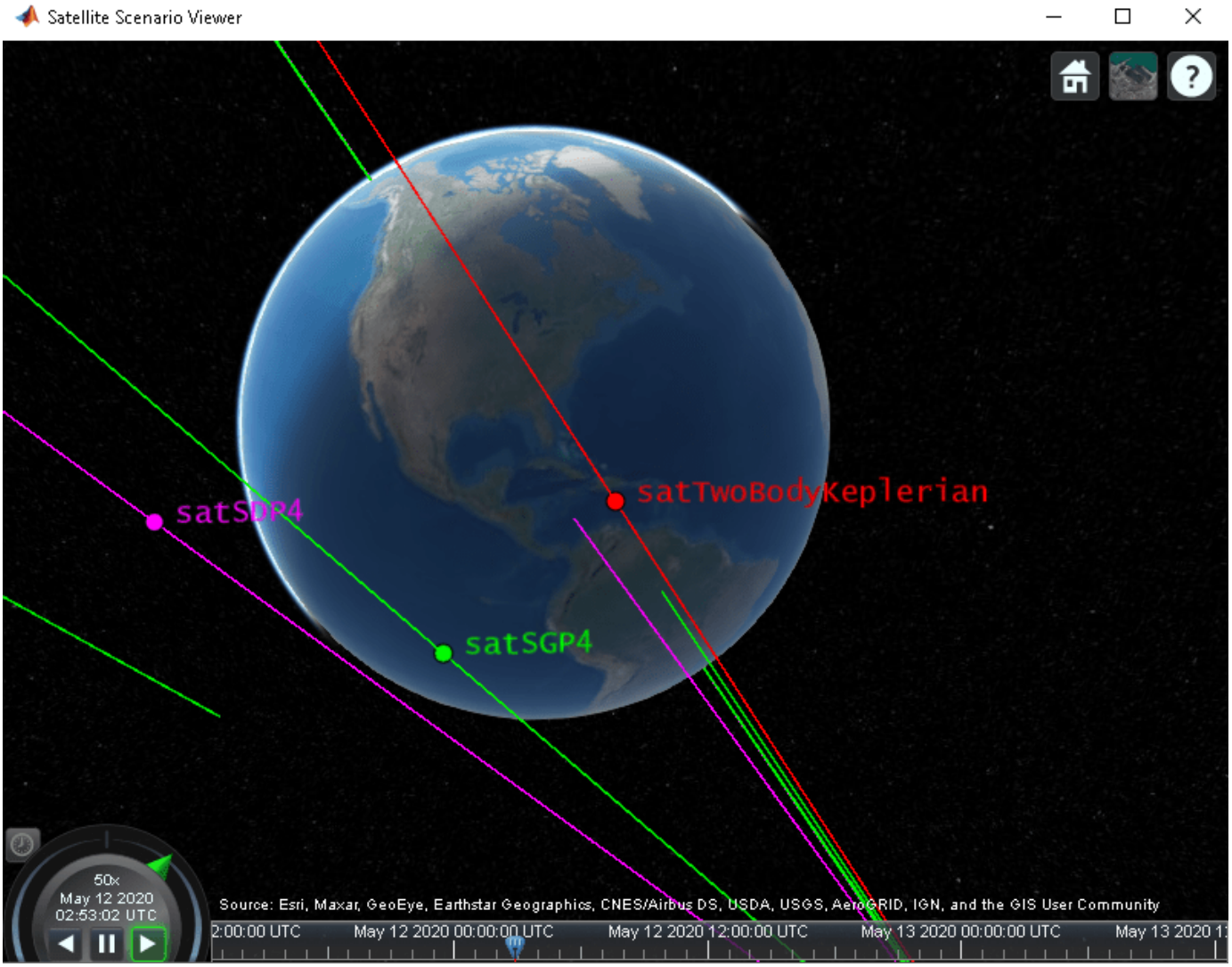
Visualize the movement of the satelliles by using the `play` function on the satellite scenario. The `play` function simulates the satellite scenario from the specified `StartTime` to `StopTime` using a step size specified by `SampleTime`, and plays the results on the satellite scenario viewer.

```
play(sc)
```

Use the playback controls located at the bottom of the satellite scenario viewer window to control the playback speed and direction. Focus the camera again on `satTwoBodyKeplerian` by using the `camtarget` function, and bring all three satellites into view by adjusting the zoom level.

```
camtarget(v,satTwoBodyKeplerian);
```

The positions of the three satellites diverge over time.



Obtain the Position and Velocity History of the Satellites

Return the position and velocity history of the satellites in the Geocentric Celestial Reference Frame (GCRF) by using the `states` function.

```
[positionTwoBodyKeplerian,velocityTwoBodyKeplerian,time] = states(satTwoBodyKeplerian);
[positionSGP4,velocitySGP4] = states(satSGP4);
[positionSDP4,velocitySDP4] = states(satSDP4);
```

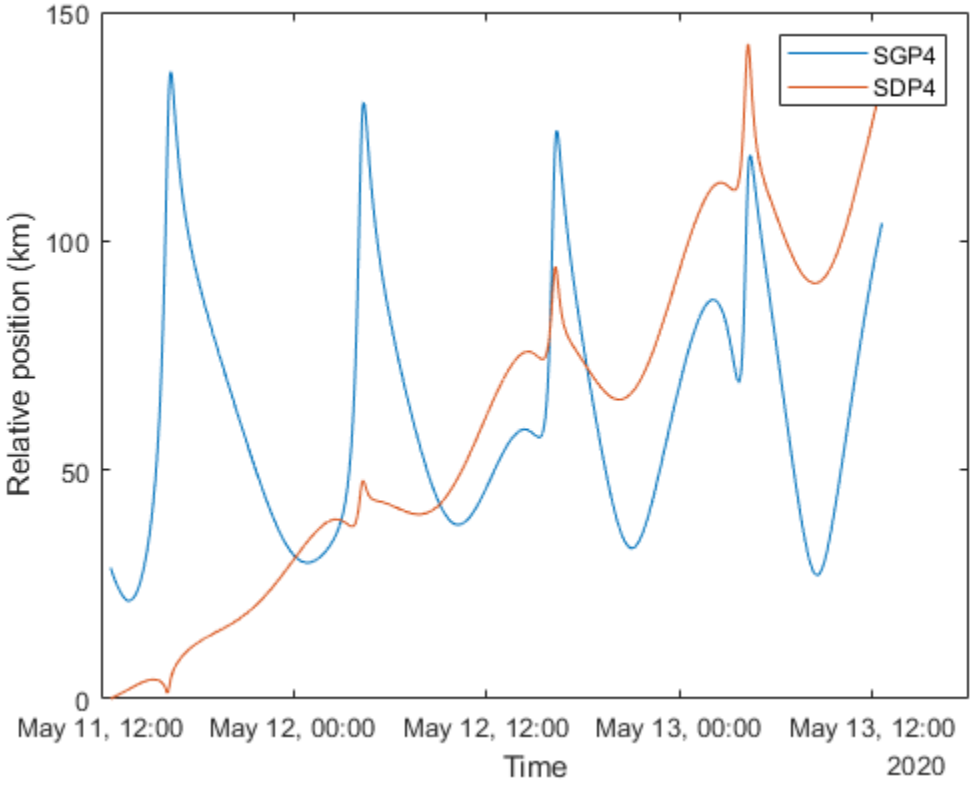
Plot Magnitude of Relative Position with Respect to Two-Body-Keplerian Prediction

Calculate the magnitude of the relative position of `satSGP4` and `satSDP4` with respect to `satTwoBodyKeplerian` by using the `vecnorm` function.

```
sgp4RelativePosition = vecnorm(positionSGP4 - positionTwoBodyKeplerian,2,1);
sdp4RelativePosition = vecnorm(positionSDP4 - positionTwoBodyKeplerian,2,1);
```

Plot the magnitude of the relative positions in kilometers of `satSGP4` and `satSDP4` with respect to that of `satTwoBodyKeplerian` by using the `plot` function.

```
sgp4RelativePositionKm = sgp4RelativePosition/1000;
sdp4RelativePositionKm = sdp4RelativePosition/1000;
plot(time,sgp4RelativePositionKm,time,sdp4RelativePositionKm)
xlabel("Time")
ylabel("Relative position (km)")
legend("SGP4","SDP4")
```



The initial relative position of `satSGP4` is non-zero and that of `satSDP4` is zero because the initial positions of `satTwoBodyKeplerian` and `satSDP4` are calculated from the TLE file using the SDP4 orbit propagator, while the initial position of `satSGP4` is calculated using the SGP4 orbit propagator. Over time, the position of `satSDP4` deviates from that of `satTwoBodyKeplerian` because the subsequent positions of the former are calculated using the SDP4 orbit propagator, while those of the latter are calculated using the Two-Body-Keplerian orbit propagator. The SDP4 orbit propagator provides higher precision because unlike the Two-Body-Keplerian orbit propagator, it accounts for oblateness of the Earth, atmospheric drag, and gravity from the sun and the moon.

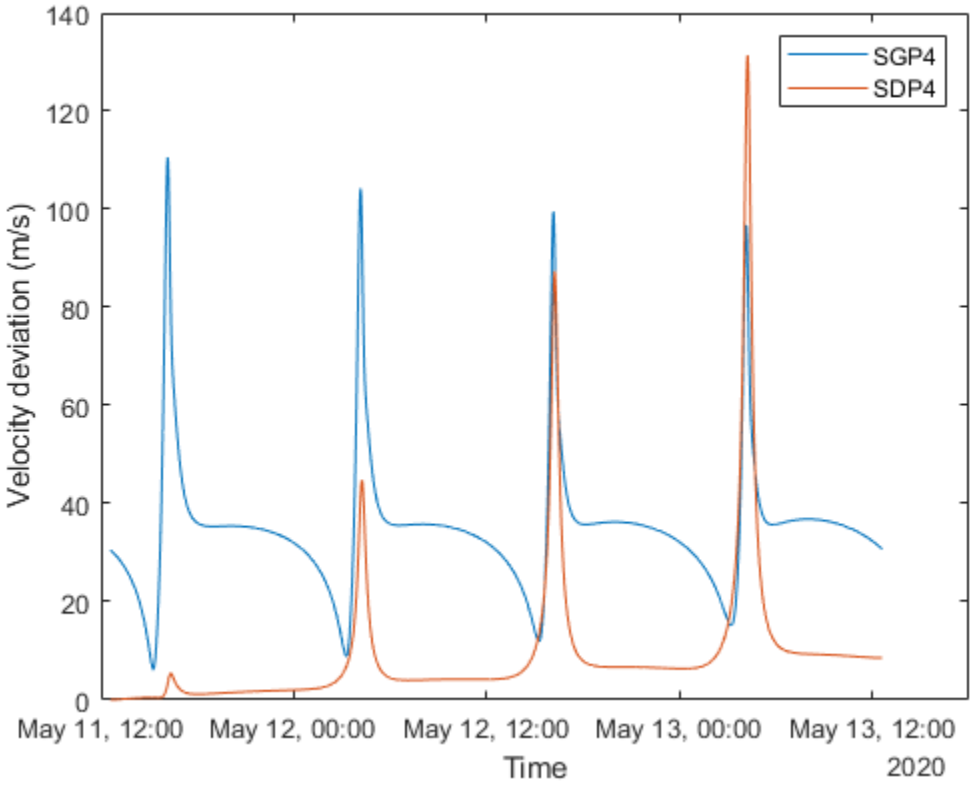
Plot Magnitude of Relative Velocity with Respect to Two-Body-Keplerian Prediction

Calculate the magnitude of the relative velocity of `satSGP4` and `satSDP4` with respect to `satTwoBodyKeplerian` by using the `vecnorm` function.

```
sgp4RelativeVelocity = vecnorm(velocitySGP4 - velocityTwoBodyKeplerian,2,1);
sdp4RelativeVelocity = vecnorm(velocitySDP4 - velocityTwoBodyKeplerian,2,1);
```

Plot the magnitude of the relative velocities in meters per second of `satSGP4` and `satSDP4` with respect to `satTwoBodyKeplerian` by using the `plot` function.

```
plot(time,sgp4RelativeVelocity,time,sdp4RelativeVelocity)
xlabel("Time")
ylabel("Velocity deviation (m/s)")
legend("SGP4","SDP4")
```



The initial relative velocity of `satSDP4` is zero because just like the initial position, the initial velocity of `satTwoBodyKeplerian` and `satSDP4` are also calculated from the TLE file using the SDP4 orbit propagator. Over time, the velocity of `satSDP4` deviates from that of `satTwoBodyKeplerian` because at all other times, the velocity of `satTwoBodyKeplerian` is calculated using the Two-Body-Keplerian orbit propagator, which has lower precision when compared to that of the SDP4 orbit propagator that is used for calculating the velocity of `satSDP4`. The spikes correspond to the periapsis (the closest point in the orbit from the center of mass of the Earth), where the magnitudes of the velocity errors are pronounced.

Conclusion

The deviations in the plots are the result of varying levels of accuracy of the three orbit propagators. The Two-Body-Keplerian orbit propagator is the least accurate as it assumes that the gravity field of the Earth is spherical, and also neglects all other sources of orbital perturbations. The SGP4 orbit propagator is more accurate as it accounts for the oblateness of the Earth and atmospheric drag. The SDP4 orbit propagator is the most accurate among the three because it also accounts for solar and

lunar gravity, which is more pronounced in this example because the orbital period is greater than 225 minutes, thereby taking the satellite farther away from the Earth.

See Also

Objects
`satelliteScenario` | `satellite` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor` | `transmitter` | `receiver`

Functions
`show` | `play` | `hide`

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-17
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-39
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-49
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Modeling Satellite Constellations Using Ephemeris Data

This example demonstrates how to add time-stamped ephemeris data for a constellation of 24 satellites (similar to ESA Galileo GNSS constellation) to a satellite scenario for access analysis. The example uses data generated by the Aerospace Blockset **`Orbit Propagator`** block. For more information, see the Aerospace Blockset example *Constellation Modeling with the Orbit Propagator Block*.

The **`satelliteScenario`** object supports loading previously generated, time-stamped satellite ephemeris data into a scenario from a **`timeseries`** or **`timetable`** object. An ephemeris is a table containing position (and optionally velocity) state information of a satellite during a given period of time. Ephemeris data used to add satellites to the scenario object is interpolated via the `makima` interpolation method to align with the scenario time steps. This allows you to incorporate data generated by a Simulink model into either a new or existing satelliteScenario.

Define Mission Parameters and Constellation Initial Conditions

Specify a start date and duration for the mission. This example uses MATLAB structures to organize mission data. These structures make accessing data later in the example more intuitive. They also help declutter the global base workspace.

```
mission.StartDate = datetime(2020, 11, 30, 22, 23, 24);
mission.Duration  = hours(24);
```

The constellation in this example is a Walker-Delta constellation modeled similar to Galileo, the European GNSS (global navigation satellite system) constellation. The constellation consists of 24 satellites in medium Earth orbit (MEO). The satellites' Keplerian orbital elements at the mission start date epoch are:

```
mission.ConstellationDefinition = table( ...
    29599.8e3 * ones(24,1), ... % Semi-major axis (m)
    0.0005    * ones(24,1), ... % Eccentricity
    56        * ones(24,1), ... % Inclination (deg)
    350       * ones(24,1), ... % Right ascension of the ascending node (deg)
    sort(repmat([0 120 240], 1,8))', ... % Argument of periapsis (deg)
    [0:45:315, 15:45:330, 30:45:345]', ... % True anomaly (deg)
    'VariableNames', ["a (m)", "e", "i (deg)", "Ω (deg)", "ω (deg)", "ν (deg)"]);
mission.ConstellationDefinition
```

```
ans=24×6 table
     a (m)        e       i (deg)    Ω (deg)    ω (deg)    ν (deg)
    ________    ______    _______    _______    _______    _______

    2.96e+07    0.0005      56         350          0          0
    2.96e+07    0.0005      56         350          0         45
    2.96e+07    0.0005      56         350          0         90
    2.96e+07    0.0005      56         350          0        135
    2.96e+07    0.0005      56         350          0        180
    2.96e+07    0.0005      56         350          0        225
    2.96e+07    0.0005      56         350          0        270
    2.96e+07    0.0005      56         350          0        315
    2.96e+07    0.0005      56         350        120         15
    2.96e+07    0.0005      56         350        120         60
    2.96e+07    0.0005      56         350        120        105
    2.96e+07    0.0005      56         350        120        150
    2.96e+07    0.0005      56         350        120        195
```

```
   2.96e+07    0.0005      56         350        120        240
   2.96e+07    0.0005      56         350        120        285
   2.96e+07    0.0005      56         350        120        330
     ⋮
```

Load Ephemeris Timeseries Data

The timeseries objects contain position and velocity data for all 24 satellites in the constellation. The data is referenced in the International Terrestrial Reference frame (ITRF), which is an Earth-centered Earth-fixed (ECEF) coordinate system. The data was generated using the Aerospace Blockset **`Orbit Propagator`** block. For more information, see the Aerospace Blockset example *Constellation Modeling with the Orbit Propagator Block*.

```
mission.Ephemeris = load("SatelliteScenarioEphemerisData.mat", "TimeseriesPosITRF", "TimeseriesVe
mission.Ephemeris.TimeseriesPosITRF
```

```
  timeseries

  Common Properties:
           Name: ''
           Time: [57x1 double]
       TimeInfo: [1x1 tsdata.timemetadata]
           Data: [24x3x57 double]
       DataInfo: [1x1 tsdata.datametadata]

  More properties, Methods
```

```
mission.Ephemeris.TimeseriesVelITRF
```

```
  timeseries

  Common Properties:
           Name: ''
           Time: [57x1 double]
       TimeInfo: [1x1 tsdata.timemetadata]
           Data: [24x3x57 double]
       DataInfo: [1x1 tsdata.datametadata]

  More properties, Methods
```

Load the Satellite Ephemerides into a `satelliteScenario` Object

Create a satellite scenario object for the analysis.

```
scenario = satelliteScenario(mission.StartDate, mission.StartDate + hours(24), 60);
```

Use the **`satellite`** method to add all 24 satellites to the satellite scenario from the ECEF position and velocity timeseries objects. This example uses position and velocity information; however satellites can also be added from position data only and velocity states are then estimated. Available coordinate frames for Name-Value pair `CoordinateFrame` are `"ECEF"`, `"Inertial"`, and `"Geographic"`. If the timeseries object contains a value for ts`.TimeInfo.StartDate`, the method uses that value as the epoch for the timeseries object. If no `StartDate` is defined, the method uses the scenario start date by default.

```
sat = satellite(scenario, mission.Ephemeris.TimeseriesPosITRF, mission.Ephemeris.TimeseriesVelITR
    "CoordinateFrame", "ecef", "Name", "GALILEO " + (1:24))
```

```
sat =
  1x24 Satellite array with properties:

    Name
    ID
    ConicalSensors
    Gimbals
    Transmitters
    Receivers
    Accesses
    GroundTrack
    Orbit
    OrbitPropagator
    MarkerColor
    MarkerSize
    ShowLabel
    LabelFontColor
    LabelFontSize
```

```
disp(scenario)
```

```
  satelliteScenario with properties:

         StartTime: 30-Nov-2020 22:23:24
          StopTime: 01-Dec-2020 22:23:24
        SampleTime: 60
           Viewers: [0×0 matlabshared.satellitescenario.Viewer]
        Satellites: [1×24 matlabshared.satellitescenario.Satellite]
    GroundStations: [1×0 matlabshared.satellitescenario.GroundStation]
          AutoShow: 1
```

Alternatively, satellites can also be added as ephemerides to the satellite scenario as a MATLAB **`timetable`**, **`table`**, or **`tscollection`**. For example, a **`timetable`** containing the first 3 satellites of the position **`timeseries`** object in the previous section, formatted for use with **`satelliteScenario`** objects is shown below.

- Satellites are represented by variables (column headers).
- Each row contains a position vector associated with the row's `Time` property.

```
timetable(...
datetime(getabstime(mission.Ephemeris.TimeseriesPosITRF), 'Locale', 'en_US'), ...
squeeze(mission.Ephemeris.TimeseriesPosITRF.Data(1,:,:))', ...
squeeze(mission.Ephemeris.TimeseriesPosITRF.Data(2,:,:))', ...
squeeze(mission.Ephemeris.TimeseriesPosITRF.Data(3,:,:))',...
'VariableNames', ["Satellite_1", "Satellite_2", "Satellite_3"])
```

```
ans=57×3 timetable
           Time                        Satellite_1                                    Satellite_2
    ____________________    ____________________________________    ____________________________________

    30-Nov-2020 22:23:24    1.8249e+07    -2.2904e+07    -4.2009e+06    2.3678e+07     -1.075e+07
    30-Nov-2020 22:23:38    1.8252e+07    -2.2909e+07    -4.1563e+06    2.3662e+07    -1.0735e+07
    30-Nov-2020 22:24:53    1.8268e+07    -2.2937e+07     -3.933e+06    2.3584e+07    -1.0663e+07
    30-Nov-2020 22:31:05    1.8326e+07    -2.3055e+07    -2.8121e+06    2.3185e+07     -1.028e+07
    30-Nov-2020 22:48:39    1.8326e+07    -2.3223e+07     3.9182e+05    2.2005e+07    -8.9966e+06
    30-Nov-2020 23:08:30    1.8076e+07    -2.3078e+07     3.9992e+06    2.0643e+07    -7.2057e+06
    30-Nov-2020 23:28:27    1.7624e+07    -2.2538e+07     7.5358e+06    1.9321e+07    -5.0678e+06
```

```
    30-Nov-2020 23:50:59    1.6968e+07    -2.1428e+07     1.1328e+07    1.7977e+07    -2.3021e+06
    01-Dec-2020 00:14:27    1.6244e+07    -1.9712e+07     1.4937e+07    1.6838e+07     8.7771e+05
    01-Dec-2020 00:38:42    1.5585e+07    -1.7375e+07     1.8189e+07    1.6017e+07      4.355e+06
    01-Dec-2020 01:04:35    1.5124e+07    -1.4345e+07     2.1006e+07    1.5585e+07     8.1065e+06
    01-Dec-2020 01:31:17    1.5035e+07     -1.079e+07     2.3096e+07     1.562e+07     1.1816e+07
    01-Dec-2020 01:58:58    1.5443e+07    -6.8501e+06     2.4303e+07    1.6102e+07     1.5274e+07
    01-Dec-2020 02:27:08    1.6406e+07    -2.8152e+06     2.4478e+07    1.6925e+07     1.8197e+07
    01-Dec-2020 02:55:18    1.7869e+07      1.001e+06     2.3582e+07    1.7894e+07     2.0376e+07
    01-Dec-2020 03:23:29    1.9711e+07      4.381e+06     2.1653e+07    1.8787e+07     2.1739e+07
      ⋮
```

Set Graphical Properties on the Satellites

Viewer windows with many satellites can become crowded and difficult to read. To keep the window readable, manually control graphical properties of the scenario elements.

Hide the satellite labels and ground tracks.

```
set(sat, "ShowLabel", false);
hide([sat(:).GroundTrack]);
```

Set satellite in the same orbital plane to have the same orbit color.

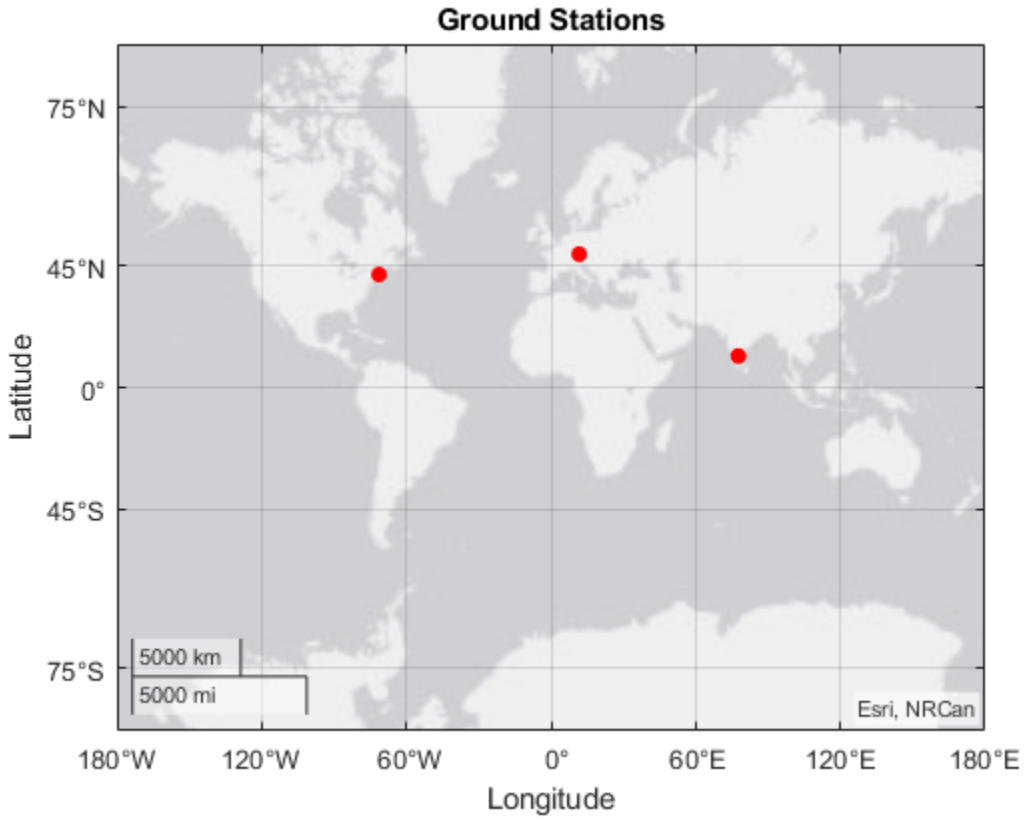
```
set(sat(1:8), "MarkerColor", "red");
set(sat(9:16), "MarkerColor", "blue");
set(sat(17:24), "MarkerColor", "green");
orbit = [sat(:).Orbit];
set(orbit(1:8), "LineColor", "red");
set(orbit(9:16), "LineColor", "blue");
set(orbit(17:24), "LineColor", "green");
```

Add Ground Stations to Scenario

To provide accurate positioning data, a location on Earth must have access to at least 4 satellites in the constellation at any given time. In this example, use three locations to compare total constellation access over the 1 day analysis window to different regions of Earth:

- Natick, Massachusetts, USA (42.30048°, -71.34908°)
- München, Germany (48.23206°, 11.68445°)
- Bangalore, India (12.94448°, 77.69256°)

```
gsUS = groundStation(scenario, 42.30048, -71.34908, ...
    "MinElevationAngle", 10, "Name", "Natick");
gsDE = groundStation(scenario, 48.23206, 11.68445, ...
    "MinElevationAngle", 10, "Name", "Munchen");
gsIN = groundStation(scenario, 12.94448, 77.69256, ...
    "MinElevationAngle", 10, "Name", "Bangalore");

figure
geoscatter([gsUS.Latitude gsDE.Latitude gsIN.Latitude], ...
    [gsUS.Longitude gsDE.Longitude gsIN.Longitude], "red", "filled")
geolimits([-75 75], [-180 180])
title("Ground Stations")
```



Compute Ground Station to Satellite Access (Line-of-Sight Visibility)

Calculate line-of-sight access between the ground stations and each individual satellite using the `access` method.

```
for idx = 1:numel(sat)
    access(gsUS, sat(idx));
    access(gsDE, sat(idx));
    access(gsIN, sat(idx));
end
accessUS = [gsUS(:).Accesses];
accessDE = [gsDE(:).Accesses];
accessIN = [gsIN(:).Accesses];
```

Set access colors to match orbital plane colors assigned earlier in the example.

```
set(accessUS(1:8), "LineColor", "red");
set(accessUS(9:16), "LineColor", "blue");
set(accessUS(17:24), "LineColor", "green");

set(accessDE(1:8), "LineColor", "red");
set(accessDE(9:16), "LineColor", "blue");
set(accessDE(17:24), "LineColor", "green");

set(accessIN(1:8), "LineColor", "red");
set(accessIN(9:16), "LineColor", "blue");
set(accessIN(17:24), "LineColor", "green");
```

View the full access table between each ground station and all satellites in the constellation as tables. Sort the access intervals by interval start time. Satellites added from ephemeris data do not display values for `StartOrbit` and `EndOrbit`.

```
intervalsUS = accessIntervals(accessUS);
intervalsUS = sortrows(intervalsUS, "StartTime", "ascend")
```

```
intervalsUS=40×8 table
     Source        Target       IntervalNumber         StartTime              EndTime
    ________    ____________    ______________    ____________________    ____________________

    "Natick"    "GALILEO 1"           1           30-Nov-2020 22:23:24    01-Dec-2020 04:04:24
    "Natick"    "GALILEO 2"           1           30-Nov-2020 22:23:24    01-Dec-2020 01:24:24
    "Natick"    "GALILEO 3"           1           30-Nov-2020 22:23:24    30-Nov-2020 22:57:24
    "Natick"    "GALILEO 12"          1           30-Nov-2020 22:23:24    01-Dec-2020 00:00:24
    "Natick"    "GALILEO 13"          1           30-Nov-2020 22:23:24    30-Nov-2020 23:05:24
    "Natick"    "GALILEO 18"          1           30-Nov-2020 22:23:24    01-Dec-2020 04:00:24
    "Natick"    "GALILEO 19"          1           30-Nov-2020 22:23:24    01-Dec-2020 01:42:24
    "Natick"    "GALILEO 20"          1           30-Nov-2020 22:23:24    30-Nov-2020 22:46:24
    "Natick"    "GALILEO 11"          1           30-Nov-2020 22:25:24    01-Dec-2020 00:18:24
    "Natick"    "GALILEO 17"          1           30-Nov-2020 22:50:24    01-Dec-2020 05:50:24
    "Natick"    "GALILEO 8"           1           30-Nov-2020 23:20:24    01-Dec-2020 07:09:24
    "Natick"    "GALILEO 7"           1           01-Dec-2020 01:26:24    01-Dec-2020 10:00:24
    "Natick"    "GALILEO 24"          1           01-Dec-2020 01:40:24    01-Dec-2020 07:12:24
    "Natick"    "GALILEO 14"          1           01-Dec-2020 03:56:24    01-Dec-2020 07:15:24
    "Natick"    "GALILEO 6"           1           01-Dec-2020 04:05:24    01-Dec-2020 12:14:24
    "Natick"    "GALILEO 23"          1           01-Dec-2020 04:10:24    01-Dec-2020 08:03:24
      ⋮
```

```
intervalsDE = accessIntervals(accessDE);
intervalsDE = sortrows(intervalsDE, "StartTime", "ascend")
```

```
intervalsDE=40×8 table
     Source         Target       IntervalNumber         StartTime              EndTime
    ________    ____________    ______________    ____________________    ____________________

    "Munchen"    "GALILEO 2"           1           30-Nov-2020 22:23:24    01-Dec-2020 04:34:24
    "Munchen"    "GALILEO 3"           1           30-Nov-2020 22:23:24    01-Dec-2020 01:58:24
    "Munchen"    "GALILEO 4"           1           30-Nov-2020 22:23:24    30-Nov-2020 23:05:24
    "Munchen"    "GALILEO 10"          1           30-Nov-2020 22:23:24    30-Nov-2020 23:58:24
    "Munchen"    "GALILEO 19"          1           30-Nov-2020 22:23:24    01-Dec-2020 01:36:24
    "Munchen"    "GALILEO 20"          1           30-Nov-2020 22:23:24    01-Dec-2020 00:15:24
    "Munchen"    "GALILEO 21"          1           30-Nov-2020 22:23:24    30-Nov-2020 22:28:24
    "Munchen"    "GALILEO 9"           1           30-Nov-2020 22:34:24    01-Dec-2020 02:22:24
    "Munchen"    "GALILEO 18"          1           30-Nov-2020 22:41:24    01-Dec-2020 02:31:24
    "Munchen"    "GALILEO 1"           1           30-Nov-2020 23:05:24    01-Dec-2020 06:42:24
    "Munchen"    "GALILEO 16"          1           30-Nov-2020 23:29:24    01-Dec-2020 04:47:24
    "Munchen"    "GALILEO 15"          1           01-Dec-2020 00:50:24    01-Dec-2020 07:27:24
    "Munchen"    "GALILEO 17"          1           01-Dec-2020 01:05:24    01-Dec-2020 03:00:24
    "Munchen"    "GALILEO 8"           1           01-Dec-2020 01:57:24    01-Dec-2020 08:25:24
    "Munchen"    "GALILEO 14"          1           01-Dec-2020 02:36:24    01-Dec-2020 10:19:24
    "Munchen"    "GALILEO 7"           1           01-Dec-2020 04:35:24    01-Dec-2020 09:43:24
      ⋮
```

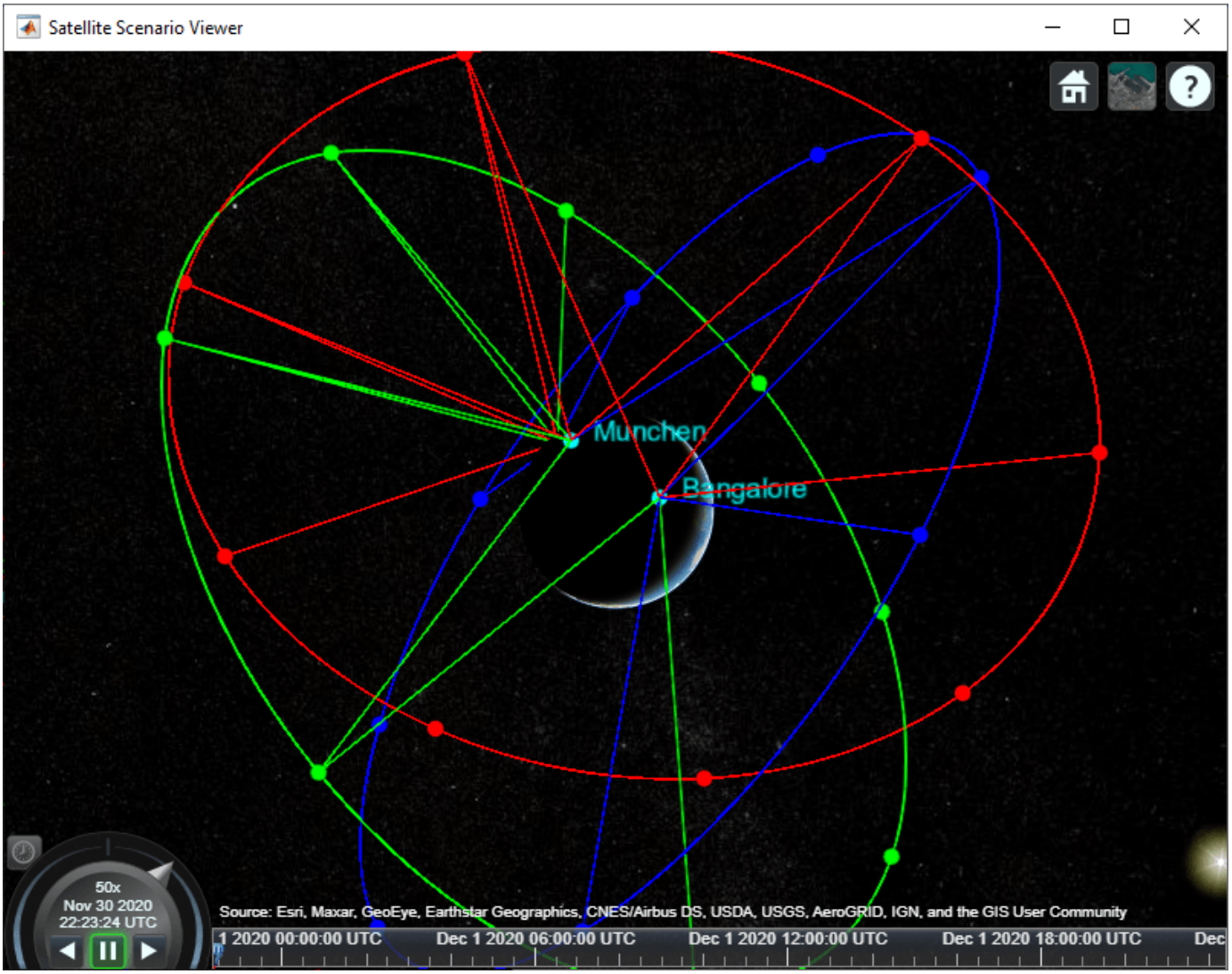
```
intervalsIN = accessIntervals(accessIN);
intervalsIN = sortrows(intervalsIN, "StartTime", "ascend")
```

```
intervalsIN=31×8 table
      Source          Target       IntervalNumber         StartTime                EndTime
    ___________    _____________    ______________    ____________________    ____________________

    "Bangalore"    "GALILEO 3"           1          30-Nov-2020 22:23:24    01-Dec-2020 05:12:24
    "Bangalore"    "GALILEO 4"           1          30-Nov-2020 22:23:24    01-Dec-2020 02:59:24
    "Bangalore"    "GALILEO 5"           1          30-Nov-2020 22:23:24    01-Dec-2020 00:22:24
    "Bangalore"    "GALILEO 9"           1          30-Nov-2020 22:23:24    01-Dec-2020 03:37:24
    "Bangalore"    "GALILEO 10"          1          30-Nov-2020 22:23:24    01-Dec-2020 00:09:24
    "Bangalore"    "GALILEO 16"          1          30-Nov-2020 22:23:24    01-Dec-2020 08:44:24
    "Bangalore"    "GALILEO 21"          1          30-Nov-2020 22:23:24    30-Nov-2020 23:25:24
    "Bangalore"    "GALILEO 22"          1          30-Nov-2020 22:23:24    30-Nov-2020 22:58:24
    "Bangalore"    "GALILEO 15"          1          01-Dec-2020 00:17:24    01-Dec-2020 11:16:24
    "Bangalore"    "GALILEO 2"           1          01-Dec-2020 00:25:24    01-Dec-2020 07:10:24
    "Bangalore"    "GALILEO 22"          2          01-Dec-2020 00:48:24    01-Dec-2020 05:50:24
    "Bangalore"    "GALILEO 21"          2          01-Dec-2020 01:32:24    01-Dec-2020 08:29:24
    "Bangalore"    "GALILEO 1"           1          01-Dec-2020 03:06:24    01-Dec-2020 07:17:24
    "Bangalore"    "GALILEO 20"          1          01-Dec-2020 03:36:24    01-Dec-2020 12:38:24
    "Bangalore"    "GALILEO 14"          1          01-Dec-2020 05:48:24    01-Dec-2020 13:29:24
    "Bangalore"    "GALILEO 19"          1          01-Dec-2020 05:53:24    01-Dec-2020 17:06:24
      ⋮
```

View the Satellite Scenario

Open a 3-D viewer window of the scenario. The viewer window contains all 24 satellites and the three ground stations defined earlier in this example. A line is drawn between each ground station and satellite during their corresponding access intervals.

```
viewer3D = satelliteScenarioViewer(scenario);
```

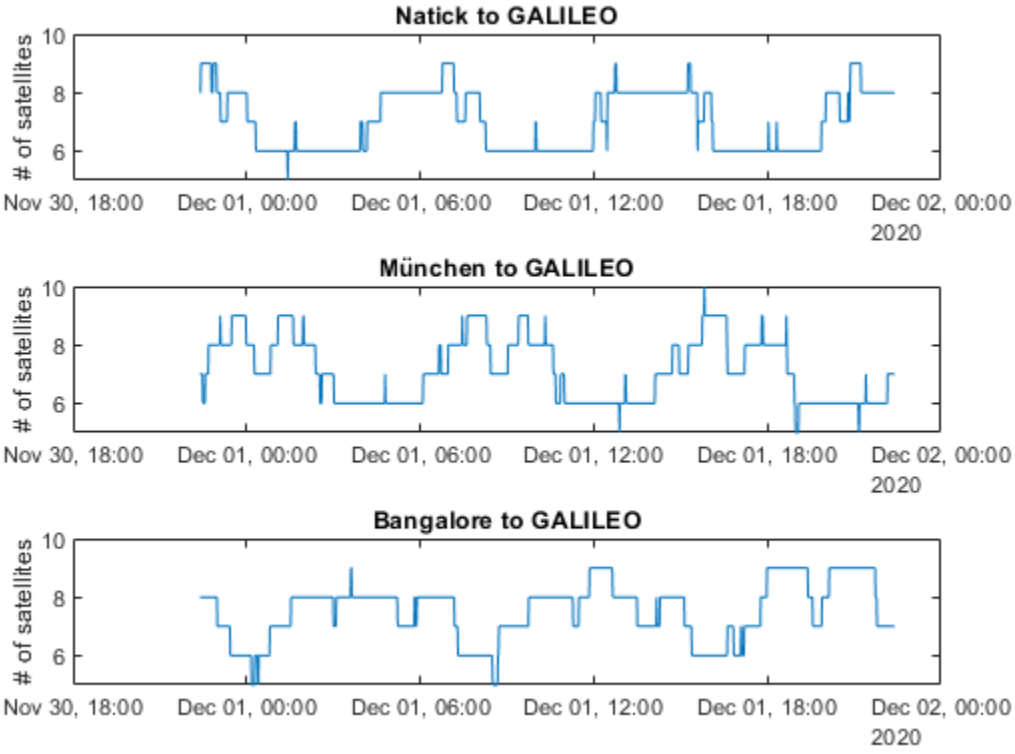


Compare Access Between Ground Stations

Calculate access status between each satellite and ground station using the accessStatus method. Plot cumulative access for each ground station over the one day analysis window.

```
% Initialize array with size equal to number of timesteps in scenario
timeSteps = mission.StartDate:seconds(60):mission.StartDate+days(1);
statusUS = zeros(1, numel(timeSteps));
statusDE = statusUS;
statusIN = statusUS;

% Sum cumulative access at each timestep
for idx = 1:24
    statusUS = statusUS + accessStatus(accessUS(idx));
    statusDE = statusDE + accessStatus(accessDE(idx));
    statusIN = statusIN + accessStatus(accessIN(idx));
end
clear idx;
```

```
subplot(3,1,1);
plot(timeSteps, statusUS);
title("Natick to GALILEO")
ylabel("# of satellites")
subplot(3,1,2);
plot(timeSteps, statusDE);
title("München to GALILEO")
ylabel("# of satellites")
subplot(3,1,3);
plot(timeSteps, statusIN);
title("Bangalore to GALILEO")
ylabel("# of satellites")
```



Collect access interval metrics for each ground station in a table for comparison.

```
statusTable = [table(height(intervalsUS), height(intervalsDE), height(intervalsIN)); ...
    table(sum(intervalsUS.Duration)/3600, sum(intervalsDE.Duration)/3600, sum(intervalsIN.Duratio
    table(mean(intervalsUS.Duration/60), mean(intervalsDE.Duration/60), mean(intervalsIN.Duration
    table(mean(statusUS, 2), mean(statusDE, 2), mean(statusIN, 2)); ...
    table(min(statusUS), min(statusDE), min(statusIN)); ...
    table(max(statusUS), max(statusDE), max(statusIN))];
statusTable.Properties.VariableNames = ["Natick", "München", "Bangalore"];
statusTable.Properties.RowNames = ["Total # of intervals", "Total interval time (hrs)",...
    "Mean interval length (min)", "Mean # of satellites in view", ...
    "Min # of satellites in view", "Max # of satellites in view"];
statusTable
```

```
statusTable=6×3 table
                                Natick    München    Bangalore
                                ______    _______    _________

    Total # of intervals            40         40          31
    Total interval time (hrs)    167.88     169.95      180.42
    Mean interval length (min)   251.82     254.93      349.19
    Mean # of satellites in view  7.018     7.1041      7.5337
    Min # of satellites in view       5          5           5
    Max # of satellites in view       9         10           9
```

Walker-Delta constellations like Galileo are evenly distributed across longitudes. Natick and München are located at similar latitudes, and therefore have very similar access characteristics with respect to the constellation. Bangalore is at a latitude closer to the equator. Despite having a lower number of individual access intervals, it has the highest average number of satellites in view, the highest overall interval time, and the longest average interval duration (by about 95 minutes). All locations always have at least 4 satellites in view, as is required for GNSS trilateration.

References

[1] Wertz, James R, David F. Everett, and Jeffery J. Puschell. *Space Mission Engineering: The New Smad*. Hawthorne, CA: Microcosm Press, 2011. Print.

[2] The European Space Agency: Galileo Facts and Figures. https://www.esa.int/Applications/Navigation/Galileo/Facts_and_figures

See Also

Objects

`satelliteScenario` | `satellite` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor` | `transmitter` | `receiver`

Functions

`show` | `play` | `hide`

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-17
- “Comparison of Orbit Propagators” on page 1-31
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-49
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Estimate GNSS Receiver Position with Simulated Satellite Constellations

Track the position of a ground vehicle using a simulated Global Navigation Satellite System (GNSS) receiver. The satellites are simulated using the `satelliteScenario` object, the satellite signal processing of the receiver are simulated using the `lookangles` (Navigation Toolbox) and `pseudoranges` (Navigation Toolbox) functions, and the receiver position is estimated with the `receiverposition` (Navigation Toolbox) function.

This example requires the Navigation Toolbox™.

Overview

This example focuses on the *space segment,* or satellite constellations, and the GNSS sensor equipment for a satellite system. To obtain an estimate of the GNSS receiver position, the navigation processor requires the satellite positions from the space segment and the pseudoranges from the ranging processor in the receiver.

Specify Simulation Parameters

Load the MAT-file that contains the ground-truth position and velocity of a ground vehicle travelling toward the Natick, MA campus of The MathWorks, Inc.

Specify the start, stop, and sample time of the simulation. Also, specify the mask angle, or minimum elevation angle, of the GNSS receiver.

```
% Load ground truth trajectory.
load("routeNatickMA.mat","lat","lon","pos","vel","lla0");
recPos = pos;
recVel = vel;

% Specify simulation times.
startTime = datetime(2020,10,28,8,0,0,"TimeZone","America/New_York");
simulationSteps = size(pos,1);
dt = 1;
stopTime = startTime + seconds((simulationSteps-1)*dt);

% Specify mask angle.
maskAngle = 5; % degrees

% Convert receiver position from North-East-Down (NED) to geodetic
% coordinates. Requires Navigation Toolbox(TM).
receiverLLA = ned2lla(recPos,lla0,"ellipsoid");

% Set RNG seed to allow for repeatable results.
rng("default");
```

Visualize the `geoplot` for the ground truth trajectory.

```
figure
geoplot(lat,lon)
geobasemap("topographic")
title("Ground Truth Trajectory")
```

Simulate Satellite Positions Over Time

The `satelliteScenario` object enables you to specify initial orbital parameters and visualize them using the `satelliteScenarioViewer` object. This example uses the `satelliteScenario` and the a MAT-file with initial orbital parameters to simulate the GPS constellations over time. Alternatively, you could use the `gnssconstellation` (Navigation Toolbox) object which simulates satellite positions using nominal orbital parameters, and only the current simulation time is needed to calculate the satellite positions.

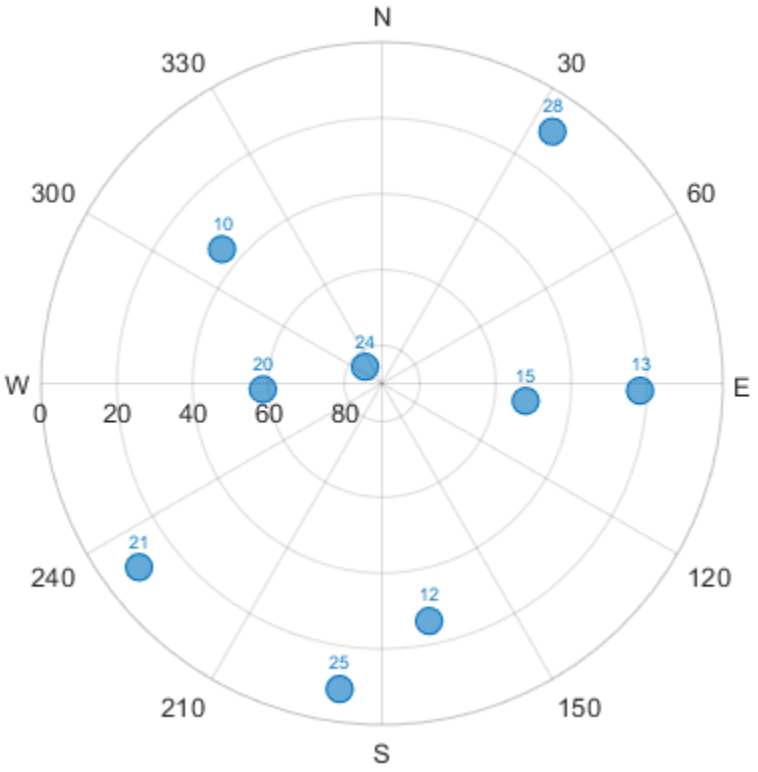
```
% Create scenario.
sc = satelliteScenario(startTime, stopTime, dt);

load("initialOrbitalParameters.mat","semiMajorAxis","eccentricity",...
    "inclination","rightAscensionOfAscendingNode",...
    "argumentOfPeriapsis","trueAnomaly","prnStr");

% Initialize satellites.
satellite(sc,semiMajorAxis,eccentricity,inclination,...
    rightAscensionOfAscendingNode,argumentOfPeriapsis,trueAnomaly,...
    "Name",prnStr);

% Preallocate results.
numSats = numel(sc.Satellites);
allSatPos = zeros(numSats,3,simulationSteps);
allSatVel = zeros(numSats,3,simulationSteps);

% Save satellite states over entire simulation.
```

```
for i = 1:numel(sc.Satellites)
    [oneSatPos, oneSatVel] = states(sc.Satellites(i),"CoordinateFrame","ecef");
    allSatPos(i,:,:) = permute(oneSatPos,[3 1 2]);
    allSatVel(i,:,:) = permute(oneSatVel,[3 1 2]);
end
```

Calculate Pseudoranges

Use the satellite positions to calculate the pseudoranges and satellite visibilities throughout the simulation. The mask angle is used to determine the satellites that are visible to the receiver. The pseudoranges are the distances between the satellites and the GNSS receiver. The term *pseudorange* is used because this distance value is calculated by multiplying the time difference between the current receiver clock time and the timestamped satellite signal by the speed of light.

```
% Preallocate results.
allP = zeros(numSats,simulationSteps);
allPDot = zeros(numSats,simulationSteps);
allIsSatVisible = false(numSats,simulationSteps);

% Use the skyplot to visualize satellites in view.
sp = skyplot([], []);

for idx = 1:simulationSteps
    satPos = allSatPos(:,:,idx);
    satVel = allSatVel(:,:,idx);

    % Calculate satellite visibilities from receiver position.
    [satAz,satEl,allIsSatVisible(:,idx)] = lookangles(receiverLLA(idx,:),satPos,maskAngle);

    % Calculate pseudoranges and pseudorange rates using satellite and
    % receiver positions and velocities.
    [allP(:,idx),allPDot(:,idx)] = pseudoranges(receiverLLA(idx,:),satPos,recVel(idx,:),satVel);

    set(sp,"AzimuthData",satAz(allIsSatVisible(:,idx)), ...
        "ElevationData",satEl(allIsSatVisible(:,idx)), ...
        "LabelData",prnStr(allIsSatVisible(:,idx)))
    drawnow limitrate
end
```



Estimate Receiver Position from Pseudoranges and Satellite Positions

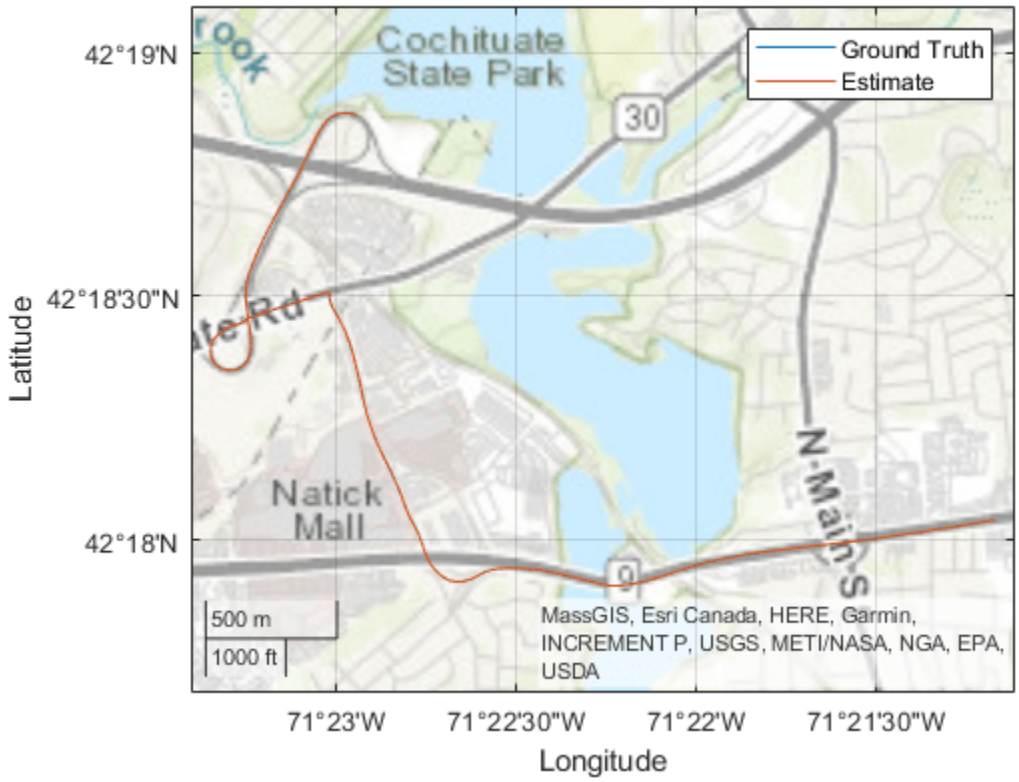
Finally, use the satellite positions and pseudoranges to estimate the receiver position with the `receiverposition` function.

```
% Preallocate results.
lla = zeros(simulationSteps,3);
gnssVel = zeros(simulationSteps,3);
hdop = zeros(simulationSteps,1);
vdop = zeros(simulationSteps,1);

for idx = 1:simulationSteps
    p = allP(:,idx);
    pdot = allPDot(:,idx);
    isSatVisible = allIsSatVisible(:,idx);
    satPos = allSatPos(:,:,idx);
    satVel = allSatVel(:,:,idx);

    % Estimate receiver position and velocity using pseudoranges,
    % pseudorange rates, and satellite positions and velocities.
    [lla(idx,:),gnssVel(idx,:),hdop(idx,:),vdop(idx,:)] = receiverposition(p(isSatVisible),...
        satPos(isSatVisible,:),pdot(isSatVisible),satVel(isSatVisible,:));
end
```

Visualize Results

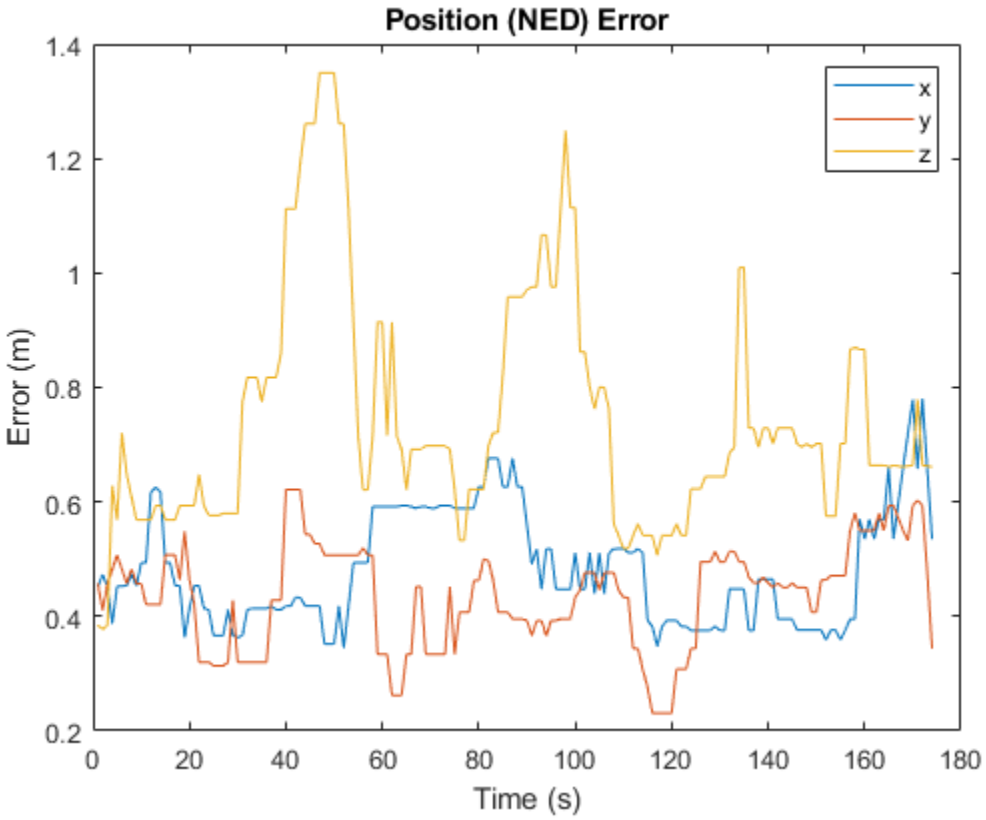
Plot the ground truth position and the estimated receiver position on a `geoplot`.

```
figure
geoplot(lat,lon,lla(:,1),lla(:,2))
geobasemap("topographic")
legend("Ground Truth","Estimate")
```



Plot the absolute error in the position estimate. The error is smoothed by a moving median to make the plot more readable. The error in the x- and y-axis is smaller because there are satellites on either side of the receiver. The error in the z-axis is larger because there are only satellites above the receiver, not below it. The error changes over time as the receiver moves and some satellites come in and out of view.

```
estPos = lla2ned(lla,lla0,"ellipsoid");
winSize = floor(size(estPos,1)/10);
figure
plot(smoothdata(abs(estPos-pos),"movmedian",winSize))
legend("x","y","z")
xlabel("Time (s)")
ylabel("Error (m)")
title("Position (NED) Error")
```



See Also

Objects
`satelliteScenario` | `satellite` | `access` | `groundStation` | `satelliteScenarioViewer` | `conicalSensor` | `transmitter` | `receiver`

Functions
`show` | `play` | `hide`

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-17
- “Comparison of Orbit Propagators” on page 1-31
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-39
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Calculate Latency and Doppler in a Satellite Scenario

This example shows how to model a satellite in orbit, analyze access between the satellite and a ground station, and calculate the latency and the doppler frequency between the satellite and the ground station.

Create a Satellite Scenario

Create a satellite scenario with a start time of 02-June-2020 8:23:00 AM UTC and a stop time 24 hours later. Set the simulation sample time to 60 seconds.

```
startTime = datetime(2020,6,02,8,23,0);
stopTime = startTime + hours(24);
sampleTime = 60;  % s
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add a Satellite to the Scenario

Add a satellite to the scenario from the `oneSatelliteConstellation` TLE file.

```
sat = satellite(sc,"oneSatelliteConstellation.tle");
```

Show the satellite in orbit and plot its future ground track, or lead time, over 20 minutes. Purple shows the future ground track, and orange shows the past ground track.

```
show(sat)
groundTrack(sat, ...
  "LeadTime",1200);  % s
```

Return Orbital Elements and Position of the Satellite

Display the orbital elements of the satellite in the scenario.

```
elements = orbitalElements(sat)
```

```
elements = struct with fields:
                       MeanMotion: 9.1649e-04
                     Eccentricity: 1.0000e-03
                      Inclination: 55
    RightAscensionOfAscendingNode: 175.0000
              ArgumentOfPeriapsis: 100
                      MeanAnomaly: 174.9900
                           Period: 6.8557e+03
                            Epoch: 02-Jun-2020 18:43:16
                            BStar: 1.0000e-04
```

Return the latitude (degrees), longitude (degrees), and altitude (km) of the satellite at time 02-June-2020 12:30:00 PM UTC.

```
time = datetime(2020,6,02,12,30,0);
pos = states(sat,time,"CoordinateFrame","geographic");
pos(3) = pos(3) / 1000;  % convert from m to km
fprintf("The satellite latitude is %3.2f degrees, its longitude is %5.2f degrees, and its altitud
  pos(1),pos(2),pos(3))
```

```
The satellite latitude is 1.07 degrees, its longitude is -83.95 degrees, and its altitude is 142
```

Add a Ground Station

Add the Madrid Deep Space Communications Complex as the ground station of interest, and specify its latitude and longitude.

```
name = "Madrid Deep Space Communications Complex";
lat = 40.43139;  % degrees
lon = -4.24806;  % degrees
gs = groundStation(sc,"Name",name,"Latitude",lat,"Longitude",lon);
```

Add an Access Analysis

Add an access analysis between the satellite and the ground station, which determines when the ground station is visible to the satellite. This determines when latency and doppler should be calculated.

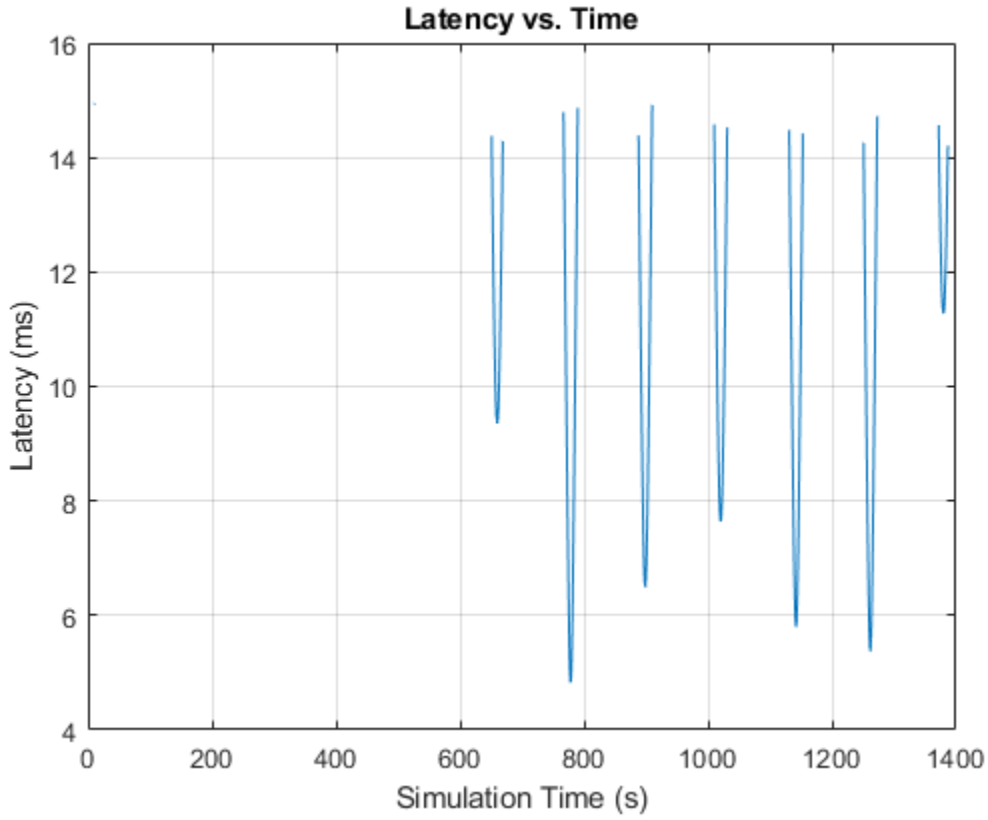
```
ac = access(sat,gs);
acStatus = accessStatus(ac);
```

Calculate Latency and Velocity

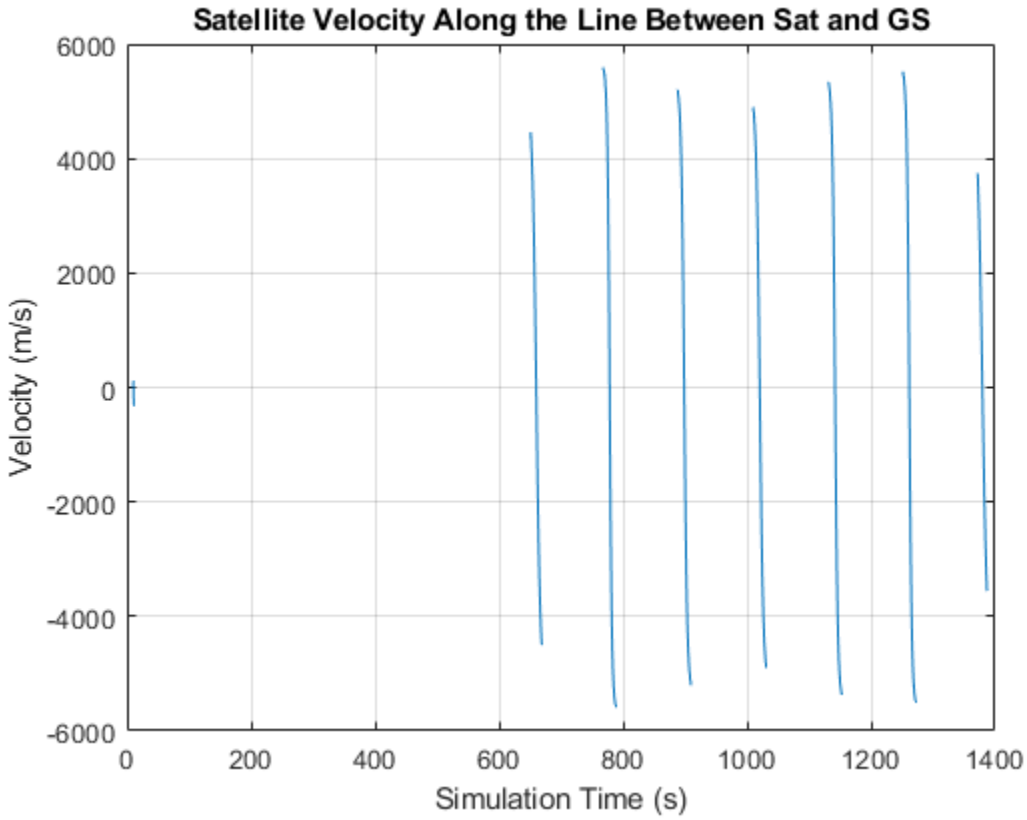
Calculate the latency between the satellite and the Madrid ground station. Also, calculate the velocity along the line between the satellite and the ground station. A positive value indicates that the satellite is moving towards the ground station, and a negative value indicates the satellite is moving away from the ground station.

```
numHours = stopTime - startTime;
numMinutes = minutes(numHours);
[az,el,r] = deal(zeros(numMinutes+1,1));
[satV,dir] = deal(zeros(3,numMinutes+1));
[latency,dopV] = deal(NaN(numMinutes+1,1));
c = physconst("Lightspeed");
for iMinute = 0:numMinutes
  time = startTime + minutes(iMinute);
  idx = iMinute+1;
  % Calculate latency and doppler only when the satellite has access to the
  % ground station.
  if acStatus(idx)
    % Calculate azimuth, elevation, and range from the satellite to the
    % ground station
    [az(idx),el(idx),r(idx)] = aer(sat,gs,time);

    % Calculate latency
    latency(idx) = r(idx) / c;

    % Calculate satellite velocity in North/East/Down (NED) frame of
    % satellite. Physically, this is the ECEF velocity, represented in NED
    % frame. Therefore, the relative velocity with respect to the ground
    % station is also the same.
    [~,satV(:,idx)] = states(sat,time,"CoordinateFrame","geographic");

    % Calculate the direction of gs with respect to sat in sat NED frame
    dir(:,idx) = [cosd(el(idx))*cosd(az(idx)); ...
      cosd(el(idx))*sind(az(idx)); ...
      -sind(el(idx))];

    % Calculate the velocity along the line between the gs and the sat.
    % This velocity determines the doppler frequency.
```

```
    dopV(idx) = dot(satV(:,idx),dir(:,idx));
  end
end
plot(1000*latency)  % ms
title("Latency vs. Time")
xlabel("Simulation Time (s)")
ylabel("Latency (ms)")
grid on
```




```
plot(dopV)
title("Satellite Velocity Along the Line Between Sat and GS")
xlabel("Simulation Time (s)")
ylabel("Velocity (m/s)")
grid on
```



Calculate Doppler Frequency from Velocity

The doppler frequency is calculated from the following equation:

$$f_o = \left(\frac{c \pm v_r}{c \pm v_s}\right) f_e,$$

where c is the speed of light in m/s,

v_r is the speed, in m/s, of the receiver relative to the medium, added to c if the receiver is moving towards the source, subtracted if the receiver is moving away from the source,

v_s is the speed, in m/s, of the source relative to the medium, added to c if the source is moving away from the receiver, subtracted if the source is moving towards the receiver,

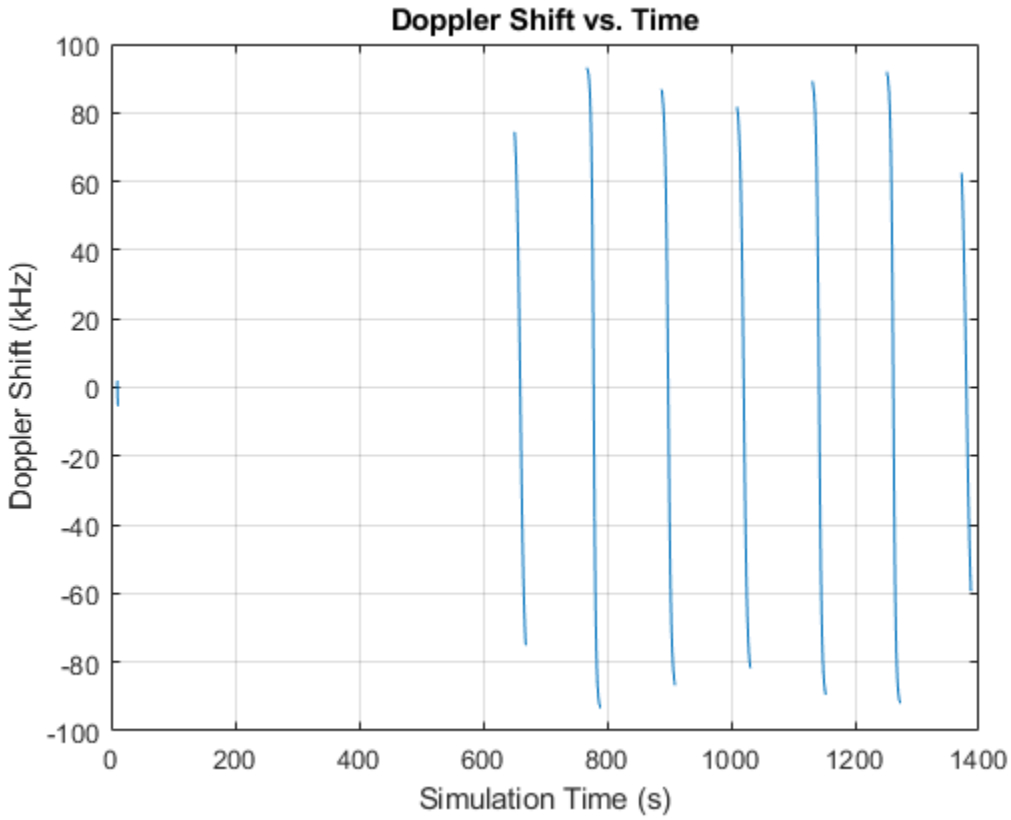
f_e is the emitted frequency, in Hz, and

f_o is the observed frequency, in Hz.

In this example, we consider the observed frequency from the standpoint of a receiving ground station. In that case, v_r is 0. We also consider a C band frequency of 5 GHz.

```
fe = 5e9;  % emitted frequency in Hz
fo = (((c ./ (c-dopV)) * fe) - 5e9);  % doppler shift in Hz
plot(fo/1e3)  % plot kHz
title("Doppler Shift vs. Time")
```

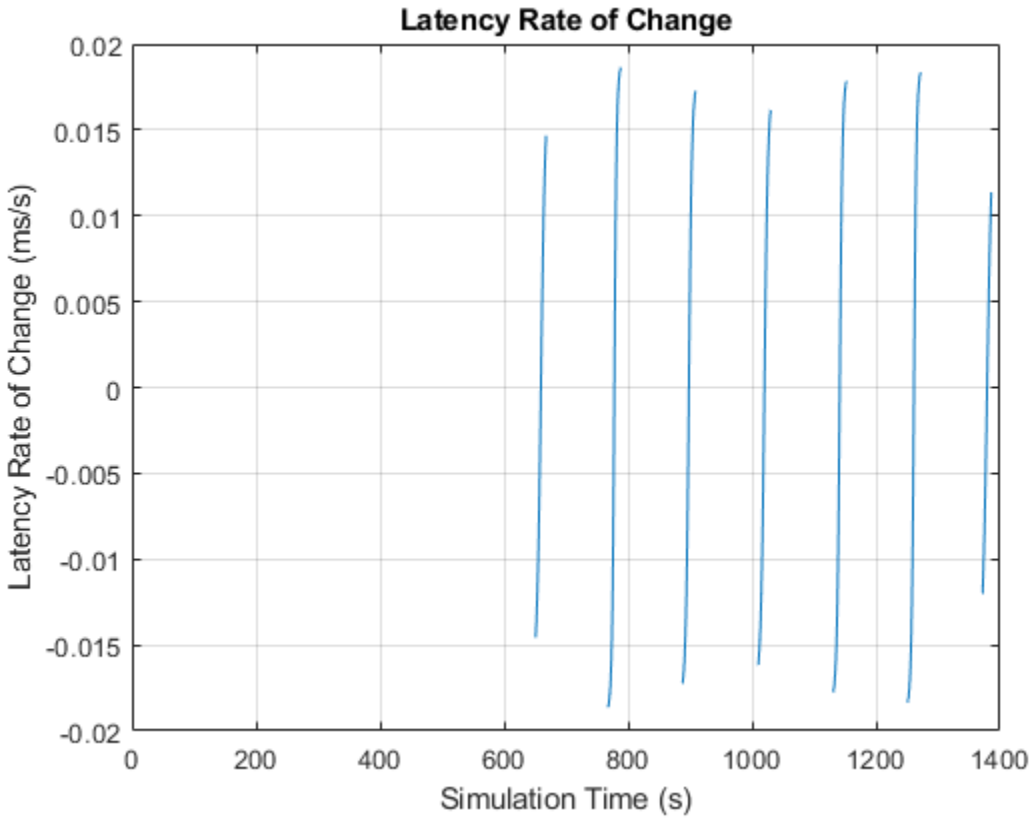
```
xlabel("Simulation Time (s)")
ylabel("Doppler Shift (kHz)")
grid on
```



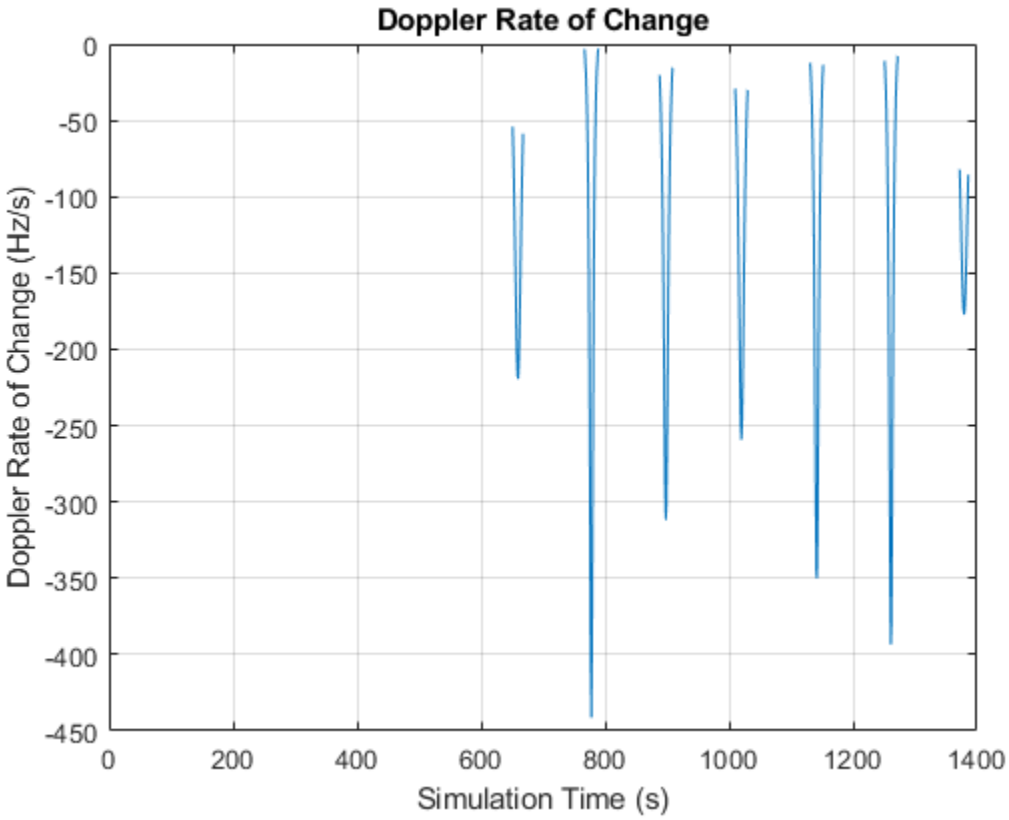
Calculate Rates of Change for Latency and Doppler

Satellite communications links need to be designed to track varying latencies and Doppler frequencies. Thus, it is important to calculate these quantities.

```
latencyRate = diff(latency)/sampleTime;
plot(1000*latencyRate)  % ms/s
title("Latency Rate of Change")
xlabel("Simulation Time (s)")
ylabel("Latency Rate of Change (ms/s)")
grid on
```

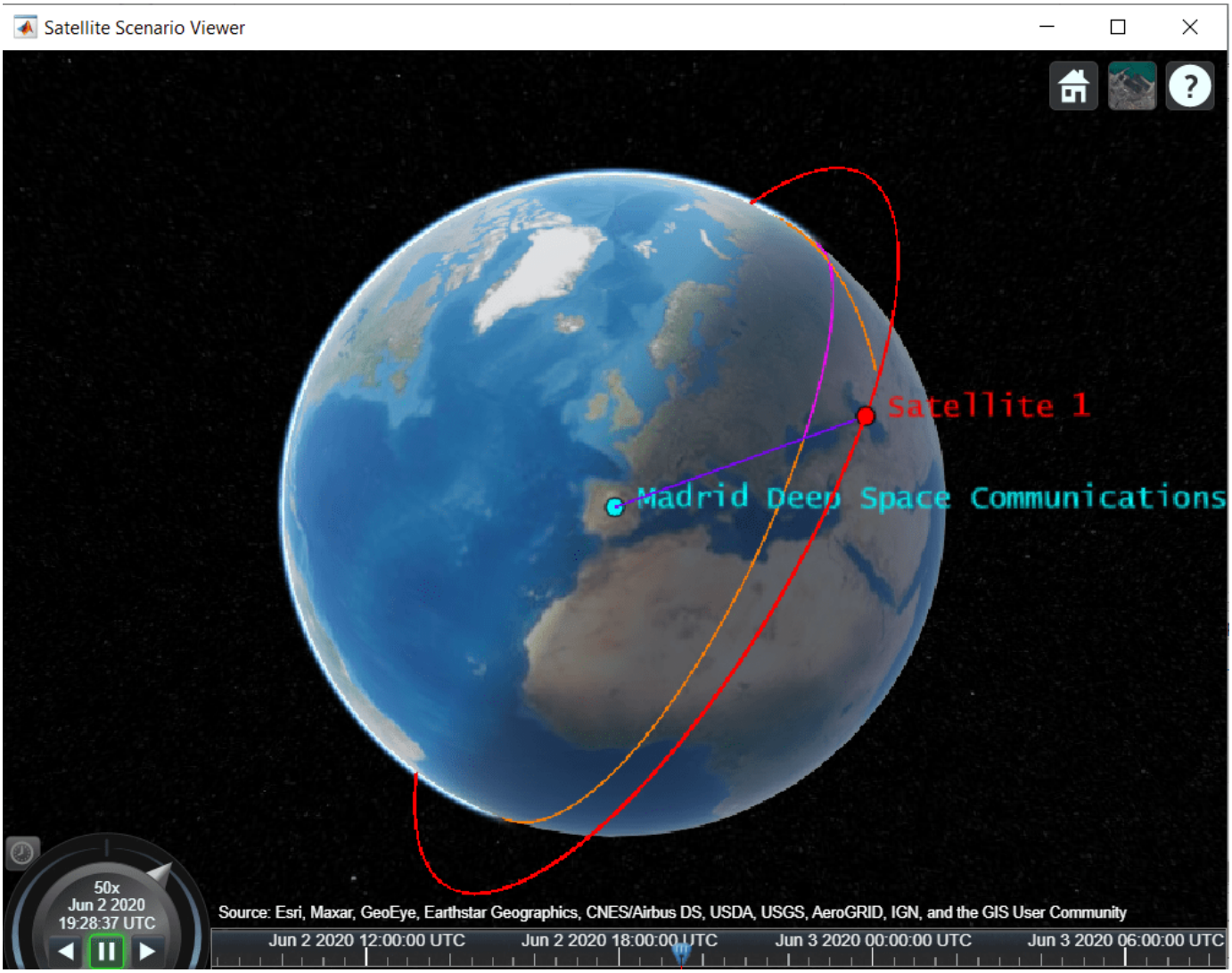



```
dopplerRate = (diff(fo)/sampleTime);
plot(dopplerRate)
title("Doppler Rate of Change")
xlabel("Simulation Time (s)")
ylabel("Doppler Rate of Change (Hz/s)")
grid on
```



Play the scenario with the satellite and ground station.

```
play(sc)
```



See Also

Objects
`satelliteScenario` | `satellite` | `groundTrack` | `groundStation` | `access`

Functions
`orbitalElements` | `aer` | `play`

Related Examples

- “Satellite Constellation Access to a Ground Station” on page 1-17
- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”

- “Satellite Scenario Basics”

Interference from Satellite Constellation on Communications Link

This example shows how to analyze interference on a downlink from a constellation of satellites. The downlink occurs from a geosynchronous satellite to a ground station located in the Pacific Ocean. The interfering constellation consists of 40 satellites in a low-Earth orbit. This example determines the times at which the downlink is closed, the carrier to noise plus interference ratio, and the link margin.

Create Satellite Scenario

Create a satellite scenario. Define the start time and stop time of the scenario. Set the sample time to 60 seconds.

```
startTime = datetime(2021,3,17);                        % 17 March 2021 12:00 AM UTC
stopTime = startTime + days(1);                         % 18 March 2021 12:00 AM UTC
sampleTime = 60;                                        % In s
sc = satelliteScenario(startTime,stopTime,sampleTime);
```

Add Geosynchronous Orbit Satellite

Add a satellite in a geosynchronous orbit by specifying its Keplerian orbital elements. This satellite is the satellite from which data is downlinked.

```
earthAngularVelocity = 0.0000729211585530;              % In rad/s
orbitalPeriod = 2*pi/earthAngularVelocity;              % In s
earthStandardGravitationalParameter = 398600.4418e9;    % In m^3/s^2
semiMajorAxis = (earthStandardGravitationalParameter ...
    /(earthAngularVelocity^2))^(1/3);                   % In m
eccentricity = 0;
inclination = 8;                                        % In degrees
raan = 0;                                               % In degrees
argOfPeriapsis = 0;                                     % In degrees
trueAnomaly = 0;                                        % In degrees
geoSat = satellite(sc, semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    raan, ...
    argOfPeriapsis, ...
    trueAnomaly, ...
    "Name","Geo Sat");
```

Add Interfering Satellite Constellation

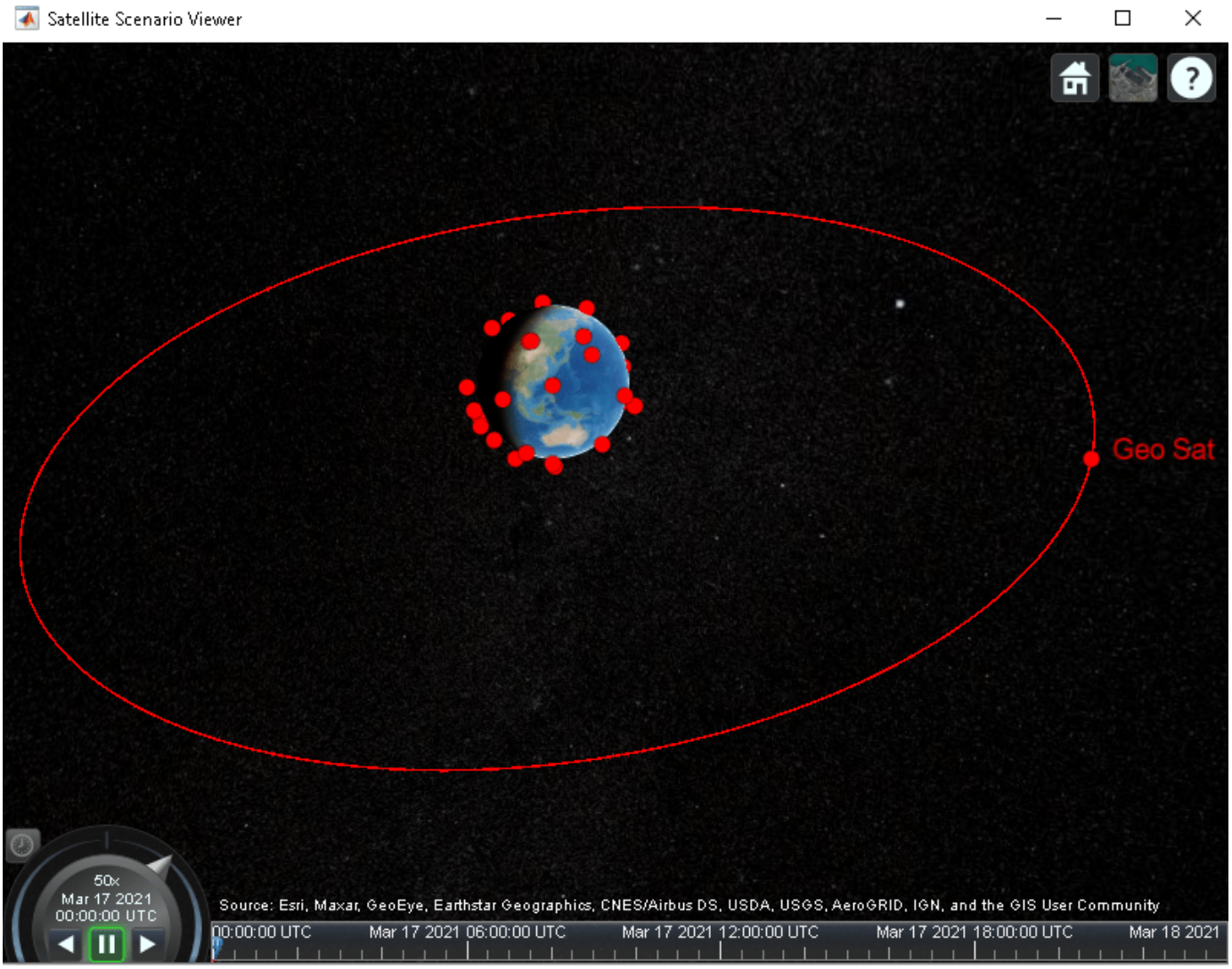
Add the interfering satellite constellation from a two-line-element (TLE) file. Declutter the visualizations that you launch later in this example by setting the `ShowLabel` property of this satellite constellation to `false`.

```
interferingSat = satellite(sc,"leoSatelliteConstellation.tle");
interferingSat.ShowLabel = false;
```

Launch Satellite Scenario Viewer

Launch the Satellite Scenario Viewer. Hide the orbits of the interfering satellites to further declutter the visualizations.

```
v = satelliteScenarioViewer(sc);
hide(interferingSat.Orbit);
```



Add Transmitter to Geosynchronous Orbit Satellite

Add a transmitter to the geosynchronous orbit satellite. This transmitter is used for the downlink. Define the antenna specifications and set the operating carrier frequency to 30 GHz.

```
txGeoFreq = 30e9;                   % In Hz
txGeoSat = transmitter(geoSat, ...
    "Frequency",txGeoFreq, ...      % In Hz
    "Power",25);                    % In dBW
gaussianAntenna(txGeoSat, ...
    "DishDiameter",1);              % In m
```

Add Transmitter to Interfering Satellites

Add a transmitter to each satellite in the interfering constellation, and then define the antenna specifications. These transmitters are the ones that interfere with the downlink from the

geosynchronous orbit satellite. Set the operating carrier frequency of the interfering satellites to 29.99 GHz. The example assigns each interfering satellite a random power in the range from 10 to 20 dBW.

```
interferenceFreq = 29.99e9;                                    % In Hz
rng("default");
for idx = 1:numel(interferingSat)
    txInterferingSat = transmitter(interferingSat(idx), ...
        "Frequency",interferenceFreq, ...                      % In Hz
        "Power",10+10*rand);                                   % In dBW
    gaussianAntenna(txInterferingSat, ...
        "DishDiameter",0.2);                                   % In m
end
```

Add Ground Station

Add a ground station to satellite scenario by specifying its latitude and longitude.

```
gs = groundStation(sc, ...
    0, ...                    % Latitude in degrees
    180, ...                  % Longitude in degrees
    "Name","Ground station");
```

Add Gimbal to Ground Station

Add a gimbal to the ground station by specifying its mounting location and angles. This gimbal is used to make the receiver antenna of the ground station track the geosynchronous orbit satellite.

```
gim = gimbal(gs, ...
    "MountingLocation",[0;0;-5], ... % In m
    "MountingAngles",[0;180;0]);     % In m
```

Add Receiver to Gimbal

Add a receiver to the ground station gimbal, specifying the receiver's mounting location with respect to the gimbal. Define the specifications of the receiver antenna.

```
rxGs = receiver(gim, ...
    "MountingLocation",[0;0;1]); % In m
gaussianAntenna(rxGs, ...
    "DishDiameter",3);           % In m
```

Set Tracking Targets for Satellites and Gimbal

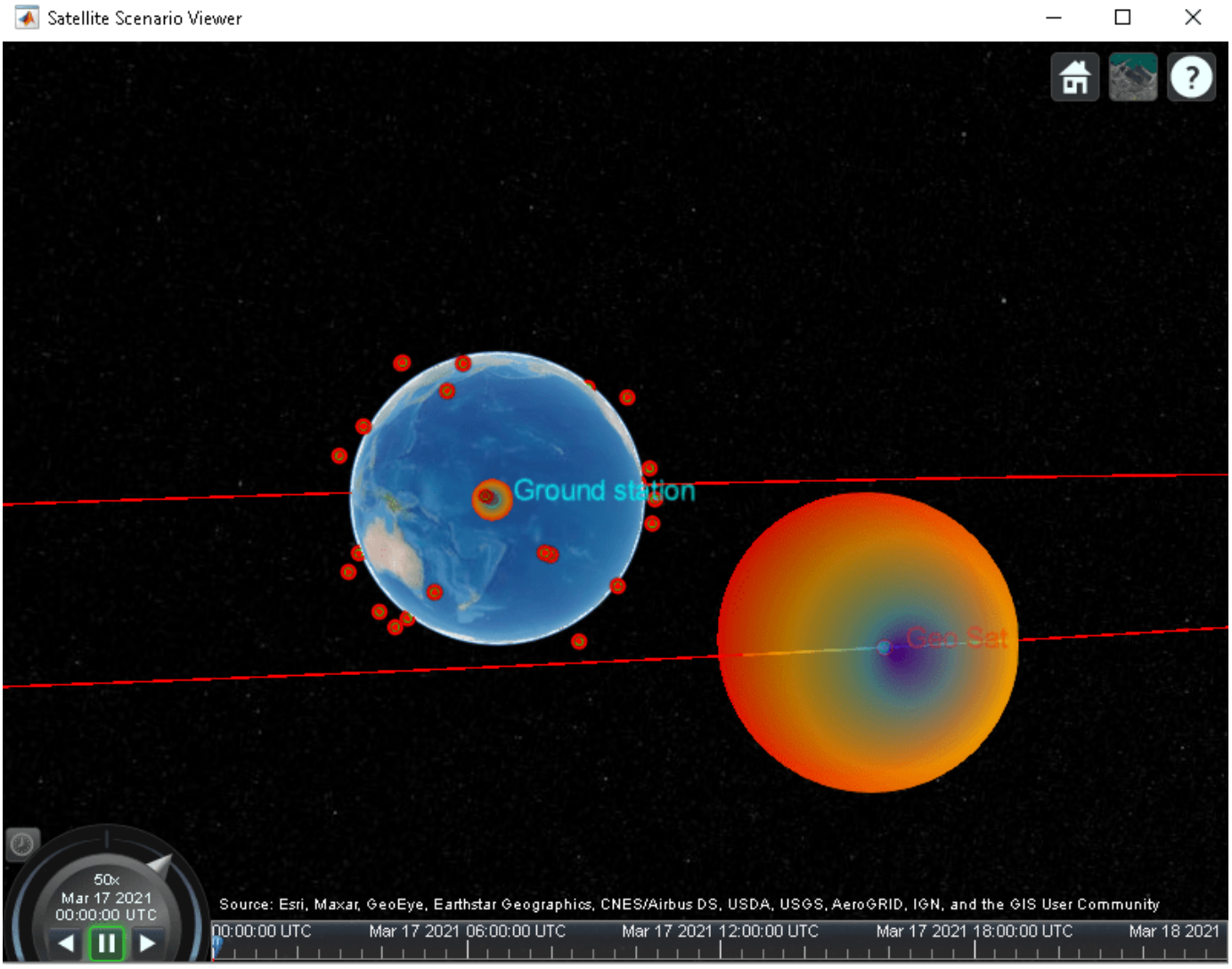
Set the satellites to track the ground station, and the ground station gimbal to track the geosynchronous orbit satellite. This ensures that the transmitter antennas on board each satellite track the ground station and that the ground station antenna tracks the geosynchronous orbit satellite. Setting the interfering satellite transmitters to track the ground station results in the worst-case interference on the downlink.

```
pointAt(geoSat,gs);
pointAt(gim,geoSat);
for idx = 1:numel(interferingSat)
    pointAt(interferingSat(idx),gs);
end
```

Visualize Radiation Pattern of Antennas Involved in Downlink

Visualize the radiation pattern of the transmitter antenna on board the geosynchronous orbit satellite and the receiver on board the ground station.

```
pattern(geoSat.Transmitters);
pattern(rxGs,geoSat.Transmitters.Frequency);
```



Create Desired Downlink

Create a downlink from the transmitter on board the geosynchronous orbit satellite to the receiver on board the ground station. This link is the downlink which encounters interference from the satellite constellation.

```
downlink= link(txGeoSat,rxGs);
```

Create Interfering Links

Create a link between the transmitter on board each interfering satellite in the constellation and the receiver on board the ground station. These links are the interferer links with the desired downlink.

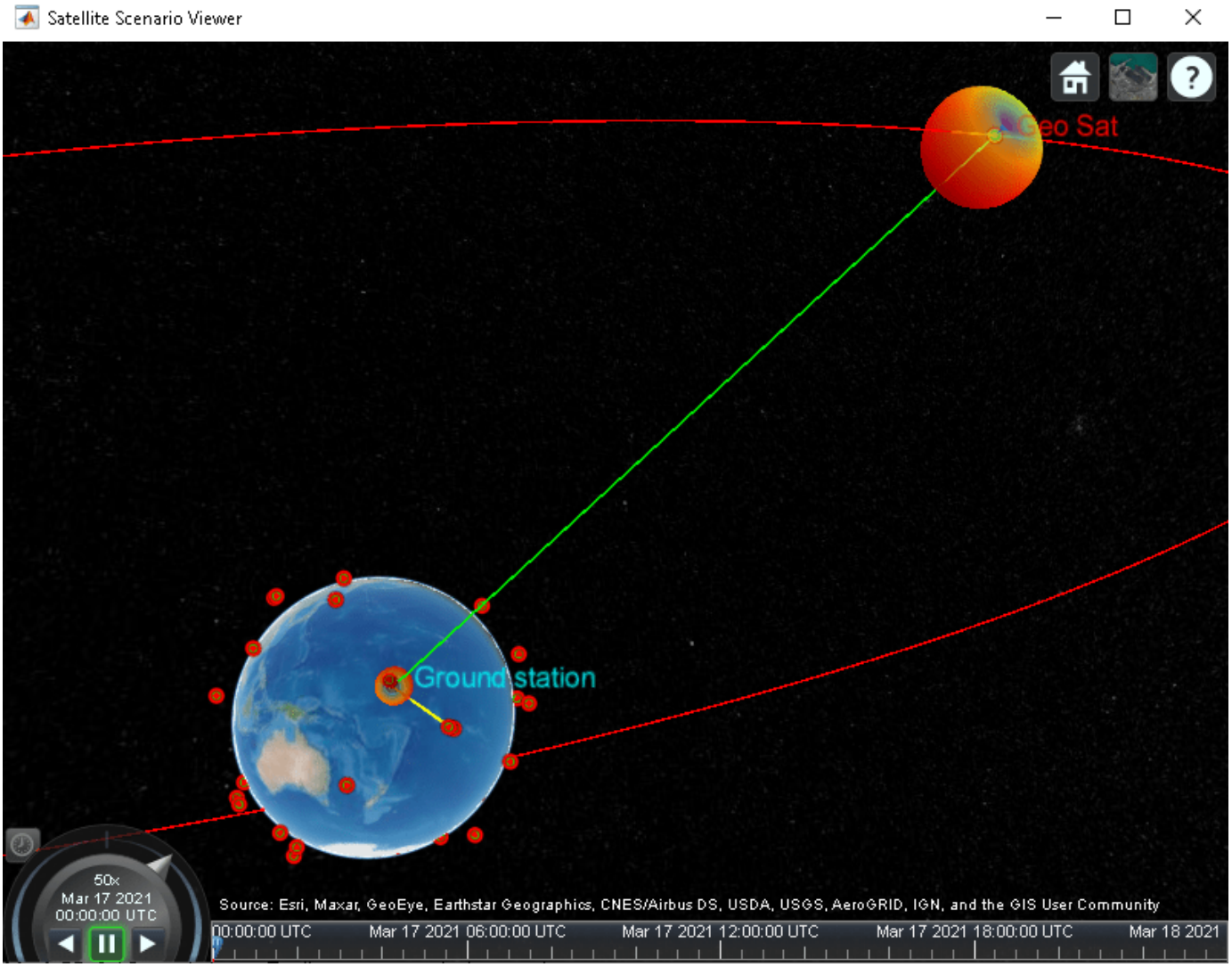
```
for idx = 1:numel(interferingSat)
    link(interferingSat(idx).Transmitters,rxGs);
end
```

Create Access Analysis Between Interfering Satellite Constellation and Ground Station

Add an access analysis between each satellite in the interfering constellation and the ground station. This analysis enables the visualization of interference. Any time a satellite in the constellation is visible to the ground station, there is some level of interference from that visible satellite.

```
for idx = 1:numel(interferingSat)
    ac = access(interferingSat(idx),gs);
    ac.LineColor = [1 1 0];              % Yellow
end
```

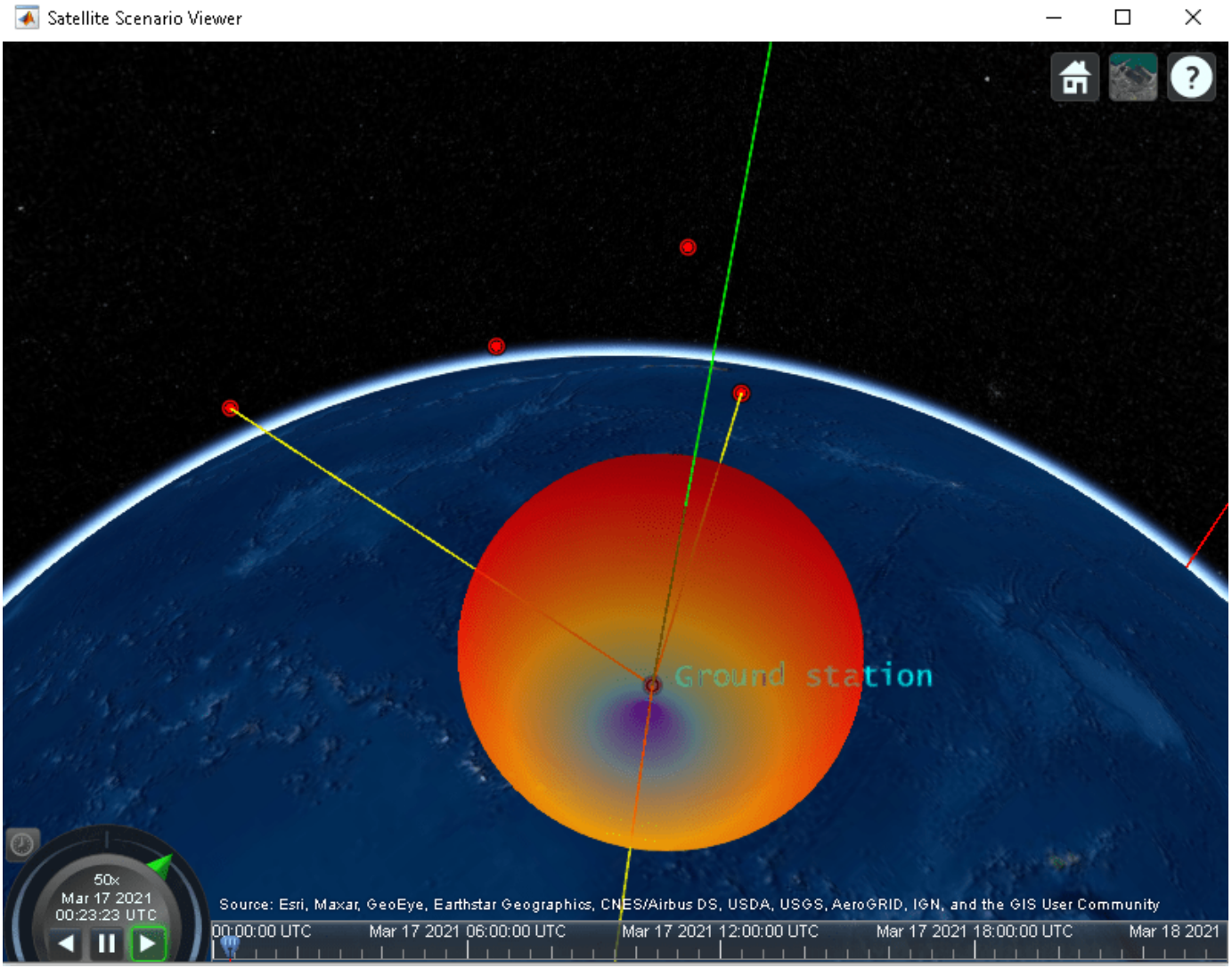
The presence of the green line between the transmitter on board the geosynchronous satellite and the receiver on board the ground station signifies that the downlink can be closed successfully assuming no interference from the satellite constellation exists. The presence of yellow lines between a given satellite in the constellation and the ground station signifies that an interference from that satellite exists.



Play Scenario

Play the scenario in the Satellite Scenario Viewer. Set the target of the camera to the ground station. Place the camera at a -10 degree latitude, 170 degree longitude, and 4,000 km altitude. Additionally, set the camera heading and pitch angles to 40 and -60 degrees, respectively.

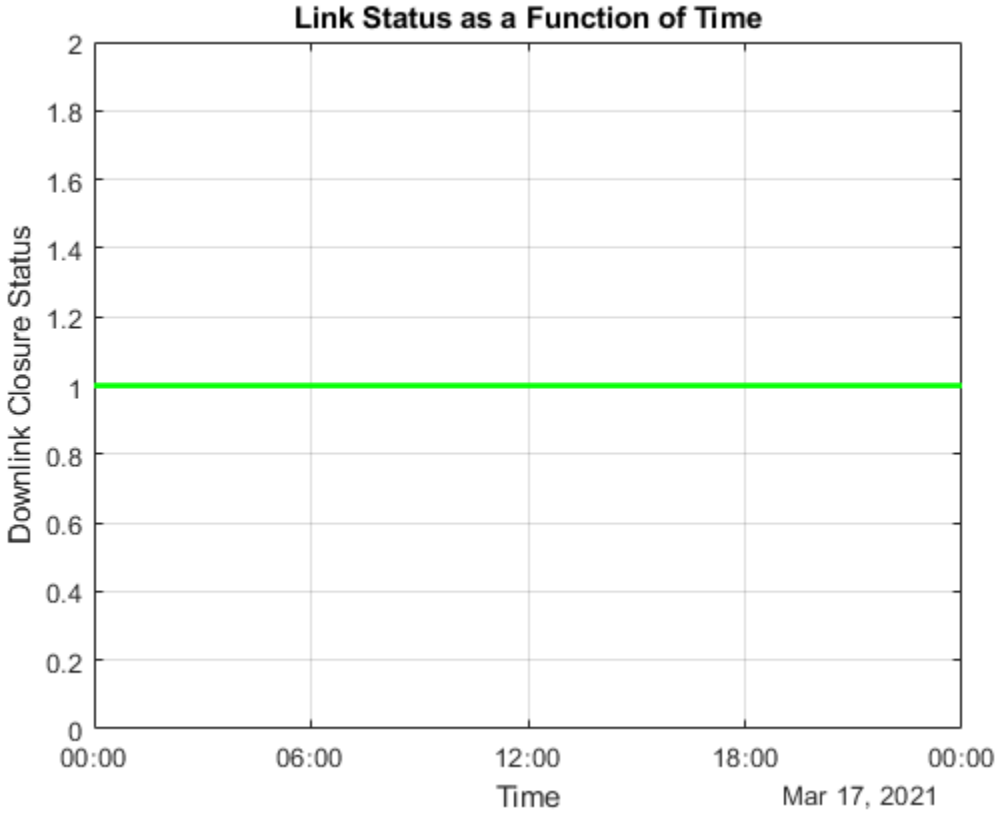
```
play(sc);
camtarget(v,gs);
campos(v,-10,170,4000000);
camheading(v,40);
campitch(v,-60);
```



Plot Downlink Closure Status Neglecting Interference

Determine the closure status of the desired downlink from the geosynchronous orbit satellite. The `linkStatus` function neglects interference from other transmitters. Any time the downlink is close, the status is true. Otherwise, the status is false. The status is indicated by 1 and 0, respectively in the plot. Because the satellite involved in the downlink is in a geosynchronous orbit above the ground station, the downlink is closed for the duration of the scenario.

```
[downlinkStatus,t] = linkStatus(downlink);
plot(t,downlinkStatus,"-g","LineWidth", 2);
xlabel("Time");
ylabel("Downlink Closure Status");
title("Link Status as a Function of Time");
grid on;
```



Calculation of Link Status

Although the `linkStatus` function neglects interference when determining the times when the downlink is closed, manually calculate the interference based on data that can be extracted from the scenario is possible. To do this manual calculation, understanding how `linkStatus` calculates the link closure, as shown in these steps, is necessary.

1) Calculate the effective isotropic radiated power (EIRP):

$$EIRP_{Tx} = P_{Tx} - LOSS_{Tx} + G_{TxAntenna},$$

where:

- $EIRP_{Tx}$ is the EIRP of the transmitter antenna (in dBW).
- P_{Tx} is the transmitter power (in dBW).
- $LOSS_{Tx}$ is the total system loss of the transmitter (in dB).
- $G_{TxAntenna}$ is the gain of the transmitter antenna in the direction of the receiver antenna (in dB).

2) In the satellite scenario, all path loss computations assume a free space propagation model. Calculate the free space path loss from the transmitter to the receiver antenna as

$$FSPL = 10\log_{10}\left(\left(\frac{4\pi df}{c}\right)^2\right),$$

where:

- $FSPL$ is the free space path loss from the transmitter to the receiver antenna (in dB).
- d is the distance between the transmitter and the receiver antenna (in m).
- f is the transmitter frequency (in Hz).
- c is the speed of light in vacuum (in m/s).

3) Calculate the received isotropic power as

$$RIP_{Rx} = EIRP_{Tx} - FSPL,$$

where RIP_{Rx} is the received isotropic power at the received antenna (in dBW).

4) Calculate the carrier to noise density ratio as

$$C/N_0 = RIP_{Rx} + (G/T)_{Rx} - 10\log_{10}k_B - LOSS_{RxSystem},$$

where:

- C/N_0 is the carrier to noise density ratio (in dB).
- $(G/T)_{Rx}$ is the gain to noise temperature ratio of the receiver antenna (in dB/K).
- k_B is the Boltzmann constant (in $m^2\,\mathrm{kg}\,s^{-2}\,K^{-1}$).
- $LOSS_{Rx}$ is the total system loss of the transmitter.

The receiver antenna gain is computed in the direction of the transmitter antenna. The noise temperature is assumed to be a constant and is derived from the `GainToNoiseTemperatureRatio` property of the receiver, which in turn is based on the receiver antenna gain along the z-axis of the receiver. In essence:

$$(G/T)_{z-axis} = (G_{Rx})_{z-axis} - 10\log_{10}T,$$

where:

- $(G/T)_{z-axis}$ is the receiver antenna gain to noise temperature ratio along the z-axis of the receiver (in dB).
- $(G_{Rx})_{z-axis}$ is the receiver antenna gain in the direction of the z-axis of the receiver (in dB).
- T is the noise temperature (in K).

Rearranging the above equation results in

$$10\log_{10}T = (G_{Rx})_{z-axis} - (G/T)_{z-axis}.$$

Consequently,

$$(G/T)_{Rx} = (G_{Rx}) - 10\log_{10}T.$$

5) Calculate the receiver energy per bit to noise power spectral density ratio as

$$E_b/N_0 = C/N_0 - 10\log_{10}(BITRATE) - 60,$$

where:

- E_b/N_0 is the received energy per bit to noise power density ratio (in dB).
- $BITRATE$ is the bit rate of the link (in Mbps).

The 60 value appears in the equation because the bit rate is in Mbps.

6) Calculate the link margin as

$MARGIN = E_b/N_0 - (E_b/N_0)_{Required}$,

where:

- $MARGIN$ is the link margin (in dB).
- $(E_b/N_0)_{Required}$ is the minimum received energy per bit to noise power spectral density ratio that is required to close the link (in dB).

The link is closed when the link margin is greater than or equal to 0 dB.

To account for interference, the signal received at the ground station antenna from each interfering transmitter must be treated as noise and must be added to the noise power. To compute the new received Eb/No and link margin, the received signal power from each transmitter in the scenario must be calculated.

Calculate Received Signal Power After Receiver Antenna Corresponding to Downlink from Geosynchronous Orbit Satellite

The received power after antenna is the quantity $RIP_{Rx} + G_{RxAntenna}$. This received power can be obtained by starting from E_b/N_0 and using the equations in the previous section backward. First, the received signal power corresponding to the transmitter on board the geosynchronous orbit satellite must be computed. Retrieve the received E_b/N_0 corresponding to this transmitter.

```
ebnoDownlink = ebno(downlink); % In dB
```

Use these equations to calculate the corresponding received signal power after antenna.

$$C/N_0 = E_b/N_0 + 10\log_{10}(BITRATE) + 60$$

$$RIP_{Rx} + G_{RxAntenna} = C/N_0 + 10\log_{10}k_B + 10\log_{10}T + LOSS_{Rx}$$

```
T = HelperGetNoiseTemperature(txGeoSat.Frequency,rxGs);                          % In K
CNoDownlink = ebnoDownlink + 10*log10(txGeoSat.BitRate) + 60;
kb = physconst("Boltzmann");
downlinkSigPower = CNoDownlink + 10*log10(kb) + 10*log10(T) + rxGs.SystemLoss; % In dBW
```

Calculate Total Interfering Signal Power After Receiver Antenna from All Interfering Transmitters

The total interfering signal power after the receiver antenna from all interfering transmitters is calculated using these steps.

1) Calculate $RIP_{Rx} + G_{Rx}$, which corresponds to each interfering transmitter, using the same equations used for the transmitter from the geosynchronous orbit satellite.

2) Convert the quantities to Watts.

3) Add the quantities in Watts and the final sum is the total interfering power, I_W, in Watts.

```
% Retrieve the interfering links
lnkInterference = interferingSat.Transmitters.Links;

% Initialize an array to store the received Eb/No history corresponding to
% each interfering transmitter
ebnoInterference = zeros(numel(lnkInterference), numel(t));

% Retrieve the Eb/No history corresponding to each interfering transmitter
% and store it in the array
for idx = 1:numel(lnkInterference)
    ebnoInterference(idx,:) = ebno(lnkInterference(idx));
end

% Calculate the interfering signal power corresponding to each interfering
% transmitter in Watts (in this example, the bit rate of each interfering
% transmitter is the same as that of that of the geosynchronous orbit
% satellite)
bitRate = txGeoSat.BitRate;
CNoInterference = ebnoInterference + 10*log10(bitRate) + 60;
interferenceSigPower = CNoInterference + 10*log10(kb) ...
    + 10*log10(T) + rxGs.SystemLoss;
interferenceSigPowerW = 10.^(interferenceSigPower/10);       % In W

% Add the interfering signal power from each transmitter
interferenceSigPowerSumW = sum(interferenceSigPowerW);       % In W
```

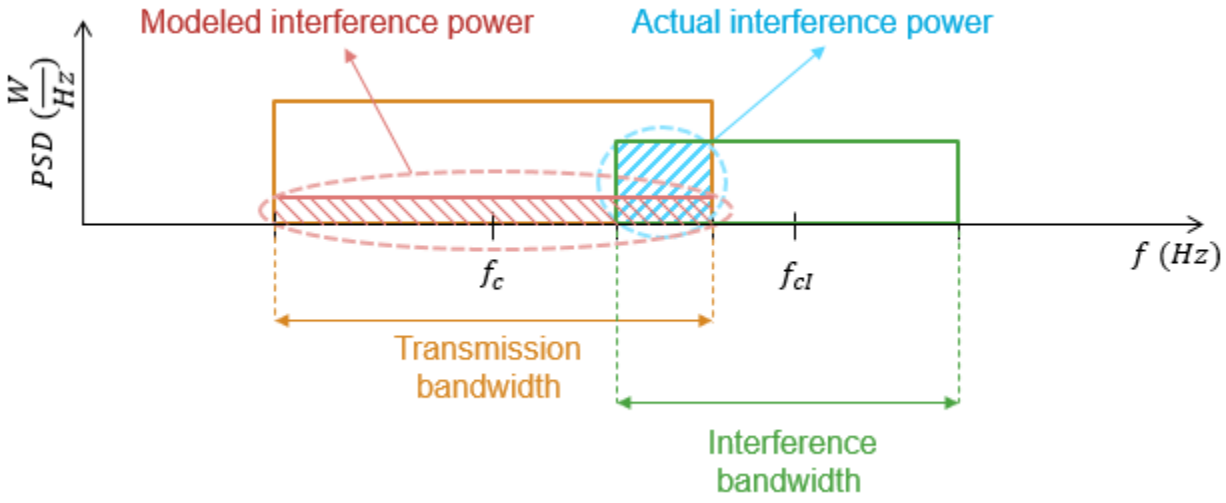
Calculate Contribution of Interfering Signal Power in Overlapped Bandwidth

Calculate the amount of total interfering signal power that contributes to interference in the signal bandwidth by following these steps.

1) Calculate the overlapping portion of the signal bandwidth with the bandwidth of the interferers. This example considers the transmission power of interfering satellites and geosynchronous satellite as constant across the whole bandwidth of respective geosynchronous satellite and interfering satellites.

2) Calculate the amount of interference power that acts as interference to signal bandwidth.

This diagram shows the power spectral density (PSD) plot, which shows the actual interference power and modeled interference power when the transmission bandwidth and interfering bandwidth overlap. The actual interference power is the area occupied by the interference power density in the overlapped bandwidth region. This actual interference power is then spread across the entire transmission bandwidth and assumed to be noise-like.



This example assumes that the transmission (or signal) bandwidth of the geosynchronous satellite is 30 MHz and that the bandwidth of the interfering signal is 20 MHz.

```
txBandwidth = 30e6;                                                 % In Hz
interferenceBandWidth = 20e6;                                       % In Hz

% Get the overlap portion of the interfering bandwidth and the bandwidth of
% interest. The assumption is to have same power across the whole
% bandwidth.
overlapFactor = getOverlapFactor(txGeoFreq,txBandwidth, ...
    interferenceFreq,interferenceBandWidth);

% Get the interference power that contributes as interference to the signal
% of interest from the total interference power
interferenceSigPowActual = interferenceSigPowerSumW*overlapFactor; % In W
```

Calculate Downlink Closure Status That Accounts for Interference

Calculate the downlink closure status that accounts for interference by following these steps.

1) Calculate the carrier to noise plus interference power density ratio as

$$C/(N_0 + I_0) = RIP_{Rx} + G_{Rx} - 10\log_{10}\left(\frac{I_W}{TxBandwidth} + k_B T\right) - LOSS_{Rx},$$

where:

- $C/(N_0 + I_0)$ is the carrier to noise plus interference power density ratio (in dB).
- I_W is the interference signal power after receiver antenna that interferes with the signal bandwidth (in W).
- $TxBandwidth$ is the downlink transmission bandwidth from geosynchronous satellite (in Hz).

2) Calculate the energy per bit to noise plus interference power spectral density ratio as

$$E_b/(N_0 + I_0) = C/(N_0 + I_0) - 10\log_{10}(BITRATE) - 60,$$

where $E_b/(N_0 + I_0)$ is the energy per bit to noise plus interference power spectral density ratio (in dB).

3) Calculate the link margin accounting for interference:

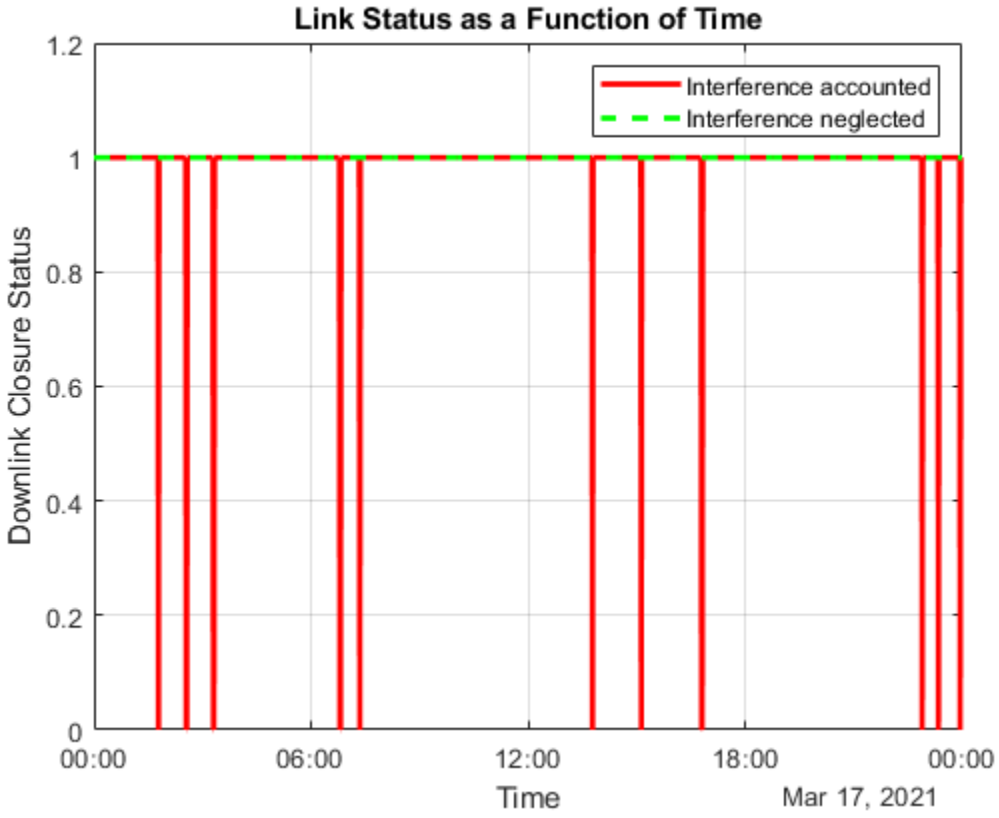
$MARGIN = E_b/(N_0 + I_0) - (E_b/N_0)_{Required}$.

```
% Calculate link status in the presence of interference
CNoPlusInterference = downlinkSigPower - ...
    10*log10((interferenceSigPowActual/txBandwidth) + T*kb) - rxGs.SystemLoss;
ebNoPlusInterference = CNoPlusInterference - 10*log10(bitRate) - 60;
marginWithInterference = ebNoPlusInterference - rxGs.RequiredEbNo;
downlinkStatusWithInterference = marginWithInterference >= 0;
```

Plot Downlink Closure Status Accounting for Interference

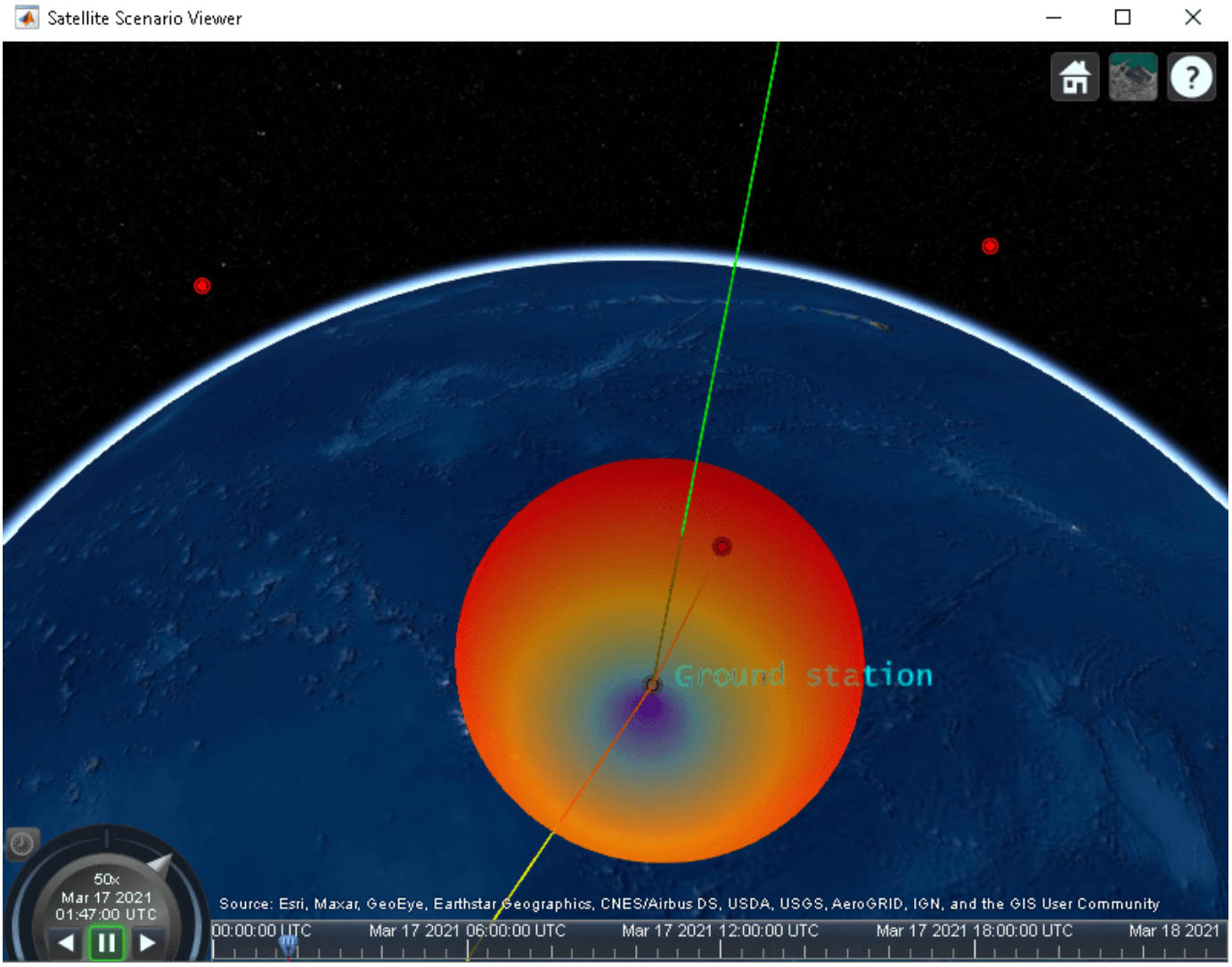
Plot the new downlink closure status that accounts for interference. Compare the new link status with the previous case when interference was neglected.

```
plot(t,downlinkStatusWithInterference,"-r",t,downlinkStatus,"--g","LineWidth",2);
legend("Interference accounted","Interference neglected");
xlabel("Time");
ylabel("Downlink Closure Status");
title("Link Status as a Function of Time");
ylim([0 1.2]);
grid on
```

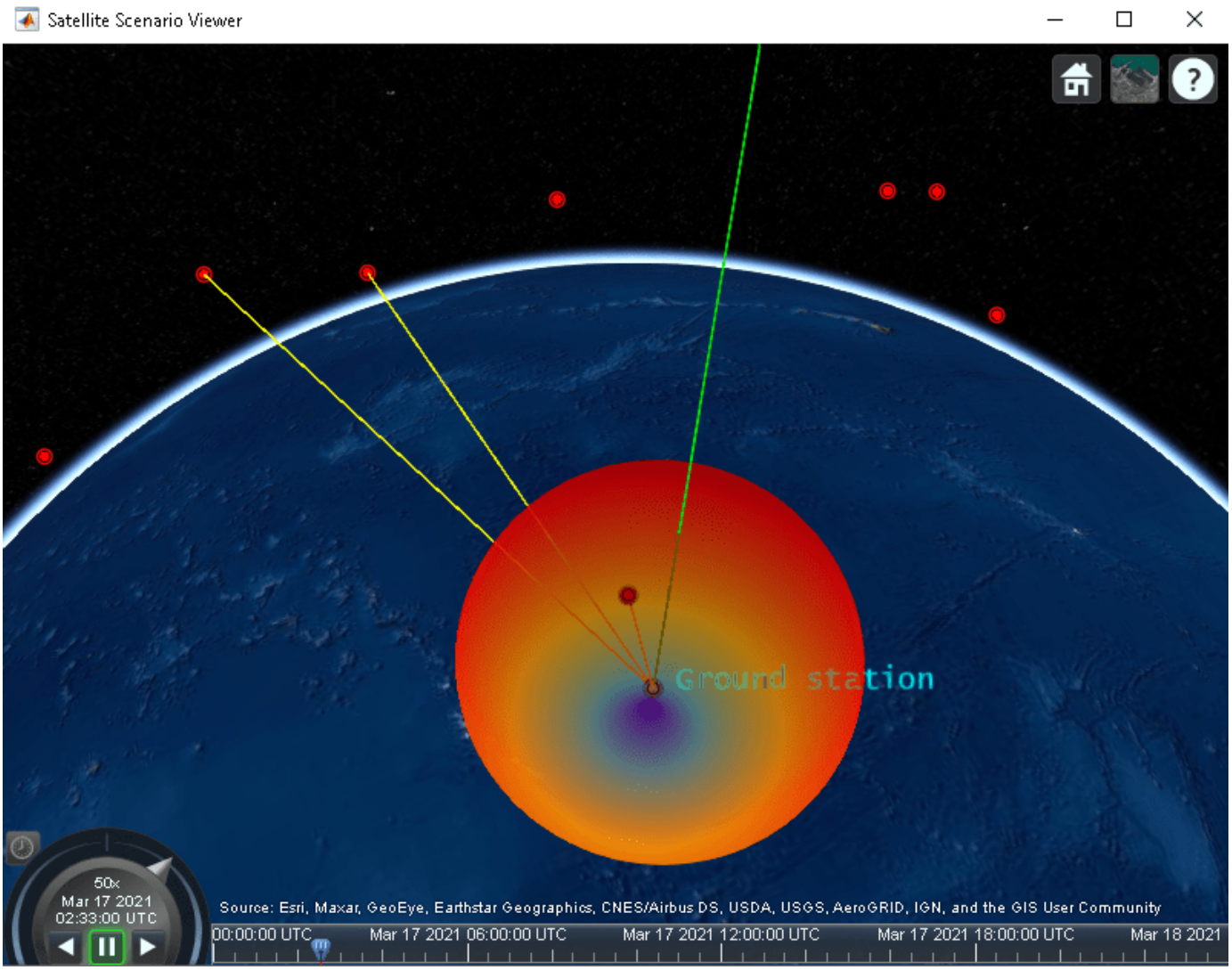


The plot shows times exist when the downlink cannot be closed because of interference. The interference is particularly severe when at least one satellite flies overhead. For instance, the plot shows that the downlink is broken at times such as 1:47 AM, 2:33 AM, and 3:26 AM. Set the `CurrentTime` property of the Satellite Scenario Viewer to these times to confirm that an interfering satellite overflying the ground station exists.

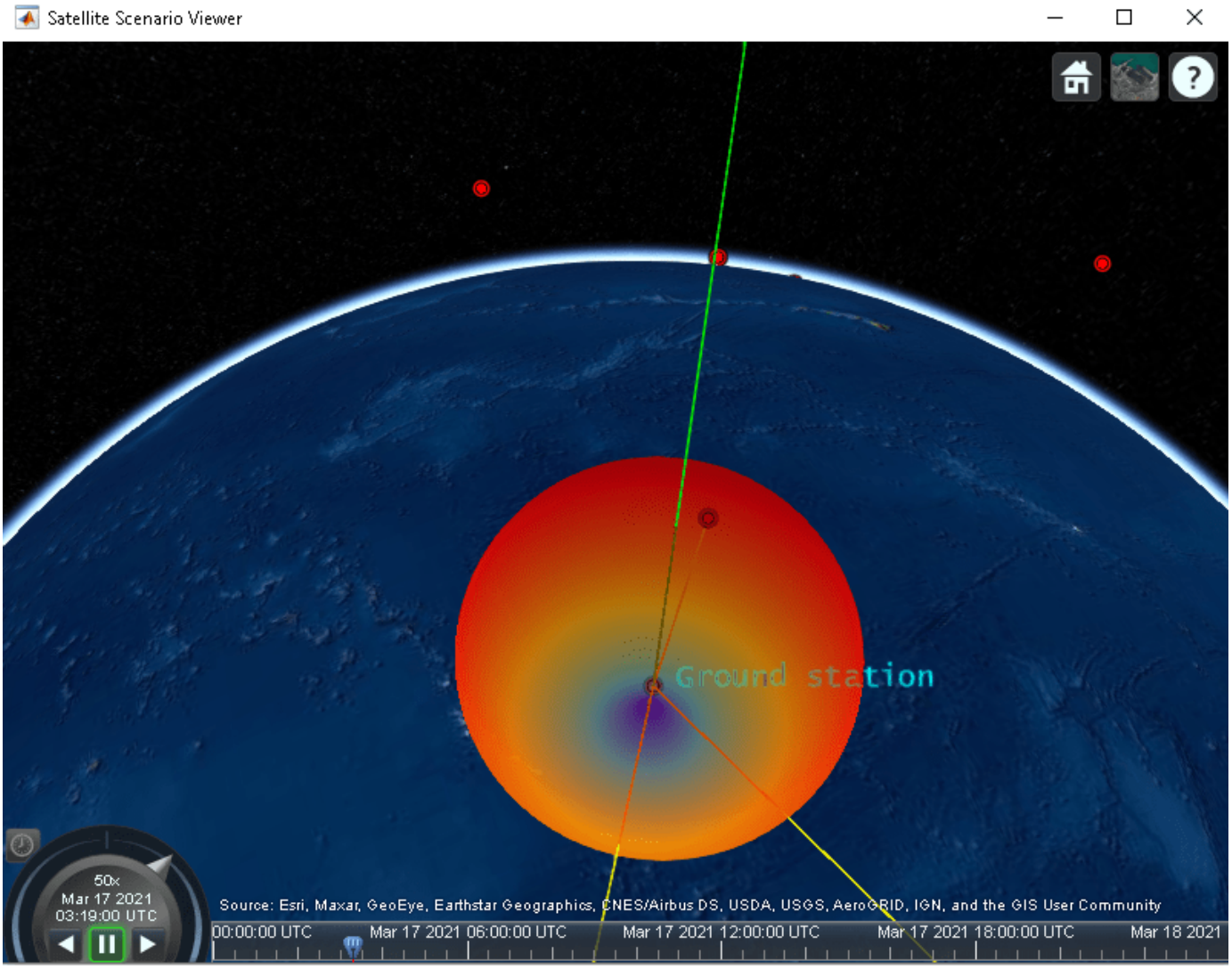
```
v.CurrentTime = datetime(2021,3,17,1,47,0);
```




```
v.CurrentTime = datetime(2021,3,17,2,33,0);
```




```
v.CurrentTime = datetime(2021,3,17,3,19,0);
```



Calculate Carrier to Noise Ratio and Carrier to Noise Plus Interference Ratio

Calculate the carrier to noise ratio (CNR) and carrier to noise plus interference ratio (CNIR) from the carrier to noise density ratio and carrier to noise plus interference power density as:

$C/N = C/N_0 - 10\log_{10}(TxBandwidth)$ and

$C/(N + I) = C/(N_0 + I_0) - 10\log_{10}(TxBandwidth)$,

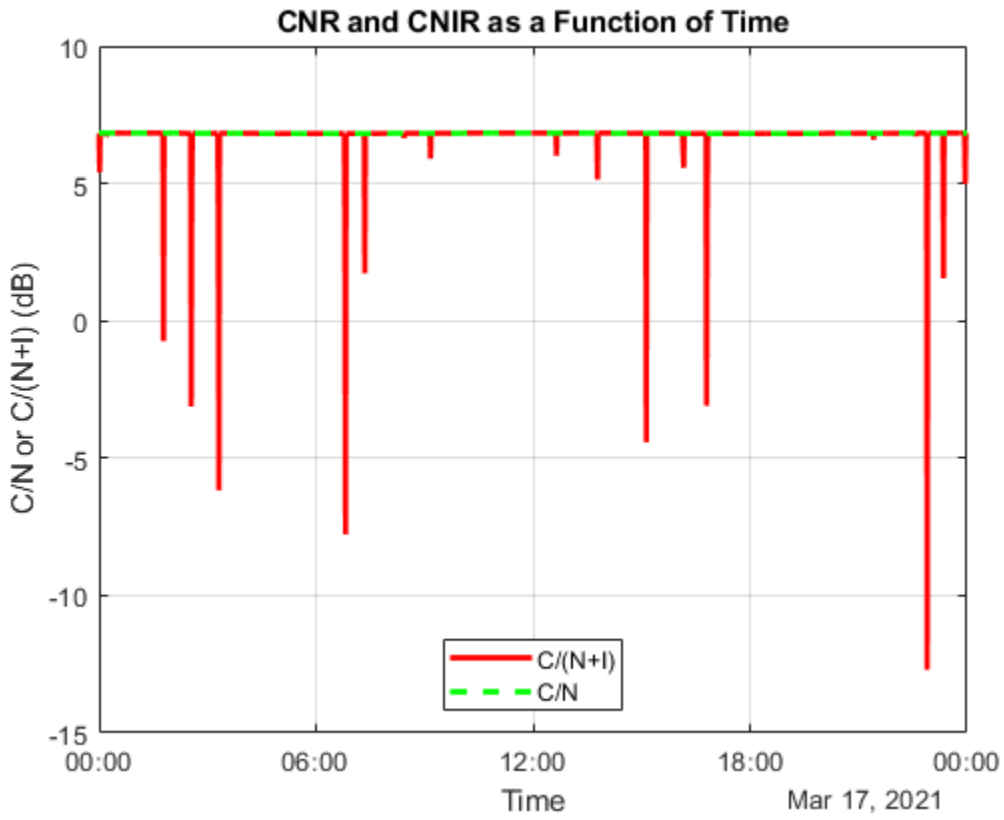
where:

- C/N is the carrier to noise ratio
- $C/(N + I)$ is the carrier to noise plus interference ratio.
- $TxBandwidth$ is the downlink transmission bandwidth from geosynchronous satellite (in Hz).

```
cByN = CNoDownlink - 10*log10(txBandwidth);
cByNPlusI = CNoPlusInterference - 10*log10(txBandwidth);
```

Plot C/N and $C/(N+I)$.

```
plot(t,cByNPlusI,"-r",t,cByN,"--g","LineWidth",2);
legend("C/(N+I)", "C/N","Location","south");
xlabel("Time");
ylabel("C/N or C/(N+I) (dB)");
title("CNR and CNIR as a Function of Time");
grid on
```



The downward spikes indicate interference. The more severe spikes are caused when simultaneous interference from multiple satellites exists or when an interfering satellite overflies the ground station.

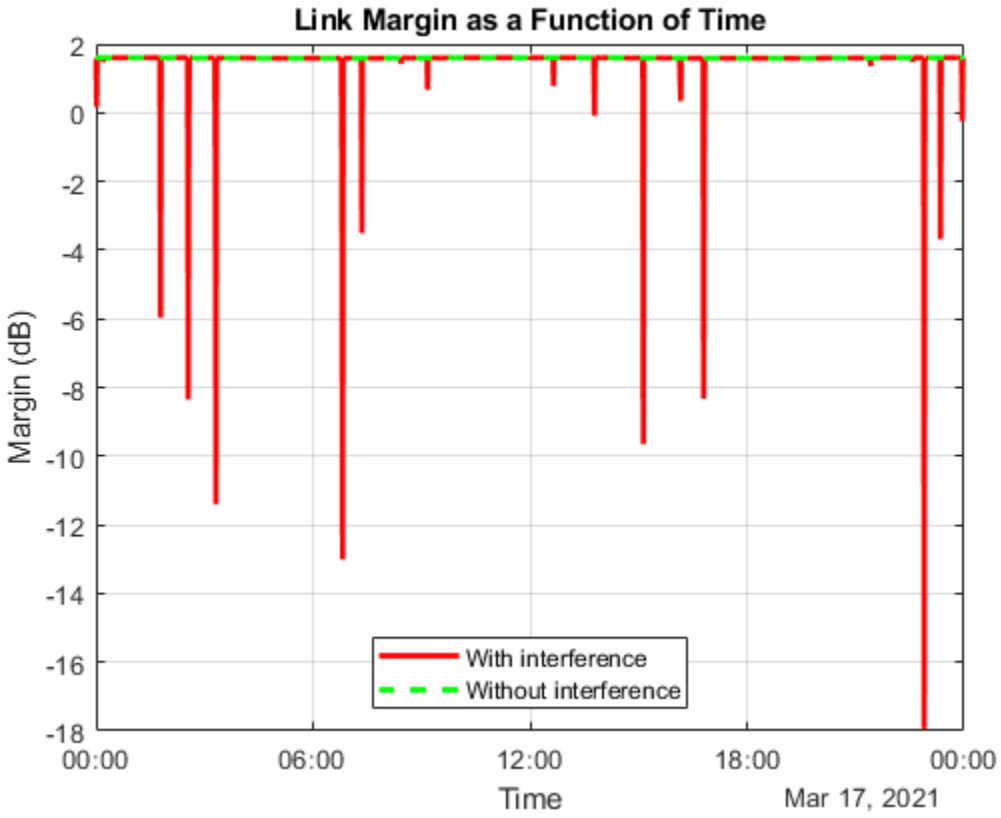
Compare Link Margins with and without Interference

Calculate the link margin without interference.

```
marginWithoutInterference = ebnoDownlink - rxGs.RequiredEbNo;
```

Plot the link margins with and without interference.

```
plot(t,marginWithInterference,"-r",t,marginWithoutInterference,"--g","LineWidth",2);
legend("With interference","Without interference","Location","south");
xlabel("Time");
ylabel("Margin (dB)");
title("Link Margin as a Function of Time");
grid on
```



Any time the link margin is greater than or equal to 0 dB, the downlink is closed. With interference, times exist when the link margin dips below 0 dB and at these times the downlink is broken because of interference.

Further Exploration

This example demonstrates how to analyze interference on a satellite communication link. The link closure times are a function of these parameters:

- The orbit of the satellites
- The position of the ground station
- The specifications of the transmitters and the receiver
- The signal and interference bandwidth

Modify these parameters to observe their influence on the level of interference on the link.

Helper Functions

The example uses the helper function HelperGetNoiseTemperature to obtain the noise temperature of the receiver antenna.

The example also uses this local function to compute the amount of overlap between the transmission bandwidth and the interfering bandwidth.

```
function overlapFactor = getOverlapFactor(txFreq,txBW,interferenceFreq,interferenceBW)
% getOverlapFactor provides the amount of interference bandwidth overlapped
```

```
% with transmission bandwidth

    txFreq_Limits = [txFreq-(txBW/2) txFreq+(txBW/2)];
    interferenceFreq_Limits = [interferenceFreq-(interferenceBW/2) ...
        interferenceFreq+(interferenceBW/2)];
    if (interferenceFreq_Limits(2) < txFreq_Limits(1)) || ...
            (interferenceFreq_Limits(1) > txFreq_Limits(2))
        % If no overlap exists between transmission bandwidth and
        % interfering bandwidth, then overlap factor is 0
        overlapFactor = 0;
    elseif (interferenceFreq_Limits(2) <= txFreq_Limits(2)) && ...
            (interferenceFreq_Limits(1) >= txFreq_Limits(1))
        % If interfering bandwidth lies completely within transmission
        % bandwidth, then overlap factor is 1
        overlapFactor = 1;
    elseif (interferenceFreq_Limits(2) > txFreq_Limits(2)) && ...
            (interferenceFreq_Limits(1) < txFreq_Limits(1))
        % If transmission bandwidth lies completely within interfering
        % bandwidth, then overlap factor is the ratio of transmission
        % bandwidth with that of interference bandwidth
        overlapFactor = txBW/interferenceBW;
    elseif (interferenceFreq_Limits(2) <= txFreq_Limits(2)) && ...
            (interferenceFreq_Limits(1) <= txFreq_Limits(1))
        % If the start edge of transmission bandwidth lies within
        % interfering bandwidth, then overlap factor is the ratio of
        % difference from last edge of interfering bandwidth and first edge
        % of signal bandwidth, with that of interference bandwidth
        overlapFactor = (interferenceFreq_Limits(2)-txFreq_Limits(1))/interferenceBW;
    else
        % If the last edge of transmission bandwidth lies within
        % interfering bandwidth, then overlap factor is the ratio of difference
        % from last edge of signal bandwidth and first edge of interfering
        % bandwidth, with that of interference bandwidth
        overlapFactor = (-interferenceFreq_Limits(1)+txFreq_Limits(2))/interferenceBW;
    end

end
```

See Also

Objects
`satelliteScenario` | `Satellite` | `Access` | `GroundStation` | `satelliteScenarioViewer` | `ConicalSensor` | `Transmitter` | `Receiver` | `Gimbal`

Functions
`show` | `play` | `hide`

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-17
- “Comparison of Orbit Propagators” on page 1-31
- “Modeling Satellite Constellations Using Ephemeris Data” on page 1-39
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-49

- “Model, Visualize, and Analyze Satellite Scenario”

More About

- “Satellite Scenario Key Concepts”
- “Satellite Scenario Basics”

Signal Transmission

GPS Waveform Generation

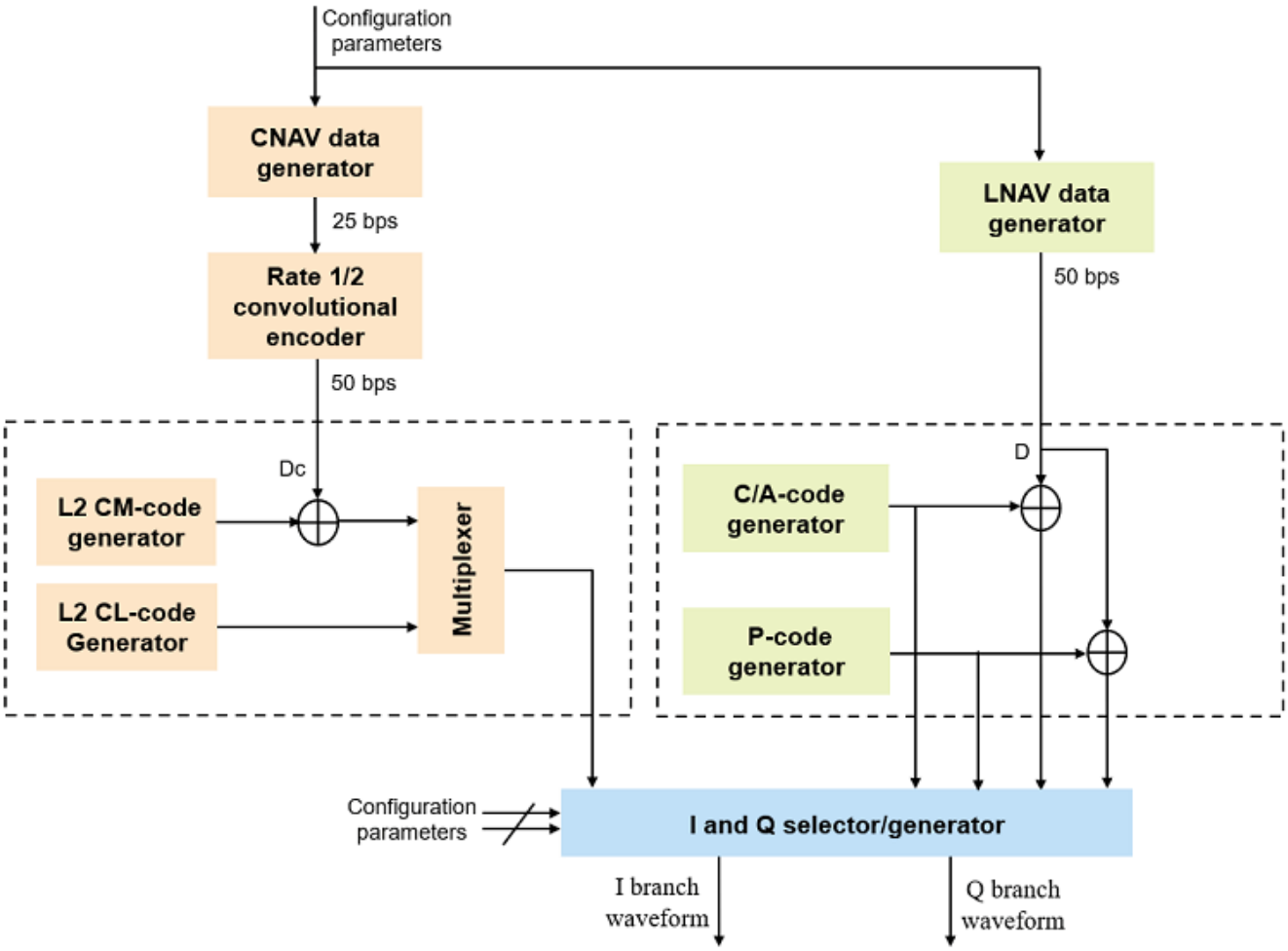
This example shows how to generate Global Positioning System (GPS) legacy navigation (LNAV) data, civil navigation (CNAV) data, and a complex baseband waveform. The spreading of the data is performed with coarse acquisition code (C/A-code), precision code (P-code), or civil moderate / civil long code (L2 CM-/L2 CL-code). This example shows GPS waveform generation according to the IS-GPS-200L standard [1] on page 2-0 . To design a navigation system based on GPS, you must test the receiver with a received signal. Because you cannot control transmitter and channel parameters, a signal that is received from a satellite is not useful for testing a receiver. To test the receiver, you must use a waveform that is generated under a controlled set of parameters.

Introduction

Generate a GPS signal by using these three steps.

1. Generate GPS data bits by using the configuration parameters that are described in later sections. The data bits are generated at a rate of 50 bits per second (bps).
2. Spread these low rate data bits by using high rate spreading codes. The GPS standard [1] on page 2-0 specifies three kinds of spreading codes: C/A-code, P-code, and L2 CM-/L2 CL-code. In addition to these three codes, this standard also specifies Y-code to use instead of P-code when anti-spoofing mode of an operation is active. P-code and Y-code together are called P(Y)-code. The configuration parameters determine the spreading code that is used to generate the waveform.
3. Generate the GPS complex baseband waveform from the bits that are spread by the spreading codes by selecting codes on the in-phase branch and quadrature-phase branch according to the set of configuration parameters.

This figure shows the block diagram of the GPS waveform generator from the configuration parameters.



GPS Signal Structure

The standard [1] on page 2-0 describes the transmission of GPS signals on two frequencies: L1 (1575.42 MHz) and L2 (1227.60 MHz). A signal of base frequency 10.23 MHz generates both of these signals. The signal frequency of L1 is 154 x 10.23 MHz = 1575.42 MHz, and the signal frequency of L2 is 120 x 10.23 MHz = 1227.60 MHz. The code-division-multiple-access technique enables you to differentiate between the satellites even though all GPS satellites transmit on the same frequency.

With the evolution of GPS, the signal structure is expanded to improve the navigation performance. For instance, LNAV data is transmitted on the L1-band from the inception of GPS. From GPS block IIR-M onwards, CNAV data is transmitted on the L2-band in addition to the LNAV data that exists on both L1 and L2 bands. This table shows these signal configurations and GPS evolution. "Dc" represents the CNAV data bits and "D" represents the LNAV data bits. Bit operator, ⊕, represents the XOR operation.

SV Blocks	L1		L2	
	In-phase	Quadrature-phase	In-phase	Quadrature-phase
Block II/IIA/IIR	P(Y)⊕D	C/A⊕D	P(Y)⊕D Or P(Y) Or C/A⊕D	None
Block IIR-M/IIF and GPS III/IIIF	P(Y)⊕D	C/A⊕D	P(Y)⊕D Or P(Y)	L2 CM⊕Dc with L2 CL Or C/A⊕D Or C/A

Using this table to choose the content that needs to be transmitted in the in-phase and quadrature-phase branches.

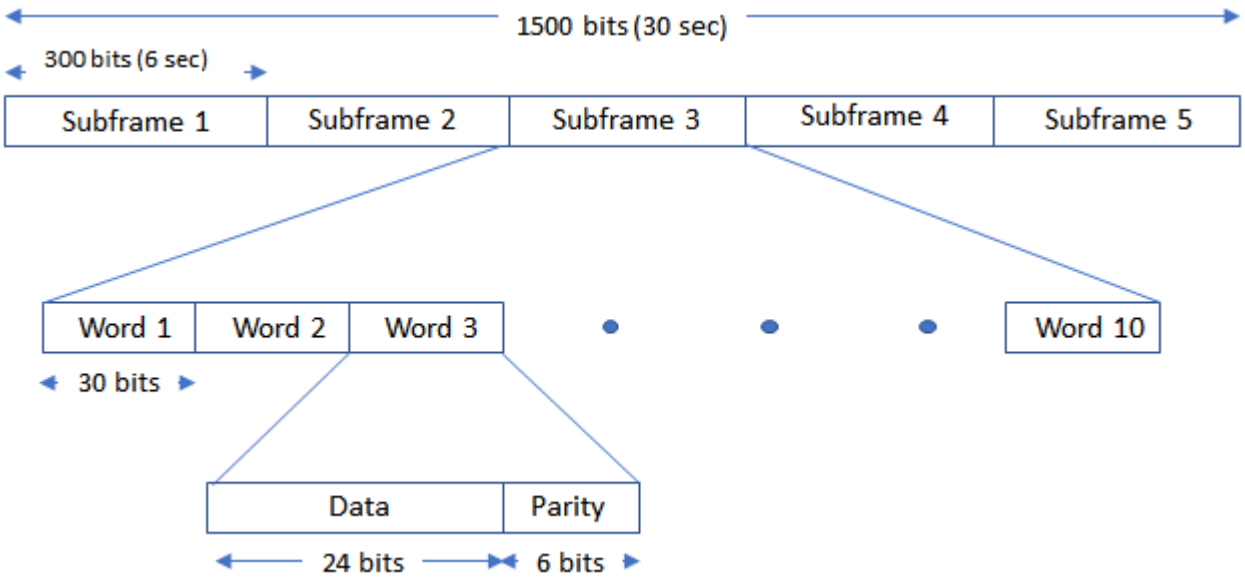
```
IBranchContent = P(Y) + D
```

```
IBranchContent =
"P(Y) + D"
```

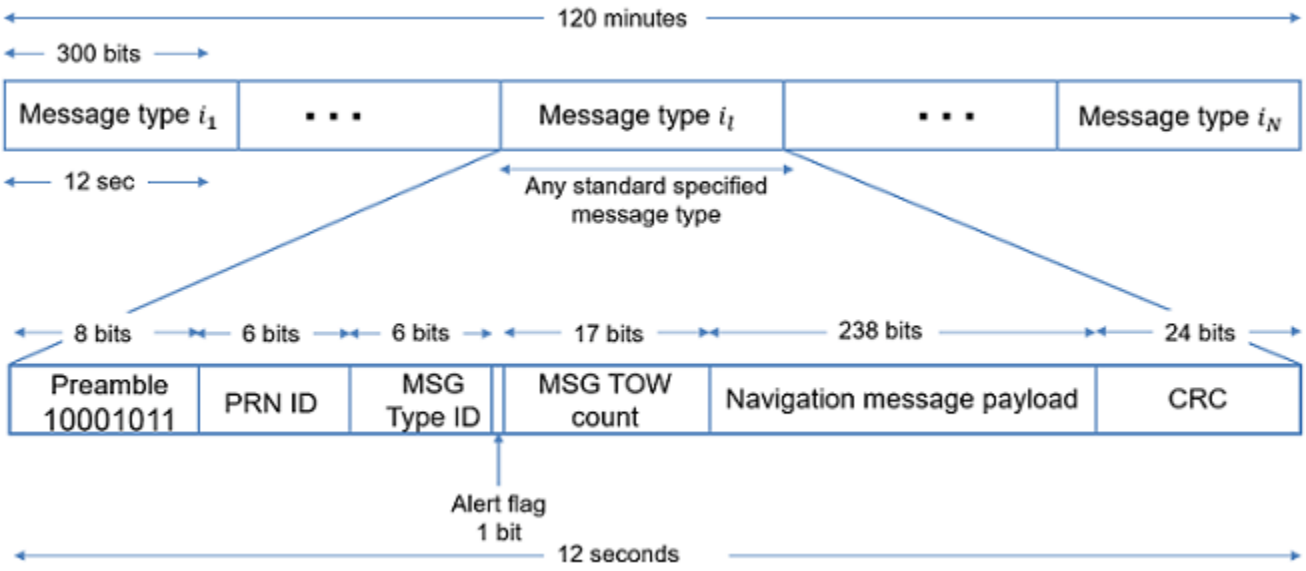
```
QBranchContent = C/A + D
```

```
QBranchContent =
"C/A + D"
```

Many of the transmitted properties in the LNAV and CNAV data are same, but the frame structure is different. The LNAV data is transmitted in 1500 bit-length frames, with each frame consisting of five subframes of 300 bits in each subframe. Because the data rate is 50 bps, transmitting each subframe takes 6 seconds and transmitting each frame takes 30 seconds. Each subframe consists of 10 words with 30 bits (24 data bits and 6 parity bits) in each word. The GPS data contains information regarding the clock and the position of the satellites. This figure shows the frame structure of the LNAV data.



The CNAV data is transmitted continuously in the form of *message types*. Each message type consists of 300 bits that are transmitted at 25 bps. These bits are passed through a rate-half convolutional-encoder to obtain 600 bits from each message type at 50 bps. Transmitting each message type takes 12 seconds. The standard [1] on page 2-0 defines 14 message types in this order: 10, 11, 30, 31, 32, 33, 34, 35, 36, 37, 12, 13, 14, and 15. For a detailed description of each message type and the data transmitted, see IS-GPS-200L Appendix III [1] on page 2-0 . The order in which each message type is transmitted is completely arbitrary but is sequenced to provide optimal user experience. In this example, you can choose the order in which these message types are transmitted. This figure shows the CNAV message structure.



The message type shown in this figure has these fields:

- PRN ID: Pseudo-random noise (PRN) index

- MSG: Message
- TOW: Time of week
- CRC: Cyclic redundancy check

Set the `ShowVisualizations` property to enable the spectrum and correlation plot visualizations. Set the `WriteWaveformToFile` property to write the complex baseband waveform to a file if needed. This example enables visualizations and disables writing the waveform to a file.

```
ShowVisualizations =  ;
```

```
WriteWaveformToFile = ;
```

Specify the satellite PRN index as an integer in the range [1,63].

```
PRNID = 1;
```

Because generating the GPS waveform for the entire navigation data can take a lot of time and memory, this example demonstrates generating a waveform for only one bit of the navigation data. You can control generating the waveform for a specified number of data bits by using the property `NumNavDataBits`.

```
% Set this value to 1 to generate the waveform from the first bit of the
% navigation data
NavDataBitStartIndex = 1321;

% Set this value to control the number of navigation data bits in the
% generated waveform
NumNavDataBits = 1;
```

GPS Data Initialization

Initialize the data configuration object to generate the CNAV data. You can create a configuration object from the `HelperGPSNavigationConfig` object. Update the configuration object properties to customize the waveform as required.

```
cnavConfig = HelperGPSNavigationConfig("DataType","CNAV","PRNID",PRNID)
```

```
cnavConfig =
  HelperGPSNavigationConfig with properties:

                 DataType: "CNAV"
                    PRNID: 1
             MessageTypes: [4x15 double]
                 Preamble: 139
                   HOWTOW: 0
              L2CPhasing: 0
              CEIDataSet: [1x1 HelperGPSCEIConfig]
          AgeOfDataOffset: 0
          AlmanacFileName: "gpsAlmanac.txt"
               Ionosphere: [1x1 struct]
         EarthOrientation: [1x1 struct]
                      UTC: [1x1 struct]
   DifferentialCorrection: [1x1 struct]
               TimeOffset: [1x1 struct]
      TextInMessageType36: 'This content is part of Satellite Communications Toolbox'
      TextInMessageType15: 'This content is part of Satellite Communications Toolbox'
```

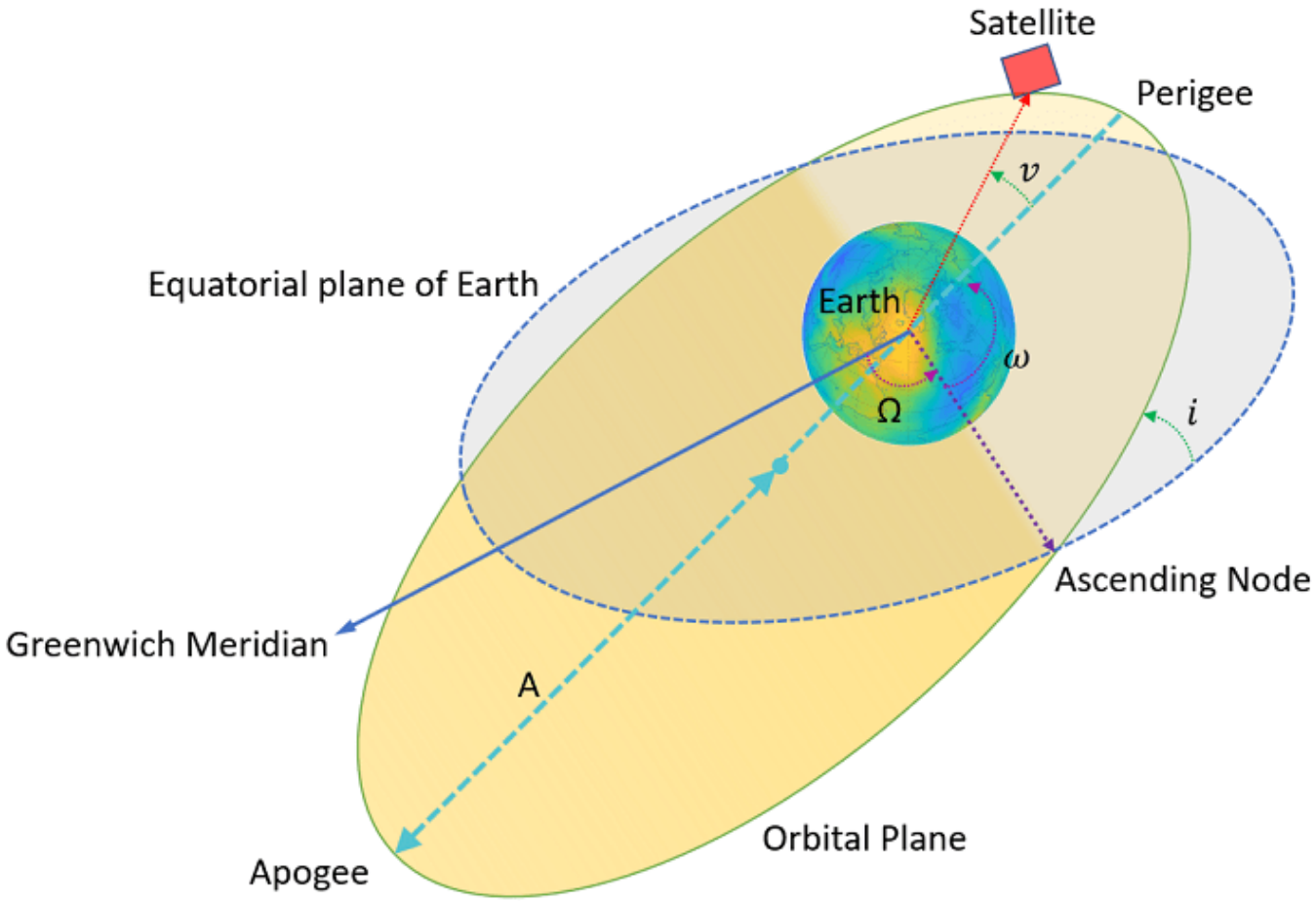
```
Read-only properties:
 No properties.
```

The property, `CEIDataSet`, is a configuration object of type `HelperGPSCEIConfig`, which contains clock, ephemeris, and integrity (CEI) data parameters.

In theory, clocks of all satellites must be synchronized, which implies that all GPS satellite clocks must show the same time at a given moment in time. In practice, deterministic satellite clock error characteristics of bias, drift, and aging as well as satellite implementation characteristics of group delay bias and mean differential group delay exist. These errors deviate the satellite clocks from the GPS system time.

GPS satellites revolve around the Earth in an elliptical orbit, with Earth at one of the focal points of the ellipse. A set of orbital parameters that accurately defines a satellite position in this elliptical orbit is called *ephemeris*. Each GPS satellite transmits its own ephemeris data on subframes 2 and 3 for LNAV data and on message types 10 and 11 for CNAV data. This figure shows five orbital parameters for a satellite in the Earth-centered Earth-fixed (ECEF) coordinate system. This figure does not show an actual GPS satellite, and the figure is for illustration only.

- Length of semimajor axis, A: Distance from the center of the elliptical orbit of the satellite to the apogee or perigee
- Inclination angle, i: Angle between the equatorial plane of Earth and the plane of the orbit of the satellite
- Longitude of ascending node, Ω: Angle between the Greenwich meridian and the direction of the ascending node
- Argument of perigee, ω: Angle between the direction of the ascending node and the direction of the perigee
- True anomaly, υ: Angle between the direction of the perigee and the direction of the current position of the satellite



Per Kepler's second law of planetary motion, the angular velocity (rate of change of true anomaly) is different at different locations in the orbit. You can define the mean anomaly, whose rate of change is constant over the entire orbit of the satellite. In GPS ephemeris parameters, rather than specifying true anomaly, mean anomaly is specified (from which true anomaly can be found). IS-GPS-200L Table 20-IV [1] on page 2-0 specifies the algorithms that relate mean anomaly and true anomaly.

The eccentricity of the ellipse also defines the orbit. Eccentricity gives a measure of deviation of the elliptical orbit from the circular shape.

This command displays the CEI data set properties that are related to the CNAV data.

```
cnavConfig.CEIDataSet
```

```
ans =
  HelperGPSCEIConfig with properties:

                       SignalHealth: [3x1 double]
                         WeekNumber: 2149
              GroupDelayDifferential: 0
        SVClockCorrectionCoefficients: [3x1 double]
                 ReferenceTimeOfClock: 0
                  SemiMajorAxisLength: 26560
             ChangeRateInSemiMajorAxis: 0
                  MeanMotionDifference: 0
```

```
          RateOfMeanMotionDifference: 0
                        Eccentricity: 0.0200
                         MeanAnomaly: 0
            ReferenceTimeOfEphemeris: 0
             HarmonicCorrectionTerms: [6x1 double]
                 IntegrityStatusFlag: 0
                   ArgumentOfPerigee: -0.5200
               RateOfRightAscension: 0
             LongitudeOfAscendingNode: -0.8400
                         Inclination: 0.3000
                    InclinationRate: 0
                             URAEDID: 0
                InterSignalCorrection: [4x1 double]
          ReferenceTimeCEIPropagation: 0
     ReferenceWeekNumberCEIPropagation: 101
                             URANEDID: [3x1 double]
                           AlertFlag: 0

   Read-only properties:
    No properties.
```

A similar set of properties exist for the LNAV data. Create a configuration object to store the LNAV data.

```
lnavConfig = HelperGPSNavigationConfig("DataType","LNAV","PRNID",PRNID)
```

```
lnavConfig =
  HelperGPSNavigationConfig with properties:

                     DataType: "LNAV"
                        PRNID: 1
                  FrameIndices: [1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 ... ]
                     Preamble: 139
                   TLMMessage: 0
                       HOWTOW: 0
                 AntiSpoofFlag: 0
                     CodesOnL2: "P-code"
                   L2PDataFlag: 0
                    CEIDataSet: [1x1 HelperGPSCEIConfig]
              AgeOfDataOffset: 0
     NMCTAvailabilityIndicator: 0
                       NMCTERD: [30x1 double]
             AlmanacFileName: "gpsAlmanac.txt"
                  Ionosphere: [1x1 struct]
                         UTC: [1x1 struct]
                 TextMessage: 'This content is part of Satellite Communications Toolbox'

   Read-only properties:
    No properties.
```

Similar to the CNAV data, the LNAV data has CEI data properties that are different from that of CNAV. This command displays the LNAV CEI data properties.

```
lnavConfig.CEIDataSet
```

```
ans =
  HelperGPSCEIConfig with properties:
```

```
                      SVHealth: 0
                IssueOfDataClock: 0
                           URAID: 0
                      WeekNumber: 2149
          GroupDelayDifferential: 0
    SVClockCorrectionCoefficients: [3x1 double]
            ReferenceTimeOfClock: 0
             SemiMajorAxisLength: 26560
            MeanMotionDifference: 0
                 FitIntervalFlag: 0
                    Eccentricity: 0.0200
                     MeanAnomaly: 0
         ReferenceTimeOfEphemeris: 0
          HarmonicCorrectionTerms: [6x1 double]
             IssueOfDataEphemeris: 0
              IntegrityStatusFlag: 0
                ArgumentOfPerigee: -0.5200
             RateOfRightAscension: 0
         LongitudeOfAscendingNode: -0.8400
                      Inclination: 0.3000
                InclinationRate: 0
                      AlertFlag: 0

   Read-only properties:
    No properties.
```

GPS Signal Generation

To generate a GPS signal at the baseband, follow these steps.

1. Generate the navigation data bits at 50 bits per second.
2. Based on the configuration, generate C/A-code, P-code, L2 CM-/L2 CL-code, or a combination.
3. Spread the CNAV or LNAV data bit with the appropriate ranging codes.
4. Collect data for the in-phase branch and quadrature-phase branch by rate-matching the codes in each branch.
5. Map the bits on both of the branches as bit 0 to +1 and bit 1 to –1.
6. (Optional) Write this baseband waveform to a file (depending on the `WriteWaveformToFile` property value).

Based on the configuration, generate the CNAV data.

```
cnavData = HelperGPSNAVDataEncode(cnavConfig);
```

Pass the CNAV data through the convolutional encoder.

```
% Initialize the trellis for convolutional encoder
trellis = poly2trellis(7,{'1+x+x^2+x^3+x^6','1+x^2+x^3+x^5+x^6'});
cenc = comm.ConvolutionalEncoder('TrellisStructure',trellis, ...
    "TerminationMethod","Continuous");
encodedCNAVData = cenc(cnavData);
```

Based on the configuration, generate the LNAV data.

```
lnavData = HelperGPSNAVDataEncode(lnavConfig);
```

Specify all of the required properties for waveform genertion.

```
CLCodeResetIdx = 75; % CL-code spans over 75 data bits before resetting
numBBSamplesPerDataBit = 204600;
CLCodeIdx = mod(NavDataBitStartIndex-1,CLCodeResetIdx);
IQContent = [IBranchContent,QBranchContent];
pgen = gpsPCode("PRNID",PRNID,"InitialTime", ...
    lnavConfig.CEIDataSet.ReferenceTimeOfEphemeris, ...
    "OutputCodeLength",numBBSamplesPerDataBit);
% Pre-initialize the baseband waveform for speed
gpsBBWaveform = zeros(numBBSamplesPerDataBit*NumNavDataBits,1);
```

Create a file into which the waveform is written.

```
if WriteWaveformToFile == 1
    bbWriter = comm.BasebandFileWriter('Waveform.bb',10.23e6,0);
end
```

Independently process each navigation data bit in a loop.

```
for iDataBit = 1:NumNavDataBits
    dataBitIdx = iDataBit+NavDataBitStartIndex-1;
    bbSamplesIndices = ((iDataBit-1)*numBBSamplesPerDataBit+1): ...
        (iDataBit*numBBSamplesPerDataBit);
    gpsBBWaveform(bbSamplesIndices) = HelperGPSBasebandWaveform(IQContent,pgen,PRNID, ...
        CLCodeIdx,lnavData(dataBitIdx),encodedCNAVData(dataBitIdx));
    CLCodeIdx = mod(CLCodeIdx+1,CLCodeResetIdx);
    if WriteWaveformToFile == 1
        bbWriter(gpsBBWaveform(bbSamplesIndices));
    end
end
```

Close the file if it is opened.

```
if WriteWaveformToFile == 1
    release(bbWriter);
end
```

Signal Visualization

Plot autocorrelation of the C/A-code and visualize the spectrum of the GPS signals.

```
if ShowVisualizations
```

Autocorrelation of the ranging code sequence is near-zero except at zero delay, and crosscorrelation of two different sequences is near-zero. Because the C/A-code is periodic with a period of 1023 bits, autocorrelation has a peak for a delay of every 1023 bits. Calculate and plot the autocorrelation of GPS spreading code.

```
    % Because P-code is 10 times faster than C/A-code or L2 CM-/L2 CL-code,
    % initialise down sample factor to 10
    downsampleFactor = 10;
    IBranchData = real(gpsBBWaveform);
    QBranchData = imag(gpsBBWaveform(1:downsampleFactor:end));
    lags = (-1023:1023).';
    plot(lags,xcorr(real(QBranchData(1:1023)),1023))
    grid on
    xlabel('Number of Samples Delayed')
```

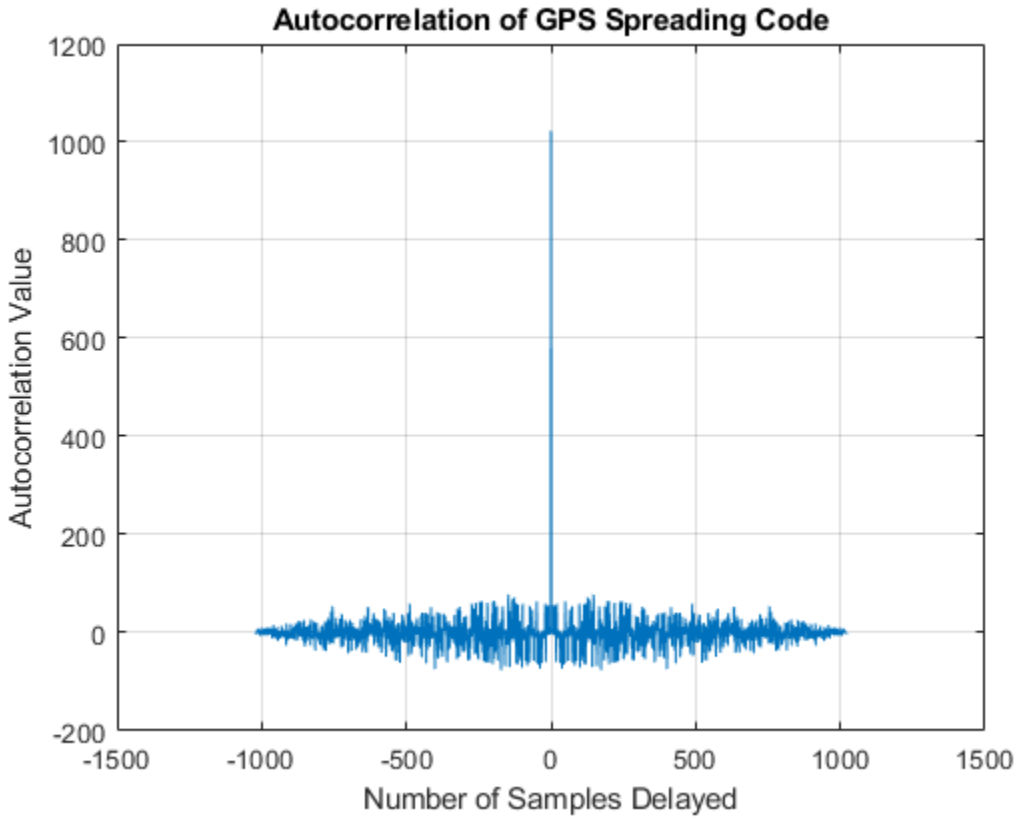
```
    ylabel('Autocorrelation Value')
    title('Autocorrelation of GPS Spreading Code')
```

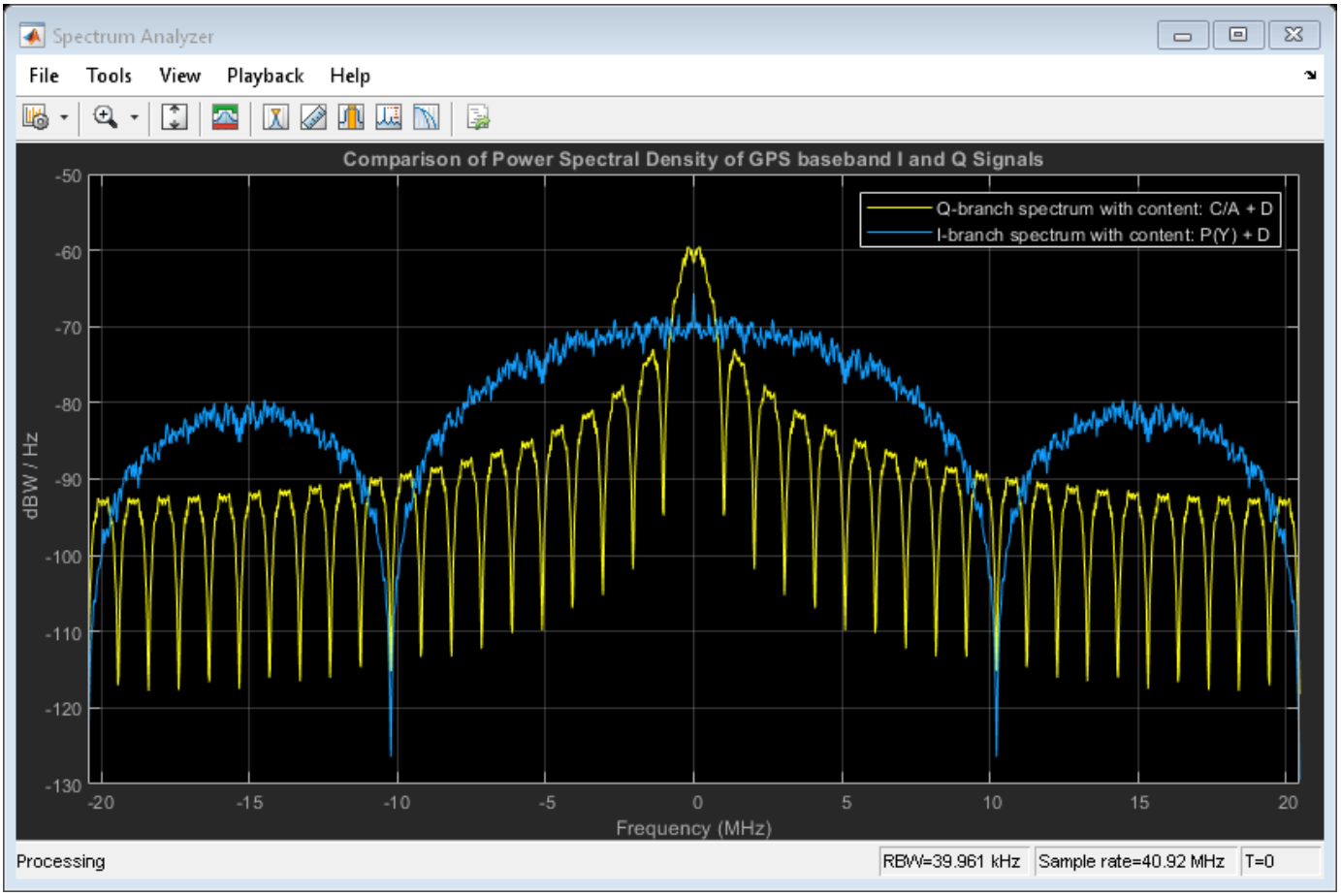
Compare the power spectral density of in-phase branch and quadrature-branch signals. This spectrum plot shows that the P-code is wider when P-code is used.

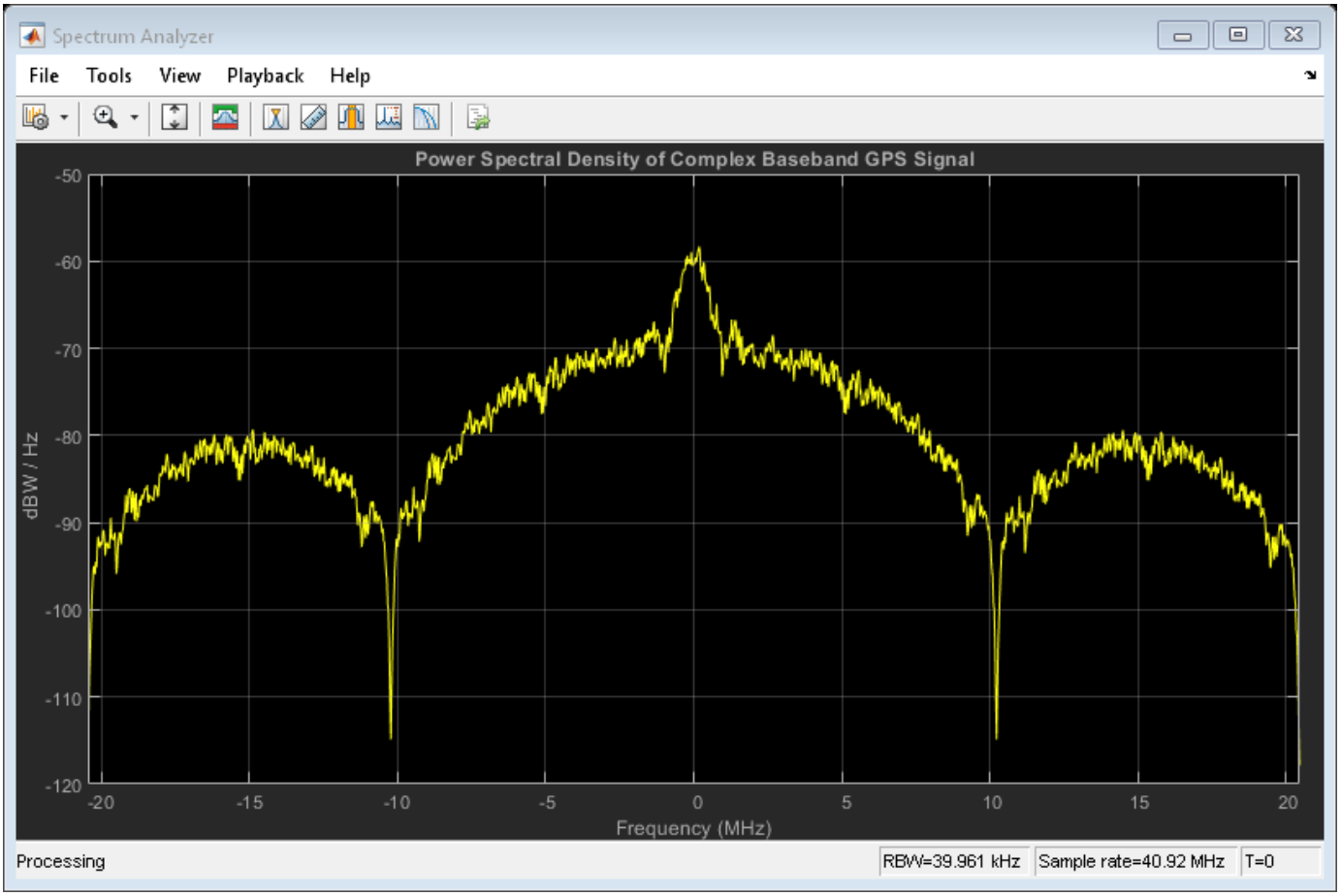
```
    repeatFactor = 40;
    % Repeat the generated BPSK signal of C/A-code to see the adjacent bands spectrum
    QBranchUpsampled = repmat(QBranchData(:).',repeatFactor,1);
    QBranchUpsampled = QBranchUpsampled(:);
    % Repeat the generated BPSK signal of in-phase component to see the
    % adjacent bands spectrum. Repeat the in-phase branch samples ten times less
    % as every sample in quadrature-branch corresponds to 10 samples in in-phase branch
    IBranchUpsampled = repmat(IBranchData(:).',repeatFactor/10,1);
    IBranchUpsampled = real(IBranchUpsampled(:));
    iqScope = dsp.SpectrumAnalyzer('SampleRate',1.023e6*repeatFactor, ...
        'PlotAsTwoSidedSpectrum',true, ...
        'SpectrumType','Power density', ...
        'AveragingMethod','Exponential', ...
        'SpectrumUnits','dBW', ...
        'YLimits',[-130, -50],'Title', ...
        'Comparison of Power Spectral Density of GPS baseband I and Q Signals', ...
        'ShowLegend',true,'ChannelNames', ...
        {['Q-branch spectrum with content: ' char(QBranchContent)], ...
        ['I-branch spectrum with content: ' char(IBranchContent)]});

    iqScope([QBranchUpsampled,IBranchUpsampled]);
```

Plot the signal power spectral density at the baseband. To observe the adjacent band spectrum for the GPS signal, repeat the signal at the baseband.

```
    repeatFactor = 4;
    % Repeat the generated BPSK signal to see the adjacent bands spectrum
    updata = repmat(gpsBBWaveform(:).',repeatFactor,1);
    updata = updata(:);
    bbscope = dsp.SpectrumAnalyzer('SampleRate',10*1.023e6*repeatFactor, ...
        'PlotAsTwoSidedSpectrum',true, ...
        'SpectrumType','Power density', ...
        'AveragingMethod','Exponential', ...
        'SpectrumUnits','dBW', ...
        'YLimits',[-120,-50], ...
        'Title','Power Spectral Density of Complex Baseband GPS Signal');
    bbscope(updata);
end
```


Autocorrelation of GPS Spreading Code
Autocorrelation Value
Number of Samples Delayed
1200
1000
800
600
400
200
0
-200
-1500
-1000
-500
0
500
1000
1500


Spectrum Analyzer
File
Tools
View
Playback
Help
Comparison of Power Spectral Density of GPS baseband I and Q Signals
Q-branch spectrum with content: C/A + D
I-branch spectrum with content: P(Y) + D
dBW / Hz
-50
-60
-70
-80
-90
-100
-110
-120
-130
-20
-15
-10
-5
0
5
10
15
20
Frequency (MHz)
Processing
RBW=39.961 kHz
Sample rate=40.92 MHz
T=0



Further Exploration

This example uses a configuration object to generate GPS data bits and the navigation signal in the baseband. You can replace the property values in this configuration object and observe how the GPS data is generated. You can also change the ephemeris parameters with an existing real data set and pass those parameters into the CEI data set. Additionally, you can specify your own almanac file. If you are using your own almanac file, the week number in the almanac file and the week number in the configuration object must match.

Further, this example shows how to generate a GPS baseband waveform, which can be extended to generate an intermediate frequency (IF) waveform from the baseband waveform by multiplying a cosine signal on the in-phase branch and a sine signal on the quadrature-branch.

Additionally, this example shows how to generate a GPS waveform from one satellite, which can be combined along with multiple satellite PRN codes to get an integrated signal.

Appendix

This example uses these data and helper files:

- gpsAlmanac.txt — Almanac data file downloaded from Navcen website
- HelperGPSAlmanac2Struct.m — Convert text file of almanac to structure

- HelperGPSBasebandWaveform.m — Create GPS baseband waveform from data bits
- HelperGPSCEIConfig.m — Create configuration object for GPS navigation data
- HelperGPSL2CRangingCode.m — Generate L2 CM-/L2 CL-code
- HelperGPSNAVDataEncode.m — Encode navigation data into bits from data that is in configuration object
- HelperGPSNavigationConfig.m — Create configuration object for GPS navigation data

Bibliography

[1] IS-GPS-200, Rev: L. *NAVSTAR GPS Space Segment/Navigation User Segment Interfaces*. May 14, 2020; Code Ident: 66RP1.

3

RF Propagation and Channel Models

Simulate and Visualize a Land Mobile-Satellite Channel

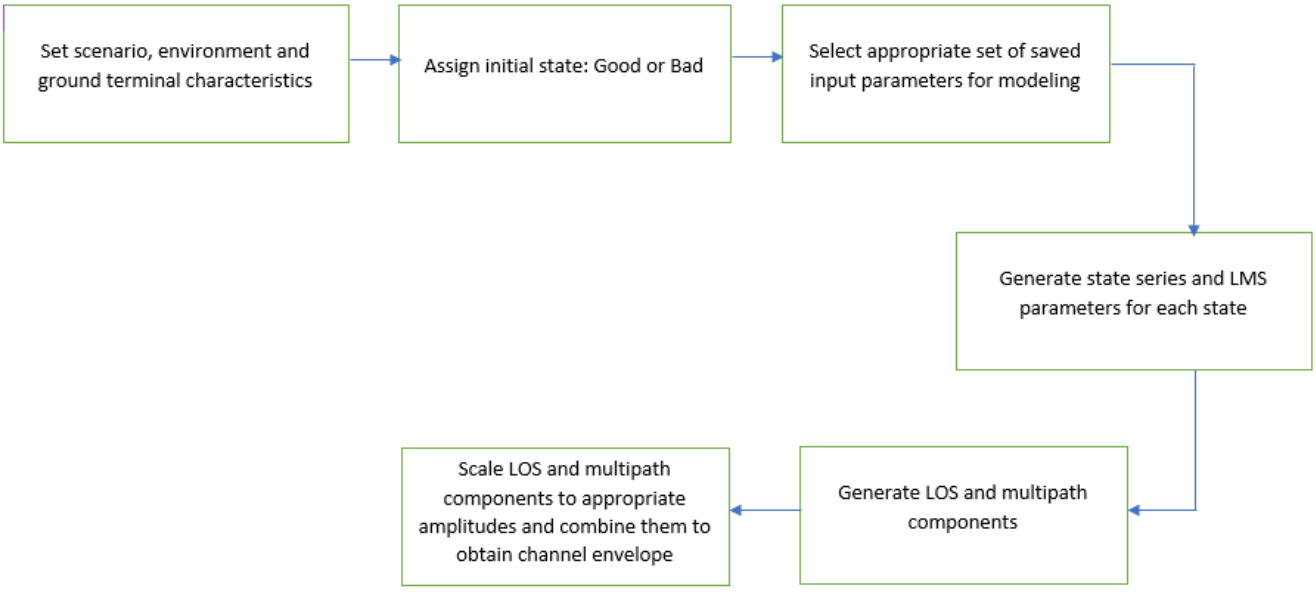
This example shows how to model a two-state land mobile-satellite (LMS) channel model by generating a state series, its respective space series, and the channel coefficients. In a scenario involving a satellite terminal and a mobile terminal, a signal being transmitted through the channel does not always have an ideal line-of-sight path. In some cases, the signal experiences phenomena such as doppler shift, shadowing, and multipath fading. Appropriately modeling the effects of such phenomena is essential to properly design end-to-end communication links that are able to handle and compensate the effects of the channel. The example assumes an urban scenario and uses a carrier frequency of 3.8 GHz. This model is applicable for frequencies in the range 3 to 5 GHz.

Introduction

An LMS channel model aims at simulating the channel envelope that is observed in a satellite-to-ground channel. Given the moving nature of the terminals, the channel envelope experiences variations due to movement of the transmitting and receiving terminals, blockage due to buildings and foliage, shadowing, and multipath.

This example models such a channel by using a two-state semi-Markov chain, where the channel alternates between a good and bad state. A *good state* is characterized by either line-of-sight conditions or partial shadowing conditions, whereas a *bad state* is characterized by either severe shadowing conditions or complete blockage.

The following block diagram shows the step-by-step procedure to model the channel:



In addition to the environment and carrier frequency defined for this example, the modeling of the channel is done by setting up the scenario. This requires defining the following parameters:

- Elevation angle
- Velocity of the ground terminal

- Sampling time of the channel
- Azimuth orientation of the ground terminal
- Initial state of the channel
- Total simulation duration

Environment Setup and Initial State Assignment

Set up the environment between the satellite terminal and the mobile terminal on the ground. Display the properties of the environment.

```
% Carrier frequency in Hertz
cfg.CarrierFrequency = 3.8e9;
% Elevation angle with respect to ground plane in degrees
cfg.ElevationAngle = 45;
% Speed of movement of ground terminal in meters per second
cfg.Velocity = 2;
% Sampling interval in seconds
cfg.SampleTime = 0.0025;
% Direction of movement of ground terminal in degrees
cfg.AzimuthOrientation = 0;
```

Assign a suitable initial state for the model.

```
cfg.InitialState = Good;
% Total duration of channel modeling in seconds
cfg.TotalSimulationTime = 100;
disp(cfg)
```

```
       CarrierFrequency: 3.8000e+09
         ElevationAngle: 45
               Velocity: 2
             SampleTime: 0.0025
     AzimuthOrientation: 0
           InitialState: "Good"
    TotalSimulationTime: 100
```

Obtain the respective LMS parameters using the configuration defined in ITU-R P.681-11 recommendation Section 3.1 Annexure 2 [2] on page 3-0 . The function `HelperGetLMSInputParams` carries out the necessary operation.

```
[paramsGoodState,paramsBadState] = HelperGetLMSInputParams(cfg);
```

Initialize random number generator with seed. Vary the seed to obtain different channel realizations. The default value 73 is an arbitrary value.

```
seed = 73;
rng(seed);
```

Channel Model

Model the LMS channel using the setup defined in the structure `cfg`.

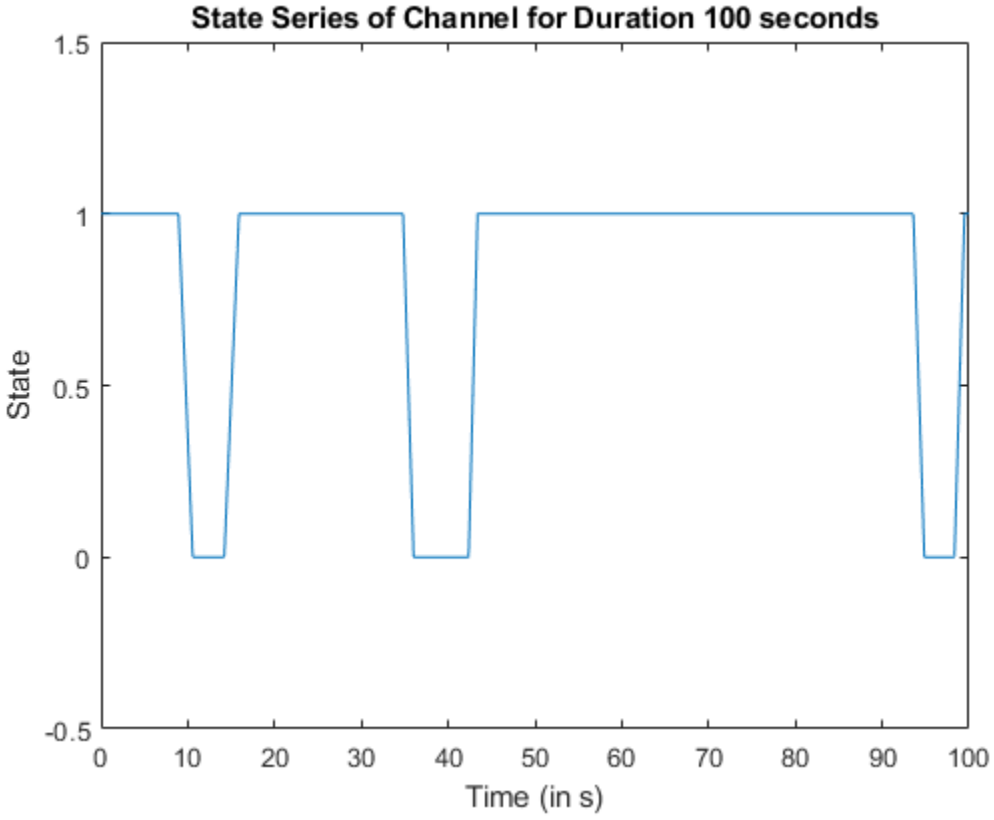
If you have LMS data, you can update the fields of the `paramsGoodState` and `paramsBadState` structures, and then pass the structures to the `HelperModelLMSChannel` helper function.

```
[stateSeries,channelCoefficients] = HelperModelLMSChannel(cfg,paramsGoodState,paramsBadState);
```

Channel Visualization

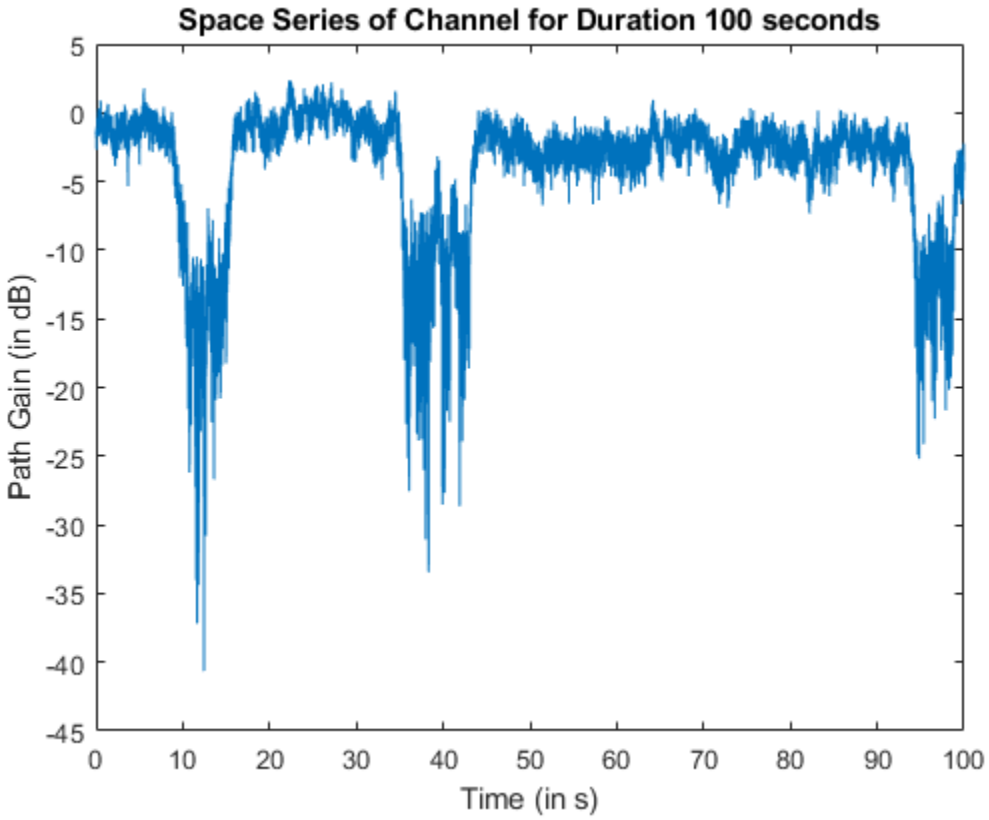
Plot the state series and its respective space series using the channel coefficients generated as a result of modeling.

```
timeVector = 0:cfg.SampleTime:cfg.TotalSimulationTime;
plot(timeVector,stateSeries)
title(['State Series of Channel for Duration ' num2str(cfg.TotalSimulationTime) ' seconds'])
axis([0 timeVector(end) -0.5 1.5])
xlabel('Time (in s)')
ylabel('State')
```



Plot the space series to show how the instantaneous power of the channel envelope varies with time.

```
figure(2)
plot(timeVector,20*log10(abs(channelCoefficients)))
title(['Space Series of Channel for Duration ' num2str(cfg.TotalSimulationTime) ' seconds'])
xlabel('Time (in s)')
ylabel('Path Gain (in dB)')
```



Further Exploration

This example uses three structures: `cfg`, `paramsGoodState`, and `paramsBadState`. The `paramsGoodState` and `paramsBadState` structures contain the LMS parameters that are used to model good and bad states, respectively. The `cfg` structure contains information related to the current setup. You can change each parameter as needed in each of these structures, and then observe how the state series and channel coefficients vary. To model the channel for different frequency bands, you can use any data you have or any of the data tables available in ITU-R P.681-11 recommendation Section 3.1 Annexure 2 [2] on page 3-0 . To filter an input signal through the channel, you can use the output channel coefficients, `channelCoefficients`.

Appendix:

This example uses these helper functions:

- HelperGetLMSInputParams.m: Get LMS parameters for urban scenario
- HelperGenerateLooTriplet.m: Generate the Loo triplet for the current state
- HelperLooTimeSeriesGenerator.m: Generate channel coefficients using Loo time series generator
- HelperModelLMSChannel.m: Model the LMS channel model

References

[1] *3GPP TR 38.811 V15.3.0* (2020-07). Study on New Radio (NR) to support non-terrestrial networks (Release 15). 3rd Generation Partnership Project; Technical Report Group Radio Access Network. https://www.3gpp.org.

[2] *ITU-R Recommendation P.681-11* (12/2019). "Propagation data required for the design systems in the land mobile satellite service." International Telecommunication Union; Radiocommunication Sector. https://www.itu.int/pub/R-REC.

End-to-End Simulation

End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) and number of communications link transmission units (CLTUs) lost in a Consultative Committee for Space Data Systems (CCSDS) telecommand (TC) link. The example adds radio frequency (RF) front-end impairments and additive white gaussian noise (AWGN) to the link.

Introduction

CCSDS TC generally is used for sending commands from a ground station to a spacecraft. CCSDS TC receivers are subjected to large frequency errors due to the frequency uncertainties in spacecraft receivers and the doppler frequency shift. To compensate large frequency offsets, the ground stations perform a carrier sweep in frequency or use an FFT-based acquisition at the spacecraft during satellite acquisition. This example shows how to add a 200 KHz frequency offset to the signal and use an FFT-based acquisition for the correction.

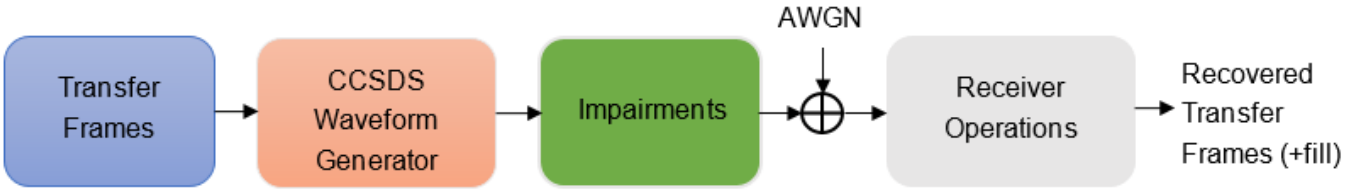
For each signal to noise ratio (SNR) point, CCSDS TC waveforms that are generated with a CLTU and acquisition sequence are distorted by RF impairments and passed through an AWGN channel. The example shows how to model these RF impairments:

- Carrier frequency and phase offset
- Subcarrier frequency and phase offset
- Timing phase offset

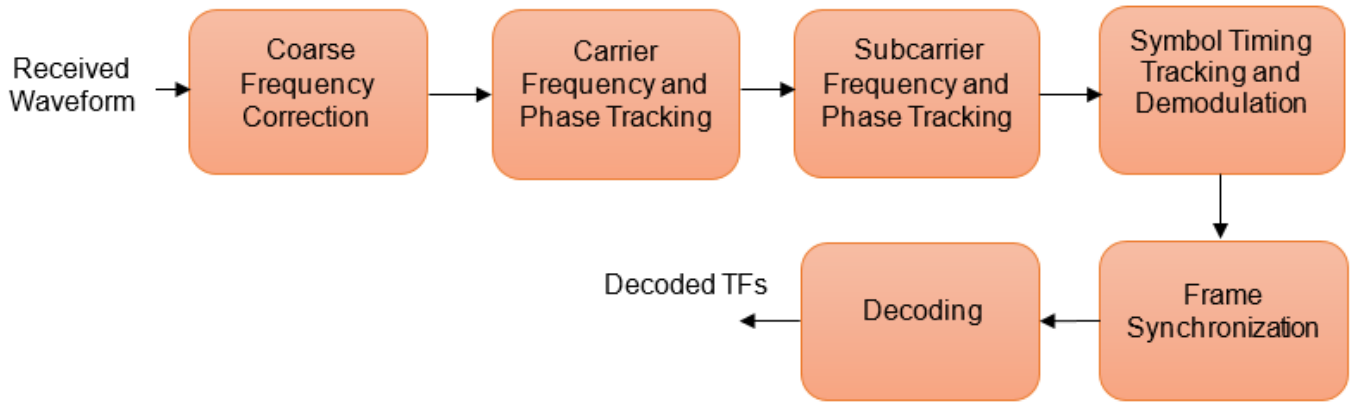
The CCSDS TC receiver compensates for the impairments, and the transfer frames (TFs) in the CLTUs are recovered. This example supports BPSK, PCM/PM/biphase-L, and PCM/PSK/PM modulation schemes. Subcarrier impairments are applicable only with the PCM/PSK/PM modulation scheme. These modulation schemes [8] on page 4-0 are used to generate the CCSDS TC waveform, in the form of baseband in-phase quadrature (IQ) samples.

- 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: 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.

This figure shows the processing steps involved in the recovery of transfer frames.



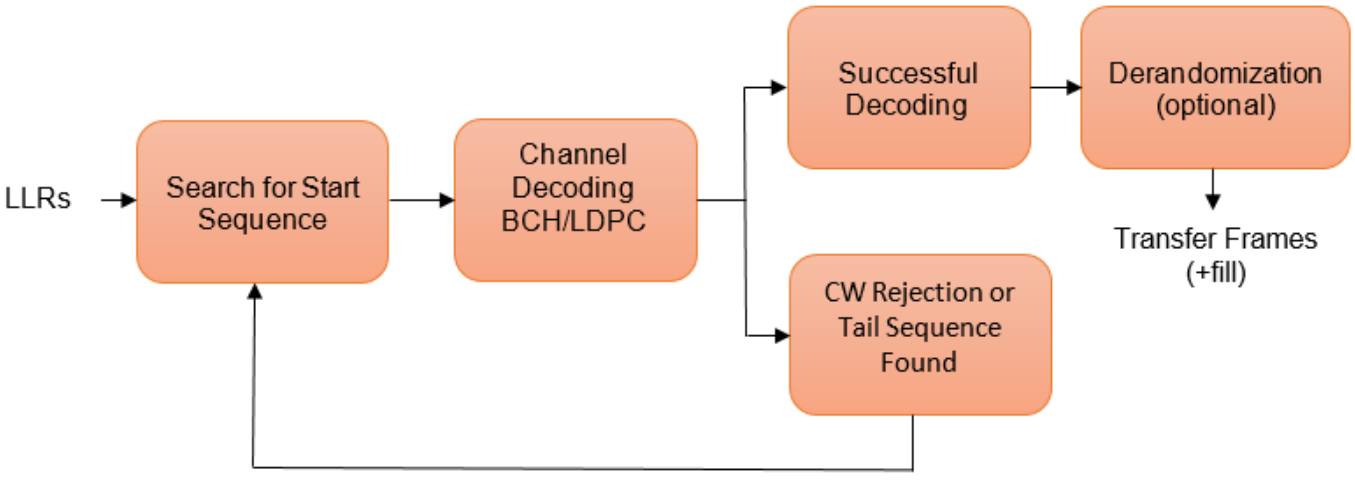
This figure shows the receiver operations, which include RF impairments compensation, demodulation, frame synchronization, and the decoding of transfer frames.



To recover the TFs from the received waveform, follow these steps.

1. Coarse frequency correction: Use the FFT-based algorithm to estimate the frequency offset.
2. Carrier frequency and phase tracking: Use the second order phase locked loop (PLL) [1] on page 4-0 for carrier tracking.
3. Subcarrier frequency and phase tracking: Use the second order Costas loop [1] on page 4-0 for subcarrier tracking.
4. Symbol timing tracking and demodulation: Use the second-order data transition tracking loop (DTTL) [3] on page 4-0 module for timing synchronization and symbol demodulation [1] on page 4-0 .
5. Frame synchronization and decoding: Use a hard symbol based algorithm for Bose Chaudhuri Hocquenghem (BCH) code and soft symbol based algorithm for low density parity check (LDPC) code.

This figure shows the processing steps that are involved in frame synchronization and the decoding of TFs [4] on page 4-0 .



1. Search for start sequence: When the channel coding is BCH, the incoming bit stream is searched bit by bit for the start sequence pattern. When the channel coding is LDPC, the incoming soft

symbols stream is searched with a soft correlator for the start sequence pattern. For BCH, the permissible number of errors in the start sequence is 0 or 1 (depending on the decoding mode). In error detecting mode, the permissible number of errors in the start sequence is 0. In error correcting mode, the permissible number of errors in the start sequence is 1.

2. Decoding: When a start sequence is detected, the decoding operation begins. The codewords (CWs) are decoded and optionally derandomized.
3. CW rejection or tail sequence detection: If the decoder has any decoding failure or any uncorrected errors in the decoded output, data from this failed CW is not transferred to the data link sublayer operations. The CW is rejected, and the search for the start sequence restarts. If a tail sequence is present, search for the tail sequence to detect the end of the CLTU. For BCH decoding, the CW rejection method is employed. For LDPC, use the tail sequence correlation or the CW rejection. When no tail sequence is used, the search for the start sequence must resume at the beginning of the uncorrected CW. When a tail sequence is used, the search can resume at the end of the uncorrected CW.

Simulation Configuration

Configure the number of samples per symbol and symbol rate.

```
% Samples per symbol
% Due to the low symbol rate and 200 KHz frequency offset, a large value of
% 200 samples per symbol must be used as a default value with
% PCM/PSK/PM modulation. For BPSK and PCM/PM/biphase-L modulation, a
% default value of 20 samples per symbol is used (due to medium and high
% symbol rates).
sps = 20;
% Symbol rate
% The symbol rates specified in TC for each modulation are:
% - For PCM/PSK/PM modulation, the coded symbol rates are 4000, 2000, 1000,
%   500, 250, 125, 62.5, 31.25, 15.625, or 7.8125 symbols/s (as specified in
%   CCSDS TC recommendation [6]).
% - For PCM/PM/biphase-L modulation, the coded symbol rates are 8000, 16000,
%   32000, 64000, 128000, or 256000 symbols/s.
% - For BPSK modulation, the coded symbol rates are 1000, 2000, 4000, 8000,
%   16000, 32000, 64000, 128000, 256000, 512000, 1024000, or 2048000
%   symbols/s.
symbolRate = 2048000;
```

The DTTL for the symbol synchronization performs better with an even number of samples per symbol. For an odd number of samples per symbol, the timing error estimate is nonzero at the perfect tracking of timing offset. The nonzero timing error drags the DTTL away from the perfect tracking condition.

Configure and display the CCSDS TC transmission parameters.

```
cfg = ccsdsTCConfig;
cfg.ChannelCoding = "BCH";
cfg.Modulation = "BPSK";
cfg.ModulationIndex = 1.2; % Applicable with PCM/PSK/PM and PCM/PM/biphase-L. Supported range in
if strcmpi(cfg.Modulation,"PCM/PSK/PM")
    cfg.SymbolRate = symbolRate;
end
cfg.SamplesPerSymbol = sps
```

```
cfg =
  ccsdsTCConfig with properties:
```

```
      DataFormat: "CLTU"
    ChannelCoding: "BCH"
    HasRandomizer: 1
       Modulation: "BPSK"

  Read-only properties:
    No properties.
```

Configure the receiver parameters.

```
normLoopBWCarrier = 0.005;      % Normalized loop bandwidth for carrier synchronizer
normLoopBWSubcarrier = 0.00005; % Normalized loop bandwidth for subcarrier synchronizer
normLoopBWSymbol = 0.005;       % Normalized loop bandwidth for symbol synchronizer
```

To reduce noise contribution in the loop, decrease the loop bandwidth. The pull-in range of the frequency offset is also reduced because of decrement in the loop bandwidth. When you use a small loop bandwidth in the synchronization modules, the acquisition takes a longer time to converge. To improve the performance at low SNRs, reduce the loop bandwidth and use a higher value for the acquisition sequence length. If the loops do not track the offsets, consider increasing the loop bandwidth to increase the pull-in range.

Simulation Parameters

This example executes two burst transmissions for a number of energy per symbol to noise power spectral density ratio (Es/No) points. Es/No can be a vector or a scalar. For statistically valid BER results, run the simulation for at least 1000 number of transmissions.

```
numBurst  = 2;                  % Number of burst transmissions
EsNodB = [8 8.5];               % Es/No in dB
SNRIn = EsNodB - 10*log10(sps); % SNR in dB from Es/No
```

Processing Chain

The distorted CCSDS TC waveform with acquisition sequence and a single CLTU is processed at a time. To synchronize the received data and recover the TFs, these processing steps occur.

1. Generate the bits in the TC TF.
2. Generate the TC waveform for the acquisition sequence with alternating ones and zeros.
3. Generate the CCSDS TC waveform for the TFs with random bits.
4. Apply pulse shaping using a square root raised cosine filter (applicable only with BPSK modulation).
5. Apply the subcarrier frequency and phase offset (applicable only with PCM/PSK/PM modulation).
6. Apply the carrier frequency and phase offset.
7. Apply the timing phase offset.
8. Pass the transmitted signal through an AWGN channel.
9. Correct the coarse frequency and phase offset.
10. Filter the received signal.
11. Correct the carrier frequency and phase offset.
12. Correct the subcarrier frequency and phase offset (applicable only with PCM/PSK/PM modulation).

13 Correct the timing offset and symbol demodulation.

14 Detect the start of CLTU and decode the TFs.

```
% Initialization of variables to store BER and number of CLTUs lost
bitsErr = zeros(length(SNRIn),1);
cltuErr = zeros(length(SNRIn),1);

% Square root raised cosine (SRRC) transmit and receive filter objects for BPSK
if strcmpi(cfg.Modulation,"BPSK")
    % SRRC transmit filter object
    txfilter = comm.RaisedCosineTransmitFilter;
    txfilter.RolloffFactor = 0.35;    % Filter rolloff
    txfilter.FilterSpanInSymbols = 6; % Filter span
    txfilter.OutputSamplesPerSymbol = sps;
    % SRRC receive filter object
    rxfilter = comm.RaisedCosineReceiveFilter;
    rxfilter.RolloffFactor = 0.35;    % Filter rolloff
    rxfilter.FilterSpanInSymbols = 6; % Filter span
    rxfilter.DecimationFactor = 1;
    rxfilter.InputSamplesPerSymbol = sps;
end

% Sample rate
if strcmpi(cfg.Modulation,"PCM/PM/biphase-L")
    % In CCSDS TC recommendation [6] section 2.2.7, coded symbol rates are
    % defined prior to biphase-L encoding.
    fs = 2*sps*symbolRate; % Biphase-L encoding has 2 symbols for each bit
else
    fs = sps*symbolRate;
end

for iSNR = 1:length(SNRIn)

    % Set the random number generator to default
    rng default

    % SNR value in the loop
    SNRdB = SNRIn(iSNR);

    % Initialization of error computing parameters
    totNumErrs = 0;
    numErr = 0;
    totNumBits = 0;
    cltuLost = 0;

    for iBurst = 1:numBurst

        % Acquisition sequence with 800 octets
        acqSeqLength = 6400;
        acqBits = repmat([0;1], 0.5*acqSeqLength, 1); % Alternating ones and zeros with zero as s

        % CCSDS TC Waveform for acquisition sequence
        % Maximum subcarrier frequency offset specified in CCSDS TC is
        % ±(2*1e-4)*fsc, where fsc is the subcarrier frequency
        subFreqOffset = 3.2; % Subcarrier frequency offset in Hz
        subPhaseOffset = 4;  % Subcarrier phase offset in degrees
        % Frequency offset in Hz
```

```
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    % Signal modulation along with subcarrier frequency and phase offset
    acqSymb = HelperCCSDSTCSubCarrierModulation(acqBits,cfg,subFreqOffset,subPhaseOffset)
else
    % Signal modulation as per the specified scheme in CCSDS telecommmand
    % Subcarrier impairments are not applicable with BPSK and PCM/PM/biphase-L
    cfg.DataFormat = 'acquisition sequence';
    acqSymb = ccsdsTCWaveform(acqBits,cfg);
    cfg.DataFormat = 'CLTU';
end

% CCSDS TC waveform for CLTU
transferFramesLength = 640;                    % Number of octets in the transfer frame
inBits = randi([0 1],transferFramesLength,1); % Bits in the TC transfer frame
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    % Encoded bits after TC synchronization and channel coding sublayer operations
    [~,encBits] = ccsdsTCWaveform(inBits,cfg);
    % Signal modulation along with subcarrier frequency and phase offset
    waveSymb = HelperCCSDSTCSubCarrierModulation(encBits,cfg,subFreqOffset,subPhaseOffset
else
    waveSymb = ccsdsTCWaveform(inBits,cfg);
end

% CCSDS TC waveform with acquisition sequence and CLTU
waveform = [acqSymb;waveSymb];

% Transmit filtering for BPSK
if strcmpi(cfg.Modulation,'BPSK')
    % Pulse shaping using SRRC filter
    data = [waveform;zeros(txfilter.FilterSpanInSymbols,1)];
    txSig = txfilter(data);
else
    txSig = waveform;
end

% Add carrier frequency and phase offset
freqOffset = 200000;  % Frequency offset in Hz
phaseOffset = 20;     % Phase offset in degrees
if fs <= (2*(freqOffset+cfg.SubcarrierFrequency)) && strcmpi(cfg.Modulation,'PCM/PSK/PM'
    error('Sample rate must be greater than twice the sum of frequency offset and subcar
elseif fs <= (2*freqOffset)
    error('Sample rate must be greater than twice the frequency offset');
end
pfo = comm.PhaseFrequencyOffset('FrequencyOffset',freqOffset, ...
    'PhaseOffset',phaseOffset,'SampleRate',fs);
txSigOffset = pfo(txSig);

% Timing offset as an integer number of samples
timingErr = 5;        % Timing error must be <= 0.4*sps
delayedSig  = [zeros(timingErr,1);txSigOffset];

% Pass the signal through an AWGN channel
rxSig = awgn(complex(delayedSig),SNRdB,'measured',iBurst);

% Coarse carrier frequency synchronization
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
     % Coarse carrier frequency synchronization for PCM/PSK/PM
    coarseSync = HelperCCSDSTCCoarseFrequencyCompensator('FrequencyResolution',100,...
```

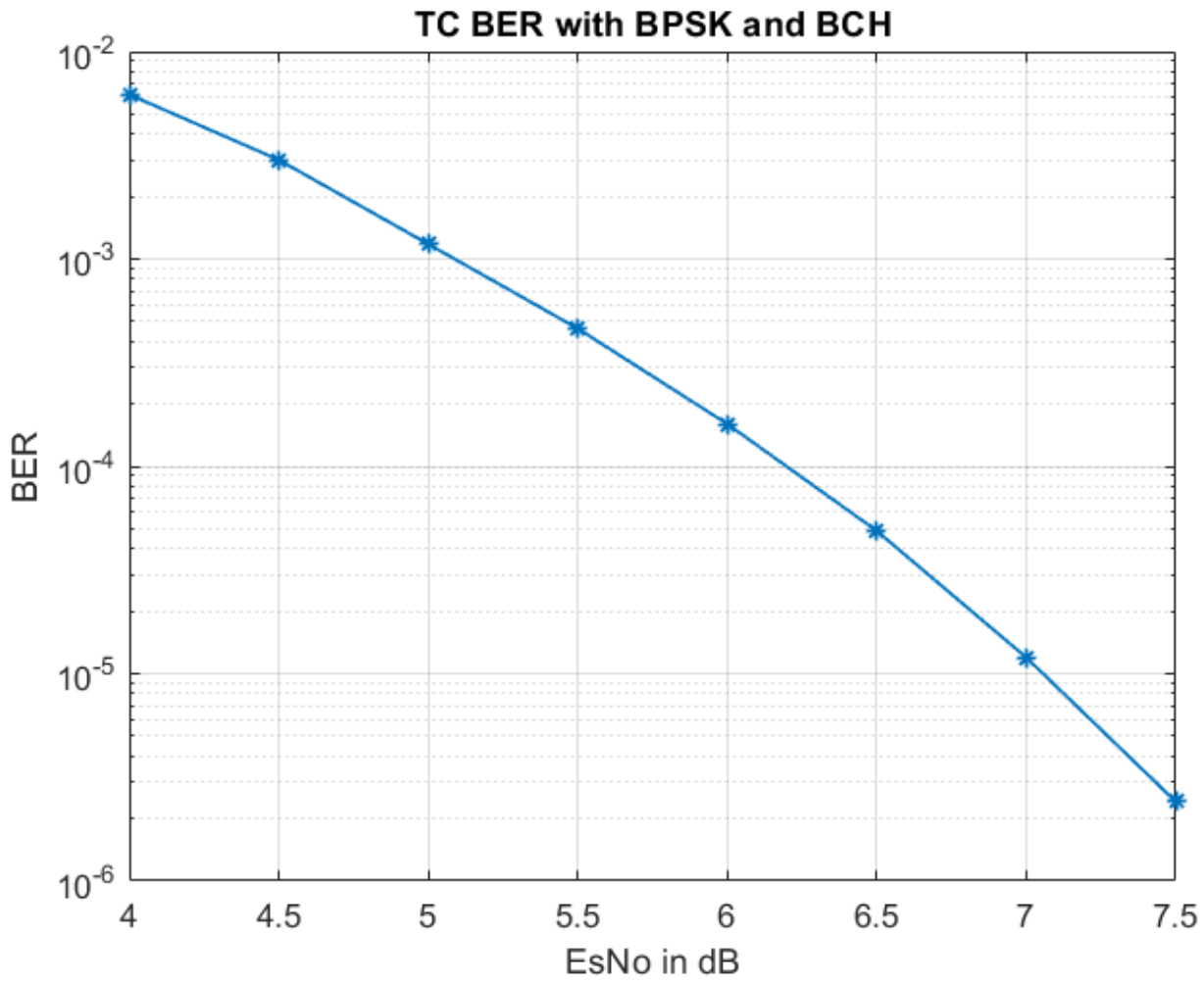
```
        'SampleRate',fs);
else
    % Coarse carrier frequency synchronization for BPSK and PCM/PSK/biphase-L
    coarseSync = comm.CoarseFrequencyCompensator( ...
        'Modulation','BPSK','FrequencyResolution',100, ...
        'SampleRate',fs);
end

% Compensation for coarse frequency offset
[rxCoarse,estCoarseFreqOffset] = coarseSync(rxSig);

% Receive filtering
if strcmpi(cfg.Modulation,'BPSK')
    % SRRC receive filtering for BPSK
    rxFiltDelayed = rxfilter(rxCoarse);
    rxFilt = rxFiltDelayed(rxfilter.FilterSpanInSymbols*sps+1:end);
else
    % Low-pass filtering for PCM/PSK/PM and PCM/PSK/biphase-L
    % Filtering is done with a lowpass filter to reduce the effect of
    % noise to the carrier phase tracking loop
    b = fir1(40,0.3); % Coefficients for 40th-order lowpass filter with cutoff frequency
    rxFiltDelayed = filter(b,1,[rxCoarse;zeros(0.5*(length(b)-1),1)]);
    % Removal of filter delay
    rxFilt = rxFiltDelayed(0.5*(length(b)-1)+1:end);
end

% Fine frequency and phase correction
if strcmpi(cfg.Modulation,'BPSK')
    fineSync = comm.CarrierSynchronizer('SamplesPerSymbol',sps, ...
        'Modulation','BPSK','NormalizedLoopBandwidth',normLoopBWCarrier);
else
    fineSync = HelperCCSDSTCCarrierSynchronizer('SamplesPerSymbol', ...
        cfg.SamplesPerSymbol,'NormalizedLoopBandwidth',normLoopBWCarrier);
end
[rxFine,phErr] = fineSync(rxFilt);

% Subcarrier frequency and phase correction
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    subSync = HelperCCSDSTCSubCarrierSynchronizer('SamplesPerSymbol',sps, ...
        'NormalizedLoopBandwidth',normLoopBWSubcarrier);
    [rxSub,subCarPhErr] = subSync(real(rxFine));
else
    rxSub = real(rxFine);
end

% Timing synchronization and symbol demodulation
timeSync = HelperCCSDSTCSymbolSynchronizer('SamplesPerSymbol',sps, ...
    'NormalizedLoopBandwidth',normLoopBWSymbol);
[rxSym,timingErr] = timeSync(rxSub);

% Search for start sequence and bit recovery
bits = HelperCCSDSTCCLTUBitRecover(rxSym,cfg,'Error Correcting',0.8);
bits = bits(~cellfun('isempty',bits)); % Removal of empty cell array contents

% Length of transfer frames with fill bits
if strcmpi(cfg.ChannelCoding,'BCH')
    messageLength = 56;
else
```

```
            messageLength = 0.5*cfg.LDPCCodewordLength;
        end
        frameLength = messageLength*ceil(length(inBits)/messageLength);

        if (isempty(bits)) || (length(bits{1})~= frameLength) ||(length(bits)>1)
            cltuLost = cltuLost + 1;
        else
            numErr = sum(abs(double(bits{1}(1:length(inBits)))-inBits));
            totNumErrs = totNumErrs + numErr;
            totNumBits = totNumBits + length(inBits);
        end
    end
    bitsErr(iSNR) = totNumErrs/totNumBits;
    cltuErr(iSNR) = cltuLost;

    % Display of bit error rate and number of CLTUs lost
    fprintf([['\nBER with ', num2str(SNRdB+10*log10(sps)) ],' dB Es/No : %1.2e\n'],bitsErr(iSNR))
    fprintf([['\nNumber of CLTUs lost with ', num2str(SNRdB+10*log10(sps)) ],' dB Es/No : %d\n']
end
```

```
BER with 8 dB Es/No : 0.00e+00

Number of CLTUs lost with 8 dB Es/No : 0

BER with 8.5 dB Es/No : 0.00e+00

Number of CLTUs lost with 8.5 dB Es/No : 0
```

BER Results

When each Es/No point is completed, the BER results for the simulation are plotted. `bitsErr` is an array with the measured BER for all simulated Es/No points. The figure shows the simulation result that are obtained with 10,000 number of transmissions and Es/No points in the range [4 7.5].



Further Exploration

Normalized Loop Bandwidth and Acquisition Sequence Length

This example uses a large value for the acquisition sequence length (800 octets) to improve the performance of synchronizers at low SNR values. This table shows the normalized loop bandwidth values and the samples per symbol used in the simulation with each modulation scheme, for an acquisition sequence of 800 octets.

```
T = table({'BPSK';'PCM/PSK/PM';'PCM/PM/biphase-L'},[0.005; 0.0002; 0.0003], ...
          {'Not Applicable';0.00005;'Not Applicable'},[0.005; 0.0005; 0.0005], ...
          [20; 200; 20],[2048000; 4000; 256000],'VariableNames',{'Modulation','Carrier Synchroni
          'Subcarrier Synchronizer','Symbol Synchronizer','Samples per symbol','Symbol rate'})
```

```
T=3×6 table
        Modulation         Carrier Synchronizer    Subcarrier Synchronizer    Symbol Synchronize
    ___________________    ____________________    ______________________    __________________

    {'BPSK'           }           0.005              {'Not Applicable'}             0.005
    {'PCM/PSK/PM'     }          0.0002              {[    5.0000e-05]}            0.0005
```

```
    {'PCM/PM/biphase-L'}          0.0003          {'Not Applicable'}          0.0005
```

You can use this example to further explore these synchronization modules.

- Carrier synchronization: To improve the accuracy of the phase estimate, you can reduce the noise contribution to the tracking loops by decreasing the normalized bandwidth. Reducing the loop bandwidth reduces the pull-in range, and the acquisition takes a longer time to converge.
- Subcarrier synchronization: You can plot the estimated subcarrier offset to identify a more accurate loop bandwidth. To help improve the accuracy of the subcarrier frequency estimate, you can increase the sample rate and SNR.
- Symbol synchronization: The DTTL for the symbol synchronization performs well at higher number of samples per symbol. As you increase the samples per symbol, the resolution increases, and the DTTL performance improves. Too many samples per symbol can reduce the SNR and affect the performance. If the SNR is less than –15 dB (due to a large number of samples per symbol), the performance of the tracking loops is affected.

For any PLL-based loop, to operate at very low SNR, the loop bandwidth must be very low. This low loop bandwidth reduces the pull-in range. For the CLTUs with LDPC channel coding, if the number of CLTUs lost is high, you can reduce the `threshold` value for the detection of the start sequence in the helper function `HelperCCSDSTCCLTUBitRecover`. You can also try improving the BER results by selecting only the CLTUs with a very high normalized correlation metric with the start sequence. To maximize the frame detection and minimize the false alarm, 0.8 is used as a detection threshold in this example. To reduce the false alarm, you can increase the detection threshold value. If you increase the detection threshold value, the frame detection rate reduces.

Subcarrier Frequency Offset with PCM/PSK/PM

The maximum subcarrier frequency offset specified in the CCSDS TC recommendation [6] on page 4-0 is $\pm\left(2\times10^{-4}\right)f_{sc}$, where f_{sc} is the frequency of telecommand subcarrier. You must consider a frequency offset maximum of 3.2 Hz or 1.6 Hz with a 16 KHz or 8 KHz sine wave subcarrier, respectively. You can plot the estimated subcarrier frequency offset to analyze the performance of the subcarrier tracking. When the synchronizer converges, the mean value of the estimate is approximately equal to the input subcarrier frequency offset value of 3.2 Hz.

```
if strcmpi(cfg.Modulation,'PCM/PSK/PM')
    estSubCarFreqOffset = diff(subCarPhErr)*fs/(2*pi);
    rmean = cumsum(estSubCarFreqOffset)./(1:length(estSubCarFreqOffset))';
    plot(rmean)
    xlabel('Symbols')
    ylabel('Estimated Subcarrier Frequency Offset (Hz)')
    title('PCM/PSK/PM: Subcarrier Frequency Offset')
    grid on
end
```

Appendix

The example uses these helper functions:

- HelperCCSDSTCCoarseFrequencyCompensator: Perform coarse carrier frequency synchronization
- HelperCCSDSTCCarrierSynchronizer: Perform fine carrier synchronization
- HelperCCSDSTCSubCarrierSynchronizer: Perform subcarrier synchronization

- HelperCCSDSTCSymbolSynchronizer: Perform symbol timing synchronization and demodulation
- HelperCCSDSTCSubCarrierModulation: Perform subcarrier modulation with frequency and phase offset
- HelperCCSDSTCCLTUBitRecover: Search for start sequence and bit recovery

Bibliography

1. J. Vilà-Valls, M. Navarro, P. Closas and M. Bertinelli, "Synchronization challenges in deep space communications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 34, no. 1, pp. 16-27, Jan. 2019.
2. M. Baldi et al., "State-of-the-art space mission telecommand receivers," *IEEE Aerospace and Electronic Systems Magazine*, vol. 32, no. 6, pp. 4-15, June 2017.
3. S. Million and S. Hinedi, "Effects of symbol transition density on the performance of the data transition tracking loop at low signal-to-noise ratios," *Proceedings IEEE International Conference on Communications ICC '95*, Seattle, WA, USA, 1995, pp. 1036-1040 vol.2.
4. TC Synchronization and Channel Coding. *Recommendation for Space Data System Standards*, CCSDS 231.0-B-3. Blue Book. Issue 3. Washington, D.C.:CCSDS, September 2017.
5. TC Synchronization and Channel Coding. *Summary of Concept and Rationale*. CCSDS 230.1-G-2. Green Book. Issue 2. Washington, D.C.: CCSDS, November 2012.
6. Radio Frequency and Modulation Systems - Part 1. *Earth Stations and Spacecraft*. CCSDS 401.0-B-29. Blue Book. Issue 29. Washington, D.C.: CCSDS, March 2019.
7. Michael Rice, *Digital Communications - A Discrete-Time Approach*. New York: Prentice Hall, 2008.
8. 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.

See Also

Objects
`ccsdsTCConfig` | `ccsdsTMWaveformGenerator`

Functions
`ccsdsTCWaveform`

Related Examples

- “End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections” on page 4-13
- “End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections” on page 4-22

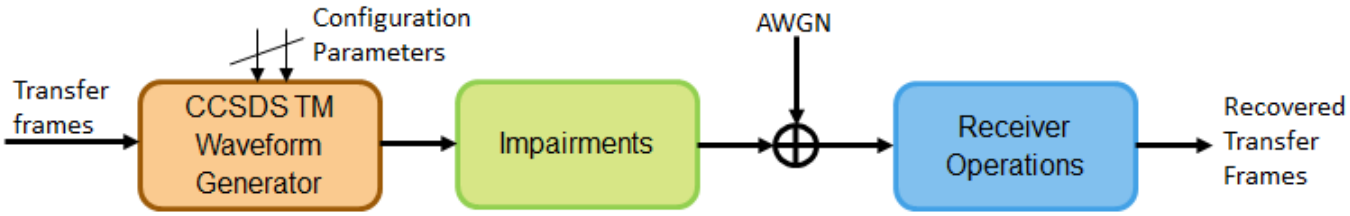
End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) of the end-to-end chain of the Consultative Committee for Space Data Systems (CCSDS) telemetry (TM) system. The simulation chain follows the coding and modulation schemes that are specified by these two standards:

- TM Synchronization and Channel Coding - CCSDS 131.0-B-3 for channel coding schemes [1] on page 4-0
- Radio Frequency and Modulation Systems - part 1 Earth Stations and Spacecraft - CCSDS 401.0-B-30 for modulation schemes [2] on page 4-0

Introduction

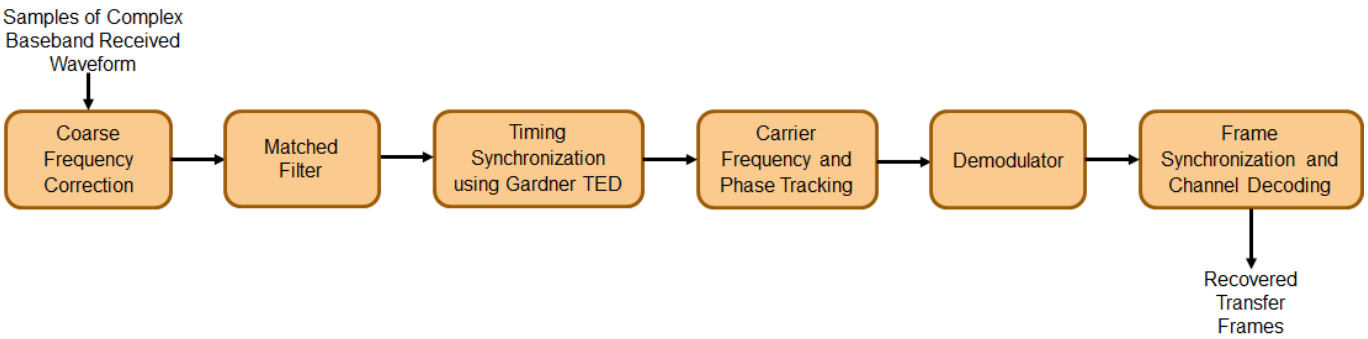
Data from various instruments is generated on board the satellite. This data collectively is called *TM data*. CCSDS specifies the coding and modulation schemes for the transmission of TM data from the satellite to an Earth station. This example shows the end-to-end simulation of the satellite to the Earth station communication link. The example shows how to generate a complex baseband CCSDS TM waveform from the randomly generated transfer frames (TFs), introduce radio frequency (RF) impairments to the baseband signal, and add additive white Gaussian noise (AWGN) to the impaired signal. Then, the example shows the synchronization, demodulation, and decoding of this impaired noisy signal to get the final bits in the form of TFs. The example also shows how to measure the BER with respect to the signal-to-noise ratio (SNR) for one configuration of the CCSDS TM signal. This figure shows the end-to-end simulation chain.



This example models these RF impairments:

- Carrier frequency offset (CFO)
- Carrier phase offset (CPO)
- Symbol timing offset (STO)

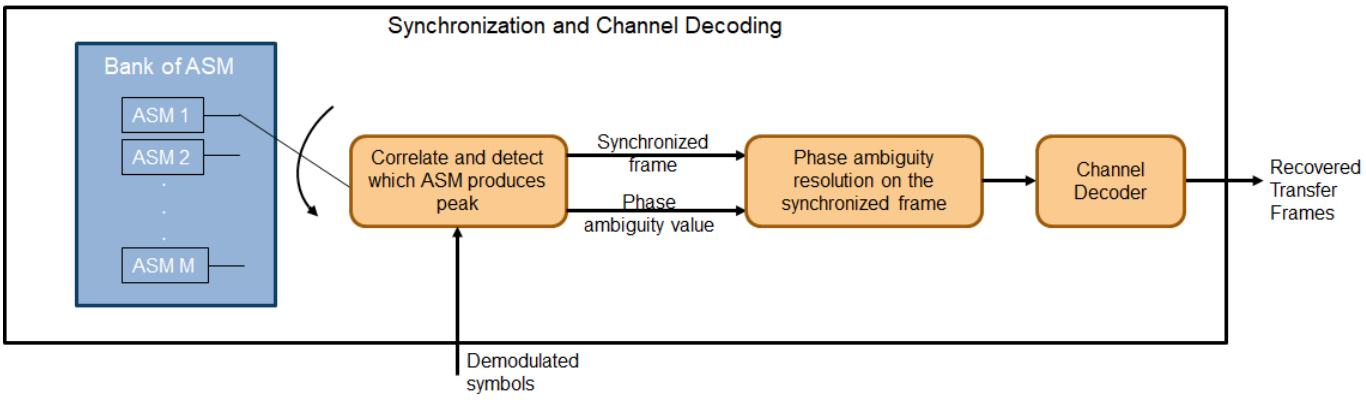
This figure shows the receiver side operations.



The frame synchronization and channel decoding processing step performs these three tasks.

1. Perform phase ambiguity resolution
2. Correctly synchronize the frame to the starting of the attached sync marker (ASM)
3. Perform channel decoding of the synchronized frame to get the recovered TF

This figure shows these three tasks. To start, form a bank of ASM sequences. Each sequence corresponds to the original ASM value in which phase ambiguity is introduced. Correlate each of these sequences with the demodulated symbols. Choose the phase ambiguity value that has the highest correlation peak. Perform the frame synchronization with this correlation process. The process to perform correlation is illustrated in section 9.3.7 in [3] on page 4-0 . This example adopts the simplified Massey algorithm for frame synchronization. Resolve the phase ambiguity on the complete set of demodulated symbols after the frame synchronization process is complete. Finally, perform channel decoding on these symbols to obtain the recovered TFs.



Simulation Parameters

This example uses a quadrature phase shift keying (QPSK) modulation scheme for signal generation and reception and a rate-1/2 convolutional coding scheme for channel coding. The end-to-end chain this example shows can also be used for the channel coding schemes that are specified in [1] on page 4-0 : Reed-Solomon (RS) codes and concatenated codes. For convolutional and concatenated codes, this example supports rates of 1/2 and 2/3 along with pulse code modulation (PCM)-format of non-return to zero-line (NRZ-L). The supported modulation schemes in this example are binary phase shift keying (BPSK) and QPSK.

```
seeConstellation = true;          % Flag to toggle the visualization of constellation
channelCoding = "convolutional"; % Channel coding scheme
transferFrameLength = 1115;       % In bytes corresponding to 223*5
modScheme = "QPSK";               % Modulation scheme
alpha = 0.35;                     % Root raised cosine filter roll-off factor
sps = 8;                          % Samples per symbol
```

Set the frequencies that are used for signal generation and the RF impairment values.

```
fSym = 2e6; % Symbol rate or Baud rate in Hz
cfo = 2e5;  % In Hz
```

Initialize the energy per bit to noise power ratio (Eb/N0), which is used to calculate the SNR using system parameters.

```
EbN0 = 10; % To see a proper BER result, run the simulation for 3.2:0.2:5
```

Initialize the parameters to terminate the simulation. The parameters are set to small values in this example to get quick results. Increase these parameters value to get a smoother BER curve.

```
maxNumErrors = 1e2;  % Simulation stops after maxNumErrors bit errors
maxNumBits = 1e5;    % Simulation stops after processing maxNumBits
                     % Set maxNumBits = 1e8 for a smoother BER curve
maxFramesLost = 1e2; % Simulation stops after maxFramesLost frames are lost
```

System Parameters

Initialize all of the objects that are required for the proper functioning of the end-to-end chain.

Create a CCSDS TM waveform generator with these parameters by using the `ccsdsTMWaveformGenerator` System object™. Display the properties of the object.

```
tmWaveGen = ccsdsTMWaveformGenerator("ChannelCoding",channelCoding, ...
    "NumBytesInTransferFrame",transferFrameLength, ...
    "Modulation",modScheme, ...
    "RolloffFactor",alpha, ...
    "SamplesPerSymbol",sps);
disp(tmWaveGen)
```

```
  ccsdsTMWaveformGenerator with properties:

            WaveformSource: "synchronization and channel coding"
    NumBytesInTransferFrame: 1115
              HasRandomizer: true
                     HasASM: true
                  PCMFormat: "NRZ-L"

   Channel coding
              ChannelCoding: "convolutional"
      ConvolutionalCodeRate: "1/2"

   Digital modulation and filter
                 Modulation: "QPSK"
          PulseShapingFilter: "root raised cosine"
               RolloffFactor: 0.3500
         FilterSpanInSymbols: 10
            SamplesPerSymbol: 8

  Use get to show all properties
```

Calculate the SNR from the Eb/N0 and initialize the parameters related to the calculation of BER.

```
rate = tmWaveGen.info.ActualCodeRate;
M = tmWaveGen.info.NumBitsPerSymbol;
numBitsInTF = tmWaveGen.NumInputBits;
snr = EbN0 + 10*log10(rate) + ...
    10*log10(M) - 10*log10(sps);      % As signal power is scaled to one while introducing noise
                                      % snr value should be reduced by a factor of SPS
numSNR = length(snr);
ber = zeros(numSNR,1);                % Initialize the BER parameter
bercalc = comm.ErrorRate;
```

Create a receive filter object by using the `comm.RaisedCosineReceiveFilter` System object.

```
b = rcosdesign(alpha,tmWaveGen.FilterSpanInSymbols,sps);
% |H(f)| = 1  for |f| < fN(1-alpha) - Annex 1 in Section 2.4.17A in [2]
```

```
Gain =  sum(b);
rxFilterDecimationFactor = sps/2;
rxfilter = comm.RaisedCosineReceiveFilter( ...
    "DecimationFactor",rxFilterDecimationFactor, ...
    "InputSamplesPerSymbol",sps, ...
    "RolloffFactor",alpha, ...
    "Gain",Gain);
```

Model frequency and phase offsets by using the `comm.PhaseFrequencyOffset` System object. Compensate for frequency and phase offset at the receiver in two steps.

1. Compensate for the coarse frequency offset by using the `comm.CoarseFrequencyCompensator` System object.
2. Compensate for the fine frequency offset and the phase offset by using the `comm.CarrierSynchronizer` System object.

```
phaseOffset = pi/8;
fqyoffsetobj = comm.PhaseFrequencyOffset( ...
    "FrequencyOffset",cfo, ...
    "PhaseOffset",phaseOffset, ...
    "SampleRate",sps*fSym);
coarseFreqSync = comm.CoarseFrequencyCompensator( ...
    "Modulation",modScheme, ...
    "FrequencyResolution",100, ...
    "SampleRate",sps*fSym);
fineFreqSync = comm.CarrierSynchronizer("DampingFactor",1/sqrt(2), ...
    "NormalizedLoopBandwidth",0.0007, ...
    "SamplesPerSymbol",1, ...
    "Modulation",modScheme);
```

Create a variable fractional delay object by using the dsp.VariableFractionalDelay System object, which introduces the fractional delay in the transmitted waveform. Create a symbol synchronization object by using the `comm.SymbolSynchronizer` System object, which performs symbol timing synchronization.

```
varDelay = dsp.VariableFractionalDelay("InterpolationMethod","Farrow");
fixedDelayVal = 10.2;
Kp = 1/(pi*(1-((alpha^2)/4)))*cos(pi*alpha/2);
symsyncobj = comm.SymbolSynchronizer( ...
    "DampingFactor",1/sqrt(2), ...
    "DetectorGain",Kp, ...
    "TimingErrorDetector","Gardner (non-data-aided)", ...
    "Modulation","PAM/PSK/QAM", ...
    "NormalizedLoopBandwidth",0.0001, ...
    "SamplesPerSymbol",sps/rxFilterDecimationFactor);
```

Demodulate and decode the received signal by using the `HelperCCSDSTMDemodulator` and `HelperCCSDSTMDecoder` helper files, respectively. Display the properties of the resulting objects.

```
demodobj = HelperCCSDSTMDemodulator("Modulation",modScheme,"ChannelCoding",channelCoding)
```

```
demodobj =
  HelperCCSDSTMDemodulator with properties:

       Modulation: "QPSK"
        PCMFormat: "NRZ-L"
    ChannelCoding: "convolutional"
```

```
decoderobj = HelperCCSDSTMDecoder("ChannelCoding",channelCoding, ...
    "NumBytesInTransferFrame",transferFrameLength, ...
    "Modulation",modScheme)

decoderobj =
  HelperCCSDSTMDecoder with properties:

                     ChannelCoding: "convolutional"
                     HasRandomizer: true
                           HasASM: true
        DisableFrameSynchronization: 0
    DisablePhaseAmbiguityResolution: 0
            NumBytesInTransferFrame: 1115
              ConvolutionalCodeRate: "1/2"
              ViterbiTraceBackDepth: 60
                     ViterbiTrellis: [1x1 struct]
                 ViterbiWordLength: 8
                        Modulation: "QPSK"
                         PCMFormat: "NRZ-L"
```

Initialize the constellation diagram object by using the `comm.ConstellationDiagram` System object to visualize how the constellation evolves as the synchronizers converge.

```
costellationobj = comm.ConstellationDiagram; % Default view is for QPSK
if strcmp(modScheme,'BPSK')
    costellationobj.ReferenceConstellation = [1, -1]
end
```

Processing Chain

To simulate the end-to-end chain and measure the BER of the CCSDS TM system, follow these steps.

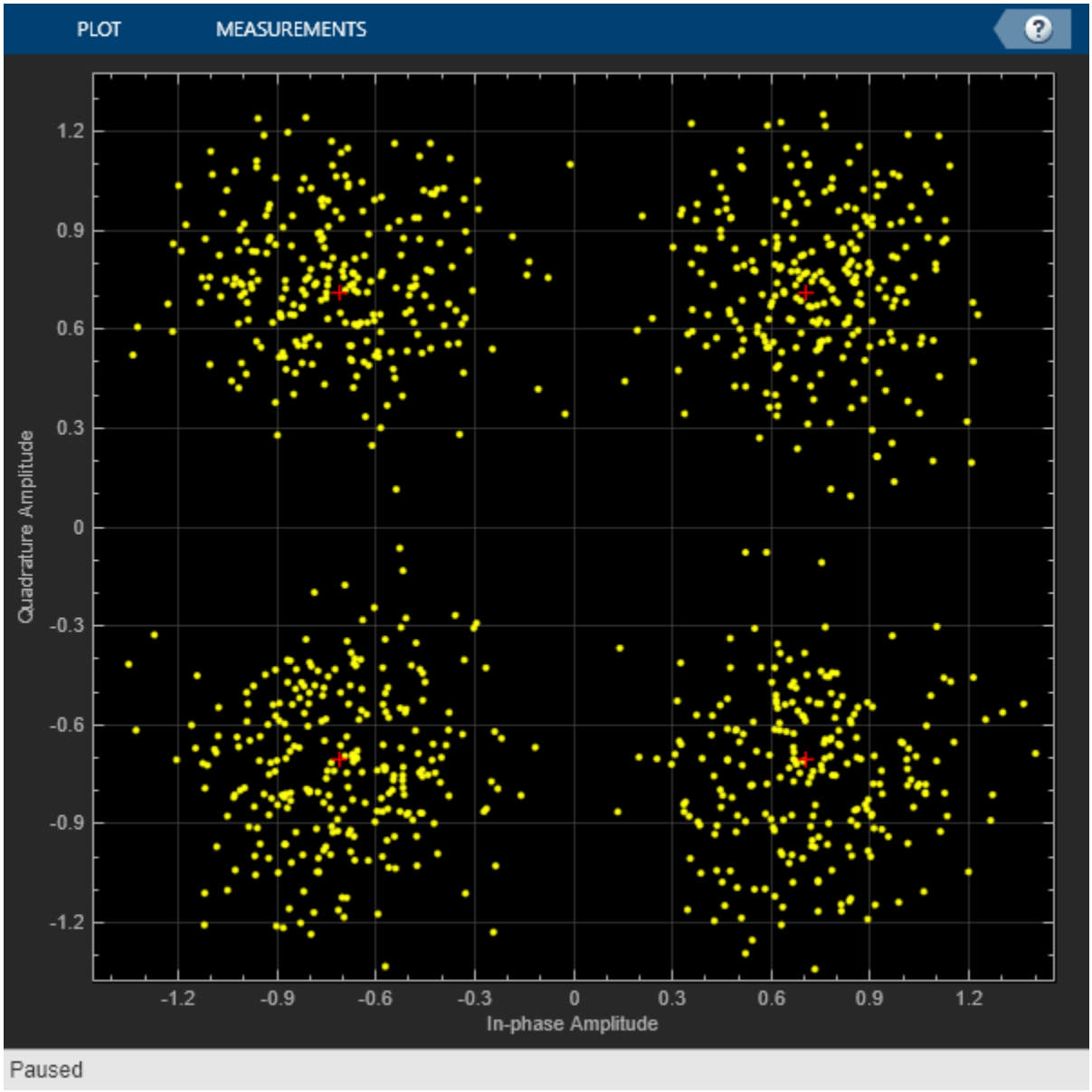
1. Generate random bits to form a TF.
2. Generate the TM waveform by passing the TF through the `ccsdsTMWaveformGenerator` System object.
3. Introduce RF impairments, such as CFO and symbol delay.
4. Add AWGN to the RF-impaired signal. This noisy signal is considered the received signal.
5. Pass the received signal through coarse frequency correction, which performs the initial coarse carrier frequency synchronization. Coarse frequency estimation is done using the "FFT-based" algorithm.
6. Use a matched filter (root raised cosine filter) with the same configuration that is applied at the transmitter end. Because the symbol timing synchronization module works at a sampling rate that is higher than the symbol rate, the complex baseband samples are not down sampled to the symbol rate after filtering. It is down sampled such that at least 2 samples per symbol exist.
7. Perform symbol timing synchronization by using the Gardner timing error detector (TED) to remove the timing offset that is present in the signal.
8. Perform carrier frequency and phase tracking by using the `comm.CarrierSynchronizer` System object, which has a type 2 phase locked loop (PLL). This System object can track a stationary carrier frequency offset. The System object also introduces phase ambiguity, which is then removed by the frame synchronization module.
9. Visualize the constellation after symbol timing and carrier frequency synchronization is complete. Observe how the constellation evolves over multiple iterations.

10 Demodulate the received signal and verify that the signal is at the symbol rate (that is, the samples per symbol is 1).

11 Perform frame synchronization and channel decoding to resolve the phase ambiguity, synchronize the frame to the start of the ASM, and then decode the synchronized frame to recover the TF.

```
numBitsForBER = 8; % For detecting which frame is synchronized
numMessagesInBlock = 2^numBitsForBER;
for isnr = 1:numSNR
    rng default;                          % Reset to get repeatable results
    reset(bercalc);
    berinfo = bercalc(int8(1), int8(1)); % Initialize berinfo before BER is calculated
    tfidx = 1;
    numFramesLost = 0;
    prevdectfidx = 0;
    inputBuffer = zeros(numBitsInTF, 256,"int8");
    while((berinfo(2) < maxNumErrors) && ...
            (berinfo(3) < maxNumBits) && ...
            (numFramesLost < maxFramesLost))
        seed = randi([0 2^32-1],1,1);    % Generate seed for repeatable simulation

        % Transmitter side processing
        bits = int8(randi([0 1],numBitsInTF-numBitsForBER,1));
        % The first 8 bits correspond to the TF index modulo 256. When
        % synchronization modules are included, there can be a few frames
        % where synchronization is lost temporarily and then locks again.
        % In such cases, to calculate the BER, these 8 bits aid in
        % identifying which TF is decoded. If an error in these 8 bits
        % exists, then this error is detected by looking at the difference
        % between consecutive decoded bits. If an error is detected, then
        % that frame is considered lost. Even though the data link layer is
        % out of scope of this example, the data link layer has a similar
        % mechanism. In this example, only for calculating the BER, this
        % mechanism is adopted. The mechanism that is adopted in this
        % example is not as specified in the data link layer of the CCSDS
        % standard. And this mechanism is not specified in the physical
        % layer of the CCSDS standard.
        msg = [de2bi(mod(tfidx-1,numMessagesInBlock),numBitsForBER,"left-msb").';bits];
        inputBuffer(:,mod(tfidx-1,numMessagesInBlock)+1) = msg;
        tx = tmWaveGen(msg);

        % Introduce RF impairments
        cfoInroduced = fqyoffsetobj(tx);                % Introduce CFO
        delayed = varDelay(cfoInroduced,fixedDelayVal); % Introduce timing offset
        rx = awgn(delayed, snr(isnr),'measured',seed);  % Add AWGN

        % Receiver-side processing
        coarseSynced = coarseFreqSync(rx);     % Apply coarse frequency synchronization
        filtered = rxfilter(coarseSynced);     % Filter received samples through RRC filter
        TimeSynced = symsyncobj(filtered);     % Apply symbol timing synchronization
        fineSynced = fineFreqSync(TimeSynced); % Track frequency and phase

        % Visualize constellation
        if seeConstellation
            % Plot constellation of first 1000 symbols in a TF so
            % that variable size of fineSynced does not impede the
            % requirement of constant input size for the
            % comm.ConstellationDiagram System object.
```

```
            costellationobj(fineSynced(1:1000));
        end

        demodData = demodobj(fineSynced); % Demodulate
        decoded = decoderobj(demodData);  % Perform phase ambiguity resolution,
                                          % frame synchronization, and channel decoding

        % Calculate BER and adjust all buffers accordingly
        dectfidx = bi2de(double(decoded(1:8).'), ...
            "left-msb")+1;                % See the value of first 8 bits
        if tfidx > 30 % Consider to calculate BER only after 30 TFs are processed
            % As the value of first 8 bits is increased by one in each
            % iteration, if the difference between the current decoded
            % decimal value of first 8 bits is not equal to the previously
            % decoded one, then it indicates a frame loss.
            if dectfidx - prevdectfidx ~= 1
                numFramesLost = numFramesLost + 1;
                disp(['Frame lost at tfidx: ' num2str(tfidx) ...
                    '. Total frames lost: ' num2str(numFramesLost)]);
            else
                berinfo = bercalc(inputBuffer(:,dectfidx),decoded);
                if nnz(inputBuffer(:,dectfidx)-decoded)
                    disp(['Errors occurred at tfidx: ' num2str(tfidx) ...
                        '. Num errors: ' num2str(nnz(inputBuffer(:,dectfidx) - decoded))])
                end
            end
        end
        prevdectfidx = dectfidx;

        % Update tfidx
        tfidx = tfidx + 1;
    end
    fprintf("\n");
    currentBer = berinfo(1);
    ber(isnr) = currentBer;
    disp(['Eb/N0: ' num2str(EbN0(isnr)) '. BER: ' num2str(currentBer) ...
        '. Num frames lost: ' num2str(numFramesLost)]);

    % Reset objects
    reset(tmWaveGen);
    reset(fqyoffsetobj);
    reset(varDelay);
    reset(coarseFreqSync);
    reset(rxfilter);
    reset(symsyncobj);
    reset(fineFreqSync);
    reset(demodobj);
    reset(decoderobj);
end
```




```
Eb/N0: 10. BER: 0. Num frames lost: 0
```

Further Exploration

This example demonstrates BER simulation for convolutional codes with QPSK modulation in the presence of several RF impairments. To observe the end-to-end simulation chain for different scenarios, change the properties related to the channel coding and modulation schemes. The modulation schemes that are supported by the receiver in this example are BPSK and QPSK. The channel coding schemes that are supported by the receiver in this example are none (that is, no channel coding), RS, convolutional, and concatenated codes.

Run a full BER simulation by setting the Eb/N0 value to 3.2:0.2:5 and observe the BER by setting `maxNumBits` to 1e8. Uncomment this code to plot the BER results.

```
% semilogy(EbN0,ber);
% grid on;
% xlabel('E_b/N_0 (dB)');
% ylabel('BER');
% title('BER plot');
```

Always reserve the initial few TFs for the symbol timing and carrier frequency synchronizers to lock. This example discards the first 30 TFs. This number can vary based on the SNR at which the receiver is operating and the parameters of the synchronization loops, such as loop bandwidth and damping factor. If you operate the receiver at low SNR and observe large errors in the initial values of `tfidx`, then the synchronizers are not yet locked. For the given simulation parameters, discard the initial TFs as appropriate. The second output arguments of `comm.CoarseFrequencyCompensator` and `comm.CarrierSynchronizer` System objects contain the information related to the estimated CFO, which can be used to assess whether the synchronization loops are locked or not.

Appendix

The example uses the following helper files:

- HelperCCSDSTMDemodulator.m - Performs the demodulation of signals that are specified in CCSDS TM [2] on page 4-0
- HelperCCSDSTMDecoder.m - Performs, phase ambiguity resolution, frame synchronization and channel decoding of the codes specified in [1] on page 4-0

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.

[2] Radio Frequency and Modulation Systems--Part 1: Earth Stations and Spacecraft. Recommendation for Space Data System Standards, CCSDS 401.0-B-30. Blue Book. Issue 30. Washington, D.C.: CCSDS, February 2020.

[3] TM Synchronization and Channel Coding - Summary of Concept and Rationale. Report Concerning Space Data System Standards, CCSDS 130.1-G-3. Green Book. Issue 3. Washington, D.C.: CCSDS, June 2020.

See Also

Objects

`ccsdsTMWaveformGenerator` | `ccsdsTCConfig`

Related Examples

- “End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections” on page 4-2
- “End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections” on page 4-22

End-to-End CCSDS Flexible Advanced Coding and Modulation Simulation with RF Impairments and Corrections

This example shows how to measure the bit error rate (BER) of the end-to-end chain of the Consultative Committee for Space Data Systems (CCSDS) flexible advanced coding and modulation (FACM) scheme for high rate telemetry (TM) applications system [1] on page 4-0 .

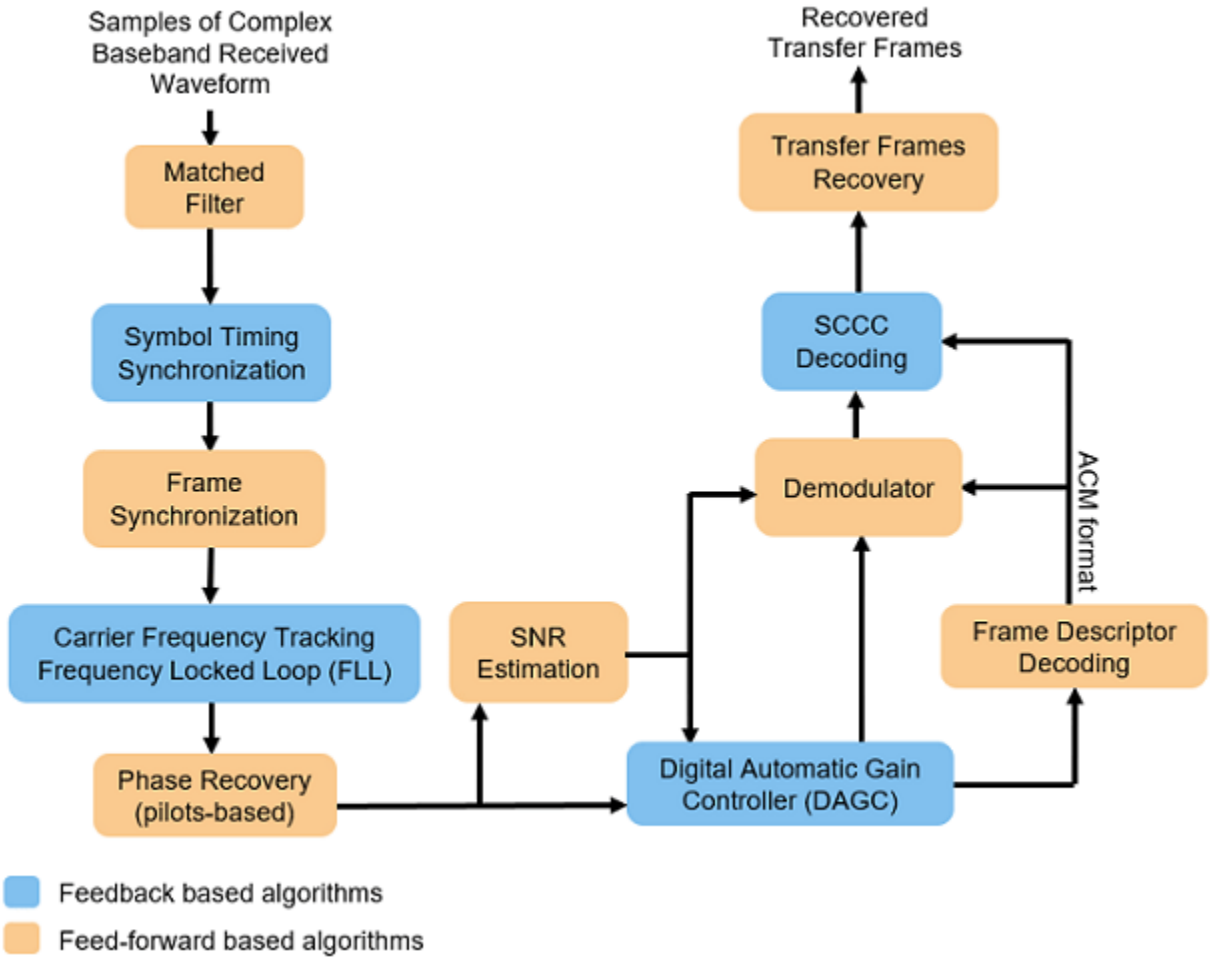
Introduction

Data from various instruments is generated in the satellite. This data collectively is called *TM data*. Earth exploration satellite service (EESS) missions carry payloads that produce substantial TM data rates, starting from a few hundreds of Mbps. To achieve high spectral efficiency for such missions, coding and modulation schemes must be adjusted based on the link budget. The CCSDS FACM scheme for high rate TM applications [1] on page 4-0 standard supports a high data rate by adopting the serial concatenated convolutional codes (SCCC) and modulation schemes from the family of phase shift keying (PSK) and amplitude phase shift keying (APSK). This example shows how to generate a complex baseband CCSDS FACM waveform from the randomly generated transfer frames (TFs), introduce radio frequency (RF) impairments to the baseband signal, and add additive white Gaussian noise (AWGN) to the impaired signal. Then, the example shows the symbol timing recovery, carrier frequency synchronization, demodulation, and decoding of this impaired noisy signal to get the final bits in the form of TFs. This example also shows how to measure the BER with respect to the signal-to-noise ratio (SNR) for one configuration of the CCSDS FACM signal.

This example models these RF impairments on the baseband signal:

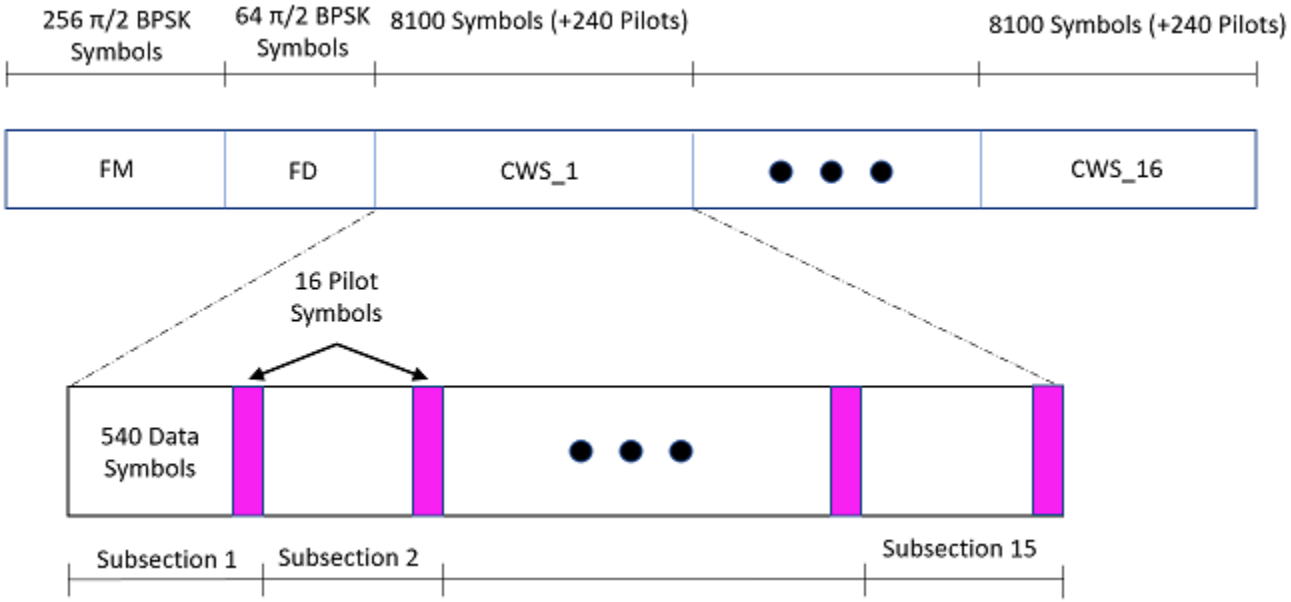
- Carrier frequency offset (CFO)
- Doppler rate
- Sampling rate offset (SRO)
- Phase noise

This figure shows the receiver process, which recovers the symbol timing, carrier frequency, and carrier phase in the presence of the RF impairments.



In the receiver chain, frequency locked loop (FLL) uses the frame marker (FM) that is available in the physical layer (PL) header. The phase recovery module uses the pilot fields to recover the phase. Because the phase recovery algorithm is pilots-based, you must enable the pilot fields for the example to work. The phase recovery module can tolerate some amount of residual CFO that is left after the FLL operation.

This figure shows the PL frame structure of the FACM waveform. In one PL frame, 16 code word sections (CWS) exist and each codeword section is formed from each SCCC encoder output. Use the `ccsdsTMWaveformGenerator` System object to generate the FACM waveform.



Configuration and Simulation Parameters

This example shows various visualizations such as constellation plots and the spectrum of the signal. You can optionally disable these visualizations. For this example, enable them.

```
showVisualizations = ☑ ;
```

Set the configuration parameters that control the waveform properties.

```
sps = 2;                                        % Samples per symbol (SPS)
rolloff = 0.35;                                 % Filter roll-off factor
cfgCCSDSFACM.NumBytesInTransferFrame = 223;
cfgCCSDSFACM.SamplesPerSymbol = sps;
cfgCCSDSFACM.RolloffFactor = rolloff;
cfgCCSDSFACM.FilterSpanInSymbols = 10;
cfgCCSDSFACM.ScramblingCodeNumber = 1;
cfgCCSDSFACM.ACMFormat = 1;
cfgCCSDSFACM.HasPilots = true;                  % HasPilots must be set to true
                                                % for this example to work
```

Specify the simulation parameters in `simParams` structure. Specify the SNR in dB as energy per symbol to noise power ratio (Es/No) in the `EsNodB` field.

```
simParams.EsNodB = 1;
```

Specify the CFO and SRO. Model CFO using the `comm.PhaseFrequencyOffset` System object. Model the sampling rate offset is using the `comm.SampleRateOffset` System object.

```
simParams.CFO = 1e6; % In Hz
simParams.SRO = 20;  % In PPM
```

Specify the attenuation factor for the signal. In the case of no attenuation, specify this value as `1`.

```
simParams.AttenuationFactor = 0.1; % Must be less than or equal to 1
```

Specify the number of PL frames to be generated by specifying the number of frames for initial synchronization that are not accounted in the BER calculation and the number of frames that are accounted in the BER calculations. In this example, set `simParams.NumFramesForBER` to 10 to complete the simulation early. To see a proper BER value, set this value to 200.

```
% Initialize the number of frames used for synchronization
simParams.InitalSyncFrames = 15;
% Initialize the number of frames that are used for calculation of BER
simParams.NumFramesForBER = 10;
simParams.NumPLFrames = simParams.InitalSyncFrames + simParams.NumFramesForBER;
```

Specify the symbol rate and samples per symbol (SPS).

```
simParams.SymbolRate = 100e6; % 100 MBaud
simParams.SPS = sps;
```

`Calculate Latency and Doppler in a Satellite Scenario` example shows how the Doppler shift changes with time based on the satellite orbit. The Doppler shift follows a sinusoidal curve with the peak Doppler shift occurring when the satellite starts to become visible to a receiver on the ground, or when the satellite is receding. Such a cyclic change in Doppler shift is modeled as a sinusoidal Doppler profile. The current example simulates Doppler frequency that changes as given in this equation from [2] on page 4-0 .

$$f(t) = f_D \cos\left(\frac{f_R}{f_D} t\right)$$

f_D is the peak Doppler shift, and f_R is the Doppler rate. Specify these properties of the Doppler profile.

```
simParams.PeakDoppler = 1e6;  % In Hz
simParams.DopplerRate = 50e3; % In Hz/Sec
```

If needed, disable RF impairments for debugging.

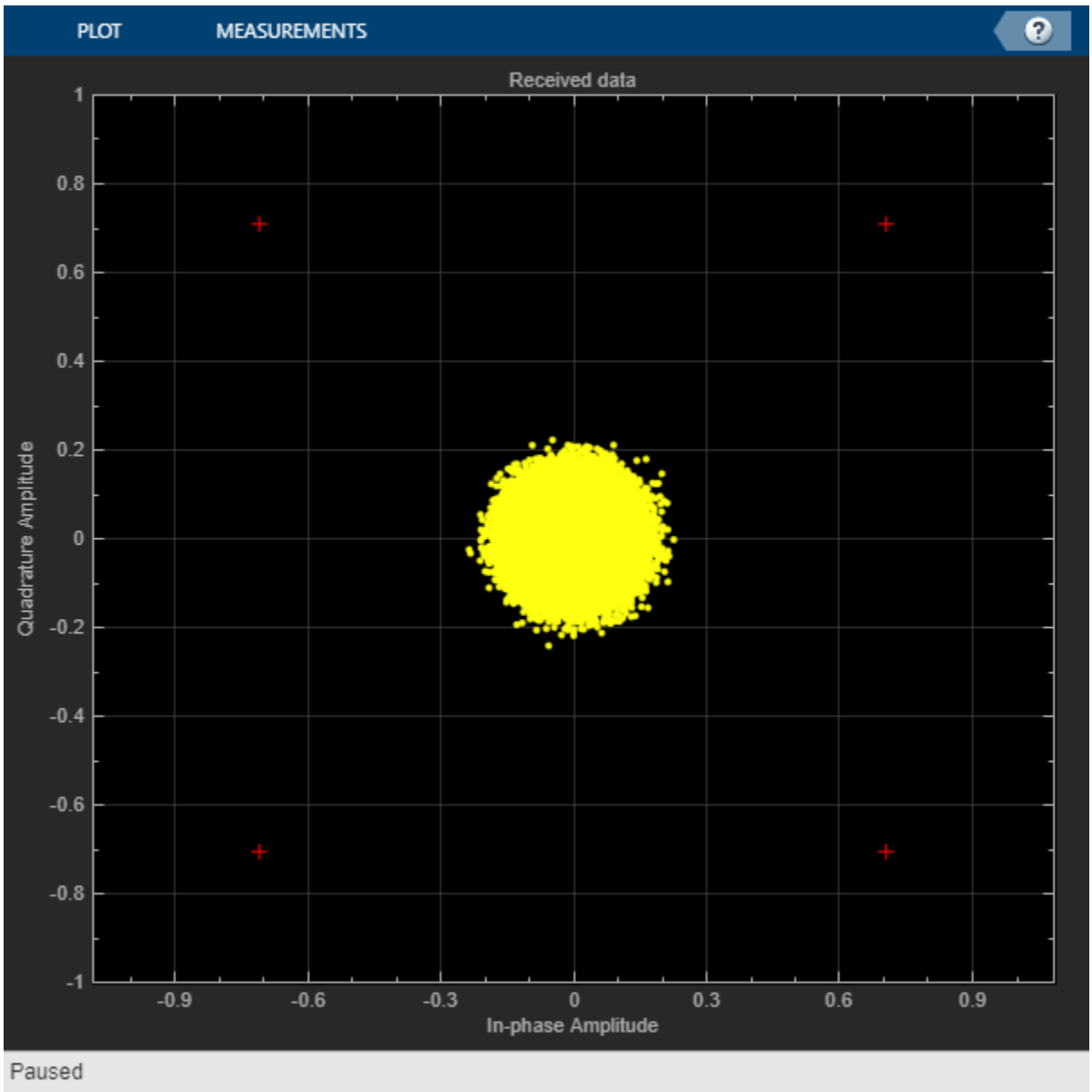
```
simParams.DisableRFImpairments = false; % Disable RF impairments
```

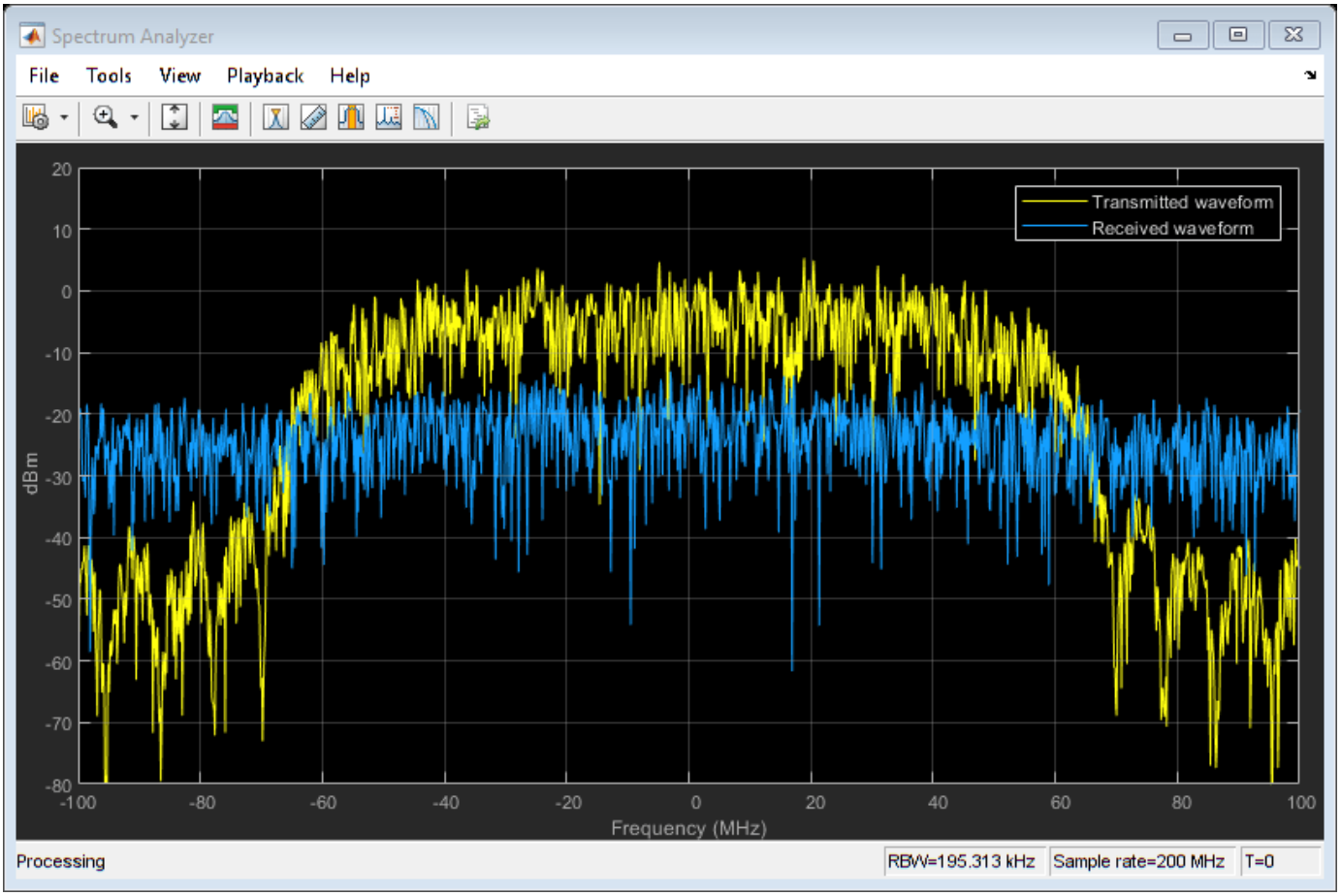
Generation of CCSDS FACM Waveform Distorted with RF Impairments

To create a CCSDS FACM waveform, use the `HelperCCSDSFACMRxInputGenerate` helper function, with the `simParams` and `cfgCCSDSFACM` structures as inputs. This function uses the `ccsdsTMWaveformGenerator` System object to generate the transmitted waveform. This function returns the data signal, transmitted and received waveforms, and receiver processing structure. The received waveform is impaired with carrier frequency, Doppler shift, timing phase offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields and pilot indices. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

```
[bits,txOut,rxIn,phyParams,rxParams] = ...
    HelperCCSDSFACMRxInputGenerate(cfgCCSDSFACM,simParams);

if showVisualizations == true

    % Plot the received signal constellation
    rxConst = comm.ConstellationDiagram('Title','Received data', ...
        'XLimits',[-1 1],'YLimits',[-1 1], ...
        'ShowReferenceConstellation',true, ...
```

```
        'SamplesPerSymbol',simParams.SPS);

    % Calculate the reference constellation for the specified ACM format.
    refConstellation = HelperCCSDSFACMReferenceConstellation(cfgCCSDSFACM.ACMFormat);
    rxConst.ReferenceConstellation = refConstellation;
    rxConst(rxIn(1:rxParams.plFrameSize*sps))

    % Plot the transmitted and received signal spectrum
    Fsamp = simParams.SymbolRate*simParams.SPS;
    scope = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
        'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
        'ShowLegend',true);
    scope([txOut,rxIn(1:length(txOut))]);
end
```





Configuration of the Receiver

Create a square root raised cosine (SRRC) receive filter.

```
rxFilterDecimationFactor = sps/2;
rrcfilt = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor',double(rolloff), ...
    'FilterSpanInSymbols',double(cfgCCSDSFACM.FilterSpanInSymbols), ...
    'InputSamplesPerSymbol',sps, ...
    'DecimationFactor',rxFilterDecimationFactor);
b = coeffs(rrcfilt);

% |H(f)| = 1  for |f| < fN(1-alpha), as per Section 6 in the standard [1]
rrcfilt.Gain = sum(b.Numerator);
```

Create a symbol timing synchronization System object, `comm.SymbolSynchronizer`.

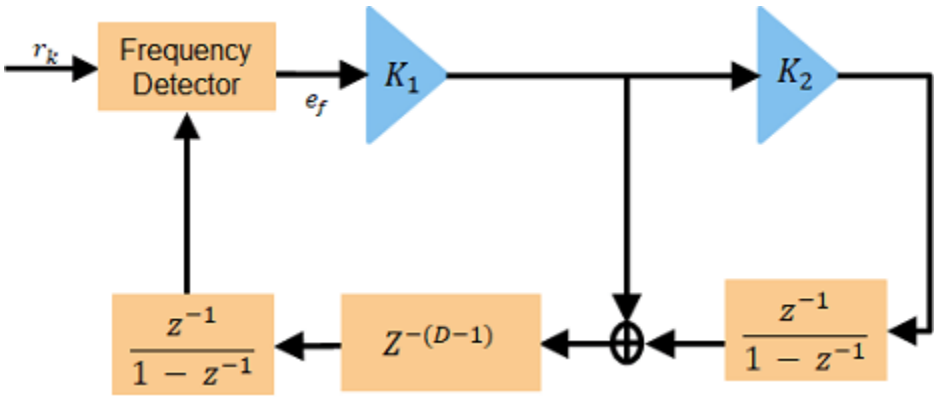
```
% Initialize the detector gain. See Eq 8.47 in Digital communications by
% Michael Rice [3].
Kp = 1/(pi*(1-((rolloff^2)/4)))*sin(pi*rolloff/2);
symsyncobj = comm.SymbolSynchronizer( ...
    'DampingFactor',1/sqrt(2), ...
    'DetectorGain',Kp, ...
    'TimingErrorDetector','Gardner (non-data-aided)', ...
    'Modulation','PAM/PSK/QAM', ...
```

```
    'NormalizedLoopBandwidth',0.0001, ...
    'SamplesPerSymbol',sps/rxFilterDecimationFactor);
```

Visualize the constellation plot after timing and frequency recovery by creating a `comm.ConstellationDiagram` System object.

```
if showVisualizations == true
    constelDiag = comm.ConstellationDiagram();
    constelDiag.ReferenceConstellation = refConstellation;
end
```

Initialize the FLL for carrier frequency synchronization. This figure shows the FLL implementation, as described in section 5.7 in [2] on page 4-0 . In this case, the frequency detector is implemented with a fast Fourier transform (FFT) based approach, where the peak in frequency domain indicates the residual carrier frequency. Because this approach is limited by the resolution of the FFT, interpolate around the peak in the frequency domain to detect the residual frequency. Because this approach uses the FM that is available in the PL header, complete the frame synchronization before passing the signal through the FLL. This figure shows the type-2 FLL that handles large Doppler rates.



Initialize the FLL. When you set K2 to 0, this FLL becomes a type-1 FLL.

```
fll = HelperCCSDSFACMFLL('SampleRate',simParams.SymbolRate, ...
    'K1',0.17,'K2',0);
```

Initialize local variables for the receiver chain to work.

```
plFrameSize = rxParams.plFrameSize;
stIdx = 0;                          % PL frame starting index

% Use one PL frame for each iteration.
endIdx = stIdx + plFrameSize*sps;
rxParams.ffBuffer = complex(zeros(plFrameSize,1));
```

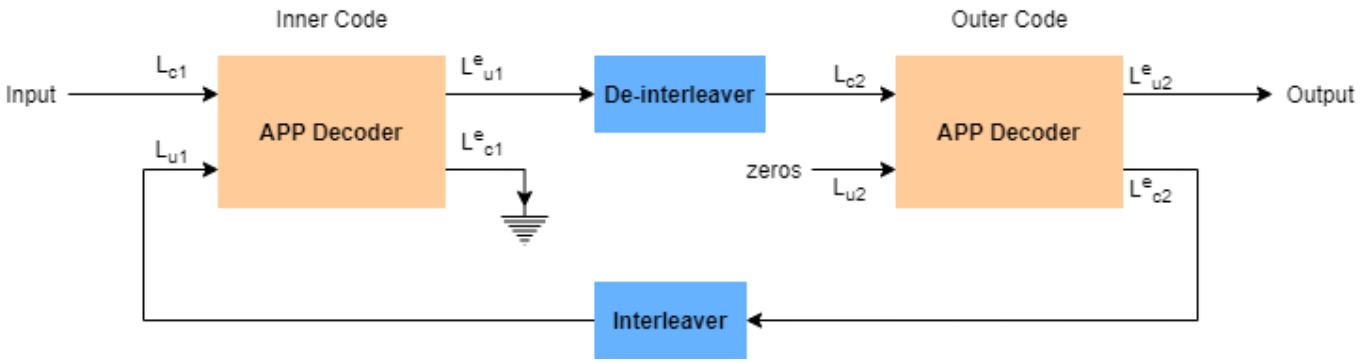
Synchronization and Data Recovery

To recover the data from the received signal, follow these steps.

1. Pass the received baseband samples through the SRRC receive filter.
2. Perform symbol timing synchronization. use the `comm.SymbolSynchronizer` System object for symbol timing synchronization. The SRO is compensated while performing symbol timing synchronization.

3. Apply frame synchronization to detect the frame boundaries.
4. Pass the frame synchronized symbols through the FLL. In the FLL, along with the CFO, the Doppler rate and Doppler shift are also tracked.
5. Estimate and compensate the remaining frequency offset in the FM using the reference FM.
6. Recover the residual phase from the phase recovery module. The phase recovery module can tolerate some residual CFO.
7. Estimate the SNR in the signal by using the phase compensated FM.
8. Pass the phase compensated signal through the digital automatic gain controller (DAGC).
9. Demodulate the synchronized symbols to get soft bits.
10. Perform SCCC decoding on the soft bits to obtain the decoded bits. SCCC decoding can be done using a-posteriori probability (APP) decoder System object, `comm.APPDecoder`.
11. Recover the transfer frames from the decoded bits.

This figure shows the SCCC decoder block diagram.



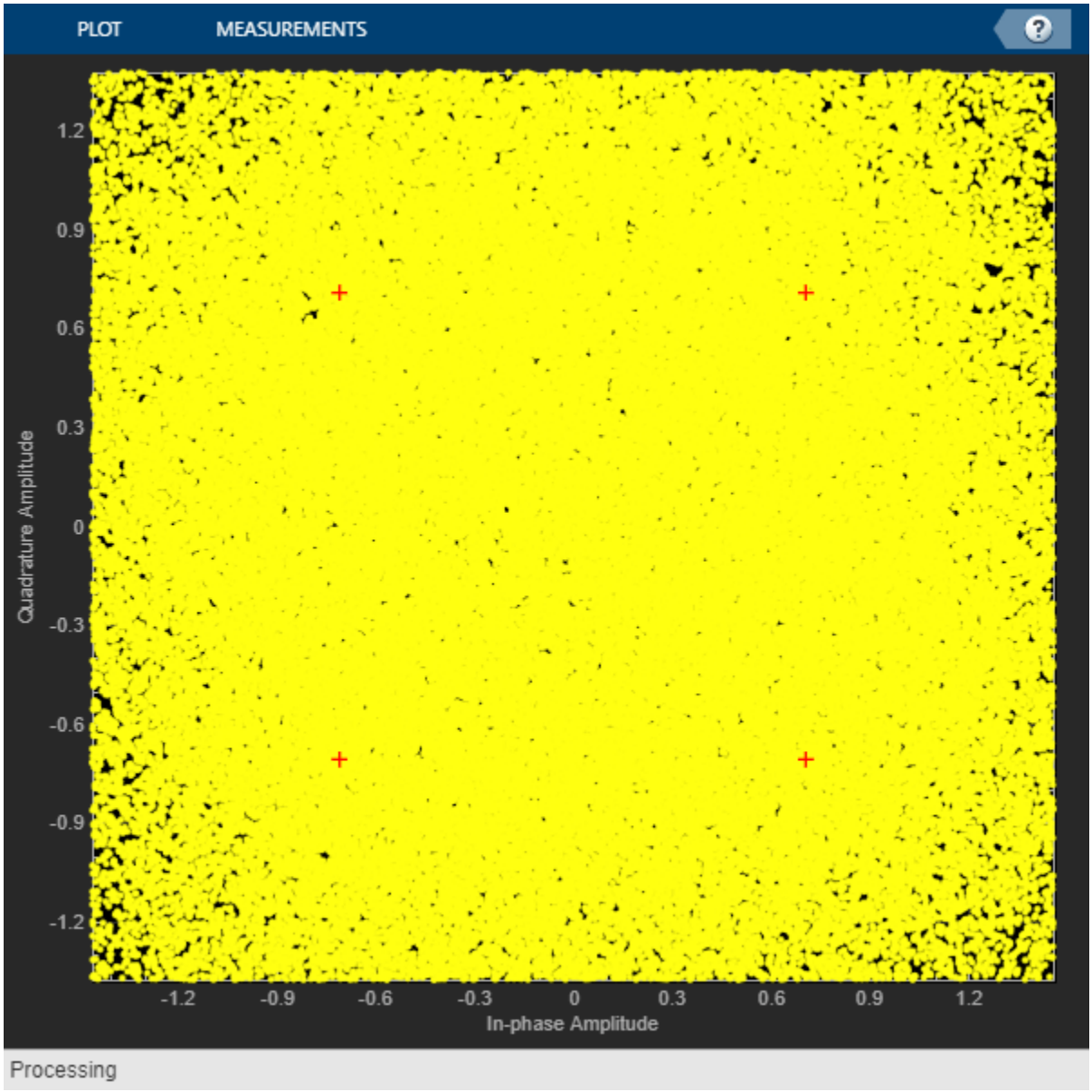

```
% Initialize the number of symbols in a code block. Assuming pilots are
% present in the signal, 15*16 pilots and 8100 data symbols exist in one
% code block.
n = 8100 + 15*16;

% Perform frame synchronization for the first frame outside the loop.

% In the last frame, consider all of the remaining samples in the received
% waveform.
isLastFrame = endIdx > length(rxIn);
endIdx(isLastFrame) = length(rxIn);
rxData = rxIn(stIdx+1:endIdx);
stIdx = endIdx; % Update start index after extracting required data

filteredRx = rrcfilt(rxData);     % RRC filter
syncsym = symsyncobj(filteredRx); % Symbol timing synchronization

% Frame synchronization
syncidx = HelperCCSDSFACMFrameSync(syncsym,rxParams.RefFM);
fineCFOSync = comm.PhaseFrequencyOffset('SampleRate',simParams.SymbolRate);

leftOutSym = syncsym(syncidx(1):end);
extraBits = [];
numIterations = 10;
frameIndex = 1;
```

```
berinfo = struct('NumBitsInError',0, ...
    'TotalNumBits',0, ...
    'BitErrorRate',0);
snrAveragingFactor = 6; % Average over 6 frames to get accurate estimate of SNR
SNREst = zeros(snrAveragingFactor,1);
idxTemp = 0;
G = 1;
numFramesLost = 0;
fqyoffset = zeros(1000,1);
while stIdx < length(rxIn)
    isFrameLost = false;

    % Use one PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*sps;

    % In the last frame, consider all of the remaining samples in the received
    % waveform.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);
    stIdx = endIdx; % Update start index after extracting required data
    if ~isLastFrame
        % Filter the received data
        filteredRx = rrcfilt(rxData); % RRC filter

        % Synchronize the symbol timing
        syncsym = symsyncobj(filteredRx);

        % Synchronize the data to the frame boundaries
        syncidx = HelperCCSDSFACMFrameSync(syncsym,rxParams.RefFM);

        % Consider one complete PL frame beginning with a header. Append
        % zeros if data is not sufficient. This situation typically happens
        % when samples are lost while doing timing synchronization or when
        % synchronization is lost.
        tempSym = [leftOutSym;syncsym(1:syncidx(1)-1)];
        leftOutSym = syncsym(syncidx(1):end);
        if length(tempSym)<n*16+320 % 16 code blocks and 320 header symbols
            fllIn = [tempSym;zeros(n*16+320-length(tempSym),1)];
        else                        % length(tempSym)>=n*16 + 320
            fllIn = tempSym(1:n*16+320);
        end

        % Track the frequency offset
        [fllOut,fqyoffset(frameIndex)] = fll(fllIn);

        % Estimate and compensate the CFO from the FM.
        cfoEst = HelperCCSDSFACMFMFrequencyEstimate(fllOut(1:256), ...
            rxParams.RefFM,simParams.SymbolRate);
        fineCFOSync.FrequencyOffset = -cfoEst;
        fqysyncedFM = fineCFOSync(fllOut(1:256));

        % Estimate and compensate for the phase offset in each frame
        % independently. Remove the pilots in the process.
        [noPilotsSym,frameDescriptor] = ...
            HelperCCSDSFACMPhaseRecovery(fllOut,rxParams.PilotSeq,rxParams.RefFM);
        agcIn = [frameDescriptor;noPilotsSym];
```

```
% Estimate the SNR. See CCSDS 130.11-G-1 Section 5.5 [2].
SNREst(idxTemp+1) = HelperCCSDSFACMSNREstimate(fqysyncedFM(1:256), ...
    rxParams.RefFM);

% Average the estimated SNR over multiple frames
finalSNREst = mean(SNREst);
idxTemp = mod(idxTemp+1,6);

% Pass the signal through DAGC
if frameIndex >= snrAveragingFactor
    [agcRecovered,G] = HelperDigitalAutomaticGainControl(agcIn,finalSNREst,G);
else
    agcRecovered = agcIn;
end

if showVisualizations == true
    % Plot the constellation.
    constelDiag(agcRecovered(:))
end

% Recover the ACM format pilots availability indicator.
[ACMFormat, hasPilots,decFail] = HelperCCSDSFACMFDRecover(agcRecovered(1:64));

if decFail
    isFrameLost = true;
end

if (ACMFormat ~= cfgCCSDSFACM.ACMFormat) || (hasPilots ~= cfgCCSDSFACM.HasPilots)
    isFrameLost = true;
end

% Continue further processing only if the frame is not lost.
if ~isFrameLost
    % De-randomize the PL-frame.
    derandomized = agcRecovered(65:end).*conj(rxParams.PLRandomSymbols(:));

    % Demodulate the signal
    nVar = 1/finalSNREst;
    demodulated = reshape(HelperCCSDSFACMDemodulate(derandomized,ACMFormat,nVar), ...
        [],16);

    fullFrameDecoded = zeros(16*phyParams.K,1);

    for iCodeWord = 1:16
        decoded = HelperSCCCDecode(demodulated(:,iCodeWord),ACMFormat,numIterations);
        fullFrameDecoded((iCodeWord-1)*phyParams.K+1:iCodeWord*phyParams.K) = ...
            decoded;
    end

    % Extract TFs.
    [initBits,deccodedBuffer,extraBits] = ...
        HelperCCSDSFACMTFSynchronize([extraBits;fullFrameDecoded], ...
        phyParams.ASM, ...
        phyParams.NumInputBits);
    PRNSeq = satcom.internal.ccsds.tmrandseq(phyParams.NumInputBits);
    if ~isempty(deccodedBuffer)
        finalBits = xor(deccodedBuffer(33:end,:)>0,PRNSeq);
    else
```

```
                isFrameLost = true;
                extraBits = [];
            end
        end

        if isFrameLost
            numFramesLost = numFramesLost + 1;
        end

        % Find BER
        if frameIndex>simParams.InitalSyncFrames && ~isFrameLost
            berinfo = HelperBitErrorRate(bits,finalBits,berinfo);
            disp(['frameIndex = ' num2str(frameIndex) '. BER = ' ...
                num2str(berinfo.BitErrorRate)])
        end
        frameIndex = frameIndex + 1;
    end
end
```

```
frameIndex = 16. BER = 0
frameIndex = 17. BER = 0
frameIndex = 18. BER = 0
frameIndex = 19. BER = 0
frameIndex = 20. BER = 0
frameIndex = 21. BER = 0
frameIndex = 22. BER = 0
frameIndex = 23. BER = 0
frameIndex = 24. BER = 0
frameIndex = 25. BER = 0
frameIndex = 26. BER = 0
frameIndex = 27. BER = 0
frameIndex = 28. BER = 0
frameIndex = 29. BER = 0
```




```
frameIndex = 30. BER = 0
```

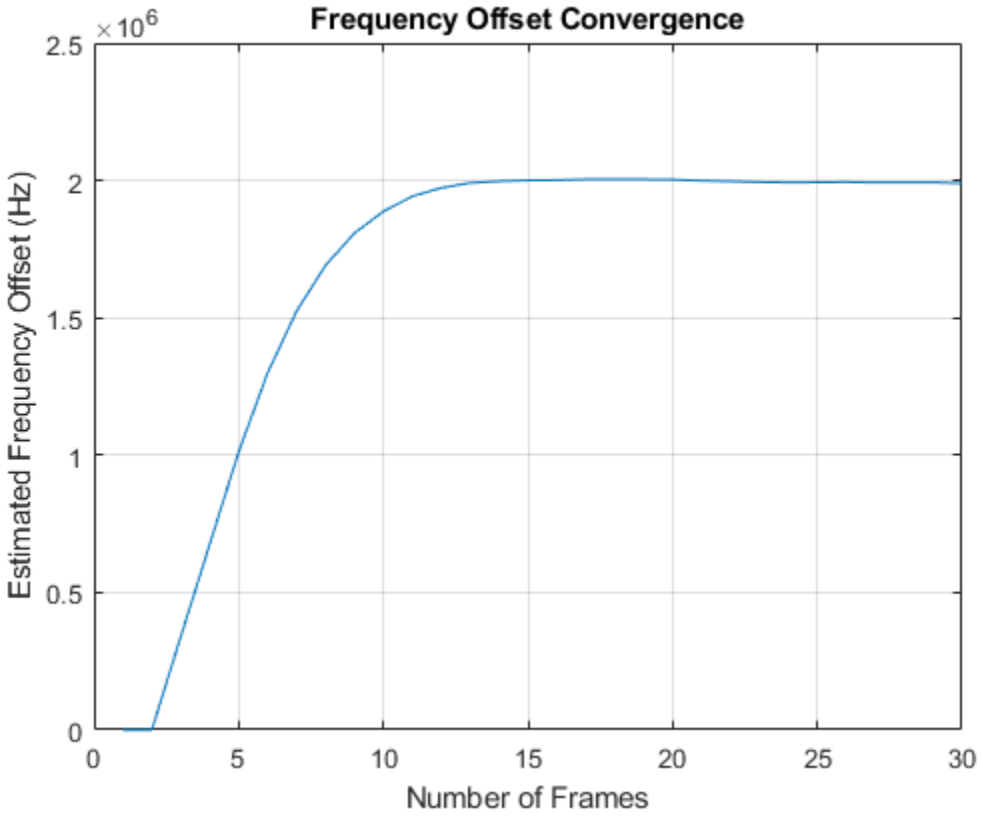
```
disp(['ACM format = ' num2str(cfgCCSDSFACM.ACMFormat) '. Es/No(dB) = ' ...
    num2str(simParams.EsNodB) '. BER = ' num2str(berinfo.BitErrorRate)])
```

```
ACM format = 1. Es/No(dB) = 1. BER = 0
```

This plot shows the frequency convergence of the estimated frequency offset. This plot is shows the number of frames required for the FLL to converge. The plot shows that the frequency offset converges even at a very low SNR (0 dB Es/No). This observation shows that the FLL can operate effectively at low SNR values.

```
if showVisualizations == true
    plot(fqyoffset(1:frameIndex-1));
    grid on
```

```
    ylabel('Estimated Frequency Offset (Hz)')
    xlabel('Number of Frames')
    title('Frequency Offset Convergence')
end
```



Further Exploration

This example shows the calculation of BER for one ACM format at one SNR point. Run BER simulations for multiple SNR points and multiple ACM formats.

This example uses Es/No as the SNR metric. To convert this to energy per bit to noise power ratio (Eb/No), use this code.

```
% ccsdsWaveGen = ccsdsTMWaveformGenerator('WaveformSource','flexible advanced coding and modulati
%     'ACMFormat',cfgCCSDSFACM.ACMFormat);
% codeRate = ccsdsWaveGen.info.ActualCodeRate;
% modOrder = ccsdsWaveGen.info.NumBitsPerSymbol;
% EbNodB = simParams.EsNodB - 10*log10(codeRate) - 10*log10(modOrder);
```

Appendix

The example uses these helper files:

- HelperBitErrorRate.m - Calculate bit error rate
- HelperCCSDSFACMDemodulate.m - Demodulate the received FACM signal
- HelperCCSDSFACMFDRecover.m - Recover frame descriptor

- HelperCCSDSFACMFLL.m - FLL for carrier frequency recovery
- HelperCCSDSFACMFMFrequencyEstimate.m - Estimate frequency offset in FM
- HelperCCSDSFACMFrameMarker.m - Generate reference FM
- HelperCCSDSFACMFrameSync.m - Estimate frame beginning
- HelperCCSDSFACMPhaseRecovery.m - Recover phase in the signal
- HelperCCSDSFACMReferenceConstellation.m - Generate reference constellation for a given ACM format
- HelperCCSDSFACMRxInputGenerate.m - Generate transmitter waveform, model RF impairments, and add AWGN
- HelperCCSDSFACMSNREstimate.m - Estimate SNR in received signal
- HelperCCSDSFACMTFSynchronize.m - Transfer frame synchronization
- HelperDigitalAutomaticGainControl.m - Digital automatic gain control
- HelperDopplerShift.m - Model sinusoidal Doppler profile
- HelperSCCCDecode.m - Iterative decoder for SCCC

References

[1] 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.

[2] CCSDS 130.11-G-1. Green Book. Issue 1. "SCCC—Summary of Definition and Performance." *Informational Report Concerning Space Data System Standards*. Washington, D.C.: CCSDS, April 2019.

[3] Rice, Michael. *Digital Communications: A Discrete-Time Approach*. Pearson/Prentice Hall, 2008.

See Also

Objects

`ccsdsTMWaveformGenerator`

Related Examples

- “Calculate Latency and Doppler in a Satellite Scenario” on page 1-55
- “End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections” on page 4-13
- “End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections” on page 4-2

End-to-End DVB-S2 Simulation with RF Impairments and Corrections

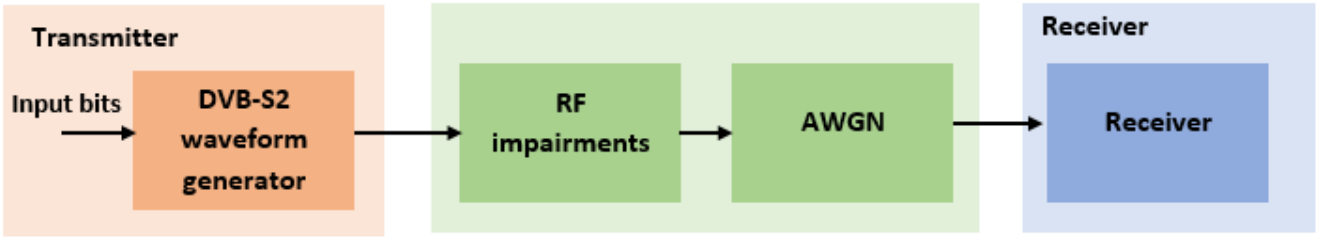
This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) link that has constant coding and modulation. The example describes the symbol timing and carrier synchronization strategies in detail, emphasizing how to estimate the RF front-end impairments under heavy noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

DVB-S2 receivers are subjected to large carrier frequency errors in the order of 20% of the input symbol rate and substantial phase noise. The use of powerful forward error correction (FEC) mechanisms, such as Bose–Chaudhuri–Hocquenghem (BCH) and low density parity check (LDPC) codes, caused the DVB-S2 system to work at very low energy per symbol to noise power spectral density ratio (E_s/N_o) values, close to the Shannon limit.

ETSI EN 302 307-1 Section 6 Table 13 [1] on page 4-0 summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. The operating E_s/N_o range for different transmission modes can be considered as +2 or -2 dB from the E_s/N_o point where QEF performance is observed. Because the operating E_s/N_o range is low, carrier and symbol timing synchronization strategies are challenging design problems.

This diagram summarizes the example workflow.



Main Processing Loop

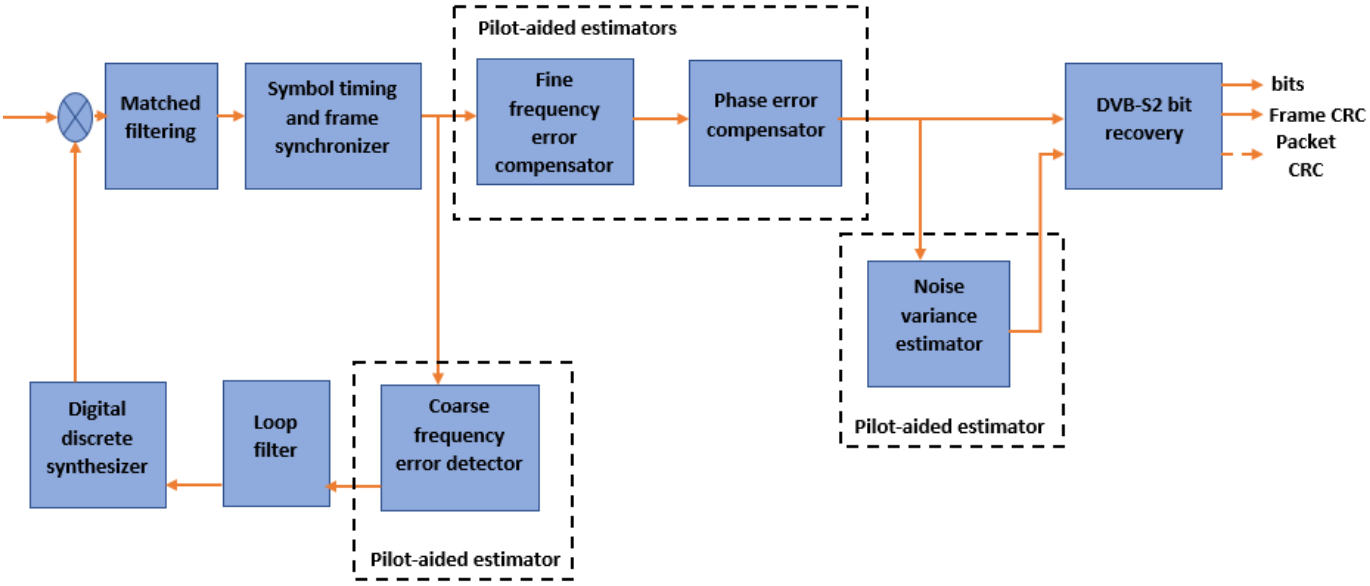
The example processes 25 physical layer (PL) frames of data with the E_s/N_o set to 20 dB, and then computes the BER and PER. Carrier frequency offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

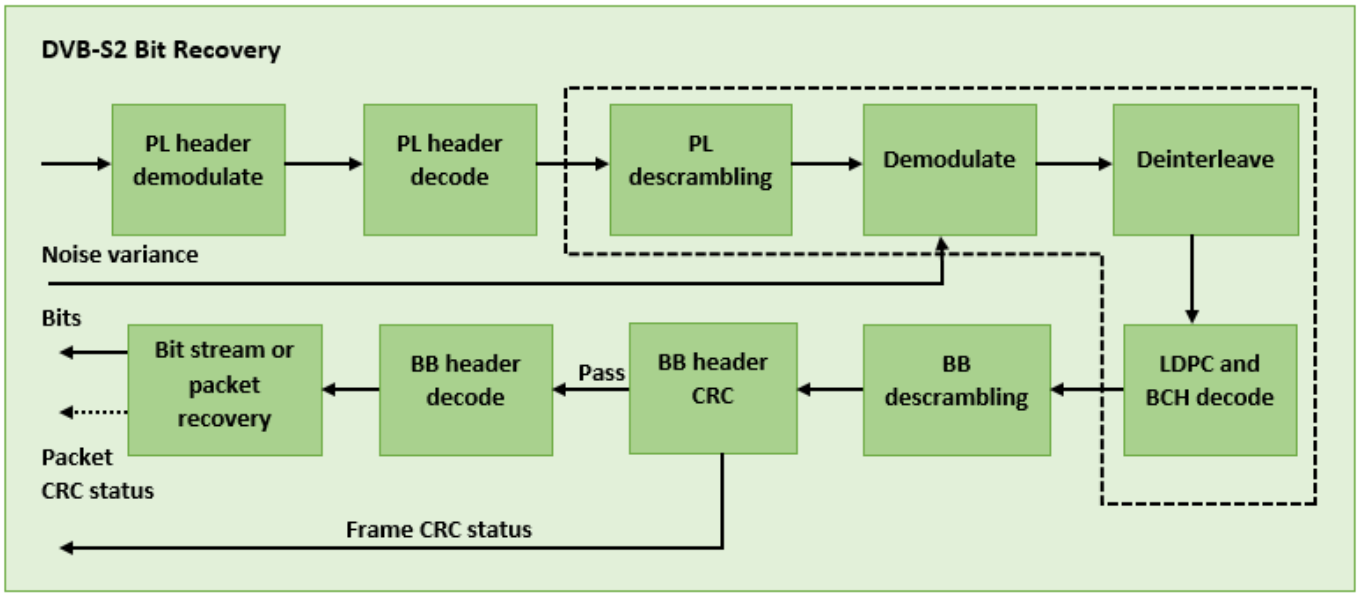
At the receiver, after matched filtering, timing and carrier recovery operations are run to recover the transmitted data. To extract PL frames, the distorted waveform is processed through various timing and carrier recovery strategies to extract PL frames. The carrier recovery algorithms are pilot-aided. To decode the data frames, the physical layer transmission parameters such as modulation scheme, code rate, and FEC frame type, are recovered from the PL header. To regenerate the input bit stream, the baseband (BB) header is decoded.

Because the DVB-S2 standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet

cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured.

These block diagrams show the synchronization and input bit recovery workflows.





Download DVB-S2 LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2 LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```
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
```

DVB-S2 Configuration in Pilot-Aided Mode

Specify the `cfgDVBS2` structure to define DVB-S2 transmission configuration parameters. The `ScalingMethod` property applies when `MODCOD` is in the range [18, 28] (that is, when the modulation scheme is APSK only). `UPL` property is applicable when you set the `StreamFormat` to "`GS`".

```
cfgDVBS2.StreamFormat = "TS";
cfgDVBS2.FECFrame = "normal";
cfgDVBS2.MODCOD = 18;                                % 16APSK 2/3
cfgDVBS2.DFL = 42960;
cfgDVBS2.ScalingMethod = "Unit average power";
cfgDVBS2.RolloffFactor = 0.35;
cfgDVBS2.HasPilots = true;
cfgDVBS2.SamplesPerSymbol = 2
```

```
cfgDVBS2 = struct with fields:
        StreamFormat: "TS"
            FECFrame: "normal"
              MODCOD: 18
                 DFL: 42960
       ScalingMethod: "Unit average power"
       RolloffFactor: 0.3500
           HasPilots: 1
    SamplesPerSymbol: 2
```

Simulation Parameters

The DVB-S2 standard supports flexible channel bandwidths. Use a typical channel bandwidth such as 36 MHz. The channel bandwidth can be varied. The coarse frequency synchronization algorithm implemented in this example can track carrier frequency offsets up to 20% of the input symbol rate. The symbol rate is calculated as B/(1+R), where B is the channel bandwidth, and R is the transmit filter roll-off factor. The algorithms implemented in this example can correct the sampling clock offset up to 10 ppm.

```
simParams.sps = cfgDVBS2.SamplesPerSymbol;           % Samples per symbol
simParams.numFrames = 25;                            % Number of frames to be processed
simParams.chanBW = 36e6;                             % Channel bandwidth in Hertz
simParams.cfo = 3e6;                                 % Carrier frequency offset in Hertz
simParams.sco = 5;                                   % Sampling clock offset in parts
                                                     % per million

simParams.phNoiseLevel = Low;                        % Phase noise level provided as
                                                     % 'Low', 'Medium', or 'High'
simParams.EsNodB = 20;                               % Energy per symbol to noise ratio
                                                     % in decibels
```

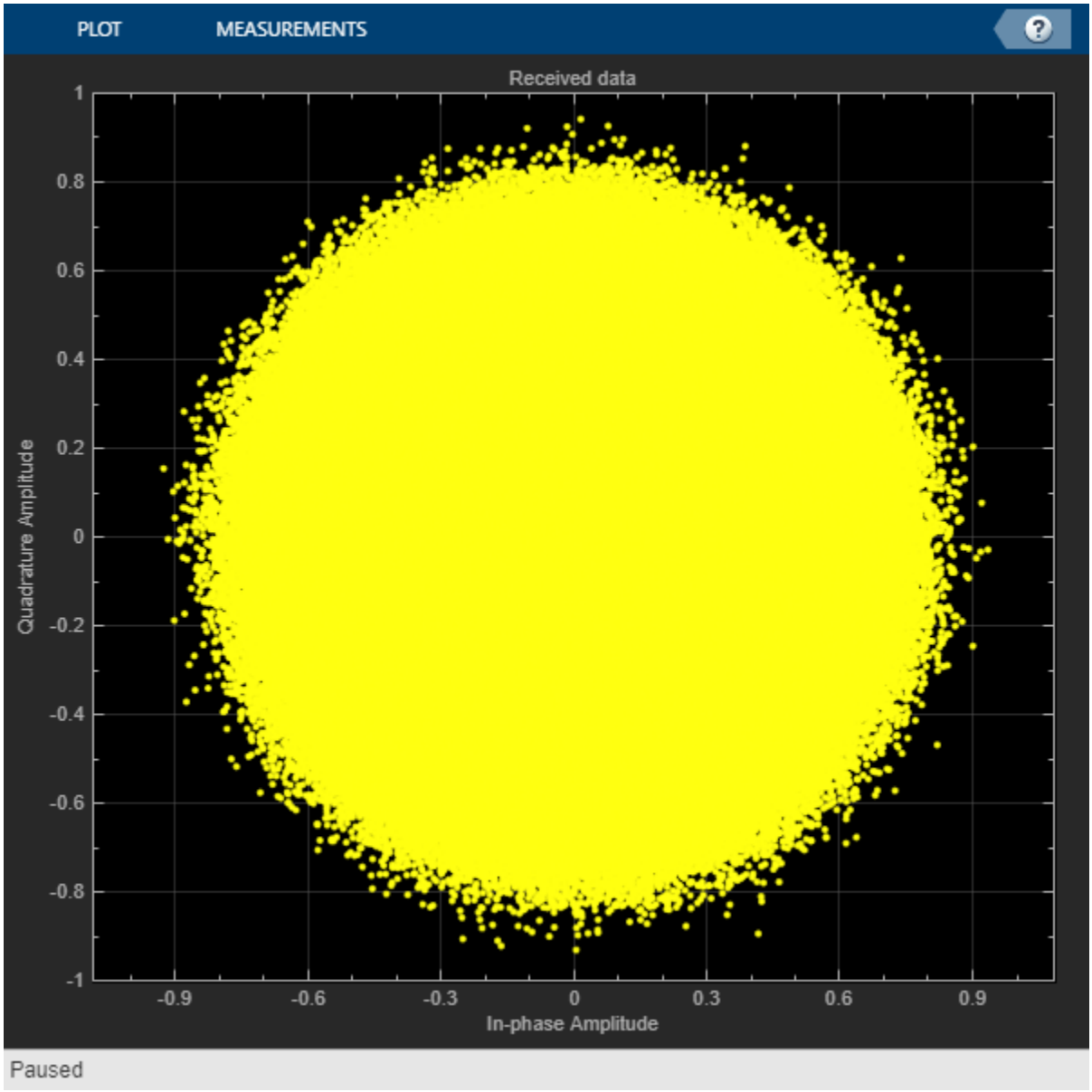
This table defines the phase noise mask (dBc/Hz) used to generate the phase noise applied to the transmitted signal.

Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz
Low	-73	-83	-93	-112	-128
Medium	-59	-77	-88	-94	-104
High	−25	-50	-73	-85	-103

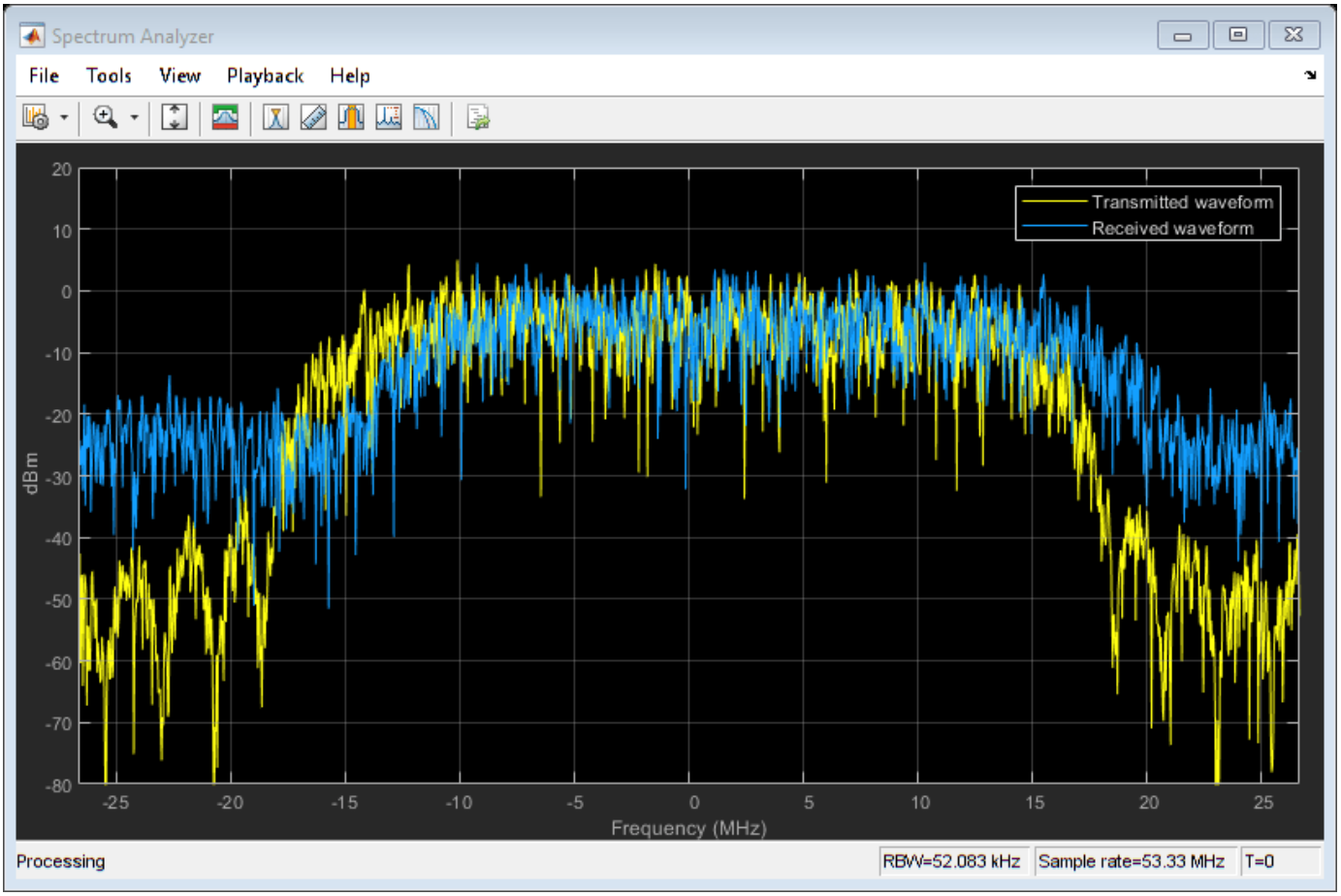
Generate DVB-S2 Waveform Distorted with RF Impairments

To create a DVB-S2 waveform, use the `HelperDVBS2RxInputGenerate` helper function with the `simParams` and `cfgDVBS2` structures as inputs. The function returns the data signal, transmitted and received waveforms, and a receiver processing structure. The received waveform is impaired with carrier frequency, timing phase offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

```
[data,txOut,rxIn,rxParams] = HelperDVBS2RxInputGenerate(cfgDVBS2,simParams);

% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:length(txOut)))
```




```
% Transmitted and received signal spectrum visualization
Rsymb = simParams.chanBW/(1 + cfgDVBS2.RolloffFactor);
Fsamp = Rsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'ShowLegend',true);
specAn([txOut, rxIn(1:length(txOut))]);
```



Configure Receiver Parameters

At the receiver, symbol timing synchronization is performed on the received data and is then followed by frame synchronization. The receiver algorithms include coarse and fine frequency impairment correction algorithms. The carrier frequency estimation algorithm can track carrier frequency offsets up to 20% of the input symbol rate. The coarse frequency estimation, implemented as a frequency locked loop (FLL), reduces the frequency offset to a level that the fine frequency estimator can track. The preferred loop bandwidth for symbol timing and coarse frequency compensation depends on the E_s/N_o setting.

A block of 36 pilots is repeated every 1476 symbols. The coarse frequency error estimation uses 34 of the 36 pilot symbols. The ratio of used pilots per block (34) and pilot periodicity (1476) is 0.023. Using the 0.023 value as a scaling factor for the coarse frequency synchronizer loop bandwidth is preferred.

When you decrease the E_s/N_o, you can reduce the loop bandwidth to filter out more noise during acquisition. The number of frames required for the symbol synchronizer and coarse FLL to converge depends on the loop bandwidth setting.

The frame synchronization uses the PL header. Because the carrier synchronization is data-aided, the frame synchronization must detect the start of frame accurately. E_s/N_o plays a crucial role in determining the accuracy of the frame synchronization. When QPSK modulated frames are being

recovered at E_s/N_ovalues below 3 dB, the frame synchronization must be performed on multiple frames for accurate detection.

The fine frequency estimation can track carrier frequency offsets up to 4% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency errors less than 0.02% of the input symbol rate. Fine phase compensation is only required for APSK modulation schemes in the presence of significant phase noise.

These settings are assigned in the `rxParams` structure for synchronization processing. For details on how to set these parameters for low E_s/N_o values, see the Further Exploration on page 4-0 section.

```
rxParams.carrSyncLoopBW = 1e-2*0.023;        % Coarse frequency estimator loop bandwidth
                                             % normalized by symbol rate
rxParams.symbSyncLoopBW = 8e-3;              % Symbol timing synchronizer loop bandwidth
                                             % normalized by symbol rate
rxParams.symbSyncLock  = 6;                  % Number of frames required for symbol
                                             % timing error convergence
rxParams.frameSyncLock = 1;                  % Number of frames required for frame
                                             % synchronization
rxParams.coarseFreqLock = 3;                 % Number of frames required for coarse
                                             % frequency acquisition
rxParams.fineFreqLock = 6;                   % Number of frames required for fine
                                             % frequency estimation
rxParams.hasFinePhaseCompensation = false;   % Flag to indicate whether fine phase
                                             % compensation is used
rxParams.finePhaseSyncLoopBW = 3.5e-4;       % Fine phase compensation loop bandwidth
                                             % normalized by symbol rate

% Total frames taken for symbol timing and coarse frequency lock to happen
rxParams.initialTimeFreqSync = rxParams.symbSyncLock + rxParams.frameSyncLock + ...
    rxParams.coarseFreqLock;
% Total frames used for overall synchronization
rxParams.totalSyncFrames = rxParams.initialTimeFreqSync + rxParams.fineFreqLock;

% Create time frequency synchronization System object by using
% HelperDVBS2TimeFreqSynchronizer helper object
timeFreqSync = HelperDVBS2TimeFreqSynchronizer( ...
    'CarrSyncLoopBW',rxParams.carrSyncLoopBW, ...
    'SymbSyncLoopBW',rxParams.symbSyncLoopBW, ...
    'SamplesPerSymbol',simParams.sps, ...
    'DataFrameSize',rxParams.xFecFrameSize, ...
    'SymbSyncTransitFrames',rxParams.symbSyncLock, ...
    'FrameSyncAveragingFrames',rxParams.frameSyncLock);

% Create fine phase compensation System object by using
% HelperDVBS2FinePhaseCompensator helper object. Fine phase
% compensation is only required for 16 and 32 APSK modulated frames
if cfgDVBS2.MODCOD >= 18 && rxParams.hasFinePhaseCompensation
    finePhaseSync = HelperDVBS2FinePhaseCompensator( ...
        'DataFrameSize',rxParams.xFecFrameSize, ...
        'NormalizedLoopBandwidth',rxParams.finePhaseSyncLoopBW);
end

normFlag = cfgDVBS2.MODCOD >= 18 && strcmpi(cfgDVBS2.ScalingMethod,'Outer radius as 1');
```

```
% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

% Initialize data indexing variables
stIdx = 0;
dataSize = rxParams.inputFrameSize;
plFrameSize = rxParams.plFrameSize;
dataStInd = rxParams.totalSyncFrames + 1;
isLastFrame = false;
symSyncOutLen = zeros(rxParams.initialTimeFreqSync,1);
```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data and recover the input bit stream, the distorted DVB-S2 waveform samples are processed one frame at a time by following these steps.

1. Apply matched filtering, outputting at the rate of two samples per symbol.
2. Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate. The Gardner TED is not data-aided, so it is performed before carrier synchronization.
3. Apply frame synchronization to detect the start of frame and to identify the pilot positions.
4. Estimate and apply coarse frequency offset correction.
5. Estimate and apply fine frequency offset correction.
6. Estimate and compensate for residual carrier frequency and phase noise.
7. Decode the PL header and compute the transmission parameters.
8. Demodulate and decode the PL frames.
9. Perform CRC check on the BB header, if the check passes, recover the header parameters.
10. Regenerate the input stream of data or packets from BB frames.

```
while stIdx < length(rxIn)

    % Use one DVB-S2 PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are considered.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % After coarse frequency offset loop is converged, the FLL works with a
    % reduced loop bandwidth.
    if rxParams.frameCount < rxParams.initialTimeFreqSync
        coarseFreqLock = false;
    else
        coarseFreqLock = true;
    end

    % Retrieve the last frame samples.
    if isLastFrame
        resSymb = plFrameSize - length(rxParams.cfBuffer);
        resSampCnt = resSymb*rxParams.sps - length(rxData);
        if resSampCnt >= 0    % Inadequate number of samples to fill last frame
            syncIn = [rxData;zeros(resSampCnt, 1)];
```

```
        else                    % Excess samples are available to fill last frame
            syncIn = rxData(1:resSymb*rxParams.sps);
        end
    else
        syncIn = rxData;
    end

    % Apply matched filtering, symbol timing synchronization, frame
    % synchronization, and coarse frequency offset compensation.
    [coarseFreqSyncOut,syncIndex,phEst] = timeFreqSync(syncIn,coarseFreqLock);
    if rxParams.frameCount <= rxParams.initialTimeFreqSync
        symSyncOutLen(rxParams.frameCount) = length(coarseFreqSyncOut);
        if any(abs(diff(symSyncOutLen(1:rxParams.frameCount))) > 5)
            error(['Symbol timing synchronization failed. The loop will not ' ...
                'converge. No frame will be recovered. Update the symbSyncLoopBW ' ...
                'parameter according to the EsNo setting for proper loop convergence.']);
        end
    end

    rxParams.syncIndex = syncIndex;

    % The PL frame start index lies somewhere in the middle of the chunk being processed.
    % From fine frequency estimation onwards, the processing happens as a PL frame.
    % A buffer is used to store symbols required to fill one PL frame.
    if isLastFrame
        fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut];
    else
        fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut(1:rxParams.syncIndex-1)];
    end

    % Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
    % helper function.
    % Add 1 to the conditional check because the buffer used to get one PL
    % frame introduces a delay of one to the loop count.
    if (rxParams.frameCount > rxParams.initialTimeFreqSync + 1) && ...
            (rxParams.frameCount <= rxParams.totalSyncFrames + 1)
        rxParams.fineFreqCorrVal = HelperDVBS2FineFreqEst( ...
            fineFreqIn(rxParams.pilotInd),rxParams.numPilots, ...
            rxParams.refPilots,rxParams.fineFreqCorrVal);
    end
    if rxParams.frameCount >= rxParams.totalSyncFrames + 1
        fineFreqLock = true;
    else
        fineFreqLock = false;
    end

    if fineFreqLock
        % Normalize the frequency estimate by the input symbol rate
        % freqEst = angle(R)/(pi*(N+1)), where N (18) is the number of elements
        % used to compute the mean of auto correlation (R) in
        % HelperDVBS2FineFreqEst.
        freqEst = angle(rxParams.fineFreqCorrVal)/(pi*(19));

        % Generate the symbol indices using frameCount and plFrameSize.
        % Subtract 2 from the rxParams.frameCount because the buffer used to get one
        % PL frame introduces a delay of one to the count.
        ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
        phErr = exp(-1j*2*pi*freqEst*ind);
```

```
        fineFreqOut = fineFreqIn.*phErr(:);

        % Estimate the phase error estimation by using the HelperDVBS2PhaseEst
        % helper function.
        [phEstRes,rxParams.prevPhaseEst] = HelperDVBS2PhaseEst( ...
            fineFreqOut(rxParams.pilotInd),rxParams.refPilots,rxParams.prevPhaseEst);

        % Compensate for the residual frequency and phase offset by using
        % the
        % HelperDVBS2PhaseCompensate helper function.
        % Use two frames for initial phase error estimation. Starting with the
        % second frame, use the phase error estimates from the previous frame and
        % the current frame in compensation.
        % Add 3 to the frame count comparison to account for delays: One
        % frame due to rxParams.cfBuffer delay and two frames used for phase
        % error estimate.
        if rxParams.frameCount >= rxParams.totalSyncFrames + 3
            coarsePhaseCompOut = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
                rxParams.pilotEst,rxParams.pilotInd,phEstRes(2));
            % MODCOD >= 18 corresponds to APSK modulation schemes
            if cfgDVBS2.MODCOD >= 18 && rxParams.hasFinePhaseCompensation
                phaseCompOut = finePhaseSync(coarsePhaseCompOut);
            else
                phaseCompOut = coarsePhaseCompOut;
            end
        end

        rxParams.ffBuffer = fineFreqOut;
        rxParams.pilotEst = phEstRes;

        % The phase compensation on the data portion is performed by
        % interpolating the phase estimates computed on consecutive pilot
        % blocks. The second phase estimate is not available for the data
        % portion after the last pilot block in the last frame. Therefore,
        % the slope of phase estimates computed on all pilot blocks in the
        % last frame is extrapolated and used to compensate for the phase
        % error on the final data portion.
        if isLastFrame
            pilotBlkLen = 36;    % Symbols
            pilotBlkFreq = 1476; % Symbols
            avgSlope = mean(diff(phEstRes(2:end)));
            chunkLen = rxParams.plFrameSize - rxParams.pilotInd(end) + ...
                rxParams.pilotInd(pilotBlkLen);
            estEndPh = phEstRes(end) + avgSlope*chunkLen/pilotBlkFreq;
            coarsePhaseCompOut1 = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
                rxParams.pilotEst,rxParams.pilotInd,estEndPh);
            % MODCOD >= 18 corresponds to APSK modulation schemes
            if cfgDVBS2.MODCOD >= 18 && rxParams.hasFinePhaseCompensation
                phaseCompOut1 = finePhaseSync(coarsePhaseCompOut1);
            else
                phaseCompOut1 = coarsePhaseCompOut1;
            end
        end
    end

    % Recover the input bit stream.
    if rxParams.frameCount >= rxParams.totalSyncFrames + 3
        isValid = true;
```

```
        if isLastFrame
            syncOut = [phaseCompOut; phaseCompOut1];
        else
            syncOut = phaseCompOut;
        end
    else
        isValid = false;
        syncOut = [];
    end

    % Update the buffers and counters.
    rxParams.cfBuffer = coarseFreqSyncOut(rxParams.syncIndex:end);
    rxParams.syncIndex = syncIndex;
    rxParams.frameCount = rxParams.frameCount + 1;

    if isValid  % Data valid signal

        % Decode the PL header by using the HelperDVBS2PLHeaderRecover helper
        % function. Start of frame (SOF) is 26 symbols, which are discarded
        % before header decoding. They are only required for frame
        % synchronization.
        rxPLSCode = syncOut(27:90);
        [M,R,fecFrame,pilotStat] = HelperDVBS2PLHeaderRecover(rxPLSCode);
        xFECFrameLen = fecFrame/log2(M);
        % Validate the decoded PL header.
        if M ~= rxParams.modOrder || R ~= rxParams.codeRate || ...
                fecFrame ~= rxParams.cwLen || ~pilotStat
            fprintf('%s\n','PL header decoding failed')
        else % Demodulation and decoding
            for frameCnt = 1:length(syncOut)/plFrameSize
                rxFrame = syncOut((frameCnt-1)*plFrameSize+1:frameCnt*plFrameSize);
                % Estimate noise variance by using
                % HelperDVBS2NoiseVarEstimate helper function.
                nVar = HelperDVBS2NoiseVarEstimate(rxFrame,rxParams.pilotInd,...
                    rxParams.refPilots,normFlag);
                % The data begins at symbol 91 (after the header symbols).
                rxDataFrame = rxFrame(91:end);
                % Recover the BB frame.
                rxBBFrame = satcom.internal.dvbs.s2BBFrameRecover(rxDataFrame,M,R, ...
                    fecFrame,pilotStat,nVar,false);
                % Recover the input bit stream by using
                % HelperDVBS2StreamRecover helper function.
                if strcmpi(cfgDVBS2.StreamFormat,'GS') && ~rxParams.UPL
                    [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame);
                    if ~isFrameLost && length(decBits) ~= dataSize
                        isFrameLost = true;
                    end
                else
                    [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame);
                    if ~isFrameLost && length(decBits) ~= dataSize
                        isFrameLost = true;
                        pktCRC = zeros(0,1,'logical');
                    end
                    % Compute the packet error rate for TS or GS packetized
                    % mode.
                    pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
                    pktsRec = pktsRec + numel(pktCRC);
                end
```

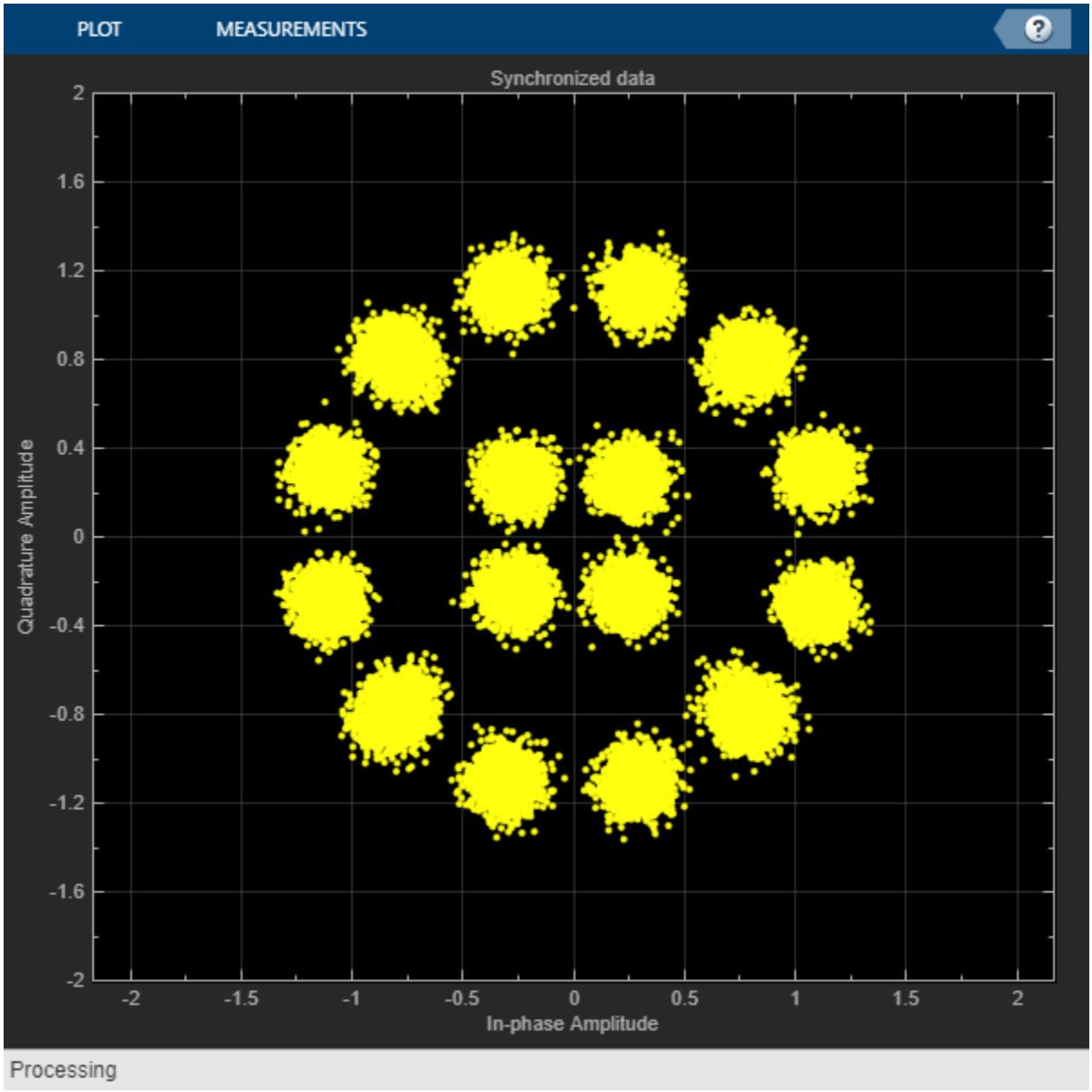
```
                if ~isFrameLost
                    ts = sprintf('%s','BB header decoding passed.');
                else
                    ts = sprintf('%s','BB header decoding failed.');
                end
                % Compute the number of frames lost. CRC failure of baseband header
                % is considered a frame loss.
                numFramesLost = isFrameLost + numFramesLost;
                fprintf('%s(Number of frames lost = %1d)\n',ts,numFramesLost)
                % Compute the bits in error.
                bitInd = (dataStInd-1)*dataSize+1:dataStInd*dataSize;
                if isLastFrame && ~isFrameLost
                    bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
                else
                    if ~isFrameLost
                        bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
                    end
                end
                dataStInd = dataStInd + 1;
            end
        end
    end
    stIdx = endIdx;
end
```

```
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
```

Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```
% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title','Synchronized data', ...
    'XLimits',[-2 2],'YLimits',[-2 2], ...
    'ShowReferenceConstellation',false);
syncConst(syncOut)
```




```
pause(0.5)
% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames-rxParams.totalSyncFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        ber = bitsErr/((dataStInd-rxParams.totalSyncFrames)*dataSize);
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
    else
```

```
        if strcmpi(cfgDVBS2.StreamFormat,'TS')
            pktLen = 1504;
        else
            pktLen = cfgDVBS2.UPL;      % UP length including sync byte
        end
        ber = bitsErr/(pktsRec*pktLen);
        per = pktsErr/pktsRec;
        fprintf('PER: %1.2e\n',per)
        fprintf('BER: %1.2e\n',ber)
    end
end
```

```
PER: 0.00e+00

BER: 0.00e+00
```

Further Exploration

The operating E_s/N_o range of the DVB-S2 standard being very low requires the normalized loop bandwidth of the symbol synchronizer and coarse FLL to be very small for accurate estimation. These parameters are set via `rxParams.symbSyncLoopBW` and `rxParams.carrSyncLoopBW`.

Configure Symbol Timing Synchronization Parameters

Try running the simulation using the symbol timing synchronizer configured with a normalized loop bandwidth of 1e-4. With loop bandwidths at this level, this table shows the typical number of frames required for convergence of the symbol timing loop for specific modulation schemes and `'normal'` FEC frames.

Modulation scheme	Number of frames
QPSK	10 - 12
8 PSK	15 - 17
16 APSK	20 - 22
32 APSK	25 - 27

For '`short`' FEC frames, the number of frames used for symbol timing synchronization is thrice the number required for '`normal`' FEC frames. To achieve convergence of the timing loop, the ratio `rxParams.symbSyncLoopBW/simParams.sps` must be greater than 1e-5. If the symbol timing loop doesn't converge, try increasing the `rxParams.carrSyncLoopBW` .

Configure Frame Synchronization Parameters

Choose a `rxParams.symbSyncLock` value from the table provided in Configure Symbol Timing Synchronization Parameters on page 4-0 section. Set `rxParams.frameSyncLock` as a value in the range of [5, 15] frames based on the E_s/N_o setting. If the output is not as expected, increase the number of frames required for frame synchronization.

Configure Carrier Synchronization Parameters

Try running the simulation using the coarse FLL configured with a normalized loop bandwidth of 1e-4*0.023 for PSK signals and 4e-4*0.023 for APSK signals.

When you set the `FECFrame` property to `'normal'`, set the `rxParams.coarseFreqLock` property to 20. When you set the `FECFrame` property to `'short'`, set the `rxParams.coarseFreqLock` property to 80. Set `simParams.EsNodB` to the lowest E_s/N_o for the chosen modulation scheme from ETSI EN 302 307-1 Section 6 [1] on page 4-0 . For the `HelperDVBS2TimeFreqSynchronizer` system object, set its properties as discussed in the previous sections based on the chosen configuration.

```
% timeFreqSync = HelperDVBS2TimeFreqSynchronizer( ...
%    'CarrFreqLoopBW',rxParams.carrSyncLoopBW, ...
%    'SymbTimeLoopBW',rxParams.symbSyncLoopBW, ...
%    'SamplesPerSymbol',simParams.sps, ...
%    'DataFrameSize',rxParams.xFecFrameSize, ...
%    'SymbSyncTransitFrames',rxParams.symbSyncLock, ...
%    'FrameSyncAveragingFrames',rxParams.frameSyncLock)
```

Replace the code in the symbol timing and coarse frequency synchronization section with these lines of code. Run the simulation for different carrier frequency offset (CFO) values. After coarse frequency compensation, view the plot and the residual CFO value (`resCoarseCFO`) over each frame to observe the performance of the coarse frequency estimation algorithm. Ideally, the coarse frequency compensation reduces the error to 2% of the symbol rate. If the coarse frequency compensation is not reduced to less than 3% of the symbol rate, try decreasing the loop bandwidth and increasing the `rxParams.coarseFreqLock`. Because the frequency error is estimated using the pilot symbols, verify that the frame synchronizer is properly locked to the beginning of frame.

```
% [out,index,phEst] = timeFreqSync(rxData,false);
% Frequency offset estimate normalized by symbol rate
% freqOffEst = diff(phEst(1:simParams.sps:end))/(2*pi);
% plot(freqOffEst)
% actFreqOff = simParams.cfo/(simParams.chanBW/(1 + cfgDVBS2.RolloffFactor));
% resCoarseCFO = abs(actFreqOff-freqOffEst(end));
```

When the residual carrier frequency offset value (`resCoarseCFO`) is reduced to approximately 0.02 or 0.03, set the `rxParams.frameCount` to the `rxParams.coarseFreqLock` value.

For `'normal'` FEC frames, set the `rxParams.fineFreqLock` value to 10. For `'short'` FEC frames, set the `rxParams.fineFreqLock` value to 40. Replace the code in the fine frequency error estimation section with this code.

```
% rxParams.fineFreqCorrVal = HelperDVBS2FineFreqEst(fineFreqIn(rxParams.pilotInd), ...
%         rxParams.numPilots,rxParams.refPilots,rxParams.fineFreqCorrVal);
% fineFreqEst = angle(rxParams.fineFreqCorrVal)/(pi*(19));
% resFineCFO = abs(actFreqOff-freqOffEst(end)-fineFreqEst);
```

Repeat the simulation process and observe the residual CFO value (`resFineCFO`) over each frame. If the fine frequency estimator does not reduce the residual carrier frequency error to approximately 0.01% of the symbol rate, try increasing the `rxParams.fineFreqLock` property value.

When the residual CFO value (`resFineCFO`) is reduced to approximately 0.0001 or 0.0002, set the `rxParams.frameCount+1` to the `rxParams.coarseFreqLock` value.

Fine phase compensation PLL is used for only 16 APSK and 32 APSK modulation schemes with substantial phase noise.

After refining the synchronization parameters set in the `rxParams` structure, perform the BER simulation for the updated configuration.

Appendix

The example uses these helper functions:

- HelperDVBS2RxInputGenerate.m: Generate DVB-S2 waveform samples distorted with RF impairments and structure of parameters for receiver processing
- HelperDVBS2PhaseNoise.m: Generate phase noise samples for different DVB-S2 phase noise masks and apply it to the input signal
- HelperDVBS2TimeFreqSynchronizer.m: Perform matched filtering, symbol timing synchronization, frame synchronization, and coarse frequency estimation and correction
- HelperDVBS2FrameSync.m: Perform frame synchronization and detect the start of frame
- HelperDVBS2FineFreqEst.m: Estimate fine frequency offset
- HelperDVBS2PhaseEst.m: Estimate carrier phase offset
- HelperDVBS2PhaseCompensate.m: Perform carrier phase compensation
- HelperDVBS2FinePhaseCompensator.m: Perform fine carrier phase error tracking and compensation for APSK modulation schemes
- HelperDVBS2PLHeaderRecover.m: Demodulate and decode PL header to recover transmission parameters
- HelperDVBS2NoiseVarEstimate.m: Estimate noise variance of received data
- HelperDVBS2StreamRecover.m: Perform CRC check of BB header and recover input stream from BB frame based on header parameters

Bibliography

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 TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
3. Mengali, Umberto, and Aldo N.D'Andrea. *Synchronization Techniques for Digital Receivers*. New York: Plenum Press,1997.
4. E. Casini, R. De Gaudenzi, and Alberto Ginesi. "DVB-S2 modem algorithms design and performance over typical satellite channels." *International Journal of Satellite Communications and Networking* 22, no. 3 (2004): 281-318.
5. Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

See Also

Objects

`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator`

Related Examples

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames” on page 4-53

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames” on page 4-68

End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames

This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) link that has constant coding and modulation for regular frames. The example describes the symbol timing and carrier synchronization strategies in detail emphasizing on how to estimate the RF front-end impairments under heavy noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

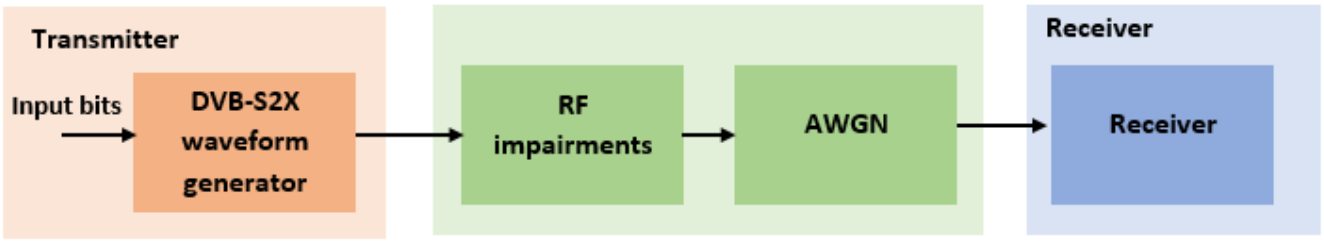
The DVB-S2X standard, an extension of the DVB-S2 specification, enhances the support provided for core DVB-S2 applications and improves overall efficiency over satellite links. The DVB-S2X standard supports these additional features:

- More granularity of modulation and code rates
- Smaller filter roll-off options for better utilization of bandwidth
- Constellations optimized for linear and nonlinear channels
- More scrambling options for critical co-channel interference scenarios

DVB-S2X caters to a variety of different target applications, and the receivers are subjected to different types and levels of RF impairments based on the application used. This example designs the synchronization aspects of a DVB-S2X receiver used for core DVB-S2 applications. The example supports the newer code rates, higher modulation schemes such as 64, 128 and 256 APSK, and smaller filter roll-off options.

ETSI EN 302 307-2 Section 6 Table 20a and Table 20c [1] on page 4-0 summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. The operating E_s/N_o range for different transmission modes can be considered as +3 or -2 dB from the E_s/N_o point where QEF performance is observed. Because the operating E_s/N_o range is low, carrier and symbol timing synchronization strategies are challenging design problems.

This diagram summarizes the example workflow.



Main Processing Loop

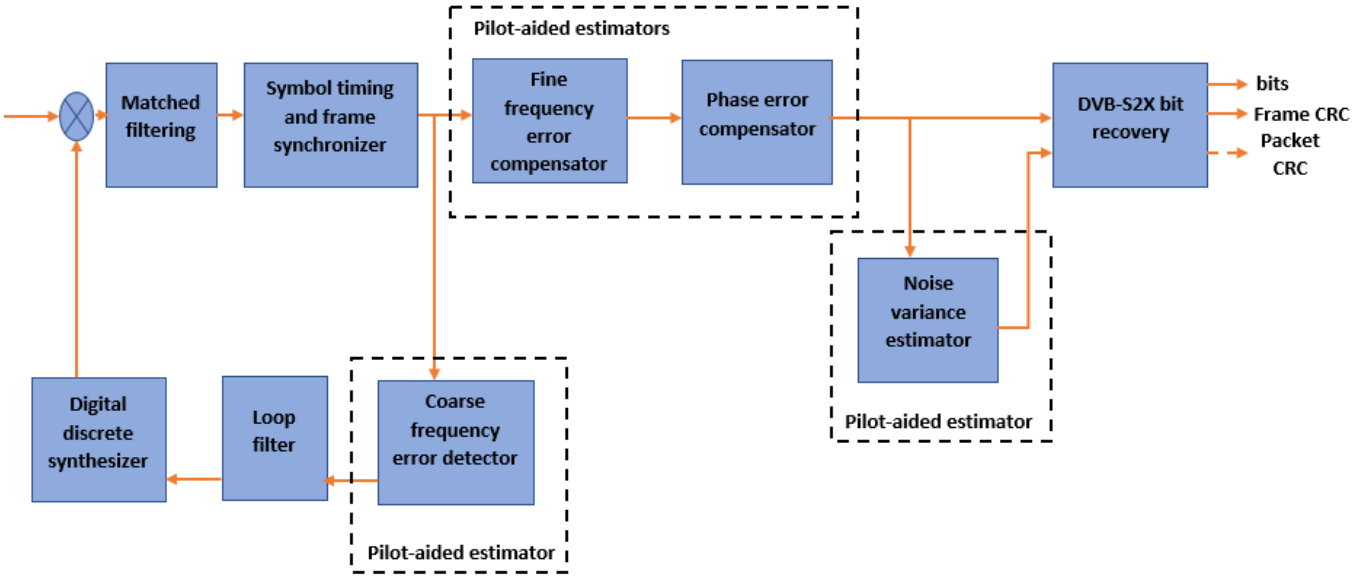
The example processes 30 physical layer (PL) frames of data with the E_s/N_o set to 25 dB, and then computes the BER and PER. Carrier frequency offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

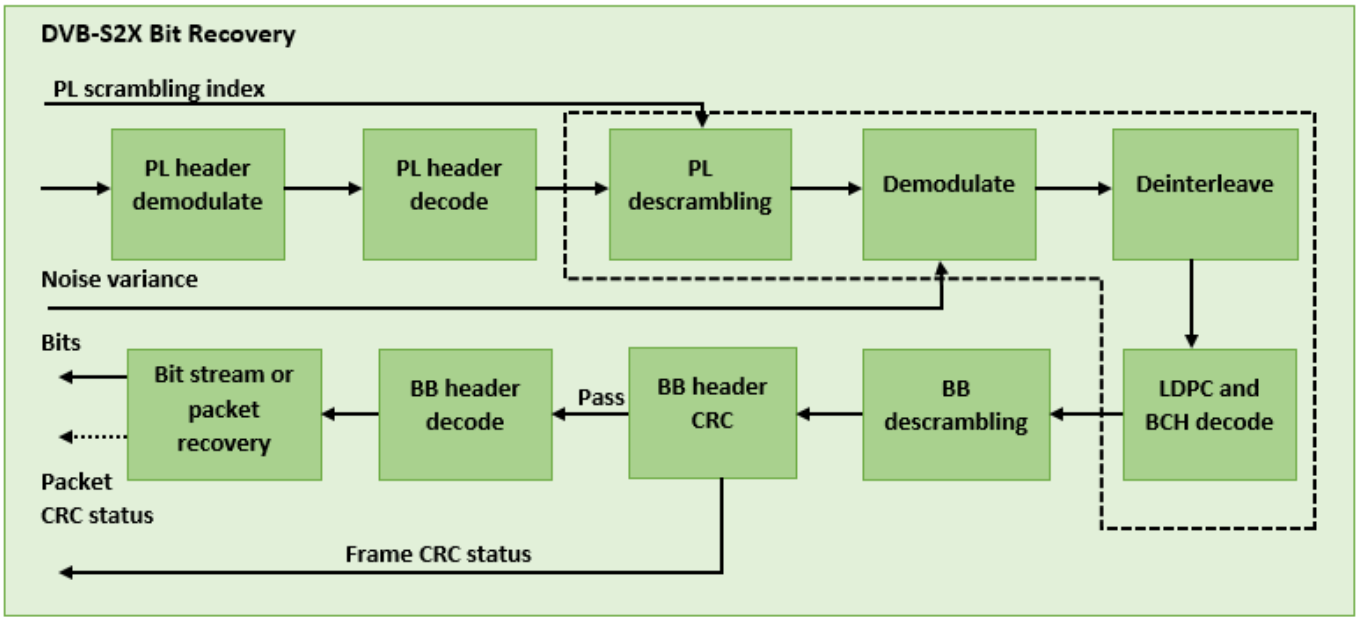
At the receiver, after matched filtering, timing and carrier recovery operations are run to recover the transmitted data. To extract PL frames, the distorted waveform is processed through various timing

and carrier recovery strategies The carrier recovery algorithms are pilot-aided. To decode the data frames, the physical layer transmission parameters, such as modulation scheme, code rate, and FEC frame type, are recovered from the PL header. To regenerate the input bit stream, the baseband (BB) header is decoded.

Because the DVB-S2X standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured.

These block diagrams show the synchronization and input bit recovery workflows.





Download DVB-S2X LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```
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
```

DVB-S2X Configuration in Pilot-Aided Mode

Specify the `cfgDVBS2X` structure to define DVB-S2X transmission configuration parameters. `PLSDecimalCode` 129 and 131 are not supported because they are used for generating VL-SNR frames. Only the regular frames are supported.

```
cfgDVBS2X.StreamFormat = "TS";
cfgDVBS2X.PLSDecimalCode = 191;                    % 64APSK 7/9 with pilots
cfgDVBS2X.DFL = 50128;
cfgDVBS2X.ScalingMethod = "Unit average power";
cfgDVBS2X.RolloffFactor = 0.35;
cfgDVBS2X.SamplesPerSymbol = 2
```

```
cfgDVBS2X = struct with fields:
        StreamFormat: "TS"
       PLSDecimalCode: 191
                  DFL: 50128
       ScalingMethod: "Unit average power"
       RolloffFactor: 0.3500
    SamplesPerSymbol: 2
```

Simulation Parameters

The DVB-S2X standard supports flexible channel bandwidths. Use a typical channel bandwidth such as 36 MHz. The channel bandwidth can be varied. The coarse frequency synchronization algorithm implemented in this example can track carrier frequency offsets up to 11% of the input symbol rate. The symbol rate is calculated as B/(1+R), where B is the channel bandwidth, and R is the transmit filter roll-off factor. The algorithms implemented in this example can correct the sampling clock offset up to 10 ppm.

```
simParams.sps = cfgDVBS2X.SamplesPerSymbol;            % Samples per symbol
simParams.numFrames = 30;                              % Number of frames to be processed
simParams.chanBW = 36e6;                               % Channel bandwidth in Hertz
simParams.cfo = 2e6;                                   % Carrier frequency offset in Hertz
simParams.sco = 2;                                     % Sampling clock offset in parts
                                                       % per million

simParams.phNoiseLevel = Low ;                          % Phase noise level provided as
                                                       % 'Low', 'Medium', or 'High'
simParams.EsNodB = 25;                                 % Energy per symbol to noise ratio
                                                       % in decibels
```

This table defines the phase noise mask (dBc/Hz) used to generate the phase noise applied to the transmitted signal. These noise masks are taken from ETSI TR 102 376-1 Section 4.3.2.1.3 Figure 12 [2] on page 4-0 .

Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz
Low	-73	-85	-93	-97	-110
Medium	-50	-60	-83	-105	-115
High	-25	-50	-73	-93	-103

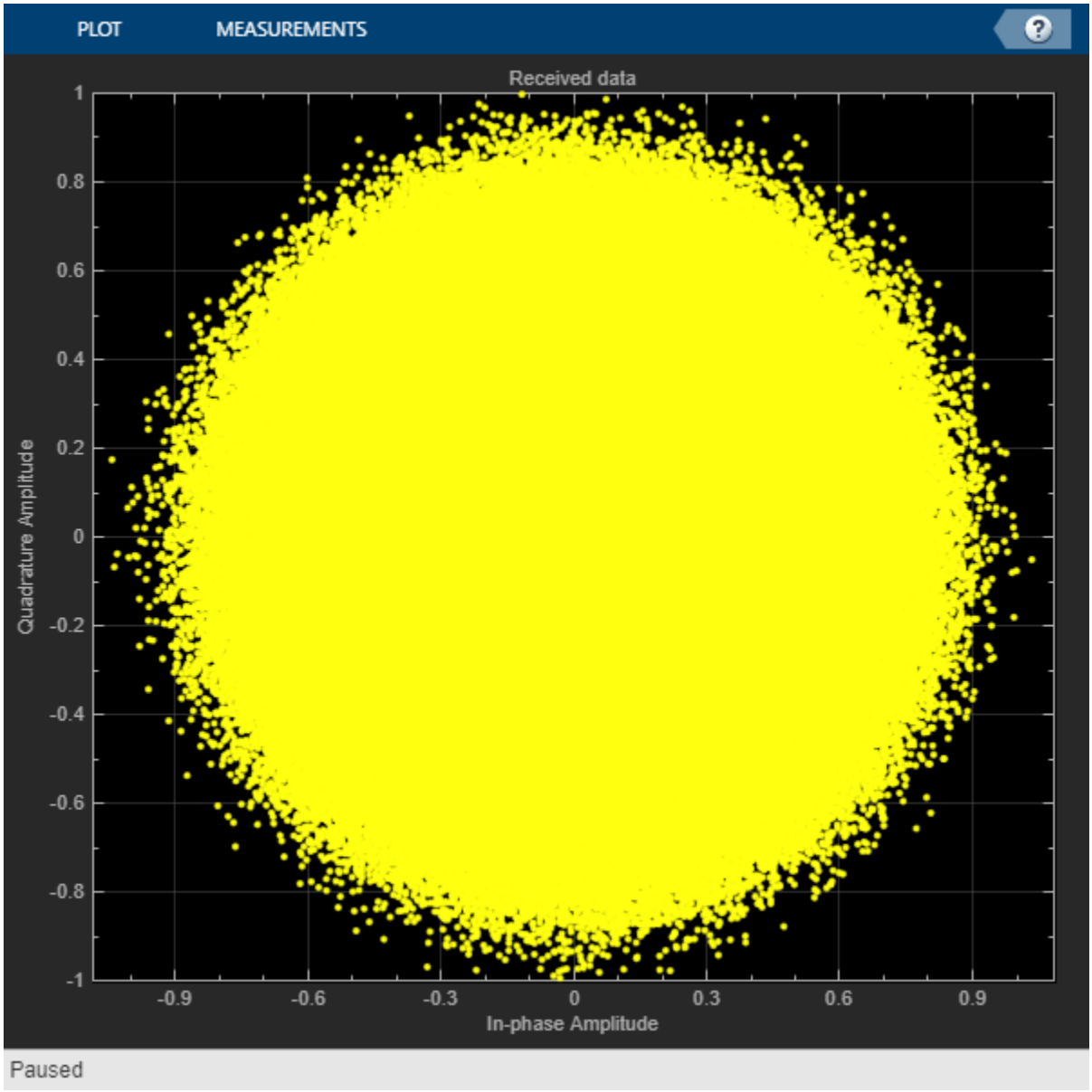
Generate DVB-S2X Waveform Distorted with RF Impairments

To create a DVB-S2X waveform, use the `HelperDVBS2XRxInputGenerate` helper function with the `simParams` and `cfgDVBS2X` structures as inputs. The function returns the data signal, transmitted and received waveforms, physical layer configuration parameters as a structure, and a receiver processing structure. The received waveform is impaired with carrier frequency, timing phase offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

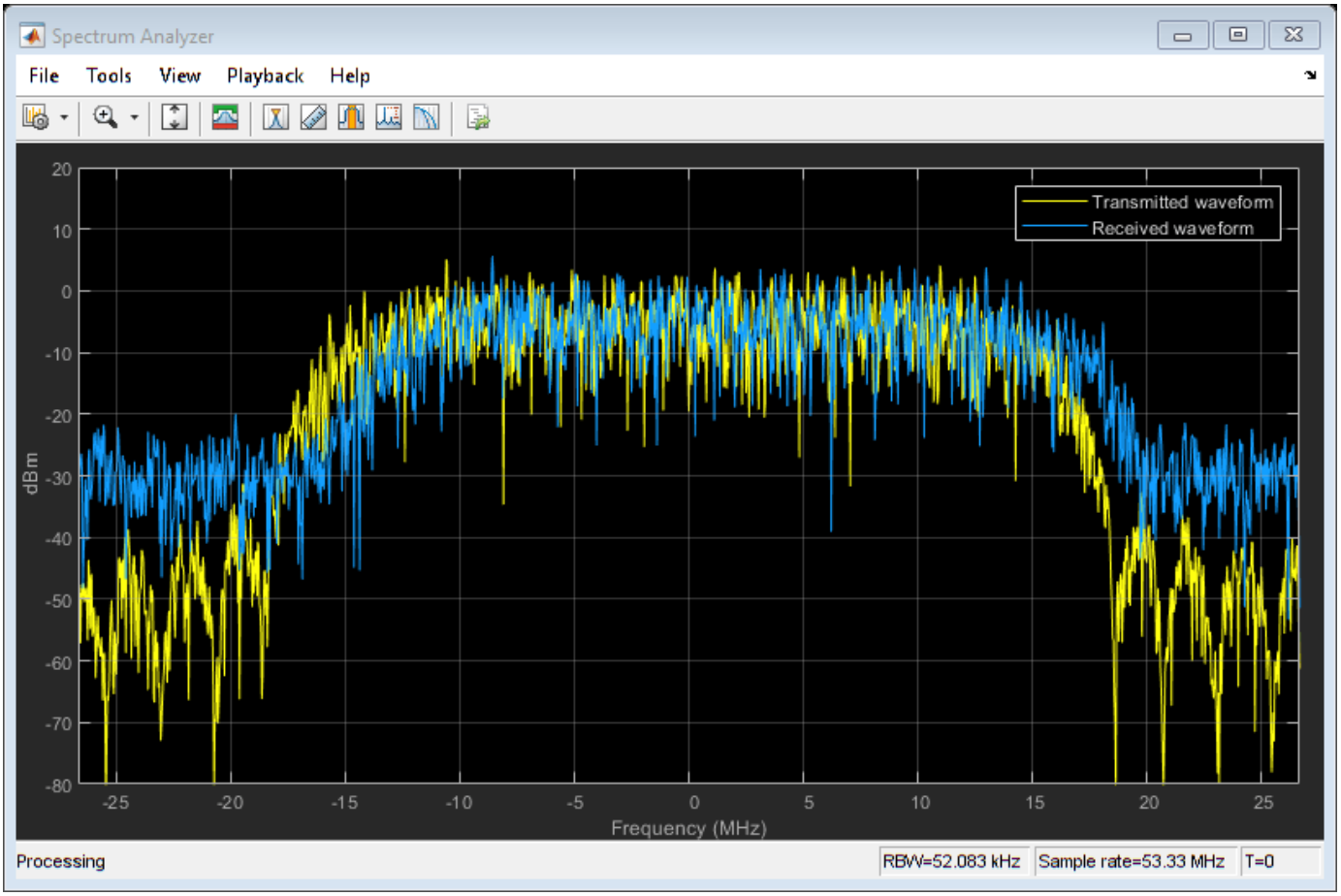
```
[data,txOut,rxIn,phyConfig,rxParams] = HelperDVBS2XRxInputGenerate(cfgDVBS2X,simParams);
disp(phyConfig)
```

```
              FECFrame: "normal"
      ModulationScheme: "64APSK"
    LDPCCodeIdentifier: "7/9"
```

```
% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:length(txOut)))
```




```
% Transmitted and received signal spectrum visualization
Rsymb = simParams.chanBW/(1 + cfgDVBS2X.RolloffFactor);
Fsamp = Rsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'ShowLegend',true);
specAn([txOut,rxIn(1:length(txOut))]);
```



Configure Receiver Parameters

At the receiver, symbol timing synchronization is performed on the received data and is then followed by frame synchronization. The receiver algorithms include coarse and fine frequency impairment correction algorithms. The carrier frequency estimation algorithm can track carrier frequency offsets up to 20% of the input symbol rate. The coarse frequency estimation, implemented as a frequency locked loop (FLL), reduces the frequency offset to a level that the fine frequency estimator can track. The preferred loop bandwidth for symbol timing and coarse frequency compensation depends on the E_s/N_o setting.

A block of 36 pilots is repeated every 1476 symbols. The coarse frequency error estimation uses 34 of the 36 pilot symbols. The ratio of used pilots per block (34) and pilot periodicity(1476) is 0.023. Using the 0.023 value as a scaling factor for the coarse frequency synchronizer loop bandwidth is preferred.

When you decrease the E_s/N_o, reduce the loop bandwidth to filter out more noise during acquisition. The number of frames required for the symbol synchronizer and coarse FLL to converge depends on the loop bandwidth setting.

The frame synchronization uses the PL header. Because the carrier synchronization is data-aided, the frame synchronization must detect the start of frame accurately. E_s/N_o plays a crucial role in determining the accuracy of the frame synchronization. When QPSK modulated frames are being recovered at E_s/N_o values below 3 dB, the frame synchronization must be performed on multiple frames for accurate detection.

The fine frequency estimation can track carrier frequency offsets up to 4% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency error less than 0.02% of the input symbol rate.

These settings are assigned in the `rxParams` structure for synchronization processing. For details on how to set these parameters for low E_s/N_o values, see the Further Exploration on page 4-0 section.

```
rxParams.carrSyncLoopBW = 1e-2*0.023;       % Coarse frequency estimator loop bandwidth
                                            % normalized by symbol rate
rxParams.symbSyncLoopBW = 8e-3;             % Symbol timing synchronizer loop bandwidth
                                            % normalized by symbol rate
rxParams.symbSyncLock  = 8;                 % Number of frames required for symbol timing
                                            % error convergence
rxParams.frameSyncLock = 1;                 % Number of frames required for frame
                                            % synchronization
rxParams.coarseFreqLock = 5;                % Number of frames required for coarse
                                            % frequency acquisition
rxParams.fineFreqLock = 4;                  % Number of frames required for fine
                                            % frequency estimation

% Total frames taken for symbol timing and coarse frequency lock to happen
rxParams.initialTimeFreqSync = rxParams.symbSyncLock + rxParams.frameSyncLock + ...
    rxParams.coarseFreqLock;
% Total frames used for overall synchronization
rxParams.totalSyncFrames = rxParams.initialTimeFreqSync + rxParams.fineFreqLock;

% Create time frequency synchronization System object by using
% HelperDVBS2TimeFreqSynchronizer helper object
timeFreqSync = HelperDVBS2TimeFreqSynchronizer( ...
    'CarrSyncLoopBW',rxParams.carrSyncLoopBW, ...
    'SymbSyncLoopBW',rxParams.symbSyncLoopBW, ...
    'SamplesPerSymbol',simParams.sps, ...
    'DataFrameSize',rxParams.xFecFrameSize, ...
    'SymbSyncTransitFrames',rxParams.symbSyncLock, ...
    'FrameSyncAveragingFrames',rxParams.frameSyncLock);

% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

% Initialize data indexing variables
stIdx = 0;
dataSize = rxParams.inputFrameSize;
plFrameSize = rxParams.plFrameSize;
dataStInd = rxParams.totalSyncFrames + 1;
isLastFrame = false;
symSyncOutLen = zeros(rxParams.initialTimeFreqSync,1);
```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data and recover the input bit stream, the distorted DVB-S2X waveform samples are processed one frame at a time by following these steps.

1. Apply matched filtering, outputting at the rate of two samples per symbol.
2. Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate. The Gardner TED is not data-aided, so it is performed before carrier synchronization.

3. Apply frame synchronization to detect the start of frame and to identify the pilot positions.
4. Estimate and apply coarse frequency offset correction.
5. Estimate and apply fine frequency offset correction.
6. Estimate and compensate for residual carrier frequency and phase noise.
7. Decode the PL header and compute the transmission parameters.
8. Demodulate and decode the PL frames.
9. Perform CRC check on the BB header, if the check passes, recover the header parameters.
10. Regenerate the input stream of data or packets from BB frames.

```
while stIdx < length(rxIn)

    % Use one DVB-S2X PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are considered.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % After coarse frequency offset loop is converged, the FLL works with a
    % reduced loop bandwidth.
    if rxParams.frameCount < rxParams.initialTimeFreqSync
        coarseFreqLock = false;
    else
        coarseFreqLock = true;
    end

    % Retrieve the last frame samples.
    if isLastFrame
        resSymb = plFrameSize - length(rxParams.cfBuffer);
        resSampCnt = resSymb*rxParams.sps - length(rxData);
        if resSampCnt >= 0    % Inadequate number of samples to fill last frame
            syncIn = [rxData; zeros(resSampCnt, 1)];
        else                  % Excess samples are available to fill last frame
            syncIn = rxData(1:resSymb*rxParams.sps);
        end
    else
        syncIn = rxData;
    end

    % Apply matched filtering, symbol timing synchronization, frame
    % synchronization, and coarse frequency offset compensation.
    [coarseFreqSyncOut,syncIndex,phEst] = timeFreqSync(syncIn,coarseFreqLock);
    if rxParams.frameCount <= rxParams.initialTimeFreqSync
        symSyncOutLen(rxParams.frameCount) = length(coarseFreqSyncOut);
        if any(abs(diff(symSyncOutLen(1:rxParams.frameCount))) > 5)
            error(['Symbol timing synchronization failed. The loop will not ' ...
                'converge. No frame will be recovered. Update the symbSyncLoopBW ' ...
                'parameter according to the EsNo setting for proper loop convergence.']);
        end
    end

    rxParams.syncIndex = syncIndex;
```

```
% The PL frame start index lies somewhere in the middle of the chunk being processed.
% From fine frequency estimation onwards, the processing happens as a PL frame.
% A buffer is used to store symbols required to fill one PL frame.
if isLastFrame
    fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut];
else
    fineFreqIn = [rxParams.cfBuffer; coarseFreqSyncOut(1:rxParams.syncIndex-1)];
end

% Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
% helper function.
% Add 1 to the conditional check because the buffer used to get one PL
% frame introduces a delay of one to the loop count.
if (rxParams.frameCount > rxParams.initialTimeFreqSync + 1) && ...
        (rxParams.frameCount <= rxParams.totalSyncFrames + 1)
    rxParams.fineFreqCorrVal = HelperDVBS2FineFreqEst( ...
        fineFreqIn(rxParams.pilotInd),rxParams.numPilotBlks, ...
        rxParams.refPilots,rxParams.fineFreqCorrVal);
end
if rxParams.frameCount >= rxParams.totalSyncFrames + 1
    fineFreqLock = true;
else
    fineFreqLock = false;
end


if fineFreqLock
    % Normalize the frequency estimate by the input symbol rate
    % freqEst = angle(R)/(pi*(N+1)) where N (18) is the number of elements
    % used to compute the mean of auto correlation (R) in
    % HelperDVBS2FineFreqEst.
    freqEst = angle(rxParams.fineFreqCorrVal)/(pi*(19));

    % Generate the symbol indices using frameCount and plFrameSize.
    % Subtract 2 from the rxParams.frameCount because the buffer used to get one
    % PL frame introduces a delay of one to the count.
    ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
    phErr = exp(-1j*2*pi*freqEst*ind);
    fineFreqOut = fineFreqIn.*phErr(:);

    % Estimate the phase error estimation by using the HelperDVBS2PhaseEst
    % helper function.
    [phEstRes,rxParams.prevPhaseEst] = HelperDVBS2PhaseEst( ...
        fineFreqOut(rxParams.pilotInd),rxParams.refPilots,rxParams.prevPhaseEst);

    % Compensate for the residual frequency and phase offset by using
    % the
    % HelperDVBS2PhaseCompensate helper function.
    % Use two frames for initial phase error estimation. Starting with the
    % second frame, use the phase error estimates from the previous frame and
    % the current frame in compensation.
    % Add 3 to the frame count comparison to account for delays: One
    % frame due to rxParams.cfBuffer delay and two frames used for phase
    % error estimate.
    if rxParams.frameCount >= rxParams.totalSyncFrames + 3
        phaseCompOut = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
            rxParams.pilotEst,rxParams.pilotInd,phEstRes(2));
    end
```

```
        rxParams.ffBuffer = fineFreqOut;
        rxParams.pilotEst = phEstRes;

        % The phase compensation on the data portion is performed by
        % interpolating the phase estimates computed on consecutive pilot
        % blocks. The second phase estimate is not available for the data
        % portion after the last pilot block in the last frame. Therefore,
        % the slope of phase estimates computed on all pilot blocks in the
        % last frame is extrapolated and used to compensate for the phase
        % error on the final data portion.
        if isLastFrame
            pilotBlkLen = 36;    % Symbols
            pilotBlkFreq = 1476; % Symbols
            avgSlope = mean(diff(phEstRes(2:end)));
            chunkLen = rxParams.plFrameSize - rxParams.pilotInd(end) + ...
                rxParams.pilotInd(pilotBlkLen);
            estEndPh = phEstRes(end) + avgSlope*chunkLen/pilotBlkFreq;
            phaseCompOut1 = HelperDVBS2PhaseCompensate(rxParams.ffBuffer, ...
                rxParams.pilotEst,rxParams.pilotInd,estEndPh);
        end
    end

    % Recover the input bit stream.
    if rxParams.frameCount >= rxParams.totalSyncFrames + 3
        isValid = true;
        if isLastFrame
            syncOut = [phaseCompOut;phaseCompOut1];
        else
            syncOut = phaseCompOut;
        end
    else
        isValid = false;
        syncOut = [];
    end

    % Update the buffers and counters.
    rxParams.cfBuffer = coarseFreqSyncOut(rxParams.syncIndex:end);
    rxParams.syncIndex = syncIndex;
    rxParams.frameCount = rxParams.frameCount + 1;

    if isValid  % Data valid signal

        % Decode the PL header by using the HelperDVBS2XPLHeaderRecover helper
        % function. Start of frame (SOF) is 26 symbols which are discarded
        % before header decoding. They are only required for frame
        % synchronization.
        rxPLSCode = syncOut(1:90); % First 90 symbols of frame is PL header
        [plsDecCode,phyParams] = HelperDVBS2XPLHeaderRecover(rxPLSCode,rxParams.s2xStatus);
        % Validate the decoded PL header.
        if plsDecCode ~= cfgDVBS2X.PLSDecimalCode
            fprintf('%s\n','PL header decoding failed')
        else % Demodulation and decoding
            for frameCnt = 1:length(syncOut)/rxParams.plFrameSize
                rxFrame = syncOut((frameCnt-1)*plFrameSize+1:frameCnt*plFrameSize);
                % Estimate noise variance by using
                % HelperDVBS2NoiseVarEstimate helper function.
                nVar = HelperDVBS2NoiseVarEstimate(rxFrame,rxParams.pilotInd,...
```

```
                    rxParams.refPilots,rxParams.normFlag);
                % The data begins at symbol 91 (after the header symbols).
                rxDataFrame = rxFrame(91:end);
                % Recover the BB frame by using HelperDVBS2XBBFrameRecover
                % helper function.
                rxBBFrame = HelperDVBS2XBBFrameRecover(rxDataFrame,phyParams,...
                    rxParams.plScramblingIndex,rxParams.numPilotBlks,nVar,false);
                % Recover the input bit stream by using
                % HelperDVBS2StreamRecover helper function.
                if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
                    [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame);
                    if ~isFrameLost && length(decBits) ~= dataSize
                        isFrameLost = true;
                    end
                else
                    [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame);
                    if ~isFrameLost && length(decBits) ~= dataSize
                        isFrameLost = true;
                        pktCRC = zeros(0,1,'logical');
                    end
                    % Compute the packet error rate for TS or GS packetized
                    % mode.
                    pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
                    pktsRec = pktsRec + numel(pktCRC);
                end
                if ~isFrameLost
                    ts = sprintf('%s','BB header decoding passed.');
                else
                    ts = sprintf('%s','BB header decoding failed.');
                end
                % Compute the number of frames lost. CRC failure of
                % baseband header is considered a frame loss.
                numFramesLost = isFrameLost + numFramesLost;
                fprintf('%s(Number of frames lost = %1d)\n',ts,numFramesLost)
                % Compute the bits in error.
                bitInd = (dataStInd-1)*dataSize+1:dataStInd*dataSize;
                if isLastFrame && ~isFrameLost
                    bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
                else
                    if ~isFrameLost
                        bitsErr = bitsErr + sum(data(bitInd) ~= decBits);
                    end
                end
                dataStInd = dataStInd + 1;
            end
        end
    end
    stIdx = endIdx;
end
```

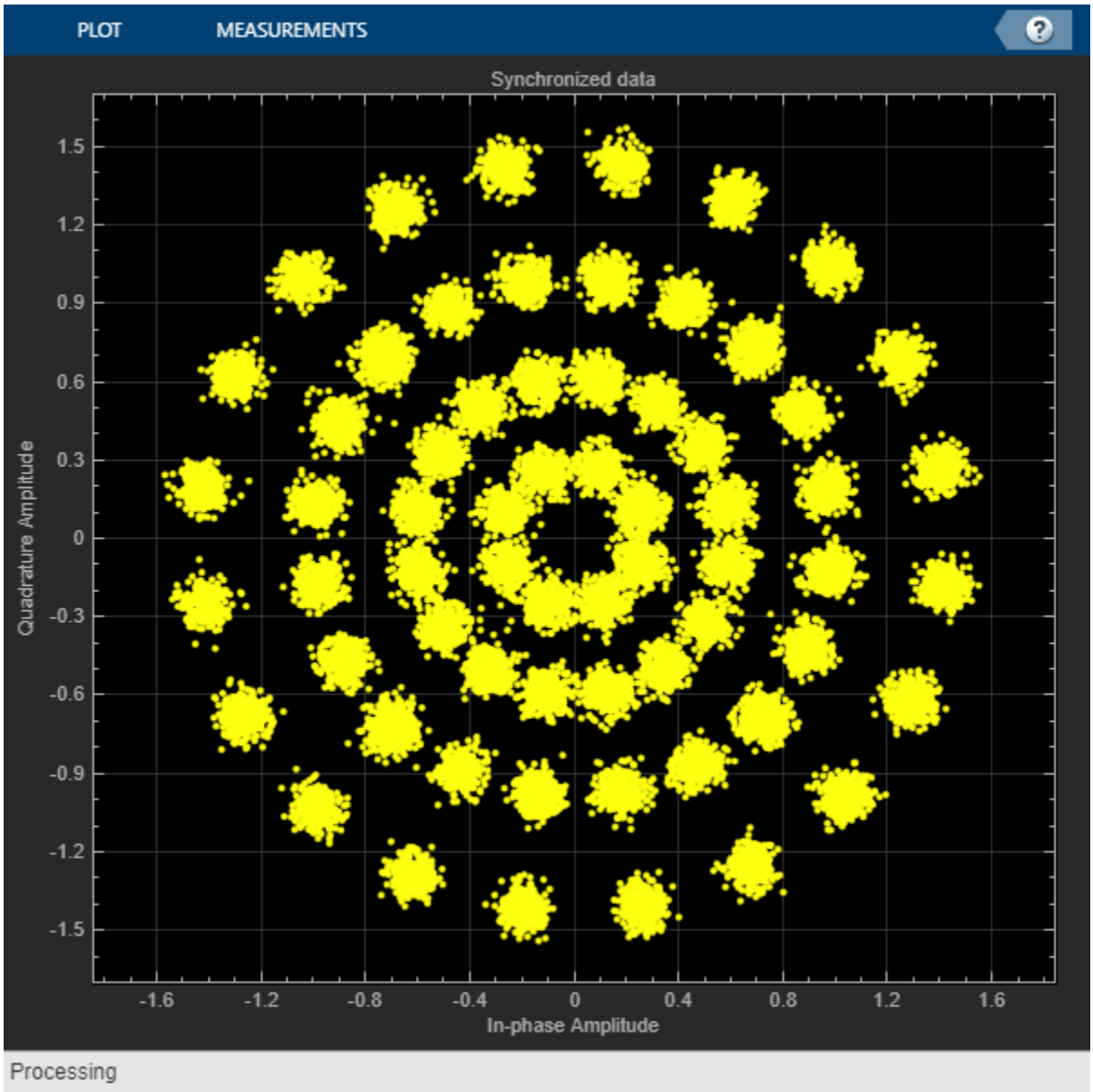
```
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
```

```
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
```

Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```
% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title','Synchronized data', ...
    'XLimits',[-1.7 1.7],'YLimits',[-1.7 1.7], ...
    'ShowReferenceConstellation',false);
syncConst(syncOut)
```



```
% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames-rxParams.totalSyncFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        ber = bitsErr/((dataStInd-rxParams.totalSyncFrames)*dataSize);
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
    else
        if strcmpi(cfgDVBS2X.StreamFormat,'TS')
            pktLen = 1504;
        else
            pktLen = cfgDVBS2X.UPL;      % UP length including sync byte
        end
        ber = bitsErr/(pktsRec*pktLen);
        per = pktsErr/pktsRec;
        fprintf('PER: %1.2e\n',per)
        fprintf('BER: %1.2e\n',ber)
    end
end
```

```
PER: 0.00e+00
```

```
BER: 0.00e+00
```

Further Exploration

For BER simulations in AWGN assuming perfect synchronization, use the `HelperDVBS2XBitRecover` helper function to evaluate the receiver performance. See the examples provided in the M-help section of the `HelperDVBS2XBitRecover` helper function. For details on how to configure the synchronization parameters of the `rxParams` for other `cfgDVBS2X` and `simParams` settings, see the 'Further Exploration section' of “End-to-End DVB-S2 Simulation with RF Impairments and Corrections” on page 4-36 on how to configure the synchronization parameters of `rxParams` for other `cfgDVBS2X` and `simParams` settings. For higher modulation schemes like 64 APSK and above, this table shows the typical number of frames required for convergence of the symbol timing loop.

Modulation scheme	Number of frames
64 APSK	30 - 35
128 APSK	35 – 40
256 APSK	38 - 43

Appendix

The example uses these helper functions:

- HelperDVBS2XRxInputGenerate.m: Generate DVB-S2X waveform samples distorted with RF impairments and structure of parameters for receiver processing

- HelperDVBS2PhaseNoise.m: Generate phase noise samples for different DVB-S2X phase noise masks and apply it to the input signal
- HelperDVBS2TimeFreqSynchronizer.m: Perform matched filtering, symbol timing synchronization, frame synchronization, and coarse frequency estimation and correction
- HelperDVBS2FrameSync.m: Perform frame synchronization and detect the start of frame
- HelperDVBS2FineFreqEst.m: Estimate fine frequency offset
- HelperDVBS2PhaseEst.m: Estimate carrier phase offset
- HelperDVBS2PhaseCompensate.m: Perform carrier phase compensation
- HelperDVBS2XPLHeaderRecover.m: Demodulate and decode the PL header to recover transmission parameters
- HelperDVBS2NoiseVarEstimate.m: Estimate noise variance of received data
- HelperDVBS2XBBFrameRecover.m: Perform PL de-scrambling, demodulation, decoding and recover BB frame from PL frame
- HelperDVBS2XDemapper.m: Perform soft demodulation for all DVB-S2X based modulation schemes
- HelperDVBS2XLDPCDecode.m: Perform LDPC decoding for all DVB-S2X based LDPC frame formats and code rates
- HelperDVBS2XBCHDecode.m: Perform BCH decoding for all DVB-S2X based frame formats and code rates
- HelperDVBS2StreamRecover.m: Perform CRC check of BB header and recover input stream from BB frame based on header parameters
- HelperDVBS2XBitRecover.m: Perform PL header demodulation and decoding, PL de-scrambling, demodulation, decoding and recover BB frame. Perform CRC check of BB header and recover the input stream from BB frame.

Bibliography

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)*.
2. ETSI Standard TR 102 376-2 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: S2 extensions (DVB-S2X)*.
3. ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
4. Mengali, Umberto, and Aldo N.D'Andrea. *Synchronization Techniques for Digital Receivers*. New York: Plenum Press,1997.
5. E. Casini, R. De Gaudenzi, and Alberto Ginesi. "DVB-S2 modem algorithms design and performance over typical satellite channels." *International Journal of Satellite Communications and Networking* 22, no. 3 (2004): 281-318.

6 Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

See Also

Objects
`dvbs2xWaveformGenerator` | `dvbs2WaveformGenerator`

Related Examples

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames” on page 4-68
- “End-to-End DVB-S2 Simulation with RF Impairments and Corrections” on page 4-36

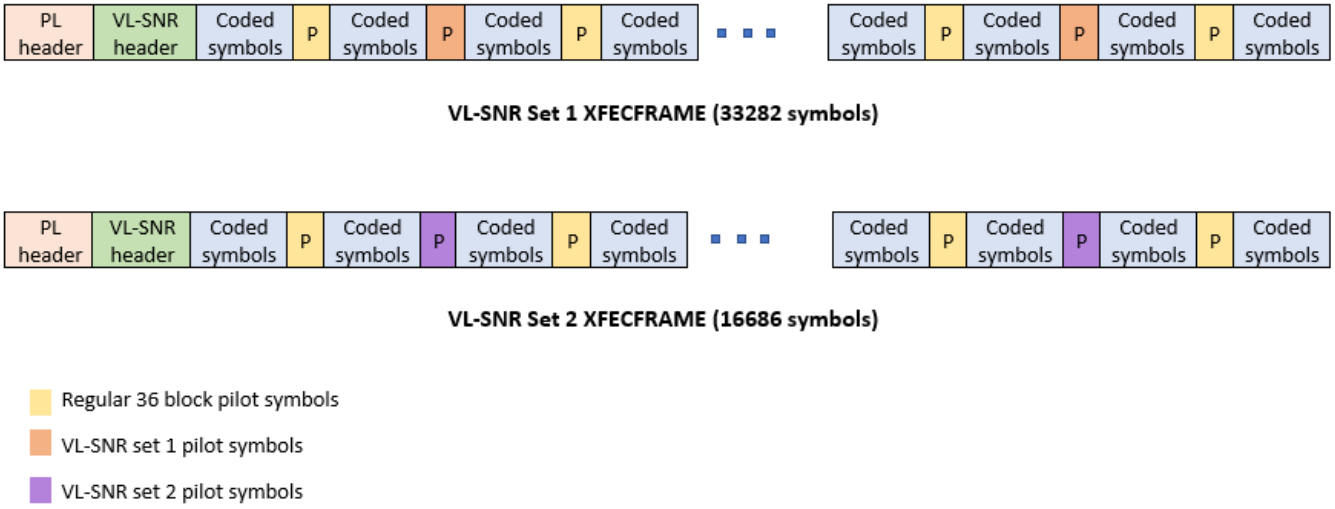
End-to-End DVB-S2X Simulation with RF Impairments and Corrections for VL-SNR Frames

This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation extended (DVB-S2X) link that has constant coding and modulation for very low signal to noise ratio (VL-SNR) frames. The example describes the symbol timing, frame and carrier synchronization strategies in detail emphasizing on how to estimate the RF front-end impairments under severe noise conditions. The single stream signal adds RF front-end impairments and then passes the waveform through an additive white Gaussian noise (AWGN) channel.

Introduction

An increasing number of DVB-S2X terminals are being used on trains, buses, boats, and airplanes. Many applications, such as sensor networks, remote infrastructure monitoring, and emergency services, require ubiquitous coverage, and low data rates. Support for these applications was therefore included in the design of DVB-S2X, introducing VL-SNR configurations to increase the operating range to cover current and emerging applications that can benefit from operation at very low SNR. The DVB-S2X standard added nine additional modulation and coding schemes (MODCODs) in the QPSK and BPSK range. These MODCODs enable the satellite networks to deal with heavy atmospheric fading and to enable use of smaller antennas for applications in motion (land, sea, and air).

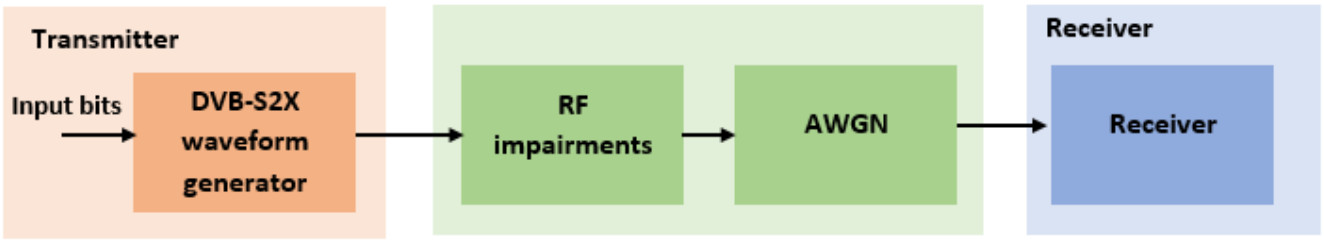
These figures show the two formats used for VL-SNR frames.



This example designs the synchronization aspects of a DVB-S2X receiver used for VL-SNR applications. The example supports all the nine MODCODs defined by the standard.

ETSI EN 302 307-2 Section 6 Table 20a, Table 20b, and Table 20c [1] on page 4-0 summarizes the Quasi-Error-Free (QEF) performance requirement over an AWGN channel for different modulation schemes and code rates. The operating E_s/N_o range for VL-SNR applications is considered from -2 dB to -10 dB. Because the operating E_s/N_o range is low, the carrier, frame, and symbol timing synchronization strategies are challenging design problems.

This diagram summarizes the example workflow.



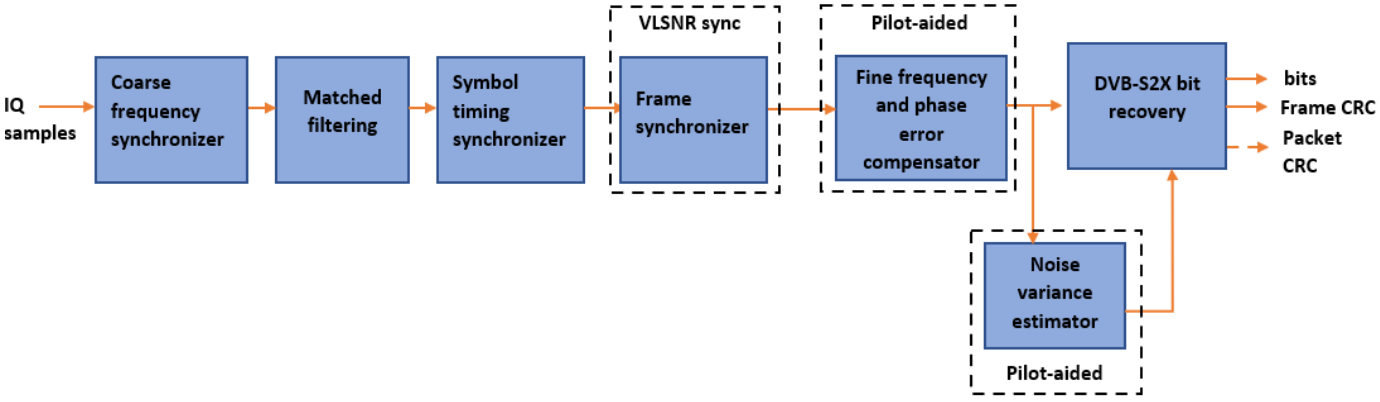
Main Processing Loop

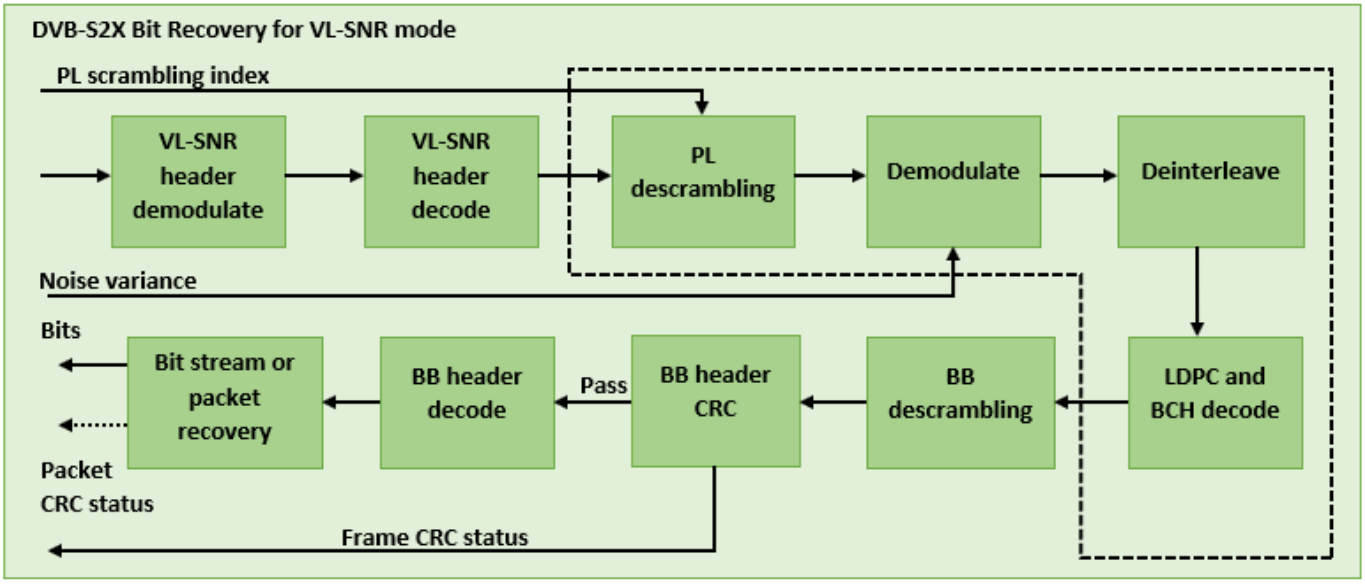
The example processes 20 physical layer (PL) frames of data with the E_s/N_o set to 5 dB, and then computes the BER and PER. Carrier frequency offset, frequency drift, symbol timing offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal. ETSI EN 302 307-2 Section 4.4.4 describes the typical RF impairment ranges used under VL-SNR conditions.

To extract PL frames, the receiver processes the distorted waveform through various timing and carrier recovery strategies. The fine frequency and carrier phase recovery algorithms are pilot-aided. To decode the data frames, the physical layer transmission parameters, such as VL-SNR set type, MODCOD, and FEC frame type are recovered from the VL-SNR header. To regenerate the input bit stream, the baseband (BB) header is decoded.

Because the DVB-S2X standard supports packetized and continuous modes of transmission, the BB frame can be either a concatenation of user packets or a stream of bits. The BB header is recovered to determine the mode of transmission. If the BB frame is a concatenation of user packets, the packet cyclic redundancy check (CRC) status of each packet is returned along with the decoded bits, and then the PER and BER are measured.

These block diagrams show the synchronization and input bit recovery workflows.





Download DVB-S2X LDPC Parity Matrices Data Set

This example loads a MAT-file with DVB-S2X LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, use these commands to download and unzip the MAT-file.

```
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
```

DVB-S2X Configuration

Specify the `cfgDVBS2X` structure to define DVB-S2X transmission configuration parameters. `PLSDecimalCode` 129 and 131 are the only supported formats because they are used for generating VL-SNR frames.

```
cfgDVBS2X.StreamFormat = "TS";
cfgDVBS2X.PLSDecimalCode = 129;
cfgDVBS2X.CanonicalMODCODName = "QPSK 2/9";
cfgDVBS2X.DFL = 14128;
cfgDVBS2X.RolloffFactor = 0.35;
cfgDVBS2X.SamplesPerSymbol = 2
```

```
cfgDVBS2X = struct with fields:
           StreamFormat: "TS"
          PLSDecimalCode: 129
    CanonicalMODCODName: "QPSK 2/9"
                    DFL: 14128
           RolloffFactor: 0.3500
       SamplesPerSymbol: 2
```

Simulation Parameters

The DVB-S2X standard supports flexible channel bandwidths. Use a typical channel bandwidth such as 36 MHz. The channel bandwidth can be varied from 10MHz to 72MHz. The coarse frequency synchronization algorithm implemented in this example can track carrier frequency offsets up to 20% of the input symbol rate. The symbol rate is calculated as B/(1+R), where B is the channel bandwidth, and R is the transmit filter roll-off factor. The algorithms implemented in this example can correct the sampling clock offset up to 10 ppm.

This table defines the phase noise mask (dBc/Hz) used to generate the phase noise that is applied to the transmitted signal. These noise masks are specified in ETSI TR 102 376-1 Section 4.3 [2] on page 4-0 . The peak doppler and frequency drift supported are specified in ETSI TR 102 376-1 Section 4.4.4.

Frequency	100 Hz	1 KHz	10 KHz	100 KHz	1 MHz
Level-1	-61.9	-78.7	-88.7	-94.8	-105.7
Level-2	-58.1	-68.5	-78.3	-88.3	-88.3
Level-3	-45	-65	-75	-85	-102

```
simParams.sps = cfgDVBS2X.SamplesPerSymbol;    % Samples per symbol
simParams.numFrames = 20;                      % Number of frames to be processed
simParams.chanBW = 36e6;                       % Channel bandwidth in Hertz
simParams.cfo = 3e6;                           % Carrier frequency offset in Hertz due
                                               % to oscillator instabilities
simParams.dopplerRate = 3e3;                   % Doppler rate in Hertz/sec
simParams.peakDoppler = 20e3;                  % Peak doppler shift due to receiver motion
simParams.sco = 10;                            % Sampling clock offset in parts per
                                               % million
simParams.phNoiseLevel = "Level-1";            % Phase noise level
simParams.EsNodB = 5;                          % Energy per symbol to noise ratio
```

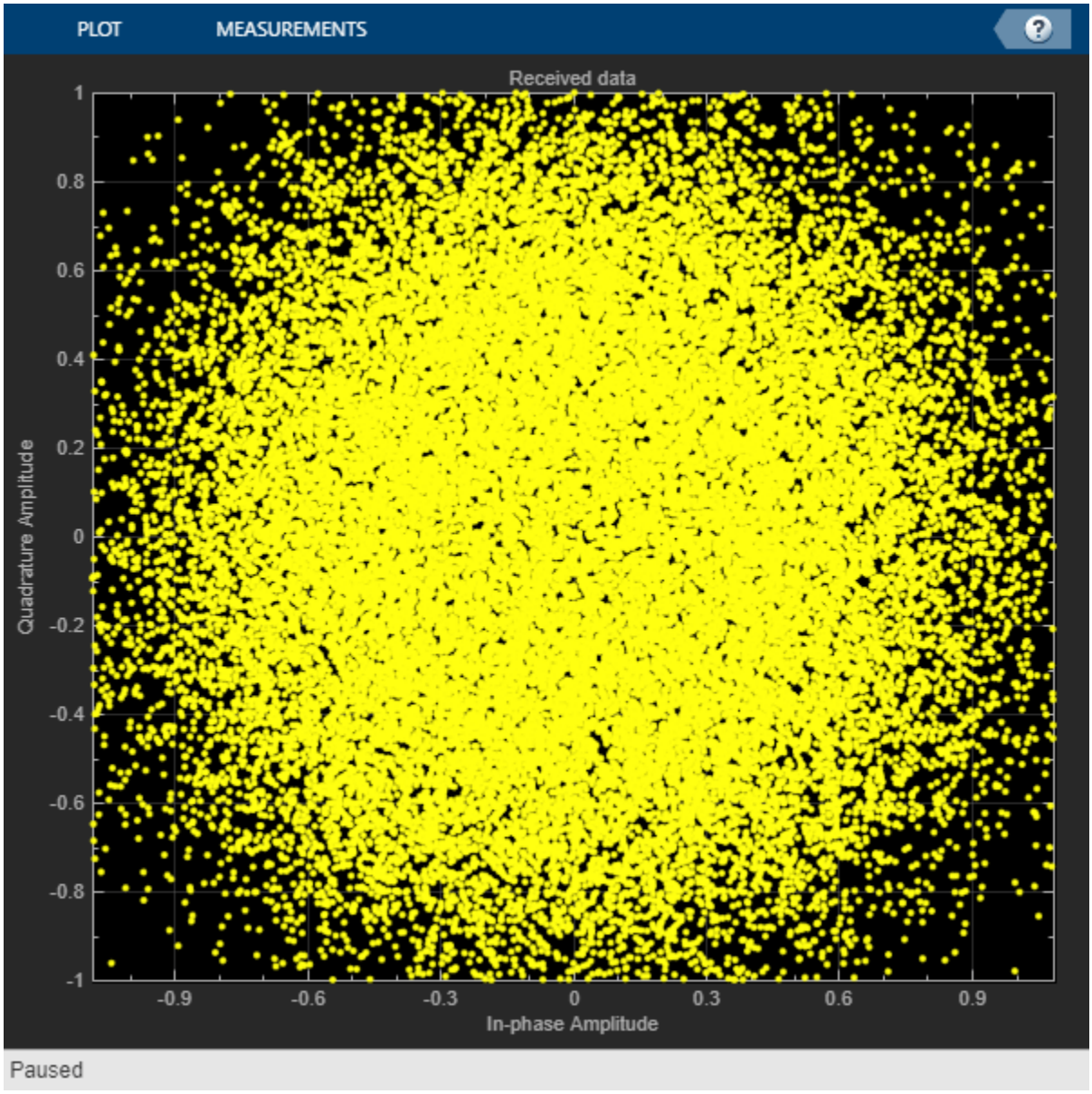
Generate DVB-S2X VL-SNR Waveform Distorted with RF Impairments

To create a DVB-S2X waveform, use the `HelperDVBS2XVLSNRRxInputGenerate` helper function with the `simParams` and `cfgDVBS2X` structures as inputs. The function returns the data signal, transmitted, and received waveforms, physical layer configuration parameters as a structure, and a receiver processing structure. The received waveform is impaired with carrier frequency, frequency drift, symbol timing, sampling clock offsets, and phase noise and then passed through an AWGN channel. The receiver processing parameters structure, `rxParams`, includes the reference pilot fields, pilot indices, counters, and buffers. Plot the constellation of the received symbols and the spectrum of the transmitted and received waveforms.

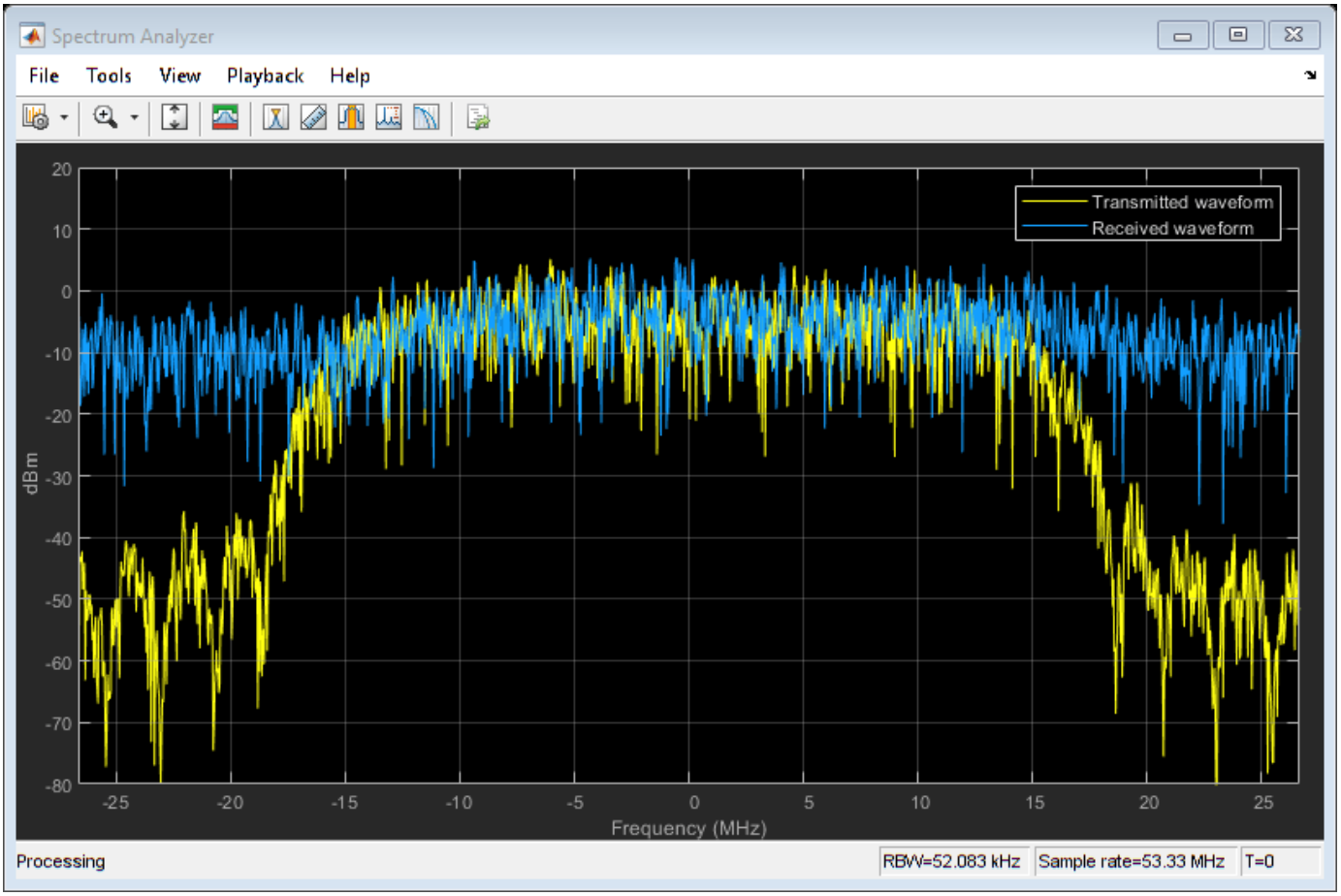
```
[data,txOut,rxIn,phyConfig,rxParams] = ...
    HelperDVBS2XVLSNRRxInputGenerate(cfgDVBS2X,simParams);
disp(phyConfig)
```

```
              FECFrame: "normal"
      ModulationScheme: "QPSK"
    LDPCCodeIdentifier: "2/9"
```

```
% Received signal constellation plot
rxConst = comm.ConstellationDiagram('Title','Received data', ...
    'XLimits',[-1 1],'YLimits',[-1 1], ...
    'ShowReferenceConstellation',false, ...
    'SamplesPerSymbol',simParams.sps);
rxConst(rxIn(1:rxParams.plFrameSize*simParams.sps))
```




```
% Transmitted and received signal spectrum visualization
Rsymb = simParams.chanBW/(1 + cfgDVBS2X.RolloffFactor);
Fsamp = Rsymb*simParams.sps;
specAn = dsp.SpectrumAnalyzer('SampleRate',Fsamp, ...
    'ChannelNames',{'Transmitted waveform','Received waveform'}, ...
    'ShowLegend',true);
specAn([txOut,rxIn(1:length(txOut))]);
```



Configure Receiver Parameters

At the receiver, coarse frequency synchronization is performed on the received data and is then followed by matched filtering and symbol timing synchronization. The coarse frequency and symbol timing estimators are non-data aided. The coarse frequency estimation algorithm can track carrier frequency offsets up to 20% of the input symbol rate. The coarse frequency estimation, implemented as a first order frequency locked loop (FLL), reduces the frequency offset to a level that the fine frequency estimator can track. The preferred loop bandwidth for symbol timing and coarse frequency compensation depends on the E_s/N_o setting.

When you decrease the E_s/N_o, reduce the loop bandwidth to filter out more noise during acquisition. The number of frames required for the symbol synchronizer and coarse FLL to converge depends on the loop bandwidth setting.

Symbol timing synchronization is followed by frame synchronization and MODCOD format detection. The frame synchronization uses the VL-SNR header. Because the fine frequency and carrier phase synchronization are data-aided, the frame synchronization must detect the start of frame accurately.

The fine frequency estimation can track carrier frequency offsets up to 3.5% of the input symbol rate. The fine frequency estimation must process multiple pilot blocks for the residual carrier frequency offset to be reduced to levels acceptable for the phase estimation algorithm. The phase estimation algorithm can handle residual carrier frequency error less than 0.05% of the input symbol rate.

These settings are assigned in the `rxParams` structure for synchronization processing. For details on how to set these parameters for low E_s/N_o values, see the Further Exploration on page 4-0 section.

```
rxParams.carrSyncLoopBW = 1e-4;          % Coarse frequency estimator loop bandwidth
                                         % normalized by symbol rate
rxParams.symbSyncLoopBW = 1e-4;          % Symbol timing synchronizer loop bandwidth
                                         % normalized by symbol rate
rxParams.initialTimeFreqSync = 5;        % Number of frames required for coarse frequency
                                         % and symbol timing error convergence
rxParams.fineFreqLock = 4;               % Number of frames required for fine
                                         % frequency estimation
rxParams.NeedSmoothening = false;        % Smoothen the phase estimate

% Total frames used for overall synchronization
rxParams.totalSyncFrames = rxParams.initialTimeFreqSync + rxParams.fineFreqLock;

% Create coarse frequency synchronization System object by using
% HelperDVBS2XVLSNRCoarseFreqSynchronizer helper object
freqSync = HelperDVBS2XVLSNRCoarseFreqSynchronizer('SamplesPerSymbol',simParams.sps, ...
    'NormalizedLoopBandwidth',rxParams.carrSyncLoopBW);

% Create symbol timing synchronization System object by using
% comm.SymbolSynchronizer object
symSync = comm.SymbolSynchronizer('TimingErrorDetector','Gardner (non-data-aided)', ...
    'NormalizedLoopBandwidth',rxParams.symbSyncLoopBW);

% Create matched filter System object by using
% comm.RaisedCosineReceiveFilter object
if simParams.sps == 2
    decFac = 1;
else
    decFac = simParams.sps/(simParams.sps/2);
end
rxFilter = comm.RaisedCosineReceiveFilter( ...
    'RolloffFactor',0.35, ...
     'InputSamplesPerSymbol',simParams.sps,'DecimationFactor',decFac);
b = rxFilter.coeffs;
rxFilter.Gain = sum(b.Numerator);

% Initialize error computing parameters
[numFramesLost,pktsErr,bitsErr,pktsRec] = deal(0);

% Initialize data indexing variables
stIdx = 0;
dataSize = rxParams.inputFrameSize;
plFrameSize = rxParams.plFrameSize;
isLastFrame = false;
rxParams.fineFreqCorrVal = zeros(rxParams.fineFreqLock,1);
[formatIdx,formatIdxTemp] = deal(1);
vlsnrSyncStIdx = 93;
payloadStIdx = 899;
vlSNRFrameLen = plFrameSize - vlsnrSyncStIdx + 1;
```

Timing and Carrier Synchronization and Data Recovery

To synchronize the received data and recover the input bit stream, process the distorted DVB-S2X waveform samples one frame at a time by following these steps.

1. Apply coarse frequency synchronization using a balanced quadricorrelator frequency error detector (BQ-FED) in an FLL [5] on page 4-0 .
2. Apply matched filtering, outputting at the rate of two samples per symbol.
3. Apply symbol timing synchronization using the Gardner timing error detector with an output generated at the symbol rate.
4. Apply frame synchronization to detect the start of frame and MODCOD format to identify the pilot positions.
5. Estimate and apply fine frequency offset correction.
6. Estimate and compensate for residual carrier frequency and phase noise.
7. Demodulate and decode the VL-SNR frames.
8. Perform CRC check on the BB header, if the check passes, recover the header parameters.
9. Regenerate the input stream of data or packets from BB frames.

```
while stIdx < length(rxIn)

    % Use one DVB-S2X PL frame for each iteration.
    endIdx = stIdx + rxParams.plFrameSize*simParams.sps;

    % In the last iteration, all the remaining samples in the received
    % waveform are considered.
    isLastFrame = endIdx > length(rxIn);
    endIdx(isLastFrame) = length(rxIn);
    rxData = rxIn(stIdx+1:endIdx);

    % After coarse frequency offset loop is converged, the FLL works with
    % previous frequency estimate.
    if rxParams.frameCount < rxParams.initialTimeFreqSync
        coarseFreqLock = false;
    else
        coarseFreqLock = true;
    end

    % Retrieve the last frame samples.
    if isLastFrame
        resSampCnt = plFrameSize*rxParams.sps - length(rxData);
        % Inadequate number of samples to fill last frame
        syncIn = [rxData; zeros(resSampCnt,1)];
    else
        syncIn = rxData;
    end

    % Apply coarse frequency offset compensation.
    [coarseFreqSyncOut,phEst] = freqSync(syncIn,coarseFreqLock);

    % Perform matched filtering and downsample the signal to 2 samples per
    % symbol.
    filtOut = rxFilter(coarseFreqSyncOut);

    % Apply symbol timing synchronization.
    symSyncOut = symSync(filtOut);

    % Apply frame synchronization and identify the MODCOD format. VL-SNR
    % sync frame is detected. PL header preceding the VL-SNR header is
    % ignored.
```

```
if rxParams.frameCount > rxParams.initialTimeFreqSync && ~isLastFrame
    [~,rxParams.syncIndex,formatIdx] = ...
        HelperDVBS2XVLSNRFrameSync(symSyncOut,rxParams.SegLength);
    % MODCOD format identification failure verification
    formatFail = formatIdxTemp ~= rxParams.refFormat;
    if formatFail && ~isLastFrame
        % Update the counters, state variables, and buffers
        stIdx = endIdx;
        rxParams.frameCount = rxParams.frameCount + 1;
        formatIdxTemp = formatIdx;
        rxParams.cfBuffer = symSyncOut(rxParams.syncIndex:end);
        fprintf('%s\n','MODCOD format detection failed')
        continue;
    else
        fprintf('%s\n','MODCOD format detection passed')
        [setNum,phyParams] = getVLSNRParams(formatIdxTemp);
    end
end

% The PL frame start index lies somewhere in the middle of the data
% being processed. From fine frequency estimation onwards, the
% processing happens as a PL frame. A buffer is used to store symbols
% required to fill one PL frame. PL frame is considered from
% the start of VL-SNR header, precisely from the start of the 896 bit
% length Walsh Hadamard (WH) sequence. 90 represents the PL header
% length, and 2 accounts for the two zeros appended before the WH sequence.
fineFreqIn = [rxParams.cfBuffer;...
    symSyncOut(1:vlSNRFrameLen-length(rxParams.cfBuffer))];

% Estimate the fine frequency error by using the HelperDVBS2FineFreqEst
% helper function.
% Add 1 to the conditional check because the buffer used to get one PL
% frame introduces a delay of one to the loop count.
if (rxParams.frameCount > rxParams.initialTimeFreqSync + 1)
    % Extract the payload by removing header
    payload = fineFreqIn(payloadStIdx:end);

    % Get the correlation estimate from the regular 36 length pilot blocks.
    est1 = HelperDVBS2FineFreqEst( ...
        payload(rxParams.regPilotInd),rxParams.regNumPilotBlks, ...
        rxParams.regPilotSeq,rxParams.fineFreqState,36,rxParams.NumLags);

    % Get the correlation estimate from the VL-SNR extra pilot blocks.
    % vlSNRPilotBlk1Params contains the VL-SNR type 1 pilot block
    % length and number of blocks in one frame.
    Lp = rxParams.vlSNRPilotBlk1Params(1);
    est2 = HelperDVBS2FineFreqEst( ...
        payload(rxParams.vlSNRPilotInd1),rxParams.vlSNRPilotBlk1Params(2), ...
        rxParams.vlSNRPilotSeq1,rxParams.fineFreqState,Lp,rxParams.NumLags);

    % vlSNRPilotBlk2Params contains the VL-SNR type 2 pilot block
    % length and number of blocks in one frame.
    Lp = rxParams.vlSNRPilotBlk2Params(1);
    est3 = HelperDVBS2FineFreqEst( ...
        payload(rxParams.vlSNRPilotInd2),rxParams.vlSNRPilotBlk2Params(2), ...
        rxParams.vlSNRPilotSeq2,rxParams.fineFreqState,Lp,rxParams.NumLags);

    estVal = est1 + est2 + est3;
```

```
        % Use the correlation values calculated over pilot fields spanning over
        % multiple frames to calculate the fine frequency error estimate.
        % The estimation uses a sliding window technique.
        rxParams.fineFreqCorrVal = [rxParams.fineFreqCorrVal(2:end);estVal];
    end

    if rxParams.frameCount >= rxParams.totalSyncFrames
        fineFreqLock = true;
    else
        fineFreqLock = false;
    end

    if fineFreqLock
        freqEst = angle(sum(rxParams.fineFreqCorrVal))/(pi*(rxParams.NumLags+1));
        ind = (rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)*plFrameSize-1;
        phErr = exp(-1j*2*pi*freqEst*ind).';
        % 92 accounts for the PL header (90), and 2 for zeros appended before the WH
        % sequence.
        fineFreqOut = fineFreqIn(1:vlSNRFrameLen).*phErr(vlsnrSyncStIdx:end);
        % 898 accounts for the 896 length WH sequence, and 2 zeros padded to the
        % header.
        rxPilots = fineFreqOut(rxParams.pilotInd+payloadStIdx-1);
        phErrEst = HelperDVBS2PhaseEst(rxPilots,rxParams.pilotSeq, ...
            rxParams.phErrState,rxParams.IsVLSNR,setNum,rxParams.Alpha);
        if rxParams.NeedSmoothening
            phErrEst =  smoothenEstimate(phErrEst);
        end
        phaseCompOut = HelperDVBS2PhaseCompensate(fineFreqOut(payloadStIdx:end), ...
            phErrEst,rxParams.pilotInd,setNum,rxParams.IsVLSNR);
    end

    % Recover the input bit stream.
    if rxParams.frameCount >= rxParams.totalSyncFrames
        isValid = true;
        syncOut = phaseCompOut;
    else
        isValid = false;
        syncOut = [];
    end

    % Update the buffers and counters.
    rxParams.cfBuffer = symSyncOut(rxParams.syncIndex:end);

    if isValid  % Data valid signal
        % Estimate noise variance by using
        % HelperDVBS2NoiseVarEstimate helper function.
        nVar = HelperDVBS2NoiseVarEstimate(syncOut,rxParams.pilotInd, ...
            rxParams.pilotSeq,false);
        % Recover the BB frame by using HelperDVBS2XBBFrameRecover
        % helper function.
        rxBBFrame = HelperDVBS2XBBFrameRecover(syncOut,phyParams, ...
            rxParams.plScramblingIndex,rxParams.regNumPilotBlks,nVar,true,setNum);
        % Recover the input bit stream by using
        % HelperDVBS2StreamRecover helper function.
        if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
            [decBits,isFrameLost] = HelperDVBS2StreamRecover(rxBBFrame);
            if ~isFrameLost && length(decBits) ~= dataSize
                isFrameLost = true;
```

```
            end
        else
            [decBits,isFrameLost,pktCRC] = HelperDVBS2StreamRecover(rxBBFrame);
            if ~isFrameLost && length(decBits) ~= dataSize
                isFrameLost = true;
                pktCRC = zeros(0,1,'logical');
            end
            % Compute the PER for TS or GS packetized
            % mode.
            pktsErr = pktsErr + numel(pktCRC) - sum(pktCRC);
            pktsRec = pktsRec + numel(pktCRC);
        end
        if ~isFrameLost
            ts = sprintf('%s','BB header decoding passed.');
        else
            ts = sprintf('%s','BB header decoding failed.');
        end
        % Compute the number of frames lost. CRC failure of the
        % baseband header is considered a frame loss.
        numFramesLost = isFrameLost + numFramesLost;
        fprintf('%s(Number of frames lost = %1d)\n',ts,numFramesLost)
        % Compute the bits in error.
        if ~isFrameLost
            dataInd = (rxParams.frameCount-2)*dataSize+1:(rxParams.frameCount-1)*dataSize;
            errs = sum(data(dataInd) ~= decBits);
            bitsErr = bitsErr + errs;
        end
    end

    stIdx = endIdx;
    rxParams.frameCount = rxParams.frameCount + 1;
    formatIdxTemp = formatIdx;
end
```

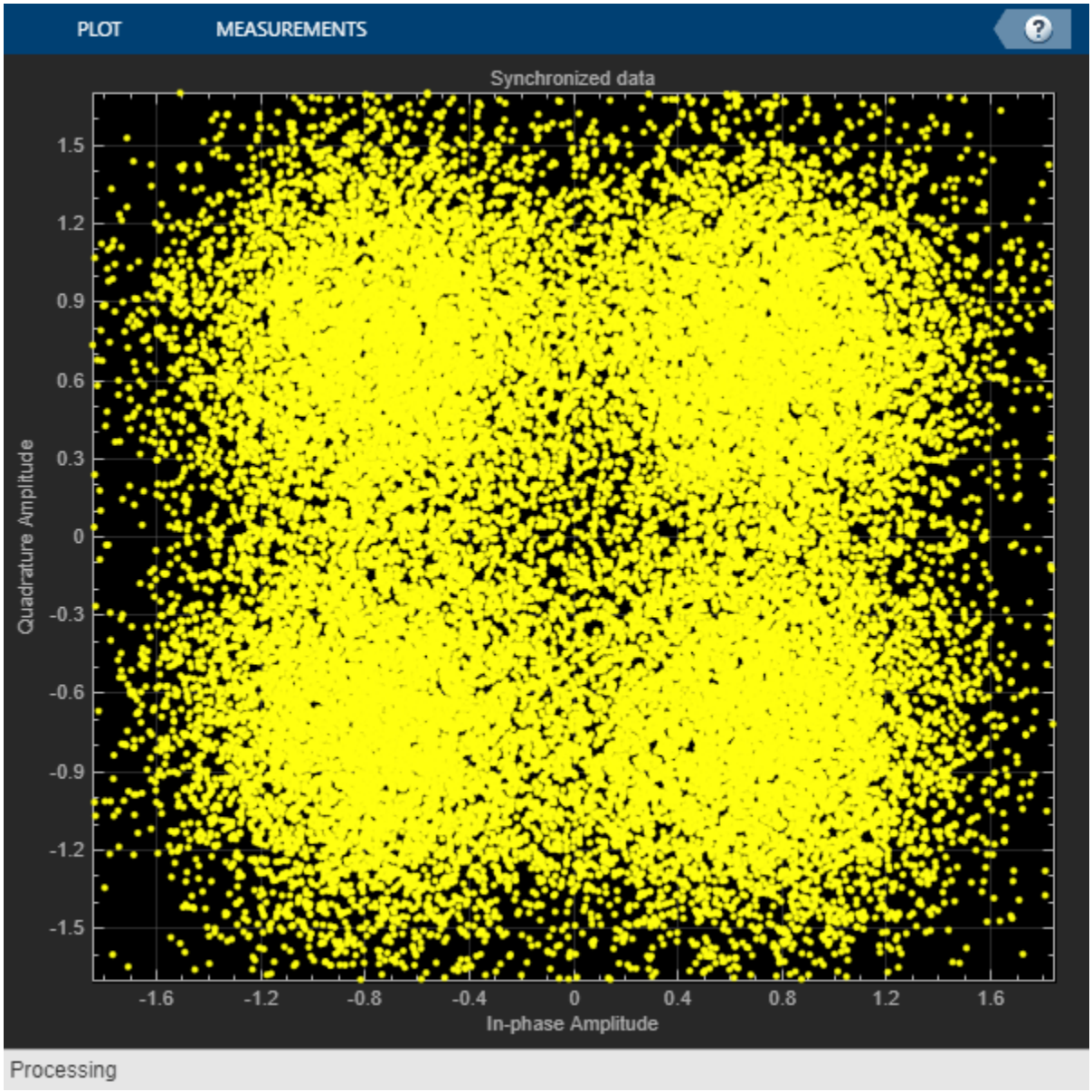
```
MODCOD format detection passed
MODCOD format detection passed
MODCOD format detection passed
MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)
```

```
MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)

MODCOD format detection passed

BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
```

Visualization and Error Logs

Plot the constellation of the synchronized data and compute the BER and PER.

```
% Synchronized data constellation plot
syncConst = comm.ConstellationDiagram('Title','Synchronized data', ...
    'XLimits',[-1.7 1.7],'YLimits',[-1.7 1.7], ...
    'ShowReferenceConstellation',false);
syncConst(syncOut)
```




```
pause(0.5)
% Error metrics display
% For GS continuous streams
if strcmpi(cfgDVBS2X.StreamFormat,'GS') && ~rxParams.UPL
    if (simParams.numFrames-rxParams.totalSyncFrames == numFramesLost)
        fprintf("All frames are lost. No bits are retrieved from BB frames.")
    else
        numFramesRec = simParams.numFrames - rxParams.totalSyncFrames - numFramesLost;
        ber = bitsErr/(numFramesRec*dataSize);
        fprintf('BER           : %1.2e\n',ber)
    end
else
    % For GS and TS packetized streams
    if pktsRec == 0
        fprintf("All frames are lost. No packets are retrieved from BB frames.")
```

```
    else
        if strcmpi(cfgDVBS2X.StreamFormat,'TS')
            pktLen = 1504;
        else
            pktLen = cfgDVBS2X.UPL;      % UP length including sync byte
        end
        ber = bitsErr/(pktsRec*pktLen);
        per = pktsErr/pktsRec;
        fprintf('PER: %1.2e\n',per)
        fprintf('BER: %1.2e\n',ber)
    end
end
```

```
PER: 0.00e+00
```

```
BER: 0.00e+00
```

Further Exploration

The operating E_s/N_o range of the VL-SNR mode being very low requires the normalized loop bandwidth of the symbol synchronizer and coarse FLL to be very small for accurate estimation. Set these parameters using the `rxParams.symbSyncLoopBW` and `rxParams.carrSyncLoopBW`.

Configure Coarse Carrier Synchronization Parameters

Initialize `HelperDVBS2XVLSNRCoarseFreqSynchronizer` System object with `rxParams.carrSyncLoopBW` set as `2e-5` and then run the simulation.

When you set the `PLSDecimalCode` property to `129`, set the `rxParams.initialTimeFreqSync` property to `15`. When you set the `PLSDecimalCode` property to `131`, set the `rxParams.coarseFreqLock` property to `30`. Set `simParams.EsNodB` to the lowest E_s/N_o for the chosen modulation scheme from ETSI EN 302 307-1 Section 6 [1] on page 4-0 .

Replace the code in the coarse frequency synchronization section with these lines of code. Run the simulation for different carrier frequency offset (CFO) values. After coarse frequency compensation, view the plot and the residual CFO value (`resCoarseCFO`) over each frame to observe the performance of the coarse frequency estimation algorithm. Ideally, the coarse frequency compensation reduces the error to 2% of the symbol rate. If the residual CFO error is not reduced to less than 2% of the symbol rate, try decreasing the loop bandwidth and increasing the `rxParams.initialTimeFreqSync`. The coarse frequency is a first order FLL, and it can only detect the static carrier frequency offset. It cannot track Doppler rate changes.

```
% [coarseFreqSyncOut,phEst] = freqSync(syncIn,false);
% Frequency offset estimate normalized by symbol rate
% freqOffEst = diff(phEst(1:simParams.sps:end))/(2*pi);
% plot(freqOffEst)
% Rsym = simParams.chanBW/(1+cfgDVBS2X.RolloffFactor);
% actFreqOff = (simParams.cfo + simParams.peakDoppler)/Rsym;
% resCoarseCFO = abs(actFreqOff-freqOffEst(end));
```

When the residual carrier frequency offset value (`resCoarseCFO`) is reduced to approximately 0.02, set the `rxParams.frameCount` as the `rxParams.initialTimeFreqSync` value.

Configure Symbol Timing Synchronization Parameters

Try running the simulation using the symbol timing synchronizer configured with a normalized loop bandwidth of 1e-4. To achieve convergence of the timing loop, the ratio

`rxParams.symbSyncLoopBW/simParams.sps` must be greater than 1e-5. If the symbol timing loop doesn't converge, try decreasing the `rxParams.symbSyncLoopBW`.

Configure Frame Synchronization Parameters

Proper frame synchronization depends on the segment length, `rxParams.SegLength`, which is used to divide the reference VL-SNR header symbols into smaller segments to perform segment coherent correlation. `rxParams.SegLength` depends on the residual CFO that is present after coarse frequency synchronization and must be less than `round`(3/(8*`resCoarseCFO`)). Prefer using a value that is an integer multiple of 896 (length of VL-SNR WH sequence). If CFO is absent, perform correlation using `rxParams.SegLength` as `896`.

Configure Fine Frequency Synchronization Parameters

When you set the `PLSDecimalCode` property to `129`, set the `rxParams.fineFreqLock` property to `10`. When you set the `PLSDecimalCode` property to `131`, set the `rxParams.coarseFreqLock` property to `20`. Set `simParams.EsNodB` to the lowest E_s/N_o for the chosen modulation scheme from ETSI EN 302 307-1 Section 6 [1] on page 4-0 .

Replace the code in the fine frequency error estimation section with this code. Fine frequency estimator tracks the Doppler rate changes. To estimate the residual CFO error, include the sinusoidal variation of the Doppler shift included in the `actFreqOff` estimate. For an easy workaround, do not introduce Doppler rate in impairment. Instead, add only static CFO and analyze the number of frames required to generate an accurate estimate. Typically, those number of frames are sufficient to estimate CFO changes due to Doppler rate. As the Doppler rate changes are handled by the fine frequency estimator, 2*`simParams.peakDoppler` must be less than the estimation range of the fine frequency estimator. The estimation range depends on the `rxParams.NumLags` parameter and the normalized CFO that can be estimated is given by 1/(`rxParams.NumLags`+1). To increase the estimation range, try reducing the `rxParams.NumLags` . You might need more pilot blocks because the estimation accuracy drops.

```
% fineFreqEst = angle(sum(rxParams.fineFreqCorrVal))/(pi*(rxParams.NumLags+1));
% resFineCFO = abs(actFreqOff-freqOffEst(end)-fineFreqEst);
```

Repeat the simulation process and observe the residual CFO value (`resFineCFO`) over each frame. If the fine frequency estimator does not reduce the residual carrier frequency error to approximately 0.03% of the symbol rate, try increasing the `rxParams.fineFreqLock` property value.

When the residual CFO value (`resFineCFO`) is reduced to approximately 0.0003, update `rxParams.totalSyncFrames` based on `rxParams.initialTimeFreqSyncrxParams` and `rxParams.fineFreqLock` values.

Configure Phase Synchronization Parameters

`HelperDVBS2PhaseEstimate` estimates the residual CFO and phase error appropriately up to -8 dB of E_s/N_o. For E_s/N_o less than -8 dB, set `rxParams.Alpha` to less than `0.5` if data that passed through `HelperDVBS2PhaseEstimate` has only phase error. If the data has residual CFO along with phase error, keep `rxParams.Alpha` as `1` and set `rxParams.NeedSmoothening` as `true`. Smoothening improves the `phaseEst` obtained from `HelperDVBS2PhaseEstimate`.

After refining the synchronization parameters set in the `rxParams` structure, perform the BER simulation for the updated configuration.

Appendix

The example uses these helper functions:

- HelperDVBS2XVLSNRRxInputGenerate.m: Generate DVB-S2X waveform samples distorted with RF impairments and structure of parameters for receiver processing
- HelperDVBS2PhaseNoise.m: Generate phase noise samples for different DVB-S2X phase noise masks and apply it to the input signal
- HelperDopplerShift.m: Generate sinusoidal varying Doppler shift and apply it to input signal
- HelperDVBS2XVLSNRCoarseFreqSynchronizer.m: Perform coarse frequency offset estimation and correction
- HelperDVBS2XVLSNRFrameSync.m: Perform frame synchronization and detect start of VL-SNR header and MODCOD identification
- HelperDVBS2FineFreqEst.m: Estimate fine frequency offset
- HelperDVBS2PhaseEst.m: Estimate carrier phase offset
- HelperDVBS2PhaseCompensate.m: Perform carrier phase compensation
- HelperDVBS2XPLHeaderRecover.m: Demodulate and decode PL header to recover transmission parameters
- HelperDVBS2NoiseVarEstimate.m: Estimate noise variance of received data
- HelperDVBS2XBBFrameRecover.m: Perform PL descrambling, demodulation, decoding and recover BB frame from PL frame
- HelperDVBS2XDemapper.m: Perform soft demodulation for all DVB-S2X based modulation schemes
- HelperDVBS2XLDPCDecode.m: Perform LDPC decoding for all DVB-S2X based LDPC frame formats and code rates
- HelperDVBS2XBCHDecode.m: Perform BCH decoding for all DVB-S2X based frame formats and code rates
- HelperDVBS2StreamRecover.m: Perform CRC check of BB header and recover input stream from BB frame based on header parameters
- HelperDVBS2XBitRecover.m: Perform PL header demodulation and decoding, PL de-scrambling, demodulation, decoding and recover BB frame. Perform CRC check of BB header, and recover input stream from BB frame.

Bibliography

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)*.
2. ETSI Standard TR 102 376-2 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications; Part 2: S2 extensions (DVB-S2X)*.
3. ETSI Standard TR 102 376-1 V1.2.1(2015-11). *Digital Video Broadcasting (DVB); Implementation Guidelines for the Second Generation System for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications (DVB-S2)*.
4. Mengali, Umberto, and Aldo N.D'Andrea. *Synchronization Techniques for Digital Receivers*. New York: Plenum Press,1997.

5 D'Andrea, A. N., and U. Mengali. "Design of Quadricorrelators for Automatic Frequency Control Systems." *IEEE Transactions on Communications,* vol. 41, no. 6, June 1993, pp. 988–97.

6 Casini, E., et al. "DVB-S2 Modem Algorithms Design and Performance over Typical Satellite Channels." *International Journal of Satellite Communications and Networking*, vol. 22, no. 3, May 2004, pp. 281–318.

7 Michael Rice, *Digital Communications: A Discrete-Time Approach*. New York: Prentice Hall, 2008.

Local Functions

```
function [setNum,phyParams] = getVLSNRParams(formatIdx)

tableVLSNR = [4 64800 2/9 0; ...
    2 32400 1/5 0; ...
    2 32400 11/45 0; ...
    2 32400 1/3 0; ...
    2 16200 1/5 1; ...
    2 16200 11/45 1; ...
    2 16200 1/5 0; ...
    2 16200 4/15 0; ...
    2 16200 1/3 0];

params = tableVLSNR(formatIdx,:);
phyParams.ModOrder = params(1);
phyParams.FECFrameLen = params(2);
phyParams.CodeRate = params(3);
[n, d] = rat(phyParams.CodeRate);
phyParams.CodeIDF = [sprintf('%0.0f',n) '/' sprintf('%0.0f',d)];
phyParams.HasPilots = true;
phyParams.HasSpread = params(4);

if formatIdx > 6
    setNum = 2;
else
    setNum = 1;
end
end
% Smoothen the phase estimate using moving average filter
function newEst = smoothenEstimate(est)
errDiff = diff(est(2:end));
thres = -2*sign(mean(errDiff));
width = 5;
if thres > 0
   index = find(errDiff > thres);
else
  index = find(errDiff < thres);
end
if ~isempty(index)
    for k = 1:length(index)
        est(index(k)+2:end) = est(index(k)+2:end)-sign(thres)*2*pi;
    end
end
temp = est(2:end);
n = length(temp);
c = filter(ones(width,1)/width,1,temp);
cbegin = cumsum(temp(1:width-2));
cbegin = cbegin(1:2:end)./(1:2:(width-2))';
cend = cumsum(temp(n:-1:n-width+3));
```

```
cend = cend(end:-2:1)./(width-2:-2:1)';
c = [cbegin;c(width:end);cend];
newEst = [est(1);c];
end
```

See Also

Objects

`dvbs2xWaveformGenerator` | `dvbs2WaveformGenerator`

Related Examples

- “End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames” on page 4-53
- “End-to-End DVB-S2 Simulation with RF Impairments and Corrections” on page 4-36

DVB-S2 Bent Pipe Simulation with RF Impairments and Corrections

This model shows a bent pipe satellite link that transmits a Digital Video Broadcasting Satellite Second Generation (DVB-S2) [5] signal from a first ground station to a satellite. The satellite receives the analog signal, amplifies and filters it without demodulation, then retransmits it to a second ground station. That ground station demodulates and decodes the signal, and a testbench calculates the end-to-end packet error rate (PER) and a low density parity check (LDPC) coding bit error rate (BER).

Introduction

The model creates a DVB-S2 signal that includes:

- Bose-Chaudhuri-Hocquenghem (BCH) encoding
- LDPC encoding (normal and short frame) [6], [7]
- Interleaving
- Modulation (QPSK, 8PSK)

The model also optionally applies multiple RF impairments to the signal on the uplink and downlink, and can also optionally correct them. These impairments include:

- Equation-based and table-based memoryless nonlinearities [1]
- Doppler error
- Receiver thermal noise [4]
- Analog filter effects
- Phase noise [2], [3]
- Amplitude and phase imbalances
- DC offset

This example combines the modeling done in the following examples:

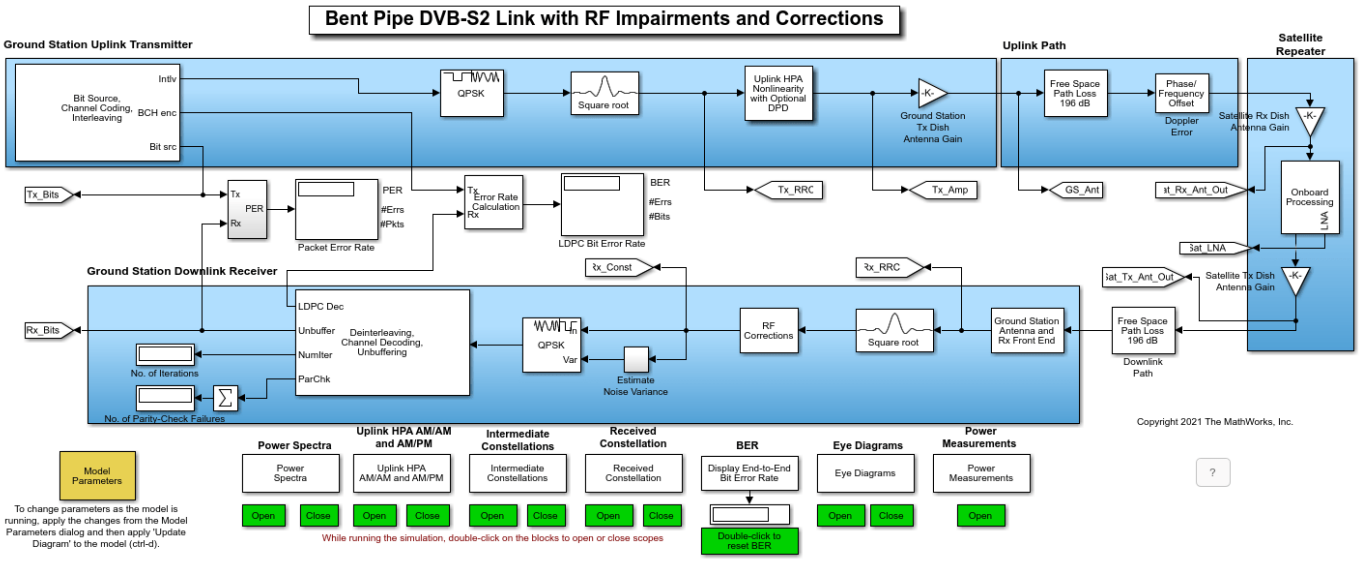
- “RF Satellite Link”
- “DVB-S.2 Link, Including LDPC Coding in Simulink”

Refer to these examples to gain the background necessary to understand this bent pipe example.

Model Overview

This example loads a MAT-file with DVB-S2 LDPC parity matrices. If the MAT-file is not available on the MATLAB® path, then the example downloads them from mathworks.com. Internet connectivity is required to perform this download operation.

The model is shown in the following figure:



The ground station transmitter and uplink path are shown in the top half of the model, and the satellite repeater on the right side of the model. The downlink and ground station receiver are shown in the bottom half of the model. You can change parameters by interacting with the `Model Parameters` block.

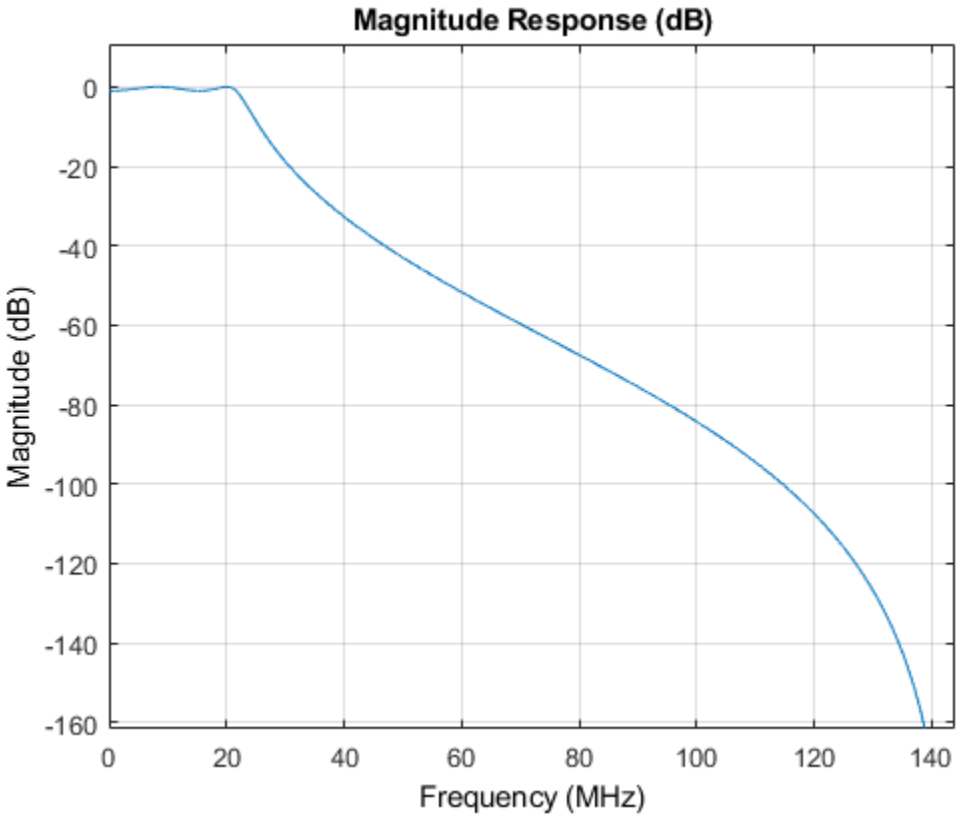
The `Model Parameters` block enables you to update uplink, satellite, and downlink parameters separately. In particular, the block enables you to specify the diameters of the ground station and satellite transmit and receive antennas. With the block you can also set the noise figures of the satellite and the receiving ground station analog front ends.

The `Model Parameters` block also enables you to define in-phase/quadrature (I/Q) amplitude imbalance in dB, I/Q phase imbalance in degrees, and an in-phase DC offset as a percentage of the mean received in-phase signal amplitude.

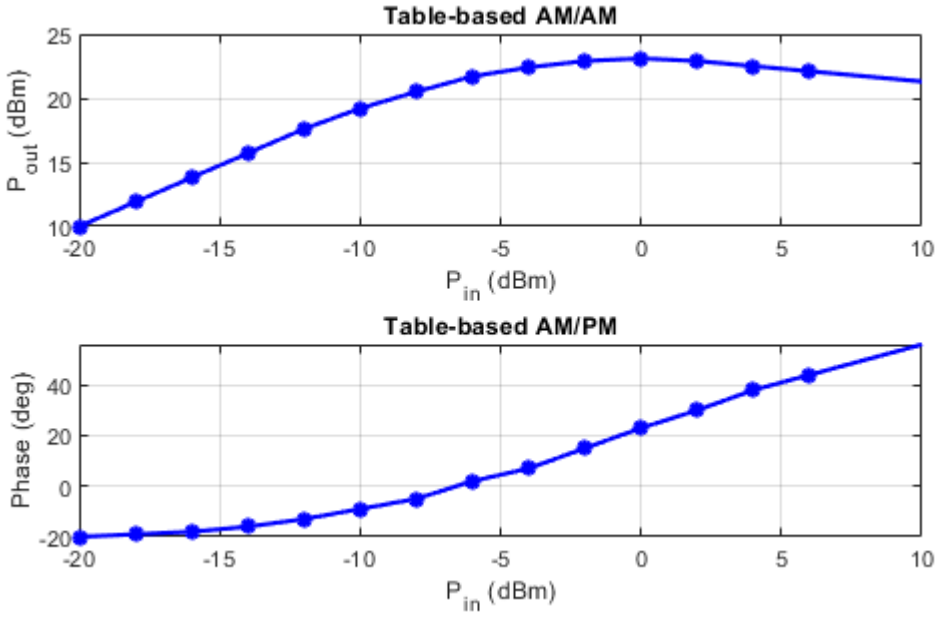
The model also enables multiple visualizations:

- Power spectra
- Constellation diagrams
- Eye diagrams
- AM/AM and AM/PM curves to show nonlinearity effects

The satellite repeater includes several operations not found in the two examples referenced above. First, the repeater models an analog Chebyshev filter to reduce the noise in the signal received by the satellite. You can examine the filter characteristics using the `fvtool` function, using the syntax `fvtool(paramRFSatLink.ChebyNumerator,paramRFSatLink.ChebyDenominator)`.



Also, the satellite repeater employs an amplifier that uses a table-based memoryless nonlinearity. You can use the "Plot Power Characteristics" button of the `Onboard Processing/HPA Nonlinearity` block to generate AM/AM and AM/PM plots for the amplifier. The following figure shows the amplifier AM/AM and AM/PM characteristics.

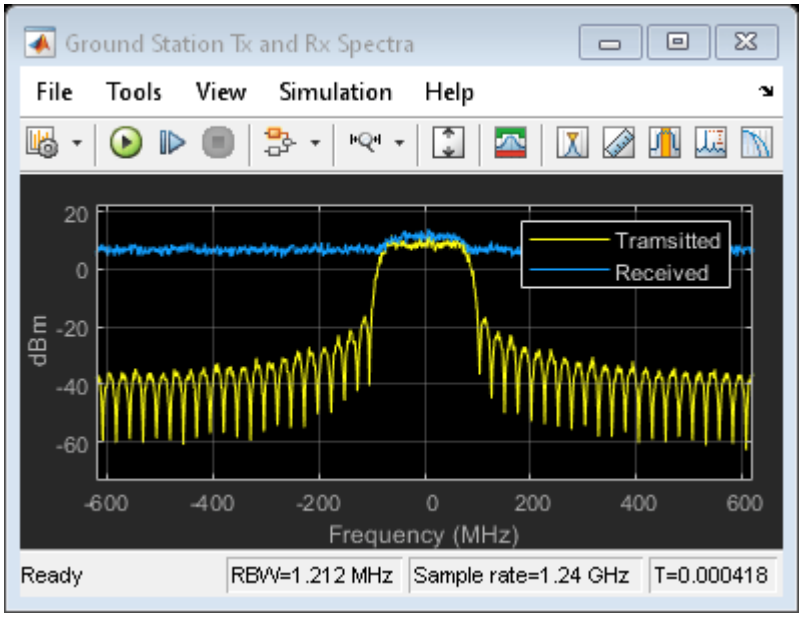


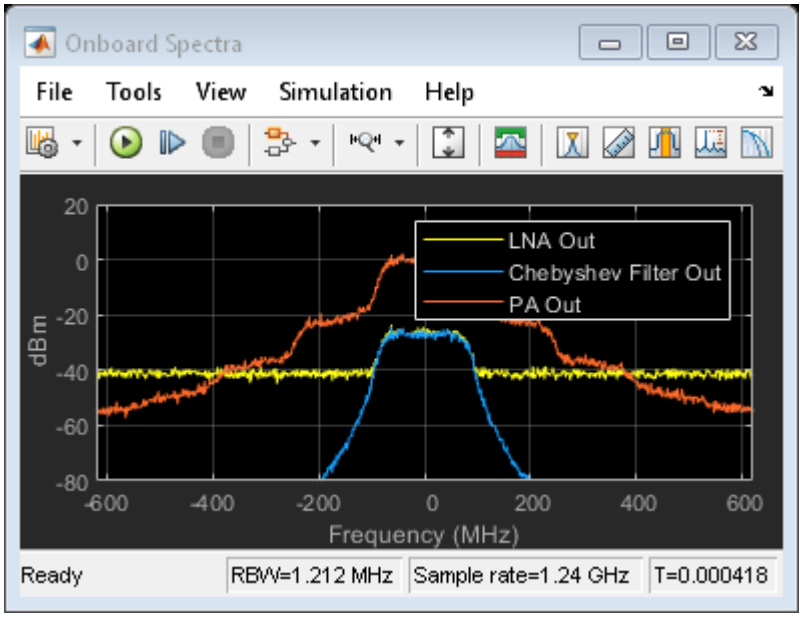
The soft decision QPSK or 8PSK demodulator requires an estimate of the noise variance at its input in order to properly calculate the approximate log-likelihood ratios. The model performs a realistic variance calculation by comparing the received signal against the ideal constellation and calculating error vectors between them. When the noise and other distortions are sufficiently small, the variance calculation is accurate. When the impairments increase such that received constellation points cross over into adjacent, incorrect decision regions, the variance calculation will be overly optimistic.

Simulation Results

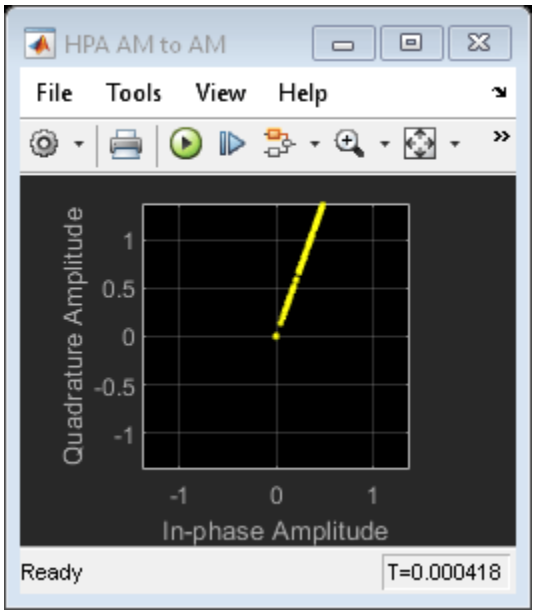
Run the example to see the following run-time visualizations:

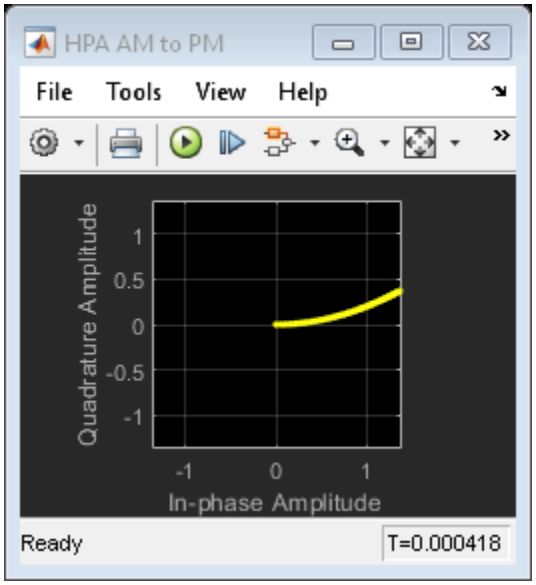
- Power spectra of the transmit and receive ground station signals, and at multiple points during the satellite onboard processing



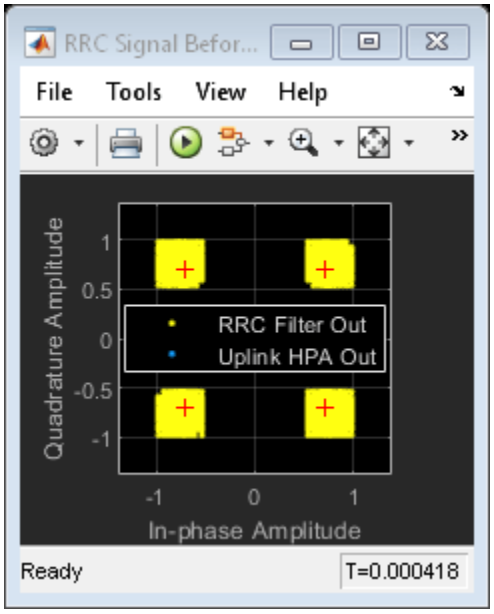


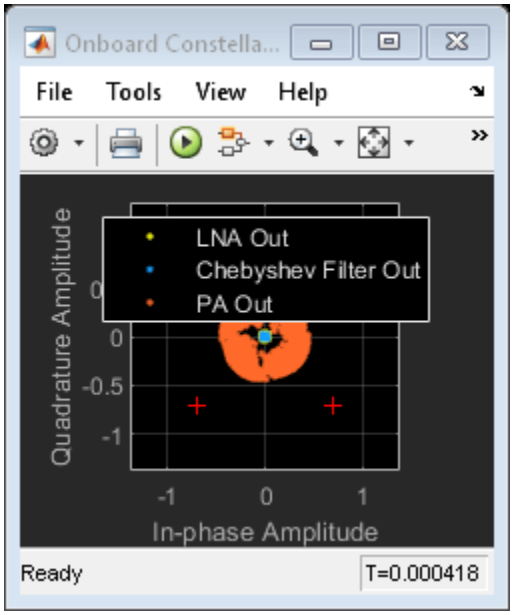
- AM/AM and AM/PM characteristics of the uplink power amplifier



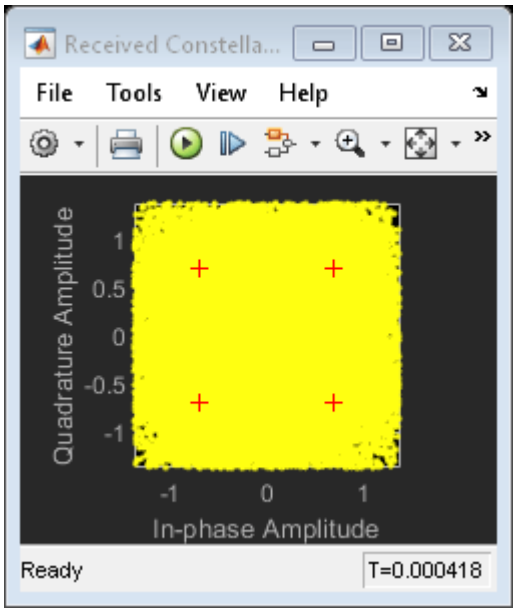


- Constellations before and after the uplink amplifier, and during onboard processing

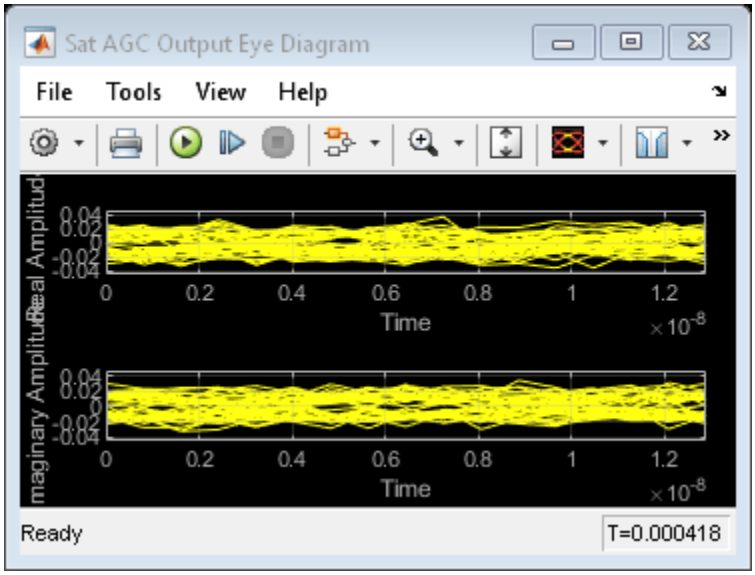


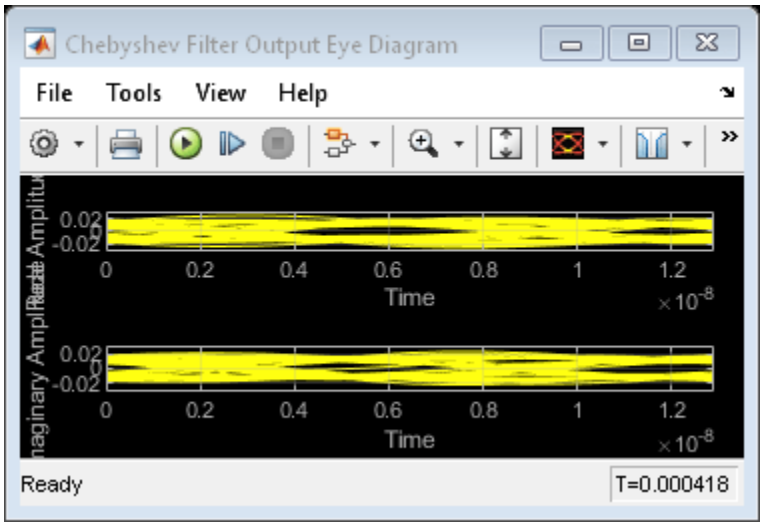


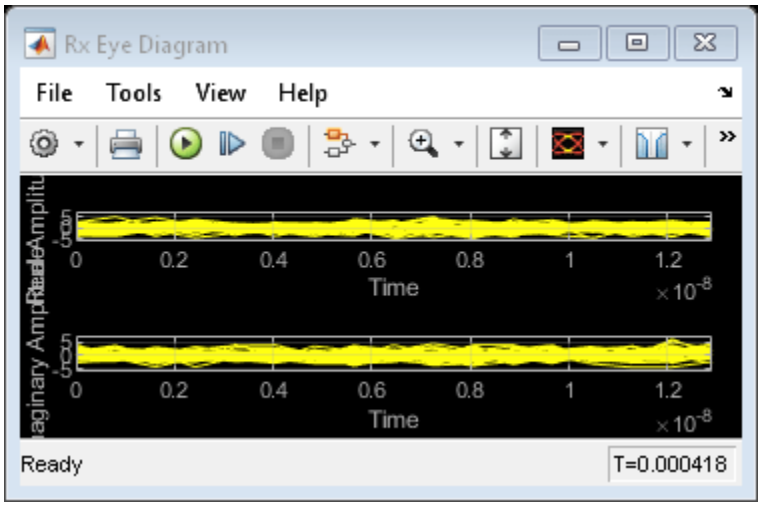
- Received constellation at the ground station input



- Eye diagrams before and after the onboard Chebyshev filter, and at the ground station receiver input







In addition, during run time you can inspect the signal power at the transmitting ground station antenna, the satellite receiver antenna output, the satellite low noise amplifier (LNA) output, and the satellite transmit antenna output.

Further Exploration

You can experiment with the example in the following ways:

- Change modulation and coding formats to determine when the BER unacceptably degrades for a given signal-to-noise ratio (SNR) scenario.
- Turn on single distortions to qualitatively and quantitatively determine their impact on PER and BER.
- Enable RF corrections to ensure that they restore signal quality and PER.
- Reduce the SNR to a level where the RF corrections are no longer effective.
- Navigate to the `RF Corrections` subsystem and tune the parameter values of the individual blocks in the subsystem, such as the `Carrier Synchronizer` or the `DC Blocker`.
- Increase the Chebyshev filter order to determine if the increased group delay distortion affects PER.

- Reduce the satellite amplifier backoff factor to examine its effect on SNR and PER.
- Instead of a geostationary altitude of 35,600 km, change the satellite altitude to a MEO altitude of 20,000 km or a LEO altitude of 2,000 km. Examine how the antenna sizes can then be reduced, or the receiver noise figure can be increased.
- Experiment with different uplink and downlink frequencies.
- Investigate the effect of digital predistortion (DPD) on PER when the uplink amplifier is driven into its saturation region.

Bibliography

[1] Saleh, A. A. M. "Frequency-Independent and Frequency-Dependent Nonlinear Models of TWT Amplifiers." *IEEE Transactions on Communications*, Vol. 29, No. 11, Nov. 1981.

[2] Kasdin, N.J. "Discrete Simulation of Colored Noise and Stochastic Processes and 1/(f^alpha); Power Law Noise Generation." *Proceedings of the IEEE*, Vol. 83, No. 5, May 1995.

[3] Kasdin, N. J., and T. Walter "Discrete .Simulation of Power Law Noise ." *Proceedings of the 1992 IEEE Frequency Control Symposium*, IEEE 1992.

[4] Sklar, Bernard, and Fredric J. Harris. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, NJ: Prentice-Hall, 1988.

[5] ETSI Standard EN 302 307 V1.1.1(2005-03). *Digital Video Broadcasting (DVB); Second Generation Framing Structure, Channel Coding and Modulation Systems for Broadcasting, Interactive Services, News Gathering and other Broadband Satellite Applications*.

[6] Gallager, Robert. "Low-Density Parity-Check Codes." *IRE Transactions on Information Theory*, Vol. 8, No. 1, Jan. 1962: 21-28.

[7] W. E. Ryan, "An Introduction to LDPC Codes." in *Coding and Signal Processing for Magnetic Recoding Systems (Bane Vasic, ed.)*. CRC Press, 2004.

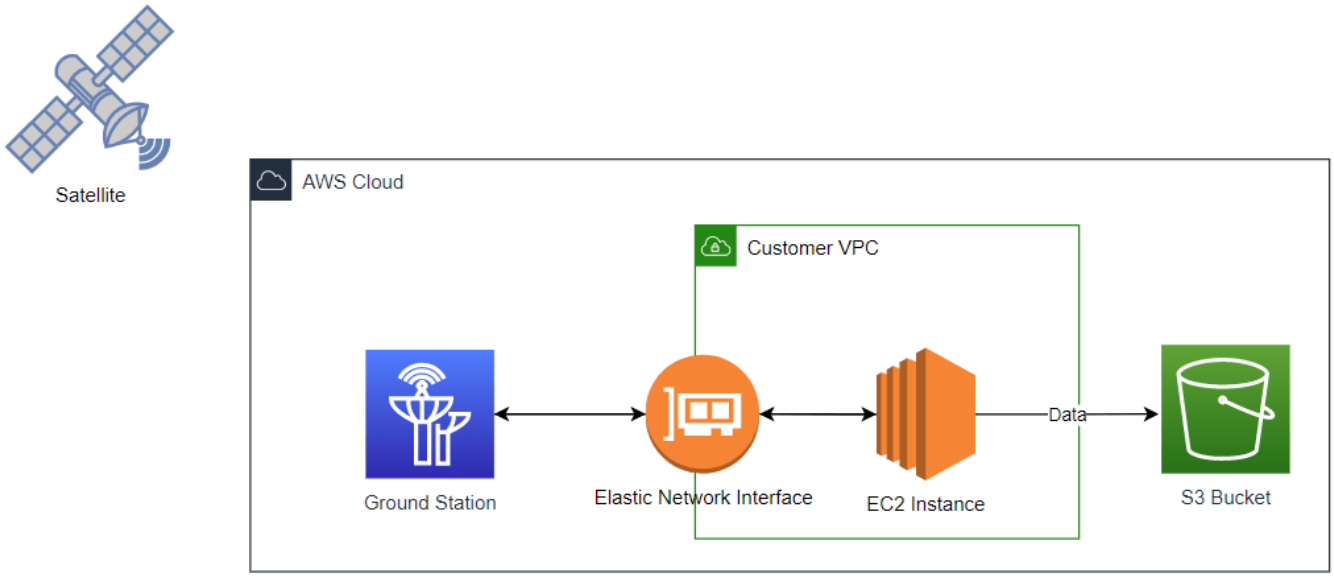
Capture Satellite Data Using AWS Ground Station

This example shows how to use Amazon® Web Services (AWS) Ground Station service from within MATLAB® to receive data from Earth observation satellite AQUA. AQUA (launched in 2002), is an Earth-orbiting National Aeronautics and Space Administration (NASA) scientific research satellite that studies precipitation, evaporation, and cycling of water. You can capture data from other satellites for which you have access permission by changing the satellite information. Using this example, you can capture satellite data as radio frequency (RF) in-phase quadrature (I/Q) samples and as demodulated and decoded data. You can further process and analyze this captured data using the Communications Toolbox™ and Image Processing Toolbox™.

Introduction

AWS Ground Station is a service that enables you to manage satellite communications and process data without the need to build or maintain your own ground station infrastructure. You can get more information about AWS Ground Station service and its capabilities from the AWS Ground Station website.

AWS Ground Station currently supports low earth orbit (LEO) and medium earth orbit (MEO) satellites. These satellites are visible from the ground station for only few minutes during each pass due to their orbital cycles. Communication is possible when the satellites are within the line of sight of a ground station. AWS Ground Station establishes contact with the satellites, and then receives, demodulates, and decodes RF signals. Ground Station then pushes data to the receiver Elastic Compute Cloud (EC2) instance as a VITA 49 stream. This figure shows the end-to-end connection of the data received into Virtual Private Cloud (VPC) from the AWS Ground Station for the scheduled duration and then pushes the data to the AWS Simple Storage Service (S3) bucket.



AWS Services and Costs

This example uses these AWS services, some of which can incur costs on your AWS account. For cost estimates, see the respective pricing page websites for each of the AWS services that this example uses.

- AWS Ground Station: For pricing information, see AWS Ground Station pricing.
- AWS EC2 and Elastic IP: This example uses the m5.4xlarge instance type. For pricing information, see Amazon EC2 pricing.
- AWS VPC and NAT gateway: For pricing information, see Amazon VPC pricing.
- AWS CloudWatch: For pricing information, see Amazon CloudWatch pricing.
- AWS Simple Notification Service (SNS): For pricing information, see Amazon SNS pricing.
- AWS S3: For pricing information, see Amazon S3 pricing.
- AWS GaurdDuty: For pricing information, see Amazon GuardDuty pricing.
- AWS CloudFormation: For pricing information, see AWS CloudFormation Pricing.

AWS lets you visualize, understand, and manage your AWS costs and usage over time. For more details, see the AWS Cost Explorer AWS website.

All the services created through this example can be released at any point of time. For more details, see Delete AWS Cloud Resources on page 4-0 section.

Set Up Access to AWS

To capture data from satellites using the AWS Ground Station service, you must have an AWS account with permission to AWS services that this example uses. This section describes one-time setup steps to gain access to the AWS Ground Station service. If you have already completed this procedure, skip this section.

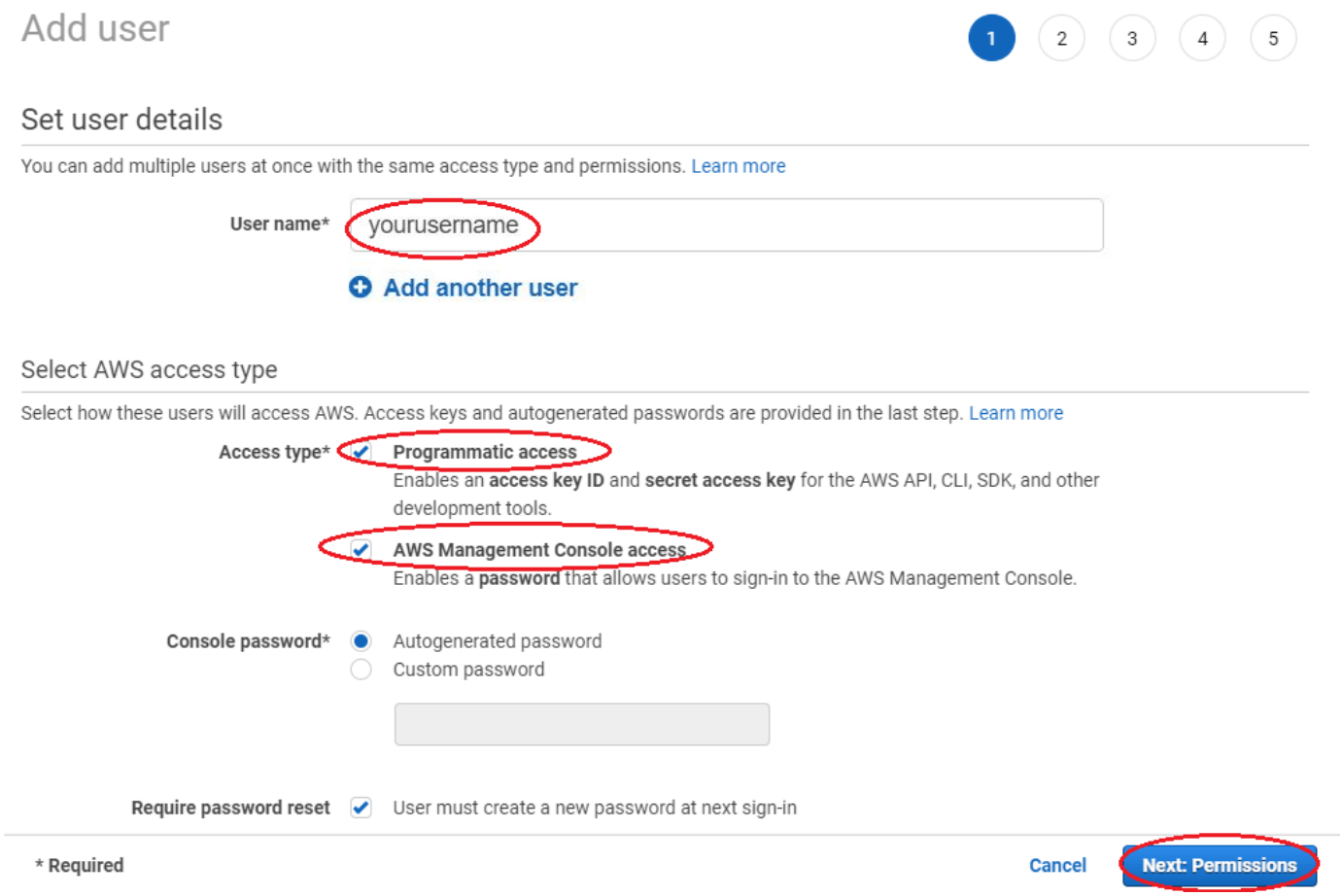
Create an IAM User

If you already have an Identity and Access Management (IAM) user, skip this section.

1. Sign up for an AWS root account. For details, see the AWS website.
2. Open the IAM console at this website: IAM Console.
3. Click **Users** in the IAM console, and then click **Add User**. To complete creating an IAM user, follow the AWS add-user steps, which the next five subsections.

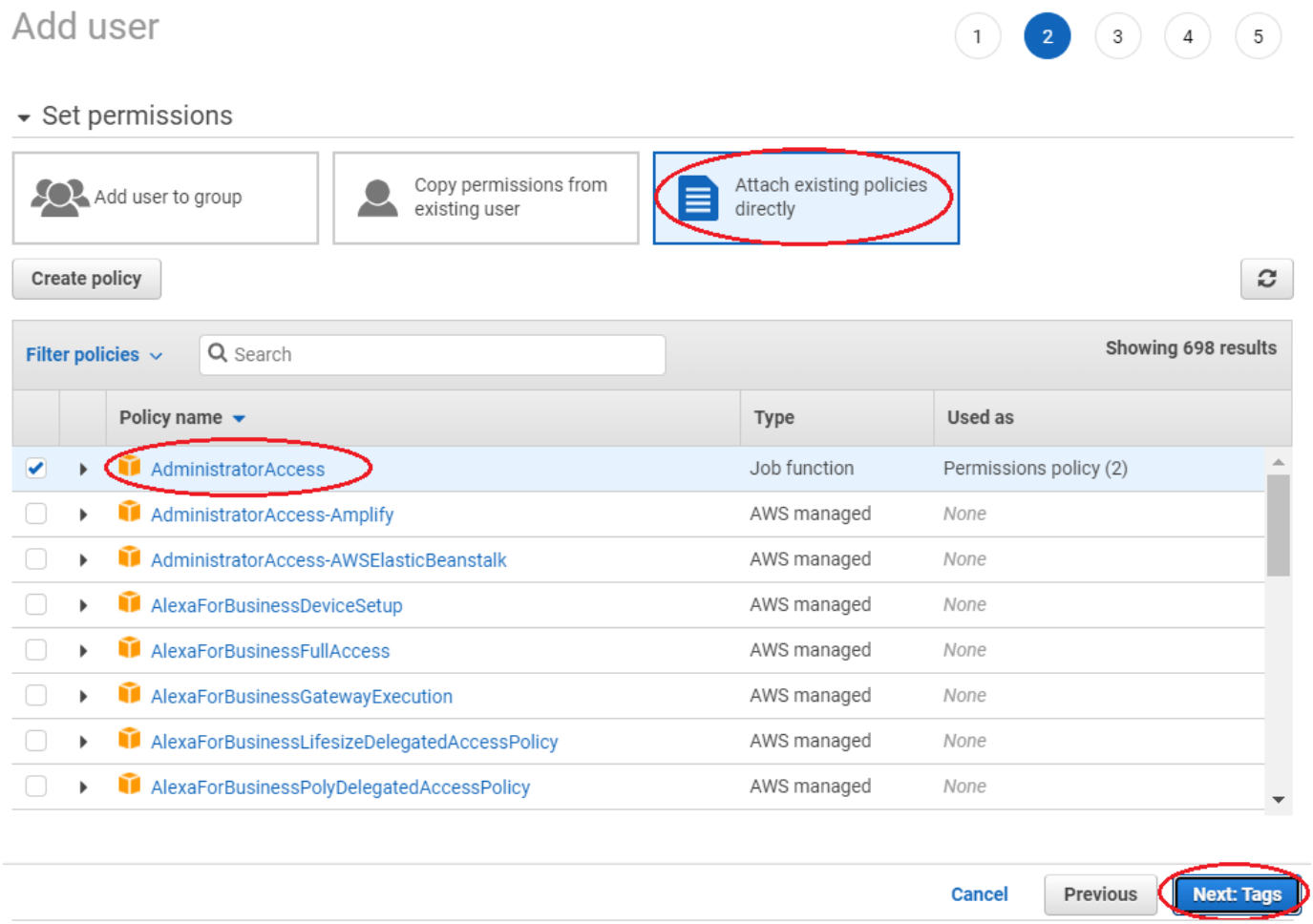
Step 1: Set user details

Follow the prompts to set user details. This figure shows an example of setting user details and AWS access type. Because this data capture example uses programmatic calls to AWS, so you must select **Programmatic access** and **AWS Management Console access**.



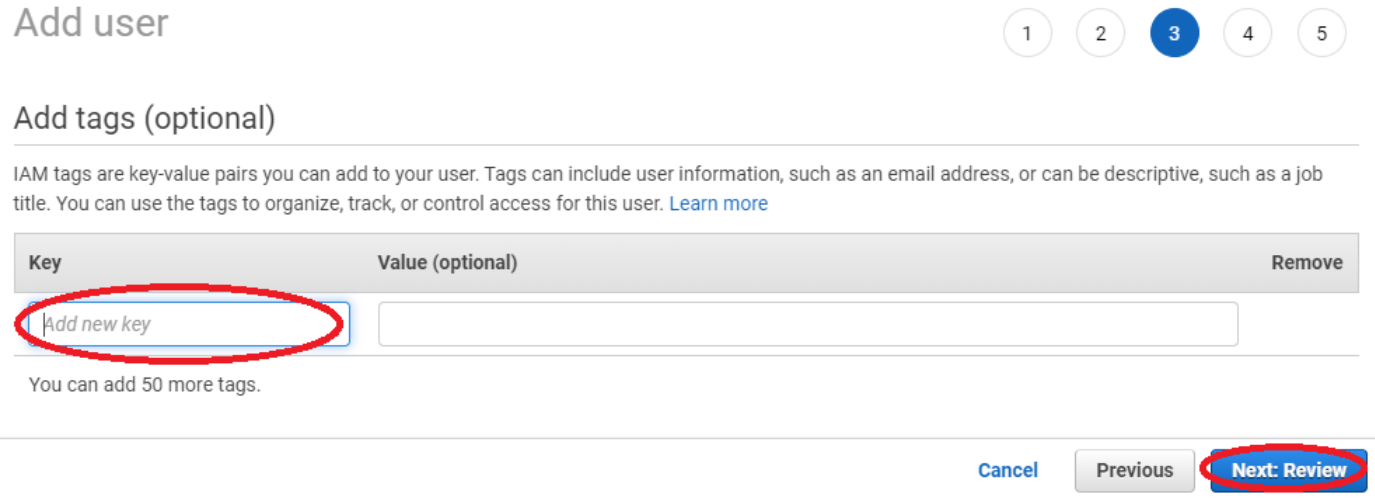
Step 2: Set permissions

Follow the prompts to set permissions. This figure shows an example of setting permissions for your IAM user. Because you must have administration permissions to access AWS services, select the `AdministratorAccess` policy from the **Attach existing policies directly** policies.



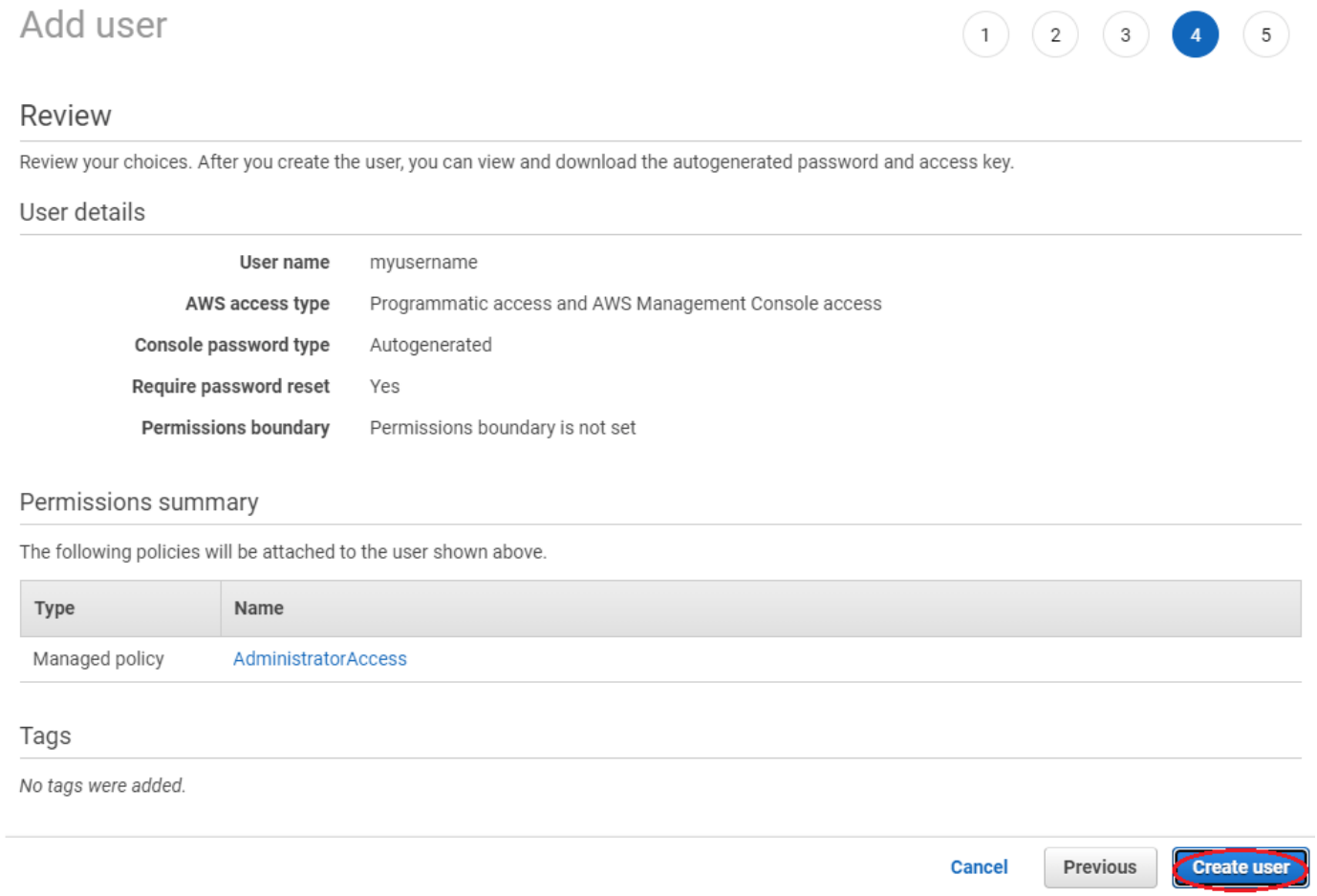
Step 3: Add tags

Follow the prompts to add tags. This figure shows an example of adding tags to your IAM user.



Step 4: Review

Follow the prompts to review the details of your IAM user and Create user. This figure shows an example of IAM user details, permission summary, and attached policies for this example.



Step 5: Complete

This figure shows an example of viewing and downloading your IAM user security credentials. You can get your credentials using any of these options.

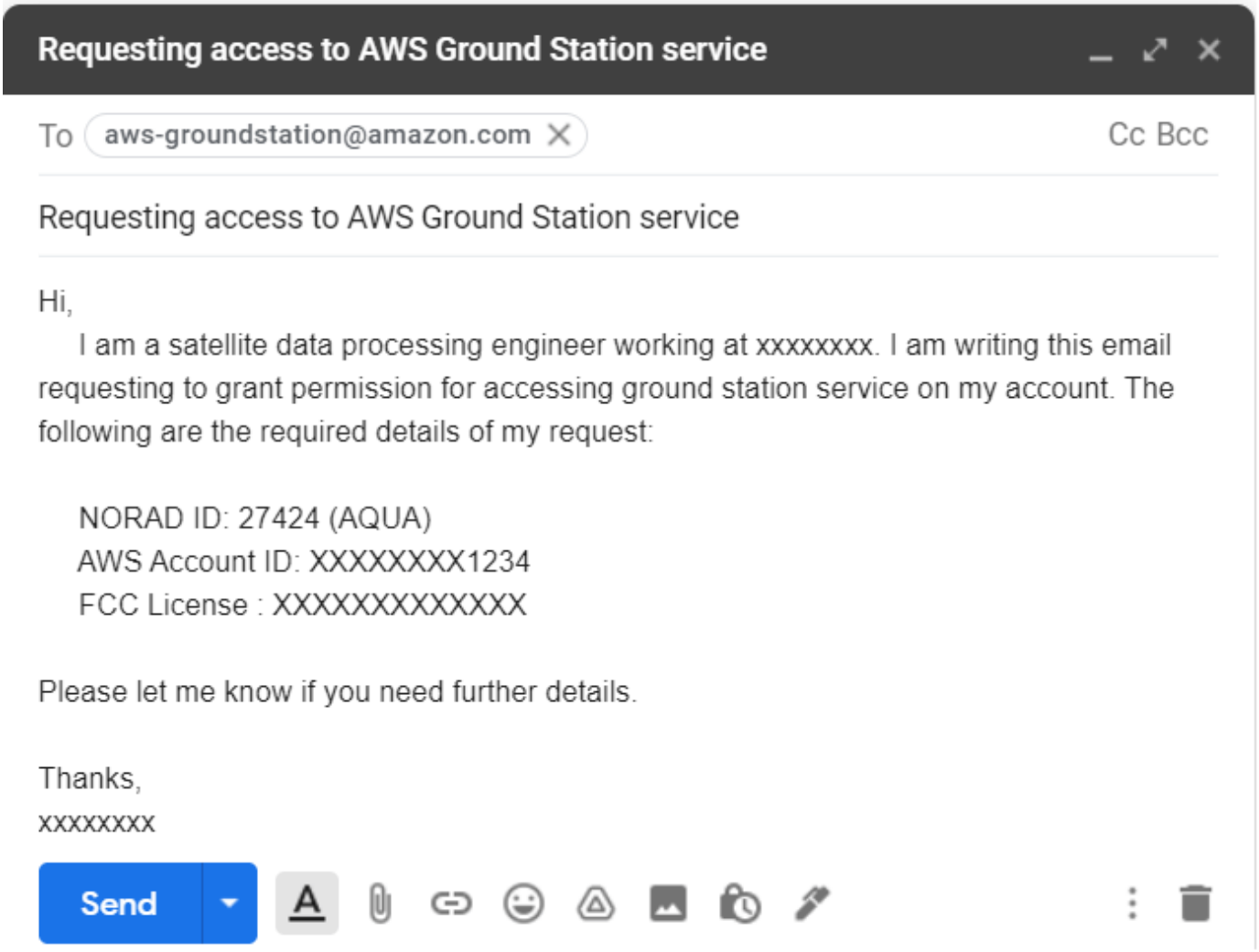
- Click **Download .csv,** to download the credentials file, which includes your IAM username, access key ID, secret access key.
- Click **Send email,** to receive an email with instructions about how to sign into the AWS Management Console.
- Copy the access key ID, secret access key manually.

For more information about IAM users. see the Creating an IAM user in your AWS account and Managing access keys for IAM users AWS websites.

Request Access to Ground Station

To get access to ground station services on your account, email aws-groundstation@amazon.com with the NORAD ID, FCC license information (see the FCC licensing), and AWS account ID.

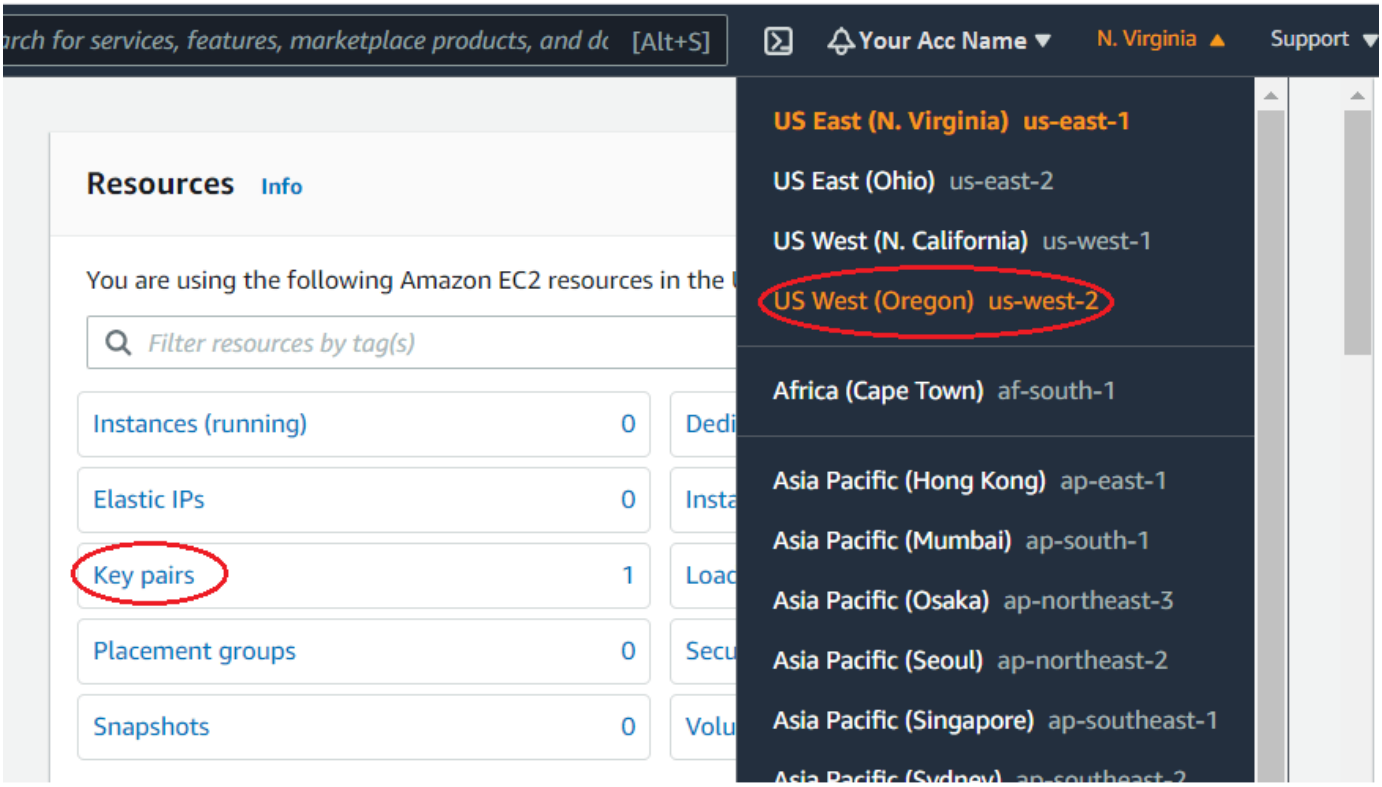
This figure shows a sample email to get access to AWS Ground Station service.



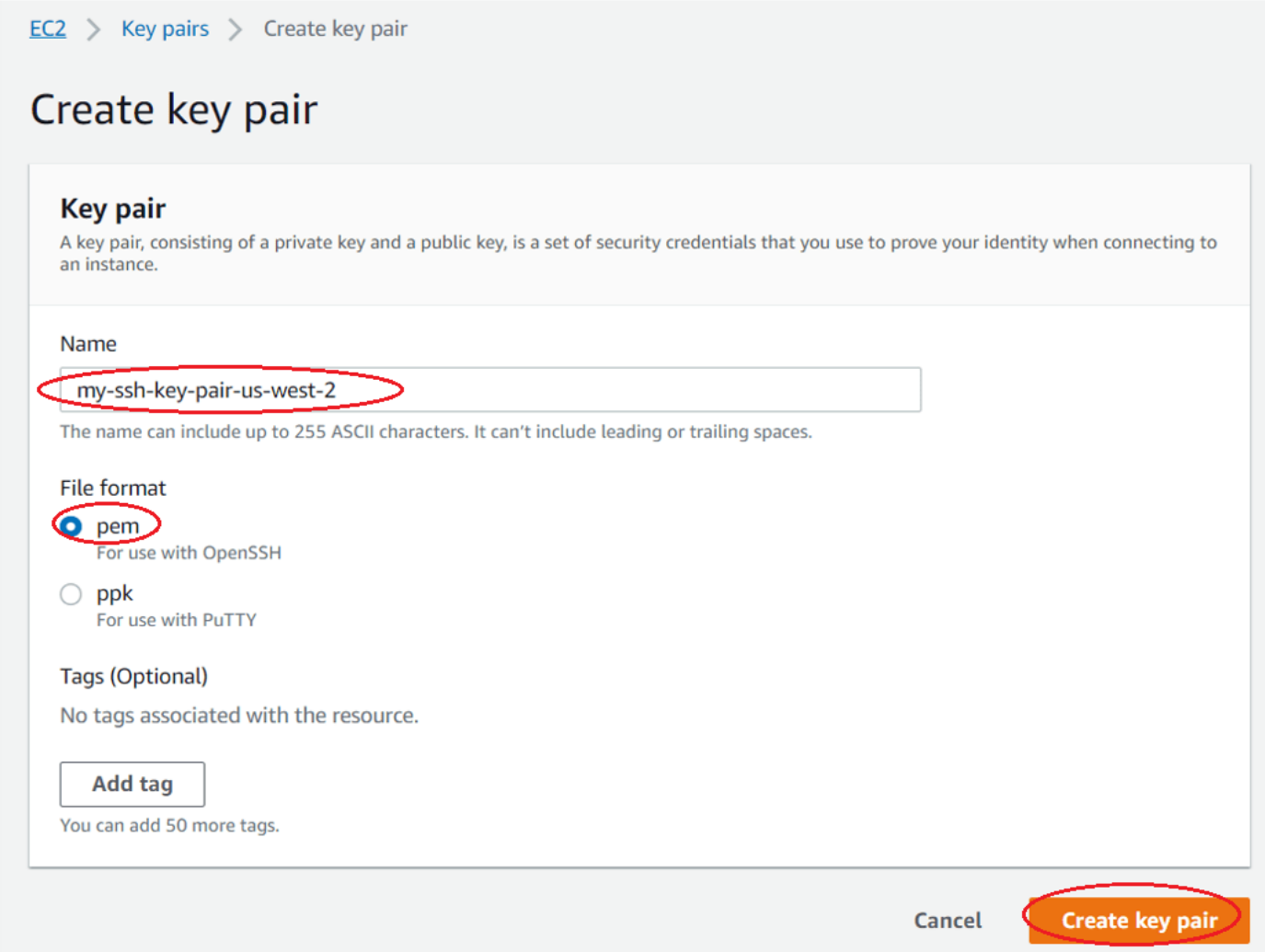
Create EC2 Key Pair

Create an EC2 key pair for the AWS region where you plan to receive data. The EC2 Key pair is used to connect to your EC2 instances for debugging and to get the status of the data capture.

1. Open the Amazon EC2 console at this website: EC2 Console.
2. Click the region list on the navigation bar. For more information on regions, see the Regions.



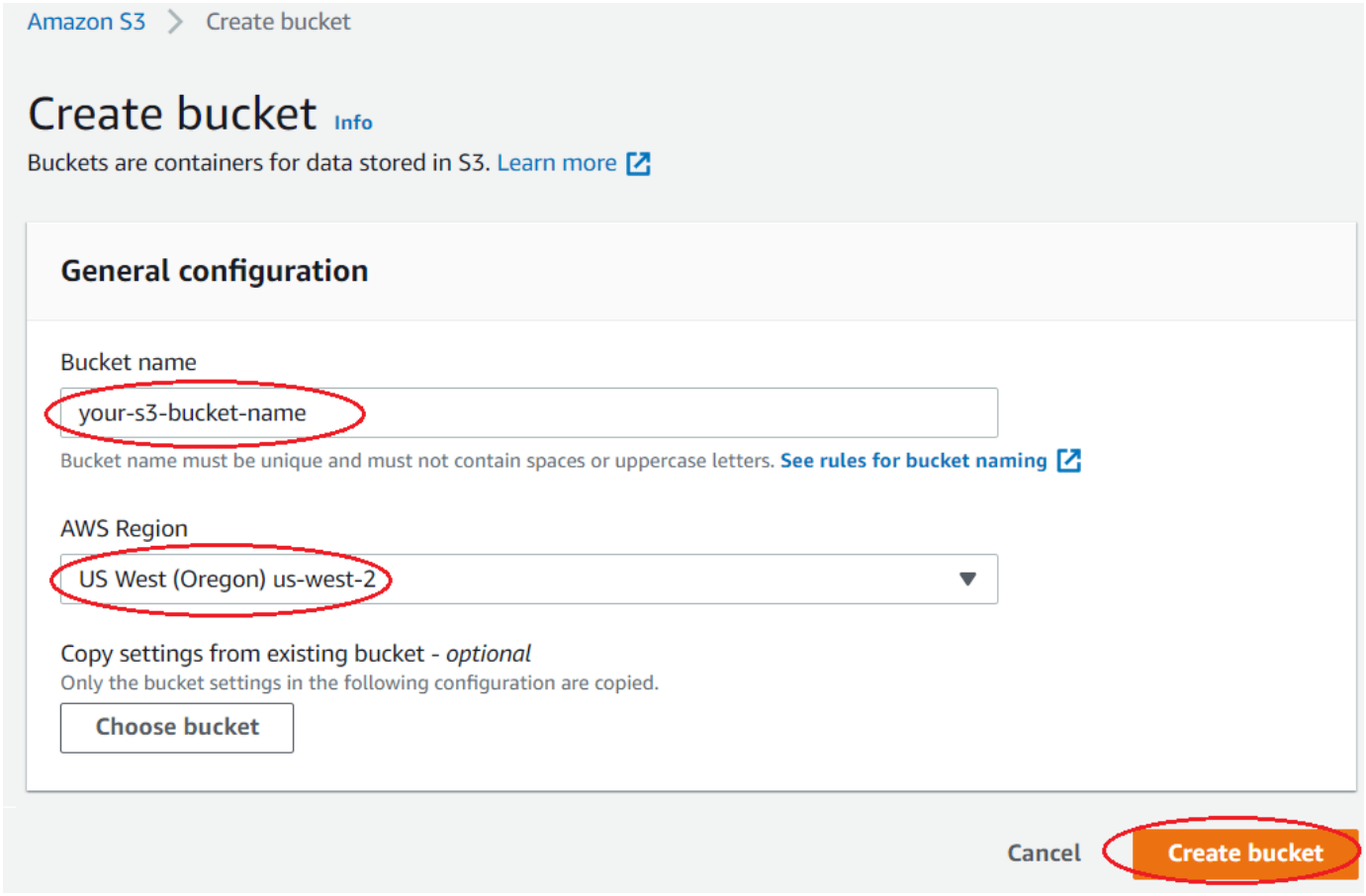
1. Select **Key pairs**, and then click **Create key pair**. For more information on EC2 key pairs, see the Create a key pair using Amazon EC2 AWS website.
2. Specify a unique name for your EC2 key pair and select the file format as `.pem` to save the private key. Click **Create key pair** to create an EC2 key pair.
3. The Key pair is created, and the private key file automatically downloads to your host PC. Save the file. This point in the process is the only point at which you have the option to save the private key file. If you lose this file or do not save it, you must repeat the process again to create new key pair.



Create S3 Bucket

Create an S3 bucket, which stores captured satellite data, by following these steps.

1. Open the S3 console at this website: S3 Console.
2. Choose **Create bucket**.
3. Specify a bucket name, select the corresponding AWS region, and click **Create bucket**.



For more information, see the Create your first S3 bucket.

Configure AWS CLI

Download and install the latest AWS command line interface (CLI). For details, see the AWS Command Line Interface. You can control multiple AWS services using this AWS CLI.

Configure the AWS CLI using `.csv` file that you downloaded in Create IAM user on page 4-0 section. This file contains your IAM username, AWS access key ID, and AWS secret access key.

```
% Specify your IAM username, (for example, IAMUserName = 'yourusername')
IAMUserName = saml;
% Provide your credentials file location
% (for example, C:\Work\new_user_credentials.csv)
csvCredentialsFile = fullfile('C:','Work','new_user_credentials.csv');
HelperAWSConfig(IAMUserName,"csvFileName",csvCredentialsFile);
```

Alternatively, if you are using single sign on (SSO) or federated login, then the AWS credentials file is created in a default directory. In such cases, run this code instead.

```
HelperAWSConfig(IAMUserName);
```

Get Satellite List

To list the satellites accessible in the given AWS region (that is, the same region where you created the EC2 key pair in the Create EC2 Key Pair on page 4-0 section), run this code.

```
% Provide the region name that you plan to use
region = US West (Oregon);
% List the satellites available in given region
[satelliteID,groundStationName] = HelperAWSListSatellites(region);
```

Data Capture Setup

Create infrastructure to capture the data (such as leasing, computing resources), and load and run the software on to the leased machines.

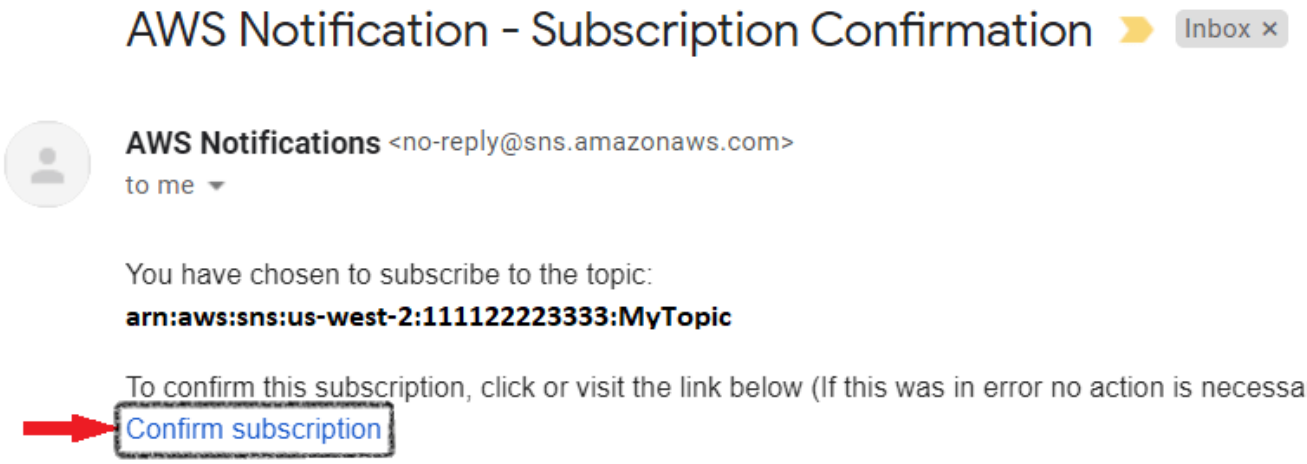
1. Lease two EC2 instances (the t2.micro and m5.4xlarge instance type).
2. Copy the capture script and data transfer scripts to the EC2 instance.
3. Configure the SNS subscription email to get notifications at various stages of the data capture process.

Provide the EC2 key pair full file path, S3 bucket name, and notification email address to the `HelperAWSDataCaptureSetup` helper function. The resources created during this process are tagged with a tag key value pair that can be used to track or control resources. To set up data capture, run this code.

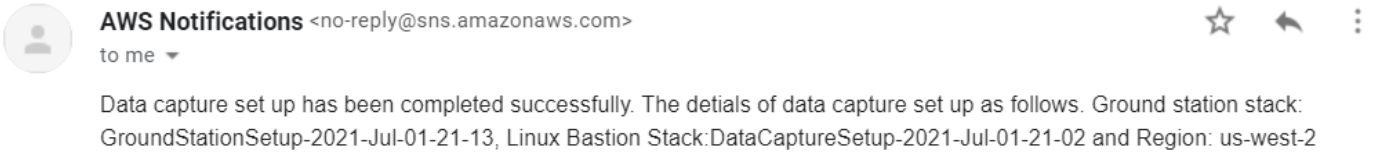
```
% AQUA satellite ID 27424
noradID = 27424;
% Provide the full path of your PEM SSH key pair name that you generated in
% the 'Create EC2 Key Pair' section
% (for example, ec2KeyPairFile = 'C:\Work\my-ssh-key-us-west-2.pem')
ec2KeyPairFile = fullfile('C:','Work','your-ec2-key-pair-us-west-2.pem');
% Provide S3 bucket name to store the satellite data that was created in
% the 'Create S3 Bucket' section, (for example, s3BucketName = 'your-s3-bucket-name')
s3BucketName = your-s3-bucket-for-us-wes;
% Provide your email address to receive the notifications
% (for example, youremail@domain.com)
NotificationEmail = yebarri@mathworks.com;
% Set up the data capture
[tagKeyName,tagValueName] = HelperAWSDataCaptureSetup(noradID,ec2KeyPairFile, ...
    s3BucketName,NotificationEmail,region);
```

During data capture setup, you receive two SNS topic subscription confirmation requests from the AWS Notification service. Confirm the subscription to receive further notifications.

This figure shows a sample email notification from the AWS Notification service. In the email notification, click the "Confirm subscription" link to receive SNS notifications.



After you complete the data capture set up, you receive a SNS notification. This figure shows an example of this notification.



Schedule Contact

Check the availability of the selected satellite in the given region and then schedule a contact. To get satellite available start time and available duration, run this code.

```
% Select a ground station (for example, groundStation = 'Ohio 1')
groundStation = 'Ohio 1';
% Lists the available contacts in the given region and ground station
[contactStartTime,contactDuration] = HelperAWSListContacts(noradID, ...
    groundStation,tagKeyName,tagValueName,region);
```

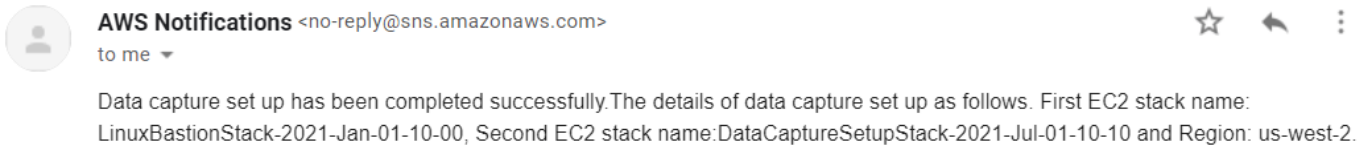
You can schedule a contact by using the `HelperAWSScheduleContact` helper function with these input parameters.

- Capture start time: Specify your start time of the contact.
- Capture duration: Specify the duration (in seconds) to capture data.

```
% Provide a start time for the contact
% (for example, captureStartTime ='26-Apr-2021 13:47:12')
captureStartTime = contactStartTime(1);
% Provide a contact duration (in seconds), making sure that the contact is available for
% the given duration
captureDuration = 60; % in seconds
% Schedule contact. This function reserves the contact for the specified time
% and captures satellite data for the specified duration.
contactArn = HelperAWSScheduleContact(noradID,captureDuration, ...
    captureStartTime,groundStation,tagKeyName,tagValueName,region);
```

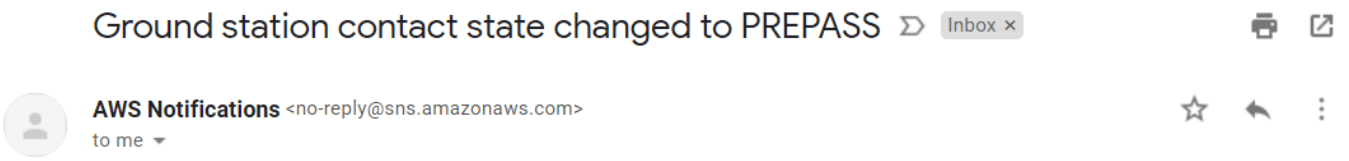
Email Notifications

After scheduling a contact, you receive an AWS notification email regarding the status of your contact. This figure shows an example of successful reservation of the contact.

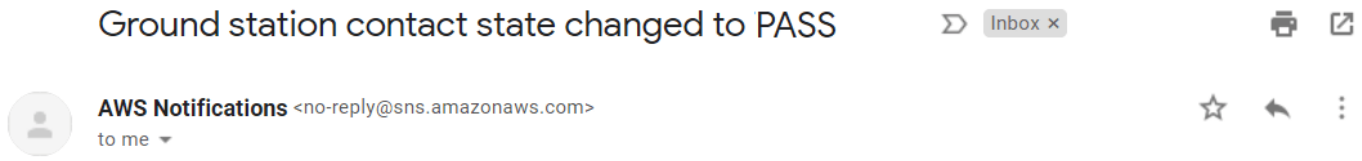


After Ground Station establishes contact with the satellite, you receive notifications for possible contact states like `PREPASS`, `PASS`, `POSTPASS`, `COMPLETED` and `FAILED`.

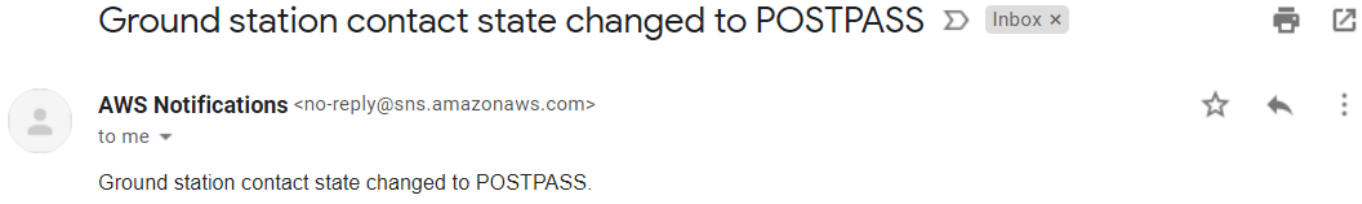
- **PREPASS:** Approximately 2 minutes prior to contact start, you receive an email notification indicating an upcoming pass. This figure shows an example of the ground station contact state changed to `PREPASS`.



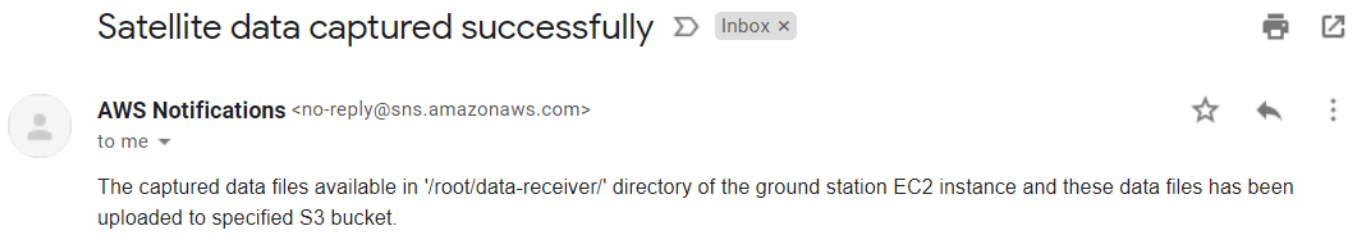
- **PASS:** The duration of the contact you scheduled, you receive an SNS event indicating that the capturing satellite data has started. This figure shows an example of the ground station contact state changed to `PASS`.



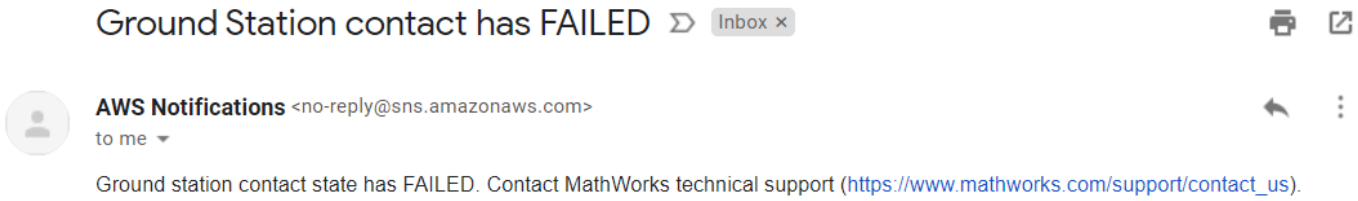
- **POSTPASS:** After one minute of pass time, you receive an SNS event indicating the pass has finished. This figure shows an example of the ground station contact state changed to `POSTPASS`.



- **COMPLETED:** When satellite data captured successfully for the capture duration, you receive an SNS event indicating that contact has been completed successfully. This figure shows an example of the ground station contact state changed to `COMPLETED`.



- **FAILED:** If any failures occur during contact state change, then you receive an SNS event indicating that the contact has failed. In this case, contact MathWorks(R) technical support. For details, see Contact Support on the MathWorks website. This figure shows an example of the ground station contact state changed to `FAILED`.



List Scheduled Contacts

List your scheduled contacts by running this code.

```
% List scheduled contacts
[scheduledContactList] = HelperAWSListScheduledContacts(region);
```

Cancel Contact

You can cancel your scheduled contact before the scheduled time. Before doing so, review the terms and conditions for canceling a scheduled contact, as well as the associated charges and pricing. To cancel a scheduled contact, uncomment and run this code.

```
% HelperAWSCancelContact(contactArn,region);
```

Get Captured Data from S3 Bucket

The raw data files (like `downlinkDemodDecode___.bin` and `downlinkData___.bin`) are pushed to the S3 bucket. To download the raw data from the S3 bucket, uncomment and run this code.

```
% % Specify the raw data file in S3 bucket
% satelliteRawDataFileName = ...%_MWLCFILEREF_matlab/help/examples/satcom/win64/CaptureSatellite
% HelperAWSGetDataFromS3(s3BucketName,satelliteRawDataFileName);
```

Make another Data Capture

To schedule another data capture, repeat the steps from Data Capture Setup on page 4-0 section**.**

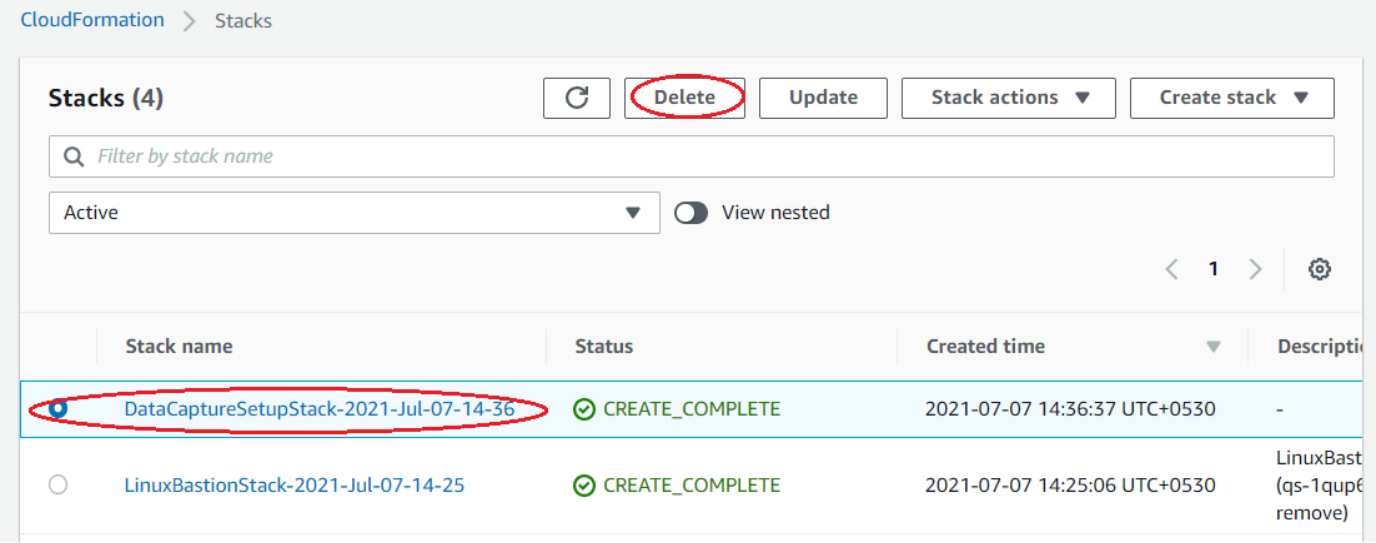
Delete AWS Cloud Resources

After you have finished capturing data, deleting all resources is recommended to avoid incurring further cost. This action deletes all of resources that are tagged with the tag key value pair. To delete the resources, uncomment and run this code.

```
% HelperAWSCleanResources(region,tagKeyName,tagValueName);
```

In case of further errors, you can delete the resources from the AWS Management Console. For example, to delete the AWS Cloud Formation Stack, follow these steps.

1. On the AWS Cloud Formation Console website, and select the stack to delete. The stacks created in this example are named with a prefix `LinuxBastionStack`, `DataCaptureSetupStack`, and are tagged with a prefix of `MW_App` and `SatComToolbox`.
2. Click **Actions,** and then click **Delete** from the menu that appears.



Close AWS Account

Only the AWS account root user can close an AWS account. For more information, see the Closing your AWS account AWS website.

Further Exploration

During data capture at the scheduled time, you can connect to the EC2 instance by using this SSH command and observe the data transfer status, such as bandwidth or bytes transferred.

```
% Get the SSH command to connect to the EC2 instance
sshCommand = HelperAWSGetSSHCmd(ec2KeyPairFile,tagKeyName,tagValueName,region);
```

Login to the EC2 instance using the `sshCommand` to see the logs from your AWS CLI. You can track the installation and configuration of the software on the EC2 instance by checking the logs at the location `/var/log/user-data.log`.

Other Satellites

You can collect data from any of these satellites by changing the satellite NORAD ID in the data capture setup function.

- NOAA 20 JPSS 1 (NORAD ID 43013): This satellite was launched in 2017 and orbits at an altitude of 825 km. It carries five sensors that are designed to study land and water.
- SUOMI NPP (NORAD ID 37849): This satellite was launched in 2011 and orbits at an altitude of 883 km. It carries four sensors that are designed to provide climate measurements.
- TERRA (NORAD ID 25994): This satellite was launched in 1999 and orbits at an altitude of 705 km. It carries five sensors that are designed to study the surface of the Earth.

Appendix

The example uses these helper functions:

- HelperAWSConfig.m: Sign to AWS CLI through MATLAB by providing IAM username
- HelperAWSListSatellites.m: List available satellites in region and checks if the region has ground station
- HelperAWSDataCaptureSetup.m: Set up Linux bastion and ground station stacks and then launches the EC2 instances respectively for the stacks
- HelperAWSListContacts.m: List contacts present in specific region
- HelperAWSScheduleContact.m: Schedule contact
- HelperAWSListScheduledContacts.m: List scheduled contacts
- HelperAWSCancelContact.m: Cancel scheduled contact
- HelperAWSGetSSHCmd.m: Get secure shell (SSH) command to ssh into ground station Linux machine
- HelperAWSCleanResources.m: Clean all stacks and unused resources in region

5

Code Generation and Deployment

What is C Code Generation from MATLAB?

You can use Satellite Communications Toolbox together with MATLAB® Coder™ to:

- Create a MEX file to speed up your MATLAB application.
- Generate ANSI®/ISO® compliant C/C++ source code that implements your MATLAB functions and models.
- Generate a standalone executable that runs independently of MATLAB on your computer or another platform.

In general, the code you generate using the toolbox is portable ANSI C code. In order to use code generation, you need a MATLAB Coder license. For more information, see “Get Started with MATLAB Coder” (MATLAB Coder).

Using MATLAB Coder

Creating a MATLAB Coder MEX file can substantially accelerate your MATLAB code. It is also a convenient first step in a workflow that ultimately leads to completely standalone code. When you create a MEX file, it runs in the MATLAB environment. Its inputs and outputs are available for inspection just like any other MATLAB variable. You can then use MATLAB tools for visualization, verification, and analysis.

The simplest way to generate MEX files from your MATLAB code is by using the `codegen` function at the command line. For example, if you have an existing function, `myfunction.m`, you can type the commands at the command line to compile and run the MEX function. `codegen` adds a platform-specific extension to this name. In this case, the `"mex"` suffix is added.

```
codegen myfunction.m
myfunction_mex;
```

Within your code, you can run specific commands either as generated C code or by using the MATLAB engine. In cases where an isolated command does not yet have code generation support, you can use the `coder.extrinsic` command to embed the command in your code. This means that the generated code reenters the MATLAB environment when it needs to run that particular command. This is also useful if you want to embed commands that cannot generate code (such as plotting functions).

To generate standalone executables that run independently of the MATLAB environment, create a MATLAB Coder project inside the MATLAB Coder Integrated Development Environment (IDE). Alternatively, you can call the `codegen` command in the command line environment with appropriate configuration parameters. A standalone executable requires you to write your own `main.c` or `main.cpp` function. See “Generating Standalone C/C++ Executables from MATLAB Code” (MATLAB Coder) for more information.

C/C++ Compiler Setup

Before using `codegen` to compile your code, you must set up your C/C++ compiler. For 32-bit Windows platforms, MathWorks® supplies a default compiler with MATLAB. If your installation does not include a default compiler, you can supply your own compiler. For the current list of supported compilers, see Supported and Compatible Compilers on the MathWorks website. Install a compiler that is suitable for your platform, then read “Setting Up the C or C++ Compiler” (MATLAB Coder).

After installation, at the MATLAB command prompt, run `mex -setup`. You can then use the `codegen` function to compile your code.

Functions and System Objects That Support Code Generation

For an alphabetized list of features supporting C/C++ code generation, see Satellite Communications Toolbox – Functions and Objects Filtered by C/C++ Code Generation.

See Also

Functions
`codegen` | `mex`

More About

- “Code Generation Workflow” (MATLAB Coder)
- Generate C Code from MATLAB Code Video