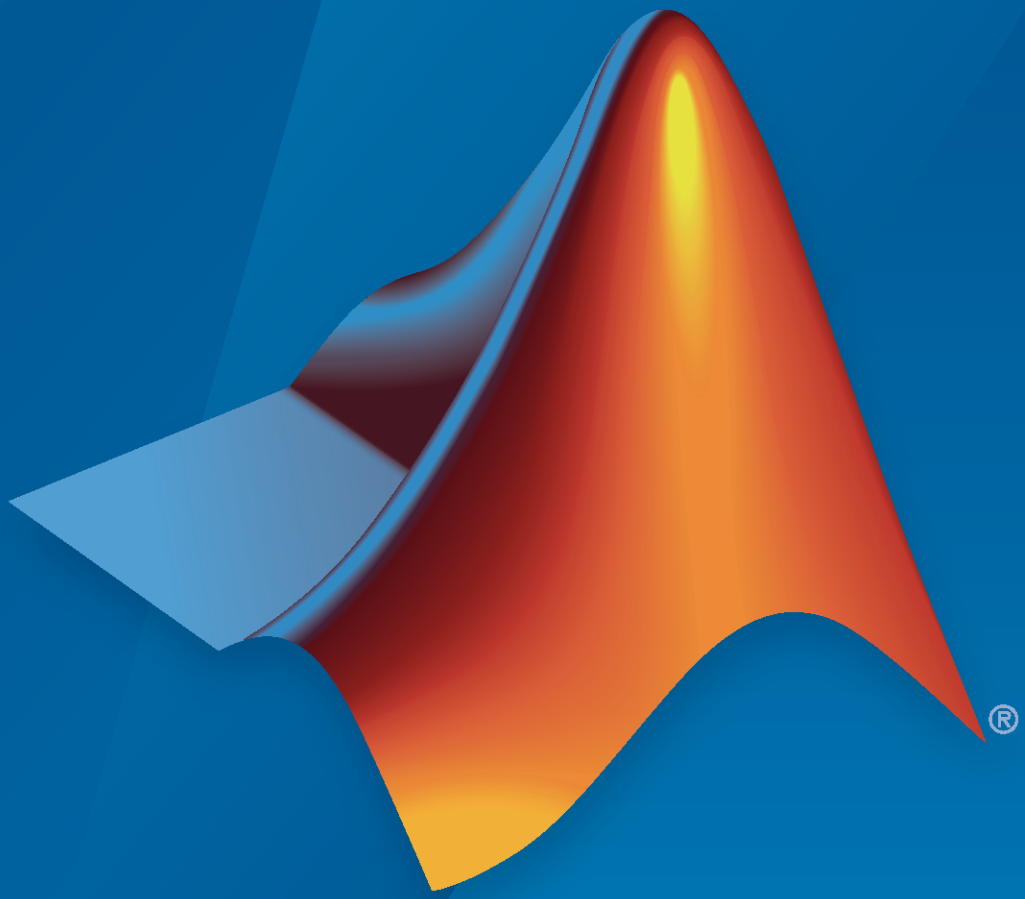
Satellite Communications Toolbox

User's Guide



MATLAB®

R2021a

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)

Contents

1 Satellite Scenario Generation

Multi-Hop Satellite Communications Link Between Two Ground Stations 1-2

Satellite Constellation Access to a Ground Station 1-16

Comparison of Orbit Propagators 1-30

Modeling Satellite Constellations using Ephemeris Data 1-38

Estimate GNSS Receiver Position with Simulated Satellite Constellations 1-48

2 Signal Transmission

GPS Waveform Generation 2-2

3 RF Propagation and Channel Models

Simulate and Visualize a Land Mobile-Satellite Channel 3-2

4 End-to-End Simulation

End-to-End CCSDS Telecommand Simulation with RF Impairments and Corrections 4-2

End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections 4-13

End-to-End DVB-S2 Simulation with RF Impairments and Corrections 4-22

End-to-End DVB-S2X Simulation with RF Impairments and Corrections for Regular Frames **4-38**

5 Code Generation and Deployment

What is C Code Generation from MATLAB? **5-2**
Using MATLAB Coder **5-2**
C/C++ Compiler Setup **5-2**
Functions and System Objects That Support Code Generation **5-3**

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 at MathWorks India, and the second ground station is located at MathWorks Australia. The link is routed via two satellites (MathWorks Sat 1 and MathWorks Sat 2). Each satellite acts as regenerative repeaters. 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 MathWorks India can send data to MathWorks Australia are determined.

Create Satellite Scenario

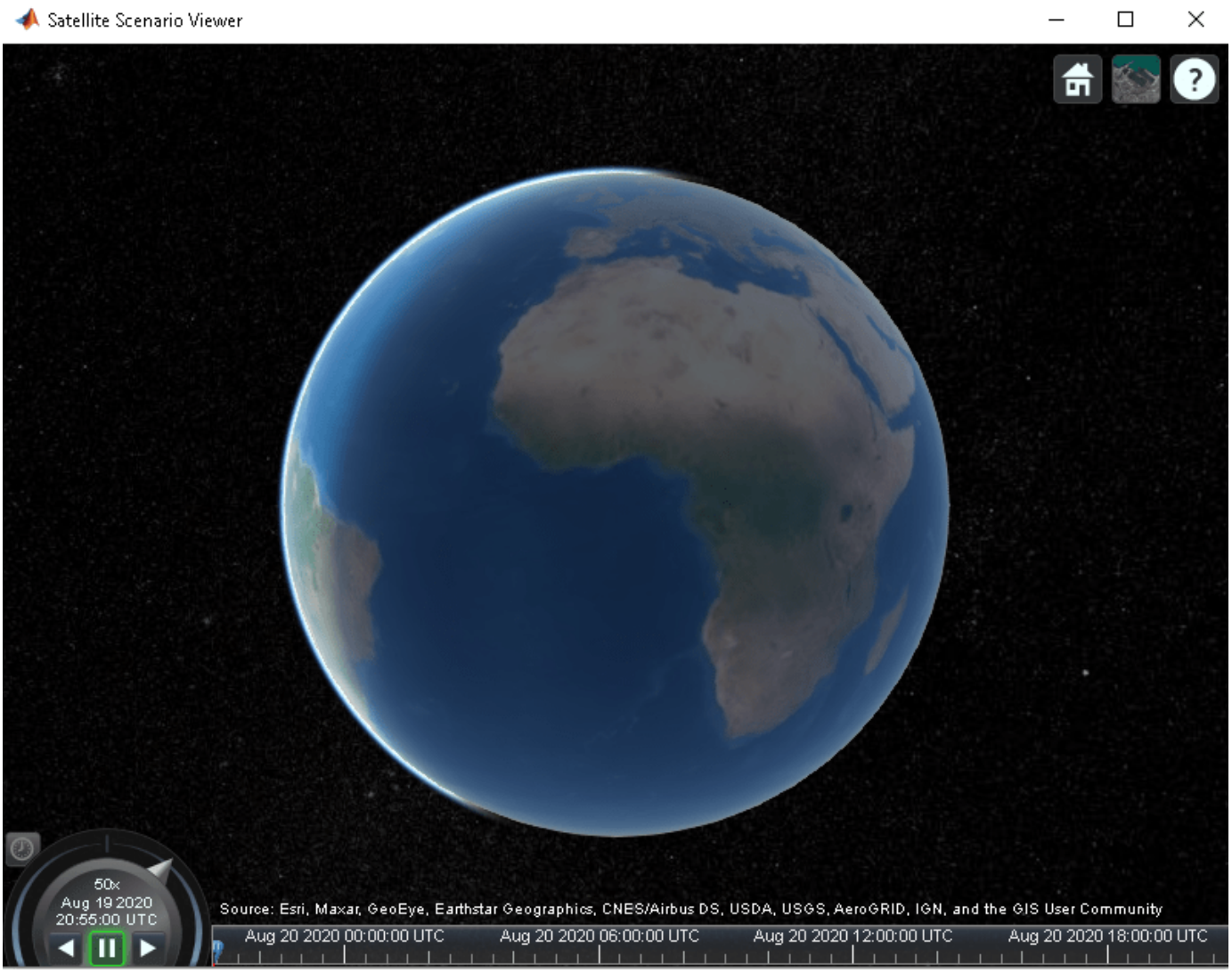
Use `satelliteScenario` to create a satellite scenario. Use `datetime` to define the start time and stop time of the scenario. Set the sample time to 60 seconds.

```
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

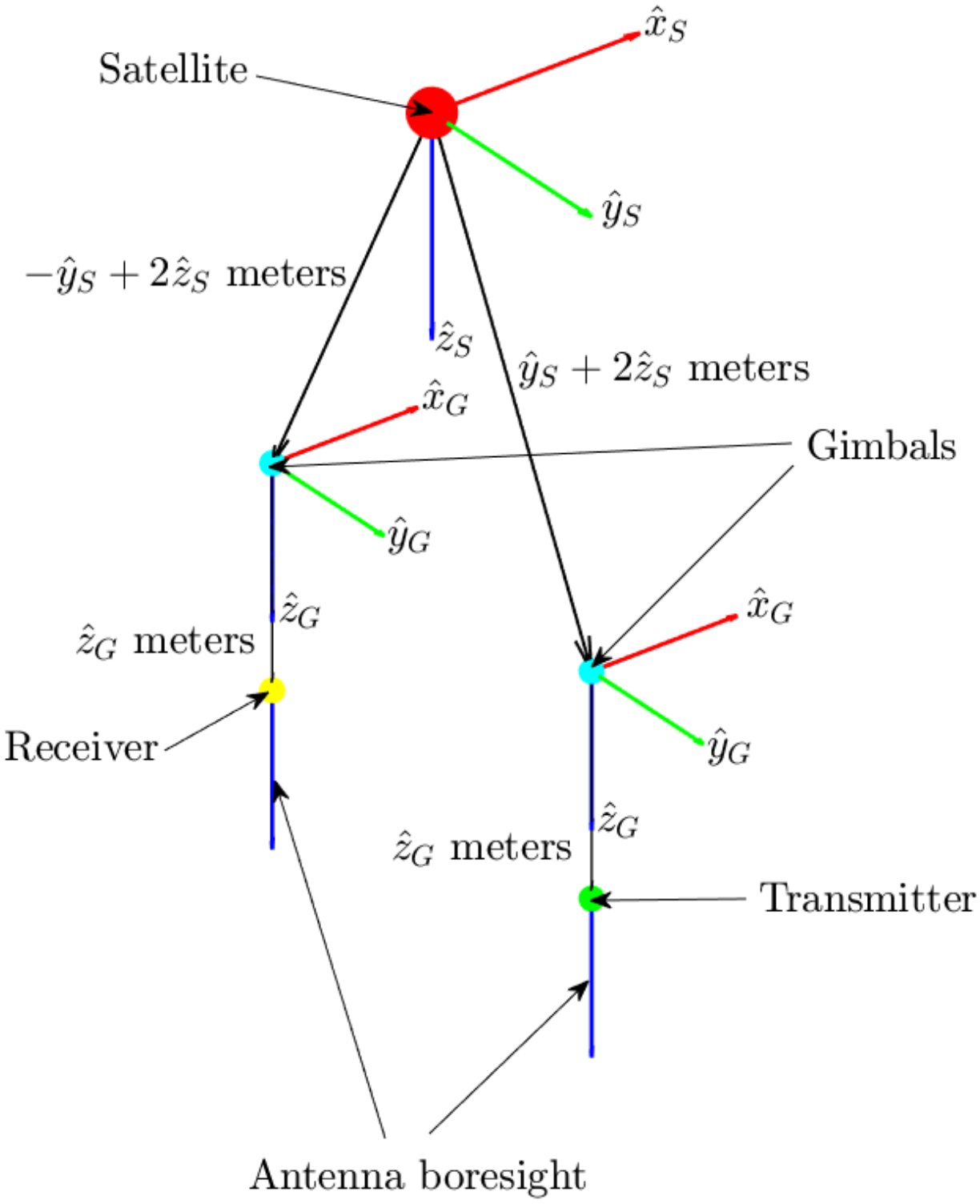
Use `satellite` to add MathWorks Sat 1 and MathWorks Sat 2 satellites to the scenario by specifying their Keplerian orbital elements corresponding to the scenario start time.

```
semiMajorAxis = 10000000;          % meters
eccentricity = 0;
inclination = 0;                   % degrees
rightAscensionOfAscendingNode = 0; % degrees
argumentOfPeriapsis = 0;           % degrees
trueAnomaly = 0;                   % degrees
mwSat1 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
```

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

semiMajorAxis = 10000000;            % meters
eccentricity = 0;
inclination = 30;                    % degrees
rightAscensionOfAscendingNode = 120; % degrees
argumentOfPeriapsis = 0;             % degrees
trueAnomaly = 300;                   % degrees
mwSat2 = satellite(sc, ...
    semiMajorAxis, ...
    eccentricity, ...
    inclination, ...
    rightAscensionOfAscendingNode, ...
    argumentOfPeriapsis, ...
    trueAnomaly, ...
    "Name","MathWorks Sat 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

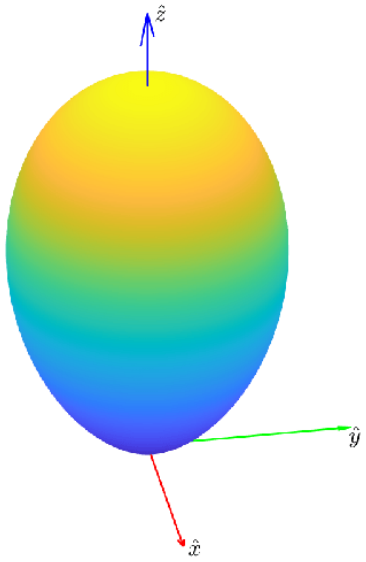
```
gimbalMWSat1Tx = gimbal(mwSat1, ...
    "MountingLocation",[0;1;2]);  % meters
gimbalMWSat2Tx = gimbal(mwSat2, ...
    "MountingLocation",[0;1;2]);  % meters
gimbalMWSat1Rx = gimbal(mwSat1, ...
    "MountingLocation",[0;-1;2]); % meters
gimbalMWSat2Rx = gimbal(mwSat2, ...
    "MountingLocation",[0;-1;2]); % 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 `gimbalMWSat1Rx` and `gimbalMWSat2Rx`. 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.

```
mwSat1Rx = receiver(gimbalMWSat1Rx, ...
    "MountingLocation",[0;0;1], ...      % meters
    "GainToNoiseTemperatureRatio",3, ... % decibels/Kelvin
    "RequiredEbNo",4);                   % decibels
mwSat2Rx = receiver(gimbalMWSat2Rx, ...
    "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(mwSat1Rx, ...
    "DishDiameter",0.5); % meters
gaussianAntenna(mwSat2Rx, ...
    "DishDiameter",0.5); % meters
```

Use `transmitter` to add a transmitter to the gimbals `gimbalMWSat1Tx` and `gimbalMWSat2Tx`. 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 MathWorks Sat 1 is used in the crosslink for sending data to MathWorks Sat 2 at a frequency of 30 GHz. The transmitter onboard MathWorks Sat 2 is used in the downlink to MathWorks Australia at a frequency of 27 GHz.

```
mwSat1Tx = transmitter(gimbalMWSat1Tx, ...
    "MountingLocation",[0;0;1], ... % meters
    "Frequency",30e9, ...           % hertz
    "Power",15);                    % decibel watts
mwSat2Tx = transmitter(gimbalMWSat2Tx, ...
    "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(mwSat1Tx, ...
    "DishDiameter",0.5); % meters
gaussianAntenna(mwSat2Tx, ...
    "DishDiameter",0.5); % meters
```

Add the Ground Stations

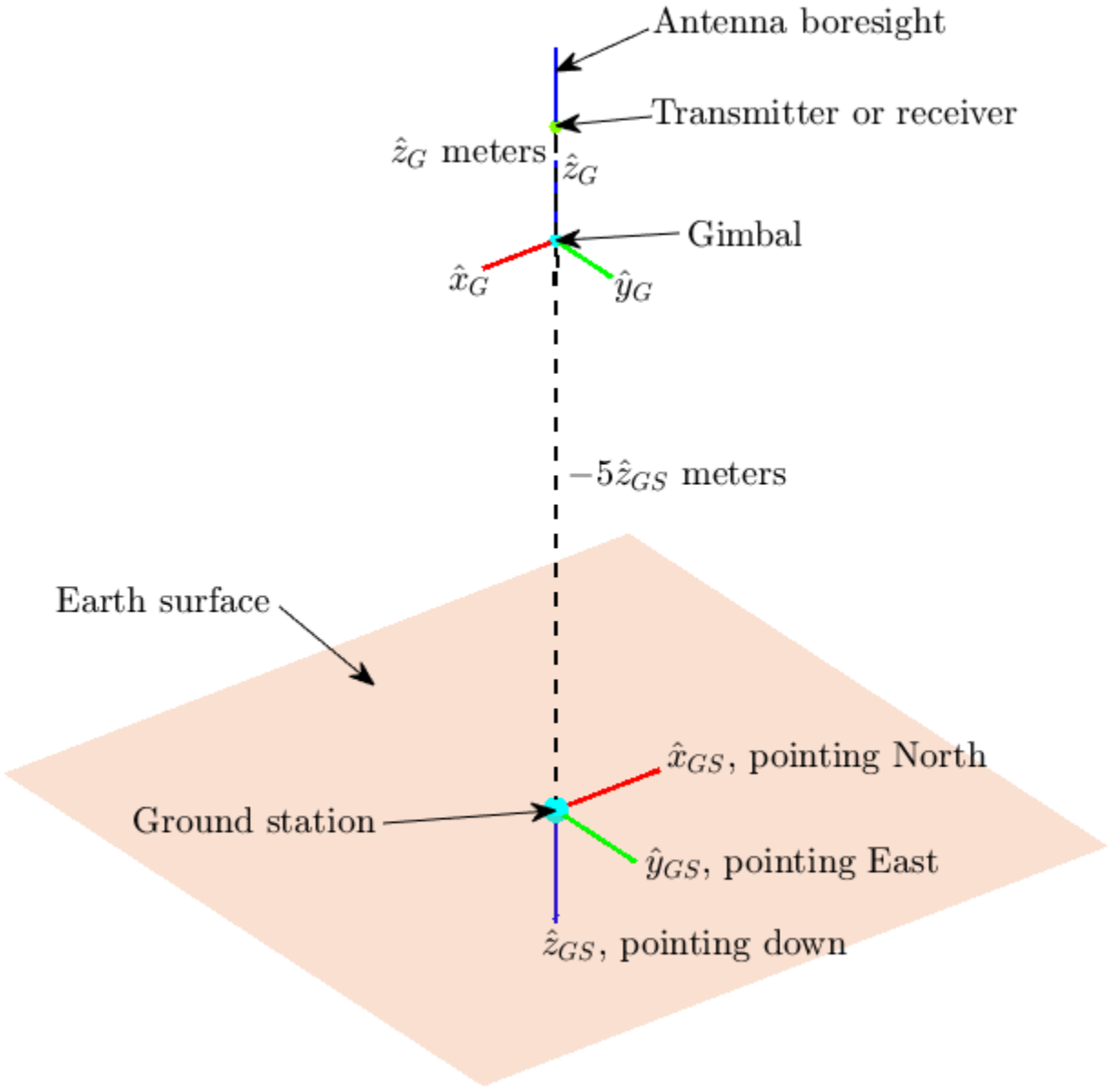
Use `groundStation` to add the ground stations at MathWorks India and MathWorks Australia.

```
latitude = 12.9436963;  % degrees
longitude = 77.6906568; % degrees
mwIndia = groundStation(sc, ...
    latitude, ...
    longitude, ...
    "Name","MathWorks India");

latitude = -33.7974039;  % degrees
longitude = 151.1768208; % degrees
mwAustralia = groundStation(sc, ...
    latitude, ...
    longitude, ...
    "Name","MathWorks Australia");
```

Add Gimbal to Each Ground Station

Use `gimbal` to add a gimbal to MathWorks India and MathWorks Australia. The gimbal at MathWorks India holds a transmitter, and the gimbal at MathWorks Australia 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.




```
gimbalMWIndia = gimbal(mwIndia, ...
    "MountingAngles",[0;180;0], ... % degrees
    "MountingLocation",[0;0;-5]);   % meters
gimbalMWAustralia = gimbal(mwAustralia, ...
```

```
    "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 MathWorks India. The uplink transmitter sends data to MathWorks Sat 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.

```
mwIndiaTx = transmitter(gimbalMWIndia, ...
    "Name","MathWorks India 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(mwIndiaTx, ...
    "DishDiameter",2); % meters
```

Use `receiver` to add a receiver to the gimbal at the MathWorks Australia ground station to receive downlink data from MathWorks Sat 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.

```
mwAustraliaRx = receiver(gimbalMWAustralia, ...
    "Name","MathWorks Australia 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(mwAustraliaRx, ...
    "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 MathWorks India points at MathWorks Sat 1
- The receiver antenna aboard MathWorks Sat 1 points at MathWorks India
- The transmitter antenna aboard MathWorks Sat 1 points at MathWorks Sat 2
- The receiver antenna aboard MathWorks Sat 2 points at MathWorks Sat 1
- The transmitter antenna aboard MathWorks Sat 2 points at MathWorks Australia
- The receiver antenna at MathWorks Australia points at MathWorks Sat 2

```
pointAt(gimbalMWIndia,mwSat1);
pointAt(gimbalMWSat1Rx,mwIndia);
pointAt(gimbalMWSat1Tx,mwSat2);
pointAt(gimbalMWSat2Rx,mwSat1);
```

```
pointAt(gimbalMWSat2Tx,mwAustralia);
pointAt(gimbalMWAustralia,mwSat2);
```

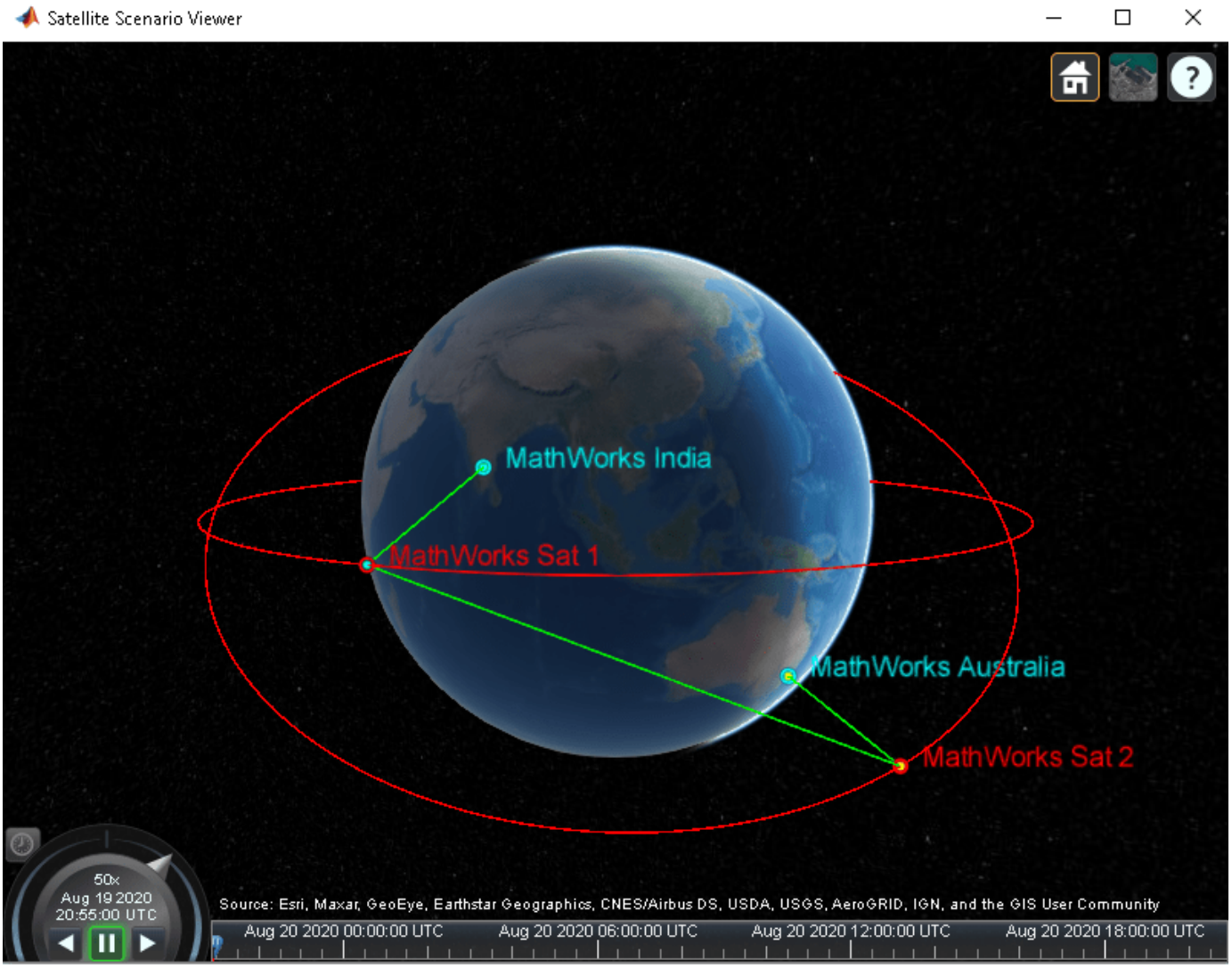
When a target for a gimbal is set, its $\hat{z}_G$ axis will track the target. Since the antenna is on $\hat{z}_G$ and its boresight is aligned with $\hat{z}_G$, the antenna will also track the desired target.

Add Link Analysis and Visualize Scenario

Use `link` to add link analysis to the transmitter at MathWorks India. The link is of regenerative repeater-type that originates at `mwIndiaTx` and ends at `mwAustraliaRx`, and is routed via `mwSat1Rx`, `mwSat1Tx`, `mwSat2Rx` and `mwSat2Tx`.

```
lnk = link(mwIndiaTx,mwSat1Rx,mwSat1Tx,mwSat2Rx,mwSat2Tx,mwAustraliaRx);
```

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

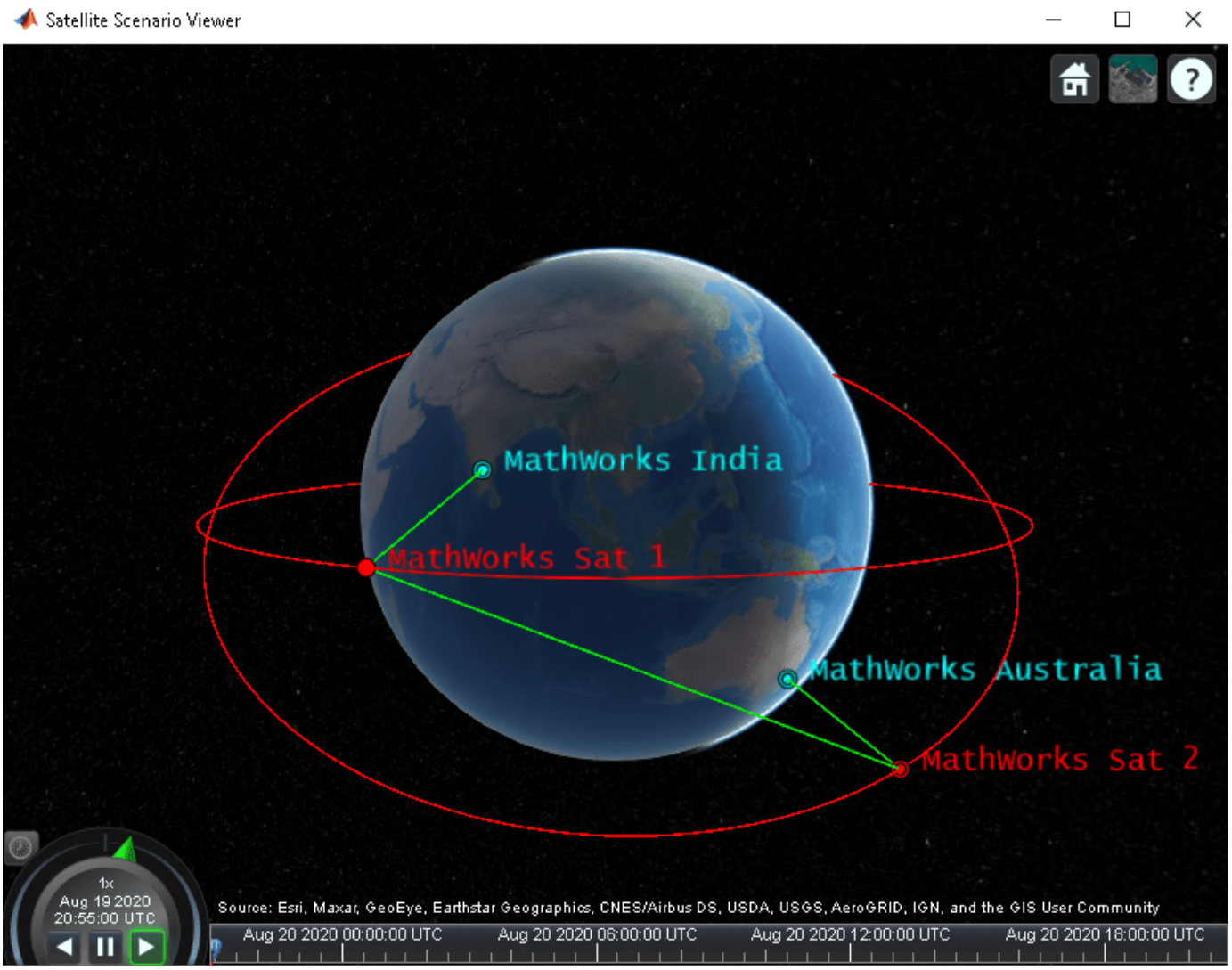
Use the `linkIntervals` method to determine the times when the link is closed. The `linkIntervals` method outputs a table of the start and stop times of link closures that represent the intervals during which MathWorks India can send data to MathWorks Australia. Source and Target are the first and last 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         Sta
    ___________________________________    ___________________________________    ______________    ____

    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          1           19-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          2           19-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          3           20-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          4           20-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          5           20-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          6           20-Aug-2
```

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

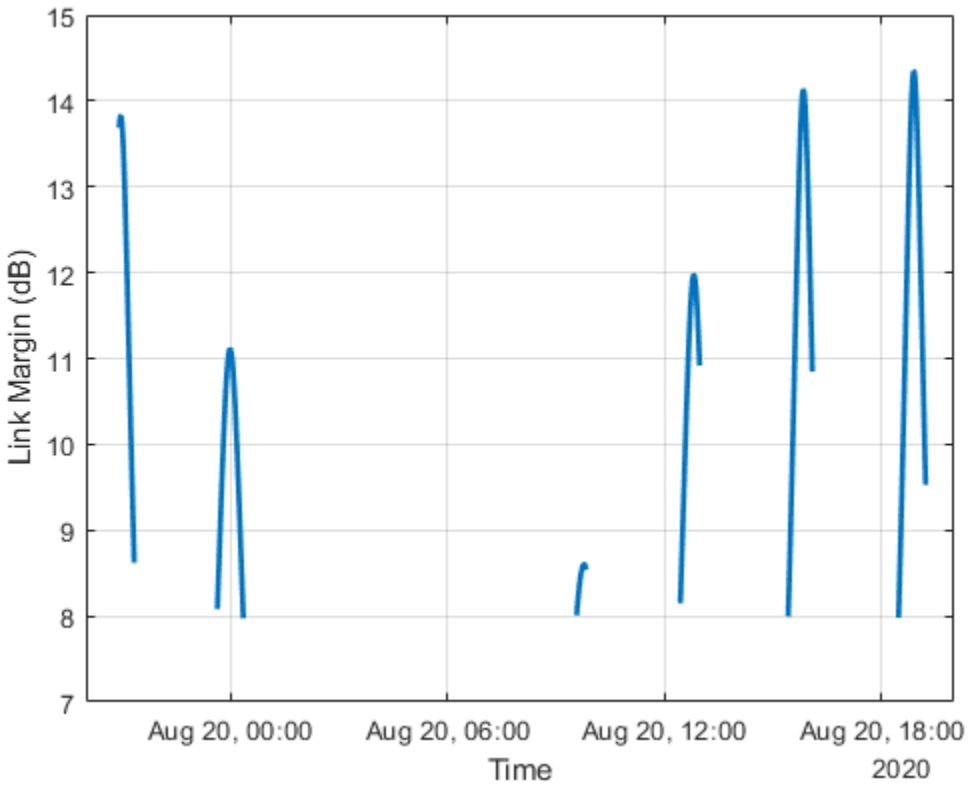
```
play(sc);
```



Plot Link Margin at MathWorks Australia

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, MathWorks Australia Receiver, use `ebno` to get the Eb/No history at the MathWorks Australia 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 - mwAustraliaRx.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, MathWorks Sat 2, was broken. At all other times, the link margin is positive. This implies that MathWorks Sat 2 Transmitter power and MathWorks Australia 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 MathWorks Australia from 1 dB to 10 dB and recompute the link intervals. Increasing `RequiredEbNo` essentially reduces the sensitivity of MathWorks Australia Receiver. This negatively impacts the resultant link closure times and the link margin. The number of closed link intervals drops from six to five, and the duration of the closed link intervals is shorter. Comparing the link margin with by re-calculating and plotting the link margin. With the required EbNo set to 10 dB, the link closure is sometimes limited by the link margin when there is line of sight between adjacent nodes.

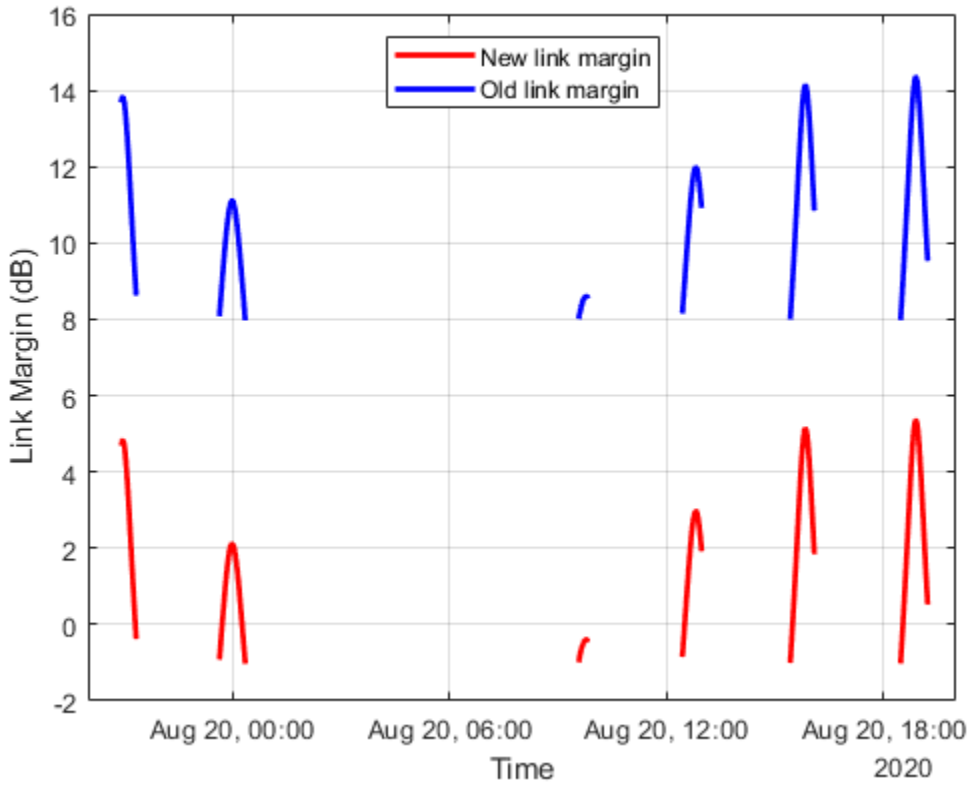
```
mwAustraliaRx.RequiredEbNo = 10; % decibels
linkIntervals(lnk)
```

```
ans=5×8 table
             Source                        Target                  IntervalNumber        Sta
    ___________________________    ___________________________    ______________    ________

    "MathWorks India Transmitter"    "MathWorks Australia Receiver"        1          19-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"        2          19-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"        3          20-Aug-2
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"        4          20-Aug-2
```

```
    "MathWorks India Transmitter"    "MathWorks Australia Receiver"          5           20-Aug-2
```

Additionally, 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. At certain instances, the link margin is negative, signifying that there are times when the link does get broken at MathWorks Australia Receiver, even if it has line of sight to MathWorks Sat 2.

```
[e, newTime] = ebno(lnk);
newMargin = e - mwAustraliaRx.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
`access` | `conicalSensor` | `groundStation` | `receiver` | `satellite` | `satelliteScenario` | `satelliteScenarioViewer` | `transmitter`

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

Related Examples

- “Satellite Constellation Access to a Ground Station” on page 1-16
- “Comparison of Orbit Propagators” on page 1-30
- “Modeling Satellite Constellations using Ephemeris Data” on page 1-38
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-48
- “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, located at MathWorks Natick. Each satellite has a camera with a field of view of 90 degrees. The entire constellation of satellites is tasked with photographing MathWorks Natick, 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 MathWorks Natick 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 MathWorks Natick 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 MathWorks Natick. It is also required to determine the percentage of time during this interval when at least one satellite's camera can see MathWorks Natick. 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: []
    GroundStations: []
          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 =
  1×40 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 =
  1×40 ConicalSensor array with properties:

    Name
    ID
    MountingLocation
    MountingAngles
    MaxViewAngle
    Accesses
    FieldOfView
```

Define MathWorks Natick Geographical Site in the Satellite Scenario

Use `groundStation` to add a ground station, which represents MathWorks Natick. Specify its `MinElevationAngle` to be 30 degrees. If latitude and longitude are not specified, they default to the coordinates of MathWorks Natick.

```
name = "MathWorks Natick";
minElevationAngle = 30; % degrees
mwNatick = groundStation(sc, ...
    "Name",name, ...
    "MinElevationAngle",minElevationAngle)
```

```
mwNatick =
  GroundStation with properties:

                Name: "MathWorks Natick"
                  ID: 81
```

```
            Latitude: 42.3001
           Longitude: -71.3504
            Altitude: 0
    MinElevationAngle: 30
       ConicalSensors: []
              Gimbals: []
        Transmitters: []
           Receivers: []
            Accesses: []
         MarkerColor: [0 1 1]
          MarkerSize: 10
           ShowLabel: 1
       LabelFontColor: [0 1 1]
        LabelFontSize: 15
```

Add Access Analysis Between the Cameras and MathWorks Natick

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

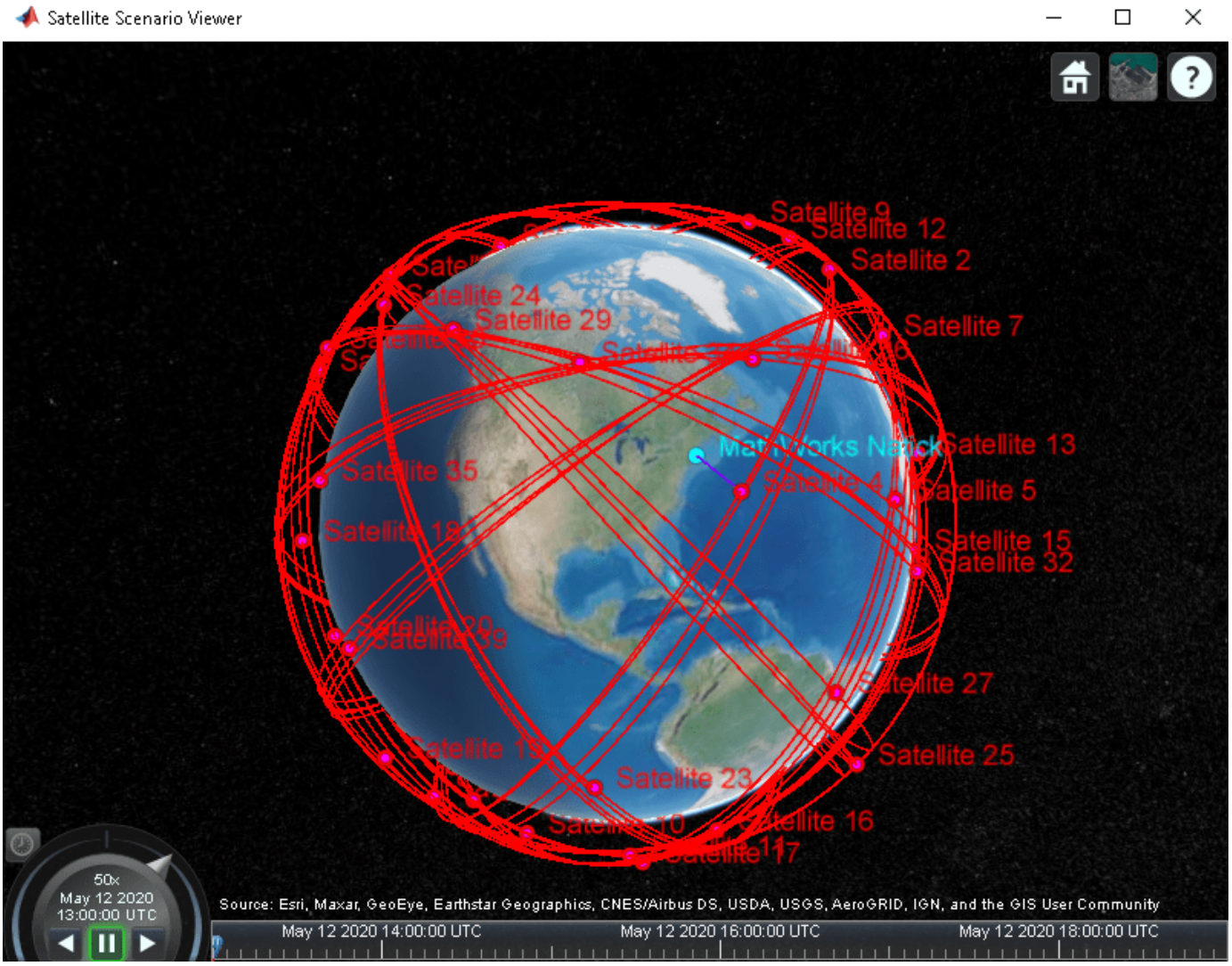
```
for idx = 1:numel(cam)
    access(cam(idx),mwNatick);
end

% 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: [0.5000 0 1]
```

Visualize the Scenario

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

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



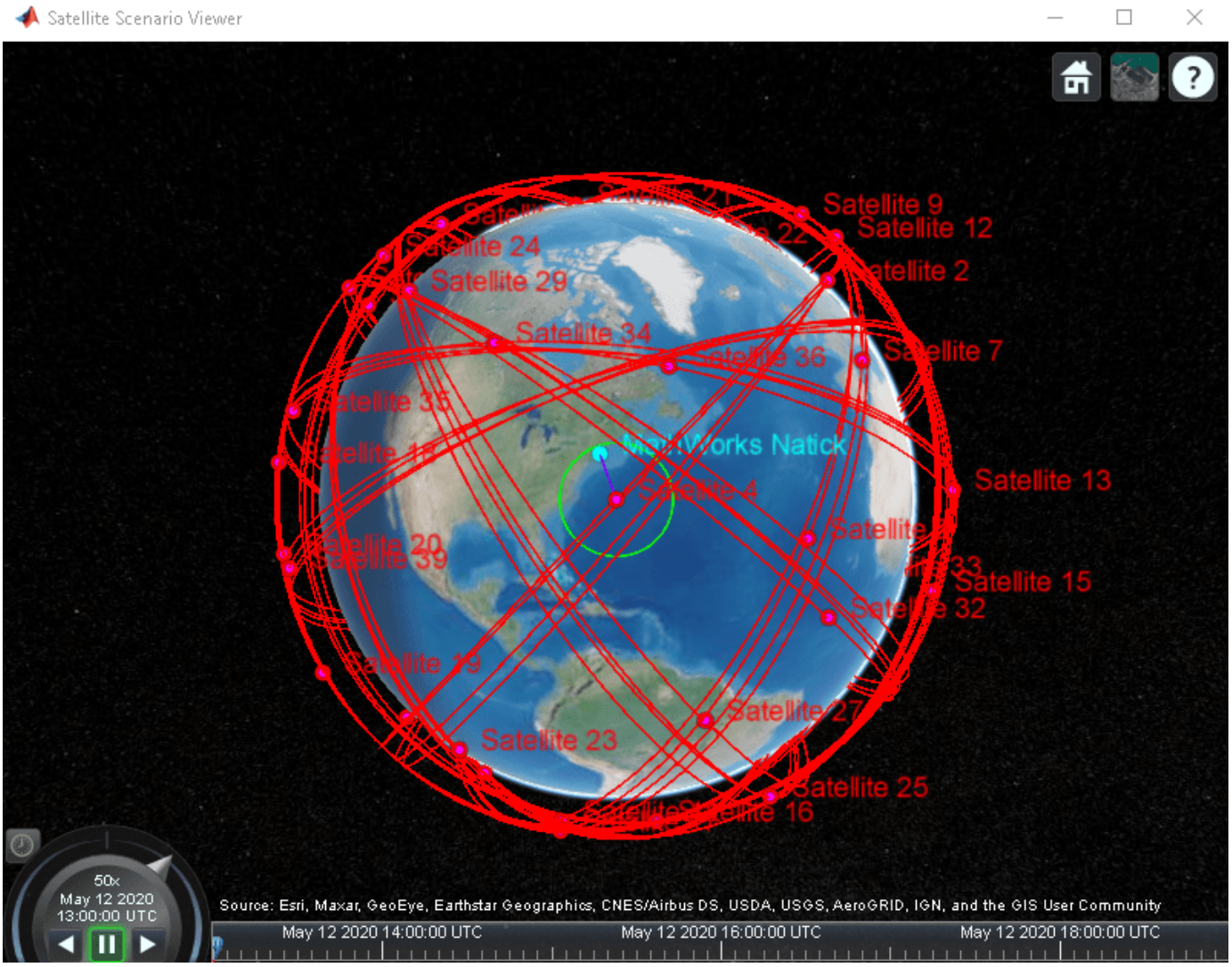
The viewer may be used as a visual confirmation that the scenario has been set up correctly. The violet line indicates that the camera on Satellite 4 and MathWorks Natick have access to one another. This means that MathWorks Natick is inside the camera's field of view and the camera's elevation angle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

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'
```

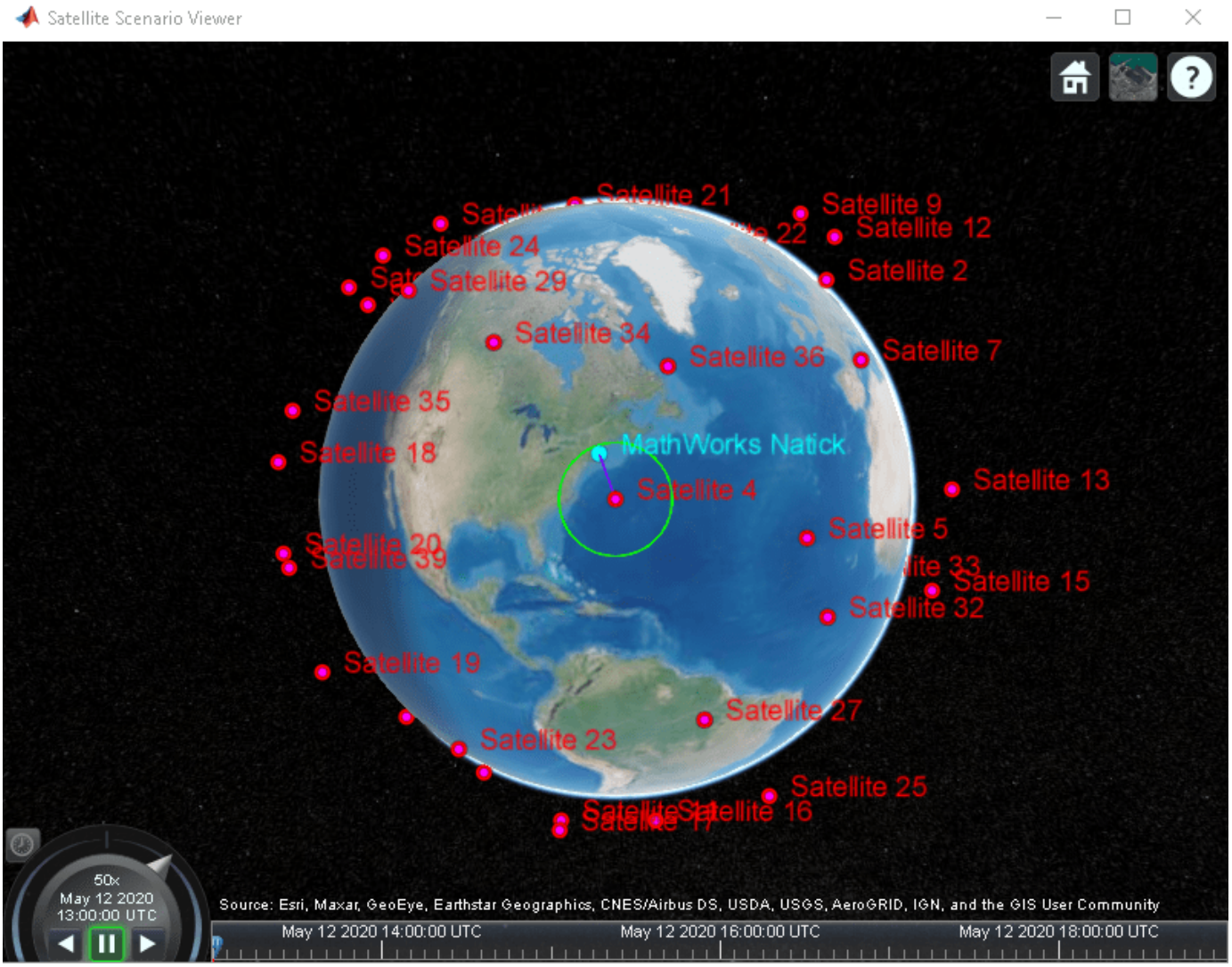


The presence of MathWorks Natick inside the contour is a visual confirmation that it is inside the field of view of the camera onboard Satellite 4.

Customize the Visualizations

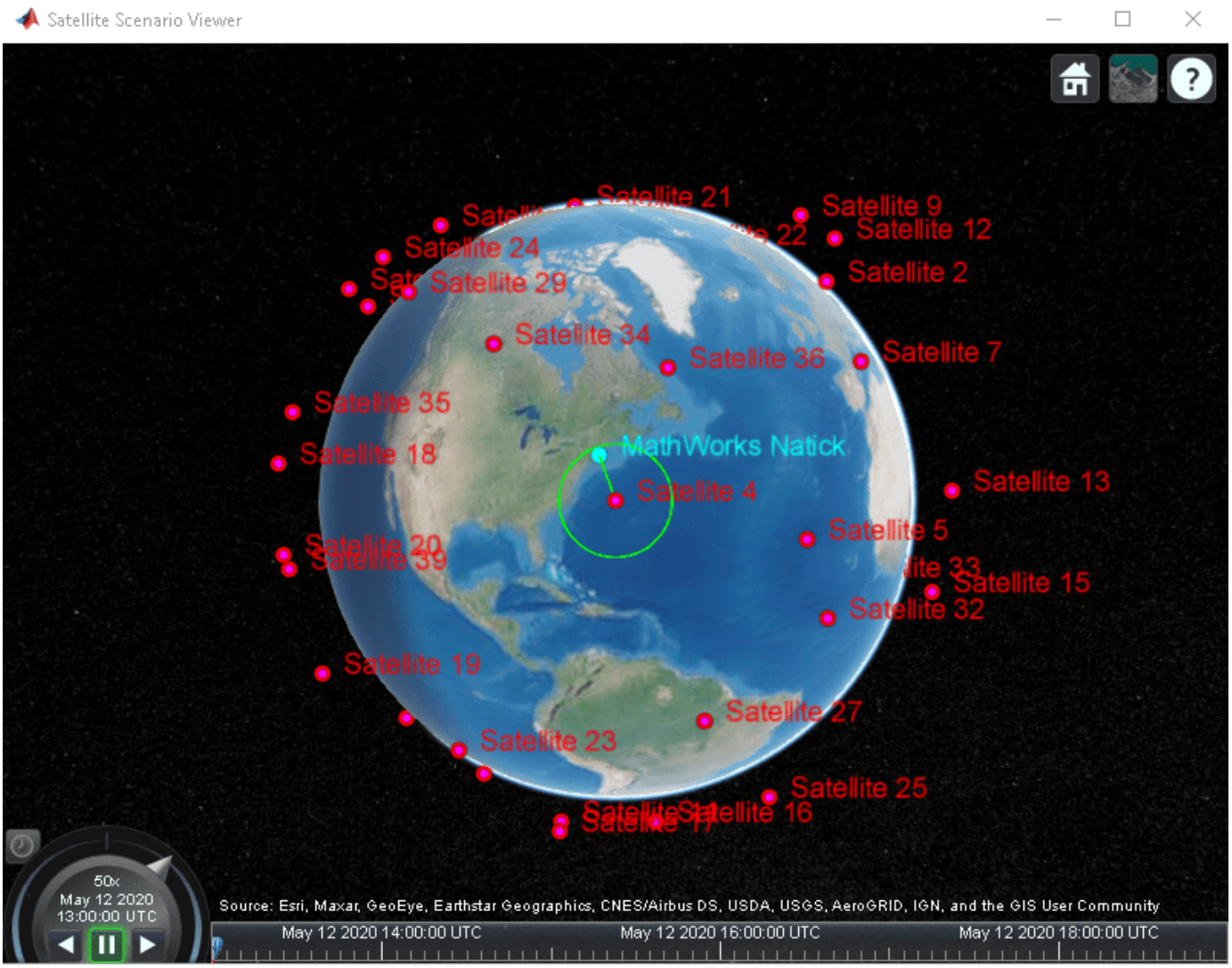
Use `hide` to hide the satellite orbits and declutter the visualization.

```
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 MathWorks Natick

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

```
accessIntervals(ac)
```

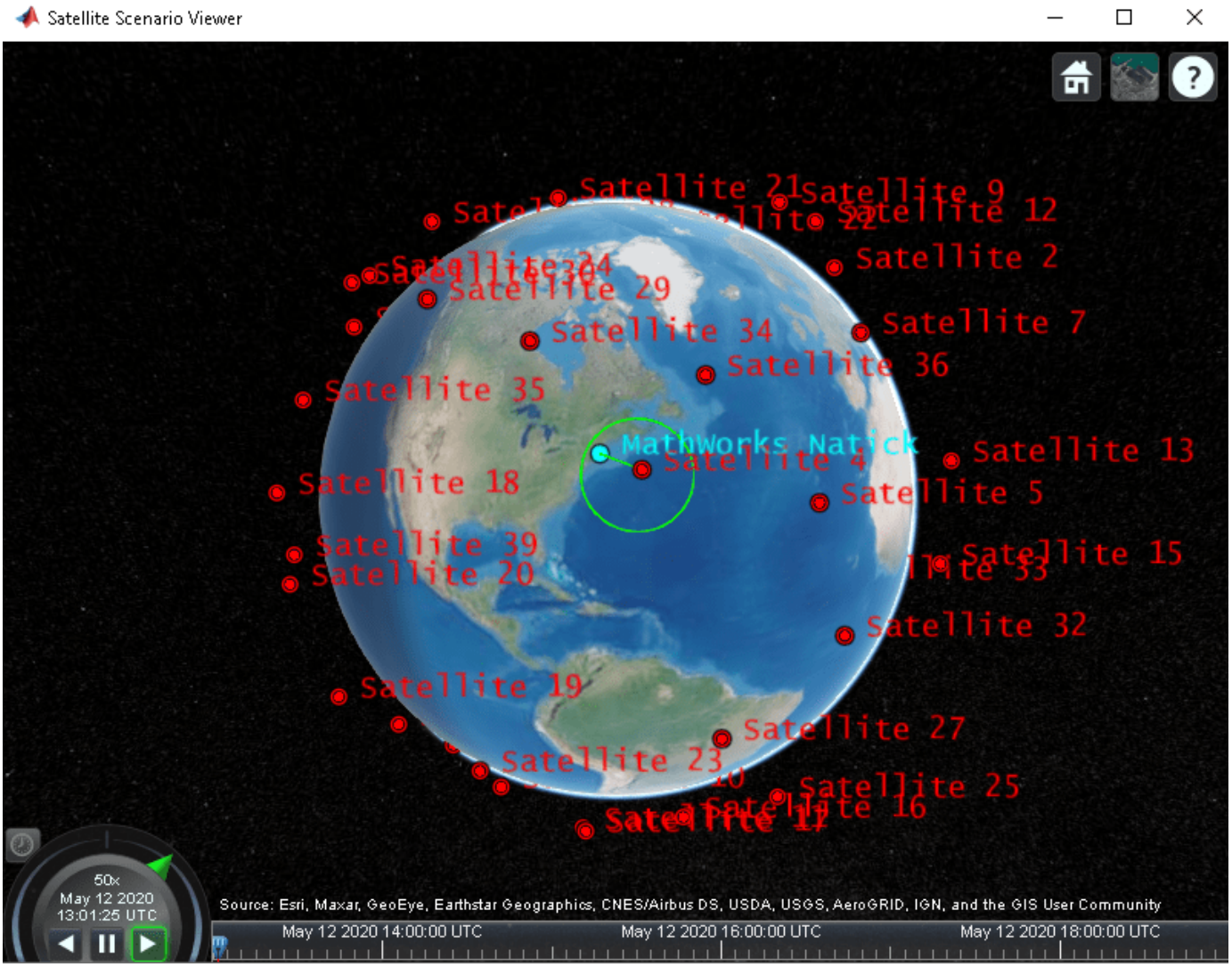
```
ans=30×8 table
          Source                 Target          IntervalNumber        StartTime          
    _____________________    _________________    ______________    ____________________    
    "Satellite 1 Camera"     "MathWorks Natick"         1          12-May-2020 13:36:00    12-
    "Satellite 1 Camera"     "MathWorks Natick"         2          12-May-2020 15:23:00    12-
    "Satellite 2 Camera"     "MathWorks Natick"         1          12-May-2020 14:30:30    12-
    "Satellite 3 Camera"     "MathWorks Natick"         1          12-May-2020 13:28:30    12-
    "Satellite 4 Camera"     "MathWorks Natick"         1          12-May-2020 13:00:00    12-
    "Satellite 4 Camera"     "MathWorks Natick"         2          12-May-2020 14:46:00    12-
    "Satellite 5 Camera"     "MathWorks Natick"         1          12-May-2020 16:28:30    12-
    "Satellite 6 Camera"     "MathWorks Natick"         1          12-May-2020 17:05:30    12-
```

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

The above table consists of the start and end times of each interval during which a given camera can photograph MathWorks Natick. 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 MathWorks Natick

```
play(sc);
```



Calculate System-Wide Access Percentage

In addition to determining the times when each camera can photograph MathWorks Natick, 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 MathWorks Natick. This is computed as follows:

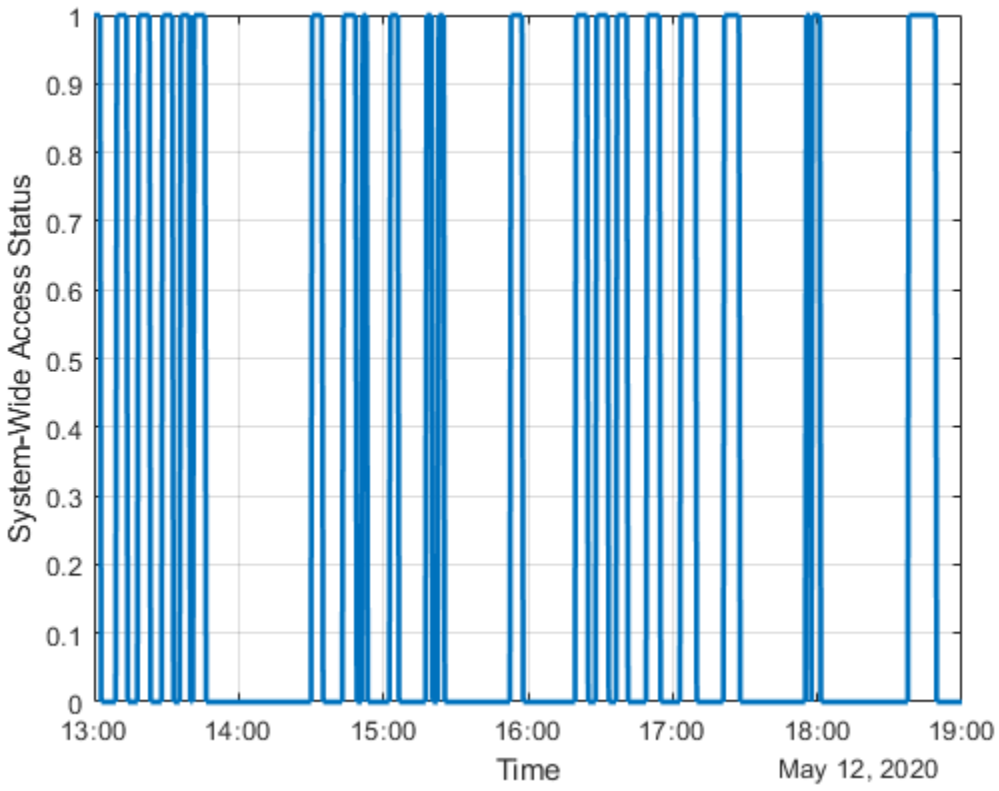
- For each camera, calculate the access status history to MathWorks Natick using `accessStatus`. For a given camera, this is a row vector of logicals, where each element in the vector represents the access status corresponding to a given time sample. A value of `True` indicates that the camera can photograph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

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 MathWorks Natick. 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 MathWorks Natick

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, MathWorks Natick goes outside the field of view of the cameras before their elevation angle dips below 30 degrees. Therefore, the system-wide access percentage is limited by the field of view of the cameras.

If instead the cameras always point at MathWorks Natick, 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 MathWorks Natick's `MinElevationAngle` as opposed to the cameras' field of view. In the former case, the access intervals began and ended when MathWorks Natick 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 MathWorks Natick, 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 MathWorks Natick. As the cameras are aligned with the yaw axis, they too will point at MathWorks Natick. Use `pointAt` to make each satellite's yaw axis track MathWorks Natick.

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

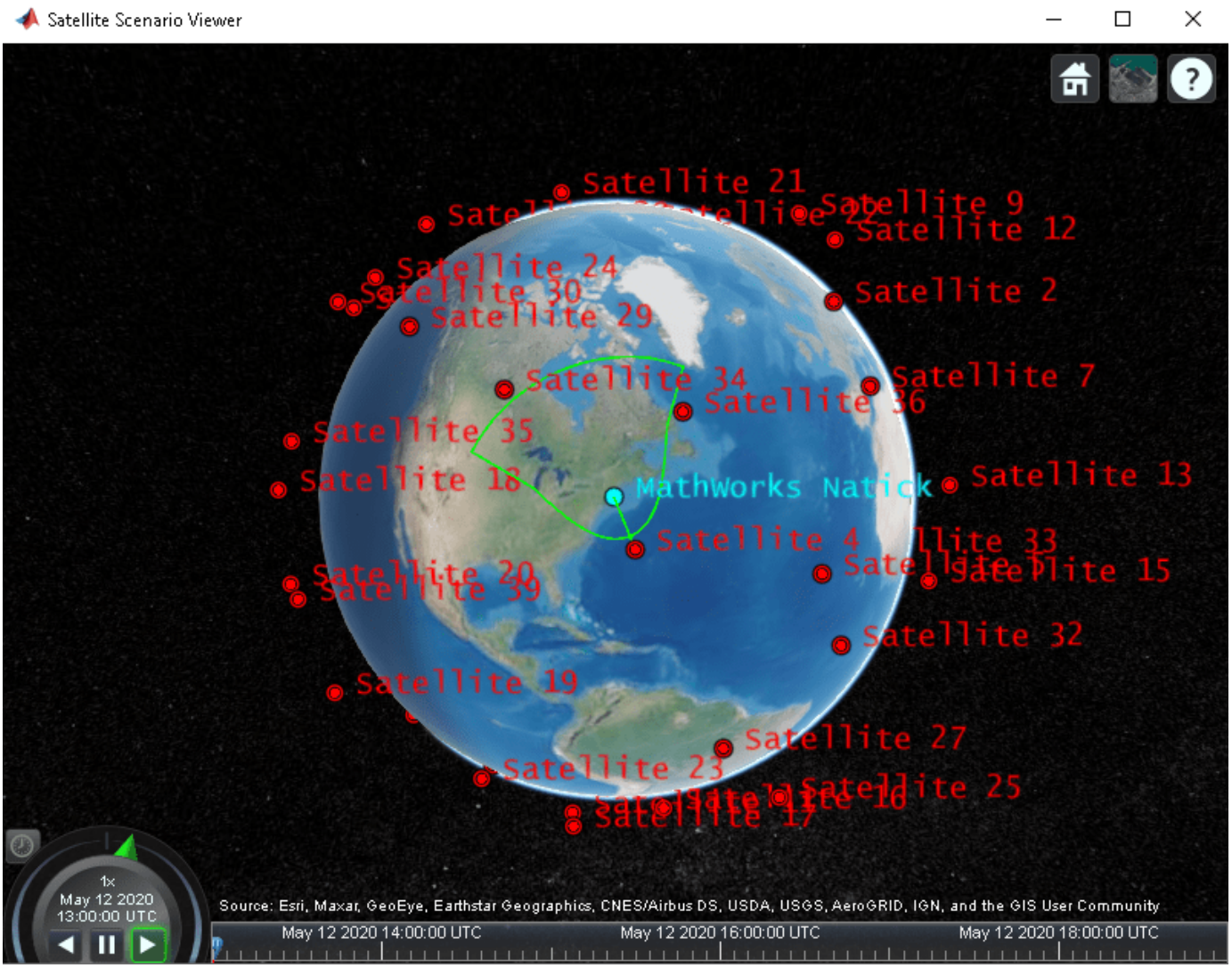
Re-calculate the system-wide access percentage.

```
% 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-MathWorks Natick
        % 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 35%. This is the result of the cameras continuously pointing at MathWorks Natick. 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 MathWorks Natick.

Exploring the Example

This example demonstrated how to determine the times at which cameras onboard satellites in a constellation can photograph a geographical site (MathWorks Natick). The cameras were modeled using conical sensors and access analysis was used to calculate the times. 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 MathWorks Natick. It was seen that these results depended on which direction the cameras were pointing.

These results are also a function of:

- Orbit of the satellites
- `MinElevationAngle` of MathWorks Natick
- 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 MathWorks Natick

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 MathWorks Natick.

See Also

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

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

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Comparison of Orbit Propagators” on page 1-30
- “Modeling Satellite Constellations using Ephemeris Data” on page 1-38
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-48
- “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")
```

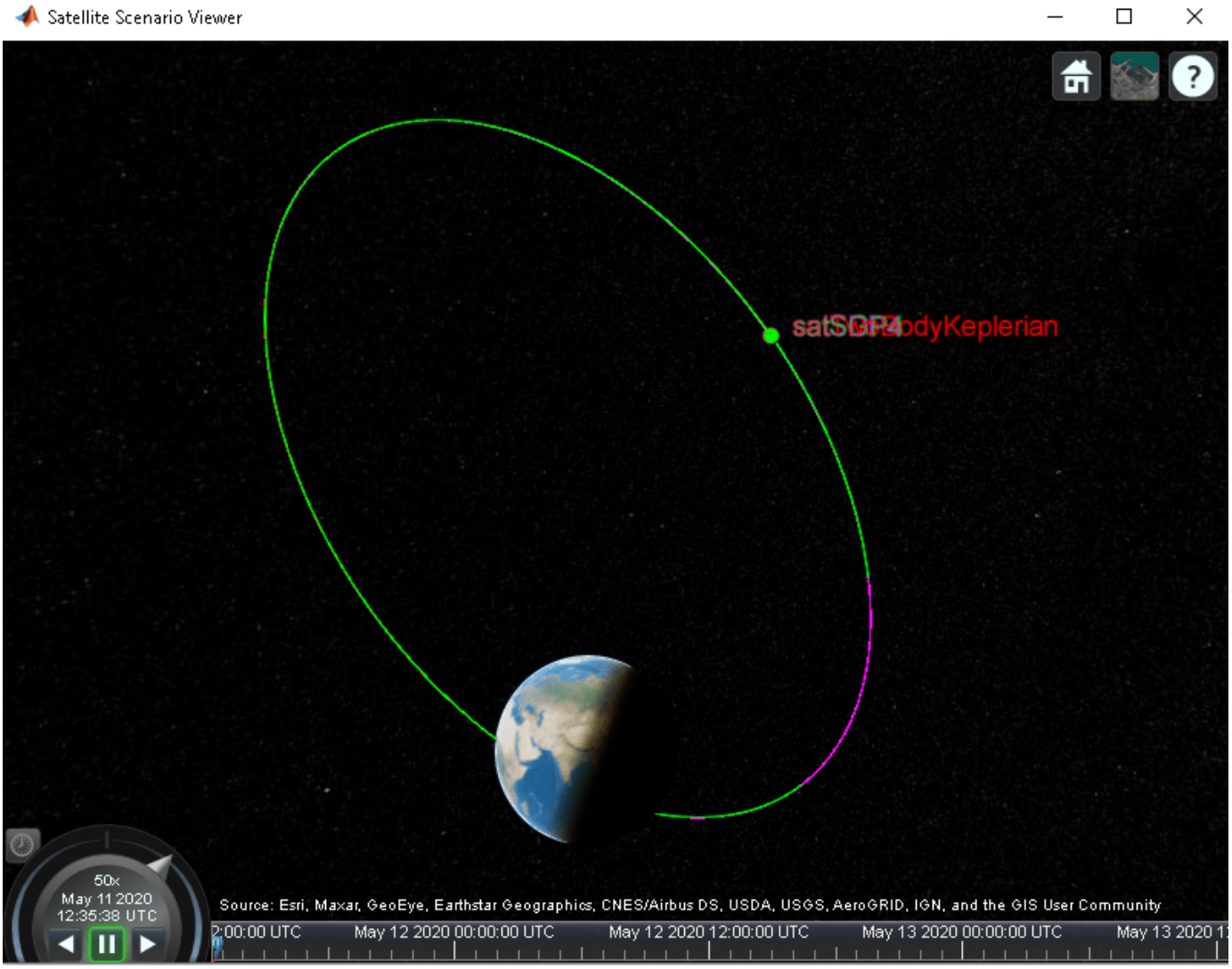
```
satSDP4 =
  Satellite with properties:

               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.

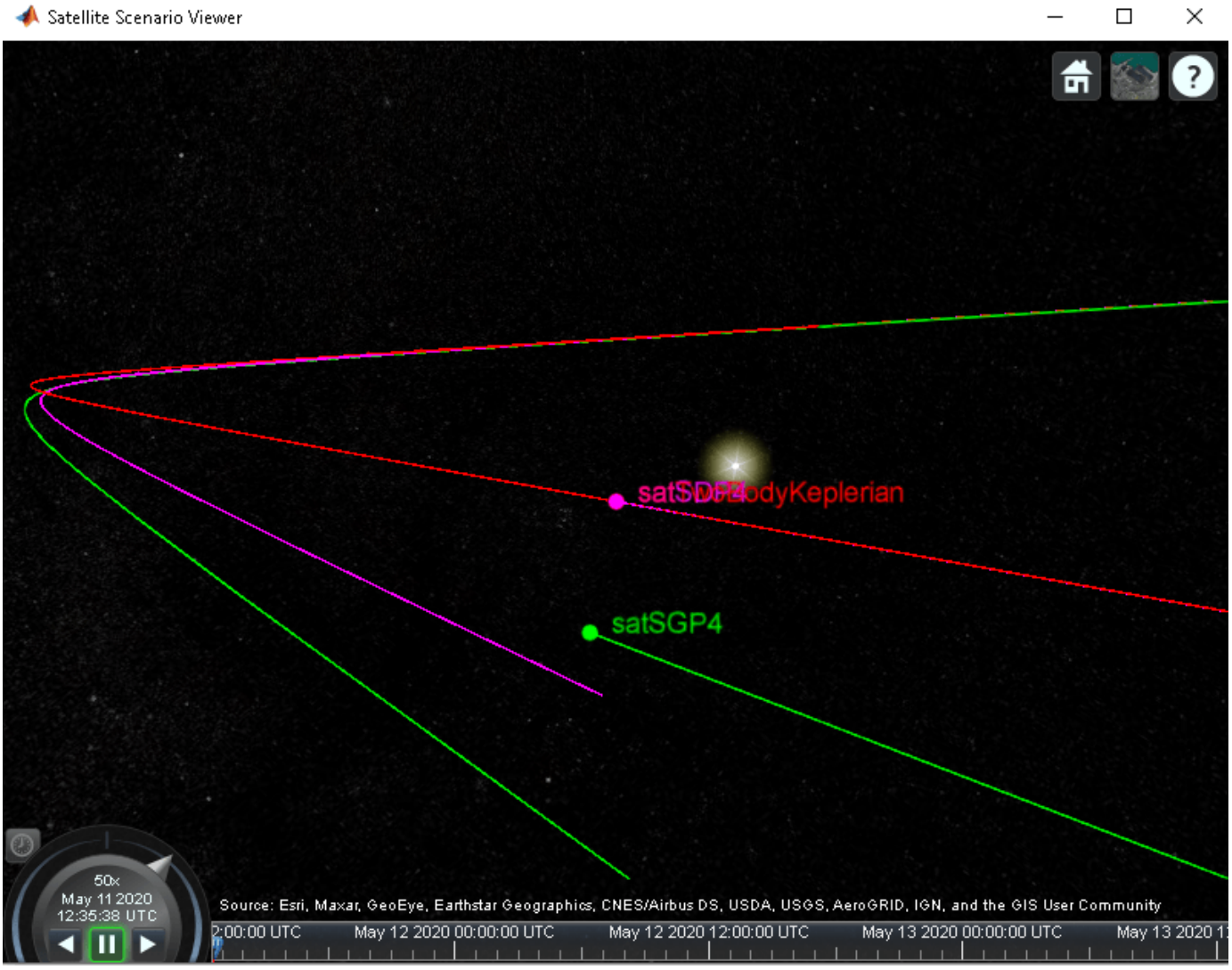
```
v = satelliteScenarioViewer(sc);
satSGP4.MarkerColor = [0 1 0];
satSGP4.Orbit.LineColor = [0 1 0];
satSGP4.LabelFontColor = [0 1 0];
satSDP4.MarkerColor = [1 0 1];
satSDP4.Orbit.LineColor = [1 0 1];
satSDP4.LabelFontColor = [1 0 1];
```



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

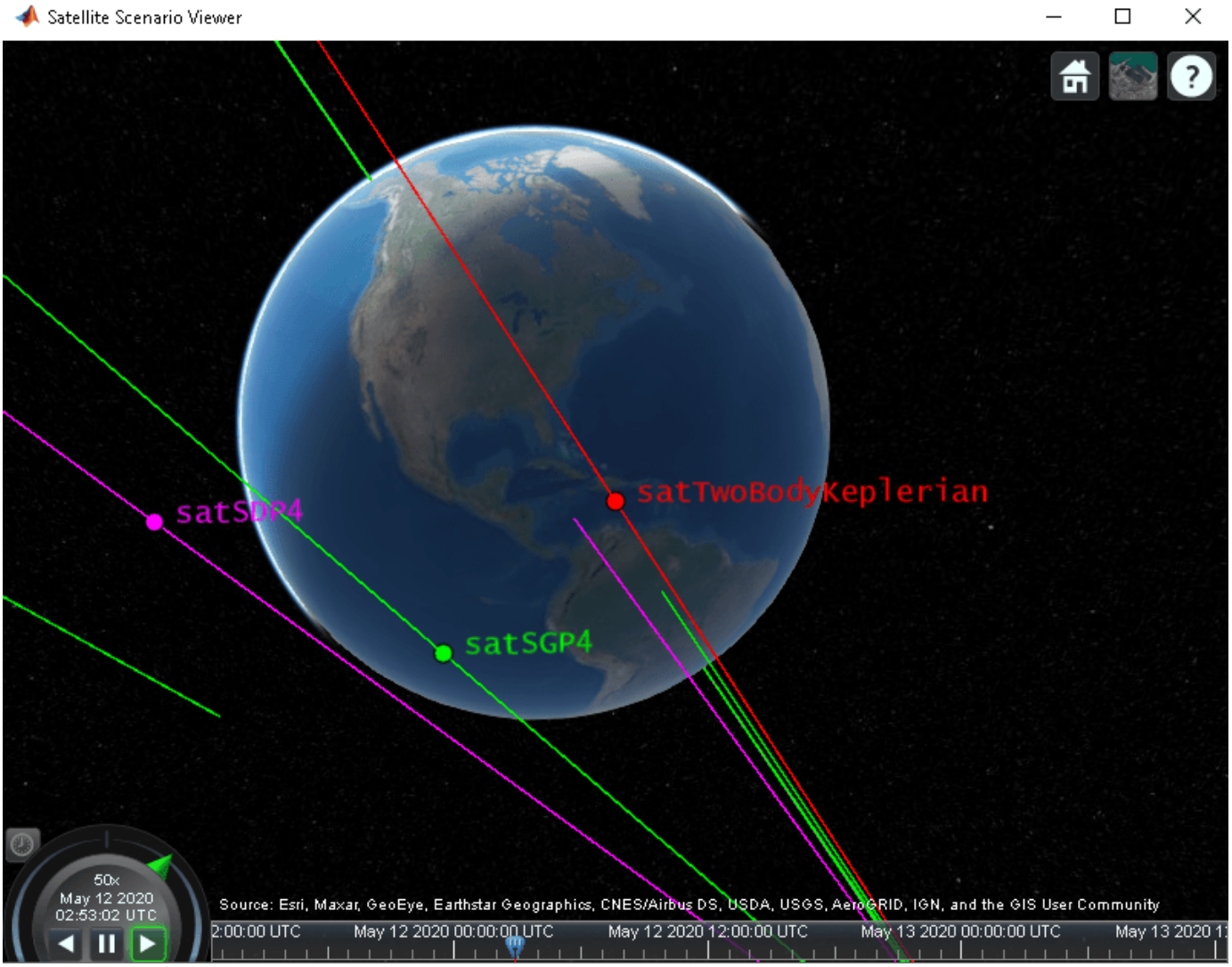
Visualize the movement of the satelliles by using the `play` function on the satellite scenario. The `play` function simulates the satellite scenario from the specified `StartTime` to `StopTime` using a step size specified by `SampleTime`, and plays the results on the satellite scenario viewer.

```
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);
```

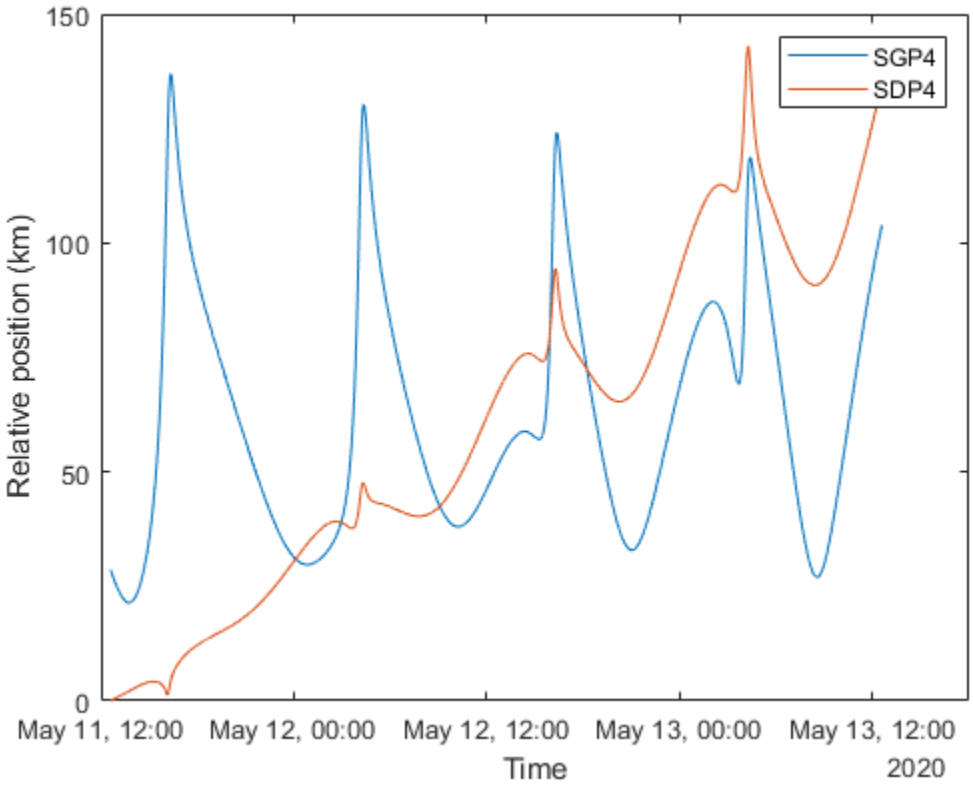
Plot Magnitude of Relative Position with Respect to Two-Body-Keplerian Prediction

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 calcluated 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.

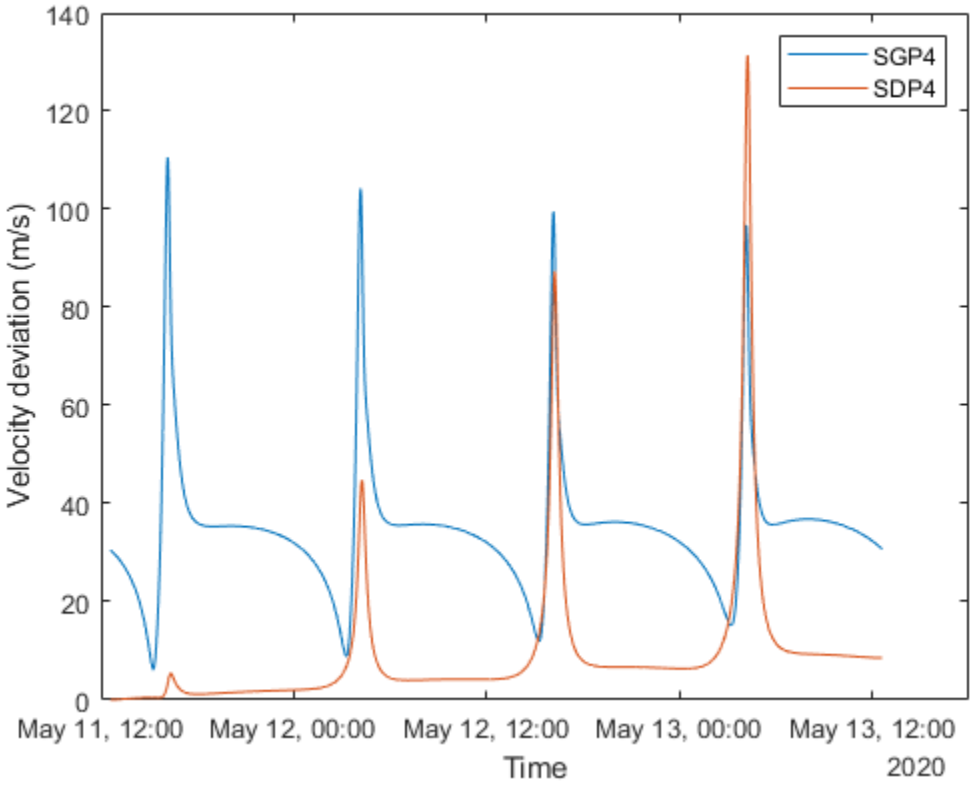
Plot Magnitude of Relative Velocity with Respect to Two-Body-Keplerian Prediction

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

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

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-16
- “Modeling Satellite Constellations using Ephemeris Data” on page 1-38
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-48
- “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 the ESA Galileo 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( ...
    23222e3 * 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 anomoly (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)
    __________    ______    _______    _______    _______    _______

    2.3222e+07    0.0005      56         350          0          0
    2.3222e+07    0.0005      56         350          0         45
    2.3222e+07    0.0005      56         350          0         90
    2.3222e+07    0.0005      56         350          0        135
    2.3222e+07    0.0005      56         350          0        180
    2.3222e+07    0.0005      56         350          0        225
    2.3222e+07    0.0005      56         350          0        270
    2.3222e+07    0.0005      56         350          0        315
    2.3222e+07    0.0005      56         350        120         15
    2.3222e+07    0.0005      56         350        120         60
    2.3222e+07    0.0005      56         350        120        105
    2.3222e+07    0.0005      56         350        120        150
    2.3222e+07    0.0005      56         350        120        195
```

```
    2.3222e+07    0.0005      56         350        120        240
    2.3222e+07    0.0005      56         350        120        285
    2.3222e+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: [79x1 double]
       TimeInfo: [1x1 tsdata.timemetadata]
           Data: [24x3x79 double]
       DataInfo: [1x1 tsdata.datametadata]

  More properties, Methods

mission.Ephemeris.TimeseriesVelITRF

  timeseries

  Common Properties:
           Name: ''
           Time: [79x1 double]
       TimeInfo: [1x1 tsdata.timemetadata]
           Data: [24x3x79 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 =
  1×24 Satellite array with properties:

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

```
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: []
          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)), ...
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=79×3 timetable
           Time                        Satellite_1                                    Satellite_2
    ____________________    ____________________________________    ____________________________________

    30-Nov-2020 22:23:24    1.4317e+07    -1.7969e+07    -3.2957e+06    1.8576e+07    -8.4335e+06
    30-Nov-2020 22:23:34    1.4326e+07    -1.7968e+07    -3.2607e+06    1.8567e+07    -8.4162e+06
    30-Nov-2020 22:24:26    1.4367e+07    -1.7966e+07    -3.0856e+06    1.8519e+07    -8.3288e+06
    30-Nov-2020 22:28:44    1.4562e+07    -1.7939e+07    -2.2062e+06    1.8271e+07     -7.878e+06
    30-Nov-2020 22:40:57    1.4983e+07    -1.7724e+07     3.0748e+05    1.7492e+07     -6.472e+06
    30-Nov-2020 22:54:44    1.5245e+07    -1.7219e+07     3.1374e+06    1.6506e+07    -4.6682e+06
    30-Nov-2020 23:08:36    1.5308e+07    -1.6416e+07     5.9118e+06    1.5446e+07    -2.6463e+06
```

```
    30-Nov-2020 23:24:15    1.5183e+07    -1.5144e+07     8.8865e+06    1.4218e+07    -1.4887e+05
    30-Nov-2020 23:40:33    1.4888e+07    -1.3412e+07     1.1718e+07    1.2964e+07     2.6318e+06
    30-Nov-2020 23:57:25    1.4477e+07    -1.1213e+07     1.4269e+07    1.1748e+07     5.6176e+06
    01-Dec-2020 00:15:24    1.3994e+07    -8.4611e+06     1.6479e+07    1.0588e+07     8.8233e+06
    01-Dec-2020 00:33:57    1.3527e+07    -5.2723e+06     1.8119e+07    9.5707e+06     1.2021e+07
    01-Dec-2020 00:53:11    1.3142e+07    -1.7143e+06     1.9066e+07    8.7131e+06     1.5075e+07
    01-Dec-2020 01:12:46    1.2901e+07     2.0218e+06     1.9203e+07    8.0192e+06     1.7771e+07
    01-Dec-2020 01:32:20    1.2826e+07     5.7119e+06     1.8499e+07     7.449e+06     1.9927e+07
    01-Dec-2020 01:51:55    1.2899e+07     9.1955e+06     1.6986e+07    6.9227e+06     2.1452e+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.

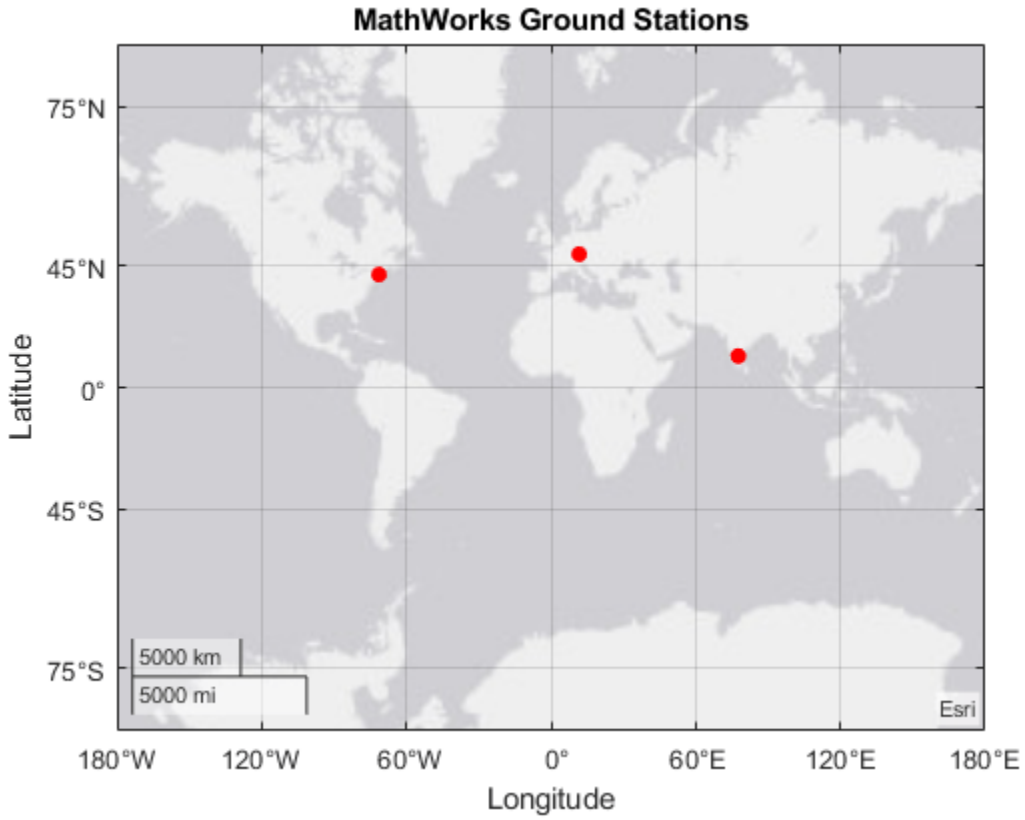
```
set(sat(1:8), "MarkerColor", "red");
set(sat(9:16), "MarkerColor", "blue");
set(sat(17:24), "MarkerColor", "green");
orbit = [sat(:).Orbit];
set(orbit(1:8), "LineColor", "red");
set(orbit(9:16), "LineColor", "blue");
set(orbit(17:24), "LineColor", "green");
```

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 MathWorks locations to compare total constellation access over the 1 day analysis window to different regions of Earth:

- MathWorks headquarters in Natick, Massachusetts, USA (42.30048°, -71.34908°)
- MathWorks München in Ismaning (near München), Germany (48.23206°, 11.68445°)
- MathWorks Bangalore in Bangalore, India (12.94448°, 77.69256°)

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

figure
geoscatter([gsUS.Latitude gsDE.Latitude gsIN.Latitude], ...
    [gsUS.Longitude gsDE.Longitude gsIN.Longitude], "red", "filled")
geolimits([-75 75], [-180 180])
title("MathWorks 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=52×8 table
      Source          Target       IntervalNumber         StartTime              EndTime
    ___________    ____________    ______________    ____________________    ____________________

    "MW Natick"    "GALILEO 1"           1          30-Nov-2020 22:23:24    01-Dec-2020 01:40:24
    "MW Natick"    "GALILEO 2"           1          30-Nov-2020 22:23:24    01-Dec-2020 00:08:24
    "MW Natick"    "GALILEO 3"           1          30-Nov-2020 22:23:24    30-Nov-2020 22:34:24
    "MW Natick"    "GALILEO 12"          1          30-Nov-2020 22:23:24    30-Nov-2020 23:35:24
    "MW Natick"    "GALILEO 13"          1          30-Nov-2020 22:23:24    30-Nov-2020 22:47:24
    "MW Natick"    "GALILEO 18"          1          30-Nov-2020 22:23:24    01-Dec-2020 01:51:24
    "MW Natick"    "GALILEO 19"          1          30-Nov-2020 22:23:24    01-Dec-2020 00:13:24
    "MW Natick"    "GALILEO 20"          1          30-Nov-2020 22:23:24    30-Nov-2020 22:27:24
    "MW Natick"    "GALILEO 11"          1          30-Nov-2020 22:38:24    01-Dec-2020 00:00:24
    "MW Natick"    "GALILEO 17"          1          30-Nov-2020 22:47:24    01-Dec-2020 03:19:24
    "MW Natick"    "GALILEO 8"           1          30-Nov-2020 23:09:24    01-Dec-2020 03:15:24
    "MW Natick"    "GALILEO 7"           1          01-Dec-2020 00:28:24    01-Dec-2020 04:55:24
    "MW Natick"    "GALILEO 24"          1          01-Dec-2020 00:29:24    01-Dec-2020 04:40:24
    "MW Natick"    "GALILEO 6"           1          01-Dec-2020 01:54:24    01-Dec-2020 06:41:24
    "MW Natick"    "GALILEO 23"          1          01-Dec-2020 02:04:24    01-Dec-2020 05:52:24
    "MW Natick"    "GALILEO 5"           1          01-Dec-2020 03:30:24    01-Dec-2020 08:25:24
      ⋮
```

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

```
intervalsDE=51×8 table
       Source          Target       IntervalNumber         StartTime              EndTime
    ____________    ____________    ______________    ____________________    ____________________

    "MW Munchen"    "GALILEO 2"           1          30-Nov-2020 22:23:24    01-Dec-2020 02:01:2
    "MW Munchen"    "GALILEO 3"           1          30-Nov-2020 22:23:24    01-Dec-2020 00:20:2
    "MW Munchen"    "GALILEO 4"           1          30-Nov-2020 22:23:24    30-Nov-2020 22:40:2
    "MW Munchen"    "GALILEO 19"          1          30-Nov-2020 22:23:24    01-Dec-2020 00:42:2
    "MW Munchen"    "GALILEO 20"          1          30-Nov-2020 22:23:24    30-Nov-2020 23:35:2
    "MW Munchen"    "GALILEO 10"          1          30-Nov-2020 22:38:24    30-Nov-2020 22:49:2
    "MW Munchen"    "GALILEO 18"          1          30-Nov-2020 22:42:24    01-Dec-2020 01:37:2
    "MW Munchen"    "GALILEO 9"           1          30-Nov-2020 22:44:24    01-Dec-2020 00:43:2
    "MW Munchen"    "GALILEO 1"           1          30-Nov-2020 22:55:24    01-Dec-2020 03:37:2
    "MW Munchen"    "GALILEO 16"          1          30-Nov-2020 23:24:24    01-Dec-2020 02:15:2
    "MW Munchen"    "GALILEO 17"          1          01-Dec-2020 00:13:24    01-Dec-2020 02:19:2
    "MW Munchen"    "GALILEO 15"          1          01-Dec-2020 00:18:24    01-Dec-2020 03:45:2
    "MW Munchen"    "GALILEO 8"           1          01-Dec-2020 00:36:24    01-Dec-2020 05:07:2
    "MW Munchen"    "GALILEO 14"          1          01-Dec-2020 01:24:24    01-Dec-2020 05:17:2
    "MW Munchen"    "GALILEO 24"          1          01-Dec-2020 01:57:24    01-Dec-2020 02:36:2
    "MW Munchen"    "GALILEO 7"           1          01-Dec-2020 02:16:24    01-Dec-2020 06:29:2
      ⋮
```

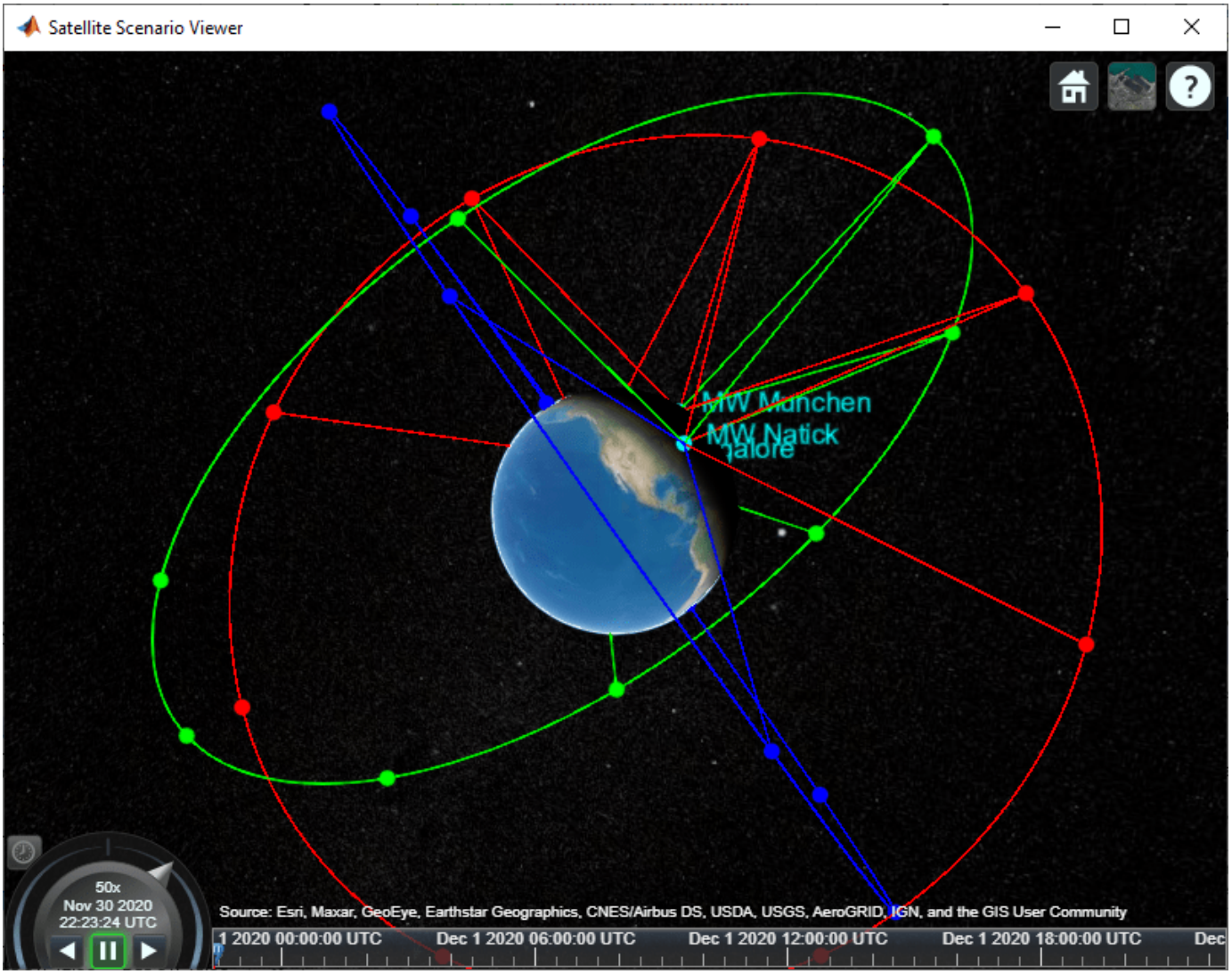
```
intervalsIN = accessIntervals(accessIN);
intervalsIN = sortrows(intervalsIN, "StartTime", "ascend")
```

```
intervalsIN=45×8 table
        Source           Target       IntervalNumber         StartTime              EndTime
    _____________    _____________    ______________    ____________________    ____________

    "MW Bangalore"    "GALILEO 3"           1          30-Nov-2020 22:23:24    01-Dec-2020 02:38
    "MW Bangalore"    "GALILEO 4"           1          30-Nov-2020 22:23:24    01-Dec-2020 01:05
    "MW Bangalore"    "GALILEO 5"           1          30-Nov-2020 22:23:24    30-Nov-2020 23:17
    "MW Bangalore"    "GALILEO 9"           1          30-Nov-2020 22:23:24    01-Dec-2020 00:55
    "MW Bangalore"    "GALILEO 10"          1          30-Nov-2020 22:23:24    30-Nov-2020 23:19
    "MW Bangalore"    "GALILEO 16"          1          30-Nov-2020 22:23:24    01-Dec-2020 02:42
    "MW Bangalore"    "GALILEO 21"          1          30-Nov-2020 22:23:24    30-Nov-2020 23:02
    "MW Bangalore"    "GALILEO 22"          1          30-Nov-2020 22:23:24    30-Nov-2020 22:29
    "MW Bangalore"    "GALILEO 15"          1          30-Nov-2020 23:36:24    01-Dec-2020 05:12
    "MW Bangalore"    "GALILEO 2"           1          30-Nov-2020 23:44:24    01-Dec-2020 04:05
    "MW Bangalore"    "GALILEO 21"          2          01-Dec-2020 01:15:24    01-Dec-2020 04:12
    "MW Bangalore"    "GALILEO 1"           1          01-Dec-2020 01:20:24    01-Dec-2020 05:26
    "MW Bangalore"    "GALILEO 14"          1          01-Dec-2020 01:29:24    01-Dec-2020 07:31
    "MW Bangalore"    "GALILEO 20"          1          01-Dec-2020 02:04:24    01-Dec-2020 05:56
    "MW Bangalore"    "GALILEO 8"           1          01-Dec-2020 02:56:24    01-Dec-2020 06:37
    "MW Bangalore"    "GALILEO 19"          1          01-Dec-2020 03:20:24    01-Dec-2020 07:32
      ⋮
```

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);
```

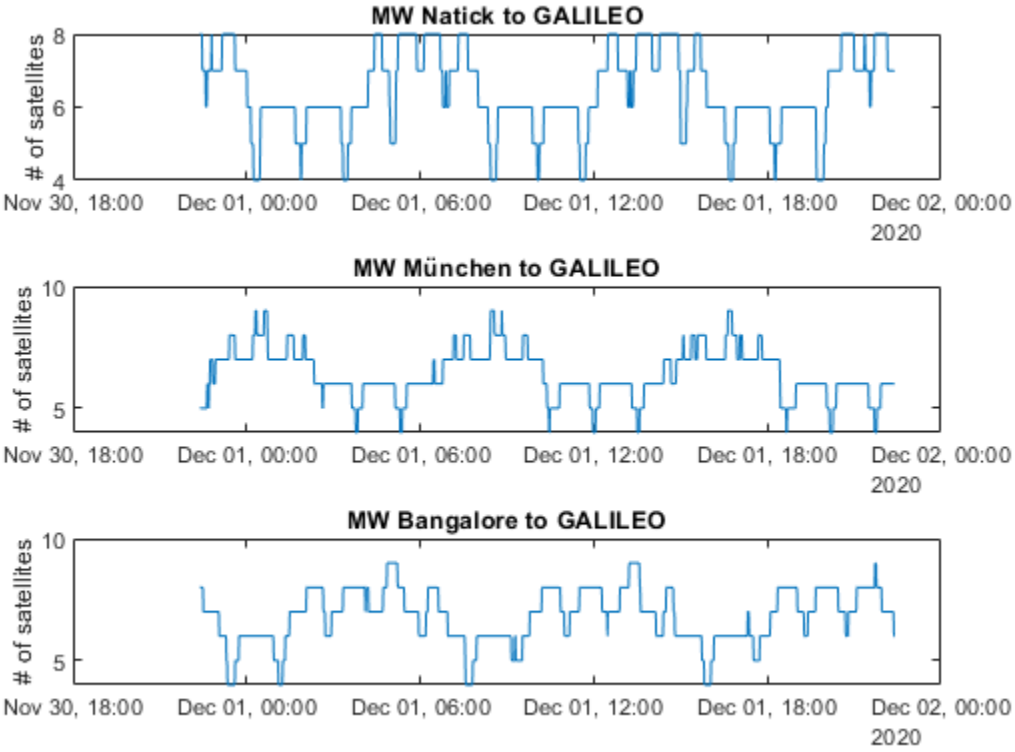


Compare Access Between Ground Stations

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("MW Natick to GALILEO")
ylabel("# of satellites")
subplot(3,1,2);
plot(timeSteps, statusDE);
title("MW München to GALILEO")
ylabel("# of satellites")
subplot(3,1,3);
plot(timeSteps, statusIN);
title("MW 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 = ["MW Natick", "MW München", "MW 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
                                MW Natick    MW München    MW Bangalore
                                _________    __________    ____________

    Total # of intervals            52           51            45
    Total interval time (hrs)      153.92       153.55        162.73
    Mean interval length (min)      177.6        180.65        216.98
    Mean # of satellites in view   6.4448       6.4289        6.8071
    Min # of satellites in view         4            4             4
    Max # of satellites in view         8            9             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 40 minutes). All locations always have a minimum of 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
`access` | `conicalSensor` | `groundStation` | `receiver` | `satellite` | `satelliteScenario` | `satelliteScenarioViewer` | `transmitter`

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

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-16
- “Comparison of Orbit Propagators” on page 1-30
- “Estimate GNSS Receiver Position with Simulated Satellite Constellations” on page 1-48
- “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 Navigation Toolbox™.

Overview

This example focuses on the *space segment,* or satellite constellations, and the GNSS sensor equipment for a statellite 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

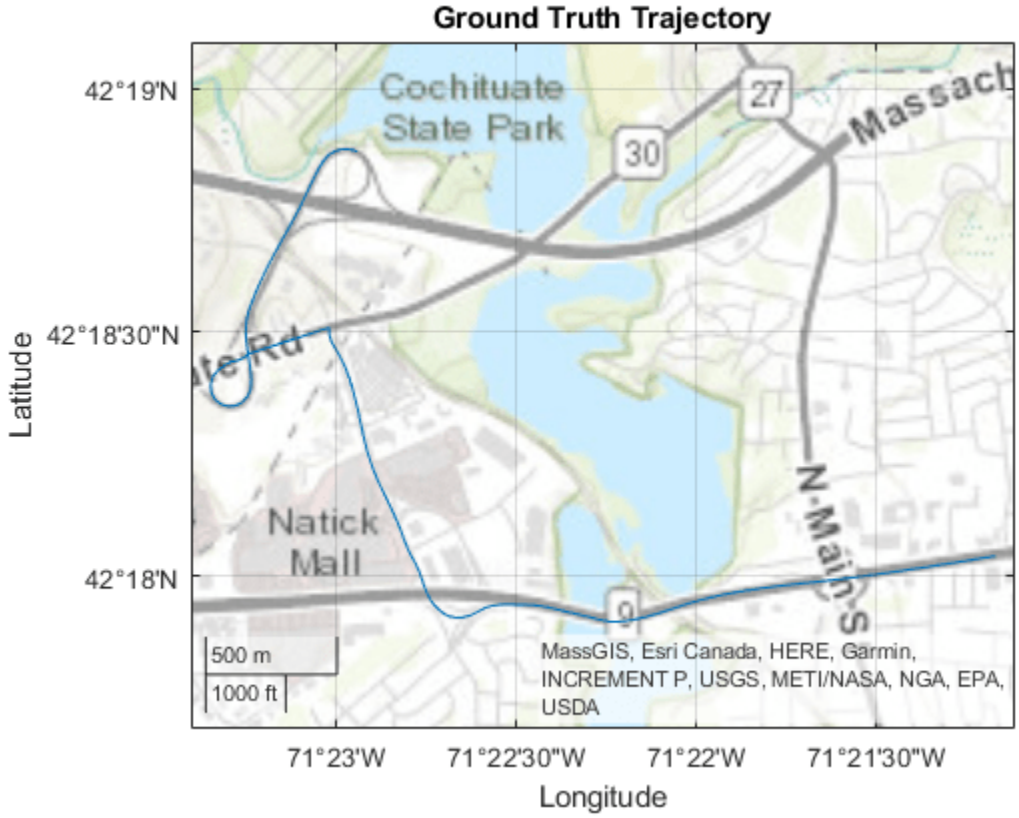
Load the MAT-file that contains the ground-truth position and velocity of a ground vehicle travelling toward the Natick, MA campus of Mathworks, Inc.

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 Navitation 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.

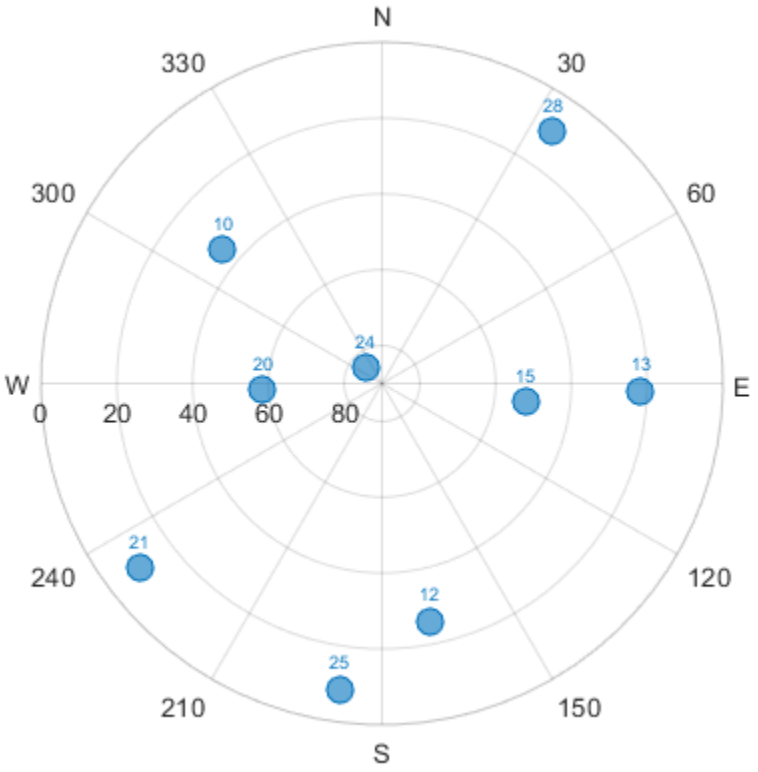
```
% Create scenario.
sc = satelliteScenario(startTime, stopTime, dt);

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 visibile 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

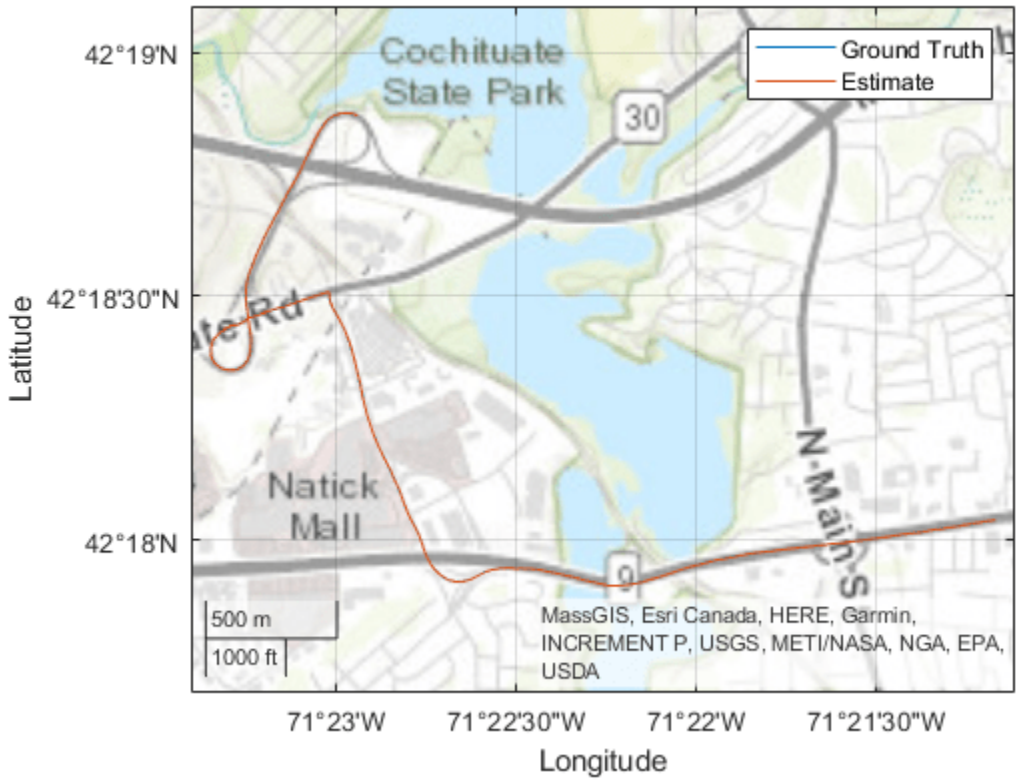
Finally, use the satellite positions and pseudoranges to estimate the receiver position with the `receiverposition` function.

```
% 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

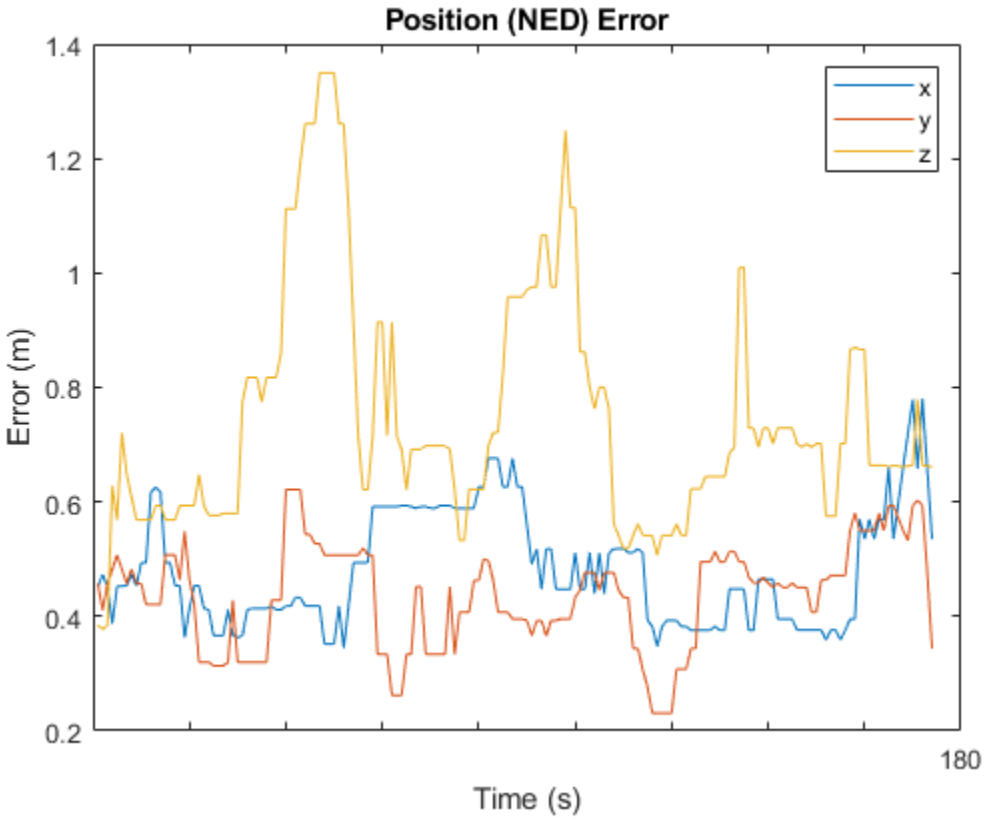
Plot the ground truth position and the estimated receiver position on a `geoplot`.

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

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

Related Examples

- “Multi-Hop Satellite Communications Link Between Two Ground Stations” on page 1-2
- “Satellite Constellation Access to a Ground Station” on page 1-16
- “Comparison of Orbit Propagators” on page 1-30
- “Modeling Satellite Constellations using Ephemeris Data” on page 1-38
- “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, its frame structure, and the complex baseband waveform. The spreading of the data is performed with coarse acquisition code (C/A-code) and precision code (P-code). To design a navigation system based on GPS, testing the receiver with a received signal is required. The signal received from a satellite is not useful for this purpose, as transmitter and channel parameters cannot be controlled. To test the receiver, a waveform generated under a controlled set of parameters is required.

GPS Frame Structure

Obtain the GPS data bits, which are transmitted at a rate of 50 bits per second (bps). Apply C/A-code spreading on the generated data. This C/A-code has a rate of 1.023 mega chips per second. Because one data bit is transmitted for every 0.02 seconds (50 bps), each data bit is spread with 0.02*1.023*1e6 = 20460 chips. The data is also spread with the P-code. This P-code has a rate of 10.23 mega chips per second which implies that each data bit is spread with 204600 chips of P-code.

GPS data contains various configuration parameters, clock parameters, and parameters related to the location of a satellite vehicle (SV) in space. All of these parameters can be classified into three broad categories:

- Ephemeris parameters
- Almanac parameters
- Parameters that are neither ephemeris nor almanac, which include configuration parameters, clock parameters, ionosphere parameters, Universal Time Coordinated (UTC) parameters, and navigation message correction tables (NMCT) parameters

GPS satellites transmit signals on two frequencies: L1 (1575.42 MHz) and L2 (1227.60 MHz). Both signals are generated from a base frequency of 10.23 MHz. The signal frequency of L1 is 154*10.23 MHz = 1575.42 MHz, and the signal frequency of L2 is 120*10.23 MHz = 1227.60 MHz. Legacy GPS satellites transmit different spreading codes and data on L1 and L2 carrier frequencies. For details, see IS-GPS-200L Table 3-III [1] on page 2-0 .

Three kinds of spreading for legacy GPS exist:

- C/A-code spreading
- P-code spreading
- Y-code spreading

Y-code is not available for public use. Because Y-code is an encrypted version of P-code, P-code and Y-code are used together as P(Y)-code.

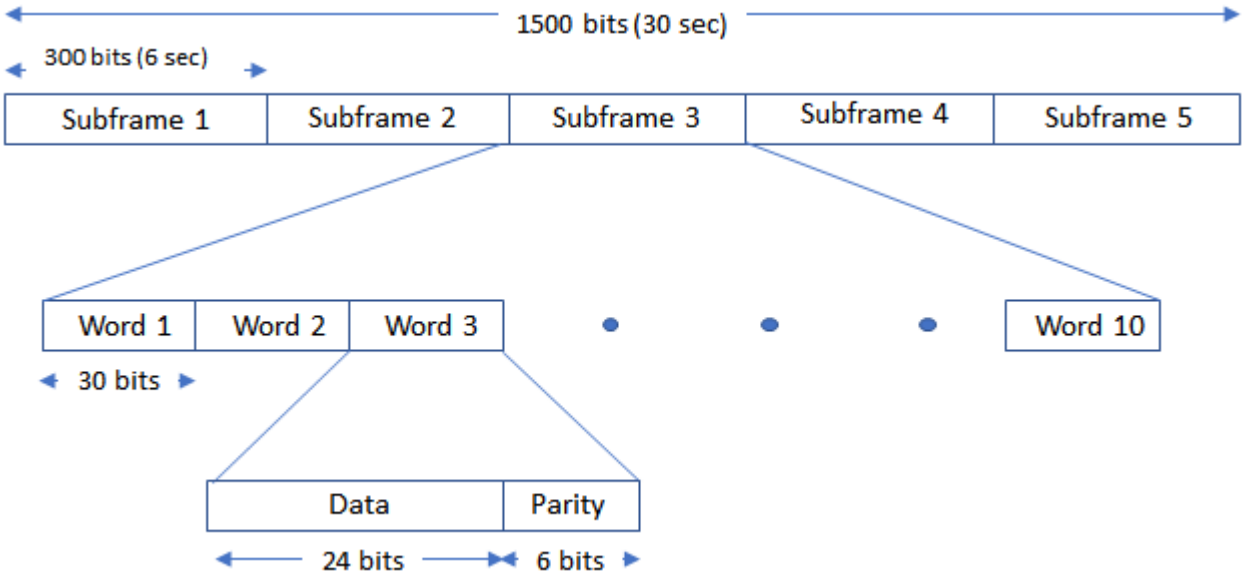
- On the L1 carrier frequency, the in-phase component carries P(Y)-code along with data.
- On the L1 carrier frequency, the quadrature-phase component carries C/A-code along with data.
- On the L2 carrier frequency, the in-phase component contains either P(Y)-code with data, P(Y)-code without data, or C/A-code with data.
- On the L2 carrier frequency, the quadrature-phase component is empty.

This example demonstrates the signal on the L1 carrier frequency, which contains P-code on the in-phase component and C/A-code on the quadrature-phase component.

GPS satellites transmit data 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 is made up of 10 words with 30 bits in each word. The GPS data contains information regarding the clock and the position of the satellites. To enable computation of the distance from a satellite to a receiver, the navigation data is spread with two types of codes: C/A-code and P-code. C/A-code is at 1.023 MHz and repeats after 1023 chips. P-code is at 10.23 MHz and repeats after each week for a given satellite.

This figure illustrates the legacy navigation (LNAV) data frame.



Subframe 1 contains information about the type of code (C/A or P) and the presence of data on the L2 channel. Subframe 1 also provides the clock information on-board a satellite.

Subframes 2 and 3 contain information about the position of the transmitting SV in space through orbital information of the satellite. This orbital information of the SV is called ephemeris.

Subframes 4 and 5 contain information about the position of the GPS satellites that are currently operating and gives an estimate of satellites that are visible to a user. This information about positions of all satellites in the constellation is collectively called almanac. Ephemeris gives the accurate position of the SV in space, and almanac contains less accurate position information of all the SVs. Almanac data of the SVs is spanned over 25 frames. Because transmitting each frame takes 30 seconds, transmitting the entire almanac spanned over 25 frames takes 12.5 minutes.

Configuration Parameters

Select the `ShowVisualizations` parameter to enable the spectrum and correlation plot visualizations. Select the `WriteWaveformToFile` parameter to write the complex baseband waveform to a file if needed.

```
ShowVisualizations = ;
WriteWaveformToFile = ;
```

Specify the satellite pseudo-random noise (PRN) index.

```
PRNIndex = 1;
```

Because generating the GPS waveform for the entire navigation data takes a lot of time and memory, this example demonstrates generating waveform for one bit of the navigation data. Generating the waveform for number of GPS LNAV data bits can be controlled by the parameters `NavDataBitStartIndex` and `NavDataBitEndIndex`.

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

Specify for frames for which legacy GPS waveform should be generated.

```
FrameIndices = 1:25; % This value must be consecutive integers in the range [1, 25]
```

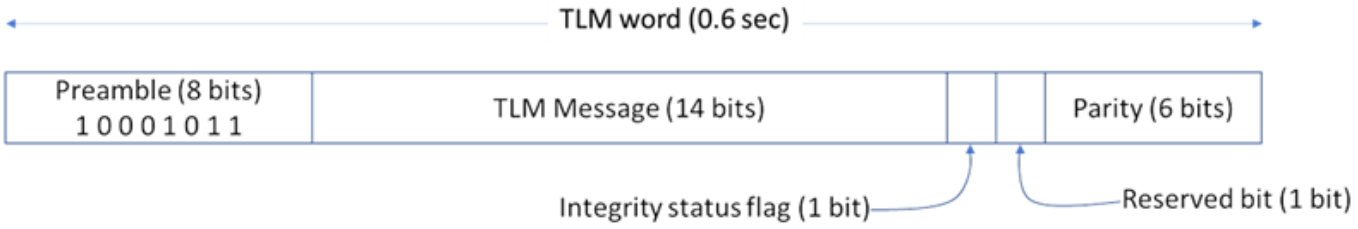
GPS Data Generation

GPS data generation involves initializing the relevant parameters. This section shows how to initialize these parameters.

```
svparams.PRNIdx = PRNIndex;           % Current satellite PRN index
svparams.FrameIndices = FrameIndices;
```

Telemetry (TLM) Word and Handover Word (HOW)

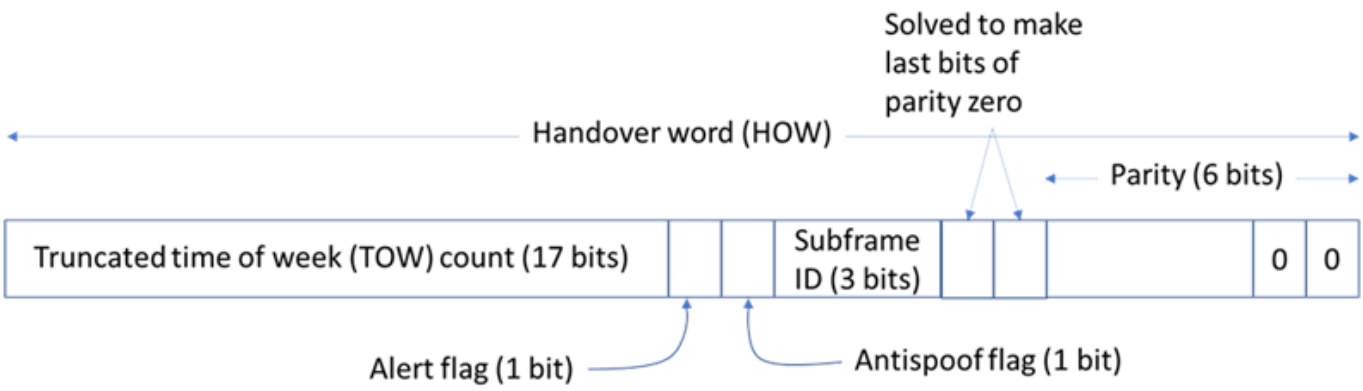
In each subframe, Word 1 is the TLM word, and word 2 is the HOW. As shown in this figure, the TLM word contains an eight-bit preamble, the TLM message, integrity status flag, and parity. The last 6 bits of each word contains parity. These parity bits are calculated based on the algorithm presented in IS-GPS-200L Table 20-XIV [1] on page 2-0 .




```
svparams.TLMPreamble            = 139; % Decimal equivalent of [1 0 0 0 1 0 1 1]
svparams.TLMMessage             = 0;   % Contains information needed by the precise positioning s
svparams.TLMIntegrityStatusFlag = 0;   % Legacy level of integrity
```

The next figure shows the fields in the HOW. The first 17 bits of the HOW indicate the 17 most significant bits (MSBs) of the time of week (TOW). The TOW is a 19-bit number, and the least significant bit (LSB) of the TOW ticks for every 1.5 seconds. The TOW in the HOW (which is the 17 MSBs of the full TOW) ticks for every 6 seconds. That is, the TOW in the HOW ticks for each subframe. For example, if the TOW value is 1000 at the current instance, then after 1.5 seconds, the TOW is 1001. After another 1.5 seconds, the TOW is 1002, and so on. If the 17 MSBs of the TOW (which is in the HOW) are considered, then for every 4 ticks of full TOW, this TOW-in-HOW ticks for one time. Because each set of 4 ticks in the full TOW corresponds to 6 seconds, the TOW-in-HOW ticks for every 6 seconds.

The 18th bit in the HOW is the alert flag that indicates that the user range accuracy (URA) of the standard positioning service (SPS) user is worse than what is indicated by the `svparams.URAIndex`, and user must use the current satellite at their own risk. Bit 19 in the HOW is the antispoof flag. If this flag is set, then in place of transmitting P-code, the satellite transmits the Y-code. Bits 20 to 22 in the HOW indicate the subframe ID. Last two bits of 6 bits parity are set to zero, and bits 23 and 24 are solved to set the last two bits of the parity to zero.




```
svparams.HOWTOW = 2000;     % Decimal equivalent of 17 MSBs of TOW
svparams.HOWAlertFlag = 0;  % Alert flag
svparams.AntiSpoofFlag = 0; % Antispoof flag
svparams.SubframeID = 1;    % Subframe ID of starting subframe
```

Clock, NMCT, Ionosphere, and UTC 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. These clock parameters are transmitted in subframe 1. Along with the clock information, subframe 1 also contains various configuration information. This code initializes these parameters. For a detailed description of parameters in subframe 1, see IS-GPS-200L [1] on page 2-0 .

```
svparams.WeekNumberMod1024 = 39; % Week number modulo 1024 to fit in 10 bits of data.
                                 % Same as in Almanac file
svparams.CodesOnL2Channel = 2;   % Value of 2 indicates C/A-code on L2 channel
                                 % Value of 1 indicates P code on L2 channel
svparams.DataFlagForL2P = 1;     % Value of 1 indicates nav data Off on P-code
                                 % of in-phase component of L2 channel
svparams.URAIdx = 0;             % User range accuracy (URA) index.
                                 % Refer IS-GPS-200L Section 20.3.3.3.1.3 [1]
svparams.SVHealth = 0;           % SV health
svparams.IODC = 0;               % Issue of data, clock (IODC)
svparams.T_GD = 0;               % Estimated group delay differential
svparams.t_oc = 0;               % Clock data reference time
svparams.a_f2 = 0;               % Second order clock correction coefficient
svparams.a_f1 = 0;               % First order clock correction coefficient
svparams.a_f0 = 0;               % Zeroth order clock correction coefficient
```

Frame 13 of subframe 4 includes the NMCT range corrections. These range corrections are done using estimated range deviation (ERD) values for each satellite. The way in which NMCT values are mapped to ERD values is defined in IS-GPS-200L [1] on page 2-0 .

`svparams.NMCTAvailabilityIndicator` is a 2-bit integer value that indicates the availability of NMCT data.

- `0` - NMCT available for both precise positioning service (PPS) user and standard positioning service (SPS) user

- 1 - NMCT is available only to authorized users
- 2 - No correction table is available to any user
- 3 - Reserved for future use

```
svparams.NMCTAvailabilityIndicator = 0; % NMCT available to both PPS and SPS users
```

The code initializes the ERD values for a particular satellite. The values are initialized to zero. The valid range is from `-9.3` to `9.3` meters. Each array element indicates the corresponding ERD value (that is, the first element is ERD01, the second element is ERD02, and so on).

```
svparams.NMCTERD = zeros(30,1); % Units are in meters
```

A frequency dependent delay is caused due to propagation of signal through ionosphere. This delay is one of the significant errors in calculation of accurate time. To compensate for this delay due to ionosphere, dual frequency receivers need to be used for accurate positioning of the receiver. To provide a rough model of the ionosphere and delay caused by it for single frequency receivers, GPS transmits some parameters on the GPS data in subframe 4. These parameters are initialized as follows.

```
svparams.Ionosphere.alpha = zeros(4,1); % Units of each element in the array is seconds
svparams.Ionosphere.beta = zeros(4,1);  % Units of each element in the array is seconds
```

Subframe 4 of the GPS LNAV data contains parameters to correlate UTC with that of the GPS time. This code initializes these parameters.

```
svparams.UTC.A_1 = 0;      % In seconds/second
svparams.UTC.A_0 = 0;      % In seconds
svparams.UTC.t_ot = 0;     % In seconds
svparams.UTC.WN_t = 39;    % In weeks. Same value as in almanac file
svparams.UTC.WN_LSF = 39;  % In weeks. Same value as in almanac file
svparams.UTC.Delt_LS = 0;  % In seconds
svparams.UTC.DN = 0;       % In days
svparams.UTC.Delt_LSF = 0  % In seconds
```

```
svparams = struct with fields:
                        PRNIdx: 1
                  FrameIndices: [1x25 double]
                   TLMPreamble: 139
                    TLMMessage: 0
         TLMIntegrityStatusFlag: 0
                        HOWTOW: 2000
                  HOWAlertFlag: 0
                 AntiSpoofFlag: 0
                    SubframeID: 1
              WeekNumberMod1024: 39
               CodesOnL2Channel: 2
                 DataFlagForL2P: 1
                        URAIdx: 0
                     SVHealth: 0
                         IODC: 0
                         T_GD: 0
                         t_oc: 0
                         a_f2: 0
                         a_f1: 0
                         a_f0: 0
    NMCTAvailabilityIndicator: 0
                      NMCTERD: [30x1 double]
```

```
                Ionosphere: [1x1 struct]
                       UTC: [1x1 struct]
```

Display the structure `svparams.Ionosphere`.

```
disp(svparams.Ionosphere)
```

```
    alpha: [4x1 double]
     beta: [4x1 double]
```

Display the structure `svparams.UTC`.

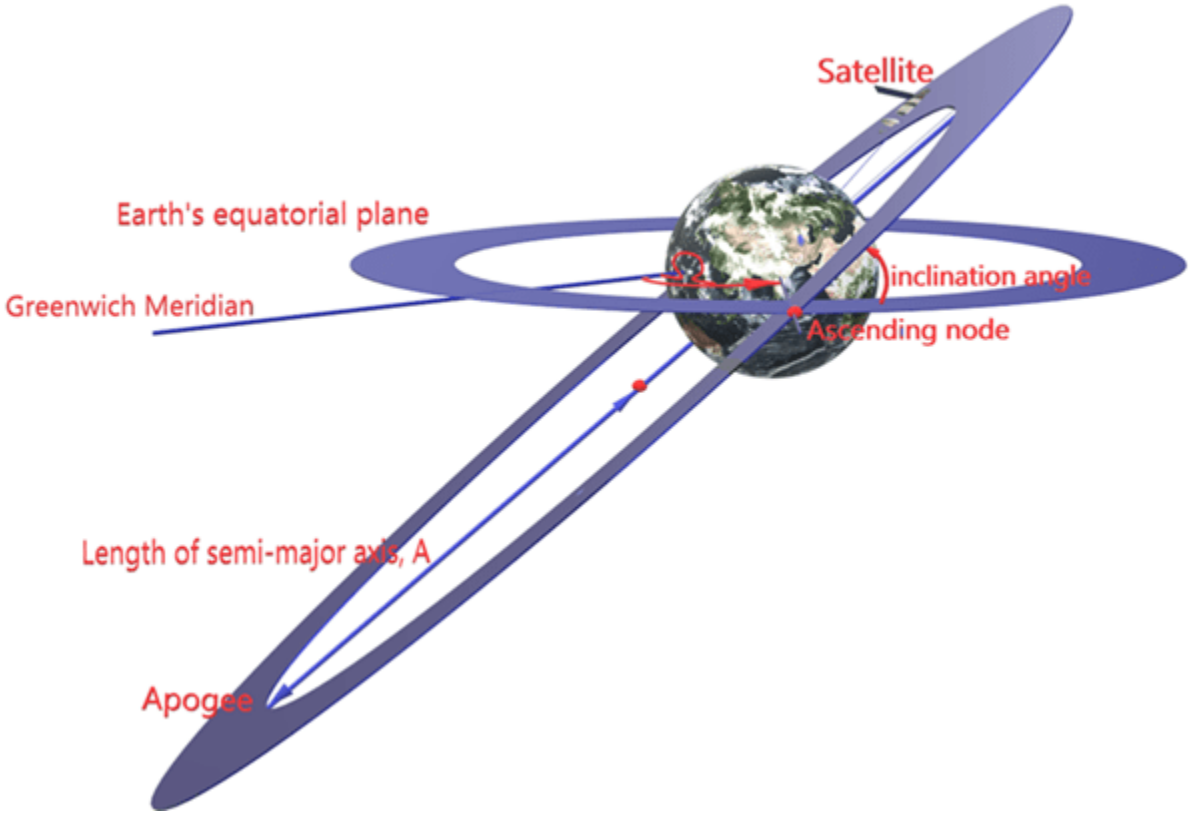
```
disp(svparams.UTC)
```

```
         A_1: 0
         A_0: 0
        t_ot: 0
        WN_t: 39
      WN_LSF: 39
     Delt_LS: 0
          DN: 0
    Delt_LSF: 0
```

Ephemeris and Almanac Data

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. This figure shows three 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
- Inclination angle: Angle between the equatorial plane of Earth and the plane of orbit of satellite
- Longitude of ascending node, Ω: Angle between the Greenwich meridian and the direction of the ascending node



Define a structure, called `ephemerisparams`, that stores the ephemeris parameters. The parameters that convey angle are in the units of semicircles, because the GPS standard [1] on page 2-0 specifies these parameters in semicircles. One semicircle is equal to π radians (180 degrees).

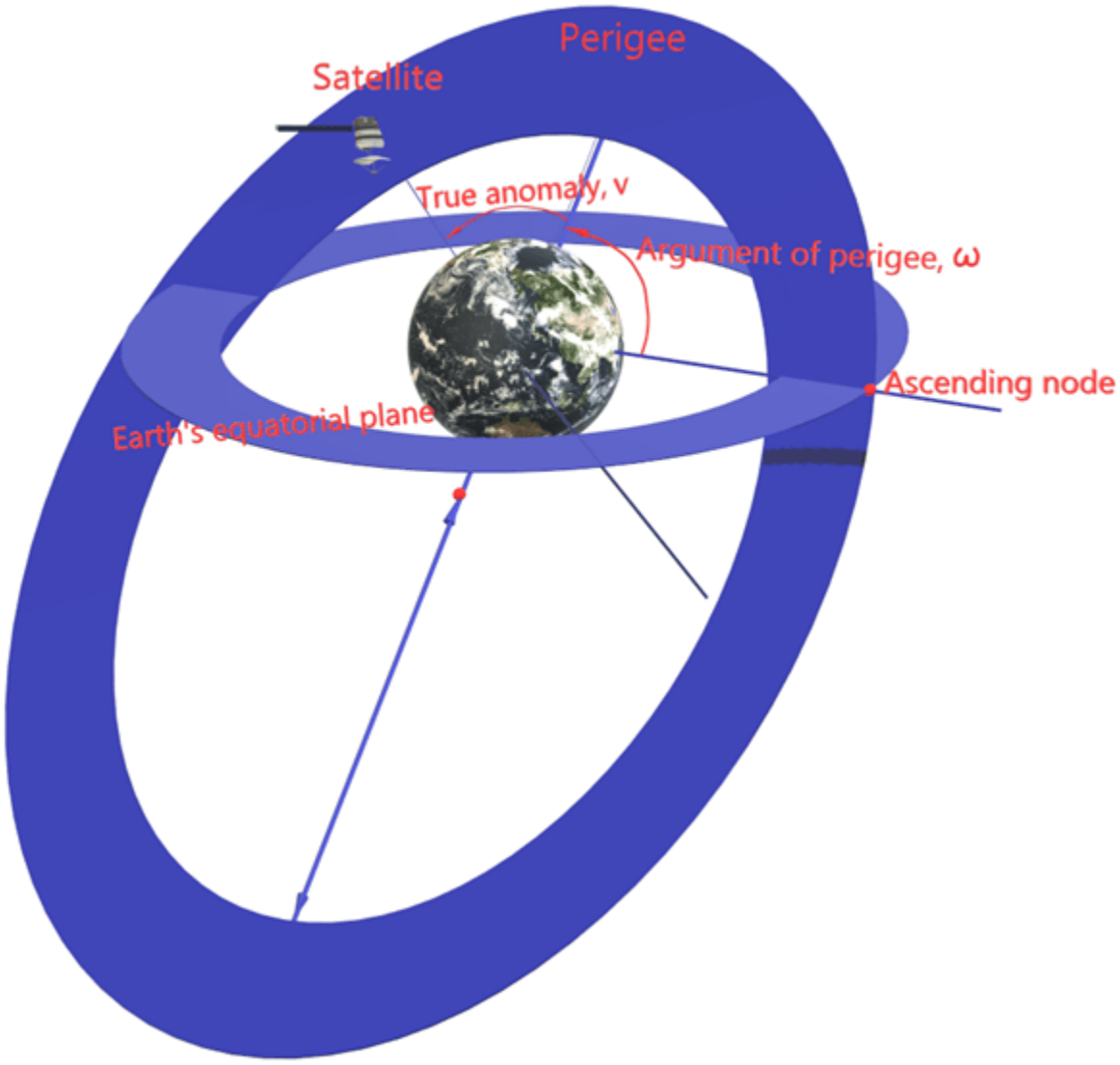
```
ephemerisparams.SqrtA = 0;    % Square root of length of semimajor axis
ephemerisparams.i_0 = 0;      % Inclination angle (in semicircles)
ephemerisparams.Omega_0 = 0; % Longitude of ascending node at weekly epoch (in semicircles)
```

This next figure shows two orbital parameters:

- Argument of perigee, ω: Angle between the direction of the ascending node and the direction of perigee
- True anomaly, υ: Angle between the direction of the perigee and the direction in which current position of the satellite is present

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). The algorithms that relate mean anomaly and true anomaly are given in IS-GPS-200L Table 20-IV [1] on page 2-0 .

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




```
ephemerisparams.omega = 0; % Argument of perigee (in semicircles)
ephemerisparams.M_0 = 0;   % Mean anomaly at reference time (in semicircles)
ephemerisparams.e = 0;     % Eccentricity of the ellipse
```

These ephemeris parameters aid in getting the exact location of satellite in the space.

```
ephemerisparams.IODE = 0;            % Issue of data, ephemeris
ephemerisparams.c_rs = 0;            % Amplitude of the Sine Harmonic Correction Term to
                                     % the Orbit Radius (in meters)
ephemerisparams.Deln = 0;            % Mean Motion Difference From Computed Value (in semi-circle
ephemerisparams.c_uc = 0;            % Amplitude of the Cosine Harmonic Correction Term to
                                     % the Argument of Latitude (in radians)
ephemerisparams.c_us = 0;            % Amplitude of the Sine Harmonic Correction Term to
                                     % the Argument of Latitude (in radians)
ephemerisparams.t_oe = 0;            % Reference time ephemeris (in seconds)
ephemerisparams.FitIntervalFlag = 0; % Fit interval flag
ephemerisparams.AODO = 0;            % Age of data offset
```

```
ephemerisparams.c_ic = 0;           % Amplitude of the Cosine Harmonic Correction Term to
                                    % the Angle of Inclination (in radians)
ephemerisparams.c_is = 0;           % Amplitude of the Sine Harmonic Correction Term to
                                    % the Angle of Inclination (in radians)
ephemerisparams.c_rc = 0;           % Amplitude of the Cosine Harmonic Correction Term to
                                    % the Orbit Radius (in meters)
ephemerisparams.OmegDot = 0;        % Rate of Right Ascension (in semi-circles/second)
ephemerisparams.IDOT = 0            % Rate of Inclination Angle (in semi-circles/secon)
```

```
ephemerisparams = struct with fields:
              SqrtA: 0
                i_0: 0
            Omega_0: 0
              omega: 0
                M_0: 0
                  e: 0
               IODE: 0
               c_rs: 0
               Deln: 0
               c_uc: 0
               c_us: 0
               t_oe: 0
    FitIntervalFlag: 0
               AODO: 0
               c_ic: 0
               c_is: 0
               c_rc: 0
            OmegDot: 0
               IDOT: 0
```

Almanac is a set of parameters that defines the position of the satellites in the orbit. In contrast with ephemeris, almanac is less precise. Additionally, ephemeris is for only one satellite, whereas almanac is for all satellites in the constellation. A receiver obtains almanac data by decoding the GPS data. A satellite continually gets this almanac data from the control segment (CS). Almanac is available to download from the navcen website. This example uses the system effective model (SEM) format of almanac data (see [2] on page 2-0) and extracts the almanac data as shown below.

```
filename = 'gpsAlmanac.txt'; % Almanac file name (this file should be in current folder)
                             % Use any almanac file
almanacparams = HelperGPSAlmanac2Struct(filename)
```

```
almanacparams = struct with fields:
    SatellitesPRNIndices: [31x1 double]
       WeekNumModulo1024: 39
                   t_oa: 589824
                   Data: [1x31 struct]
```

Generate GPS Signal

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

1. Generate the navigation data bits at 50 bits per second.
2. Generate the C/A-code at 1.023 mega chips per second.
3. Perform the XOR operation of C/A-code with the navigation data to spread the data.

4. BPSK-modulate the C/A-code chips by mapping bit 0 to +1 and bit 1 to -1.
5. Repeat above three steps for P-code at a chip rate of 10.23 mega chips per second.
6. Repeat each C/A-code chip 10 times to match the P-code frequency.
7. Place P-code and C/A-code on appropriate I and Q branches to create a complex baseband signal.
8. If needed, write this baseband waveform to a file for later use.

After all of the parameters for the navigation data and spreading code are initialized, generate legacy GPS navigation data using the helper function `HelperGPSLNAVData`.

```
lnavData = HelperGPSLNAVData(svparams,ephemerisparams,almanacparams); % Generates data for given
```

Because the GPS data is at 50 bits per second, and the C/A-code is at 1.023 mega chips per second, every data bit corresponds to 20460 C/A-code chips. Because creating chips for all of the data at a single time costs a great deal of memory, C/A-code spreading is done for one LNAV data bit at a time within a loop. The PRN code rate is referred to as "chip rate" rather than "bit rate" to differentiate the notation with the data bit rate. In a navigation system, the spreading code (in this example C/A-code or P-code) is called a *ranging code* because this code is used at the receiver to calculate the propagation time of the signal. The distance from the satellite to the receiver is measured by multiplying the propagation time with the speed of light in free space.

```
prnCAChipRate = 1.023e6; % In chips/second
gpsDataRate = 50;        % In bits/second
```

Number of chips in one data bit is chip rate/data rate.

```
numCAChipsPerDataBit = prnCAChipRate/gpsDataRate
```

```
numCAChipsPerDataBit = 20460
```

The C/A-code is a fixed binary-valued vector for a given data bit, so generating the C/A-code for every data bit that is transmitted is not necessary. Generate the C/A-code corresponding to one navigation data bit outside the loop.

```
CACode = HelperGPSCACode(svparams.PRNIdx,numCAChipsPerDataBit);
```

Take one bit of navigation data and spread using the P-code. Take length of the P-code to match in time the C/A-code length that is already taken. Because the C/A-code is at the rate of 1.023 mega chips per second with 20,460 chips per one bit of navigation data, and P-code is at the rate of 10.23 mega chips per second, 204,600 chips of P-code exist for one bit of navigation data.

```
numPChipsPerDataBit = numCAChipsPerDataBit*10 % Because chip rate of P-code is 10 times the chip
```

```
numPChipsPerDataBit = 204600
```

The length of the P-code is such that it has a periodicity of 1 week. P-code chipping at the rate of 10.23 mega chips per second implies that 10.23x10^6 chips exist for every second. Because one week contains 604,800 seconds, the length of the P-code in one period is 10.23*10^6 * 604800 = 6187104000000 chips. To generate the P-code along with the length of sequence, the number of chips that are elapsed from the beginning of the week is needed, which is calculated from the `HOWTOW` value of the first subframe and its bit index.

```
numSecondsElapsed = (svparams.HOWTOW-1)*6; % Because each subframe is processed in 6 seconds
```

Initialize the 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 = NavDataBitStartIndex:NavDataBitEndIndex
    spreadedCA = xor(CACode,lnavData(iDataBit));
    M = 2;                                      % Modulation order (for BPSK, M = 2)
    bpskSigCA = pskmod(double(spreadedCA),M);
```

Due to these two reasons, the number of P-code chips that are processed just before the start of the data bit in consideration is calculated as shown in this code.

- Every second, 10.23*10^6 P-code chips are processed.
- Each data bit is spread with `numPChipsPerDataBit` number of P-code chips.

```
    numPChipsPassed = round(numSecondsElapsed*10.23*10^6+(iDataBit-1)*numPChipsPerDataBit);
```

Generate the P-code.

```
    PCode = HelperGPSPCode(numPChipsPerDataBit,svparams.PRNIdx,numPChipsPassed);
    spreadedWithP = xor(PCode,lnavData(iDataBit));
```

BPSK-modulate the P-code.

```
    bpskSigP = pskmod(double(spreadedWithP),M);
```

Generate the baseband waveform of the GPS L1 signal by placing the P-code on the I-branch and C/A-code on the Q-branch. Before the baseband waveform is generated, the BPSK signal that is generated by spreading with the C/A-code is repeated 10 times (because the P-code is 10 times faster than the C/A-code). Unlike a communication link, which typically has a pulse shaping filter in the digital baseband, a GPS transmitter does not have a pulse shaping filter. For precise ranging, a satellite receiver must reproduce the spreading waveform, which can be accomplished by a rectangular pulse. Spectral efficiency of the signal is not a major concern in navigation applications.

```
    repeatedBPSKSigCA = repelem(bpskSigCA(:),10);
    gpsBBWaveform = bpskSigP(:) + 1j*repeatedBPSKSigCA(:);
    if WriteWaveformToFile == 1
        bbWriter(gpsBBWaveform);
    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
```

The ranging codes designed for GPS have very good autocorrelation and crosscorrelation properties. Autocorrelation of the sequence is near-zero except at zero delay, and crosscorrelation of two

different sequences is near-zero. As the C/A-code is periodic with period of 1023 bits, autocorrelation has a peak for a delay of every 1023 bits.

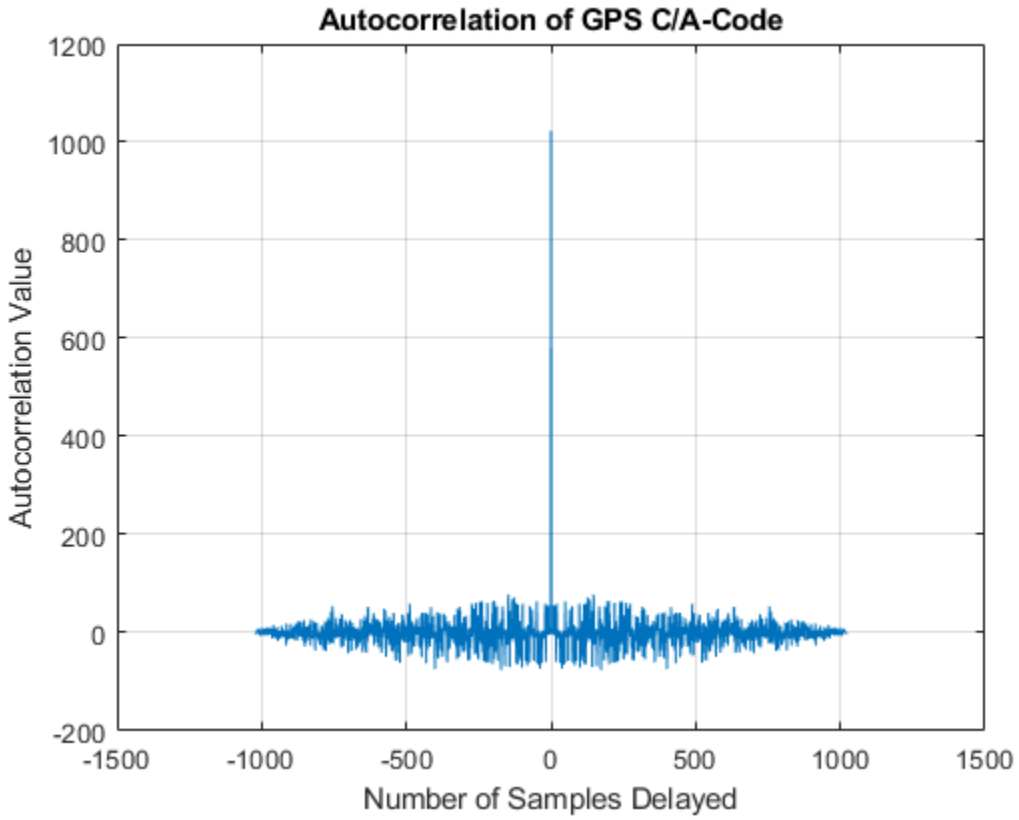
```
lags = (-1023:1023).';
plot(lags,xcorr(real(bpskSigCA(1:1023)),1023));
grid on;
xlabel('Number of Samples Delayed')
ylabel('Autocorrelation Value')
title('Autocorrelation of GPS C/A-Code')
```

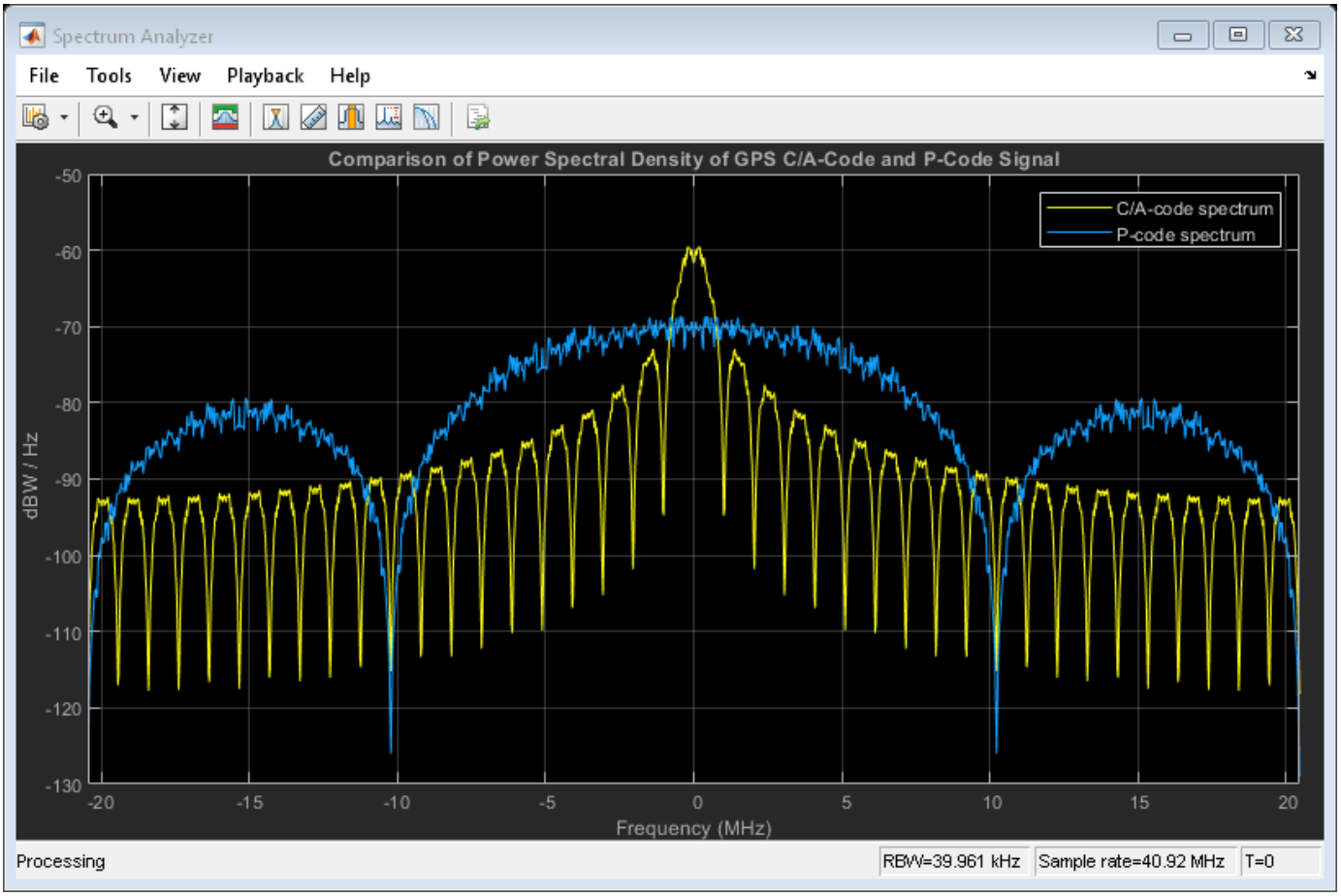
The spectrum plot shows the comparison of the power spectral density of signals spread with C/A-code and P-code. The plot shows that the P-code is wider.

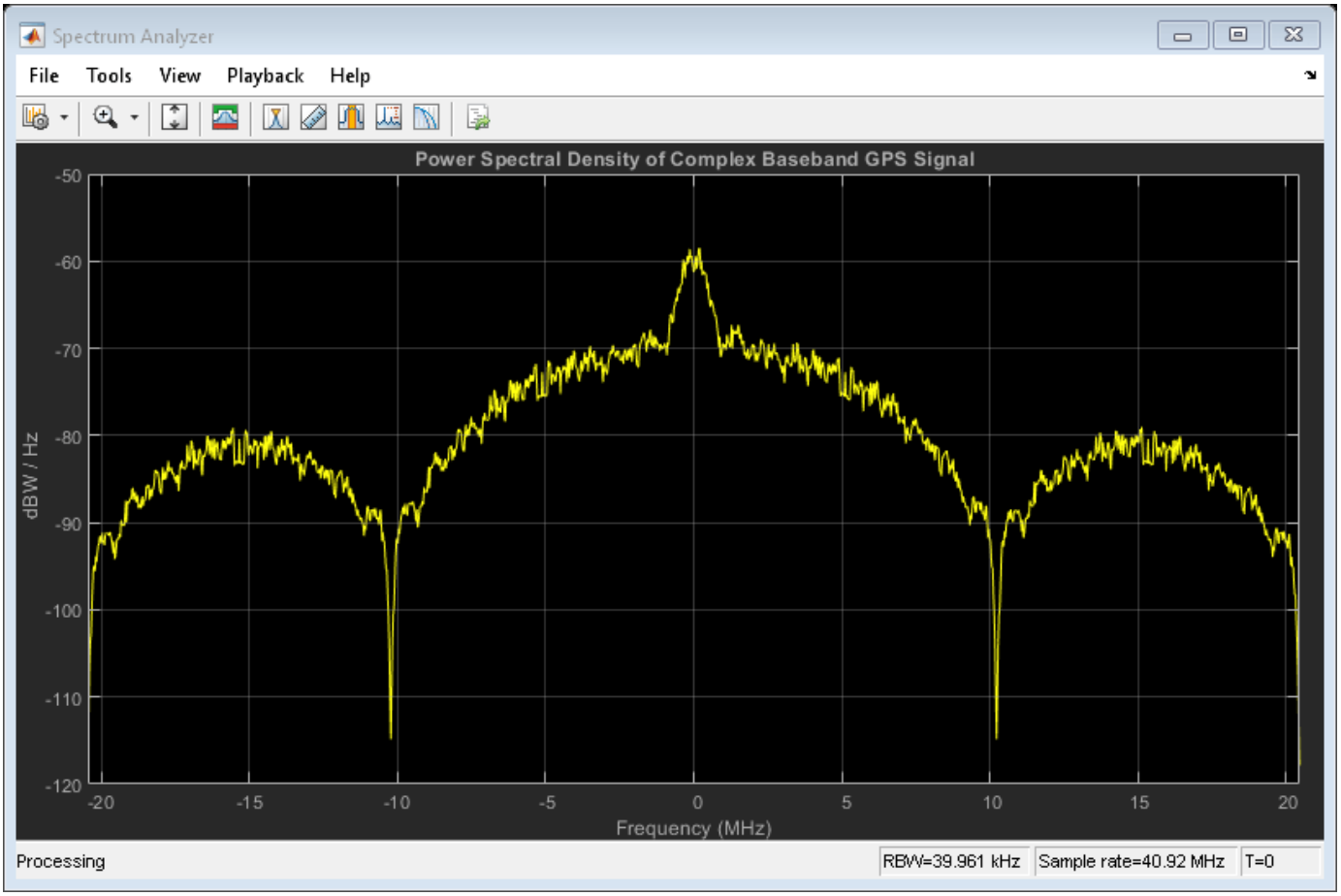
```
repeatFactor = 40;
% Repeat the generated BPSK signal of C/A-code to see the adjacent bands spectrum
updataCA = repmat(bpskSigCA(:).',repeatFactor,1);
updataCA = real(updataCA(:));
% Repeat the generated BPSK signal of P-code to see the adjacent bands spectrum
updataP = repmat(bpskSigP(:).',repeatFactor/10,1); % Repeat the P-code ten times less as ever
updataP = real(updataP(:));
caAndPCodeScope = dsp.SpectrumAnalyzer('SampleRate',prnCAChipRate*repeatFactor, ...
    'PlotAsTwoSidedSpectrum',true, ...
    'SpectrumType','Power density', ...
    'AveragingMethod','Exponential', ...
    'SpectrumUnits','dBW', ...
    'YLimits',[-130, -50], ...
    'Title','Comparison of Power Spectral Density of GPS C/A-Code and P-Code Signal', ...
    'ShowLegend',true, ...
    'ChannelNames',{'C/A-code spectrum','P-code spectrum'});

caAndPCodeScope([updataCA,updataP]);
```

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;
% Repeating the generated BPSK signal to see the adjacent bands spectrum
updata = repmat(gpsBBWaveform(:).', repeatFactor, 1);
updata = updata(:);
bbscope = dsp.SpectrumAnalyzer('SampleRate',10*prnCAChipRate*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 C/A-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 C/A-Code and P-Code Signal
C/A-code spectrum
P-code spectrum
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 three structures: `svparams`, `ephemerisparams`, and `almanacparams`. These structures generate GPS data bits and the navigation signal in the baseband. You can also replace the parameters in each of these structures and observe how the GPS data is generated. You can change the ephemeris parameters with an existing real data set and pass those parameters into the ephemeris. Additionally, you can specify your own almanac file. If using your own almanac file, the week number in the almanac file and the week number in the `svparams` structure 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 I-branch and a sine signal on the Q-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

- HelperGPSCACode.m - Generate C/A-code to spread data
- HelperGPSPCode.m - Generate P-code to spread data
- HelperGPSLNAVData.m - Create legacy navigation frame from data that is in structures

Bibliography

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

[2] ICD-GPS-240, Rev: C; Navstar GPS Control Segment to User Support Community Interfaces; March 4, 2019; Code Ident: 66RP1.

3

RF Propagation and Channel Models

Simulate and Visualize a Land Mobile-Satellite Channel

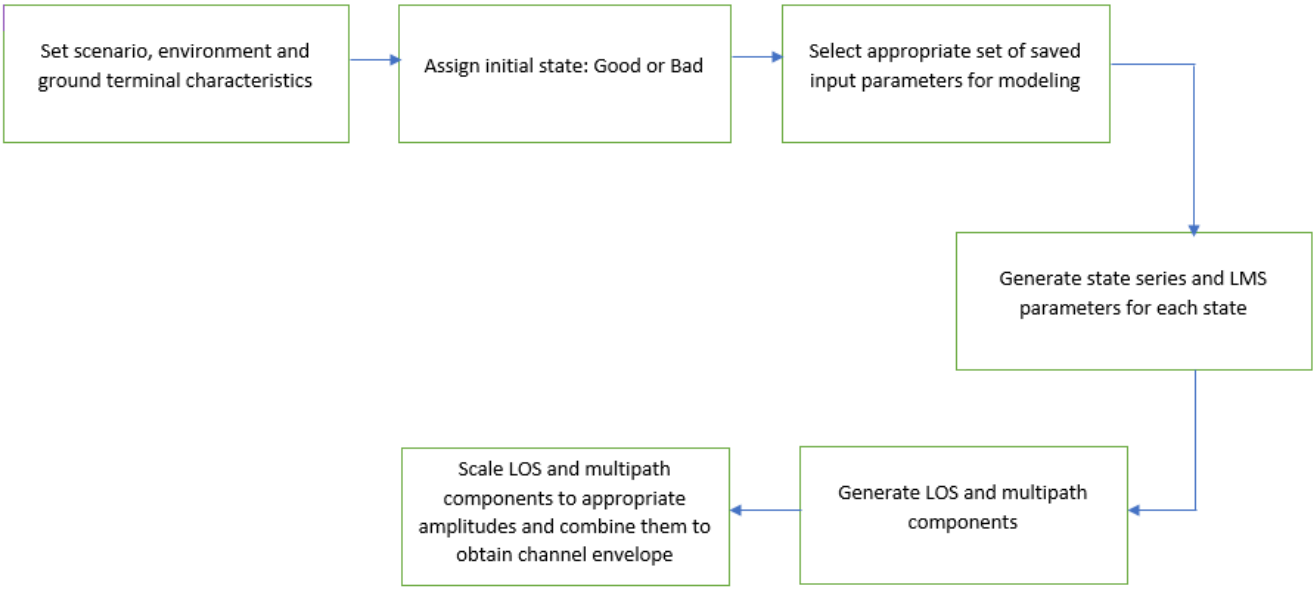
This example shows how to model a two-state land mobile-satellite (LMS) channel model by generating a state series, its respective space series, and the channel coefficients. In a scenario involving a satellite terminal and a mobile terminal, a signal being transmitted through the channel does not always have an ideal line-of-sight path, but 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 scenario 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.

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

Assign a suitable initial state for the model.

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

```
       CarrierFrequency: 3.8000e+09
         ElevationAngle: 45
               Velocity: 2
             SampleTime: 0.0025
     AzimuthOrientation: 0
           InitialState: "Bad"
    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`.

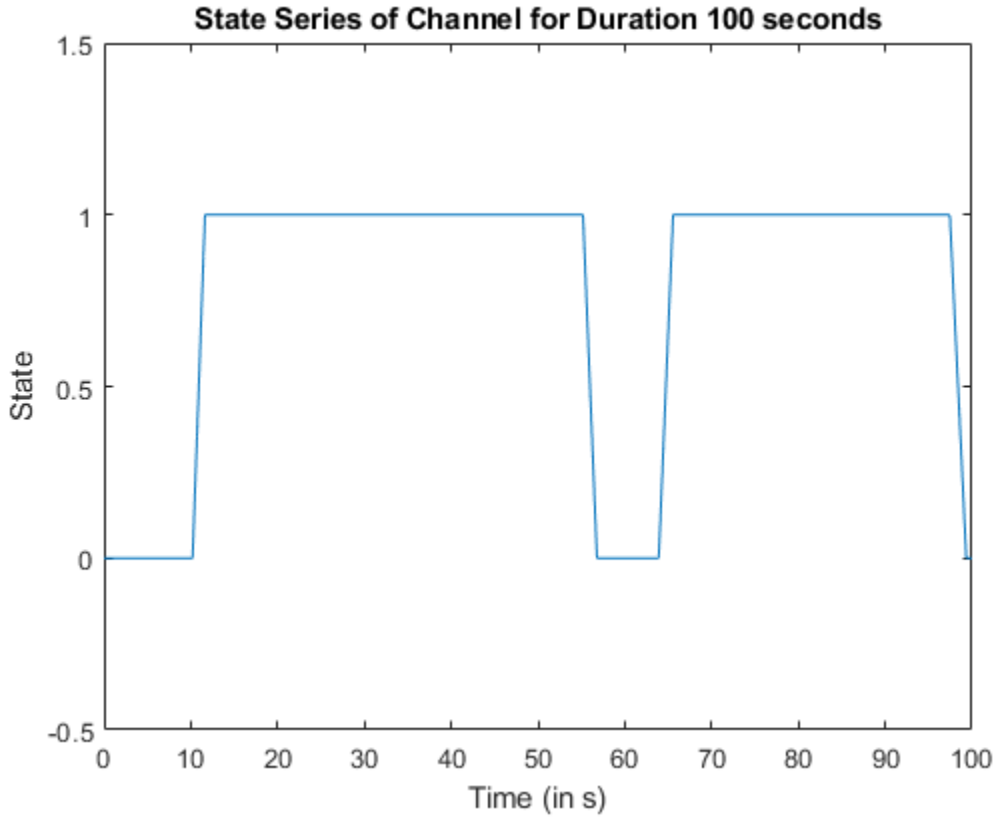
If you have LMS data, you can update the fields of the `paramsGoodState` and `paramsBadState` structures, and then pass the structures to the `HelperModelLMSChannel` helper function.

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

Channel Visualization

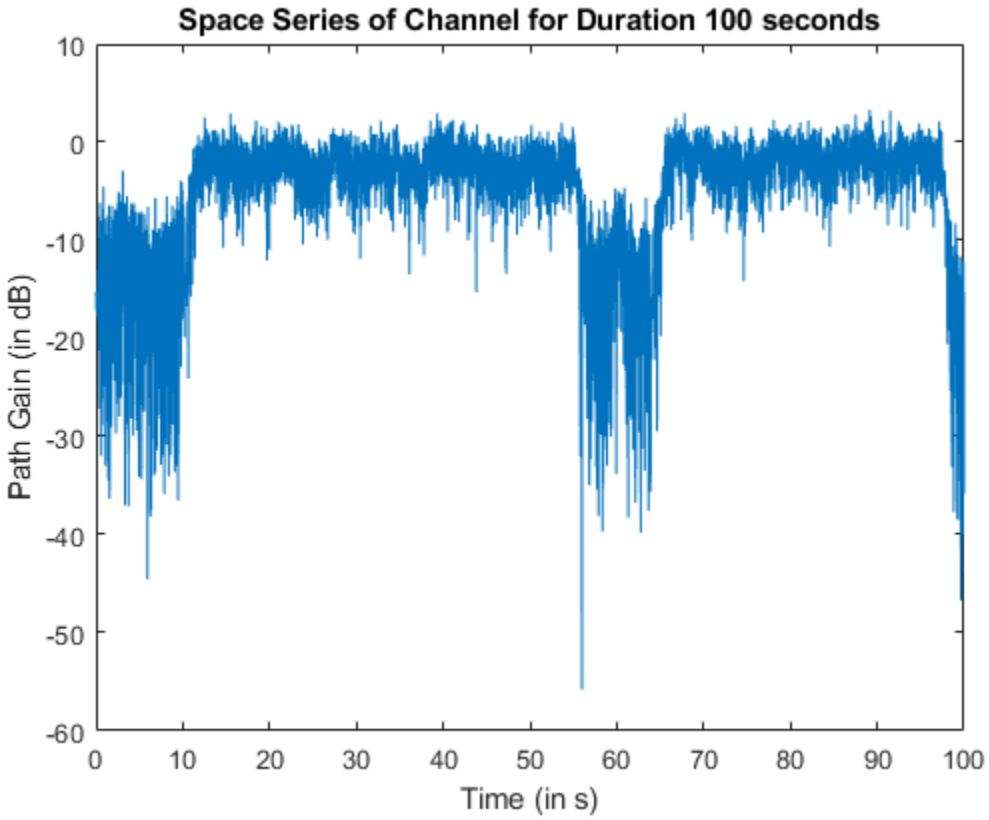
Plot the state series and its respective space series using the channel coefficients generated as a result of modeling.

```
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.

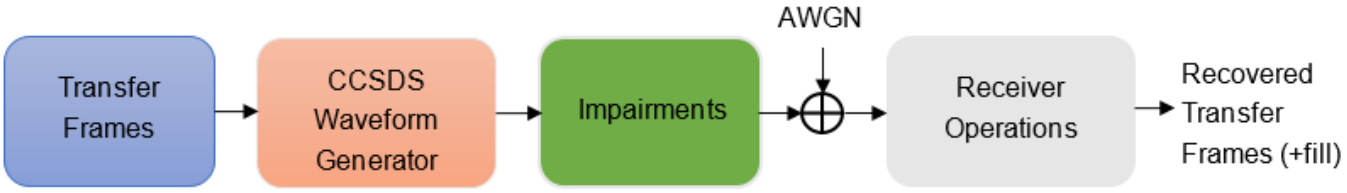
For each signal to noise ratio (SNR) point, CCSDS TC waveforms that are generated with a CLTU and acquisition sequence are distorted by RF impairments and passed through an AWGN channel. The example shows how to model these RF impairments:

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

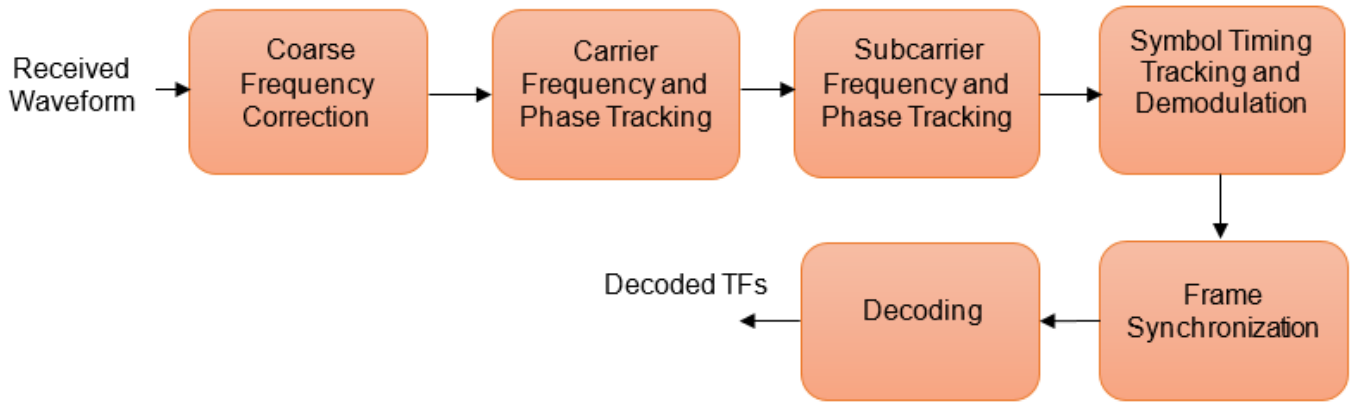
The CCSDS TC receiver compensates for the impairments, and the transfer frames (TFs) in the CLTUs are recovered. This example supports BPSK, PCM/PM/biphase-L, and PCM/PSK/PM modulation schemes. Subcarrier impairments are applicable only with the PCM/PSK/PM modulation scheme. These modulation schemes [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.



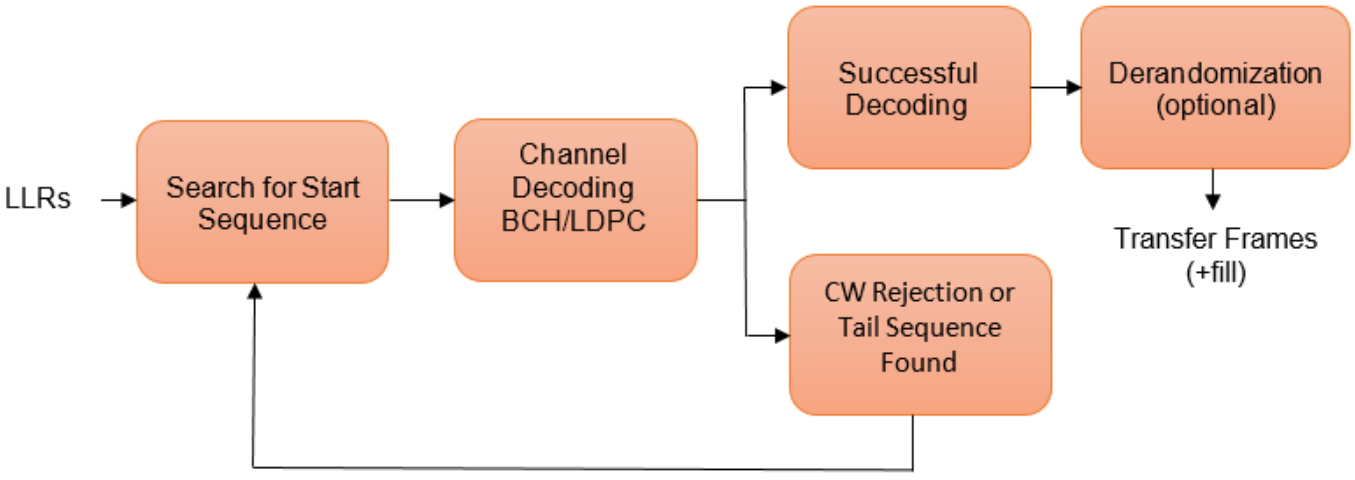
This figure shows the receiver operations, which include RF impairments compensation, demodulation, frame synchronization, and the decoding of transfer frames.



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"
```

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,subPhaseOffse
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( ...
```

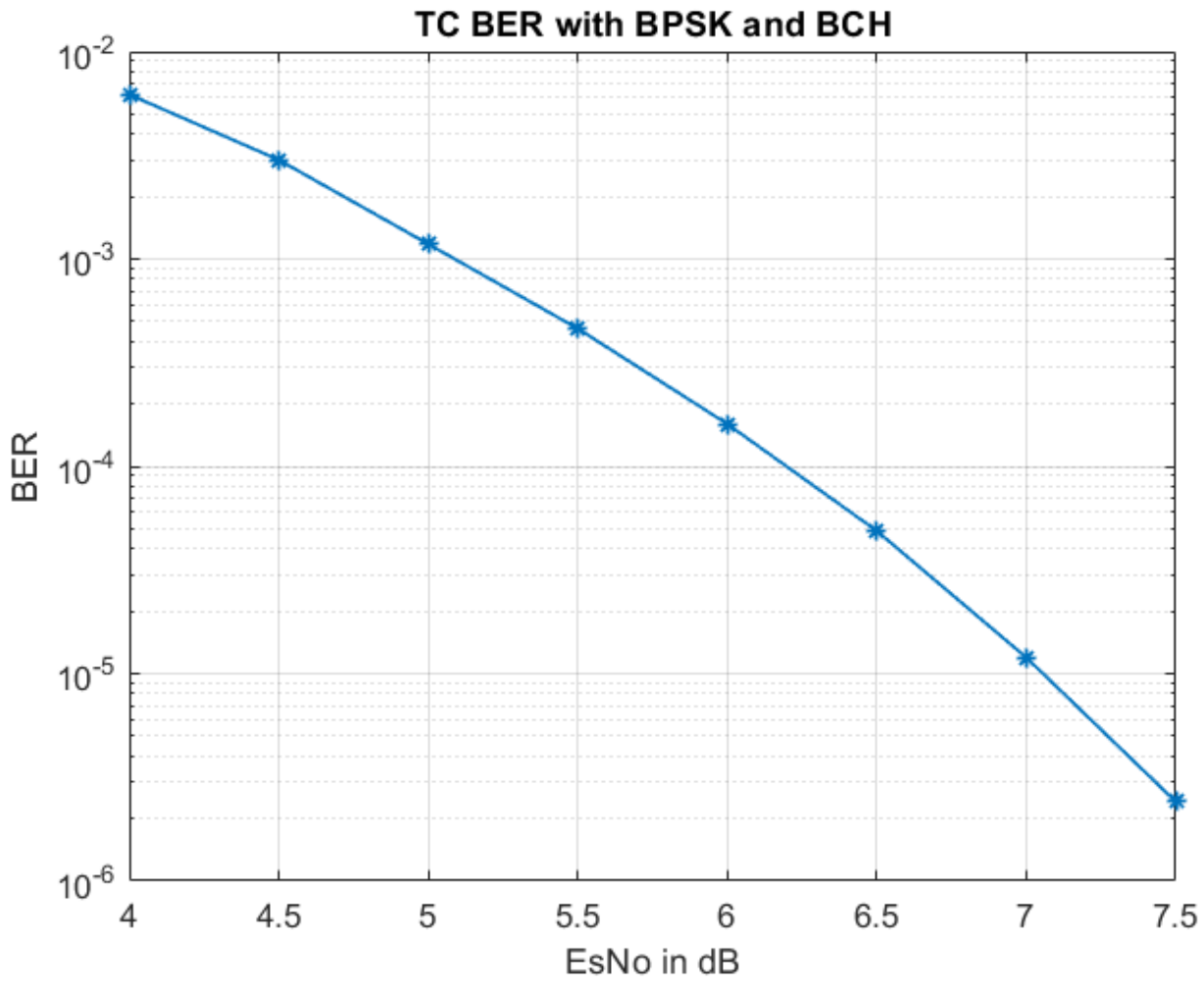
```
        'Modulation','BPSK','FrequencyResolution',100, ...
        'SampleRate',fs);
end

% 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

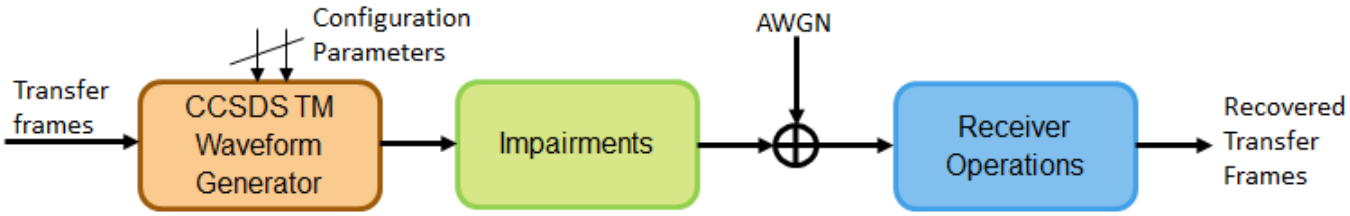
End-to-End CCSDS Telemetry Synchronization and Channel Coding Simulation with RF Impairments and Corrections

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

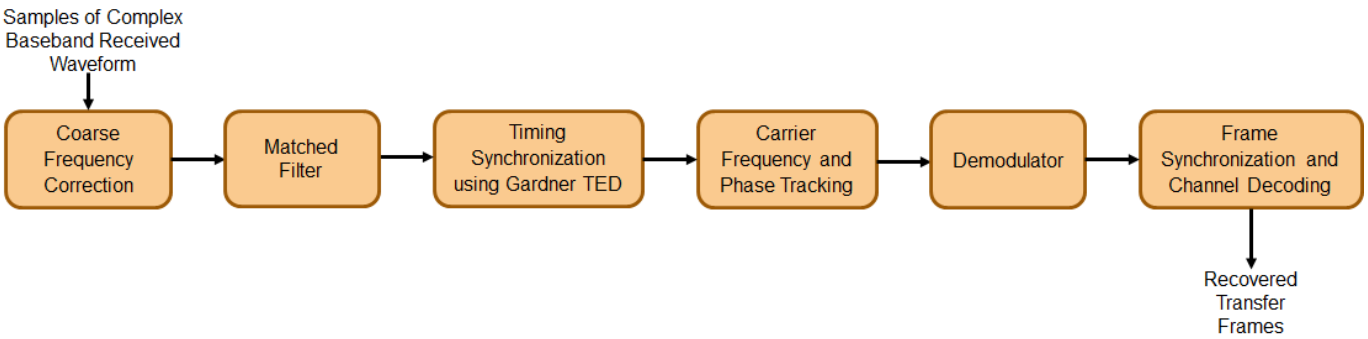
Data from various instruments is generated onboard the satellite. This data collectively is called as TM data. CCSDS specifies the coding and modulation schemes for the transmission of TM data from the satellite to an Earth station. This example shows the end-to-end simulation of the satellite to the 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)

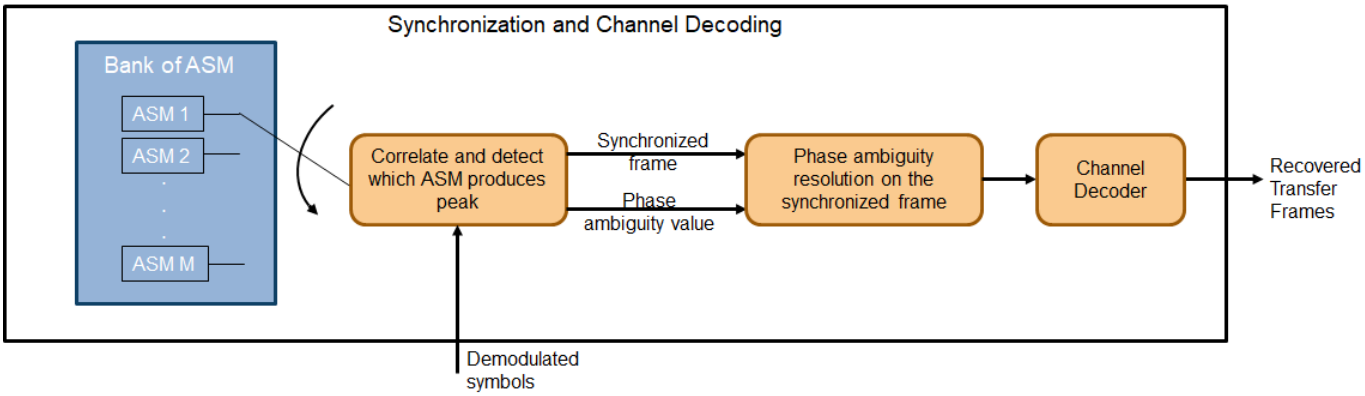
This figure shows the receiver side operations.



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 QPSK modulation scheme for signal generation and reception and 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 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 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 - this is useful while modeling RF impairments
cfo = 2e5;  % Percentage of symbol rate as CFO
```

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 BER curve, 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 irrespective of number of er
                     % 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 properties:
              ChannelCoding: "convolutional"
      ConvolutionalCodeRate: "1/2"

   Digital modulation and filter properties:
                 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 whi
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);
decoderobj = HelperCCSDSTMDecoder("ChannelCoding",channelCoding, ...
    "NumBytesInTransferFrame",transferFrameLength, ...
    "Modulation",modScheme);
disp(demodobj)
```

```
  HelperCCSDSTMDemodulator with properties:

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

```
disp(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
```

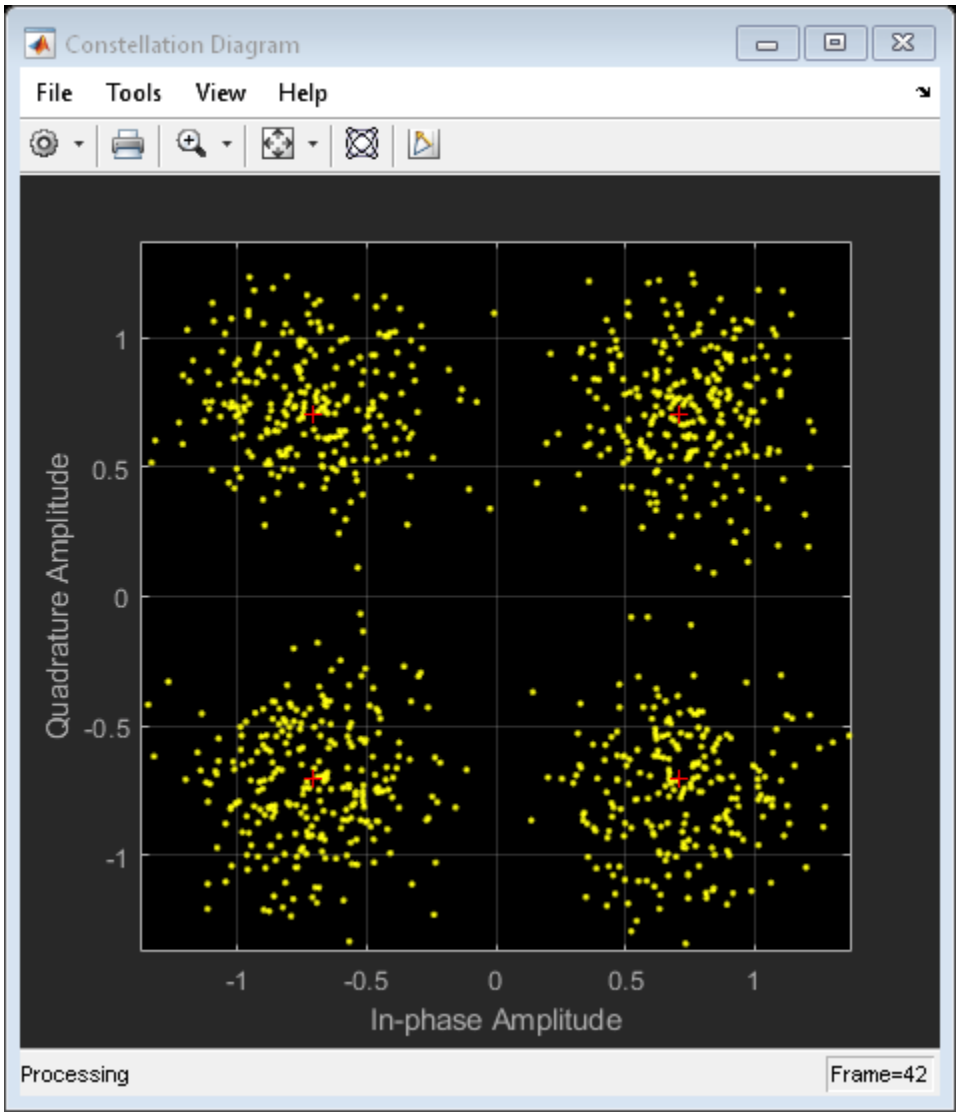
Processing Chain

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 for every SNR value to get repeatable results
    reset(bercalc);                       % Reset the comm.ErrorRate object for every SNR point
    berinfo = bercalc(int8(1), int8(1)); % Initialize berinfo to be used before BER is calculated
    tfidx = 1;
    numFramesLost = 0;
    prevdectfidx = 0;
    inputBuffer = zeros(numBitsInTF, 256,"int8");
    while((berinfo(2) < maxNumErrors) && (berinfo(3) < maxNumBits) && (numFramesLost < maxFramesL
        seed = randi([0 2^32-1],1,1);    % Generate seed and use in AWGN channel for repeatable s

        % 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 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 synchroniza

        % 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
                % value of decimal value of first 8 bits is not equal to the
                % previous 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(numl
                else
                    berinfo = bercalc(inputBuffer(:,dectfidx),decoded);
                    if nnz(inputBuffer(:,dectfidx)-decoded)
                        disp(['Errors occurred at tfidx: ' num2str(tfidx) '. Num errors: ' num2str(nr
                    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: ' num2s

        % 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 Eb/N0 value to 3.2:0.2:5 and observe the BER by setting `maxNumBits` to 1e8. Use 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 synchronizers (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

`ccsdsTCConfig` | `ccsdsTMWaveformGenerator`

Related Examples

- “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

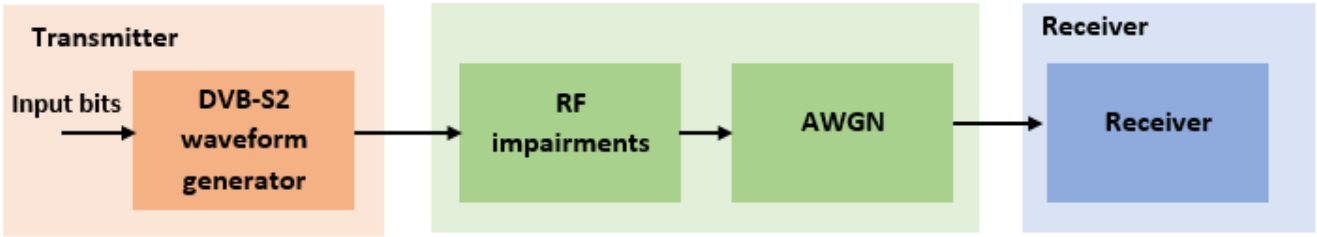
This example shows how to measure the bit error rate (BER) and packet error rate (PER) of a single stream Digital Video Broadcasting Satellite Second Generation (DVB-S2) link that has constant coding and modulation. The example describes the symbol timing and carrier synchronization strategies in detail, 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

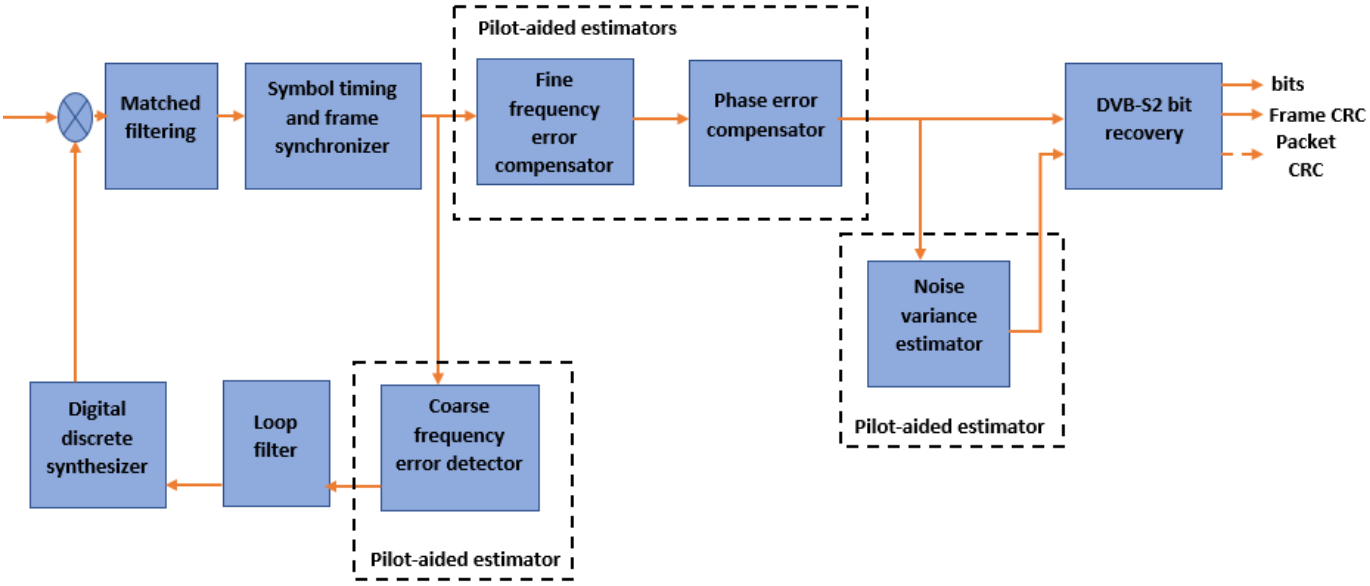
The example processes 25 physical layer (PL) frames of data with the E_s/N_o set to 20 dB, and then computes the BER and PER. Carrier frequency offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

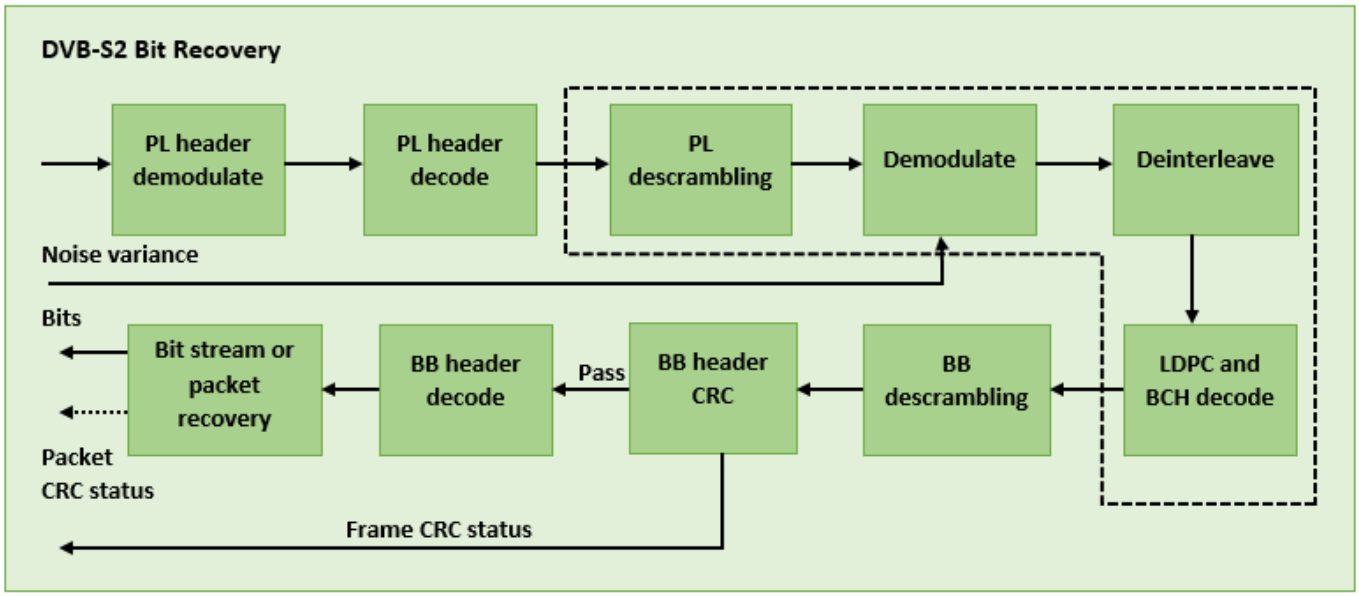
At the receiver, after matched filtering, timing and carrier recovery operations are run to recover the transmitted data. To extract PL frames, the distorted waveform is processed through various timing and carrier recovery strategies to extract PL frames. The carrier recovery algorithms are pilot-aided. 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 mi

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

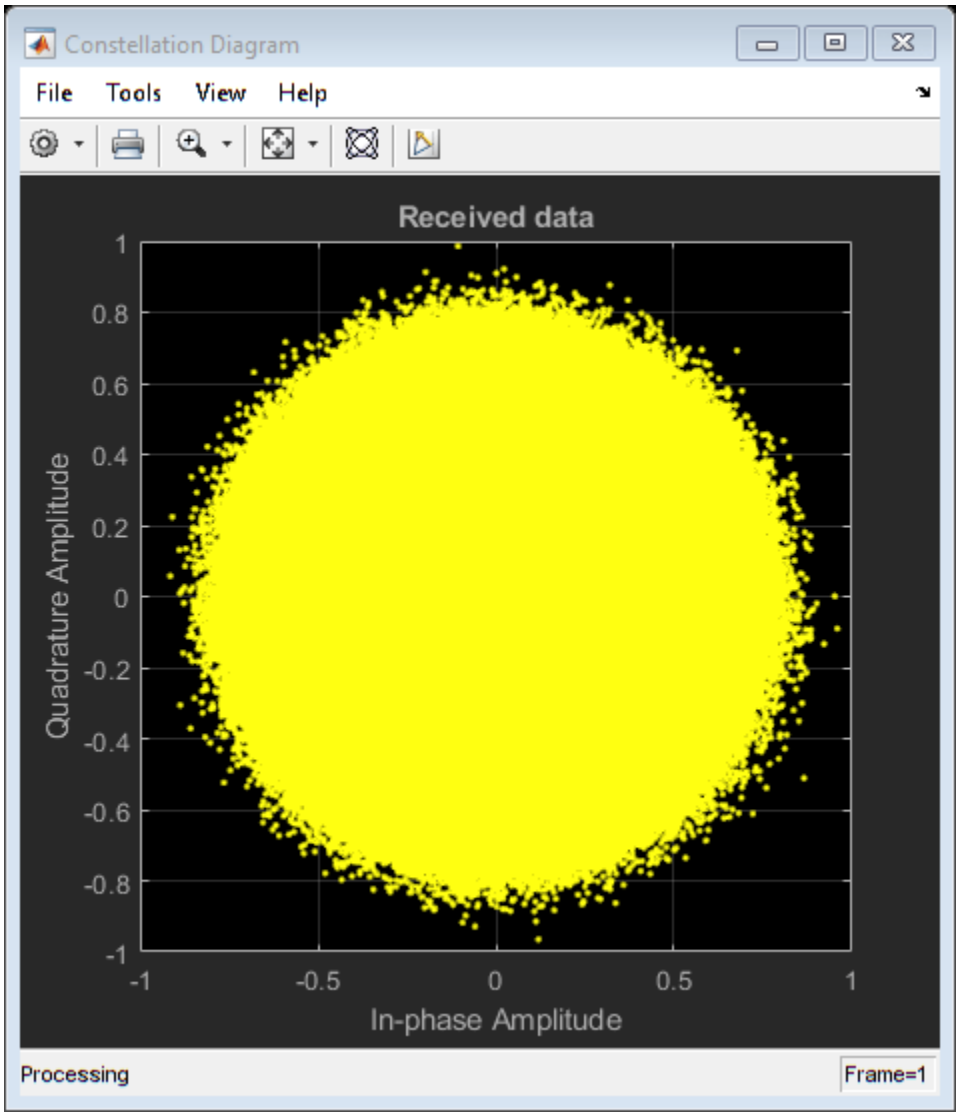
This table defines the phase noise mask (dBc/Hz) used to generate the phase noise applied to the transmitted signal.

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

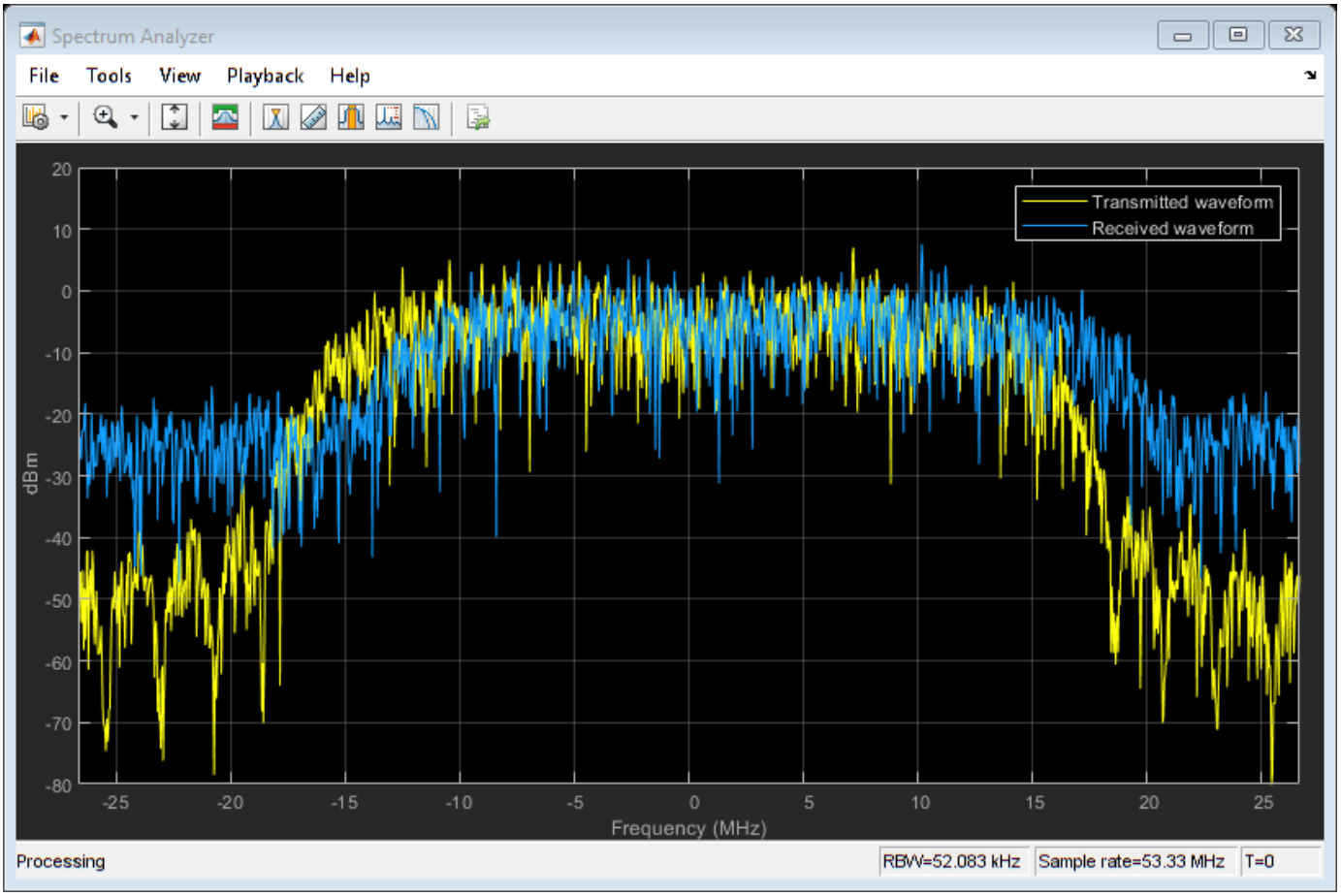
Generate DVB-S2 Waveform Distorted with RF Impairments

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 normali
rxParams.symbSyncLoopBW = 8e-3;              % Symbol timing synchronizer loop bandwidth normali
rxParams.symbSyncLock  = 6;                  % Number of frames required for symbol timing error
rxParams.frameSyncLock = 1;                  % Number of frames required for frame synchronizatio
rxParams.coarseFreqLock = 3;                 % Number of frames required for coarse frequency acq
rxParams.fineFreqLock = 6;                   % Number of frames required for fine frequency estim
rxParams.hasFinePhaseCompensation = false;   % Flag to indicate whether fine phase compensation i
rxParams.finePhaseSyncLoopBW = 3.5e-4;       % Fine phase compensation loop bandwidth normalized

% Total frames taken for symbol timing and coarse frequency lock to happen
rxParams.initialTimeFreqSync = rxParams.symbSyncLock + rxParams.frameSyncLock + rxParams.coarseFr
% 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 bandwid
    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 f
            syncIn = [rxData;zeros(resSampCnt, 1)];
        else                                                    % Excess samples are available to f
            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 wi
    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 de
% 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.
    phErr = exp(-1j*2*pi*freqEst*((rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1
    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,phEstRes(2),rxParams.pilotInd);
            % 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,estEndPh,rxParams.pilotInd);
            % 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)/rxParams.plFrameSize
            rxFrame = syncOut((frameCnt-1)*rxParams.plFrameSize+1:frameCnt*rxParams.plFrameSi
            % Estimate noise variance by using
            % HelperDVBS2NoiseVarEstimate helper function.
            nVar = HelperDVBS2NoiseVarEstimate(rxFrame,rxParams.pilotInd,rxParams.refPilots,r
            % 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.
            if isLastFrame && ~isFrameLost
                bitsErr = bitsErr + sum(data((dataStInd-1)*dataSize+1:dataStInd*dataSize) ~=
            else
```

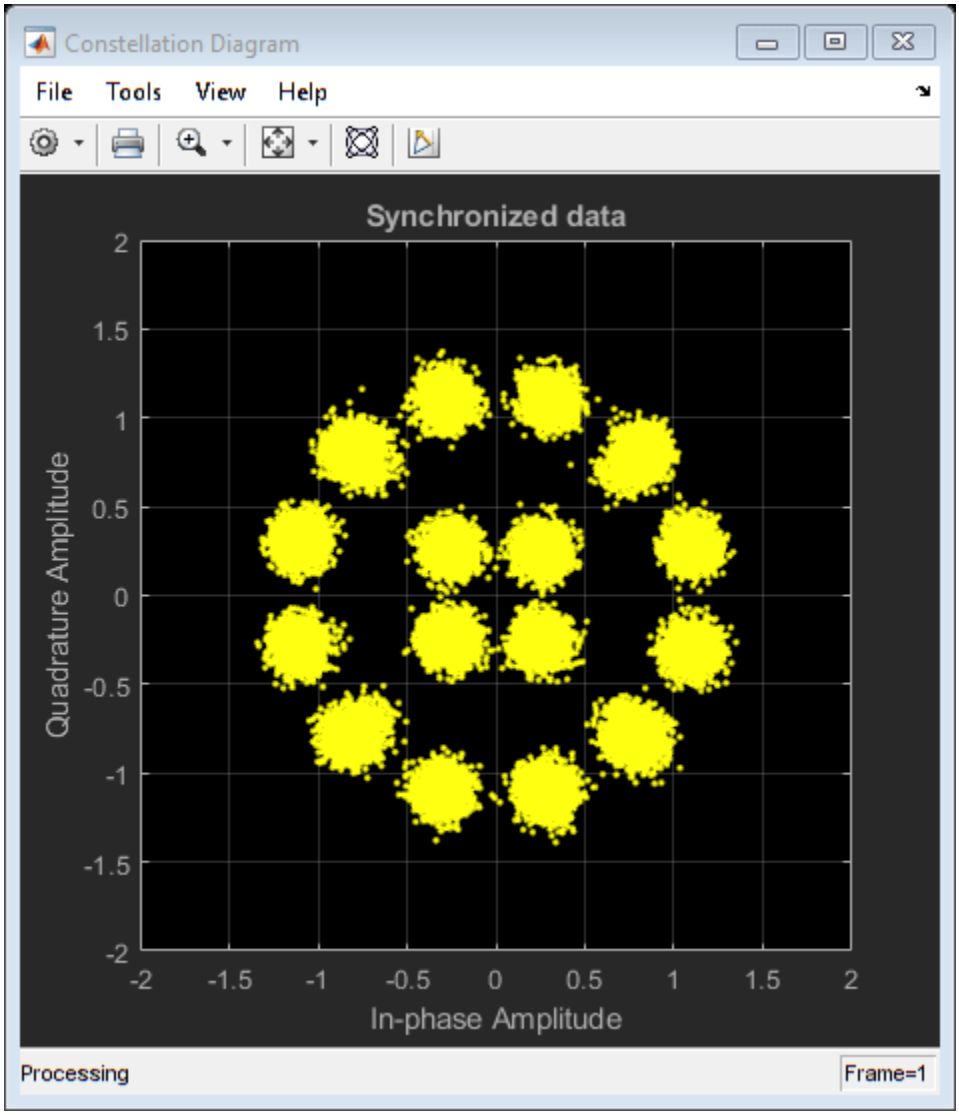
```
                    if ~isFrameLost
                        bitsErr = bitsErr + sum(data((dataStInd-1)*dataSize+1:dataStInd*dataSize)
                    end
                end
                dataStInd = dataStInd + 1;
            end
        end
    end
    stIdx = endIdx;
end
```

```
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)
```




```
% 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 4-38

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

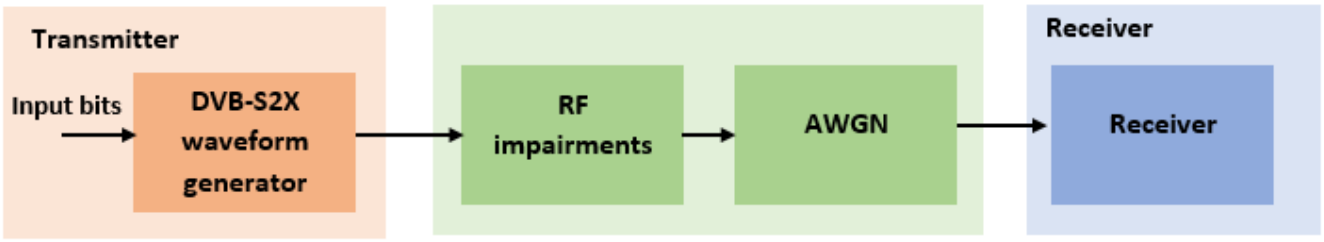
The DVB-S2X standard, an extension of the DVB-S2 specification, enhances the support provided for core DVB-S2 applications and improves overall efficiency over satellite links. The DVB-S2X standard supports these additional features:

- 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

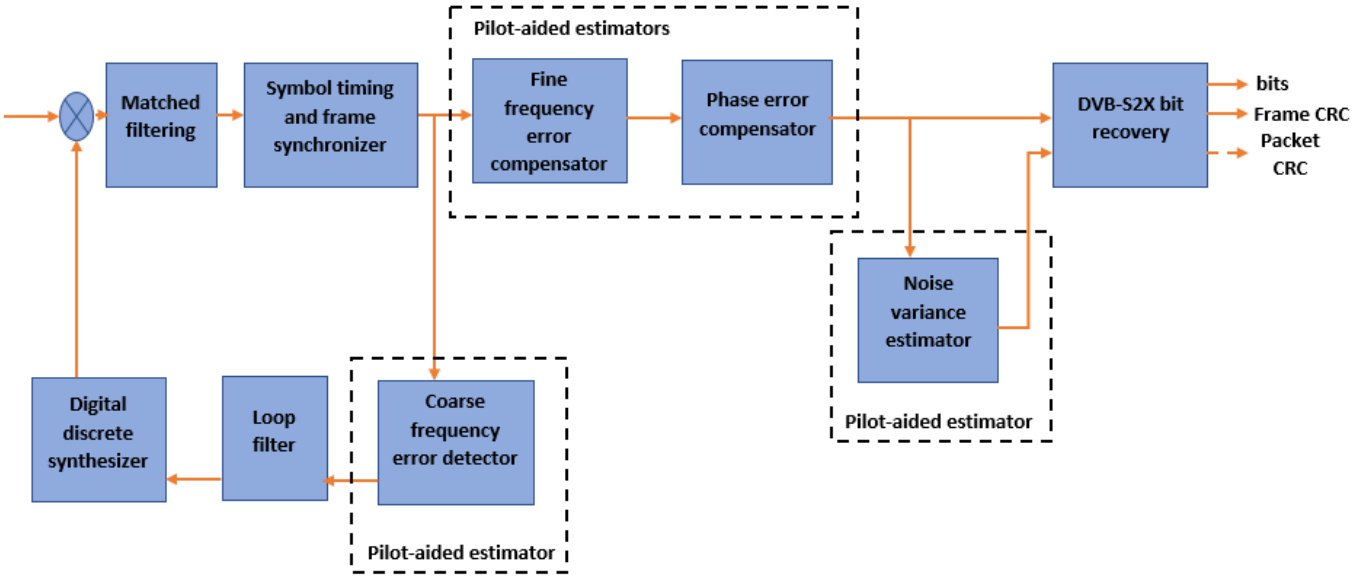
The example processes 30 physical layer (PL) frames of data with the E_s/N_o set to 25 dB, and then computes the BER and PER. Carrier frequency offset, sampling clock offset, and phase noise impairments are applied to the modulated signal, and AWGN is added to the signal.

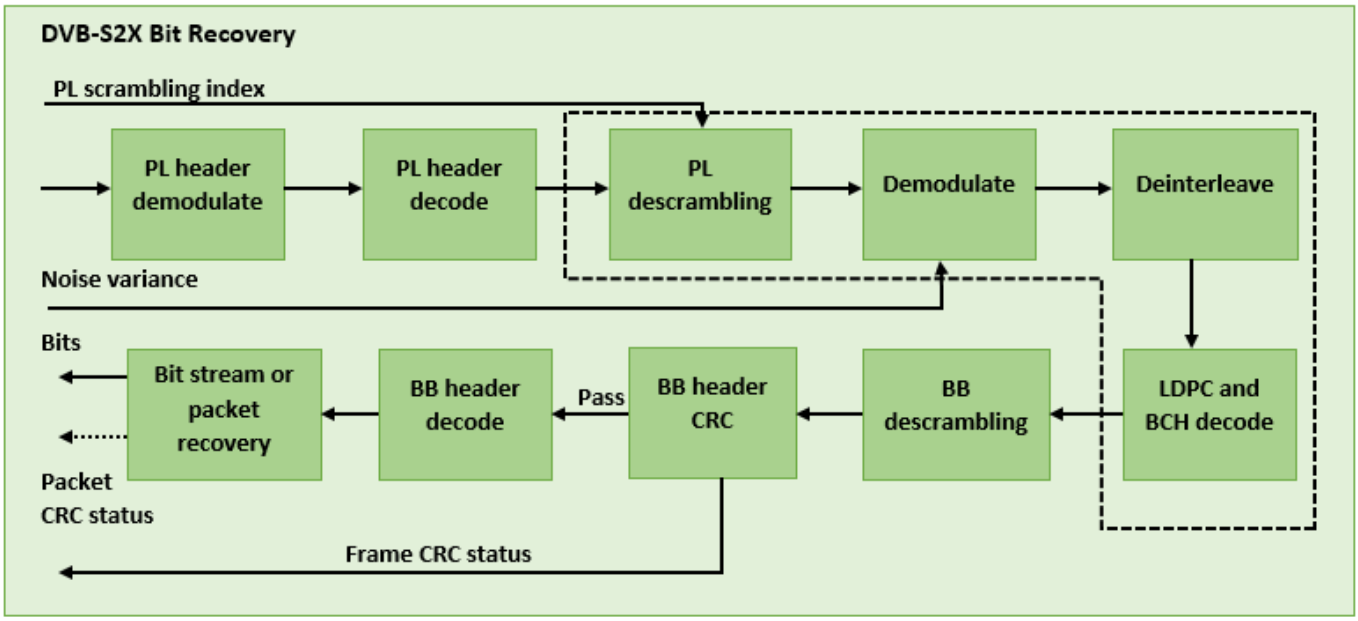
At the receiver, after matched filtering, timing and carrier recovery operations are run to recover the transmitted data. To extract PL frames, the distorted waveform is processed through various timing

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 mi

simParams.phNoiseLevel = Low ;                            % Phase noise level provided as 'Low',
simParams.EsNodB = 25;                                   % Energy per symbol to noise ratio in de
```

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

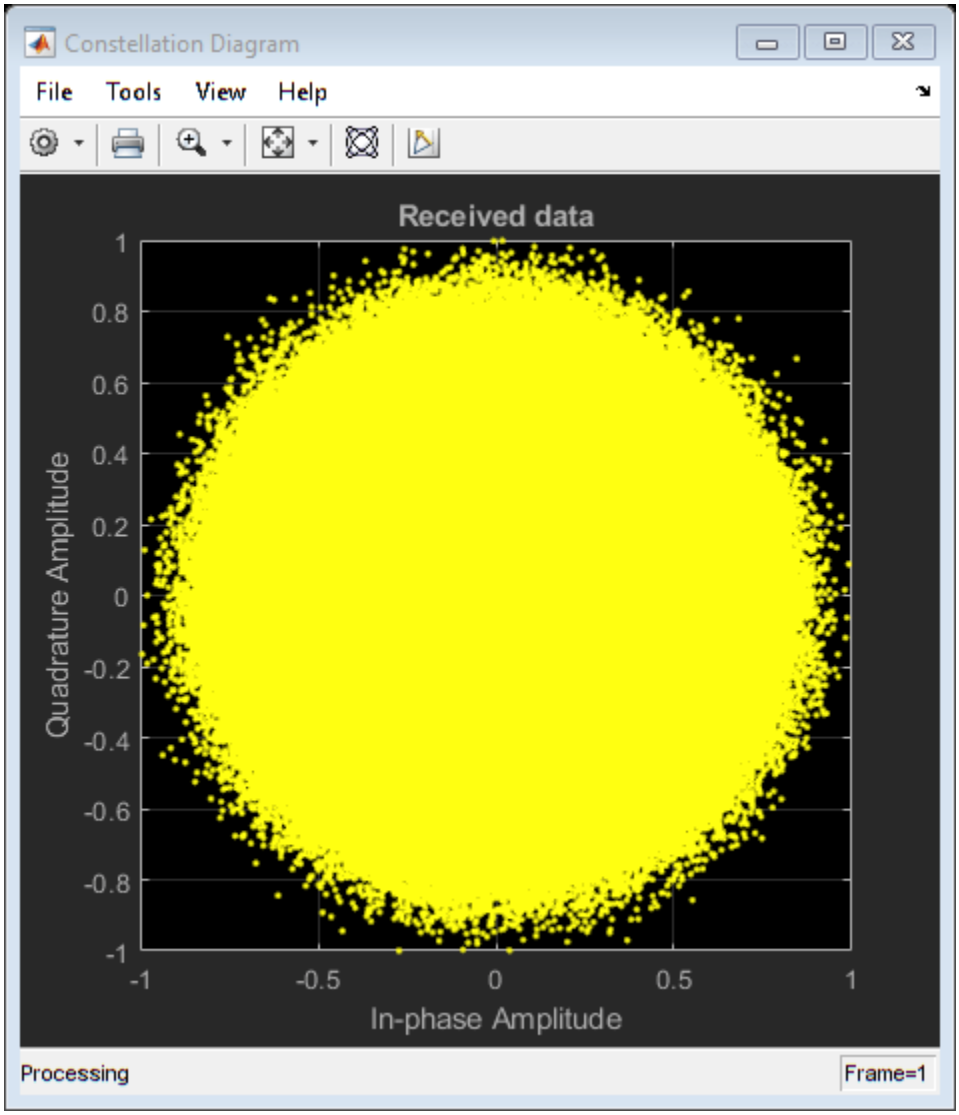
Generate DVB-S2X Waveform Distorted with RF Impairments

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.

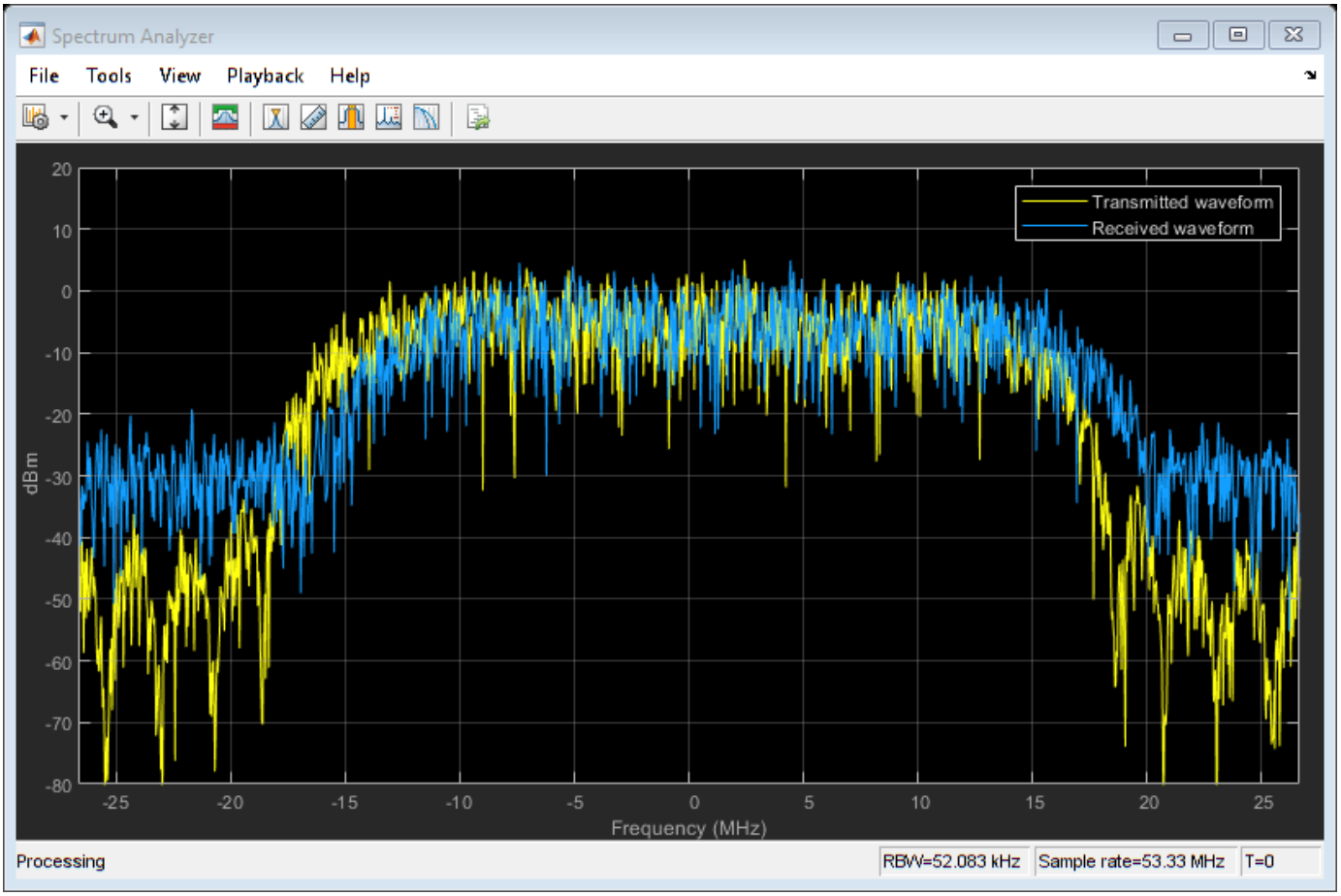
```
[data,txOut,rxIn,phyConfig,rxParams] = HelperDVBS2XRxInputGenerate(cfgDVBS2X,simParams);
disp(phyConfig)
```

```
              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 normali
rxParams.symbSyncLoopBW = 8e-3;              % Symbol timing synchronizer loop bandwidth normali
rxParams.symbSyncLock  = 8;                  % Number of frames required for symbol timing error
rxParams.frameSyncLock = 1;                  % Number of frames required for frame synchronizatio
rxParams.coarseFreqLock = 5;                 % Number of frames required for coarse frequency acq
rxParams.fineFreqLock = 4;                   % Number of frames required for fine frequency estim

% Total frames taken for symbol timing and coarse frequency lock to happen
rxParams.initialTimeFreqSync = rxParams.symbSyncLock + rxParams.frameSyncLock + rxParams.coarseFr
% 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 bandwid
    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
            syncIn = [rxData; zeros(resSampCnt, 1)];
        else                                        % Excess samples are available to fill last
            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 wi
        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 de
% 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.
    phErr = exp(-1j*2*pi*freqEst*((rxParams.frameCount-2)*plFrameSize:(rxParams.frameCount-1)
    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,phEstRes(2),rxParams.pilotInd);
    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,estEndPh,rxParams.pilotInd);
    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)*rxParams.plFrameSize+1:frameCnt*rxParams.plFrameSi
            % Estimate noise variance by using
            % HelperDVBS2NoiseVarEstimate helper function.
            nVar = HelperDVBS2NoiseVarEstimate(rxFrame,rxParams.pilotInd,rxParams.refPilots,r
            % 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.plScramblin
                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.
                if isLastFrame && ~isFrameLost
                    bitsErr = bitsErr + sum(data((dataStInd-1)*dataSize+1:dataStInd*dataSize) ~=
                else
                    if ~isFrameLost
                        bitsErr = bitsErr + sum(data((dataStInd-1)*dataSize+1:dataStInd*dataSize
                    end
                end
                dataStInd = dataStInd + 1;
            end
        end
    end
    stIdx = endIdx;
end
```

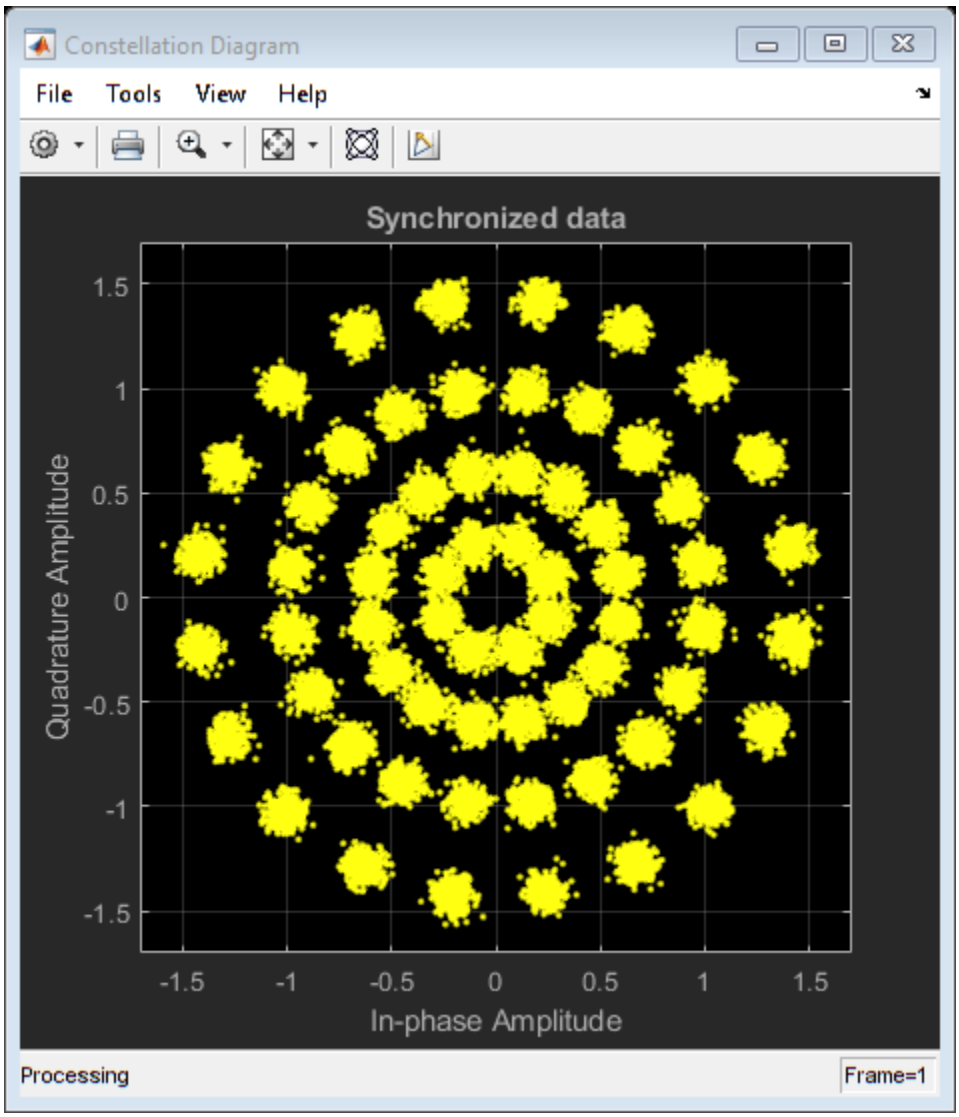
```
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number of frames lost = 0)
BB header decoding passed.(Number 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 4-22 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

`dvbs2WaveformGenerator` | `dvbs2xWaveformGenerator`

Related Examples

- “End-to-End DVB-S2 Simulation with RF Impairments and Corrections” on page 4-22

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