ZEIT8219
Satellite Communications

Assignment 1

NINA AVERILL
z3531215

UNSW Canberra
Apr 2022

Contents

1 Report Summary **3**

2 Satellite Constellation **4**
2.1 Definition 4
2.2 Technical Reasoning 4
2.2.1 Launch Costs 5
2.2.2 Satellite Costs 5
2.2.3 Satellite Lifespan 5
2.2.4 Ground Infrastructure complexity and costs 5
2.2.5 Service Availability 5
2.2.6 Coverage Extent 6
2.2.7 Latency 6

3 Ground Station Link Analysis **6**
3.1 Coverage 6
3.2 Link Analysis 8

4 List of Figures **11**

5 Appendix **18**
5.1 GMAT Script 18
5.2 Python Code Samples 31

1 Report Summary

This report will outline a constellation of three inclined geosynchronous satellites designed to provide continuous coverage of mainland Australia, Tasmania and various remote offshore territories. The locations of the remote offshore territories are described below:

Remote Territory	Latitude (deg)	Longitude (deg)
Ashmore Island	12°11'S	122°59'E
Cartier Island	12°31'S	123°33'E
Casey Research Station	66°17'S	110°32'E
Davis Research Station	68°35'S	77°58'E
Mawson Research Station	67°36'S	62°52'E
Macquarie Island Research Station	54°30'S	158°57'E
Christmas Island	10°25'S	105°43'E
Cocos (Keeling) Island	12°10'S	96°50'E
Coral Sea Islands	23°15'S	155°32'E
Heard & McDonald Islands	53°05'S	73°30'E
Norfolk Island	29°02'S	167°57'E

Table 1: Remote Offshore Territories locations in Geographic Coordinates

In the following, we will define the constellation of satellites, as well as provide proof of coverage and perform a minimal link analysis for three of the remote offshore territories. For the purposes of this analysis we will assume a spherical Earth with radius equal to 6371 km and no orbital perturbations, i.e. gravitational attraction is the only force acting on the satellite.

GMAT is the primary tool used for orbit propagation and visualisation, as well as coordinate transformation. All equations can be assumed to be sourced from Principles of Satellite Communications (Ryan, 2021) unless stated otherwise.

2 Satellite Constellation

2.1 Definition

To guarantee continuous coverage for all of Australia and its offshore territories, we defined a constellation of three inclined geosynchronous satellites. The Keplerian orbital parameters of the constellation in the J2000 inertial reference frame are defined in Table 2.

Satellite name	Semi-major axis (km)	Eccen-tricity	Inclination (deg)	Right ascension of the ascending node (deg)	Argument of perigee (deg)	True Anomaly (deg)
COMMSAT1	42165	0	30	30	0	0
COMMSAT2	42164	0	30	150	0	240
COMMSAT3	42165	0	30	270	0	120

Table 2: Keplerian orbital parameters of the constellation in the J2000 inertial coordinate system

The semi-major axis and eccentricity were specified such that the orbital period is equal to the length of the sidereal day, defining the orbital regime as geo-synchronous. An inclined orbit increases the north-south ground trace, ensuring coverage for the southernmost Earth stations. A constellation of three satellites ensured that there was always a satellite in view of the Earth stations, and there is no risk of collision at the point of intersection of the orbits. The right ascension of the ascending node and true anomaly are offset such that the satellites had to be equally placed in the ground trace, which is located roughly in the centre of the span of coverage.

Given these initial orbital parameters, GMAT was used to propagate the orbit for a full sidereal day from the UTC datetime 01 Jan 2000 11:59:28.000 [5.1]. Figures 2 and 3 show the orbits of the constellation with respect to the J2000 inertial frame defined above, displaying the X-Y and ecliptic planes respectively. Figures 4 and 5 further show the orbits with respect to an earth-fixed reference frame, displaying the X-Y and ecliptic planes respectively.

2.2 Technical Reasoning

The constellation of inclined geosynchronous orbits was selected as it met the minimal criteria of continuous coverage of all of the mainland and remote ground stations. The broad benefit of this orbital height is that it provides a large coverage area relative to the number of satellites in the constellation. This orbit has a number of benefits and implications that will be explored below.

2.2.1 Launch Costs

The orbit regime selected will require a launch inclination greater than the minimum orbital inclination. At thirty degrees inclination, there are a number of launch sites where this is possible. Therefore, the satellite will have to perform a smaller, and therefore less expensive, plane change manoeuvre than most geostationary launches. However, the satellites will still have to perform a geostationary transfer orbit to reach the final parking orbit. The greater fuel (and therefore weight) requirements will incur additional launch costs than a comparable Lower Earth Orbit (LEO) constellation.

2.2.2 Satellite Costs

Geosynchronous Equatorial Orbit (GEO) satellites tend to be more expensive to develop due to their increased size and system complexity compared to LEO and Medium Earth Orbit (MEO) constellations. However, this comes with the benefit of a reduced number of satellites in the constellation required to provide the same area of coverage.

2.2.3 Satellite Lifespan

Satellites in the geostationary orbit tend to have much greater mission lifespans than lower orbits, with most missions planning for around fifteen year life-spans. The main limitations on the satellite's lifetime is the fuel requirements for station-keeping and disposal, and the degradation of on-board systems from the harsh radiation environment. Lower orbit regimes generally have a much shorter lifespan of 7-10 years, with the main limitation being orbit deterioration as a result of atmospheric drag.

2.2.4 Ground Infrastructure complexity and costs

Geosynchronous orbits tend to have less complex ground station requirements as remain effectively stationary with respect to the Earth. However as the orbit regime chosen has a significantly inclined orbit, the ground track traces a figure-eight path and the ground stations will need to have some tracking capabilities. Moreover, the large propagation distance will lead to greater requirements for high-power transmitters, sensitive receivers and high-gain antennas at the ground station.

2.2.5 Service Availability

The main risks to service availability for the geo-inclined orbits come from solar interference, and solar eclipses from the Earth and the Moon. Sun-transit outages occur when the beam of the ground station antenna is pointed directly at the sun, with the broad-spectrum radio frequency energy emitted from the sun causing disruption to the receive signal. As seen in Figures 3 and 5, the Earth's ecliptic plane intersects with the orbital plane of the satellite and so there is some significant risk of interference. Geo-synchronous orbits tend to experience less frequent Earth and Lunar eclipses than lower orbits, but precautionary measures still need to be taken to ensure that the satellites have access to backup power sources, or reduced service capabilities during the eclipse period.

2.2.6 Coverage Extent

The constellation meets the minimal requirement for continuous coverage for all ground stations. In addition to this, Figure 6, 8, and 7 indicate that there is often multiple satellites in view at any one moment, this provides additional redundancy in case one satellite in the constellation is experiencing an outage.

2.2.7 Latency

Due to the greater orbital height, geo-synchronous satellites have significantly greater latency than lower orbit regimes. The minimum time taken for a communication round trip is around 250 ms, making it ineffective for some communication protocols that require near real-time interactions.

3 Ground Station Link Analysis

3.1 Coverage

A best-case analysis of the ground station coverage was conducted for the Cocos Islands, Mawson Research Station and Norfolk Island. To evaluate the optimal point in the satellites' orbit, the minimum slant range was calculated with respect to each ground station. Slant range is generally considered to be a good indicator of the quality of communication as a smaller propagation distance generally indicates less path attenuation.

As all three satellites trace the same path and have the same transmit characteristics, each would exhibit the same link behaviour with each ground station over the course of the orbit. The optimal link characteristic were therefore calculated with reference to COMMSAT1, but equivalent values could be found with reference to COMMSAT2 and COMMSAT3.

GMAT was used to propagate the orbits over the course of a sidereal day and output the latitude and longitude of the sub-satellite point and orbital radius for each time step. Equation 1 (2.20, Ryan) [??:57-86] calculates the angle γ at the centre of the Earth.

$$\cos(\gamma) = \cos(L_E)\cos(L_S)\cos(l_e - l_s) + \sin(L_e)\sin(L_S) \tag{1}$$

Where:

L_E is the latitude of the Earth Station

L_S is the latitude of the satellite

l_E is the longitude of the Earth Station

l_S is the longitude of the satellite

From this, we can calculate the slant range using Equation 2 (2.22, Ryan) [??:111-132].

$$d_s = \sqrt{r_E^2 + r_S^2 - 2r_E r_S \cos(\gamma)} \quad (km) \tag{2}$$

Where:

r_E is the radius of the Earth

r_S is the orbital radius of the satellite

The slant range was calculated for each time step and the minimum value was identified. The optimal position for each ground station is listed in Table 3:

Ground Station	Datetime (UTC)	Sub-Satellite Latitude (deg)	Sub-Satellite Longitude (deg)	Altitude (km)
Cocos Island	2000-01-02 01:54:28	-14.2	106.3	35785.8
Mawson Station	2000-01-02 05:34:28	-29.9	108.8	35790.7
Norfolk Island	2000-01-02 07:44:28	-29.4	111.3	35790.4

Table 3: Optimal Position with respect to COMMSAT1 over the course of a sidereal day

The elevation angle was then calculated using Equation 3 (2.27, Ryan) [??:135-161].

$$\epsilon = \arccos \frac{\sin\gamma}{\sqrt{1 + \frac{r_E}{r_S}^2 - 2\frac{r_E}{r_S}\cos\gamma}} (deg) \tag{3}$$

The optimal slant range and elevation angle for each ground station are shown in Table 4

Ground Station	Datetime (UTC)	Min Slant Range (km)	Elevation (deg)
Cocos Island	2000-01-02 01:54:28	35885.9	78.9
Mawson Station	2000-01-02 05:34:28	38042.8	36.7
Norfolk Island	2000-01-02 07:44:28	38203.1	41.3

Table 4: Optimal Slant Range and Elevation with respect to COMMSAT1 over the course of a sidereal day

Figures 6, 7 and 8 show the elevation of the satellites with respect to each ground station over a sidereal day. At each time step, there are one or more satellites that have elevations above the minimum of 20 degrees. Therefore, the constellation guarantees contact across the full orbit period.

3.2 Link Analysis

For the purposes of this analysis, the satellite transmit antennas were assumed to have the following propagation characteristics:

Transmit power (W)	5
Transmit gain (dB)	30
Frequency (GHz)	20

The ground station receive antennas were assumed to have the following propagation characteristics:

Receive gain (dB)	16

Feeder losses, pointing losses, and atmospheric propagation losses were omitted from this analysis. The power collected by the ground station's receive antenna is therefore calculated from the propagation model given in Equation 8.

The Effective Isotropic Radiated Power (EIRP) was calculated using equation 5 (4.5, Ryan) [3:25-43].

$$EIRP = P_t G_t \quad (W) \tag{4}$$

Where:

P_t is the power of the transmit antenna

G_t is the gain of the transmit antenna

From this, we can calculate the receive power density using equation 6 (4.7, Ryan) [3:46-60].

$$P_{\text{dens(r)}} = \frac{EIRP}{4\pi d_s^2} \quad (W) \tag{5}$$

Given a transmit frequency of 20 GHz, the transmit wavelength can be calculated as:

$$\lambda = \frac{C}{f} \quad (m) \tag{6}$$

Where:

f is the transmit frequency

C is the speed of light

The effective aperture of the receive antenna can then be calculated using Equation 7 (4.10, Ryan) [3:63-76].

$$A_e = \frac{\lambda^2}{4\pi} G_r \quad (m^2) \tag{7}$$

Where:

G_r is the gain of the receive antenna

The total receive power can therefore be represented by Equation 8 (4.13, Ryan) [3:79-121].

$$P_r = P_{\text{dens(r)}} A_e G_r \quad = P_t G_t \frac{\lambda}{4\pi}^2 G_r \quad (W) \tag{8}$$

Where:

G_r is the gain of the receive antenna

The receive power is then converted to decibel-watts (dBW) using Equation 9

$$P_{\text{r(dbW)}} = 10 \log_{10} P_{\text{r(W)}} \tag{9}$$

The receive power model for each ground station is shown in Figures 9, 10, and 11. The receive power at the optimal time step is shown in Table 5

Ground Station	Receive Power (dBW)
Cocos Island	-154.58
Mawson Research Station	-155.08
Norfolk Island	-155.12

Table 5: Optimal receive power with respect to COMMSAT1 over the course of a sidereal day

This receive power is reflective of the large free path loss that is characteristic of satellites at similar orbital heights. To mitigate this effect, geosynchronous satellites generally require large antenna arrays with enough signal power to provide sufficient margins when accounting for path loss and atmospheric attenuation. Geo-synchronous satellites tend to be larger and can therefore support such antennas. A high gain for the ground station's receive antenna is preferred for better performance and directivity.

4 List of Figures

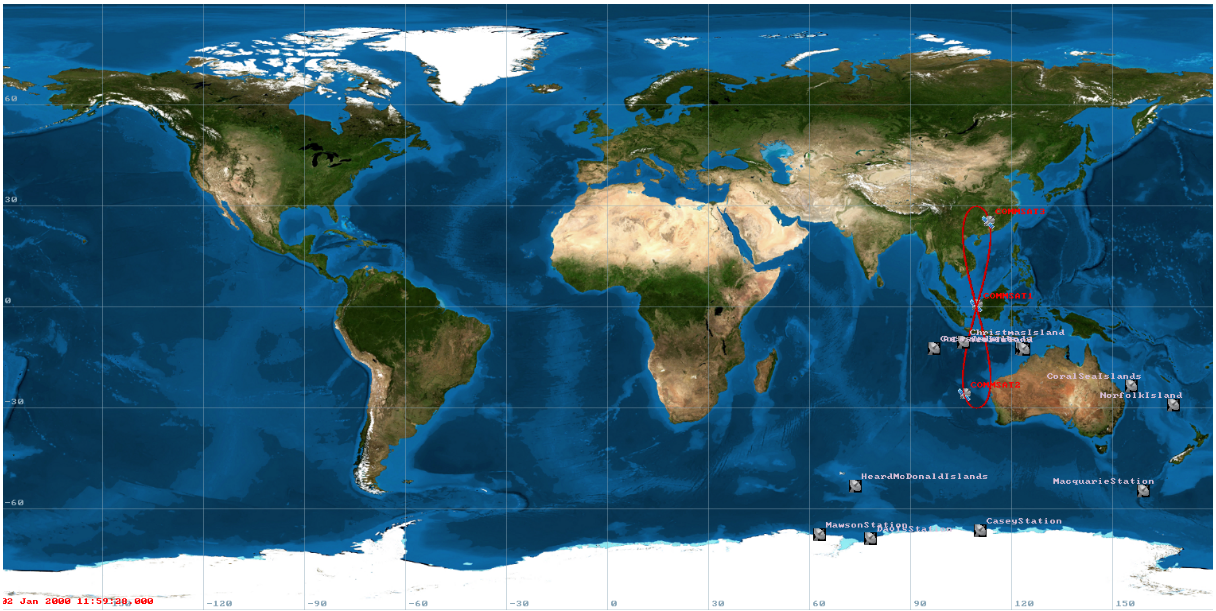

Figure 1: Ground track of constellation over a sidereal day

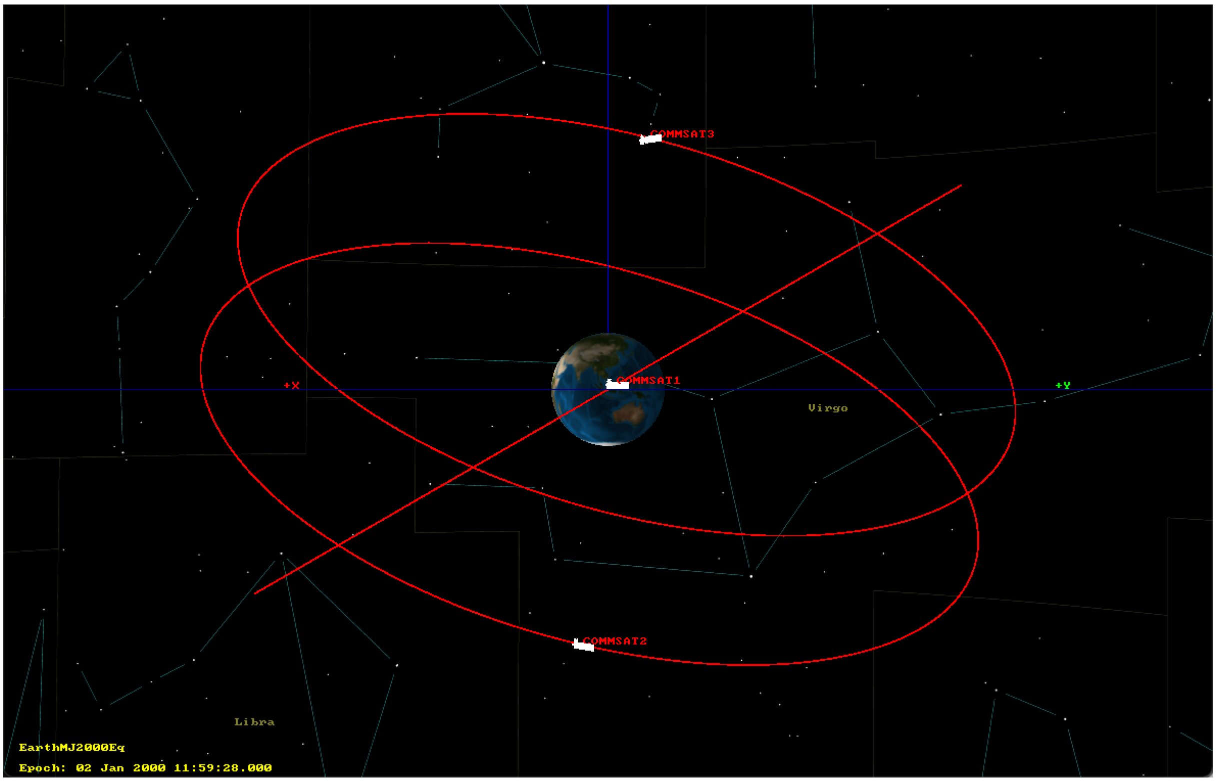

Figure 2: Constellation orbits over a sidereal day in the J2000 reference frame, showing the Earth's XY plane

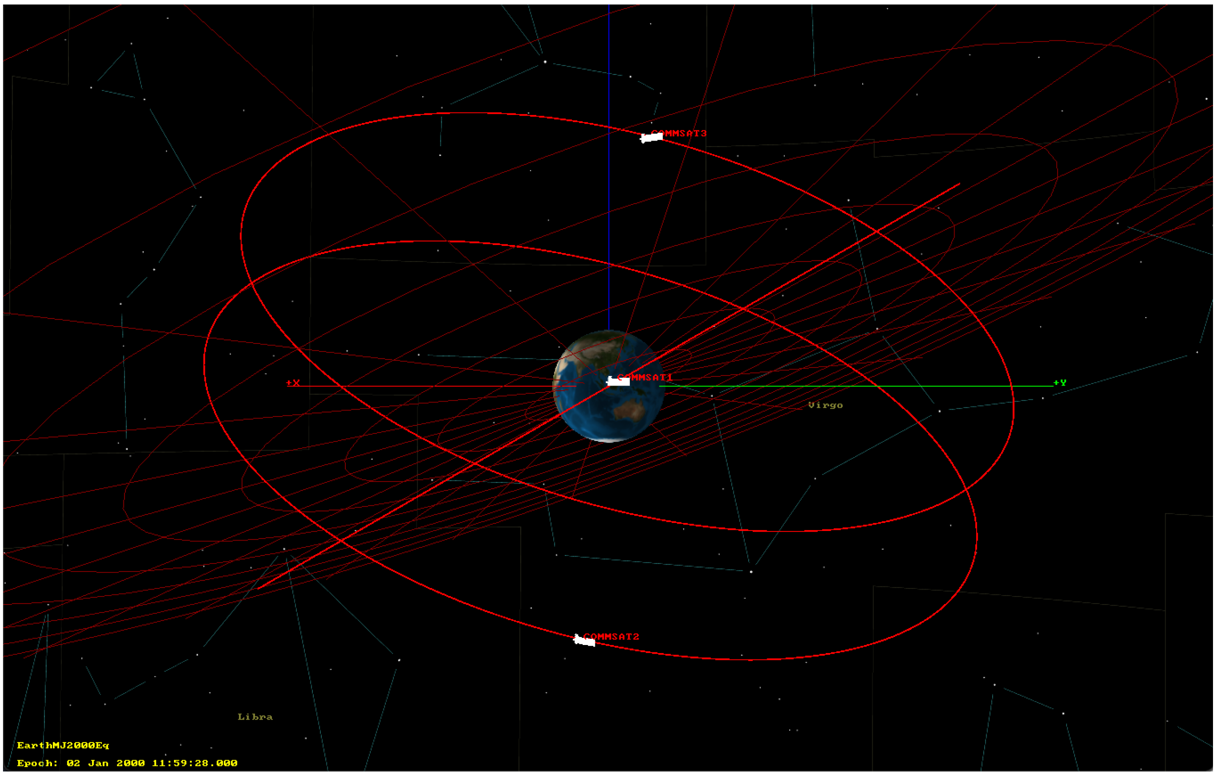

Figure 3: Constellation orbits over a sidereal day in the J2000 reference frame, showing the Earth's ecliptic plane

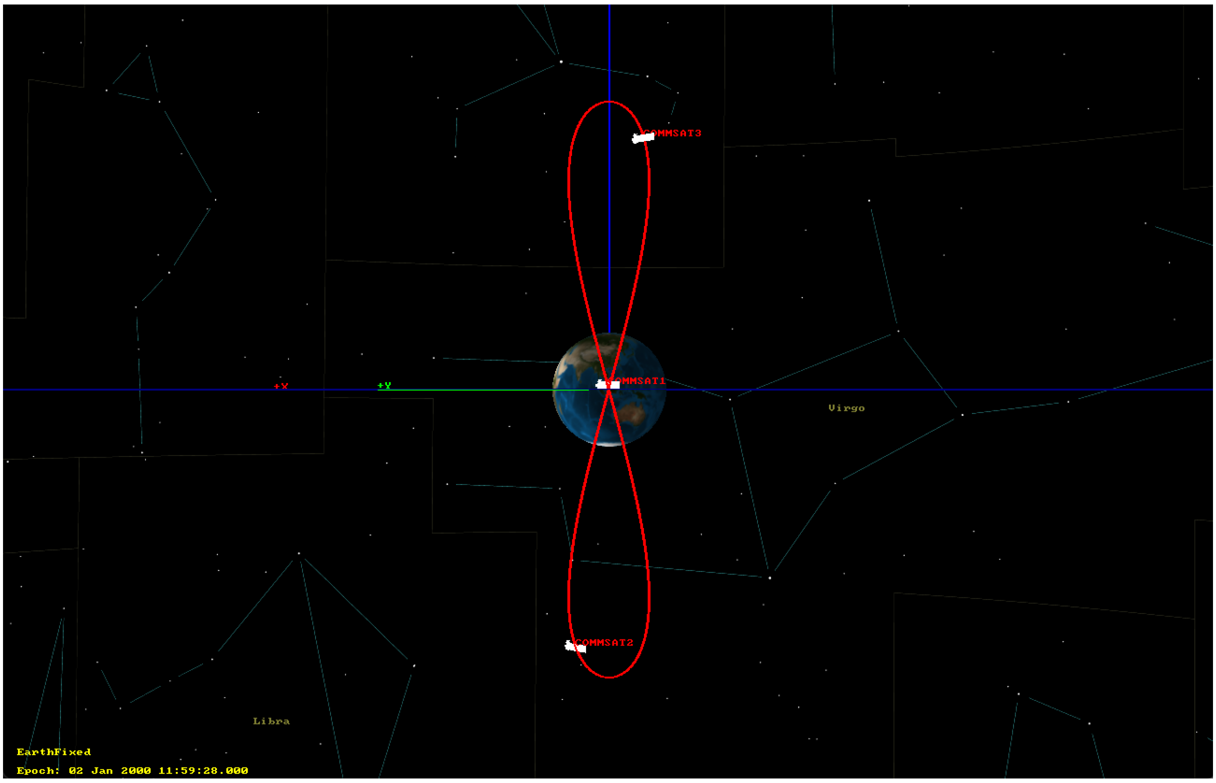

Figure 4: Constellation orbits over a sidereal day in the Earth-Fixed reference frame, showing the Earth's XY plane

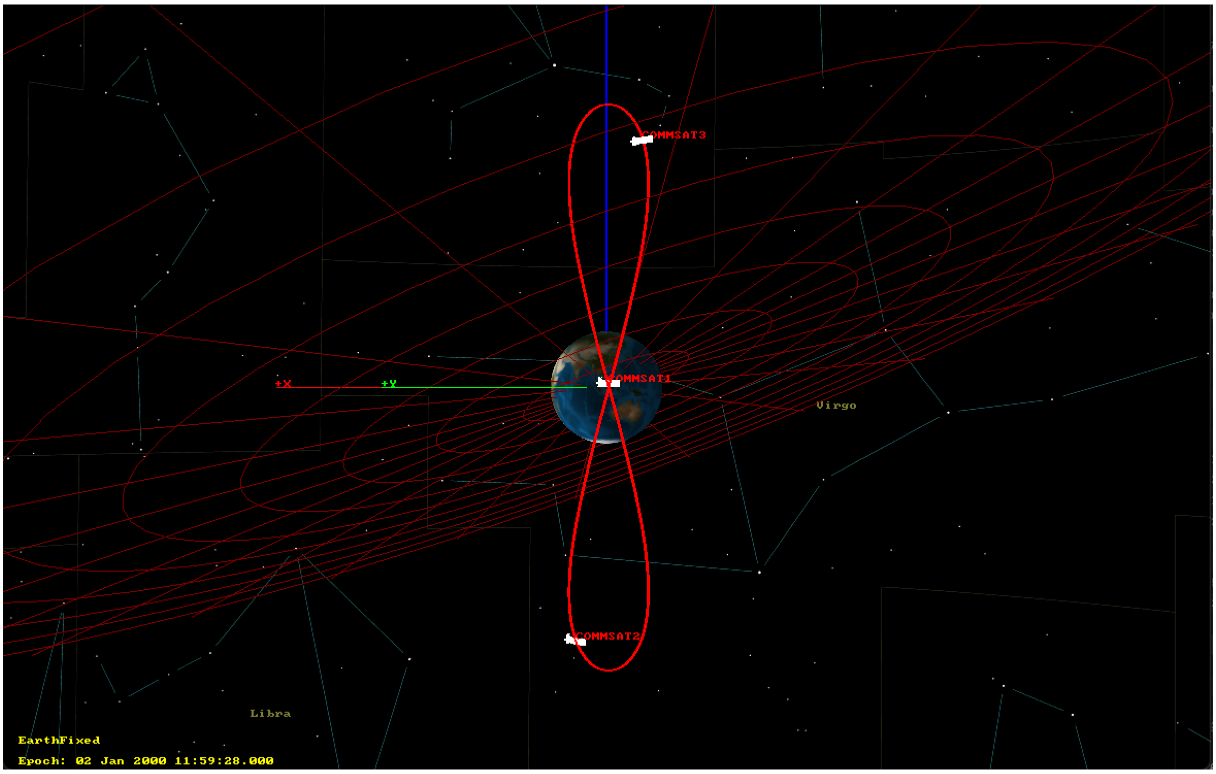

Figure 5: Constellation orbits over a sidereal day in the Earth-Fixed reference frame, showing the Earth's ecliptic plane

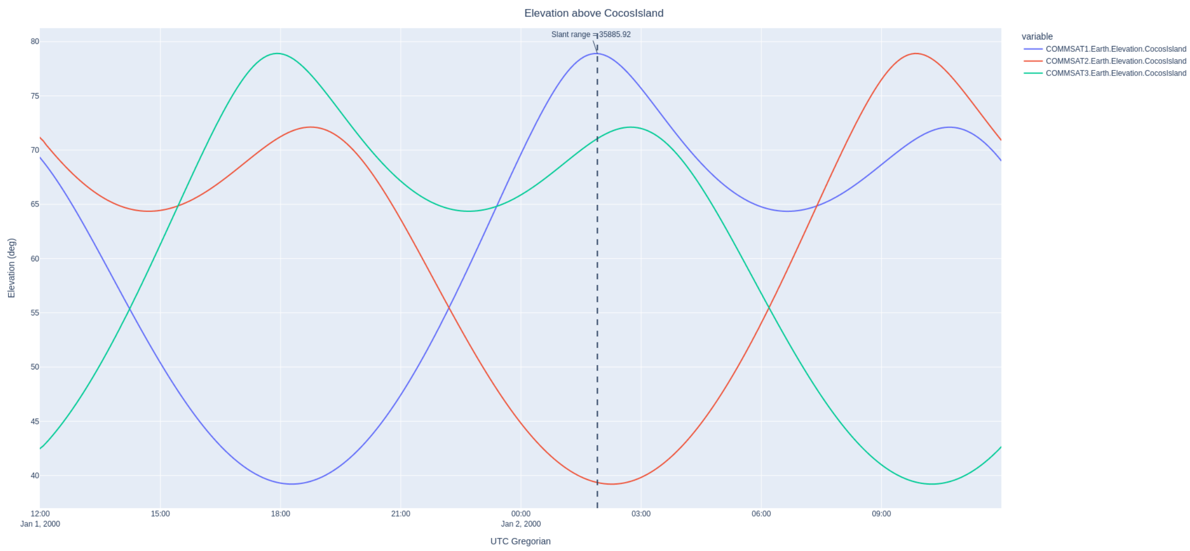

Figure 6: Elevation of constellation over with reference to Cocos (Keeling) Island over a sidereal day

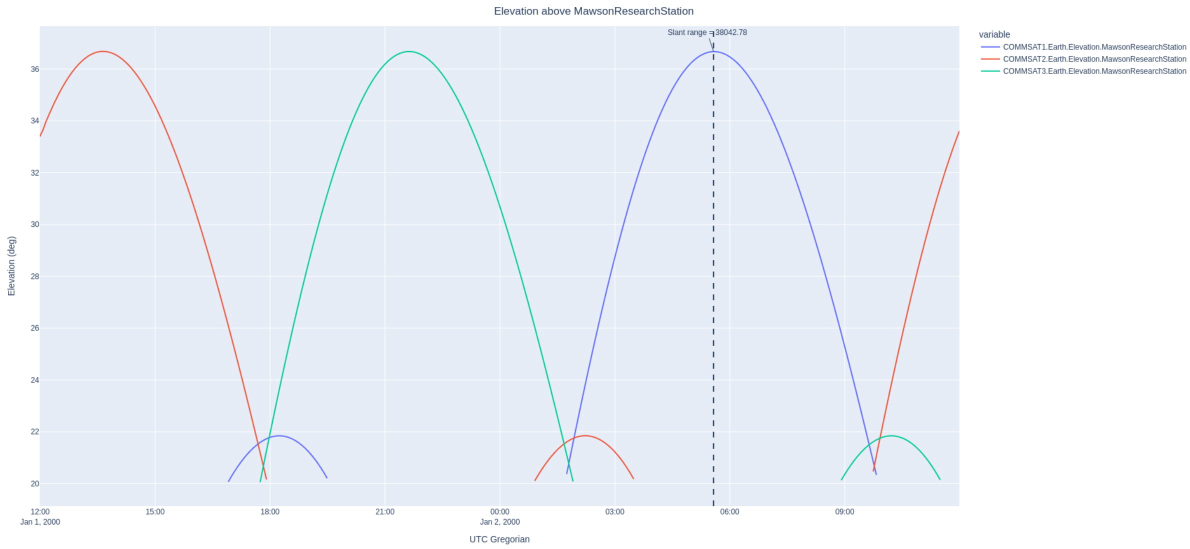

Figure 7: Elevation of constellation with reference to Mawson Research Station over a sidereal day

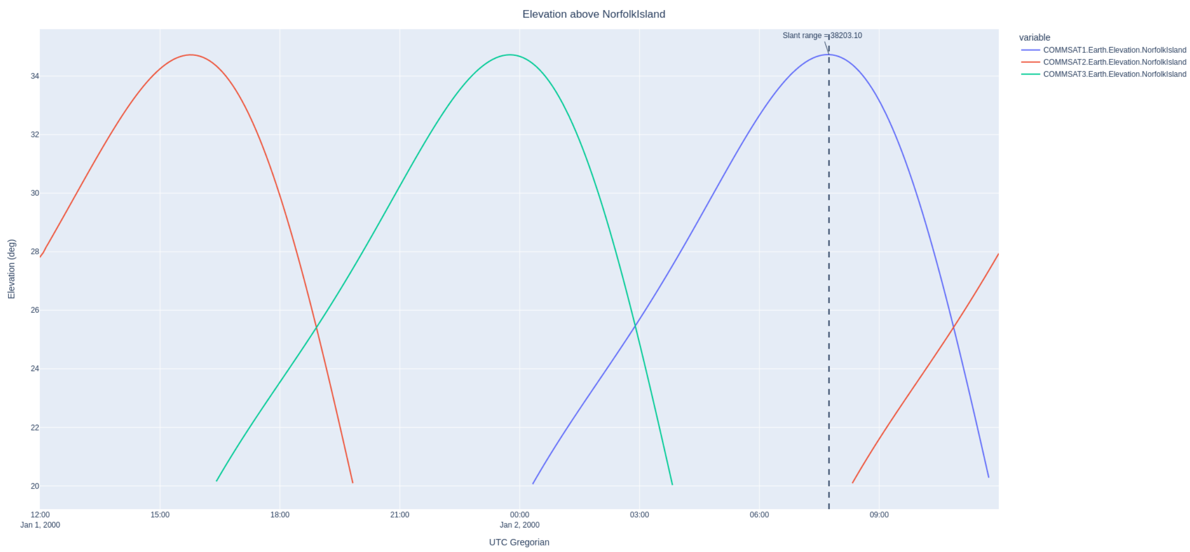

Figure 8: Elevation of constellation over with reference to Norfolk Island over a sidereal day

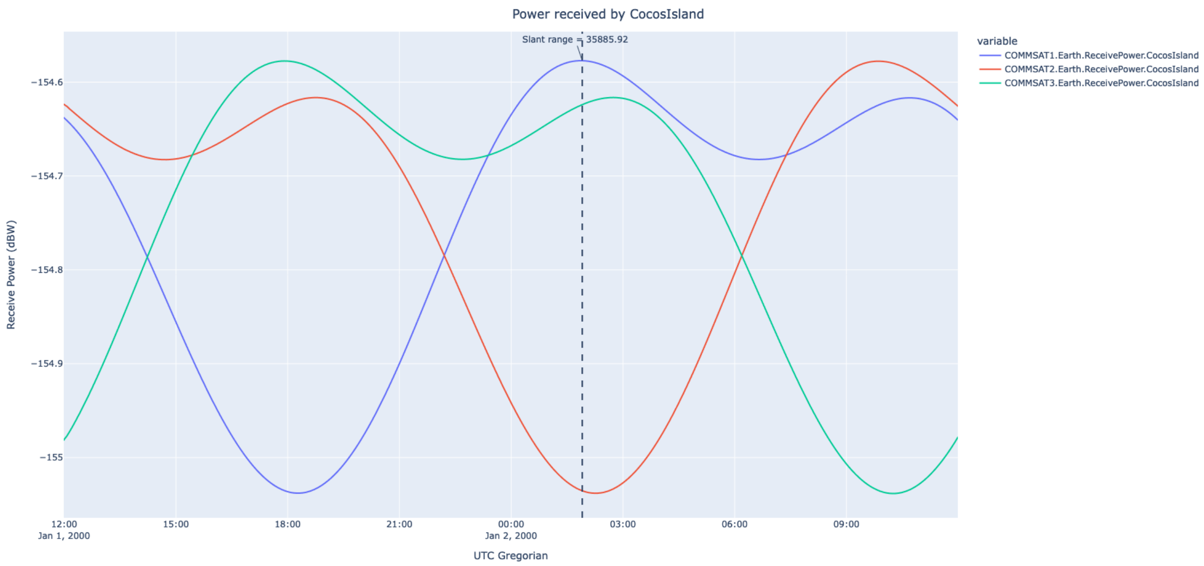

Figure 9: Power received at Cocos Island from constellation over a sidereal day

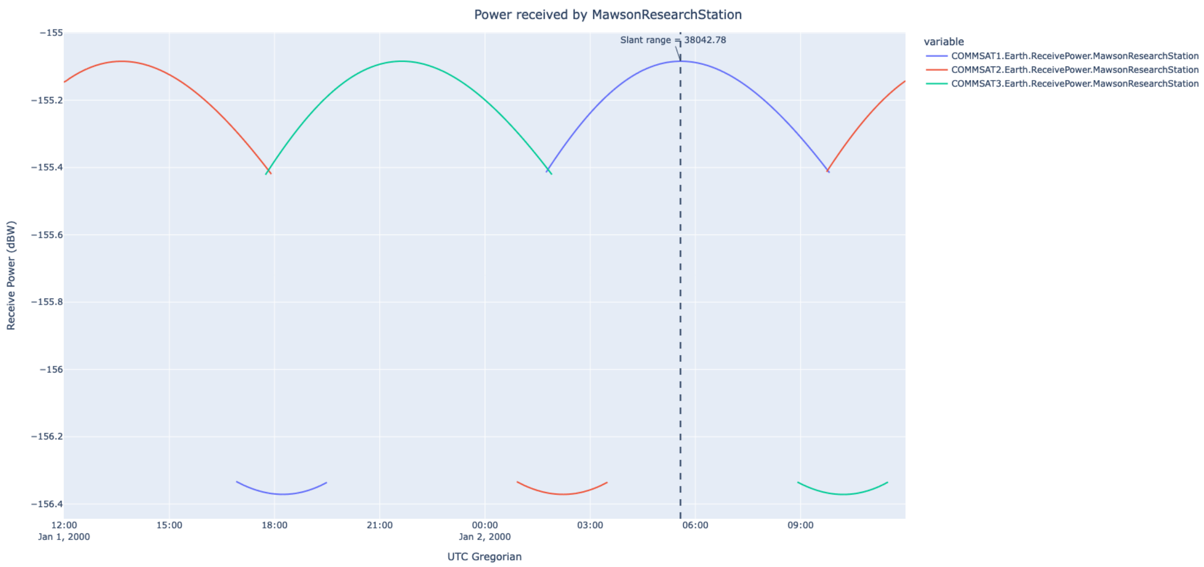

Figure 10: Power received at Mawson Research Station from constellation over a sidereal day

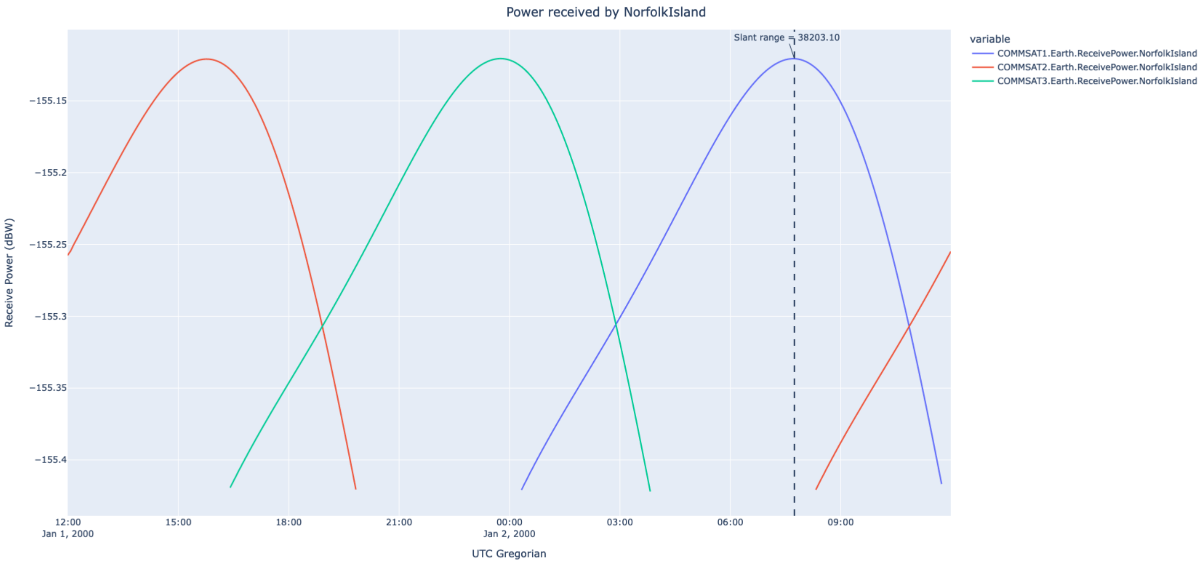

Figure 11: Power received at Norfolk Island from constellation over a sidereal day

5 Appendix

5.1 GMAT Script

```
%General Mission Analysis Tool(GMAT) Script
%Created: 2022-04-07 12:29:17


%----------------------------------------
%---------- Spacecraft
%----------------------------------------

Create Spacecraft COMMSAT1;
GMAT COMMSAT1.DateFormat = TAIModJulian;
GMAT COMMSAT1.Epoch = '21545';
GMAT COMMSAT1.CoordinateSystem = EarthMJ2000Eq;
GMAT COMMSAT1.DisplayStateType = Keplerian;
GMAT COMMSAT1.SMA = 42164.99999999999;
GMAT COMMSAT1.ECC = 3.794636528864267e-16;
GMAT COMMSAT1.INC = 29.99999999999998;
GMAT COMMSAT1.RAAN = 30;
GMAT COMMSAT1.AOP = 0;
GMAT COMMSAT1.TA = 1.207418269725733e-06;
GMAT COMMSAT1.DryMass = 850;
GMAT COMMSAT1.Cd = 2.2;
GMAT COMMSAT1.Cr = 1.8;
GMAT COMMSAT1.DragArea = 15;
GMAT COMMSAT1.SRPArea = 1;
GMAT COMMSAT1.SPADDragScaleFactor = 1;
GMAT COMMSAT1.SPADSRPScaleFactor = 1;
GMAT COMMSAT1.NAIFId = -10001001;
GMAT COMMSAT1.NAIFIdReferenceFrame = -9001001;
GMAT COMMSAT1.OrbitColor = Red;
GMAT COMMSAT1.TargetColor = Teal;
GMAT COMMSAT1.OrbitErrorCovariance = [ 1e+70 0 0 0 0 0 ; 0 1e+70 0 0 0 0 ; 0 0 1e+70 0 0 0 ; 0 0 0 1e+70 0 0 ; 0 0 0 0 1e+70 0 ; 0 0 0 0 0 1e+70 ];
GMAT COMMSAT1.CdSigma = 1e+70;
GMAT COMMSAT1.CrSigma = 1e+70;
GMAT COMMSAT1.Id = 'SatId';
GMAT COMMSAT1.Attitude = CoordinateSystemFixed;
GMAT COMMSAT1.SPADSRPInterpolationMethod = Bilinear;
GMAT COMMSAT1.SPADSRPScaleFactorSigma = 1e+70;
GMAT COMMSAT1.SPADDragInterpolationMethod = Bilinear;
GMAT COMMSAT1.SPADDragScaleFactorSigma = 1e+70;
GMAT COMMSAT1.ModelFile = 'aura.3ds';
GMAT COMMSAT1.ModelOffsetX = 0;
GMAT COMMSAT1.ModelOffsetY = 0;
GMAT COMMSAT1.ModelOffsetZ = 0;
GMAT COMMSAT1.ModelRotationX = 0;
GMAT COMMSAT1.ModelRotationY = 0;
GMAT COMMSAT1.ModelRotationZ = 0;
```

```
GMAT COMMSAT1.ModelScale = 1;
GMAT COMMSAT1.AttitudeDisplayStateType = 'Quaternion';
GMAT COMMSAT1.AttitudeRateDisplayStateType = 'AngularVelocity';
GMAT COMMSAT1.AttitudeCoordinateSystem = EarthMJ2000Eq;
GMAT COMMSAT1.EulerAngleSequence = '321';

%----------------------------------------
%---------- Spacecraft
%----------------------------------------

Create Spacecraft COMMSAT2;
GMAT COMMSAT2.DateFormat = TAIModJulian;
GMAT COMMSAT2.Epoch = '21545';
GMAT COMMSAT2.CoordinateSystem = EarthMJ2000Eq;
GMAT COMMSAT2.DisplayStateType = Keplerian;
GMAT COMMSAT2.SMA = 42164.99999999999;
GMAT COMMSAT2.ECC = 1.192684599010012e-15;
GMAT COMMSAT2.INC = 30;
GMAT COMMSAT2.RAAN = 150;
GMAT COMMSAT2.AOP = 0;
GMAT COMMSAT2.TA = 239.9999999999998;
GMAT COMMSAT2.DryMass = 850;
GMAT COMMSAT2.Cd = 2.2;
GMAT COMMSAT2.Cr = 1.8;
GMAT COMMSAT2.DragArea = 15;
GMAT COMMSAT2.SRPArea = 1;
GMAT COMMSAT2.SPADDragScaleFactor = 1;
GMAT COMMSAT2.SPADSRPScaleFactor = 1;
GMAT COMMSAT2.NAIFId = -10001001;
GMAT COMMSAT2.NAIFIdReferenceFrame = -9001001;
GMAT COMMSAT2.OrbitColor = Red;
GMAT COMMSAT2.TargetColor = Teal;
GMAT COMMSAT2.OrbitErrorCovariance = [ 1e+70 0 0 0 0 0 ; 0 1e+70 0 0 0 0 ; 0 0 1e+70 0 0 0 ; 0 0 0 1e+70 0 0 ; 0 0 0 0 1e+70 0 ; 0 0 0 0 0 1e+70 ];
GMAT COMMSAT2.CdSigma = 1e+70;
GMAT COMMSAT2.CrSigma = 1e+70;
GMAT COMMSAT2.Id = 'SatId';
GMAT COMMSAT2.Attitude = CoordinateSystemFixed;
GMAT COMMSAT2.SPADSRPInterpolationMethod = Bilinear;
GMAT COMMSAT2.SPADSRPScaleFactorSigma = 1e+70;
GMAT COMMSAT2.SPADDragInterpolationMethod = Bilinear;
GMAT COMMSAT2.SPADDragScaleFactorSigma = 1e+70;
GMAT COMMSAT2.ModelFile = 'aura.3ds';
GMAT COMMSAT2.ModelOffsetX = 0;
GMAT COMMSAT2.ModelOffsetY = 0;
GMAT COMMSAT2.ModelOffsetZ = 0;
GMAT COMMSAT2.ModelRotationX = 0;
GMAT COMMSAT2.ModelRotationY = 0;
GMAT COMMSAT2.ModelRotationZ = 0;
GMAT COMMSAT2.ModelScale = 1;
GMAT COMMSAT2.AttitudeDisplayStateType = 'Quaternion';
GMAT COMMSAT2.AttitudeRateDisplayStateType = 'AngularVelocity';
GMAT COMMSAT2.AttitudeCoordinateSystem = EarthMJ2000Eq;
GMAT COMMSAT2.EulerAngleSequence = '321';
```

```

%----------------------------------------
%---------- Spacecraft
%----------------------------------------

Create Spacecraft COMMSAT3;
GMAT COMMSAT3.DateFormat = TAIModJulian;
GMAT COMMSAT3.Epoch = '21545';
GMAT COMMSAT3.CoordinateSystem = EarthMJ2000Eq;
GMAT COMMSAT3.DisplayStateType = Keplerian;
GMAT COMMSAT3.SMA = 42164.99999999999;
GMAT COMMSAT3.ECC = 1.293181873398937e-15;
GMAT COMMSAT3.INC = 29.99999999999997;
GMAT COMMSAT3.RAAN = 270;
GMAT COMMSAT3.AOP = 0;
GMAT COMMSAT3.TA = 120;
GMAT COMMSAT3.DryMass = 850;
GMAT COMMSAT3.Cd = 2.2;
GMAT COMMSAT3.Cr = 1.8;
GMAT COMMSAT3.DragArea = 15;
GMAT COMMSAT3.SRPArea = 1;
GMAT COMMSAT3.SPADDragScaleFactor = 1;
GMAT COMMSAT3.SPADSRPScaleFactor = 1;
GMAT COMMSAT3.NAIFId = -10001001;
GMAT COMMSAT3.NAIFIdReferenceFrame = -9001001;
GMAT COMMSAT3.OrbitColor = Red;
GMAT COMMSAT3.TargetColor = Teal;
GMAT COMMSAT3.OrbitErrorCovariance = [ 1e+70 0 0 0 0 0 ; 0 1e+70 0 0 0 0 ; 0 0 1e+70 0 0 0 ; 0 0 0 1e+70 0 0 ; 0 0 0 0 1e+70 0 ; 0 0 0 0 0 1e+70 ];
GMAT COMMSAT3.CdSigma = 1e+70;
GMAT COMMSAT3.CrSigma = 1e+70;
GMAT COMMSAT3.Id = 'SatId';
GMAT COMMSAT3.Attitude = CoordinateSystemFixed;
GMAT COMMSAT3.SPADSRPInterpolationMethod = Bilinear;
GMAT COMMSAT3.SPADSRPScaleFactorSigma = 1e+70;
GMAT COMMSAT3.SPADDragInterpolationMethod = Bilinear;
GMAT COMMSAT3.SPADDragScaleFactorSigma = 1e+70;
GMAT COMMSAT3.ModelFile = 'aura.3ds';
GMAT COMMSAT3.ModelOffsetX = 0;
GMAT COMMSAT3.ModelOffsetY = 0;
GMAT COMMSAT3.ModelOffsetZ = 0;
GMAT COMMSAT3.ModelRotationX = 0;
GMAT COMMSAT3.ModelRotationY = 0;
GMAT COMMSAT3.ModelRotationZ = 0;
GMAT COMMSAT3.ModelScale = 1;
GMAT COMMSAT3.AttitudeDisplayStateType = 'Quaternion';
GMAT COMMSAT3.AttitudeRateDisplayStateType = 'AngularVelocity';
GMAT COMMSAT3.AttitudeCoordinateSystem = EarthMJ2000Eq;
GMAT COMMSAT3.EulerAngleSequence = '321';

%----------------------------------------
%---------- GroundStations
%----------------------------------------

```

```
Create GroundStation AshmoreIslands;
GMAT AshmoreIslands.OrbitColor = Thistle;
GMAT AshmoreIslands.TargetColor = [252 102 101];
GMAT AshmoreIslands.CentralBody = Earth;
GMAT AshmoreIslands.StateType = Spherical;
GMAT AshmoreIslands.HorizonReference = Sphere;
GMAT AshmoreIslands.Location1 = -12.183;
GMAT AshmoreIslands.Location2 = 122.983;
GMAT AshmoreIslands.Location3 = 0;
GMAT AshmoreIslands.Id = 'Ashmore';
GMAT AshmoreIslands.IonosphereModel = 'None';
GMAT AshmoreIslands.TroposphereModel = 'None';
GMAT AshmoreIslands.DataSource = 'Constant';
GMAT AshmoreIslands.Temperature = 295.1;
GMAT AshmoreIslands.Pressure = 1013.5;
GMAT AshmoreIslands.Humidity = 55;
GMAT AshmoreIslands.MinimumElevationAngle = 20;

Create GroundStation CartierIsland;
GMAT CartierIsland.OrbitColor = Thistle;
GMAT CartierIsland.TargetColor = [253 204 101];
GMAT CartierIsland.CentralBody = Earth;
GMAT CartierIsland.StateType = Spherical;
GMAT CartierIsland.HorizonReference = Sphere;
GMAT CartierIsland.Location1 = -12.51;
GMAT CartierIsland.Location2 = 123.55;
GMAT CartierIsland.Location3 = 0;
GMAT CartierIsland.Id = 'Cartier';
GMAT CartierIsland.IonosphereModel = 'None';
GMAT CartierIsland.TroposphereModel = 'None';
GMAT CartierIsland.DataSource = 'Constant';
GMAT CartierIsland.Temperature = 295.1;
GMAT CartierIsland.Pressure = 1013.5;
GMAT CartierIsland.Humidity = 55;
GMAT CartierIsland.MinimumElevationAngle = 20;

Create GroundStation CaseyStation;
GMAT CaseyStation.OrbitColor = Thistle;
GMAT CaseyStation.TargetColor = [254 255 102];
GMAT CaseyStation.CentralBody = Earth;
GMAT CaseyStation.StateType = Spherical;
GMAT CaseyStation.HorizonReference = Sphere;
GMAT CaseyStation.Location1 = -66.28;
GMAT CaseyStation.Location2 = 110.53;
GMAT CaseyStation.Location3 = 0;
GMAT CaseyStation.Id = 'StationId';
GMAT CaseyStation.IonosphereModel = 'None';
GMAT CaseyStation.TroposphereModel = 'None';
GMAT CaseyStation.DataSource = 'Constant';
GMAT CaseyStation.Temperature = 295.1;
GMAT CaseyStation.Pressure = 1013.5;
GMAT CaseyStation.Humidity = 55;
GMAT CaseyStation.MinimumElevationAngle = 20;

Create GroundStation DavisStation;
```

```
GMAT DavisStation.OrbitColor = Thistle;
GMAT DavisStation.TargetColor = [203 255 102];
GMAT DavisStation.CentralBody = Earth;
GMAT DavisStation.StateType = Spherical;
GMAT DavisStation.HorizonReference = Sphere;
GMAT DavisStation.Location1 = -68.583;
GMAT DavisStation.Location2 = 77.967;
GMAT DavisStation.Location3 = 0;
GMAT DavisStation.Id = 'StationId';
GMAT DavisStation.IonosphereModel = 'None';
GMAT DavisStation.TroposphereModel = 'None';
GMAT DavisStation.DataSource = 'Constant';
GMAT DavisStation.Temperature = 295.1;
GMAT DavisStation.Pressure = 1013.5;
GMAT DavisStation.Humidity = 55;
GMAT DavisStation.MinimumElevationAngle = 20;

Create GroundStation MawsonStation;
GMAT MawsonStation.OrbitColor = Thistle;
GMAT MawsonStation.TargetColor = [101 255 102];
GMAT MawsonStation.CentralBody = Earth;
GMAT MawsonStation.StateType = Spherical;
GMAT MawsonStation.HorizonReference = Sphere;
GMAT MawsonStation.Location1 = -67.59999999999999;
GMAT MawsonStation.Location2 = 62.867;
GMAT MawsonStation.Location3 = 0;
GMAT MawsonStation.Id = 'StationId';
GMAT MawsonStation.IonosphereModel = 'None';
GMAT MawsonStation.TroposphereModel = 'None';
GMAT MawsonStation.DataSource = 'Constant';
GMAT MawsonStation.Temperature = 295.1;
GMAT MawsonStation.Pressure = 1013.5;
GMAT MawsonStation.Humidity = 55;
GMAT MawsonStation.MinimumElevationAngle = 20;

Create GroundStation MacquarieStation;
GMAT MacquarieStation.OrbitColor = Thistle;
GMAT MacquarieStation.TargetColor = [101 255 204];
GMAT MacquarieStation.CentralBody = Earth;
GMAT MacquarieStation.StateType = Spherical;
GMAT MacquarieStation.HorizonReference = Sphere;
GMAT MacquarieStation.Location1 = -54.5;
GMAT MacquarieStation.Location2 = 158.95;
GMAT MacquarieStation.Location3 = 0;
GMAT MacquarieStation.Id = 'StationId';
GMAT MacquarieStation.IonosphereModel = 'None';
GMAT MacquarieStation.TroposphereModel = 'None';
GMAT MacquarieStation.DataSource = 'Constant';
GMAT MacquarieStation.Temperature = 295.1;
GMAT MacquarieStation.Pressure = 1013.5;
GMAT MacquarieStation.Humidity = 55;
GMAT MacquarieStation.MinimumElevationAngle = 20;

Create GroundStation ChristmasIsland;
GMAT ChristmasIsland.OrbitColor = Thistle;
```

```
GMAT ChristmasIsland.TargetColor = [101 255 254];
GMAT ChristmasIsland.CentralBody = Earth;
GMAT ChristmasIsland.StateType = Spherical;
GMAT ChristmasIsland.HorizonReference = Sphere;
GMAT ChristmasIsland.Location1 = -10.41;
GMAT ChristmasIsland.Location2 = 105.716;
GMAT ChristmasIsland.Location3 = 0;
GMAT ChristmasIsland.Id = 'ChristmasIsland';
GMAT ChristmasIsland.IonosphereModel = 'None';
GMAT ChristmasIsland.TroposphereModel = 'None';
GMAT ChristmasIsland.DataSource = 'Constant';
GMAT ChristmasIsland.Temperature = 295.1;
GMAT ChristmasIsland.Pressure = 1013.5;
GMAT ChristmasIsland.Humidity = 55;
GMAT ChristmasIsland.MinimumElevationAngle = 20;

Create GroundStation CocosIslands;
GMAT CocosIslands.OrbitColor = Thistle;
GMAT CocosIslands.TargetColor = [101 204 254];
GMAT CocosIslands.CentralBody = Earth;
GMAT CocosIslands.StateType = Spherical;
GMAT CocosIslands.HorizonReference = Sphere;
GMAT CocosIslands.Location1 = -12.167;
GMAT CocosIslands.Location2 = 96.833;
GMAT CocosIslands.Location3 = 0;
GMAT CocosIslands.Id = 'StationId';
GMAT CocosIslands.IonosphereModel = 'None';
GMAT CocosIslands.TroposphereModel = 'None';
GMAT CocosIslands.DataSource = 'Constant';
GMAT CocosIslands.Temperature = 295.1;
GMAT CocosIslands.Pressure = 1013.5;
GMAT CocosIslands.Humidity = 55;
GMAT CocosIslands.MinimumElevationAngle = 20;

Create GroundStation CoralSeaIslands;
GMAT CoralSeaIslands.OrbitColor = Thistle;
GMAT CoralSeaIslands.TargetColor = [102 102 254];
GMAT CoralSeaIslands.CentralBody = Earth;
GMAT CoralSeaIslands.StateType = Spherical;
GMAT CoralSeaIslands.HorizonReference = Sphere;
GMAT CoralSeaIslands.Location1 = -23.25;
GMAT CoralSeaIslands.Location2 = 155.533;
GMAT CoralSeaIslands.Location3 = 0;
GMAT CoralSeaIslands.Id = 'StationId';
GMAT CoralSeaIslands.IonosphereModel = 'None';
GMAT CoralSeaIslands.TroposphereModel = 'None';
GMAT CoralSeaIslands.DataSource = 'Constant';
GMAT CoralSeaIslands.Temperature = 295.1;
GMAT CoralSeaIslands.Pressure = 1013.5;
GMAT CoralSeaIslands.Humidity = 55;
GMAT CoralSeaIslands.MinimumElevationAngle = 20;

Create GroundStation HeardMcDonaldIslands;
GMAT HeardMcDonaldIslands.OrbitColor = Thistle;
GMAT HeardMcDonaldIslands.TargetColor = [204 102 254];
```

```
GMAT HeardMcDonaldIslands.CentralBody = Earth;
GMAT HeardMcDonaldIslands.StateType = Spherical;
GMAT HeardMcDonaldIslands.HorizonReference = Sphere;
GMAT HeardMcDonaldIslands.Location1 = -53.08333;
GMAT HeardMcDonaldIslands.Location2 = 73.5;
GMAT HeardMcDonaldIslands.Location3 = 0;
GMAT HeardMcDonaldIslands.Id = 'StationId';
GMAT HeardMcDonaldIslands.IonosphereModel = 'None';
GMAT HeardMcDonaldIslands.TroposphereModel = 'None';
GMAT HeardMcDonaldIslands.DataSource = 'Constant';
GMAT HeardMcDonaldIslands.Temperature = 295.1;
GMAT HeardMcDonaldIslands.Pressure = 1013.5;
GMAT HeardMcDonaldIslands.Humidity = 55;
GMAT HeardMcDonaldIslands.MinimumElevationAngle = 20;

Create GroundStation NorfolkIsland;
GMAT NorfolkIsland.OrbitColor = Thistle;
GMAT NorfolkIsland.TargetColor = [252 102 254];
GMAT NorfolkIsland.CentralBody = Earth;
GMAT NorfolkIsland.StateType = Spherical;
GMAT NorfolkIsland.HorizonReference = Sphere;
GMAT NorfolkIsland.Location1 = -29.033;
GMAT NorfolkIsland.Location2 = 167.95;
GMAT NorfolkIsland.Location3 = 0;
GMAT NorfolkIsland.Id = 'StationId';
GMAT NorfolkIsland.IonosphereModel = 'None';
GMAT NorfolkIsland.TroposphereModel = 'None';
GMAT NorfolkIsland.DataSource = 'Constant';
GMAT NorfolkIsland.Temperature = 295.1;
GMAT NorfolkIsland.Pressure = 1013.5;
GMAT NorfolkIsland.Humidity = 55;
GMAT NorfolkIsland.MinimumElevationAngle = 20;













%----------------------------------------
%---------- ForceModels
%----------------------------------------

Create ForceModel DefaultProp_ForceModel;
GMAT DefaultProp_ForceModel.CentralBody = Earth;
GMAT DefaultProp_ForceModel.PrimaryBodies = {Earth};
GMAT DefaultProp_ForceModel.Drag = None;
GMAT DefaultProp_ForceModel.SRP = Off;
GMAT DefaultProp_ForceModel.RelativisticCorrection = Off;
```

```
GMAT DefaultProp_ForceModel.ErrorControl = RSSStep;
GMAT DefaultProp_ForceModel.GravityField.Earth.Degree = 4;
GMAT DefaultProp_ForceModel.GravityField.Earth.Order = 4;
GMAT DefaultProp_ForceModel.GravityField.Earth.StmLimit = 100;
GMAT DefaultProp_ForceModel.GravityField.Earth.PotentialFile = 'JGM2.cof';
GMAT DefaultProp_ForceModel.GravityField.Earth.TideModel = 'None';

%----------------------------------------
%---------- Propagators
%----------------------------------------

Create Propagator DefaultProp;
GMAT DefaultProp.FM = DefaultProp_ForceModel;
GMAT DefaultProp.Type = RungeKutta89;
GMAT DefaultProp.InitialStepSize = 60;
GMAT DefaultProp.Accuracy = 9.999999999999999e-12;
GMAT DefaultProp.MinStep = 0.001;
GMAT DefaultProp.MaxStep = 2700;
GMAT DefaultProp.MaxStepAttempts = 50;
GMAT DefaultProp.StopIfAccuracyIsViolated = true;

%----------------------------------------
%---------- EventLocators
%----------------------------------------

Create ContactLocator ContactLocator1;
GMAT ContactLocator1.Target = COMMSAT1;
GMAT ContactLocator1.Filename = 'ContactLocator1.txt';
GMAT ContactLocator1.InputEpochFormat = 'TAIModJulian';
GMAT ContactLocator1.InitialEpoch = '21545';
GMAT ContactLocator1.StepSize = 10;
GMAT ContactLocator1.FinalEpoch = '21545.138';
GMAT ContactLocator1.UseLightTimeDelay = true;
GMAT ContactLocator1.UseStellarAberration = true;
GMAT ContactLocator1.WriteReport = true;
GMAT ContactLocator1.RunMode = Automatic;
GMAT ContactLocator1.UseEntireInterval = true;
GMAT ContactLocator1.Observers = {AshmoreIslands, CartierIsland, CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands, DavisStation, HeardMcDonaldIslands, MacquarieStation, MawsonStation, NorfolkIsland};
GMAT ContactLocator1.LightTimeDirection = Transmit;

Create ContactLocator ContactLocator2;
GMAT ContactLocator2.Target = COMMSAT2;
GMAT ContactLocator2.Filename = 'ContactLocator2.txt';
GMAT ContactLocator2.InputEpochFormat = 'TAIModJulian';
GMAT ContactLocator2.InitialEpoch = '21545';
GMAT ContactLocator2.StepSize = 10;
GMAT ContactLocator2.FinalEpoch = '21545.138';
GMAT ContactLocator2.UseLightTimeDelay = true;
GMAT ContactLocator2.UseStellarAberration = true;
GMAT ContactLocator2.WriteReport = true;
GMAT ContactLocator2.RunMode = Automatic;
```

```
GMAT ContactLocator2.UseEntireInterval = true;
GMAT ContactLocator2.Observers = {AshmoreIslands, CartierIsland, CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands, DavisStation, HeardMcDonaldIslands, MacquarieStation, MawsonStation, NorfolkIsland};
GMAT ContactLocator2.LightTimeDirection = Transmit;

Create ContactLocator ContactLocator3;
GMAT ContactLocator3.Target = COMMSAT3;
GMAT ContactLocator3.Filename = 'ContactLocator3.txt';
GMAT ContactLocator3.InputEpochFormat = 'TAIModJulian';
GMAT ContactLocator3.InitialEpoch = '21545';
GMAT ContactLocator3.StepSize = 10;
GMAT ContactLocator3.FinalEpoch = '21545.138';
GMAT ContactLocator3.UseLightTimeDelay = true;
GMAT ContactLocator3.UseStellarAberration = true;
GMAT ContactLocator3.WriteReport = true;
GMAT ContactLocator3.RunMode = Automatic;
GMAT ContactLocator3.UseEntireInterval = true;
GMAT ContactLocator3.Observers = {AshmoreIslands, CartierIsland, CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands, DavisStation, HeardMcDonaldIslands, MacquarieStation, MawsonStation, NorfolkIsland};
GMAT ContactLocator3.LightTimeDirection = Transmit;

Create EclipseLocator EclipseLocator1;
GMAT EclipseLocator1.Spacecraft = COMMSAT3;
GMAT EclipseLocator1.Filename = 'EclipseLocator1.txt';
GMAT EclipseLocator1.OccultingBodies = {Earth, Luna};
GMAT EclipseLocator1.InputEpochFormat = 'TAIModJulian';
GMAT EclipseLocator1.InitialEpoch = '21545';
GMAT EclipseLocator1.StepSize = 10;
GMAT EclipseLocator1.FinalEpoch = '21545.138';
GMAT EclipseLocator1.UseLightTimeDelay = true;
GMAT EclipseLocator1.UseStellarAberration = true;
GMAT EclipseLocator1.WriteReport = true;
GMAT EclipseLocator1.RunMode = Automatic;
GMAT EclipseLocator1.UseEntireInterval = true;
GMAT EclipseLocator1.EclipseTypes = {'Umbra', 'Penumbra', 'Antumbra'};

Create EclipseLocator EclipseLocator2;
GMAT EclipseLocator2.Spacecraft = COMMSAT2;
GMAT EclipseLocator2.Filename = 'EclipseLocator1.txt';
GMAT EclipseLocator2.OccultingBodies = {Earth, Luna};
GMAT EclipseLocator2.InputEpochFormat = 'TAIModJulian';
GMAT EclipseLocator2.InitialEpoch = '21545';
GMAT EclipseLocator2.StepSize = 10;
GMAT EclipseLocator2.FinalEpoch = '21545.138';
GMAT EclipseLocator2.UseLightTimeDelay = true;
GMAT EclipseLocator2.UseStellarAberration = true;
GMAT EclipseLocator2.WriteReport = true;
GMAT EclipseLocator2.RunMode = Automatic;
GMAT EclipseLocator2.UseEntireInterval = true;
GMAT EclipseLocator2.EclipseTypes = {'Umbra', 'Penumbra', 'Antumbra'};

```

```
Create EclipseLocator EclipseLocator3;
GMAT EclipseLocator3.Spacecraft = COMMSAT2;
GMAT EclipseLocator3.Filename = 'EclipseLocator1.txt';
GMAT EclipseLocator3.OccultingBodies = {Earth, Luna};
GMAT EclipseLocator3.InputEpochFormat = 'TAIModJulian';
GMAT EclipseLocator3.InitialEpoch = '21545';
GMAT EclipseLocator3.StepSize = 10;
GMAT EclipseLocator3.FinalEpoch = '21545.138';
GMAT EclipseLocator3.UseLightTimeDelay = true;
GMAT EclipseLocator3.UseStellarAberration = true;
GMAT EclipseLocator3.WriteReport = true;
GMAT EclipseLocator3.RunMode = Automatic;
GMAT EclipseLocator3.UseEntireInterval = true;
GMAT EclipseLocator3.EclipseTypes = {'Umbra', 'Penumbra', 'Antumbra'};

%----------------------------------------
%---------- Subscribers
%----------------------------------------

Create OrbitView EarthFixedEcliptic;
GMAT EarthFixedEcliptic.SolverIterations = None;
GMAT EarthFixedEcliptic.UpperLeft = [ 0.09761904761904762
    ↪ 0.08190476190476191 ];
GMAT EarthFixedEcliptic.Size = [ 0.6976190476190476 0.7342857142857143
    ↪ ];
GMAT EarthFixedEcliptic.RelativeZOrder = 388;
GMAT EarthFixedEcliptic.Maximized = false;
GMAT EarthFixedEcliptic.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
GMAT EarthFixedEcliptic.CoordinateSystem = EarthFixed;
GMAT EarthFixedEcliptic.DrawObject = [ true true true true ];
GMAT EarthFixedEcliptic.DataCollectFrequency = 1;
GMAT EarthFixedEcliptic.UpdatePlotFrequency = 50;
GMAT EarthFixedEcliptic.NumPointsToRedraw = 0;
GMAT EarthFixedEcliptic.ShowPlot = true;
GMAT EarthFixedEcliptic.MaxPlotPoints = 20000;
GMAT EarthFixedEcliptic.ShowLabels = true;
GMAT EarthFixedEcliptic.ViewPointReference = Earth;
GMAT EarthFixedEcliptic.ViewPointVector = COMMSAT1;
GMAT EarthFixedEcliptic.ViewDirection = Earth;
GMAT EarthFixedEcliptic.ViewScaleFactor = 2.5;
GMAT EarthFixedEcliptic.ViewUpCoordinateSystem = EarthMJ2000Eq;
GMAT EarthFixedEcliptic.ViewUpAxis = Z;
GMAT EarthFixedEcliptic.EclipticPlane = On;
GMAT EarthFixedEcliptic.XYPlane = Off;
GMAT EarthFixedEcliptic.WireFrame = Off;
GMAT EarthFixedEcliptic.Axes = On;
GMAT EarthFixedEcliptic.Grid = Off;
GMAT EarthFixedEcliptic.SunLine = Off;
GMAT EarthFixedEcliptic.UseInitialView = On;
GMAT EarthFixedEcliptic.StarCount = 7000;
GMAT EarthFixedEcliptic.EnableStars = On;
GMAT EarthFixedEcliptic.EnableConstellations = On;

Create GroundTrackPlot DefaultGroundTrackPlot;
GMAT DefaultGroundTrackPlot.SolverIterations = Current;
```

```
GMAT DefaultGroundTrackPlot.UpperLeft = [ 0.05773809523809524 0.03142857142857143 ];
GMAT DefaultGroundTrackPlot.Size = [ 0.9898809523809524 0.9533333333333334 ];
GMAT DefaultGroundTrackPlot.RelativeZOrder = 408;
GMAT DefaultGroundTrackPlot.Maximized = false;
GMAT DefaultGroundTrackPlot.Add = {AshmoreIslands, CartierIsland, CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands, DavisStation, COMMSAT1, COMMSAT2, COMMSAT3, HeardMcDonaldIslands, MacquarieStation, MawsonStation, NorfolkIsland};
GMAT DefaultGroundTrackPlot.DataCollectFrequency = 1;
GMAT DefaultGroundTrackPlot.UpdatePlotFrequency = 50;
GMAT DefaultGroundTrackPlot.NumPointsToRedraw = 0;
GMAT DefaultGroundTrackPlot.ShowPlot = true;
GMAT DefaultGroundTrackPlot.MaxPlotPoints = 20000;
GMAT DefaultGroundTrackPlot.CentralBody = Earth;
GMAT DefaultGroundTrackPlot.TextureMap = 'ModifiedBlueMarble.jpg';

Create ReportFile ReportFile1;
GMAT ReportFile1.SolverIterations = Current;
GMAT ReportFile1.UpperLeft = [ 0.1458333333333333 0.03656998738965952 ];
GMAT ReportFile1.Size = [ 0.9940476190476191 0.9609079445145019 ];
GMAT ReportFile1.RelativeZOrder = 121;
GMAT ReportFile1.Maximized = true;
GMAT ReportFile1.Filename = 'OrbitParams.txt';
GMAT ReportFile1.Precision = 16;
GMAT ReportFile1.Add = {COMMSAT1.UTCGregorian, COMMSAT1.Earth.TA, COMMSAT1.Earth.Latitude, COMMSAT1.Earth.Longitude, COMMSAT1.Earth.RMAG, COMMSAT1.Earth.Altitude, COMMSAT2.Earth.TA, COMMSAT2.Earth.Latitude, COMMSAT2.Earth.Longitude, COMMSAT2.Earth.RMAG, COMMSAT3.Earth.TA, COMMSAT3.Earth.Latitude, COMMSAT3.Earth.Longitude, COMMSAT3.Earth.RMAG};
GMAT ReportFile1.WriteHeaders = true;
GMAT ReportFile1.LeftJustify = On;
GMAT ReportFile1.ZeroFill = Off;
GMAT ReportFile1.FixedWidth = true;
GMAT ReportFile1.Delimiter = ' ';
GMAT ReportFile1.ColumnWidth = 23;
GMAT ReportFile1.WriteReport = true;

Create OrbitView J2000Ecliptic;
GMAT J2000Ecliptic.SolverIterations = None;
GMAT J2000Ecliptic.UpperLeft = [ 0.1035714285714286 0.04571428571428571 ];
GMAT J2000Ecliptic.Size = [ 0.8101190476190476 0.8504761904761905 ];
GMAT J2000Ecliptic.RelativeZOrder = 406;
GMAT J2000Ecliptic.Maximized = false;
GMAT J2000Ecliptic.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
GMAT J2000Ecliptic.CoordinateSystem = EarthMJ2000Eq;
GMAT J2000Ecliptic.DrawObject = [ true true true true ];
GMAT J2000Ecliptic.DataCollectFrequency = 1;
GMAT J2000Ecliptic.UpdatePlotFrequency = 50;
GMAT J2000Ecliptic.NumPointsToRedraw = 0;
GMAT J2000Ecliptic.ShowPlot = true;
GMAT J2000Ecliptic.MaxPlotPoints = 20000;
```

```
GMAT J2000Ecliptic.ShowLabels = true;
GMAT J2000Ecliptic.ViewPointReference = Earth;
GMAT J2000Ecliptic.ViewPointVector = COMMSAT1;
GMAT J2000Ecliptic.ViewDirection = Earth;
GMAT J2000Ecliptic.ViewScaleFactor = 2.5;
GMAT J2000Ecliptic.ViewUpCoordinateSystem = EarthMJ2000Eq;
GMAT J2000Ecliptic.ViewUpAxis = Z;
GMAT J2000Ecliptic.EclipticPlane = On;
GMAT J2000Ecliptic.XYPlane = Off;
GMAT J2000Ecliptic.WireFrame = Off;
GMAT J2000Ecliptic.Axes = On;
GMAT J2000Ecliptic.Grid = Off;
GMAT J2000Ecliptic.SunLine = Off;
GMAT J2000Ecliptic.UseInitialView = On;
GMAT J2000Ecliptic.StarCount = 7000;
GMAT J2000Ecliptic.EnableStars = On;
GMAT J2000Ecliptic.EnableConstellations = On;

%----------------------------------------
%---------- Subscribers
%----------------------------------------

Create OrbitView EarthFixedXY;
GMAT EarthFixedXY.SolverIterations = None;
GMAT EarthFixedXY.UpperLeft = [ 0.09345238095238095 0.03619047619047619
    ↪ ];
GMAT EarthFixedXY.Size = [ 0.7738095238095238 0.820952380952381 ];
GMAT EarthFixedXY.RelativeZOrder = 400;
GMAT EarthFixedXY.Maximized = false;
GMAT EarthFixedXY.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
GMAT EarthFixedXY.CoordinateSystem = EarthFixed;
GMAT EarthFixedXY.DrawObject = [ true true true true ];
GMAT EarthFixedXY.DataCollectFrequency = 1;
GMAT EarthFixedXY.UpdatePlotFrequency = 50;
GMAT EarthFixedXY.NumPointsToRedraw = 0;
GMAT EarthFixedXY.ShowPlot = true;
GMAT EarthFixedXY.MaxPlotPoints = 20000;
GMAT EarthFixedXY.ShowLabels = true;
GMAT EarthFixedXY.ViewPointReference = Earth;
GMAT EarthFixedXY.ViewPointVector = COMMSAT1;
GMAT EarthFixedXY.ViewDirection = Earth;
GMAT EarthFixedXY.ViewScaleFactor = 2.5;
GMAT EarthFixedXY.ViewUpCoordinateSystem = EarthMJ2000Eq;
GMAT EarthFixedXY.ViewUpAxis = Z;
GMAT EarthFixedXY.EclipticPlane = Off;
GMAT EarthFixedXY.XYPlane = On;
GMAT EarthFixedXY.WireFrame = Off;
GMAT EarthFixedXY.Axes = On;
GMAT EarthFixedXY.Grid = Off;
GMAT EarthFixedXY.SunLine = Off;
GMAT EarthFixedXY.UseInitialView = On;
GMAT EarthFixedXY.StarCount = 7000;
GMAT EarthFixedXY.EnableStars = On;
GMAT EarthFixedXY.EnableConstellations = On;

```

```
Create OrbitView J2000XY;
GMAT J2000XY.SolverIterations = None;
GMAT J2000XY.UpperLeft = [ 0.1571428571428571 0.08476190476190476 ];
GMAT J2000XY.Size = [ 0.7053571428571429 0.7495238095238095 ];
GMAT J2000XY.RelativeZOrder = 394;
GMAT J2000XY.Maximized = false;
GMAT J2000XY.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
GMAT J2000XY.CoordinateSystem = EarthMJ2000Eq;
GMAT J2000XY.DrawObject = [ true true true true ];
GMAT J2000XY.DataCollectFrequency = 1;
GMAT J2000XY.UpdatePlotFrequency = 50;
GMAT J2000XY.NumPointsToRedraw = 0;
GMAT J2000XY.ShowPlot = true;
GMAT J2000XY.MaxPlotPoints = 20000;
GMAT J2000XY.ShowLabels = true;
GMAT J2000XY.ViewPointReference = Earth;
GMAT J2000XY.ViewPointVector = COMMSAT1;
GMAT J2000XY.ViewDirection = Earth;
GMAT J2000XY.ViewScaleFactor = 2.5;
GMAT J2000XY.ViewUpCoordinateSystem = EarthMJ2000Eq;
GMAT J2000XY.ViewUpAxis = Z;
GMAT J2000XY.EclipticPlane = Off;
GMAT J2000XY.XYPlane = On;
GMAT J2000XY.WireFrame = Off;
GMAT J2000XY.Axes = On;
GMAT J2000XY.Grid = Off;
GMAT J2000XY.SunLine = Off;
GMAT J2000XY.UseInitialView = On;
GMAT J2000XY.StarCount = 7000;
GMAT J2000XY.EnableStars = On;
GMAT J2000XY.EnableConstellations = On;

%----------------------------------------
%---------- Functions
%----------------------------------------

Create GmatFunction CalcAzElRange;
GMAT CalcAzElRange.FunctionPath = '/Users/ninaaverill/unispace/2022/
   ↪ zeit8219-Satellite-Communications/assmt1/CalcAzElRange.gmf';

%----------------------------------------
%---------- Arrays, Variables, Strings
%----------------------------------------
Create Variable EarthRad;
GMAT EarthRad = 6371;



%----------------------------------------
%---------- Mission Sequence
%----------------------------------------

BeginMissionSequence;
While COMMSAT1.ElapsedSecs < 86400
   Propagate DefaultProp(COMMSAT1) DefaultProp(COMMSAT2) DefaultProp(
```

```
   COMMSAT3) {COMMSAT1.ElapsedSecs = 300};
   Report ReportFile1 COMMSAT1.UTCGregorian COMMSAT1.Earth.TA COMMSAT1.Earth.Latitude COMMSAT1.Earth.Longitude COMMSAT1.Earth.RMAG COMMSAT1.Earth.Altitude COMMSAT2.Earth.TA COMMSAT2.Earth.Latitude COMMSAT2.Earth.Longitude COMMSAT2.Earth.RMAG COMMSAT3.Earth.TA COMMSAT3.Earth.Latitude COMMSAT3.Earth.Longitude COMMSAT3.Earth.RMAG;
EndWhile;
```

5.2 Python Code Samples

```python
import numpy as np
import pandas as pd
import plotly.express as px

from link_calculator.components import Antenna, GroundStation, Satellite
from link_calculator.orbits.utils import (
    angle_sat_to_ground_station,
    elevation_angle,
    slant_range,
)
from link_calculator.propagation.conversions import (
    decibel_to_watt,
    frequency_to_wavelength,
    watt_to_decibel,
)
from link_calculator.propagation.utils import receive_power


def load_gmat_report(file_path):
    return pd.read_csv(file_path, sep="\s{2,}", engine="python")


def q2(
    report: pd.DataFrame, satellites: list[str], ground_stations: list[GroundStation]
):
    for sat in satellites:
        for gs in ground_stations:
            report[f"{sat.name}.Earth.Gamma.{gs.name}"] = report.apply(
                lambda row: angle_sat_to_ground_station(
                    gs.latitude,
                    gs.longitude,
                    row[f"{sat.name}.Earth.Latitude"],
                    row[f"{sat.name}.Earth.Longitude"],
                ),
                axis=1,
            )
            report[f"{sat.name}.Earth.SlantRange.{gs.name}"] = report.apply(
                lambda row: slant_range(
                    row[f"{sat.name}.Earth.RMAG"],
                    row[f"{sat.name}.Earth.Gamma.{gs.name}"],
```

```
                ),
                axis=1,
            )
            report[f"{sat.name}.Earth.Elevation.{gs.name}"] = report.apply(
                lambda row: elevation_angle(
                    row[f"{sat.name}.Earth.RMAG"],
                    row[f"{sat.name}.Earth.Gamma.{gs.name}"],
                ),
                axis=1,
            )

    min_elevation = 20
    for gs in ground_stations:
        assert all(
            np.logical_or.reduce(
                [
                    report[f"{sat.name}.Earth.Elevation.{gs.name}"] > min_elevation
                    for sat in satellites
                ]
            )
        )

    plot_elevation(report, satellites, ground_stations)
    return report


def q3(
    report: pd.DataFrame,
    satellites: list[str],
    ground_stations: list[GroundStation],
):
    for gs in ground_stations:
        for sat in satellites:
            report[f"{sat.name}.Earth.ReceivePower.{gs.name}"] = report.apply(
                lambda row: watt_to_decibel(
                    receive_power(
                        sat.antenna.power,
                        sat.antenna.loss,
                        sat.antenna.gain,
                        row[f"{sat.name}.Earth.SlantRange.{gs.name}"] * 1000,
                        gs.antenna.loss,
                        gs.antenna.gain,
                        1,
                        wavelength=frequency_to_wavelength(sat.antenna.frequency),
                    )
                )
                if row[f"{sat.name}.Earth.Elevation.{gs.name}"] > 20
                else None,
                axis=1,
            )
```

```python
    plot_receive_power(report, satellites, ground_stations)
    return report


def plot_receive_power(report, satellites, ground_stations):
    for gs in ground_stations:
        min_slant = report.iloc[
            report[f"{satellites[0].name}.Earth.SlantRange.{gs.name}"].idxmin()
        ]

        fig = px.line(
            report,
            x="UTC Gregorian",
            y=[f"{sat.name}.Earth.ReceivePower.{gs.name}" for sat in satellites],
            labels={"value": "Receive Power (dBW)"},
        )
        fig.add_vline(x=min_slant["UTC Gregorian"], line_dash="dash")
        fig.add_annotation(
            x=min_slant["UTC Gregorian"],
            y=min_slant[f"{satellites[0].name}.Earth.ReceivePower.{gs.name}"],
            text=f"Slant range = {min_slant[f'{satellites[0].name}.Earth.SlantRange.{gs.name}']:.2f}",
        )
        fig.update_layout(
            title={
                "text": f"Power received by {gs.name}",
                "x": 0.5,
                "xanchor": "center",
                "yanchor": "top",
            }
        )
        fig.show()
        print(gs.name, min_slant[f"{satellites[0].name}.Earth.ReceivePower.{gs.name}"], "dBW")


def plot_elevation(report, satellites, ground_stations):
    for gs in ground_stations:
        min_slant = report.iloc[
            report[f"{satellites[0].name}.Earth.SlantRange.{gs.name}"].idxmin()
        ]

        cols = [f"{sat.name}.Earth.Elevation.{gs.name}" for sat in satellites]
        elevation = report[cols]
        elevation = elevation[elevation > 20]
        elevation["UTC Gregorian"] = report["UTC Gregorian"]

        fig = px.line(
            elevation,
            x="UTC Gregorian",
```

```
            y=cols,
            labels={"value": "Elevation (deg)"},
        )
        fig.add_vline(x=min_slant["UTC Gregorian"], line_dash="dash")
        fig.add_annotation(
            x=min_slant["UTC Gregorian"],
            y=min_slant[f"{satellites[0].name}.Earth.Elevation.{gs.name}"],
            text=f"Slant range = {min_slant[f'{satellites[0].name}.Earth.SlantRange.{gs.name}']:.2f}",
        )
        fig.update_layout(
            title={
                "text": f"Elevation above {gs.name}",
                "x": 0.5,
                "xanchor": "center",
                "yanchor": "top",
            }
        )
        fig.show()
        print(gs.name, min_slant[f"{satellites[0].name}.Earth.Altitude"], "km")


if __name__ == "__main__":
    # ka_band = 26.5  40   GHz
    sat_frequency = 20
    gs_frequency = 30
    ground_stations = [
        GroundStation(
            "CocosIsland",
            -12.167,
            96.833,
            0,
            Antenna("CocosReceive", 0, decibel_to_watt(18), 1, gs_frequency),
        ),
        GroundStation(
            "MawsonResearchStation",
            -67.6,
            62.867,
            0,
            Antenna("MawsonReceive", 0, decibel_to_watt(18), 1, gs_frequency),
        ),
        GroundStation(
            "NorfolkIsland",
            -29.033,
            167.95,
            0,
            Antenna("MawsonReceive", 0, decibel_to_watt(18), 1, gs_frequency),
        ),
        GroundStation(
            "MacquarieResearchStation",
```

```
            -54.5,
            158.95,
            0,
            Antenna("MacquarieReceive", 0, decibel_to_watt(18), 1, gs_frequency),
        ),
    ]

    satellites = [
        Satellite(
            "COMMSAT1",
            Antenna("COMMSAT1Transmit", 5, decibel_to_watt(30), 1, sat_frequency),
        ),
        Satellite(
            "COMMSAT2",
            Antenna("COMMSAT2Transmit", 5, decibel_to_watt(30), 1, sat_frequency),
        ),
        Satellite(
            "COMMSAT3",
            Antenna("COMMSAT3Transmit", 5, decibel_to_watt(30), 1, sat_frequency),
        ),
    ]

    report = load_gmat_report("input/OrbitParams.txt")
    report["UTC Gregorian"] = pd.to_datetime(report["COMMSAT1.UTCGregorian"])
    report = q2(report, satellites, ground_stations)
    report = q3(report, satellites, ground_stations)
    report.to_csv("output/report.csv")
```

Listing 1: main.py

```
# Orbits
EARTH_GRAVITATIONAL_CONSTANT = 6.6743e-11  # Nm^2 / kg^2
EARTH_MASS = 5.98e24  # km^3/s^2
EARTH_MU = 3.986004418e5  # km^3/s^-2
EARTH_RADIUS = 6378.14  # km^3/s^2
EARTH_SOLAR_YEAR = 365.25  # days
SIDEREAL_DAY = 23.935  # hours

# Propagation
SPEED_OF_LIGHT = 299792458  # m/s
```

Listing 2: constants.py

```
from math import atan2, cos, degrees, radians

import numpy as np

from link_calculator.constants import EARTH_RADIUS


```

```python
def power_density(power: float, gain: float, distance: float) -> float:
    """
    Calculate the power density of the wavefront

    Parameters
    ---------
        power (float, W): the transmitted power
        gain (float, W): the power gained
        distance (float, m): the distance between the transmit and receive antennas

    Returns
    ------
        power_density (float, W/m^2): the power density at distance d
    """
    return (power * gain) / (4 * np.pi * distance**2)


def eirp(power: float, loss: float, gain: float) -> float:
    """
    Calculate the Effetive Isotropic Radiated Power

    Parameters
    ----------
        power (float, W): the total output amplifier power
        loss (float, ): coupling loss between transmitter and antenna
            in the range [0, 1]
        gain (float, ): transmitter gain in the direction of the
            receiving antenna

    Returns
    -------
        eirp (float, dB): power incident at the receiver that would have  had to be radiated
            from an isotropic antenna to achieve the same power incident  at the
            receiver  as that of a transmitter with a specific antenna gain
    """
    return power * loss * gain


def power_density_eirp(eirp: float, distance: float, atmospheric_loss: float) -> float:
    """
    Calculate the power density of the wavefront using EIRP

    Parameters
    ---------
        eirp (float, dB)
        distance (float, m): the distance between the transmit and receive antennas
        atmospheric_loss (float, ): the total losses due to the atmosphere

```

```
    Returns
    ------
        power_density (float, W/m^2): the power density at distance d
    """
    return eirp / (4 * np.pi * distance**2) * atmospheric_loss


def effective_aperture(gain: float, wavelength: float) -> float:
    """
    Calculate the effective area of the receiving antenna

    Parameters
    ----------
        gain (float, ): gain of the receive antenna
        wavelength (float, m): the radiation wavelength

    Returns
    -------
        effective_aperture (float, m^2): the effive aperture of the receive antenna
    """
    return gain * wavelength**2 / (4 * np.pi)


def receive_power(
    transmit_power: float,
    transmit_loss: float,
    transmit_gain: float,
    distance: float,
    receive_loss: float,
    receive_gain: float,
    atmospheric_loss: float,
    wavelength: float = None,
    eff_aperture: float = None
) -> float:
    """
    Calculate the power collected by the receive antenna

    Parameters
    ----------
        amp_power (float, W): the total output amplifier power
        power_density (float, W/m^2): the power density at distance d
        transmit_loss (float, ): coupling loss between transmitter and antenna
            in the range [0, 1]
        transmit_gain (float, ): transmitter gain in the direction of the
            receiving antenna
        distance (float, m): the distance between the transmit and receive antennas
        receive_gain (float, ): the gain at the recieve antenna
        receive_loss (float, ): coupling loss between receiver terminals  and antenna
            in the range [0, 1]
        wavelength (float, m): the radiation wavelength
```

```
        eff_aperture (float, m): effective aperture of the receive antenna
        atmospheric_loss (float, ): The loss due to the atmosphere

    Returns
    -------
        receive_power (float, W): the total collected power at the receiver's terminals
    """
    eirp_ = eirp(transmit_power, transmit_loss, transmit_gain)
    pow_density = power_density_eirp(eirp_, distance, atmospheric_loss)

    if not eff_aperture and wavelength:
      eff_aperture = effective_aperture(receive_gain, wavelength)
    if not (eff_aperture or wavelength):
        raise ValueError("No effective aperture or wavelength supplied")

    return pow_density * eff_aperture * receive_loss


def free_space_loss(distance: float, wavelength: float) -> float:
    """
    Calculate the free space loss between two antennas

    Parameters
    ----------
        distance (float, km): distance between the transmit and receive antennas
        frequency (float, GHz): frequency of the transmitter

    Returns
    -------
        path_loss (float, )

    """
    return (wavelength / (4 * np.pi * distance)) ** 2


def free_space_loss_db(slant_range: float, frequency: float):
    """
    Calculate the free space loss between two antennas

    Parameters
    ----------
        slant_range (float, km): slant range between the transmit and receive antennas
        frequency (float, GHz): frequency of the transmitter

    Returns
    -------
        path_loss (float, dB): The path loss over the
    """
    return -92.44 - 20 * np.log10(slant_range * frequency)


```

```
def slant_path(
    elevation_angle: float,
    rain_altitude: float,
    station_altitude: float,
) -> float:
    """
    Calculate the slant path

    Parameters
    ----------
        angle_of_elevation (float, deg): the angle between the Earth station and the satellite
        rain_height (float, km): the rain height
        station_altitude (float, km): the rain height of the Earth station above sea level
        refraction_radius (float, km): The modified radius of the Earth to account for the
            refraction of the wave by thr troposphere

    Returns
    -------
        d_s (float, km): The slant height
    """
    refraction_radius = 8500 if station_altitude < 1.0 else EARTH_RADIUS
    elevation_angle_rad = radians(elevation_angle)
    if elevation_angle < 5:
        return (
            2
            * (rain_altitude - station_altitude)
            / np.sqrt(
                np.sin(elevation_angle_rad) ** 2
                + 2 * (rain_altitude - station_altitude) / refraction_radius
            )
        )
    else:
        return (rain_altitude - station_altitude) / np.sin(elevation_angle_rad)


def rain_specific_attenuation(frequency: float, rain_rate: float, polarization: str):
    """
    TODO()

    Parameters
    ----------

    Returns
    -------

    """
    _f = [
        1,
        2,
```

```
            4,
            6,
            7,
            8,
            10,
            12,
            15,
            20,
            25,
            30,
            35,
            40,
            45,
            50,
            60,
            70,
            80,
            90,
            100,
            120,
            150,
            200,
            300,
            400,
        ]

        _kH = [
            0.0000387,
            0.000154,
            0.00065,
            0.00175,
            0.00301,
            0.00454,
            0.0101,
            0.0188,
            0.0367,
            0.0751,
            0.124,
            0.187,
            0.263,
            0.35,
            0.442,
            0.536,
            0.707,
            0.851,
            0.975,
            1.06,
            1.12,
            1.18,
            1.31,
            1.45,
            1.36,
            1.32,
        ]

```

```
    _kV = [
        0.0000352,
        0.000138,
        0.000591,
        0.00155,
        0.00265,
        0.00395,
        0.00887,
        0.0168,
        0.0335,
        0.0691,
        0.113,
        0.167,
        0.233,
        0.31,
        0.393,
        0.479,
        0.642,
        0.784,
        0.906,
        0.999,
        1.06,
        1.13,
        1.27,
        1.42,
        1.35,
        1.31,
    ]

    _alphaH = [
        0.912,
        0.963,
        1.121,
        1.308,
        1.332,
        1.327,
        1.276,
        1.217,
        1.154,
        1.099,
        1.061,
        1.021,
        0.979,
        0.939,
        0.903,
        0.873,
        0.826,
        0.793,
        0.769,
        0.753,
        0.743,
        0.731,
        0.71,
        0.689,
        0.688,
```

```
        0.683,
    ]

    _alphaV = [
        0.88,
        0.923,
        1.075,
        1.265,
        1.312,
        1.31,
        1.264,
        1.2,
        1.128,
        1.065,
        1.03,
        1,
        0.963,
        0.929,
        0.897,
        0.868,
        0.824,
        0.793,
        0.769,
        0.754,
        0.744,
        0.732,
        0.711,
        0.69,
        0.689,
        0.684,
    ]

    KH = np.exp(np.interp(np.log(frequency), np.log(_f), np.log(_kH)))
    KV = np.exp(np.interp(np.log(frequency), np.log(_f), np.log(_kV)))

    alphaH = np.interp(np.log(frequency), np.log(_f), _alphaH)
    alphaV = np.interp(np.log(frequency), np.log(_f), _alphaV)

    if polarization == "circular":
        k = (KH + KV) / 2
        alpha = (KH * alphaH + KV * alphaV) / (2 * k)
    elif polarization == "vertical":
        k = KV
        alpha = alphaV
    elif polarization == "horizontal":
        k = KH
        alpha = alphaH
    else:
        raise Exception("Invalid Polarization")

    return k, alpha, k * rain_rate**alpha


def horizontal_reduction(
    horizontal_projection: float, specific_attenuation: float, frequency
```

```
    ↪ : float
) -> float:
    """
    TODO()

    Parameters
    ----------

    Returns
    -------
    """
    return 1 / (
        1
        + 0.78 * np.sqrt(horizontal_projection * specific_attenuation /
    ↪ frequency)
        - 0.38 * (1 - np.exp(-2 * horizontal_projection))
    )


def vertical_adjustment(
    elevation_angle: float,
    specific_attenuation: float,
    d_r: float,
    frequency: float,
    chi: float,
) -> float:
    """
    Calculate the vertical adjustment factor

    Parameters
    ----------
        elevation_angle (float, deg):
        specific_attenation (float, dBKm-1)
        d_r (float, km):
        frequency (float, GHz):
        chi (float, deg):

    Returns
    -------
        vert_adj (float, )
    """
    return 1 / (
        1
        + np.sqrt(np.sin(radians(elevation_angle)))
        * (
            31
            * (1 - np.exp(-elevation_angle / (1 + chi)))
            * (np.sqrt(d_r * specific_attenuation) / (frequency**2))
            - 0.45
        )
    )


def zeta(
    rain_altitude: float,
```

```
    station_altitude: float,
    horizontal_projection: float,
    horizontal_reduction: float,
) -> float:
    """
    Calculate interim vertical adjustment value

    Parameters
    ----------

    Returns
    -------
    """
    return degrees(
        atan2(
            rain_altitude - station_altitude,
            horizontal_projection * horizontal_reduction,
        )
    )


def rain_attenuation(
    elevation_angle: float,
    slant_path: float,
    frequency: float,
    rain_altitude: float,
    station_altitude: float,
    station_latitude: float,
    rain_rate: float = 0.01,
    polarization: str = "vertical",
) -> float:
    """

    Parameters
    ----------

    Returns
    -------

    """
    elevation_angle_rad = radians(elevation_angle)
    horiz_proj = slant_path * np.cos(elevation_angle_rad)

    _, _, specific_att = rain_specific_attenuation(frequency, rain_rate,
     ↪ polarization)

    horiz_reduction = horizontal_reduction(horiz_proj, specific_att,
     ↪ frequency)

    zeta_ = zeta(rain_altitude, station_altitude, horiz_proj,
     ↪ horiz_reduction)

    if zeta_ > elevation_angle:
        d_r = horiz_proj * horiz_reduction / np.cos(elevation_angle_rad)
    else:
```

```
        d_r = slant_path

    if abs(station_latitude) < 36:
        chi = 36 - abs(station_latitude)
    else:
        chi = 0

    vert_adj = vertical_adjustment(elevation_angle, specific_att, d_r, frequency, chi)
    effective_path = slant_path * vert_adj

    return specific_att * effective_path


def worst_rain_rate(rain_rate: float) -> float:
    return (rain_rate / 0.3) ** 0.87


def polarization_loss(faraday_rotation: float) -> float:
    rotation_rad = radians(faraday_rotation)
    return 20 * np.log(cos(rotation_rad))
```

Listing 3: propagation.py

```
from math import acos, atan, atan2, cos, degrees, pi, radians, sin, sqrt, tan

from link_calculator.constants import EARTH_MU, EARTH_RADIUS


def velocity(
    semi_major_axis: float, orbital_radius: float, mu: float = EARTH_MU
) -> float:
    """
    Calculate the velocity of a satellite in orbit according to Kepler's second law

    Parameters
    ---------
        semi_major_axis (float, km): The semi-major axis of the orbit
            orbital_radius (float, km): distance from the centre of mass to the satellite
        mu (float, optional): Kepler's gravitational constant

    Returns
    ------
        velocity (float, km/s): the orbit speed of the satellite
    """
    return sqrt(mu * (2 / orbital_radius - 1 / semi_major_axis))


def velocity_circular(orbital_radius: float, mu: float = EARTH_MU) -> float:
    """
    Special case of velocity where r = a
```

```

    Parameters
    ---------
        orbital_radius (float, km): distance from the centre of mass to the satellite
        mu (float, optional): Kepler's gravitational constant

    Returns
    ------
        velocity (float, km/s): the orbit speed of the satellite
    """
    return velocity(orbital_radius, orbital_radius, mu)


def period(semi_major_axis: float, mu: float = EARTH_MU) -> float:
    """
    Calculate the period of the satellite's orbit according to Kepler's third law

    Parameters
    ---------
        semi_major_axis (float, km): The semi-major axis of the orbit
        mu (float, km^3/s^-2, optional): Kepler's gravitational constant

    Returns
    ------
        period (float, s): the time taken for the satellite to complete a revolution
    """
    return 2 * pi * sqrt(semi_major_axis**3 / mu)


def angle_sat_to_ground_station(
    ground_station_lat: float,
    ground_station_long: float,
    sat_lat: float,
    sat_long: float,
) -> float:
    """
    Calculate angle gamma at the centre of the ground, between the Earth station and the satellite

    Parameters
    ---------
        ground_station_lat (float, deg): the latitude of the ground station
        ground_station_long (float, deg): the longitude of the ground station
        sat_lat (float, deg): the latitude of the satellite
        sat_long (float, deg): the longitude of the satellite

    Returns
    ------
        gamma (float, rad): angle between satellite and ground station
    """
```

```python
    gs_lat_rad = radians(ground_station_lat)
    gs_long_rad = radians(ground_station_long)
    sat_lat_rad = radians(sat_lat)
    sat_long_rad = radians(sat_long)

    gamma = acos(
        cos(gs_lat_rad) * cos(sat_lat_rad) * cos(sat_long_rad - gs_long_rad)
        + sin(gs_lat_rad) * sin(sat_lat_rad)
    )
    return degrees(gamma)


def angle_sat_to_gs_orbital_radius(
    orbital_radius: float, planet_radius: float = EARTH_RADIUS, elevation: float = 0
):
    """
    Calculate angle gamma at the centre of the ground, between the Earth
    station and the satellite, given the orbital radius of the satellite

    Parameters
    ---------
        orbital_radius (float, km): distance from the centre of mass to the satellite
        planet_radius (float, km, optional): radius of the planet
        elevation (float, deg): the angle of elevation over the horizon

    Returns
    ------
        gamma (float, deg): angle between satellite and ground station
    """
    elevation_rad = radians(elevation)
    gamma = acos((planet_radius * cos(elevation_rad)) / orbital_radius) - elevation_rad
    return degrees(gamma)


def slant_range(
    orbital_radius: float, angle_sat_to_gs: float, planet_radius: float = EARTH_RADIUS
) -> float:
    """
    Calculate the slant range from the ground station to the satellite

    Parameters
    ----------
        orbital_radius (float, km): distance from the centre of mass to the satellite
        angle_sat_to_gs (float, deg): angle from the satellite to the ground station, centred at the centre of mass
        planet_radius (float, km, optional): radius of the planet

    Return
    ------
```

```python
        slant_range (float, km): the distance from the ground station to the satellite

    """
    return sqrt(
        planet_radius**2
        + orbital_radius**2
        - 2 * planet_radius * orbital_radius * cos(radians(angle_sat_to_gs))
    )


def elevation_angle(
    orbital_radius: float, angle_sat_to_gs: float, planet_radius: float = EARTH_RADIUS
) -> float:
    """
    Calculate the elevation angle from the ground station to the satellite

    Parameters
    ----------
        orbital_radius (float, km): distance from the centre of mass to the satellite
        angle_sat_to_gs (float, rad): angle from the satellite to the ground station, centred at the centre of mass
        planet_radius (float, km, optional): radius of the planet

    Return
    ------
        elevation_angle (float, rad): the distance from the ground station to the satellite

    """
    gamma_rad = radians(angle_sat_to_gs)
    elev = acos(
        sin(gamma_rad)
        / sqrt(
            1
            + (planet_radius / orbital_radius) ** 2
            - 2 * (planet_radius / orbital_radius) * cos(gamma_rad)
        )
    )
    return degrees(elev)


def area_of_coverage(angle_sat_to_gs: float, planet_radius: float = EARTH_RADIUS):
    """
    Calculate the surface area coverage of the Earth from a satellite

    Parameters
    ----------
        angle_sat_to_gs (float, deg): angle from the satellite to the ground station, centred at the centre of mass
```

```
        planet_radius (float, km, optional): radius of the planet

    Return
    ------
        area_coverage (float, km): the area of the Earth's surface visible from a satellite

    """
    angle_sat_to_gs_rad = radians(angle_sat_to_gs)
    return 2 * pi * (planet_radius**2) * (1 - cos(angle_sat_to_gs_rad))


def percentage_of_coverage(
    angle_sat_to_gs: float, planet_radius: float = EARTH_RADIUS
) -> float:
    """
    Calculate the surface area coverage of the Earth from a satellite as a percentage of the total
        surface area

    Parameters
    ----------
        angle_sat_to_gs (float, rad): angle from the satellite to the ground station, centred at the centre of mass
        planet_radius (float, km, optional): radius of the planet

    Return
    ------
        area_coverage (float, %): the percentage of the Earth's surface visible from a satellite

    """
    return (
        area_of_coverage(angle_sat_to_gs, planet_radius)
        / (4 * pi * planet_radius**2)
        * 100
    )


def percentage_of_coverage_gamma(angle_sat_to_gs: float) -> float:
    """
    Calculate the surface area coverage of the Earth from a satellite as a percentage of the total
        surface area

    Parameters
    ----------
        angle_sat_to_gs (float, rad): angle from the satellite to the ground station, centred at the centre of mass

    Return
    ------
        area_coverage (float, %): the percentage of the Earth's surface visible from a satellite

```

```
    """
    gamma_rad = radians(angle_sat_to_gs)
    return 50 * (1 - cos(gamma_rad))


def azimuth_intermediate(
    ground_station_lat: float, ground_station_long: float, sat_long: float
) -> float:
    """
    Calculate the azimuth of a geostationary satellite

    Parameters
    ---------
        ground_station_lat (float, deg): the latitude of the ground station
        ground_station_long (float, deg): the longitude of the ground station
        sat_long (float, deg): the longitude of the satellite

    Returns
    -------
        azimuth (float, rad): horizontal pointing angle of the ground station antenna to
            the satellite. The azimuth angle is usually measured in clockwise direction
            in degrees from true north.
    """
    gs_lat_rad = radians(ground_station_lat)
    gs_long_rad = radians(ground_station_long)
    sat_long_rad = radians(sat_long)
    az = atan(tan(abs(sat_long_rad - gs_long_rad)) / sin(gs_lat_rad))
    return degrees(az)


def max_visible_distance(
    orbital_radius: float,
    ground_station_lat: float,
    min_angle: float = 0.0,
    planet_radius: float = EARTH_RADIUS,
) -> float:
    """
    Calculate the radius of visibility for a satellite and a ground station

    Parameters
    ---------
        orbital_radius (float, km): the radius of the satellite from the centre of the Earth
        ground_station_lat (float, deg): the latitude of the ground station
        min_angle (float, deg): the minimum angle of visibility over the horizon
        planet_radius (float, km, optional): radius of the planet

```

```
    Returns
    ------

    """
    gs_lat_rad = radians(ground_station_lat)
    min_angle_rad = radians(min_angle)
    return acos(
        cos(
            (
                acos((planet_radius / orbital_radius) * cos(min_angle_rad))
                - min_angle_rad
            )
            / cos(gs_lat_rad)
        )
    )


def azimuth(
    ground_station_lat: float,
    ground_station_long: float,
    sat_lat: float,
    sat_long: float,
) -> float:
    """
    Calculate the azimuth of a satellite

    Parameters
    ----------
        ground_station_lat (float, deg): the latitude of the ground station
        ground_station_long (float, deg): the longitude of the ground station
        sat_lat (float, deg): the latitude of the satellite
        sat_long (float, deg): the longitude of the satellite

    Returns
    ------
        azimuth (float, rad): horizontal pointing angle of the ground station antenna to
            the satellite. The azimuth angle is usually measured in clockwise direction
            in degrees from true north.
    """
    # Ground station is in northern hemisphere
    # rel_pos = ground_station_long - sat_long
    #
    # if 90 > gs_long_rad > 0:
    #
    ## At the equator
    ## In the southern hemisphere
    # elif 0 > gs_long_rad >= -`90:
    #    if (sat_lat - ground_station_lat)
    #        return pi / 2
    #    else:
```

```
    # raise ValueError("Invalid value for the ground station's longitude
↪ : must be in range -90 < L < 90")
```

Listing 4: orbits.py

```
class Antenna:
    def __init__(
        self, name: str, power: float, gain: float, loss: float,
↪ frequency: float
    ):
        """
        Instantiate an Antenna object

        Parameters
        ----------
            name (str): Name of antenna
            power (float, W): transmit/receive power of the antenna
            gain (float, W): the power gain of the antenna
            loss (float, W): the power loss of the antenna
            frequency (float, GHz): the transmit frequency of the
↪ antenna
        """
        self.power = power
        self.gain = gain
        self.loss = loss
        self.frequency = frequency


class GroundStation:
    def __init__(
        self,
        name: str,
        latitude: float,
        longitude: float,
        altitude: float,
        antenna: Antenna,
    ):
        """

        Parameters
        ----------
            name (str,): name of the ground station
            latitude (str, deg): the latitude of the groundstation
            longitude (str, deg): the longitude of the groundstation
            altitude (str, km): the altitude of the groundstation above
↪ sea level
        """
        self.name = name
        self.latitude = latitude
        self.longitude = longitude
        self.altitude = altitude
        self.antenna = antenna


class Satellite:
```

```
    def __init__(self, name: str, antenna: Antenna):
        self.name = name
        self.antenna = antenna
```

Listing 5: components.py

```
import numpy as np

def axes_to_eccentricity(float a, float b):
    return np.sqrt(a**2, b**2) / a
```

Listing 6: conversions.py