

# ZEIT8219

## Satellite Communications

*Assignment 1*

NINA AVERILL  
z3531215

UNSW Canberra  
Apr 2022

# Contents

<b>1</b>	<b>Report Summary</b>	<b>3</b>
<b>2</b>	<b>Satellite Constellation</b>	<b>4</b>
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
<b>3</b>	<b>Ground Station Link Analysis</b>	<b>6</b>
3.1	Coverage . . . . .	6
3.2	Link Analysis . . . . .	8
<b>4</b>	<b>List of Figures</b>	<b>11</b>
<b>5</b>	<b>Appendix</b>	<b>18</b>
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)	Eccentricity	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  $\gamma$  at the centre of the Earth.

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

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].

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

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].

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

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].

$$EIRP = P_t G_t \quad (W) \quad (4)$$

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].

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

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

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

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].

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

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].

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

Where:

$G_r$  is the gain of the receive antenna

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

$$P_{r(\text{dBW})} = 10 \log_{10} P_{r(W)} \quad (9)$$

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 side-real 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

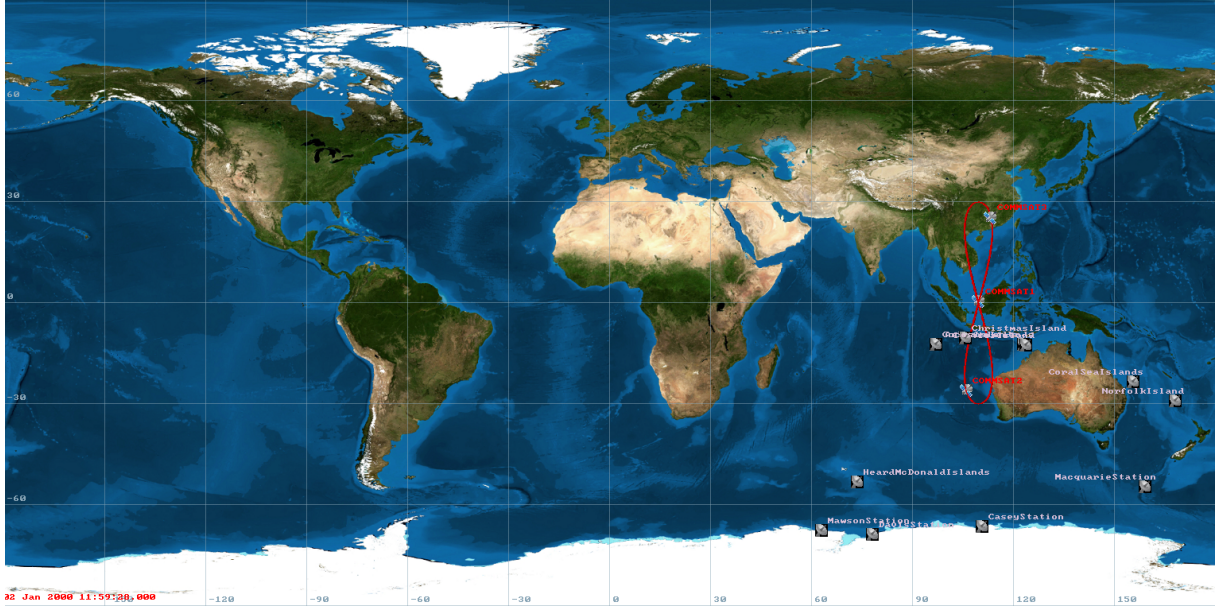


Figure 1: Ground track of constellation over a sidereal day

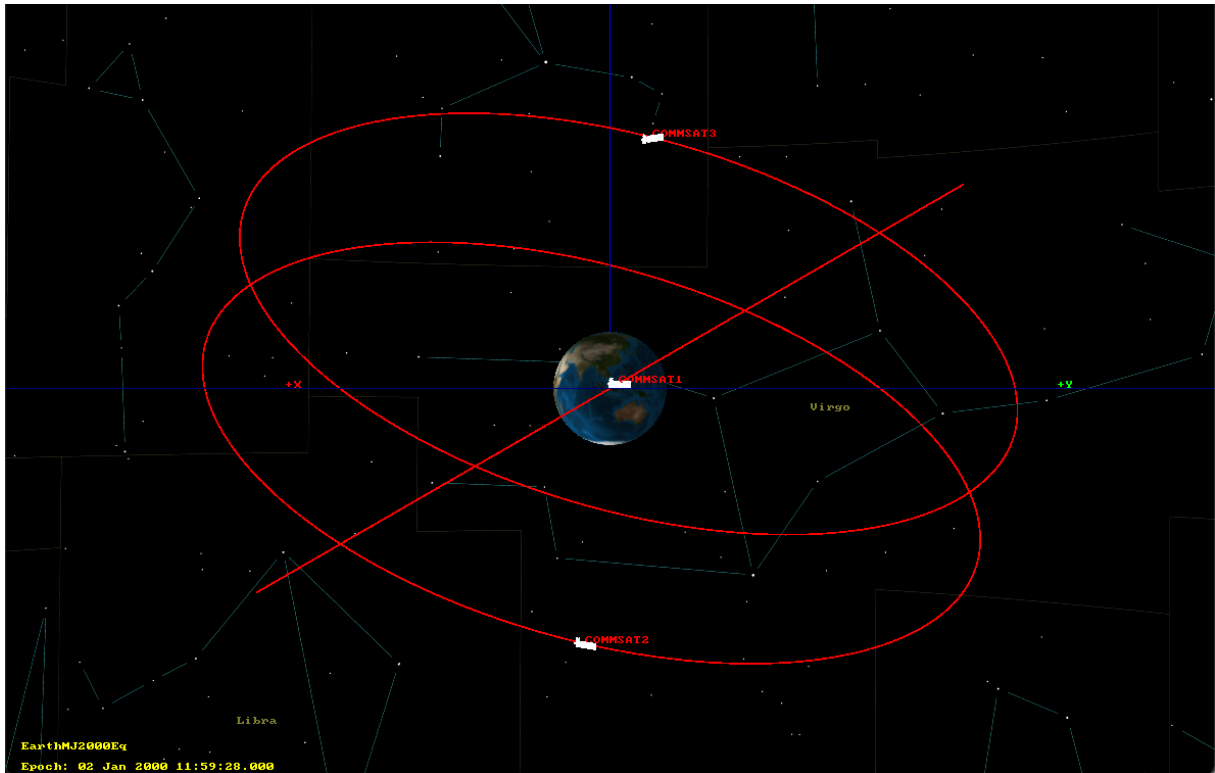


Figure 2: Constellation orbits over a sidereal day in the J2000 reference frame, showing the Earth's XY plane

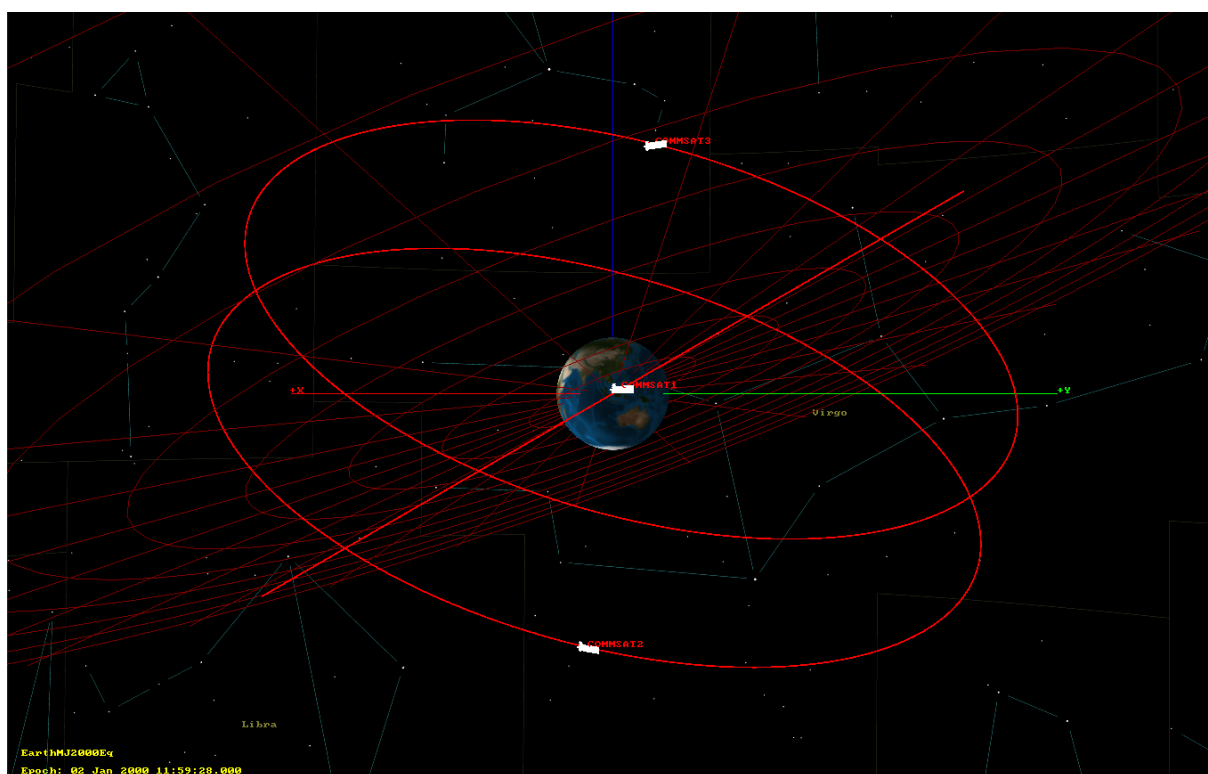


Figure 3: Constellation orbits over a sidereal day in the J2000 reference frame, showing the Earth's ecliptic plane

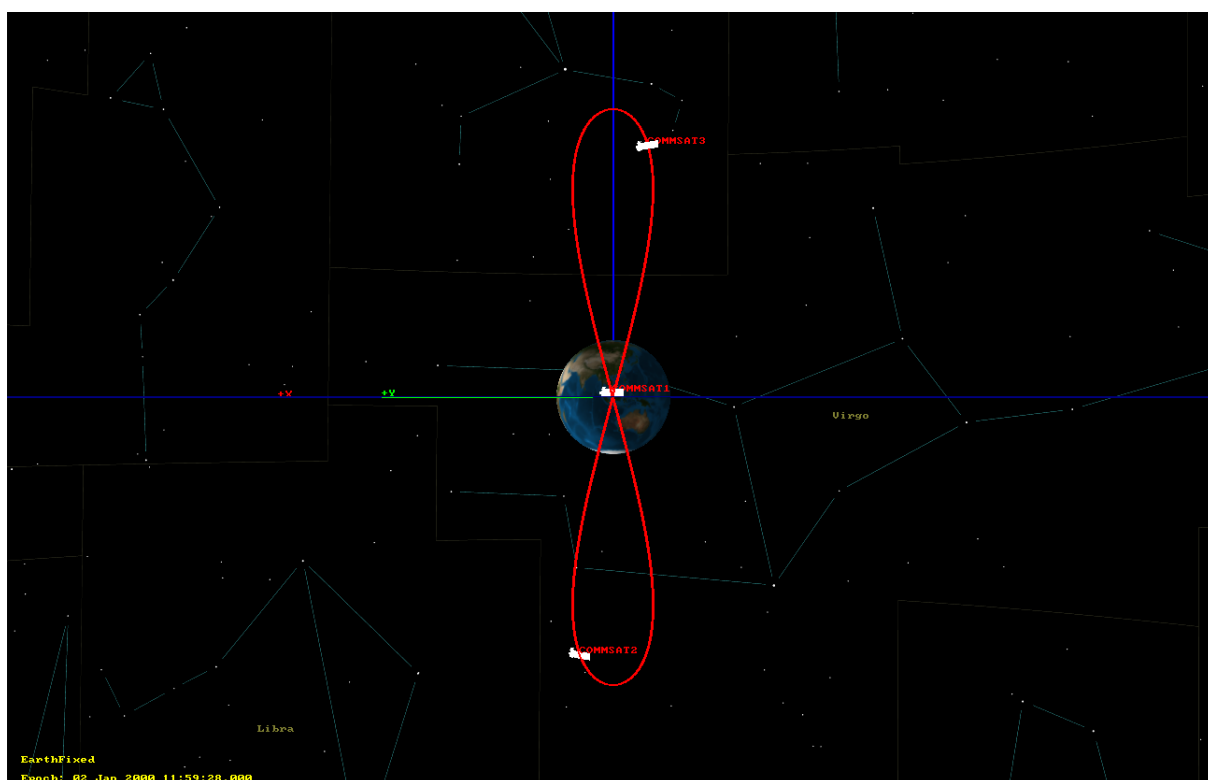


Figure 4: Constellation orbits over a sidereal day in the Earth-Fixed reference frame, showing the Earth's XY plane

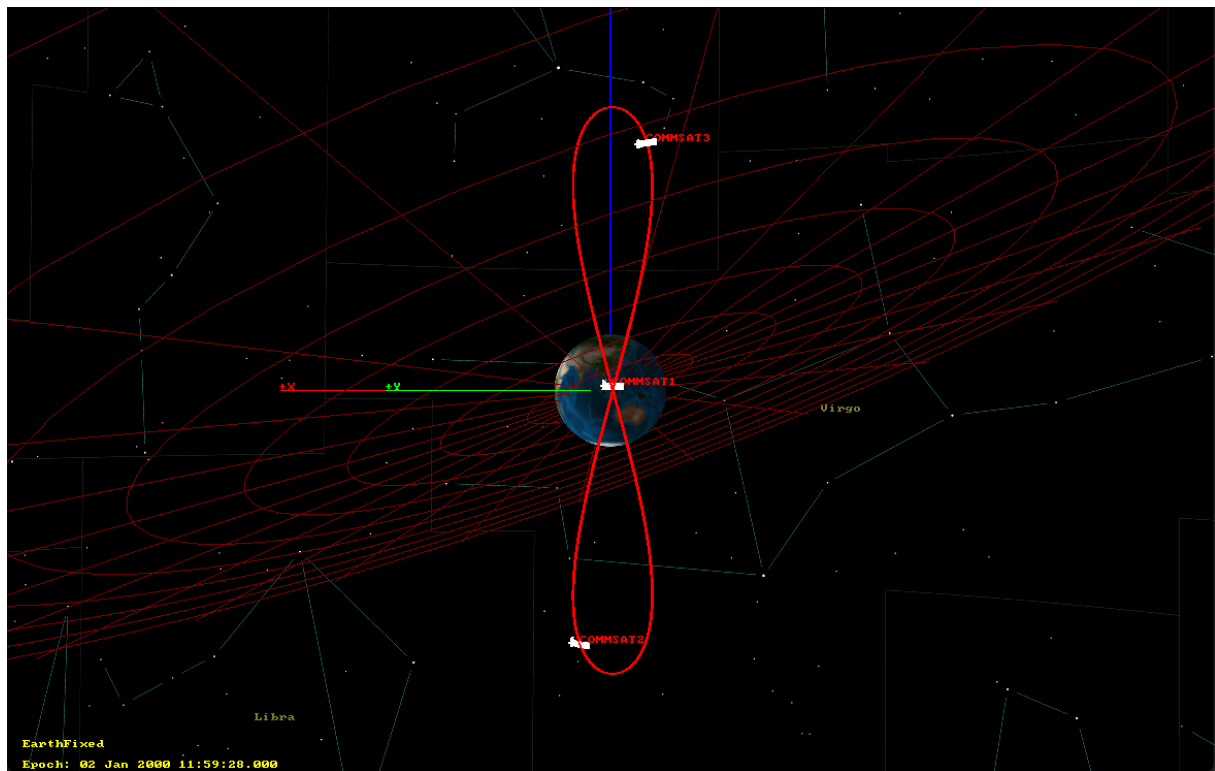


Figure 5: Constellation orbits over a sidereal day in the Earth-Fixed reference frame, showing the Earth's ecliptic plane

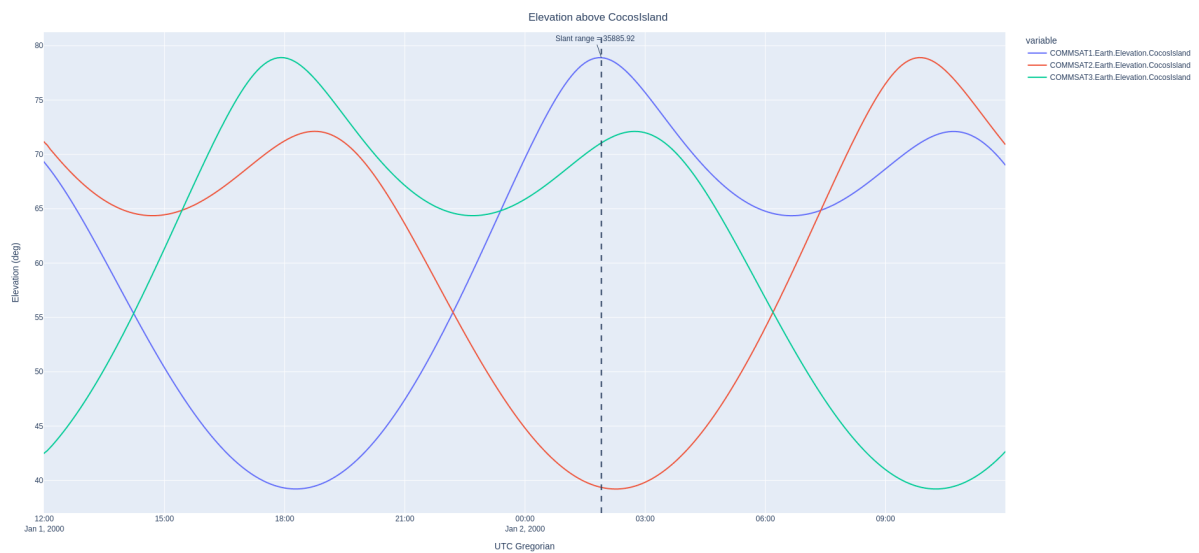


Figure 6: Elevation of constellation over with reference to Cocos (Keeling) Island over a sidereal day

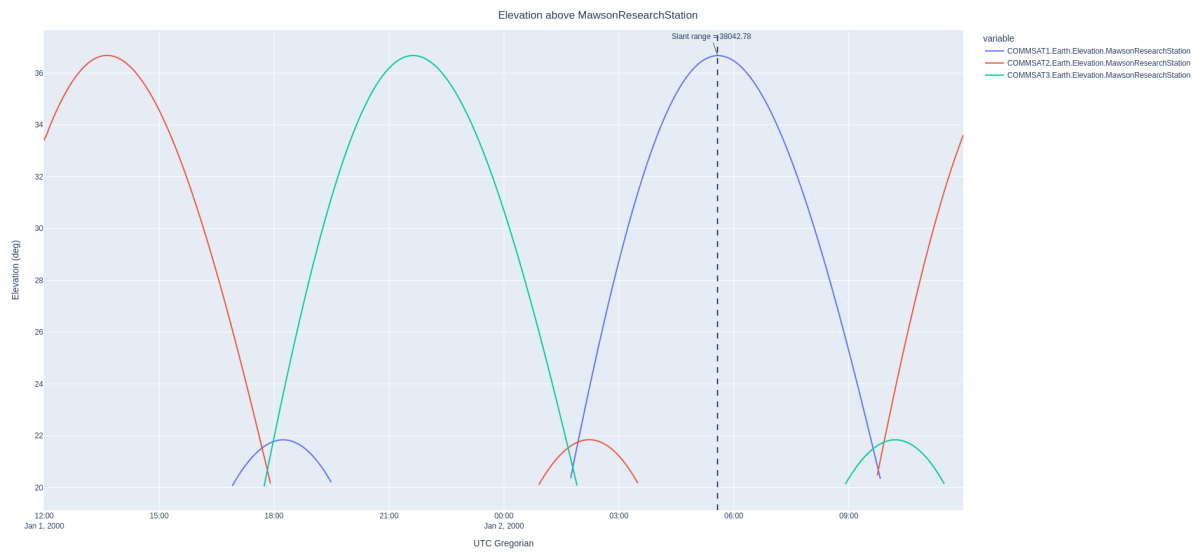


Figure 7: Elevation of constellation with reference to Mawson Research Station over a sidereal day

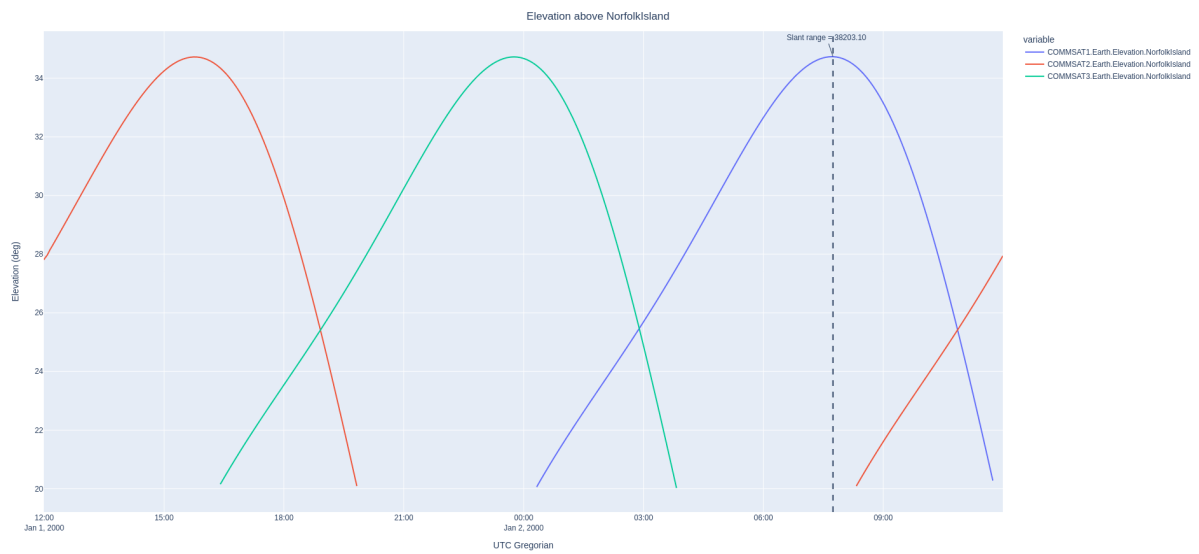


Figure 8: Elevation of constellation over with reference to Norfolk Island over a sidereal day

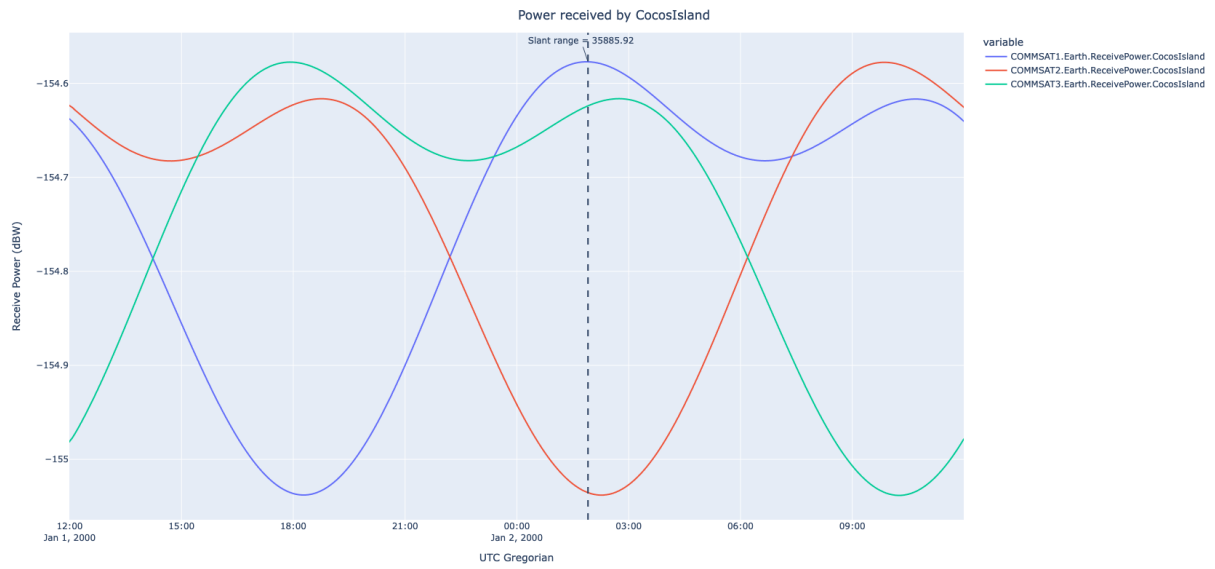


Figure 9: Power received at Cocos Island from constellation over a sidereal day

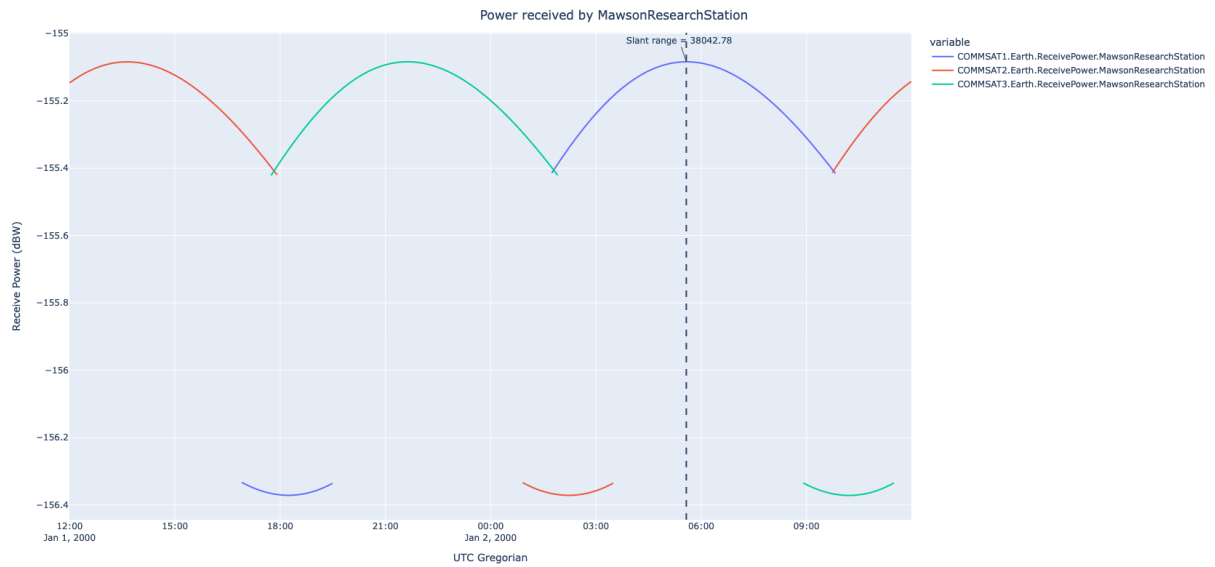


Figure 10: Power received at Mawson Research Station from constellation over a sidereal day



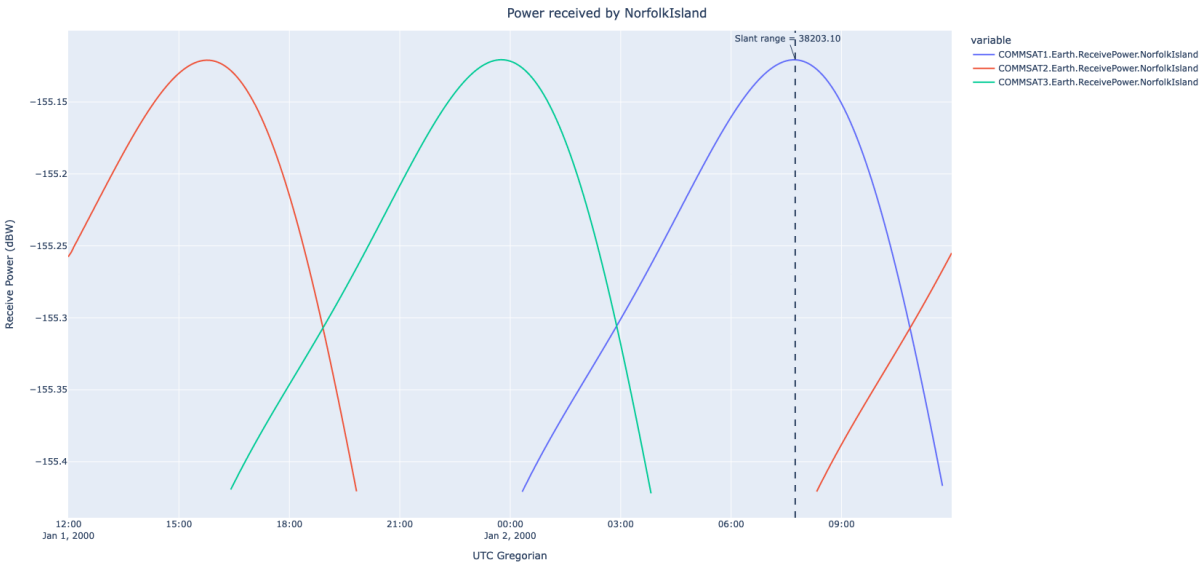


Figure 11: Power received at Norfolk Island from constellation over a sidereal day

## 5 Appendix

### 5.1 GMAT Script

```

1 %General Mission Analysis Tool(GMAT) Script
2 %Created: 2022-04-07 12:29:17
3
4
5 %-----
6 %----- Spacecraft
7 %-----
8
9 Create Spacecraft COMMSAT1;
10 GMAT COMMSAT1.DateFormat = TAIModJulian;
11 GMAT COMMSAT1.Epoch = '21545';
12 GMAT COMMSAT1.CoordinateSystem = EarthMJ2000Eq;
13 GMAT COMMSAT1.DisplayStateType = Keplerian;
14 GMAT COMMSAT1.SMA = 42164.999999999999;
15 GMAT COMMSAT1.ECC = 3.794636528864267e-16;
16 GMAT COMMSAT1.INC = 29.999999999999998;
17 GMAT COMMSAT1.RAAN = 30;
18 GMAT COMMSAT1.AOP = 0;
19 GMAT COMMSAT1.TA = 1.207418269725733e-06;
20 GMAT COMMSAT1.DryMass = 850;
21 GMAT COMMSAT1.Cd = 2.2;
22 GMAT COMMSAT1.Cr = 1.8;
23 GMAT COMMSAT1.DragArea = 15;
24 GMAT COMMSAT1.SRPArea = 1;
25 GMAT COMMSAT1.SPADDragScaleFactor = 1;
26 GMAT COMMSAT1.SPADSRPScaleFactor = 1;
27 GMAT COMMSAT1.NAIFId = -10001001;
28 GMAT COMMSAT1.NAIFIdReferenceFrame = -9001001;
29 GMAT COMMSAT1.OrbitColor = Red;
30 GMAT COMMSAT1.TargetColor = Teal;
31 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 ];
32 GMAT COMMSAT1.CdSigma = 1e+70;
33 GMAT COMMSAT1.CrSigma = 1e+70;
34 GMAT COMMSAT1.Id = 'SatId';
35 GMAT COMMSAT1.Attitude = CoordinateSystemFixed;
36 GMAT COMMSAT1.SPADSRPInterpolationMethod = Bilinear;
37 GMAT COMMSAT1.SPADSRPScaleFactorSigma = 1e+70;
38 GMAT COMMSAT1.SPADDragInterpolationMethod = Bilinear;
39 GMAT COMMSAT1.SPADDragScaleFactorSigma = 1e+70;
40 GMAT COMMSAT1.ModelFile = 'aura.3ds';
41 GMAT COMMSAT1.ModelOffsetX = 0;
42 GMAT COMMSAT1.ModelOffsetY = 0;
43 GMAT COMMSAT1.ModelOffsetZ = 0;
44 GMAT COMMSAT1.ModelRotationX = 0;
45 GMAT COMMSAT1.ModelRotationY = 0;
46 GMAT COMMSAT1.ModelRotationZ = 0;

```

```

47 GMAT COMMSAT1.ModelScale = 1;
48 GMAT COMMSAT1.AttitudeDisplayStateType = 'Quaternion';
49 GMAT COMMSAT1.AttitudeRateDisplayStateType = 'AngularVelocity';
50 GMAT COMMSAT1.AttitudeCoordinateSystem = EarthMJ2000Eq;
51 GMAT COMMSAT1.EulerAngleSequence = '321';
52
53 %-----
54 %----- Spacecraft
55 %-----
56
57 Create Spacecraft COMMSAT2;
58 GMAT COMMSAT2.DateFormat = TAIModJulian;
59 GMAT COMMSAT2.Epoch = '21545';
60 GMAT COMMSAT2.CoordinateSystem = EarthMJ2000Eq;
61 GMAT COMMSAT2.DisplayStateType = Keplerian;
62 GMAT COMMSAT2.SMA = 42164.999999999999;
63 GMAT COMMSAT2.ECC = 1.192684599010012e-15;
64 GMAT COMMSAT2.INC = 30;
65 GMAT COMMSAT2.RAAN = 150;
66 GMAT COMMSAT2.AOP = 0;
67 GMAT COMMSAT2.TA = 239.9999999999998;
68 GMAT COMMSAT2.DryMass = 850;
69 GMAT COMMSAT2.Cd = 2.2;
70 GMAT COMMSAT2.Cr = 1.8;
71 GMAT COMMSAT2.DragArea = 15;
72 GMAT COMMSAT2.SRPArea = 1;
73 GMAT COMMSAT2.SPADDragScaleFactor = 1;
74 GMAT COMMSAT2.SPADSRPScaleFactor = 1;
75 GMAT COMMSAT2.NAIFid = -10001001;
76 GMAT COMMSAT2.NAIFidReferenceFrame = -9001001;
77 GMAT COMMSAT2.OrbitColor = Red;
78 GMAT COMMSAT2.TargetColor = Teal;
79 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 ];
80 GMAT COMMSAT2.CdSigma = 1e+70;
81 GMAT COMMSAT2.CrSigma = 1e+70;
82 GMAT COMMSAT2.Id = 'SatId';
83 GMAT COMMSAT2.Attitude = CoordinateSystemFixed;
84 GMAT COMMSAT2.SPADSRPInterpolationMethod = Bilinear;
85 GMAT COMMSAT2.SPADSRPScaleFactorSigma = 1e+70;
86 GMAT COMMSAT2.SPADDragInterpolationMethod = Bilinear;
87 GMAT COMMSAT2.SPADDragScaleFactorSigma = 1e+70;
88 GMAT COMMSAT2.ModelFile = 'aura.3ds';
89 GMAT COMMSAT2.ModelOffsetX = 0;
90 GMAT COMMSAT2.ModelOffsetY = 0;
91 GMAT COMMSAT2.ModelOffsetZ = 0;
92 GMAT COMMSAT2.ModelRotationX = 0;
93 GMAT COMMSAT2.ModelRotationY = 0;
94 GMAT COMMSAT2.ModelRotationZ = 0;
95 GMAT COMMSAT2.ModelScale = 1;
96 GMAT COMMSAT2.AttitudeDisplayStateType = 'Quaternion';
97 GMAT COMMSAT2.AttitudeRateDisplayStateType = 'AngularVelocity';
98 GMAT COMMSAT2.AttitudeCoordinateSystem = EarthMJ2000Eq;
99 GMAT COMMSAT2.EulerAngleSequence = '321';

```

```

100
101 %-----
102 %----- Spacecraft
103 %-----
104
105 Create Spacecraft COMMSAT3;
106 GMAT COMMSAT3.DateFormat = TAIModJulian;
107 GMAT COMMSAT3.Epoch = '21545';
108 GMAT COMMSAT3.CoordinateSystem = EarthMJ2000Eq;
109 GMAT COMMSAT3.DisplayStateType = Keplerian;
110 GMAT COMMSAT3.SMA = 42164.999999999999;
111 GMAT COMMSAT3.ECC = 1.293181873398937e-15;
112 GMAT COMMSAT3.INC = 29.99999999999997;
113 GMAT COMMSAT3.RAAN = 270;
114 GMAT COMMSAT3.AOP = 0;
115 GMAT COMMSAT3.TA = 120;
116 GMAT COMMSAT3.DryMass = 850;
117 GMAT COMMSAT3.Cd = 2.2;
118 GMAT COMMSAT3.Cr = 1.8;
119 GMAT COMMSAT3.DragArea = 15;
120 GMAT COMMSAT3.SRPArea = 1;
121 GMAT COMMSAT3.SPADDragScaleFactor = 1;
122 GMAT COMMSAT3.SPADSRPScaleFactor = 1;
123 GMAT COMMSAT3.NAIFid = -10001001;
124 GMAT COMMSAT3.NAIFidReferenceFrame = -9001001;
125 GMAT COMMSAT3.OrbitColor = Red;
126 GMAT COMMSAT3.TargetColor = Teal;
127 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 ];
128 GMAT COMMSAT3.CdSigma = 1e+70;
129 GMAT COMMSAT3.CrSigma = 1e+70;
130 GMAT COMMSAT3.Id = 'SatId';
131 GMAT COMMSAT3.Attitude = CoordinateSystemFixed;
132 GMAT COMMSAT3.SPADSRPInterpolationMethod = Bilinear;
133 GMAT COMMSAT3.SPADSRPScaleFactorSigma = 1e+70;
134 GMAT COMMSAT3.SPADDragInterpolationMethod = Bilinear;
135 GMAT COMMSAT3.SPADDragScaleFactorSigma = 1e+70;
136 GMAT COMMSAT3.ModelFile = 'aura.3ds';
137 GMAT COMMSAT3.ModelOffsetX = 0;
138 GMAT COMMSAT3.ModelOffsetY = 0;
139 GMAT COMMSAT3.ModelOffsetZ = 0;
140 GMAT COMMSAT3.ModelRotationX = 0;
141 GMAT COMMSAT3.ModelRotationY = 0;
142 GMAT COMMSAT3.ModelRotationZ = 0;
143 GMAT COMMSAT3.ModelScale = 1;
144 GMAT COMMSAT3.AttitudeDisplayStateType = 'Quaternion';
145 GMAT COMMSAT3.AttitudeRateDisplayStateType = 'AngularVelocity';
146 GMAT COMMSAT3.AttitudeCoordinateSystem = EarthMJ2000Eq;
147 GMAT COMMSAT3.EulerAngleSequence = '321';
148
149 %-----
150 %----- GroundStations
151 %-----
152

```

```
153 Create GroundStation AshmoreIslands;
154 GMAT AshmoreIslands.OrbitColor = Thistle;
155 GMAT AshmoreIslands.TargetColor = [252 102 101];
156 GMAT AshmoreIslands.CentralBody = Earth;
157 GMAT AshmoreIslands.StateType = Spherical;
158 GMAT AshmoreIslands.HorizonReference = Sphere;
159 GMAT AshmoreIslands.Location1 = -12.183;
160 GMAT AshmoreIslands.Location2 = 122.983;
161 GMAT AshmoreIslands.Location3 = 0;
162 GMAT AshmoreIslands.Id = 'Ashmore';
163 GMAT AshmoreIslands.IonosphereModel = 'None';
164 GMAT AshmoreIslands.TroposphereModel = 'None';
165 GMAT AshmoreIslands.DataSource = 'Constant';
166 GMAT AshmoreIslands.Temperature = 295.1;
167 GMAT AshmoreIslands.Pressure = 1013.5;
168 GMAT AshmoreIslands.Humidity = 55;
169 GMAT AshmoreIslands.MinimumElevationAngle = 20;
170
171 Create GroundStation CartierIsland;
172 GMAT CartierIsland.OrbitColor = Thistle;
173 GMAT CartierIsland.TargetColor = [253 204 101];
174 GMAT CartierIsland.CentralBody = Earth;
175 GMAT CartierIsland.StateType = Spherical;
176 GMAT CartierIsland.HorizonReference = Sphere;
177 GMAT CartierIsland.Location1 = -12.51;
178 GMAT CartierIsland.Location2 = 123.55;
179 GMAT CartierIsland.Location3 = 0;
180 GMAT CartierIsland.Id = 'Cartier';
181 GMAT CartierIsland.IonosphereModel = 'None';
182 GMAT CartierIsland.TroposphereModel = 'None';
183 GMAT CartierIsland.DataSource = 'Constant';
184 GMAT CartierIsland.Temperature = 295.1;
185 GMAT CartierIsland.Pressure = 1013.5;
186 GMAT CartierIsland.Humidity = 55;
187 GMAT CartierIsland.MinimumElevationAngle = 20;
188
189 Create GroundStation CaseyStation;
190 GMAT CaseyStation.OrbitColor = Thistle;
191 GMAT CaseyStation.TargetColor = [254 255 102];
192 GMAT CaseyStation.CentralBody = Earth;
193 GMAT CaseyStation.StateType = Spherical;
194 GMAT CaseyStation.HorizonReference = Sphere;
195 GMAT CaseyStation.Location1 = -66.28;
196 GMAT CaseyStation.Location2 = 110.53;
197 GMAT CaseyStation.Location3 = 0;
198 GMAT CaseyStation.Id = 'StationId';
199 GMAT CaseyStation.IonosphereModel = 'None';
200 GMAT CaseyStation.TroposphereModel = 'None';
201 GMAT CaseyStation.DataSource = 'Constant';
202 GMAT CaseyStation.Temperature = 295.1;
203 GMAT CaseyStation.Pressure = 1013.5;
204 GMAT CaseyStation.Humidity = 55;
205 GMAT CaseyStation.MinimumElevationAngle = 20;
206
207 Create GroundStation DavisStation;
```

```
208 GMAT DavisStation.OrbitColor = Thistle;
209 GMAT DavisStation.TargetColor = [203 255 102];
210 GMAT DavisStation.CentralBody = Earth;
211 GMAT DavisStation.StateType = Spherical;
212 GMAT DavisStation.HorizonReference = Sphere;
213 GMAT DavisStation.Location1 = -68.583;
214 GMAT DavisStation.Location2 = 77.967;
215 GMAT DavisStation.Location3 = 0;
216 GMAT DavisStation.Id = 'StationId';
217 GMAT DavisStation.IonosphereModel = 'None';
218 GMAT DavisStation.TroposphereModel = 'None';
219 GMAT DavisStation.DataSource = 'Constant';
220 GMAT DavisStation.Temperature = 295.1;
221 GMAT DavisStation.Pressure = 1013.5;
222 GMAT DavisStation.Humidity = 55;
223 GMAT DavisStation.MinimumElevationAngle = 20;
224
225 Create GroundStation MawsonStation;
226 GMAT MawsonStation.OrbitColor = Thistle;
227 GMAT MawsonStation.TargetColor = [101 255 102];
228 GMAT MawsonStation.CentralBody = Earth;
229 GMAT MawsonStation.StateType = Spherical;
230 GMAT MawsonStation.HorizonReference = Sphere;
231 GMAT MawsonStation.Location1 = -67.59999999999999;
232 GMAT MawsonStation.Location2 = 62.867;
233 GMAT MawsonStation.Location3 = 0;
234 GMAT MawsonStation.Id = 'StationId';
235 GMAT MawsonStation.IonosphereModel = 'None';
236 GMAT MawsonStation.TroposphereModel = 'None';
237 GMAT MawsonStation.DataSource = 'Constant';
238 GMAT MawsonStation.Temperature = 295.1;
239 GMAT MawsonStation.Pressure = 1013.5;
240 GMAT MawsonStation.Humidity = 55;
241 GMAT MawsonStation.MinimumElevationAngle = 20;
242
243 Create GroundStation MacquarieStation;
244 GMAT MacquarieStation.OrbitColor = Thistle;
245 GMAT MacquarieStation.TargetColor = [101 255 204];
246 GMAT MacquarieStation.CentralBody = Earth;
247 GMAT MacquarieStation.StateType = Spherical;
248 GMAT MacquarieStation.HorizonReference = Sphere;
249 GMAT MacquarieStation.Location1 = -54.5;
250 GMAT MacquarieStation.Location2 = 158.95;
251 GMAT MacquarieStation.Location3 = 0;
252 GMAT MacquarieStation.Id = 'StationId';
253 GMAT MacquarieStation.IonosphereModel = 'None';
254 GMAT MacquarieStation.TroposphereModel = 'None';
255 GMAT MacquarieStation.DataSource = 'Constant';
256 GMAT MacquarieStation.Temperature = 295.1;
257 GMAT MacquarieStation.Pressure = 1013.5;
258 GMAT MacquarieStation.Humidity = 55;
259 GMAT MacquarieStation.MinimumElevationAngle = 20;
260
261 Create GroundStation ChristmasIsland;
262 GMAT ChristmasIsland.OrbitColor = Thistle;
```

```
263 GMAT ChristmasIsland.TargetColor = [101 255 254];
264 GMAT ChristmasIsland.CentralBody = Earth;
265 GMAT ChristmasIsland.StateType = Spherical;
266 GMAT ChristmasIsland.HorizonReference = Sphere;
267 GMAT ChristmasIsland.Location1 = -10.41;
268 GMAT ChristmasIsland.Location2 = 105.716;
269 GMAT ChristmasIsland.Location3 = 0;
270 GMAT ChristmasIsland.Id = 'ChristmasIsland';
271 GMAT ChristmasIsland.IonosphereModel = 'None';
272 GMAT ChristmasIsland.TroposphereModel = 'None';
273 GMAT ChristmasIsland.DataSource = 'Constant';
274 GMAT ChristmasIsland.Temperature = 295.1;
275 GMAT ChristmasIsland.Pressure = 1013.5;
276 GMAT ChristmasIsland.Humidity = 55;
277 GMAT ChristmasIsland.MinimumElevationAngle = 20;
278
279 Create GroundStation CocosIslands;
280 GMAT CocosIslands.OrbitColor = Thistle;
281 GMAT CocosIslands.TargetColor = [101 204 254];
282 GMAT CocosIslands.CentralBody = Earth;
283 GMAT CocosIslands.StateType = Spherical;
284 GMAT CocosIslands.HorizonReference = Sphere;
285 GMAT CocosIslands.Location1 = -12.167;
286 GMAT CocosIslands.Location2 = 96.833;
287 GMAT CocosIslands.Location3 = 0;
288 GMAT CocosIslands.Id = 'StationId';
289 GMAT CocosIslands.IonosphereModel = 'None';
290 GMAT CocosIslands.TroposphereModel = 'None';
291 GMAT CocosIslands.DataSource = 'Constant';
292 GMAT CocosIslands.Temperature = 295.1;
293 GMAT CocosIslands.Pressure = 1013.5;
294 GMAT CocosIslands.Humidity = 55;
295 GMAT CocosIslands.MinimumElevationAngle = 20;
296
297 Create GroundStation CoralSeaIslands;
298 GMAT CoralSeaIslands.OrbitColor = Thistle;
299 GMAT CoralSeaIslands.TargetColor = [102 102 254];
300 GMAT CoralSeaIslands.CentralBody = Earth;
301 GMAT CoralSeaIslands.StateType = Spherical;
302 GMAT CoralSeaIslands.HorizonReference = Sphere;
303 GMAT CoralSeaIslands.Location1 = -23.25;
304 GMAT CoralSeaIslands.Location2 = 155.533;
305 GMAT CoralSeaIslands.Location3 = 0;
306 GMAT CoralSeaIslands.Id = 'StationId';
307 GMAT CoralSeaIslands.IonosphereModel = 'None';
308 GMAT CoralSeaIslands.TroposphereModel = 'None';
309 GMAT CoralSeaIslands.DataSource = 'Constant';
310 GMAT CoralSeaIslands.Temperature = 295.1;
311 GMAT CoralSeaIslands.Pressure = 1013.5;
312 GMAT CoralSeaIslands.Humidity = 55;
313 GMAT CoralSeaIslands.MinimumElevationAngle = 20;
314
315 Create GroundStation HeardMcDonaldIslands;
316 GMAT HeardMcDonaldIslands.OrbitColor = Thistle;
317 GMAT HeardMcDonaldIslands.TargetColor = [204 102 254];
```

```

318 GMAT HeardMcDonaldIslands.CentralBody = Earth;
319 GMAT HeardMcDonaldIslands.StateType = Spherical;
320 GMAT HeardMcDonaldIslands.HorizonReference = Sphere;
321 GMAT HeardMcDonaldIslands.Location1 = -53.08333;
322 GMAT HeardMcDonaldIslands.Location2 = 73.5;
323 GMAT HeardMcDonaldIslands.Location3 = 0;
324 GMAT HeardMcDonaldIslands.Id = 'StationId';
325 GMAT HeardMcDonaldIslands.IonosphereModel = 'None';
326 GMAT HeardMcDonaldIslands.TroposphereModel = 'None';
327 GMAT HeardMcDonaldIslands.DataSource = 'Constant';
328 GMAT HeardMcDonaldIslands.Temperature = 295.1;
329 GMAT HeardMcDonaldIslands.Pressure = 1013.5;
330 GMAT HeardMcDonaldIslands.Humidity = 55;
331 GMAT HeardMcDonaldIslands.MinimumElevationAngle = 20;
332
333 Create GroundStation NorfolkIsland;
334 GMAT NorfolkIsland.OrbitColor = Thistle;
335 GMAT NorfolkIsland.TargetColor = [252 102 254];
336 GMAT NorfolkIsland.CentralBody = Earth;
337 GMAT NorfolkIsland.StateType = Spherical;
338 GMAT NorfolkIsland.HorizonReference = Sphere;
339 GMAT NorfolkIsland.Location1 = -29.033;
340 GMAT NorfolkIsland.Location2 = 167.95;
341 GMAT NorfolkIsland.Location3 = 0;
342 GMAT NorfolkIsland.Id = 'StationId';
343 GMAT NorfolkIsland.IonosphereModel = 'None';
344 GMAT NorfolkIsland.TroposphereModel = 'None';
345 GMAT NorfolkIsland.DataSource = 'Constant';
346 GMAT NorfolkIsland.Temperature = 295.1;
347 GMAT NorfolkIsland.Pressure = 1013.5;
348 GMAT NorfolkIsland.Humidity = 55;
349 GMAT NorfolkIsland.MinimumElevationAngle = 20;
350
351
352
353
354
355
356
357
358
359
360
361
362
363 %-----
364 %----- ForceModels
365 %-----
366
367 Create ForceModel DefaultProp_ForceModel;
368 GMAT DefaultProp_ForceModel.CentralBody = Earth;
369 GMAT DefaultProp_ForceModel.PrimaryBodies = {Earth};
370 GMAT DefaultProp_ForceModel.Drag = None;
371 GMAT DefaultProp_ForceModel.SRP = Off;
372 GMAT DefaultProp_ForceModel.RelativisticCorrection = Off;

```



```

373 GMAT DefaultProp_ForceModel.ErrorControl = RSSStep;
374 GMAT DefaultProp_ForceModel.GravityField.Earth.Degree = 4;
375 GMAT DefaultProp_ForceModel.GravityField.Earth.Order = 4;
376 GMAT DefaultProp_ForceModel.GravityField.Earth.StmLimit = 100;
377 GMAT DefaultProp_ForceModel.GravityField.Earth.PotentialFile = 'JGM2.cof
    ↪ ';
378 GMAT DefaultProp_ForceModel.GravityField.Earth.TideModel = 'None';
379
380 %-----
381 %----- Propagators
382 %-----
383
384 Create Propagator DefaultProp;
385 GMAT DefaultProp.FM = DefaultProp_ForceModel;
386 GMAT DefaultProp.Type = RungeKutta89;
387 GMAT DefaultProp.InitialStepSize = 60;
388 GMAT DefaultProp.Accuracy = 9.999999999999999e-12;
389 GMAT DefaultProp.MinStep = 0.001;
390 GMAT DefaultProp.MaxStep = 2700;
391 GMAT DefaultProp.MaxStepAttempts = 50;
392 GMAT DefaultProp.StopIfAccuracyIsViolated = true;
393
394 %-----
395 %----- EventLocators
396 %-----
397
398 Create ContactLocator ContactLocator1;
399 GMAT ContactLocator1.Target = COMMSAT1;
400 GMAT ContactLocator1.Filename = 'ContactLocator1.txt';
401 GMAT ContactLocator1.InputEpochFormat = 'TAIModJulian';
402 GMAT ContactLocator1.InitialEpoch = '21545';
403 GMAT ContactLocator1.StepSize = 10;
404 GMAT ContactLocator1.FinalEpoch = '21545.138';
405 GMAT ContactLocator1.UseLightTimeDelay = true;
406 GMAT ContactLocator1.UseStellarAberration = true;
407 GMAT ContactLocator1.WriteReport = true;
408 GMAT ContactLocator1.RunMode = Automatic;
409 GMAT ContactLocator1.UseEntireInterval = true;
410 GMAT ContactLocator1.Observers = {AshmoreIslands, CartierIsland,
    ↪ CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands,
    ↪ DavisStation, HeardMcDonaldIslands, MacquarieStation,
    ↪ MawsonStation, NorfolkIsland};
411 GMAT ContactLocator1.LightTimeDirection = Transmit;
412
413 Create ContactLocator ContactLocator2;
414 GMAT ContactLocator2.Target = COMMSAT2;
415 GMAT ContactLocator2.Filename = 'ContactLocator2.txt';
416 GMAT ContactLocator2.InputEpochFormat = 'TAIModJulian';
417 GMAT ContactLocator2.InitialEpoch = '21545';
418 GMAT ContactLocator2.StepSize = 10;
419 GMAT ContactLocator2.FinalEpoch = '21545.138';
420 GMAT ContactLocator2.UseLightTimeDelay = true;
421 GMAT ContactLocator2.UseStellarAberration = true;
422 GMAT ContactLocator2.WriteReport = true;
423 GMAT ContactLocator2.RunMode = Automatic;

```

```

424 GMAT ContactLocator2.UseEntireInterval = true;
425 GMAT ContactLocator2.Observers = {AshmoreIslands, CartierIsland,
    ↪ CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands,
    ↪ DavisStation, HeardMcDonaldIslands, MacquarieStation,
    ↪ MawsonStation, NorfolkIsland};
426 GMAT ContactLocator2.LightTimeDirection = Transmit;
427
428 Create ContactLocator ContactLocator3;
429 GMAT ContactLocator3.Target = COMMSAT3;
430 GMAT ContactLocator3.Filename = 'ContactLocator3.txt';
431 GMAT ContactLocator3.InputEpochFormat = 'TAIModJulian';
432 GMAT ContactLocator3.InitialEpoch = '21545';
433 GMAT ContactLocator3.StepSize = 10;
434 GMAT ContactLocator3.FinalEpoch = '21545.138';
435 GMAT ContactLocator3.UseLightTimeDelay = true;
436 GMAT ContactLocator3.UseStellarAberration = true;
437 GMAT ContactLocator3.WriteReport = true;
438 GMAT ContactLocator3.RunMode = Automatic;
439 GMAT ContactLocator3.UseEntireInterval = true;
440 GMAT ContactLocator3.Observers = {AshmoreIslands, CartierIsland,
    ↪ CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands,
    ↪ DavisStation, HeardMcDonaldIslands, MacquarieStation,
    ↪ MawsonStation, NorfolkIsland};
441 GMAT ContactLocator3.LightTimeDirection = Transmit;
442
443 Create EclipseLocator EclipseLocator1;
444 GMAT EclipseLocator1.Spacecraft = COMMSAT3;
445 GMAT EclipseLocator1.Filename = 'EclipseLocator1.txt';
446 GMAT EclipseLocator1.OccultingBodies = {Earth, Luna};
447 GMAT EclipseLocator1.InputEpochFormat = 'TAIModJulian';
448 GMAT EclipseLocator1.InitialEpoch = '21545';
449 GMAT EclipseLocator1.StepSize = 10;
450 GMAT EclipseLocator1.FinalEpoch = '21545.138';
451 GMAT EclipseLocator1.UseLightTimeDelay = true;
452 GMAT EclipseLocator1.UseStellarAberration = true;
453 GMAT EclipseLocator1.WriteReport = true;
454 GMAT EclipseLocator1.RunMode = Automatic;
455 GMAT EclipseLocator1.UseEntireInterval = true;
456 GMAT EclipseLocator1.EclipseTypes = {'Umbra', 'Penumbra', 'Antumbra'};
457
458 Create EclipseLocator EclipseLocator2;
459 GMAT EclipseLocator2.Spacecraft = COMMSAT2;
460 GMAT EclipseLocator2.Filename = 'EclipseLocator1.txt';
461 GMAT EclipseLocator2.OccultingBodies = {Earth, Luna};
462 GMAT EclipseLocator2.InputEpochFormat = 'TAIModJulian';
463 GMAT EclipseLocator2.InitialEpoch = '21545';
464 GMAT EclipseLocator2.StepSize = 10;
465 GMAT EclipseLocator2.FinalEpoch = '21545.138';
466 GMAT EclipseLocator2.UseLightTimeDelay = true;
467 GMAT EclipseLocator2.UseStellarAberration = true;
468 GMAT EclipseLocator2.WriteReport = true;
469 GMAT EclipseLocator2.RunMode = Automatic;
470 GMAT EclipseLocator2.UseEntireInterval = true;
471 GMAT EclipseLocator2.EclipseTypes = {'Umbra', 'Penumbra', 'Antumbra'};
472

```

```

473 Create EclipseLocator EclipseLocator3;
474 GMAT EclipseLocator3.Spacecraft = COMMSAT2;
475 GMAT EclipseLocator3.Filename = 'EclipseLocator1.txt';
476 GMAT EclipseLocator3.OccultingBodies = {Earth, Luna};
477 GMAT EclipseLocator3.InputEpochFormat = 'TAIModJulian';
478 GMAT EclipseLocator3.InitialEpoch = '21545';
479 GMAT EclipseLocator3.StepSize = 10;
480 GMAT EclipseLocator3.FinalEpoch = '21545.138';
481 GMAT EclipseLocator3.UseLightTimeDelay = true;
482 GMAT EclipseLocator3.UseStellarAberration = true;
483 GMAT EclipseLocator3.WriteReport = true;
484 GMAT EclipseLocator3.RunMode = Automatic;
485 GMAT EclipseLocator3.UseEntireInterval = true;
486 GMAT EclipseLocator3.EclipseTypes = {'Umbra', 'Penumbra', 'Antumbra'};
487
488 %-----
489 %----- Subscribers
490 %-----
491
492 Create OrbitView EarthFixedEcliptic;
493 GMAT EarthFixedEcliptic.SolverIterations = None;
494 GMAT EarthFixedEcliptic.UpperLeft = [ 0.09761904761904762
    ↪ 0.08190476190476191 ];
495 GMAT EarthFixedEcliptic.Size = [ 0.6976190476190476 0.7342857142857143
    ↪ ];
496 GMAT EarthFixedEcliptic.RelativeZOrder = 388;
497 GMAT EarthFixedEcliptic.Maximized = false;
498 GMAT EarthFixedEcliptic.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
499 GMAT EarthFixedEcliptic.CoordinateSystem = EarthFixed;
500 GMAT EarthFixedEcliptic.DrawObject = [ true true true true ];
501 GMAT EarthFixedEcliptic.DataCollectFrequency = 1;
502 GMAT EarthFixedEcliptic.UpdatePlotFrequency = 50;
503 GMAT EarthFixedEcliptic.NumPointsToRedraw = 0;
504 GMAT EarthFixedEcliptic.ShowPlot = true;
505 GMAT EarthFixedEcliptic.MaxPlotPoints = 20000;
506 GMAT EarthFixedEcliptic.ShowLabels = true;
507 GMAT EarthFixedEcliptic.ViewPointReference = Earth;
508 GMAT EarthFixedEcliptic.ViewPointVector = COMMSAT1;
509 GMAT EarthFixedEcliptic.ViewDirection = Earth;
510 GMAT EarthFixedEcliptic.ViewScaleFactor = 2.5;
511 GMAT EarthFixedEcliptic.ViewUpCoordinateSystem = EarthMJ2000Eq;
512 GMAT EarthFixedEcliptic.ViewUpAxis = Z;
513 GMAT EarthFixedEcliptic.EclipticPlane = On;
514 GMAT EarthFixedEcliptic.XYPlane = Off;
515 GMAT EarthFixedEcliptic.WireFrame = Off;
516 GMAT EarthFixedEcliptic.Axes = On;
517 GMAT EarthFixedEcliptic.Grid = Off;
518 GMAT EarthFixedEcliptic.SunLine = Off;
519 GMAT EarthFixedEcliptic.UseInitialView = On;
520 GMAT EarthFixedEcliptic.StarCount = 7000;
521 GMAT EarthFixedEcliptic.EnableStars = On;
522 GMAT EarthFixedEcliptic.EnableConstellations = On;
523
524 Create GroundTrackPlot DefaultGroundTrackPlot;
525 GMAT DefaultGroundTrackPlot.SolverIterations = Current;

```

```

526 GMAT DefaultGroundTrackPlot.UpperLeft = [ 0.05773809523809524
      ↪ 0.03142857142857143 ];
527 GMAT DefaultGroundTrackPlot.Size = [ 0.9898809523809524
      ↪ 0.9533333333333334 ];
528 GMAT DefaultGroundTrackPlot.RelativeZOrder = 408;
529 GMAT DefaultGroundTrackPlot.Maximized = false;
530 GMAT DefaultGroundTrackPlot.Add = {AshmoreIslands, CartierIsland,
      ↪ CaseyStation, ChristmasIsland, CocosIslands, CoralSeaIslands,
      ↪ DavisStation, COMMSAT1, COMMSAT2, COMMSAT3, HeardMcDonaldIslands,
      ↪ MacquarieStation, MawsonStation, NorfolkIsland};
531 GMAT DefaultGroundTrackPlot.DataCollectFrequency = 1;
532 GMAT DefaultGroundTrackPlot.UpdatePlotFrequency = 50;
533 GMAT DefaultGroundTrackPlot.NumPointsToRedraw = 0;
534 GMAT DefaultGroundTrackPlot.ShowPlot = true;
535 GMAT DefaultGroundTrackPlot.MaxPlotPoints = 20000;
536 GMAT DefaultGroundTrackPlot.CentralBody = Earth;
537 GMAT DefaultGroundTrackPlot.TextureMap = 'ModifiedBlueMarble.jpg';
538
539 Create ReportFile ReportFile1;
540 GMAT ReportFile1.SolverIterations = Current;
541 GMAT ReportFile1.UpperLeft = [ 0.1458333333333333 0.03656998738965952 ];
542 GMAT ReportFile1.Size = [ 0.9940476190476191 0.9609079445145019 ];
543 GMAT ReportFile1.RelativeZOrder = 121;
544 GMAT ReportFile1.Maximized = true;
545 GMAT ReportFile1.Filename = 'OrbitParams.txt';
546 GMAT ReportFile1.Precision = 16;
547 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};
548 GMAT ReportFile1.WriteHeaders = true;
549 GMAT ReportFile1.LeftJustify = On;
550 GMAT ReportFile1.ZeroFill = Off;
551 GMAT ReportFile1.FixedWidth = true;
552 GMAT ReportFile1.Delimiter = ' ';
553 GMAT ReportFile1.ColumnWidth = 23;
554 GMAT ReportFile1.WriteReport = true;
555
556 Create OrbitView J2000Ecliptic;
557 GMAT J2000Ecliptic.SolverIterations = None;
558 GMAT J2000Ecliptic.UpperLeft = [ 0.1035714285714286 0.04571428571428571
      ↪ ];
559 GMAT J2000Ecliptic.Size = [ 0.8101190476190476 0.8504761904761905 ];
560 GMAT J2000Ecliptic.RelativeZOrder = 406;
561 GMAT J2000Ecliptic.Maximized = false;
562 GMAT J2000Ecliptic.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
563 GMAT J2000Ecliptic.CoordinateSystem = EarthMJ2000Eq;
564 GMAT J2000Ecliptic.DrawObject = [ true true true true ];
565 GMAT J2000Ecliptic.DataCollectFrequency = 1;
566 GMAT J2000Ecliptic.UpdatePlotFrequency = 50;
567 GMAT J2000Ecliptic.NumPointsToRedraw = 0;
568 GMAT J2000Ecliptic.ShowPlot = true;
569 GMAT J2000Ecliptic.MaxPlotPoints = 20000;

```

```

570 GMAT J2000Ecliptic.ShowLabels = true;
571 GMAT J2000Ecliptic.ViewPointReference = Earth;
572 GMAT J2000Ecliptic.ViewPointVector = COMMSAT1;
573 GMAT J2000Ecliptic.ViewDirection = Earth;
574 GMAT J2000Ecliptic.ViewScaleFactor = 2.5;
575 GMAT J2000Ecliptic.ViewUpCoordinateSystem = EarthMJ2000Eq;
576 GMAT J2000Ecliptic.ViewUpAxis = Z;
577 GMAT J2000Ecliptic.EclipticPlane = On;
578 GMAT J2000Ecliptic.XYPlane = Off;
579 GMAT J2000Ecliptic.WireFrame = Off;
580 GMAT J2000Ecliptic.Axes = On;
581 GMAT J2000Ecliptic.Grid = Off;
582 GMAT J2000Ecliptic.SunLine = Off;
583 GMAT J2000Ecliptic.UseInitialView = On;
584 GMAT J2000Ecliptic.StarCount = 7000;
585 GMAT J2000Ecliptic.EnableStars = On;
586 GMAT J2000Ecliptic.EnableConstellations = On;
587
588 %-----
589 %----- Subscribers
590 %-----
591
592 Create OrbitView EarthFixedXY;
593 GMAT EarthFixedXY.SolverIterations = None;
594 GMAT EarthFixedXY.UpperLeft = [ 0.09345238095238095 0.03619047619047619
    ↪ ];
595 GMAT EarthFixedXY.Size = [ 0.7738095238095238 0.820952380952381 ];
596 GMAT EarthFixedXY.RelativeZOrder = 400;
597 GMAT EarthFixedXY.Maximized = false;
598 GMAT EarthFixedXY.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
599 GMAT EarthFixedXY.CoordinateSystem = EarthFixed;
600 GMAT EarthFixedXY.DrawObject = [ true true true true ];
601 GMAT EarthFixedXY.DataCollectFrequency = 1;
602 GMAT EarthFixedXY.UpdatePlotFrequency = 50;
603 GMAT EarthFixedXY.NumPointsToRedraw = 0;
604 GMAT EarthFixedXY.ShowPlot = true;
605 GMAT EarthFixedXY.MaxPlotPoints = 20000;
606 GMAT EarthFixedXY.ShowLabels = true;
607 GMAT EarthFixedXY.ViewPointReference = Earth;
608 GMAT EarthFixedXY.ViewPointVector = COMMSAT1;
609 GMAT EarthFixedXY.ViewDirection = Earth;
610 GMAT EarthFixedXY.ViewScaleFactor = 2.5;
611 GMAT EarthFixedXY.ViewUpCoordinateSystem = EarthMJ2000Eq;
612 GMAT EarthFixedXY.ViewUpAxis = Z;
613 GMAT EarthFixedXY.EclipticPlane = Off;
614 GMAT EarthFixedXY.XYPlane = On;
615 GMAT EarthFixedXY.WireFrame = Off;
616 GMAT EarthFixedXY.Axes = On;
617 GMAT EarthFixedXY.Grid = Off;
618 GMAT EarthFixedXY.SunLine = Off;
619 GMAT EarthFixedXY.UseInitialView = On;
620 GMAT EarthFixedXY.StarCount = 7000;
621 GMAT EarthFixedXY.EnableStars = On;
622 GMAT EarthFixedXY.EnableConstellations = On;
623

```

```

624 Create OrbitView J2000XY;
625 GMAT J2000XY.SolverIterations = None;
626 GMAT J2000XY.UpperLeft = [ 0.1571428571428571 0.08476190476190476 ];
627 GMAT J2000XY.Size = [ 0.7053571428571429 0.7495238095238095 ];
628 GMAT J2000XY.RelativeZOrder = 394;
629 GMAT J2000XY.Maximized = false;
630 GMAT J2000XY.Add = {COMMSAT1, COMMSAT2, COMMSAT3, Earth};
631 GMAT J2000XY.CoordinateSystem = EarthMJ2000Eq;
632 GMAT J2000XY.DrawObject = [ true true true true ];
633 GMAT J2000XY.DataCollectFrequency = 1;
634 GMAT J2000XY.UpdatePlotFrequency = 50;
635 GMAT J2000XY.NumPointsToRedraw = 0;
636 GMAT J2000XY.ShowPlot = true;
637 GMAT J2000XY.MaxPlotPoints = 20000;
638 GMAT J2000XY.ShowLabels = true;
639 GMAT J2000XY.ViewPointReference = Earth;
640 GMAT J2000XY.ViewPointVector = COMMSAT1;
641 GMAT J2000XY.ViewDirection = Earth;
642 GMAT J2000XY.ViewScaleFactor = 2.5;
643 GMAT J2000XY.ViewUpCoordinateSystem = EarthMJ2000Eq;
644 GMAT J2000XY.ViewUpAxis = Z;
645 GMAT J2000XY.EclipticPlane = Off;
646 GMAT J2000XY.XYPlane = On;
647 GMAT J2000XY.WireFrame = Off;
648 GMAT J2000XY.Axes = On;
649 GMAT J2000XY.Grid = Off;
650 GMAT J2000XY.SunLine = Off;
651 GMAT J2000XY.UseInitialView = On;
652 GMAT J2000XY.StarCount = 7000;
653 GMAT J2000XY.EnableStars = On;
654 GMAT J2000XY.EnableConstellations = On;
655
656 %-----
657 %----- Functions -----
658 %-----
659
660 Create GmatFunction CalcAzElRange;
661 GMAT CalcAzElRange.FunctionPath = '/Users/ninaaverill/unispace/2022/
    ↪ zeit8219-Satellite-Communications/assmt1/CalcAzElRange.gmf';
662
663 %-----
664 %----- Arrays, Variables, Strings -----
665 %-----
666 Create Variable EarthRad;
667 GMAT EarthRad = 6371;
668
669
670
671 %-----
672 %----- Mission Sequence -----
673 %-----
674
675 BeginMissionSequence;
676 While COMMSAT1.ElapsedSecs < 86400
677     Propagate DefaultProp(COMMSAT1) DefaultProp(COMMSAT2) DefaultProp(

```



```

678   ↪ 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;
679 EndWhile;

```

## 5.2 Python Code Samples

```

1 import numpy as np
2 import pandas as pd
3 import plotly.express as px
4
5 from link_calculator.components import Antenna, GroundStation, Satellite
6 from link_calculator.orbits.utils import (
7     angle_sat_to_ground_station,
8     elevation_angle,
9     slant_range,
10 )
11 from link_calculator.propagation.conversions import (
12     decibel_to_watt,
13     frequency_to_wavelength,
14     watt_to_decibel,
15 )
16 from link_calculator.propagation.utils import receive_power
17
18
19 def load_gmat_report(file_path):
20     return pd.read_csv(file_path, sep="\s{2,}", engine="python")
21
22
23 def q2(
24     report: pd.DataFrame, satellites: list[str], ground_stations: list[
25     ↪ GroundStation]
26 ):
27     for sat in satellites:
28         for gs in ground_stations:
29             report[f"{sat.name}.Earth.Gamma.{gs.name}"] = report.apply(
30                 lambda row: angle_sat_to_ground_station(
31                     gs.latitude,
32                     gs.longitude,
33                     row[f"{sat.name}.Earth.Latitude"],
34                     row[f"{sat.name}.Earth.Longitude"],
35                 ),
36                 axis=1,
37             )
38             report[f"{sat.name}.Earth.SlantRange.{gs.name}"] = report.
39             ↪ apply(
40                 lambda row: slant_range(
41                     row[f"{sat.name}.Earth.RMAG"],
42                     row[f"{sat.name}.Earth.Gamma.{gs.name}"],

```

```

41         ),
42         axis=1,
43     )
44     report[f"{sat.name}.Earth.Elevation.{gs.name}"] = report.
↪ apply(
45         lambda row: elevation_angle(
46             row[f"{sat.name}.Earth.RMAG"],
47             row[f"{sat.name}.Earth.Gamma.{gs.name}"],
48         ),
49         axis=1,
50     )
51
52     min_elevation = 20
53     for gs in ground_stations:
54         assert all(
55             np.logical_or.reduce(
56                 [
57                     report[f"{sat.name}.Earth.Elevation.{gs.name}"] >
↪ min_elevation
58                     for sat in satellites
59                 ]
60             )
61         )
62
63     plot_elevation(report, satellites, ground_stations)
64     return report
65
66
67 def q3(
68     report: pd.DataFrame,
69     satellites: list[str],
70     ground_stations: list[GroundStation],
71 ):
72     for gs in ground_stations:
73         for sat in satellites:
74             report[f"{sat.name}.Earth.ReceivePower.{gs.name}"] = report.
↪ apply(
75                 lambda row: watt_to_decibel(
76                     receive_power(
77                         sat.antenna.power,
78                         sat.antenna.loss,
79                         sat.antenna.gain,
80                         row[f"{sat.name}.Earth.SlantRange.{gs.name}"] *
↪ 1000,
81                         gs.antenna.loss,
82                         gs.antenna.gain,
83                         1,
84                         wavelength=frequency_to_wavelength(sat.antenna.
↪ frequency),
85                     )
86                 )
87                 if row[f"{sat.name}.Earth.Elevation.{gs.name}"] > 20
88                 else None,
89                 axis=1,
90             )

```



```

91     plot_receive_power(report, satellites, ground_stations)
92     return report
93
94
95 def plot_receive_power(report, satellites, ground_stations):
96     for gs in ground_stations:
97         min_slant = report.iloc[
98             report[f"{satellites[0].name}.Earth.SlantRange.{gs.name}"].
99             ↪ idxmin()
100             ]
101
102         fig = px.line(
103             report,
104             x="UTC Gregorian",
105             y=[f"{sat.name}.Earth.ReceivePower.{gs.name}" for sat in
106             ↪ satellites],
107             labels={"value": "Receive Power (dBW)"},
108         )
109         fig.add_vline(x=min_slant["UTC Gregorian"], line_dash="dash")
110         fig.add_annotation(
111             x=min_slant["UTC Gregorian"],
112             y=min_slant[f"{satellites[0].name}.Earth.ReceivePower.{gs.
113             ↪ name}"],
114             text=f"Slant range = {min_slant[f'{satellites[0].name}.Earth
115             ↪ .SlantRange.{gs.name}']:.2f}",
116         )
117         fig.update_layout(
118             title={
119                 "text": f"Power received by {gs.name}",
120                 "x": 0.5,
121                 "xanchor": "center",
122                 "yanchor": "top",
123             }
124         )
125         fig.show()
126         print(gs.name, min_slant[f"{satellites[0].name}.Earth.
127         ↪ ReceivePower.{gs.name}"], "dBW")
128
129
130
131 def plot_elevation(report, satellites, ground_stations):
132     for gs in ground_stations:
133         min_slant = report.iloc[
134             report[f"{satellites[0].name}.Earth.SlantRange.{gs.name}"].
135             ↪ idxmin()
136             ]
137
138         cols = [f"{sat.name}.Earth.Elevation.{gs.name}" for sat in
139         ↪ satellites]
140         elevation = report[cols]
141         elevation = elevation[elevation > 20]
142         elevation["UTC Gregorian"] = report["UTC Gregorian"]
143
144         fig = px.line(
145             elevation,
146             x="UTC Gregorian",

```

```

139         y=cols,
140         labels={"value": "Elevation (deg)"},
141     )
142     fig.add_vline(x=min_slant["UTC Gregorian"], line_dash="dash")
143     fig.add_annotation(
144         x=min_slant["UTC Gregorian"],
145         y=min_slant[f"{satellites[0].name}.Earth.Elevation.{gs.name}
↪ "],
146         text=f"Slant range = {min_slant[f'{satellites[0].name}.Earth
↪ .SlantRange.{gs.name}']:.2f}",
147     )
148     fig.update_layout(
149         title={
150             "text": f"Elevation above {gs.name}",
151             "x": 0.5,
152             "xanchor": "center",
153             "yanchor": "top",
154         }
155     )
156     fig.show()
157     print(gs.name, min_slant[f"{satellites[0].name}.Earth.Altitude"
↪ ], "km")
158
159
160 if __name__ == "__main__":
161     # ka_band = 26.5 40 GHz
162     sat_frequency = 20
163     gs_frequency = 30
164     ground_stations = [
165         GroundStation(
166             "CocosIsland",
167             -12.167,
168             96.833,
169             0,
170             Antenna("CocosReceive", 0, decibel_to_watt(18), 1,
↪ gs_frequency),
171         ),
172         GroundStation(
173             "MawsonResearchStation",
174             -67.6,
175             62.867,
176             0,
177             Antenna("MawsonReceive", 0, decibel_to_watt(18), 1,
↪ gs_frequency),
178         ),
179         GroundStation(
180             "NorfolkIsland",
181             -29.033,
182             167.95,
183             0,
184             Antenna("MawsonReceive", 0, decibel_to_watt(18), 1,
↪ gs_frequency),
185         ),
186         GroundStation(
187             "MacquarieResearchStation",

```

```

188         -54.5,
189         158.95,
190         0,
191         Antenna("MacquarieReceive", 0, decibel_to_watt(18), 1,
↪ gs_frequency),
192     ),
193 ]
194
195     satellites = [
196         Satellite(
197             "COMMSAT1",
198             Antenna("COMMSAT1Transmit", 5, decibel_to_watt(30), 1,
↪ sat_frequency),
199         ),
200         Satellite(
201             "COMMSAT2",
202             Antenna("COMMSAT2Transmit", 5, decibel_to_watt(30), 1,
↪ sat_frequency),
203         ),
204         Satellite(
205             "COMMSAT3",
206             Antenna("COMMSAT3Transmit", 5, decibel_to_watt(30), 1,
↪ sat_frequency),
207         ),
208     ]
209
210     report = load_gmat_report("input/OrbitParams.txt")
211     report["UTC Gregorian"] = pd.to_datetime(report["COMMSAT1.
↪ UTCGregorian"])
212     report = q2(report, satellites, ground_stations)
213     report = q3(report, satellites, ground_stations)
214     report.to_csv("output/report.csv")

```

Listing 1: main.py

```

1 # Orbits
2 EARTH_GRAVITATIONAL_CONSTANT = 6.6743e-11 # Nm^2 / kg^2
3 EARTH_MASS = 5.98e24 # km^3/s^2
4 EARTH_MU = 3.986004418e5 # km^3/s^-2
5 EARTH_RADIUS = 6378.14 # km^3/s^2
6 EARTH_SOLAR_YEAR = 365.25 # days
7 SIDEREAL_DAY = 23.935 # hours
8
9 # Propagation
10 SPEED_OF_LIGHT = 299792458 # m/s

```

Listing 2: constants.py

```

1 from math import atan2, cos, degrees, radians
2
3 import numpy as np
4
5 from link_calculator.constants import EARTH_RADIUS
6
7

```

```

8 def power_density(power: float, gain: float, distance: float) -> float:
9     """
10     Calculate the power density of the wavefront
11
12     Parameters
13     -----
14         power (float, W): the transmitted power
15         gain (float, W): the power gained
16         distance (float, m): the distance between the transmit and
    ↪ receive antennas
17
18     Returns
19     -----
20         power_density (float, W/m^2): the power density at distance d
21     """
22     return (power * gain) / (4 * np.pi * distance**2)
23
24
25 def eirp(power: float, loss: float, gain: float) -> float:
26     """
27     Calculate the Effetive Isotropic Radiated Power
28
29     Parameters
30     -----
31         power (float, W): the total output amplifier power
32         loss (float, ): coupling loss between transmitter and antenna
33             in the range [0, 1]
34         gain (float, ): transmitter gain in the direction of the
35             receiving antenna
36
37     Returns
38     -----
39         eirp (float, dB): power incident at the receiver that would have
    ↪ had to be radiated
40             from an isotropic antenna to achieve the same power incident
    ↪ at the
41             receiver as that of a transmitter with a specific antenna
    ↪ gain
42     """
43     return power * loss * gain
44
45
46 def power_density_eirp(eirp: float, distance: float, atmospheric_loss:
    ↪ float) -> float:
47     """
48     Calculate the power density of the wavefront using EIRP
49
50     Parameters
51     -----
52         eirp (float, dB)
53         distance (float, m): the distance between the transmit and
    ↪ receive antennas
54         atmospheric_loss (float, ): the total losses due to the
    ↪ atmosphere
55

```

```

56     Returns
57     -----
58     power_density (float, W/m^2): the power density at distance d
59     """
60     return eirp / (4 * np.pi * distance**2) * atmospheric_loss
61
62
63 def effective_aperture(gain: float, wavelength: float) -> float:
64     """
65     Calculate the effective area of the receiving antenna
66
67     Parameters
68     -----
69     gain (float, ): gain of the receive antenna
70     wavelength (float, m): the radiation wavelength
71
72     Returns
73     -----
74     effective_aperture (float, m^2): the effive aperture of the
    ↪ receive antenna
75     """
76     return gain * wavelength**2 / (4 * np.pi)
77
78
79 def receive_power(
80     transmit_power: float,
81     transmit_loss: float,
82     transmit_gain: float,
83     distance: float,
84     receive_loss: float,
85     receive_gain: float,
86     atmospheric_loss: float,
87     wavelength: float = None,
88     eff_aperture: float = None
89 ) -> float:
90     """
91     Calculate the power collected by the receive antenna
92
93     Parameters
94     -----
95     amp_power (float, W): the total output amplifier power
96     power_density (float, W/m^2): the power density at distance d
97     transmit_loss (float, ): coupling loss between transmitter and
    ↪ antenna
98         in the range [0, 1]
99     transmit_gain (float, ): transmitter gain in the direction of
    ↪ the
100         receiving antenna
101     distance (float, m): the distance between the transmit and
    ↪ receive antennas
102     receive_gain (float, ): the gain at the recieve antenna
103     receive_loss (float, ): coupling loss between receiver terminals
    ↪ and antenna
104         in the range [0, 1]
105     wavelength (float, m): the radiation wavelength

```

```

106     eff_aperture (float, m): effective aperture of the receive
    ↪ antenna
107     atmospheric_loss (float, ): The loss due to the atmosphere
108
109     Returns
110     -----
111     receive_power (float, W): the total collected power at the
    ↪ receiver's terminals
112     """
113     eirp_ = eirp(transmit_power, transmit_loss, transmit_gain)
114     pow_density = power_density_eirp(eirp_, distance, atmospheric_loss)
115
116     if not eff_aperture and wavelength:
117         eff_aperture = effective_aperture(receive_gain, wavelength)
118     if not (eff_aperture or wavelength):
119         raise ValueError("No effective aperture or wavelength supplied")
120
121     return pow_density * eff_aperture * receive_loss
122
123
124 def free_space_loss(distance: float, wavelength: float) -> float:
125     """
126     Calculate the free space loss between two antennas
127
128     Parameters
129     -----
130     distance (float, km): distance between the transmit and receive
    ↪ antennas
131     frequency (float, GHz): frequency of the transmitter
132
133     Returns
134     -----
135     path_loss (float, )
136
137     """
138     return (wavelength / (4 * np.pi * distance)) ** 2
139
140
141 def free_space_loss_db(slant_range: float, frequency: float):
142     """
143     Calculate the free space loss between two antennas
144
145     Parameters
146     -----
147     slant_range (float, km): slant range between the transmit and
    ↪ receive antennas
148     frequency (float, GHz): frequency of the transmitter
149
150     Returns
151     -----
152     path_loss (float, dB): The path loss over the
153     """
154     return -92.44 - 20 * np.log10(slant_range * frequency)
155
156

```

```

157 def slant_path(
158     elevation_angle: float,
159     rain_altitude: float,
160     station_altitude: float,
161 ) -> float:
162     """
163     Calculate the slant path
164
165     Parameters
166     -----
167         angle_of_elevation (float, deg): the angle between the Earth
168     ↪ station and the satellite
169         rain_height (float, km): the rain height
170         station_altitude (float, km): the rain height of the Earth
171     ↪ station above sea level
172         refraction_radius (float, km): The modified radius of the Earth
173     ↪ to account for the
174         refraction of the wave by thr troposphere
175
176     Returns
177     -----
178         d_s (float, km): The slant height
179
180     """
181     refraction_radius = 8500 if station_altitude < 1.0 else EARTH_RADIUS
182     elevation_angle_rad = radians(elevation_angle)
183     if elevation_angle < 5:
184         return (
185             2
186             * (rain_altitude - station_altitude)
187             / np.sqrt(
188                 np.sin(elevation_angle_rad) ** 2
189                 + 2 * (rain_altitude - station_altitude) /
190                 ↪ refraction_radius
191             )
192         )
193     else:
194         return (rain_altitude - station_altitude) / np.sin(
195             ↪ elevation_angle_rad
196         )
197
198 def rain_specific_attenuation(frequency: float, rain_rate: float,
199     ↪ polarization: str):
200     """
201     TODO()
202
203     Parameters
204     -----
205
206     Returns
207     -----
208
209     """
210     _f = [
211         1,
212         2,

```

```
206         4,
207         6,
208         7,
209         8,
210         10,
211         12,
212         15,
213         20,
214         25,
215         30,
216         35,
217         40,
218         45,
219         50,
220         60,
221         70,
222         80,
223         90,
224         100,
225         120,
226         150,
227         200,
228         300,
229         400,
230     ]
231
232     _kH = [
233         0.0000387,
234         0.000154,
235         0.00065,
236         0.00175,
237         0.00301,
238         0.00454,
239         0.0101,
240         0.0188,
241         0.0367,
242         0.0751,
243         0.124,
244         0.187,
245         0.263,
246         0.35,
247         0.442,
248         0.536,
249         0.707,
250         0.851,
251         0.975,
252         1.06,
253         1.12,
254         1.18,
255         1.31,
256         1.45,
257         1.36,
258         1.32,
259     ]
260
```



```
261     _kV = [  
262         0.0000352,  
263         0.000138,  
264         0.000591,  
265         0.00155,  
266         0.00265,  
267         0.00395,  
268         0.00887,  
269         0.0168,  
270         0.0335,  
271         0.0691,  
272         0.113,  
273         0.167,  
274         0.233,  
275         0.31,  
276         0.393,  
277         0.479,  
278         0.642,  
279         0.784,  
280         0.906,  
281         0.999,  
282         1.06,  
283         1.13,  
284         1.27,  
285         1.42,  
286         1.35,  
287         1.31,  
288     ]  
289  
290     _alphaH = [  
291         0.912,  
292         0.963,  
293         1.121,  
294         1.308,  
295         1.332,  
296         1.327,  
297         1.276,  
298         1.217,  
299         1.154,  
300         1.099,  
301         1.061,  
302         1.021,  
303         0.979,  
304         0.939,  
305         0.903,  
306         0.873,  
307         0.826,  
308         0.793,  
309         0.769,  
310         0.753,  
311         0.743,  
312         0.731,  
313         0.71,  
314         0.689,  
315         0.688,
```

```

316         0.683,
317     ]
318
319     _alphaV = [
320         0.88,
321         0.923,
322         1.075,
323         1.265,
324         1.312,
325         1.31,
326         1.264,
327         1.2,
328         1.128,
329         1.065,
330         1.03,
331         1,
332         0.963,
333         0.929,
334         0.897,
335         0.868,
336         0.824,
337         0.793,
338         0.769,
339         0.754,
340         0.744,
341         0.732,
342         0.711,
343         0.69,
344         0.689,
345         0.684,
346     ]
347
348     KH = np.exp(np.interp(np.log(frequency), np.log(_f), np.log(_kH)))
349     KV = np.exp(np.interp(np.log(frequency), np.log(_f), np.log(_kV)))
350
351     alphaH = np.interp(np.log(frequency), np.log(_f), _alphaH)
352     alphaV = np.interp(np.log(frequency), np.log(_f), _alphaV)
353
354     if polarization == "circular":
355         k = (KH + KV) / 2
356         alpha = (KH * alphaH + KV * alphaV) / (2 * k)
357     elif polarization == "vertical":
358         k = KV
359         alpha = alphaV
360     elif polarization == "horizontal":
361         k = KH
362         alpha = alphaH
363     else:
364         raise Exception("Invalid Polarization")
365
366     return k, alpha, k * rain_rate**alpha
367
368
369 def horizontal_reduction(
370     horizontal_projection: float, specific_attenuation: float, frequency

```

```

    ↪ : float
371 ) -> float:
372     """
373     TODO()
374
375     Parameters
376     -----
377
378     Returns
379     -----
380     """
381     return 1 / (
382         1
383         + 0.78 * np.sqrt(horizontal_projection * specific_attenuation /
    ↪ frequency)
384         - 0.38 * (1 - np.exp(-2 * horizontal_projection))
385     )
386
387
388 def vertical_adjustment(
389     elevation_angle: float,
390     specific_attenuation: float,
391     d_r: float,
392     frequency: float,
393     chi: float,
394 ) -> float:
395     """
396     Calculate the vertical adjustment factor
397
398     Parameters
399     -----
400         elevation_angle (float, deg):
401         specific_attenuation (float, dBKm-1)
402         d_r (float, km):
403         frequency (float, GHz):
404         chi (float, deg):
405
406     Returns
407     -----
408         vert_adj (float, )
409     """
410     return 1 / (
411         1
412         + np.sqrt(np.sin(radians(elevation_angle)))
413         * (
414             31
415             * (1 - np.exp(-elevation_angle / (1 + chi)))
416             * (np.sqrt(d_r * specific_attenuation) / (frequency**2))
417             - 0.45
418         )
419     )
420
421
422 def zeta(
423     rain_altitude: float,

```

```

424     station_altitude: float,
425     horizontal_projection: float,
426     horizontal_reduction: float,
427 ) -> float:
428     """
429     Calculate interim vertical adjustment value
430
431     Parameters
432     -----
433
434     Returns
435     -----
436     """
437     return degrees(
438         atan2(
439             rain_altitude - station_altitude,
440             horizontal_projection * horizontal_reduction,
441         )
442     )
443
444
445 def rain_attenuation(
446     elevation_angle: float,
447     slant_path: float,
448     frequency: float,
449     rain_altitude: float,
450     station_altitude: float,
451     station_latitude: float,
452     rain_rate: float = 0.01,
453     polarization: str = "vertical",
454 ) -> float:
455     """
456
457     Parameters
458     -----
459
460     Returns
461     -----
462
463     """
464     elevation_angle_rad = radians(elevation_angle)
465     horiz_proj = slant_path * np.cos(elevation_angle_rad)
466
467     _, _, specific_att = rain_specific_attenuation(frequency, rain_rate,
468     ↪ polarization)
469
470     horiz_reduction = horizontal_reduction(horiz_proj, specific_att,
471     ↪ frequency)
472
473     zeta_ = zeta(rain_altitude, station_altitude, horiz_proj,
474     ↪ horiz_reduction)
475
476     if zeta_ > elevation_angle:
477         d_r = horiz_proj * horiz_reduction / np.cos(elevation_angle_rad)
478     else:

```

```

476         d_r = slant_path
477
478         if abs(station_latitude) < 36:
479             chi = 36 - abs(station_latitude)
480         else:
481             chi = 0
482
483         vert_adj = vertical_adjustment(elevation_angle, specific_att, d_r,
484 ↪ frequency, chi)
485         effective_path = slant_path * vert_adj
486
487         return specific_att * effective_path
488
489 def worst_rain_rate(rain_rate: float) -> float:
490     return (rain_rate / 0.3) ** 0.87
491
492
493 def polarization_loss(faraday_rotation: float) -> float:
494     rotation_rad = radians(faraday_rotation)
495     return 20 * np.log(cos(rotation_rad))

```

Listing 3: propagation.py

```

1 from math import acos, atan, atan2, cos, degrees, pi, radians, sin, sqrt
  ↪ , tan
2
3 from link_calculator.constants import EARTH_MU, EARTH_RADIUS
4
5
6 def velocity(
7     semi_major_axis: float, orbital_radius: float, mu: float = EARTH_MU
8 ) -> float:
9     """
10     Calculate the velocity of a satellite in orbit according to Kepler's
11     ↪ second law
12
13     Parameters
14     -----
15     semi_major_axis (float, km): The semi-major axis of the orbit
16     orbital_radius (float, km): distance from the centre of mass
17     ↪ to the satellite
18     mu (float, optional): Kepler's gravitational constant
19
20     Returns
21     -----
22     velocity (float, km/s): the orbit speed of the satellite
23     """
24     return sqrt(mu * (2 / orbital_radius - 1 / semi_major_axis))
25
26 def velocity_circular(orbital_radius: float, mu: float = EARTH_MU) ->
27     ↪ float:
28     """
29     Special case of velocity where r = a

```

```

28
29     Parameters
30     -----
31         orbital_radius (float, km): distance from the centre of mass to
↪ the satellite
32         mu (float, optional): Kepler's gravitational constant
33
34     Returns
35     -----
36         velocity (float, km/s): the orbit speed of the satellite
37     """
38     return velocity(orbital_radius, orbital_radius, mu)
39
40
41 def period(semi_major_axis: float, mu: float = EARTH_MU) -> float:
42     """
43     Calculate the period of the satellite's orbit according to Kepler's
↪ third law
44
45     Parameters
46     -----
47         semi_major_axis (float, km): The semi-major axis of the orbit
48         mu (float, km3/s-2, optional): Kepler's gravitational constant
49
50     Returns
51     -----
52         period (float, s): the time taken for the satellite to complete
↪ a revolution
53     """
54     return 2 * pi * sqrt(semi_major_axis**3 / mu)
55
56
57 def angle_sat_to_ground_station(
58     ground_station_lat: float,
59     ground_station_long: float,
60     sat_lat: float,
61     sat_long: float,
62 ) -> float:
63     """
64     Calculate angle gamma at the centre of the ground, between the Earth
↪ station and the satellite
65
66     Parameters
67     -----
68         ground_station_lat (float, deg): the latitude of the ground
↪ station
69         ground_station_long (float, deg): the longitude of the ground
↪ station
70         sat_lat (float, deg): the latitude of the satellite
71         sat_long (float, deg): the longitude of the satellite
72
73     Returns
74     -----
75         gamma (float, rad): angle between satellite and ground station
76     """

```

```

77     gs_lat_rad = radians(ground_station_lat)
78     gs_long_rad = radians(ground_station_long)
79     sat_lat_rad = radians(sat_lat)
80     sat_long_rad = radians(sat_long)
81
82     gamma = acos(
83         cos(gs_lat_rad) * cos(sat_lat_rad) * cos(sat_long_rad -
↪ gs_long_rad)
84         + sin(gs_lat_rad) * sin(sat_lat_rad)
85     )
86     return degrees(gamma)
87
88
89 def angle_sat_to_gs_orbital_radius(
90     orbital_radius: float, planet_radius: float = EARTH_RADIUS,
↪ elevation: float = 0
91 ):
92     """
93     Calculate angle gamma at the centre of the ground, between the Earth
94     station and the satellite, given the orbital radius of the satellite
95
96     Parameters
97     -----
98     orbital_radius (float, km): distance from the centre of mass to
↪ the satellite
99     planet_radius (float, km, optional): radius of the planet
100     elevation (float, deg): the angle of elevation over the horizon
101
102     Returns
103     -----
104     gamma (float, deg): angle between satellite and ground station
105     """
106     elevation_rad = radians(elevation)
107     gamma = acos((planet_radius * cos(elevation_rad)) / orbital_radius)
↪ - elevation_rad
108     return degrees(gamma)
109
110
111 def slant_range(
112     orbital_radius: float, angle_sat_to_gs: float, planet_radius: float
↪ = EARTH_RADIUS
113 ) -> float:
114     """
115     Calculate the slant range from the ground station to the satellite
116
117     Parameters
118     -----
119     orbital_radius (float, km): distance from the centre of mass to
↪ the satellite
120     angle_sat_to_gs (float, deg): angle from the satellite to the
↪ ground station, centred at the centre of mass
121     planet_radius (float, km, optional): radius of the planet
122
123     Return
124     -----

```

```

125     slant_range (float, km): the distance from the ground station to
    ↪ the satellite
126
127     """
128     return sqrt(
129         planet_radius**2
130         + orbital_radius**2
131         - 2 * planet_radius * orbital_radius * cos(radians(
    ↪ angle_sat_to_gs))
132     )
133
134
135 def elevation_angle(
136     orbital_radius: float, angle_sat_to_gs: float, planet_radius: float
    ↪ = EARTH_RADIUS
137 ) -> float:
138     """
139     Calculate the elevation angle from the ground station to the
    ↪ satellite
140
141     Parameters
142     -----
143     orbital_radius (float, km): distance from the centre of mass to
    ↪ the satellite
144     angle_sat_to_gs (float, rad): angle from the satellite to the
    ↪ ground station, centred at the centre of mass
145     planet_radius (float, km, optional): radius of the planet
146
147     Return
148     -----
149     elevation_angle (float, rad): the distance from the ground
    ↪ station to the satellite
150
151     """
152     gamma_rad = radians(angle_sat_to_gs)
153     elev = acos(
154         sin(gamma_rad)
155         / sqrt(
156             1
157             + (planet_radius / orbital_radius) ** 2
158             - 2 * (planet_radius / orbital_radius) * cos(gamma_rad)
159         )
160     )
161     return degrees(elev)
162
163
164 def area_of_coverage(angle_sat_to_gs: float, planet_radius: float =
    ↪ EARTH_RADIUS):
165     """
166     Calculate the surface area coverage of the Earth from a satellite
167
168     Parameters
169     -----
170     angle_sat_to_gs (float, deg): angle from the satellite to the
    ↪ ground station, centred at the centre of mass

```



```

171     planet_radius (float, km, optional): radius of the planet
172
173     Return
174     -----
175     area_coverage (float, km): the area of the Earth's surface
176     ↪ visible from a satellite
177
178     """
179     angle_sat_to_gs_rad = radians(angle_sat_to_gs)
180     return 2 * pi * (planet_radius**2) * (1 - cos(angle_sat_to_gs_rad))
181
182 def percentage_of_coverage(
183     angle_sat_to_gs: float, planet_radius: float = EARTH_RADIUS
184 ) -> float:
185     """
186     Calculate the surface area coverage of the Earth from a satellite as
187     ↪ a percentage of the total
188     surface area
189
190     Parameters
191     -----
192     angle_sat_to_gs (float, rad): angle from the satellite to the
193     ↪ ground station, centred at the centre of mass
194     planet_radius (float, km, optional): radius of the planet
195
196     Return
197     -----
198     area_coverage (float, %): the percentage of the Earth's surface
199     ↪ visible from a satellite
200
201     """
202     return (
203         area_of_coverage(angle_sat_to_gs, planet_radius)
204         / (4 * pi * planet_radius**2)
205         * 100
206     )
207
208 def percentage_of_coverage_gamma(angle_sat_to_gs: float) -> float:
209     """
210     Calculate the surface area coverage of the Earth from a satellite as
211     ↪ a percentage of the total
212     surface area
213
214     Parameters
215     -----
216     angle_sat_to_gs (float, rad): angle from the satellite to the
217     ↪ ground station, centred at the centre of mass
218
219     Return
220     -----
221     area_coverage (float, %): the percentage of the Earth's surface
222     ↪ visible from a satellite

```

```

219     """
220     gamma_rad = radians(angle_sat_to_gs)
221     return 50 * (1 - cos(gamma_rad))
222
223
224 def azimuth_intermediate(
225     ground_station_lat: float, ground_station_long: float, sat_long:
226     ↪ float
227 ) -> float:
228     """
229     Calculate the azimuth of a geostationary satellite
230
231     Parameters
232     -----
233     ground_station_lat (float, deg): the latitude of the ground
234     ↪ station
235     ground_station_long (float, deg): the longitude of the ground
236     ↪ station
237     sat_long (float, deg): the longitude of the satellite
238
239     Returns
240     -----
241     azimuth (float, rad): horizontal pointing angle of the ground
242     ↪ station antenna to
243     the satellite. The azimuth angle is usually measured in
244     ↪ clockwise direction
245     in degrees from true north.
246     """
247     gs_lat_rad = radians(ground_station_lat)
248     gs_long_rad = radians(ground_station_long)
249     sat_long_rad = radians(sat_long)
250     az = atan(tan(abs(sat_long_rad - gs_long_rad)) / sin(gs_lat_rad))
251     return degrees(az)
252
253 def max_visible_distance(
254     orbital_radius: float,
255     ground_station_lat: float,
256     min_angle: float = 0.0,
257     planet_radius: float = EARTH_RADIUS,
258 ) -> float:
259     """
260     Calculate the radius of visibility for a satellite and a ground
261     ↪ station
262
263     Parameters
264     -----
265     orbital_radius (float, km): the radius of the satellite from the
266     ↪ centre of the Earth
267     ground_station_lat (float, deg): the latitude of the ground
268     ↪ station
269     min_angle (float, deg): the minimum angle of visibility over the
270     ↪ horizon
271     planet_radius (float, km, optional): radius of the planet

```

```

265 Returns
266 -----
267
268 """
269 gs_lat_rad = radians(ground_station_lat)
270 min_angle_rad = radians(min_angle)
271 return acos(
272     cos(
273         (
274             acos((planet_radius / orbital_radius) * cos(
↪ min_angle_rad))
275             - min_angle_rad
276         )
277         / cos(gs_lat_rad)
278     )
279 )
280
281
282 def azimuth(
283     ground_station_lat: float,
284     ground_station_long: float,
285     sat_lat: float,
286     sat_long: float,
287 ) -> float:
288     """
289     Calculate the azimuth of a satellite
290
291     Parameters
292     -----
293     ground_station_lat (float, deg): the latitude of the ground
↪ station
294     ground_station_long (float, deg): the longitude of the ground
↪ station
295     sat_lat (float, deg): the latitude of the satellite
296     sat_long (float, deg): the longitude of the satellite
297
298     Returns
299     -----
300     azimuth (float, rad): horizontal pointing angle of the ground
↪ station antenna to
301     the satellite. The azimuth angle is usually measured in
↪ clockwise direction
302     in degrees from true north.
303     """
304     # Ground station is in northern hemisphere
305     # rel_pos = ground_station_long - sat_long
306     #
307     # if 90 > gs_long_rad > 0:
308     #
309     ## At the equator
310     ## In the southern hemisphere
311     # elif 0 > gs_long_rad >= -90:
312     #     if (sat_lat - ground_station_lat)
313     #         return pi / 2
314     #     else:

```

```

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

```

Listing 4: orbits.py

```

1 class Antenna:
2     def __init__(
3         self, name: str, power: float, gain: float, loss: float,
    ↪ frequency: float
4     ):
5         """
6         Instantiate an Antenna object
7
8         Parameters
9         -----
10            name (str): Name of antenna
11            power (float, W): transmit/receive power of the antenna
12            gain (float, W): the power gain of the antenna
13            loss (float, W): the power loss of the antenna
14            frequency (float, GHz): the transmit frequency of the
    ↪ antenna
15        """
16        self.power = power
17        self.gain = gain
18        self.loss = loss
19        self.frequency = frequency
20
21
22 class GroundStation:
23     def __init__(
24         self,
25         name: str,
26         latitude: float,
27         longitude: float,
28         altitude: float,
29         antenna: Antenna,
30     ):
31         """
32
33         Parameters
34         -----
35            name (str,): name of the ground station
36            latitude (str, deg): the latitude of the groundstation
37            longitude (str, deg): the longitude of the groundstation
38            altitude (str, km): the altitude of the groundstation above
    ↪ sea level
39        """
40        self.name = name
41        self.latitude = latitude
42        self.longitude = longitude
43        self.altitude = altitude
44        self.antenna = antenna
45
46
47 class Satellite:

```

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

Listing 5: components.py

```
1 import numpy as np  
2  
3 def axes_to_eccentricity(float a, float b):  
4     return np.sqrt(a**2, b**2) / a
```

Listing 6: conversions.py