Communication link budget calculator

ANSYS, Inc.

This tutorial demonstrates how to model a communications system including transmitters and receivers, and calculate link budgets considering factors like terrain, rain models, and atmospheric losses using Python and PySTK. It is inspired by this tutorial.

What are communications links, and how are they evaluated?

Communication links define the ways that a transmitter and a receiver access each other. STK allows the modeling of two kinds of links: basic and multi-hop. A basic communications link is access between just a single transmitter and a single receiver. A multi-hop communications link is defined by a group of communication objects consisting of a transmitter, receiver, re-transmitter, and another receiver. STK also allows setting constraints on communication links, including terrain masking constraints.

Communications links in STK can be analyzed using link budget reports, which include all the link parameters associated with the selected receiver or transmitter. Link budget reports take light speed delay into account. They also take computed refraction into account if enabled on the transmitter or receiver objects. Link budget reports include many communications-specific fields, including EIRP (effective isotropic radiated power in the link direction), C/N (Carrier-to-Noise ratio at the receiver input), Eb/No (Signal-to-Noise ratio at the receiver), and BER (Bit Error Rate). Link budget reports can also include access-specific information, such as the loss calculated by different selected models, including atmospheric, rain, cloud/fog, and terrain models.

Problem statement

A team of scientists is monitoring glacial meltwater in a remote, mountainous location (latitude 47.5605°, longitude 11.5027°). They need to determine how their location impacts a link budget between them and a low earth orbit (LEO), Earth observation satellite which is downloading data to the team. The satellite has a simple transmitter with an isotropic antenna pattern, a frequency of 1.7045 GHz, an EIRP of 10 dBW, a data rate of 4.2 Mb/sec, and a right-hand circular polarization. The camp's receiver is steerable, points at the satellite, and is placed on

a half power sensor located 6 ft above the ground, with a frequency of 1.7045 GHz and 1.6 m. The receiver has a parabolic antenna with a design frequency of 1.7 GHz, a 1.6 m diameter, and right-hand circular polarization.

Model and analyze a link budget between the ground site and the Earth observation satellite, taking into account different factors such as terrain, rain and atmospheric absorption, and system noise temperature from sun, atmosphere, rain, and cosmic background.

Launch a new STK instance

Start by launching a new STK instance. In this example, STKEngine is used.

```
from ansys.stk.core.stkengine import STKEngine


stk = STKEngine.start_application(no_graphics=False)
print(f"Using {stk.version}")
```

```
Using STK Engine v13.0.0
```

Create a new scenario

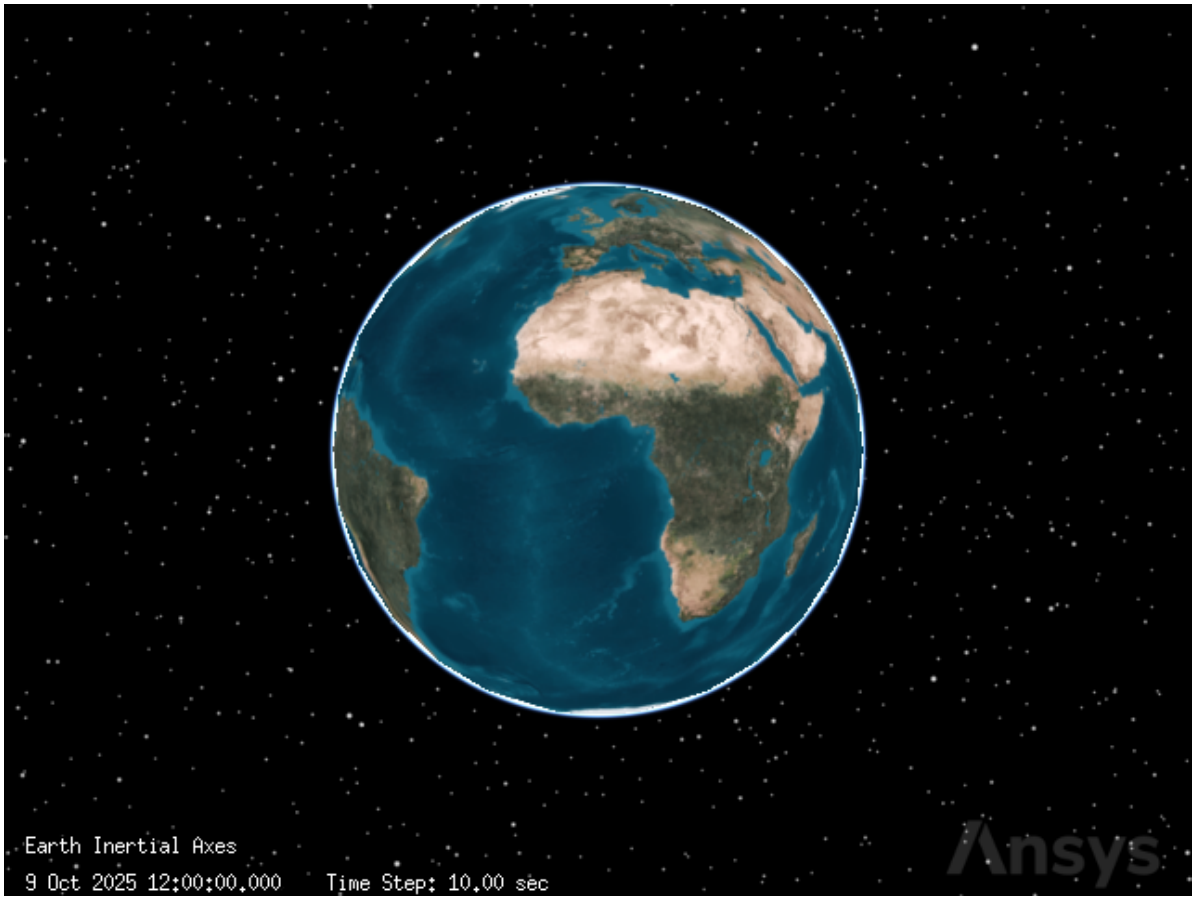
Create a new scenario in STK by running:

```
root = stk.new_object_root()
root.new_scenario("Communications")
```

Once the scenario is created, it is possible to show a 3D graphics window by running:

```
from ansys.stk.core.stkengine.experimental.jupyterwidgets import GlobeWidget


globe = GlobeWidget(root, 640, 480)
globe.show()
```

```
RFBOutputContext()
```



Set the scenario time period

Using the newly created scenario, set the start and stop times. Rewind the scenario so that the graphics match the start and stop times of the scenario:

```
scenario = root.current_scenario
scenario.set_time_period("15 Mar 2024 06:00:00.000", "16 Mar 2024 06:00:00.000")
root.rewind()
```

Add analytical and visual terrain

Use an STK terrain file (file extension PDTT) included with the STK install to add analytical terrain to the scenario. The file contains information on the terrain around the scientists' camp site. Use a Connect command to find the path to the RaistingStation.pdtt file:

```
import pathlib

from ansys.stk.core.stkobjects import TerrainFileType


install_dir = root.execute_command("GetDirectory / STKHome")[0]
terrain_path = str(
    pathlib.Path(install_dir)
    / "Data"
    / "Resources"
    / "stktraining"
    / "imagery"
    / "RaistingStation.pdtt"
)
```

Then, add the file to the Earth central body's terrain collection:

```
terrain = scenario.terrain.item("Earth").terrain_collection.add(
    terrain_path, TerrainFileType.PDTT
)
```

This file is used for analysis by default after it is inserted.

Add a satellite

The Earth observation satellite is in a sun-synchronous orbit. It can thus be modelled by an SGP4 propagator, which is used for LEO satellites. The satellite communicating with the camp has a common name of TerraSarX, which corresponds to a space surveillance catalog number of 31698.

To add the satellite, first insert a satellite object:

```
from ansys.stk.core.stkobjects import PropagatorType, STKObjectType


satellite = root.current_scenario.children.new(
    STKObjectType.SATELLITE, "TerraSarX_31698"
)
```

Then, set the satellite's propagator to the SGP4 propagator:

```
satellite.set_propagator_type(PropagatorType.SGP4)
propagator = satellite.propagator
```

Finally, use the propagator's `common_tasks` property to add the satellite's orbit from an online source, and propagate the satellite:

```
propagator.common_tasks.add_segments_from_online_source("31698")
propagator.propagate()
```

Add the camp site

Add a place object to represent the camp site:

```
camp_site = root.current_scenario.children.new(STKObjectType.PLACE, "CampSite")
```

Assign the site's position to latitude 47.5605° and longitude 11.5027°, with an elevation of 6 ft (0.0018288 km) to simulate the height of the equipment at the site:

```
camp_site.position.assign_planetodetic(47.5605, 11.5027, 0.0018288)
```

Model a simple transmitter

The satellite has a simple transmitter model, a model type which uses an isotropic, omnidirectional antenna, which is an ideal spherical pattern antenna with constant gain. Insert the transmitter on the satellite:

```
transmitter = satellite.children.new(STKObjectType.TRANSMITTER, "DownloadTransmitter")
```

The transmitter's `model` property now contains a `TransmitterModelSimple` object. Set the model's frequency to 1.7045 GHz:

```
from ansys.stk.core.stkobjects import TransmitterModelSimple

transmitter_model = TransmitterModelSimple(
    transmitter.model_component_linking.component
)
transmitter_model.frequency = 1.7045
```

Then, set the model's EIRP, which is the effective isotropic radiated power at the output of the transmit antenna. Set the EIRP to 10 dBW:

```
transmitter_model.eirp = 10
```

Next, set the model's data rate to 4.2 Mb/sec:

```
transmitter_model.data_rate = 4.2
```

Then, enable polarization on the model:

```
transmitter_model.enable_polarization = True
```

Finally, set the model's polarization type to right-hand circular:

```
from ansys.stk.core.stkobjects import PolarizationType


transmitter_model.set_polarization_type(PolarizationType.RIGHT_HAND_CIRCULAR)
```

Add a steerable sensor

The receiver antenna at the camp site is steerable. To create a steering device, add a sensor object:

```
sensor = camp_site.children.new(STKObjectType.SENSOR, "ServoMotor")
```

Then, set the sensor's pattern to a half power pattern, which is designed to visually model parabolic antennas. The sensor half angle is determined by frequency and antenna diameter.

```
from ansys.stk.core.stkobjects import SensorPattern


sensor.set_pattern_type(SensorPattern.HALF_POWER)
```

The sensor's `pattern` property now holds a `SensorHalfPowerPattern` object, through which it is possible to configure the half power model. First, set the sensor's frequency to 1.7045 GHz:

```
sensor.pattern.frequency = 1.7045
```

Then, set the sensor's antenna diameter to 1.6 m:

```
sensor.pattern.antenna_diameter = 1.6
```

The sensor is steerable and tracks the satellite, so set the sensor's pointing type to targeted:

```
from ansys.stk.core.stkobjects import SensorPointing


sensor.set_pointing_type(SensorPointing.TARGETED)
```

The sensor's `pointing` property now holds a `SensorPointingTargeted` object, through which it is possible to set the satellite as the sensor's target:

```
sensor.pointing.targets.add("Satellite/TerraSarX_31698")
```

Calculate access

Get and compute the access between the camp site's sensor and the satellite:

```
sensor_to_satellite_access = sensor.get_access_to_object(satellite)
sensor_to_satellite_access.compute_access()
```

Then, get the access data during the entire scenario as a dataframe:

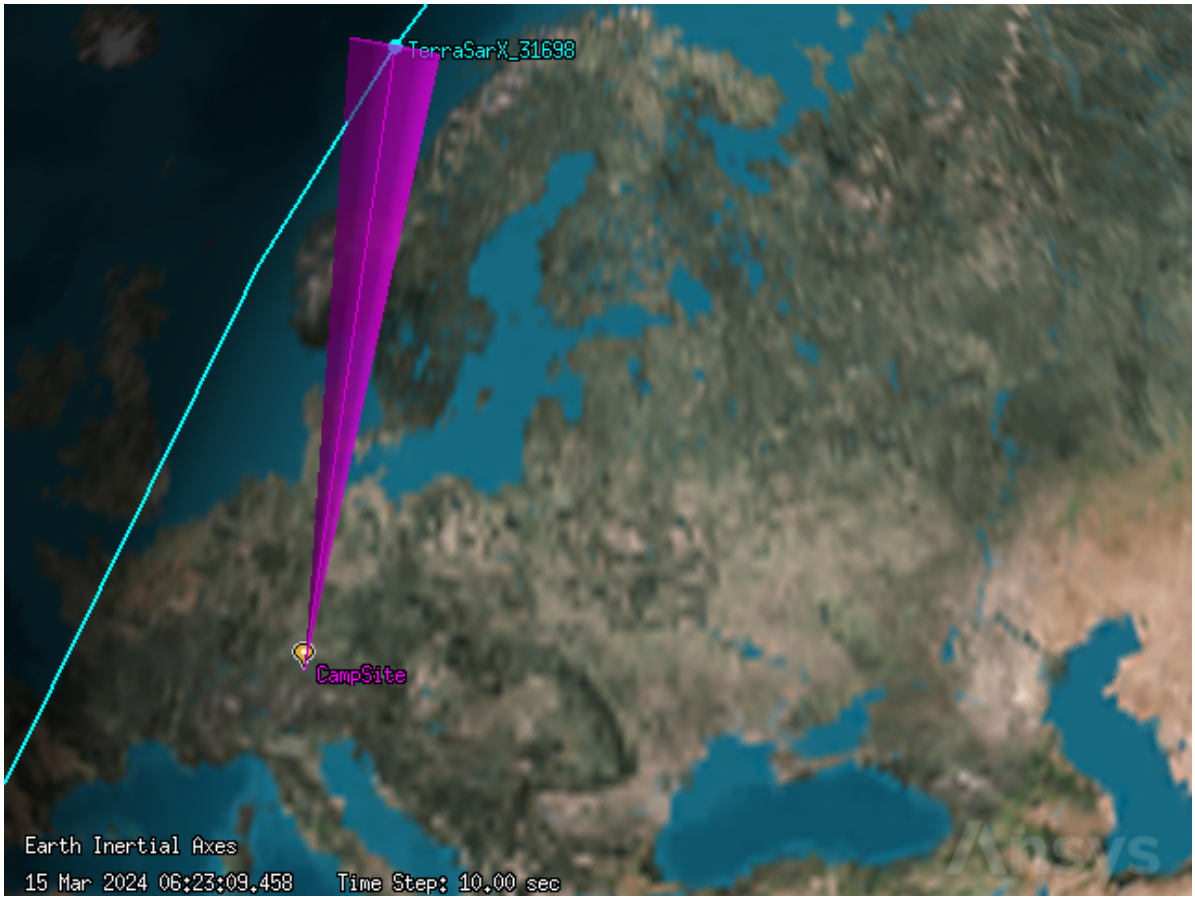
```
sensor_to_satellite_access_df = (
    sensor_to_satellite_access.data_providers.item("Access Data")
    .execute(scenario.start_time, scenario.stop_time)
    .data_sets.to_pandas_dataframe()
)
sensor_to_satellite_access_df
```

	access number	start time	stop time	duration
0	1	15 Mar 2024 06:22:09.458394650	15 Mar 2024 06:33:14.874552309	665.4161576585711
1	2	15 Mar 2024 07:58:01.500105812	15 Mar 2024 08:02:54.221900294	292.7217944814738
2	3	15 Mar 2024 15:31:09.921082116	15 Mar 2024 15:41:19.040164169	609.1190820532502
3	4	15 Mar 2024 17:04:09.317277119	15 Mar 2024 17:15:35.046016598	685.7287394791856
4	5	15 Mar 2024 18:42:20.852340085	15 Mar 2024 18:47:07.001005909	286.14866582359537
5	6	16 Mar 2024 04:31:32.112160260	16 Mar 2024 04:41:28.374312584	596.2621523233247

The sensor is able to access the satellite six times throughout the duration of the scenario.

The access between the sensor and the satellite can be seen in the 3D graphics window approximately 1389 seconds after the scenario begins:

```
root.current_time = 1389.458394
globe.camera.position = [2703.568833967754, -5862.711497073381, 9424.82507431983]
globe.show()
```



Model the receiver

The sensor's receiver is modelled using a complex receiver model, which allows selecting among a variety of analytical and realistic antenna models and defining the characteristics of the selected antenna type.

First, add the receiver on the sensor:

```
receiver = sensor.children.new(STKObjectType.RECEIVER, "DownloadReceiver")
```

Then, set the receiver's model type to the complex receiver model:

```
receiver.model_component_linking.set_component("Complex Receiver Model")
```

Next, use the model's `antenna_control` property to set the receiver's embedded model to a parabolic antenna:

```
from ansys.stk.core.stkobjects import ReceiverModelComplex
```

```
receiver_model = ReceiverModelComplex(receiver.model_component_linking.component)
receiver_model.antenna_control.embedded_model_component_linking.set_component(
    "Parabolic"
)
```

The receiver model's antenna control's `embedded_model` property now holds an `AntennaModelParabolic` object, through which it is possible to configure the antenna model. First, configure the antenna model to use diameter as its input type:

```
from ansys.stk.core.stkobjects import AntennaModelInputType, AntennaModelParabolic


antenna_control = AntennaModelParabolic(
    receiver_model.antenna_control.embedded_model_component_linking.component
)
antenna_control.input_type = AntennaModelInputType.DIAMETER
```

Then, set the diameter to 1.6 m:

```
antenna_control.diameter = 1.6
```

Set the design frequency to 1.7 GHz:

```
antenna_control.design_frequency = 1.7
```

Next, enable the use of polarization on the receiver model:

```
receiver_model.enable_polarization = True
```

The receiver's polarization type is the same as the transmitter's polarization, so set the model's polarization type to right-hand circular:

```
receiver_model.set_polarization_type(PolarizationType.RIGHT_HAND_CIRCULAR)
```

Calculate access

Get and calculate the access between the receiver and transmitter:

```
receiver_basic_access = receiver.get_access_to_object(transmitter)
receiver_basic_access.compute_access()
```

Then, get the link information for the access for the entire scenario, using a time step of 30 s:

```
receiver_basic_link_df = (
    receiver_basic_access.data_providers.item("Link Information")
    .execute(scenario.start_time, scenario.stop_time, 30)
    .data_sets.to_pandas_dataframe()
)
```

Get the columns corresponding to time, atmospheric loss (atmos loss), rain loss, EIRP in the link direction (eirp), received isotropic power at the receiver antenna (rcvd. iso. power), power flux density at the receiver antenna (flux density), receiver gain over equivalent noise temperature (g/T), carrier-to-noise density at the receiver input (c/no), bandwidth, carrier-to-noise ratio at the receiver input (c/n), signal-to-noise ratio at the receiver (eb/no), bit error rate (ber), and calculated system noise temperatures (tatmos, train, tsun, tearth, tcosmic, tantenna, tequivalent):

```
link_budget_columns = [
    "time",
    "atmos loss",
    "rain loss",
    "eirp",
    "rcvd. frequency",
    "rcvd. iso. power",
    "flux density",
    "g/t",
    "c/no",
    "bandwidth",
    "c/n",
    "eb/no",
    "ber",
    "tatmos",
    "train",
    "tsun",
    "tearth",
    "tcosmic",
    "tantenna",
    "tequivalent",
]
receiver_basic_link_df.head(10)[link_budget_columns]
```

	time	atmos loss	rain loss	eirp	rcvd. frequency	rcvd. iso. power	flux d
0	15 Mar 2024 06:22:09.458394650	0.0	0.0	10.0	1.704539	-155.457601	-129.3
1	15 Mar 2024 07:58:01.500105812	0.0	0.0	10.0	1.704517	-155.454205	-129.3
2	15 Mar 2024 15:31:09.921082116	0.0	0.0	10.0	1.704535	-155.391722	-129.3

	time	atmos loss	rain loss	eirp	rcvd. frequency	rcvd. iso. power	flux d
3	15 Mar 2024 17:04:09.317277119	0.0	0.0	10.0	1.704539	-155.372865	-129.2
4	15 Mar 2024 18:42:20.852340085	0.0	0.0	10.0	1.704515	-155.405099	-129.3
5	16 Mar 2024 04:31:32.112160260	0.0	0.0	10.0	1.704533	-155.452536	-129.3
6	15 Mar 2024 06:22:39.000000000	0.0	0.0	10.0	1.704539	-154.765272	-128.6
7	15 Mar 2024 07:58:31.000000000	0.0	0.0	10.0	1.704514	-155.18293	-129.0
8	15 Mar 2024 15:31:39.000000000	0.0	0.0	10.0	1.704535	-154.772723	-128.6
9	15 Mar 2024 17:04:39.000000000	0.0	0.0	10.0	1.704539	-154.657114	-128.5

The dataframe now shows information corresponding to an STK link budget report. From the data, it is possible to see that as the satellite rises over the horizon of the central body, the site receives transmissions. When the satellite falls below the horizon, the site loses transmissions. Additionally, because the access calculation does not include any environmental factor models or system noise temperature considerations, the columns corresponding to the losses/noise all have values of 0.

Add terrain to the analysis

Next, add a terrain mask to the receiver to add terrain into the access analysis. A terrain mask causes STK to constrain access to an object by any terrain data in the line of sight to which access is being calculated. Add a terrain mask access constraint:

```
from ansys.stk.core.stkobjects import AccessConstraintType


terrain_constraint = receiver.access_constraints.add_constraint(
    AccessConstraintType.TERRAIN_MASK
)
```

Recalculate the access between the receiver and transmitter, then get the data corresponding to a link budget report:

```
receiver_basic_access.compute_access()
receiver_terrain_link_df = (
    receiver_basic_access.data_providers.item("Link Information")
    .execute(scenario.start_time, scenario.stop_time, 30)
    .data_sets.to_pandas_dataframe()
)
receiver_terrain_link_df.head(10)[link_budget_columns]
```

	time	atmos loss	rain loss	eirp	rcvd. frequency	rcvd. iso. power	flux d
0	15 Mar 2024 06:23:47.759570010	0.0	0.0	10.0	1.704537	-152.945069	-126.8
1	15 Mar 2024 15:33:36.548124287	0.0	0.0	10.0	1.704527	-152.052659	-125.9
2	15 Mar 2024 17:07:34.413895835	0.0	0.0	10.0	1.704534	-148.927067	-122.8
3	16 Mar 2024 04:32:28.173285490	0.0	0.0	10.0	1.704531	-154.328235	-128.2
4	15 Mar 2024 06:24:17.000000000	0.0	0.0	10.0	1.704536	-152.071471	-125.9
5	15 Mar 2024 15:34:06.000000000	0.0	0.0	10.0	1.704524	-151.374141	-125.2
6	15 Mar 2024 17:08:04.000000000	0.0	0.0	10.0	1.70453	-147.664614	-121.5
7	16 Mar 2024 04:32:58.000000000	0.0	0.0	10.0	1.704529	-153.707919	-127.0
8	15 Mar 2024 06:24:47.000000000	0.0	0.0	10.0	1.704534	-151.110464	-125.0
9	15 Mar 2024 15:34:36.000000000	0.0	0.0	10.0	1.70452	-150.737395	-124.0

Next, get the access data for the updated access:

```
receiver_basic_access.data_providers.item("Access Data").execute(
    scenario.start_time, scenario.stop_time
).data_sets.to_pandas_dataframe()
```

	access number	start time	stop time	duration
0	1	15 Mar 2024 06:23:47.759570010	15 Mar 2024 06:30:05.292544675	377.53297466502795
1	2	15 Mar 2024 15:33:36.548124287	15 Mar 2024 15:39:49.757492637	373.2093683504718
2	3	15 Mar 2024 17:07:34.413895835	15 Mar 2024 17:13:56.389724097	381.97582826247526
3	4	16 Mar 2024 04:32:28.173285490	16 Mar 2024 04:38:33.018850881	364.84556539097684

Before adding a terrain mask, there were 6 accesses between the receiver and transmitter. By adding a terrain mask, the accesses blocked by terrain have been removed from the report, and there are now only 4 accesses between the receiver and transmitter.

Model environmental factors

Environmental factors can affect the performance of a communications link. In STK, it is possible to enable or disable the use of different environmental factors at three levels: scenario, platform (facilities, places, targets, and all vehicles except satellites), and subobject (transmitter, receiver, radar, and antenna). In this case, since the scenario only includes a single receiver/transmitter pair, set the environmental factors at the scenario level.

First, add a rain model, which is used to estimate the amount of degradation (or fading) of signal when passing through rain. The degradation is primarily due to absorption by water molecules and is a function of frequency and elevation angle. Generally speaking, the rain loss increases with increasing frequency. The loss also increases with decreasing ground elevation

angle due to a greater path distance through the portion of the atmosphere where rain occurs. Rain also causes an increase in the antenna noise temperature. Set the `enable_rain_loss` property to `True` on the scenario's RF environment's propagation channel:

```
scenario.rf_environment.propagation_channel.enable_rain_loss = True
```

Then, enable the use of the atmospheric absorption model:

```
scenario.rf_environment.propagation_channel.enable_atmospheric_absorption = True
```

It is possible to configure which specific model is used for the different environmental factors. However, in this case, the default models are sufficient.

Next, recalculate the access between the receiver and transmitter:

```
receiver_basic_access.compute_access()
receiver_environmental_link_df = (
    receiver_basic_access.data_providers.item("Link Information")
    .execute(scenario.start_time, scenario.stop_time, 30)
    .data_sets.to_pandas_dataframe()
)
receiver_environmental_link_df.head(10)[link_budget_columns]
```

	time	atmos loss	rain loss	eirp	rcvd. frequency	rcvd. iso. power	flux
0	15 Mar 2024 06:23:47.759570010	0.23797	0.005026	10.0	1.704537	-153.188065	-127.
1	15 Mar 2024 15:33:36.548124287	0.174104	0.003672	10.0	1.704527	-152.230434	-126.
2	15 Mar 2024 17:07:34.413895835	0.085919	0.00175	10.0	1.704534	-149.014736	-122.
3	16 Mar 2024 04:32:28.173285490	0.479177	0.012249	10.0	1.704531	-154.81966	-128.
4	15 Mar 2024 06:24:17.000000000	0.17687	0.003635	10.0	1.704536	-152.251976	-126.
5	15 Mar 2024 15:34:06.000000000	0.144554	0.002995	10.0	1.704524	-151.52169	-125.
6	15 Mar 2024 17:08:04.000000000	0.069449	0.001407	10.0	1.70453	-147.735471	-121.
7	16 Mar 2024 04:32:58.000000000	0.332202	0.007488	10.0	1.704529	-154.047609	-127.
8	15 Mar 2024 06:24:47.000000000	0.136309	0.00276	10.0	1.704534	-151.249533	-125.
9	15 Mar 2024 15:34:36.000000000	0.124049	0.00254	10.0	1.70452	-150.863984	-124.

After adding environmental factor models, the atmospheric and rain losses are now greater than 0. However, the losses are minimal, so the losses in C/N and Eb/No are also minimal.

Model system noise temperature

The receiver's system noise temperature enables specifying the system's inherent noise characteristics, which can help simulate real-world RF situations more accurately. STK can use either a constant system noise temperature value, or can calculate it based off of different noise sources. In this case, configure the receiver model's system noise temperature to use calculated values:

```
from ansys.stk.core.stkobjects import NoiseTemperatureComputeType


receiver_model.system_noise_temperature.compute_type = (
    NoiseTemperatureComputeType.CALCULATE
)
```

Do the same for the model's antenna noise temperature:

```
receiver_model.system_noise_temperature.antenna_noise_temperature.compute_type = (
    NoiseTemperatureComputeType.CALCULATE
)
```

Then, enable the use of sun, atmosphere, rain, and cosmic background in computations:

```
receiver_model.system_noise_temperature.antenna_noise_temperature.use_sun = True
receiver_model.system_noise_temperature.antenna_noise_temperature.use_atmosphere = True
receiver_model.system_noise_temperature.antenna_noise_temperature.use_rain = True
receiver_model.system_noise_temperature.antenna_noise_temperature.use_cosmic_background = Tru
```

Finally, recompute the access and get the updated link information:

```
receiver_basic_access.compute_access()
receiver_noise_link_df = (
    receiver_basic_access.data_providers.item("Link Information")
    .execute(scenario.start_time, scenario.stop_time, 30)
    .data_sets.to_pandas_dataframe()
)
receiver_noise_link_df.head(10)[link_budget_columns]
```

	time	atmos loss	rain loss	eirp	rcvd. frequency	rcvd. iso. power	flux
0	15 Mar 2024 06:23:47.759570010	0.23797	0.005026	10.0	1.704537	-153.188065	-127.
1	15 Mar 2024 15:33:36.548124287	0.174104	0.003672	10.0	1.704527	-152.230434	-126.
2	15 Mar 2024 17:07:34.413895835	0.085919	0.00175	10.0	1.704534	-149.014736	-122.
3	16 Mar 2024 04:32:28.173285490	0.479177	0.012249	10.0	1.704531	-154.81966	-128.
4	15 Mar 2024 06:24:17.000000000	0.17687	0.003635	10.0	1.704536	-152.251976	-126.

	time	atmos loss	rain loss	eirp	rcvd. frequency	rcvd. iso. power	flux
5	15 Mar 2024 15:34:06.000000000	0.144554	0.002995	10.0	1.704524	-151.52169	-125.
6	15 Mar 2024 17:08:04.000000000	0.069449	0.001407	10.0	1.70453	-147.735471	-121.
7	16 Mar 2024 04:32:58.000000000	0.332202	0.007488	10.0	1.704529	-154.047609	-127.
8	15 Mar 2024 06:24:47.000000000	0.136309	0.00276	10.0	1.704534	-151.249533	-125.
9	15 Mar 2024 15:34:36.000000000	0.124049	0.00254	10.0	1.70452	-150.863984	-124.

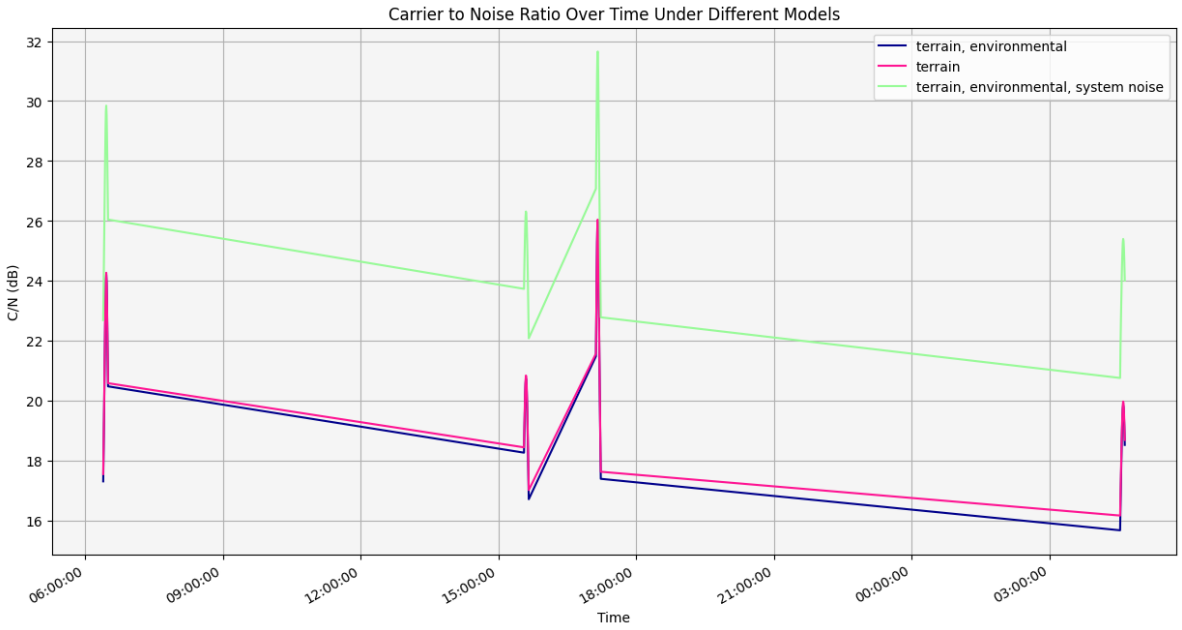
There are now non-zero values in the tatmos, train, tsun, tcosmic, tantenna, and tequivalent columns. Calculating system noise temperature improved the report and extended the time available for downloading data.

Plot the different link budgets

Visualize the calculated link budgets under the different modeling factors:

```
import matplotlib.dates as md
import matplotlib.pyplot as plt
import pandas as pd


# Create plot
fig, ax = plt.subplots()

# Format data
receiver_environmental_link_df["time"] = pd.to_datetime(
    receiver_environmental_link_df["time"]
)
receiver_environmental_link_df.sort_values(by="time", inplace=True)
receiver_terrain_link_df["time"] = pd.to_datetime(receiver_terrain_link_df["time"])
receiver_terrain_link_df.sort_values(by="time", inplace=True)
receiver_noise_link_df["time"] = pd.to_datetime(receiver_noise_link_df["time"])
receiver_noise_link_df.sort_values(by="time", inplace=True)

# Plot dataframes
receiver_environmental_link_df.plot(
    x="time", y="c/no", ax=ax, label="terrain, environmental", color="darkblue"
)
receiver_terrain_link_df.plot(
    x="time", y="c/no", ax=ax, label="terrain", color="deeppink"
)
receiver_noise_link_df.plot(
```

```
    x="time",
    y="c/no",
    ax=ax,
    label="terrain, environmental, system noise",
    color="palegreen",
)

# Configure plot style
ax.set_facecolor("whitesmoke")
ax.grid(visible=True, which="both")

# Set title and labels
ax.set_title("Carrier to Noise Ratio Over Time Under Different Models")
ax.set_xlabel("Time")
ax.set_ylabel("C/N (dB)")

# Improve x-axis formatting
formatter = md.DateFormatter("%H:%M:%S")
ax.xaxis.set_major_formatter(formatter)

# Set figure size
fig.set_size_inches(15, 8)

# Show figure
plt.show()
```



The model accounting for only terrain has similar C/N values to the model accounting for terrain and environmental factors, as the losses from environmental factors were minimal in this analysis. The model accounting for terrain, environmental factors, and system noise consistently had the highest C/N values, as calculating system noise temperature improved the link budget.