An Earth-Venus-Earth Link Budget from Open Research Institute

This document contains a detailed link budget analysis for Earth-Venus-Earth (EVE) communications. It begins with a Python dataclass for each fixed earth station. Dataclasses are a type of Python object where only variables are declared. No methods are included. The site-specific dataclasses have parameters that are true for the site regardless of the target. Following the site-specific dataclasses are link budget classes, which contain target-specific values and methods that return various gains and losses. The most important output of a link budget object is a carrier to noise ratio at a particular receive bandwidth. A link budget class inherits a particular site dataclass. The site-specific dataclasses and the link budgets can be mixed and matched. This gives flexibility, as a link budget for a particular target, such as EVE, can be calculated for different sites by having that link budget inherit different site-specific dataclasses.

What is a Link Budget?

If you have ever planned out income and expenses with respect to pay, and handled monthly bills, then you have used a budget. A budget is a balance between an amount of something coming into an account or collection, and an amount of something going out. A link budget uses the same concepts as a financial budget, with income (gain) and expenses (losses) counted up over a period of time and space. The link in link budget is the physical path or link between transmitter and receiver. Like financial budgeting, link budgets can range from a very simple picture of financial activity to a very complex accounting of a wide variety of types of income and expenses.

Link budgets are models of the real world. There are many ways to represent the physical environment and almost any model of that environment can be improved. A simple link budget can be very useful. A complex one can be very wrong.

A high quality link budget is accurate, accessible, and flexible. Link budgets can be used in a variety of ways. The most familiar role is predictive, where a link budget is used to design a communications experiment or system implementation. Link budgets can also be used as documentation. These are accurate only after the fact of a communications experiment or system implementation. Finally, link budgets are an excellent educational resource, providing a system-level view of radio communications. The goal for this link budget is to be useful in all three of these roles: predictive, documentary, and educational.

This link budget is written in an object oriented coding style. An object oriented style means that we construct our calculations so that they can be controlled and manipulated

in a modular manner. The gains and losses are described using classes. Classes are structures that contain members and methods. Members, which are things like variables and constants, are the "nouns" of our model. The methods, or functions, are the "verbs". When we think about our link budget, we group members and methods that are related to particular concepts. For example, one of the first classes in the link budget has to do with noise temperature, which is a very important part of our radio environment. The amount of noise impacts signal reception and performance. We need to know where sources of noise come from, and account for them.

Once we have decided on a topic or subject, we define a class. We list the members and the methods that belong to that class. Then we use our class definition to create a calculator. Think of a calculator as if you had a physical custom-designed calculator that let you enter in everything having to do with, say, noise temperature. Each member could be given a value through a keypad. Each method might have its own button. You press the button and, as long as you've defined all the members needed by that method, the answer to that particular calculation pops out. We use several different calculators along the way as we build up all the gains and losses of our link budget. Deciding what work goes into what class is part of the art of computer programming. Two different people, when given the same problem, and each deciding to use object oriented techniques, may come up with very different class structures for their code. The first person might have approached the problem in a highly modular way, with many classes and lots of intermediate results. The second person might have solved the problem with one class, doing things in less steps but with more complexity per step. As long as the right answer comes out and the program doesn't use more resources than are available, both approaches are valid.

EVE compared to EME

The communications mode most similar to EVE is Earth-Moon-Earth (EME). EVE is more challenging than EME communications due to several factors.

1. Much greater distances involved
2. Greater variability in the distances involved
3. Doppler rate of change due to orbital positions of Earth and Venus changing with respect to each other
4. Different signal reflection characteristics from Venus compared to the Moon
5. Doppler spreading, which is a type of Doppler due to the signal being reflected off a rotating object.

Contributors

The team contributing to the link budget includes but is not limited to Michelle Thompson, Matthew Wishek, Paul Williamson, Rose Easton, Thomas Telkamp, Pete

Wyckoff, Gary K6MG, and Lee Blanton. Questions or comments about this document can be directed to the issues and pull request functions in the repository below, or one can write an email to hello at openresearch dot institute.

Imports

This link budget is written in a Jupyter Lab Notebook, using the Python computer language. We import the Python modules that we need for the project.

In [214…

```
# Imports
import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass
from typing import Optional, Dict, Any
from skyfield.api import load, wgs84
from datetime import datetime, timedelta, timezone
import matplotlib.pyplot as plt
import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import math
import plotly.io as pio
pio.renderers.default = "notebook"
```

Dataclass Definitions

Dataclasses in Python are classes that contain only variables and constants. In other words, they are only "nouns", containing no methods. Our dataclasses contain information about specific sites. Each site gets its own dataclass. We set up dataclasses using the naming format SiteNameLinkParameters. Values for the members of the class have been given to us from sites such as Deep Space Exploration Society (DSES), who first asked for assistance with an EVE link budget. Our dataclasses set the values that are true for these sites regardless of the celestial target. For example, 1296 MHz (23cm band) is a common frequency for all sorts of communications, not just EVE. The location and elevation of a site remains the same no matter what it's pointing at. Values like this belong to the SiteNameLinkParameters class.

Can you see three values that could belong to a PhysicalConstants dataclass? Unless the speed of light, Boltzmann's constant, and reference temperature are specific to a particular site, then these values are the sort of thing that should be moved to dataclass dedicated to Physical Constants. This is just one of many decisions along the way that we might need to make in order to improve our link budget.

In [215…

```
# Dataclass Definitions
@dataclass
class DSESLinkParameters: #things that are true for the site regardless of t
```

```
    # location of this dish in Google Earth is 38°22'51"N 103°09'22"W
    latitude: float = 38.380833 # from FCC converter
    longitude: float = -103.156111 # from FCC converter
    elevation: float = 1311 # meters, from online calculator
#    tx_frequency_mhz: float = 1296.0 # Transmit frequency
    tx_frequency_mhz: float = 2304.0 # Transmit frequency anticipated for Oc
    tx_power_w: float = 1500.0 # Transmit power in watts
    tx_antenna_diameter_m: float = 18.29 # Transmit antenna diameter
    tx_antenna_efficiency: float = 0.69 # Transmit antenna efficiency
    rx_antenna_diameter_m: float = 18.29 # Receive antenna diameter
    rx_antenna_efficiency: float = 0.69 # Receive antenna efficiency
    tx_line_loss_db: float = 0.5 # provided
    rx_line_loss_db: float = 0.5 # provided
    pointing_error_deg: float = 0.01 # provided
    lna_noise_figure_db: float = 0.4 # from datasheet for the Kuhne MKU LNA
    lna_noise_figure_K: float = 290 * (10**(lna_noise_figure_db/10) - 1) # c
    receiver_noise_bandwidth: float = 100e3 # operational receiver bandwidth
    c: float = 299792458  # Speed of light in m/s
    k: float = 1.380649e-23  # Boltzmann constant in J/K
    t0: float = 290  # Reference temperature in K


@dataclass
class DwingelooLinkParameters: #things that are true for the site regardless
    latitude: float = 52.81213723180477 # from Thomas
    longitude: float = 6.396346463227839 # from Thomas
    elevation: float = 70.26 # from Thomas
#    tx_frequency_mhz: float = 1299.0 # Transmit frequency from Thomas
    tx_frequency_mhz: float = 2304.0 # Transmit frequency anticipated for Oc
    tx_power_w: float = 1000.0 # Transmit power in watts
    tx_antenna_diameter_m: float = 25 # Transmit antenna diameter from Thoma
    tx_antenna_efficiency: float = 0.69 # Transmit antenna efficiency from T
    rx_antenna_diameter_m: float = 25 # Receive antenna diameter from Thomas
    rx_antenna_efficiency: float = 0.69 # Receive antenna efficiency from Th
    tx_line_loss_db: float = 0.5 # best guess
    rx_line_loss_db: float = 0.5 # best guess
    pointing_error_deg: float = 0.01 # best guess
    lna_noise_figure_db: float = 0.629 # best guess working backwards from a
    lna_noise_figure_K: float = 290 * (10**(lna_noise_figure_db/10) - 1) # c
    receiver_noise_bandwidth: float = 100e3 # operational receiver bandwidth
    c: float = 299792458  # Speed of light in m/s
    k: float = 1.380649e-23  # Boltzmann constant in J/K
    t0: float = 290  # Reference temperature in K
```

Select the Site

We select the site that we want to use for the link budget by setting the variable SiteLinkParameters to the name of one of the desired dataclasses defined above.

In [216…

```
SiteLinkParameters = DSESLinkParameters
```

What Sites Might be Involved?

Let's take a moment to talk about what sites are involved in this sort of amateur radio and amateur radio astronomy work.

Dwingeloo Radio Observatory (Netherlands)

Completed in 1956, the Dwingeloo Radio Telescope features a 25-meter dish that was briefly the world's largest fully steerable radio telescope. After ending scientific operations in 2000, it was designated as a national heritage site in 2009. Since 2007, the C.A. Muller Radio Astronomy Station (CAMRAS) foundation has restored and operates the telescope for amateur radio astronomy and Earth-Moon-Earth (EME) communication. Today, Dwingeloo stands as the world's largest radio telescope for amateur use, with volunteers conducting radio astronomy observations, supporting educational outreach, and participating in special projects like visual moonbounce. This information is from Wikipedia and CAMRAS.

Stockert Radio Telescope (Germany)

Inaugurated on September 17, 1956, the 25-meter Stockert dish was Germany's first telescope for radio astronomy. After serving the University of Bonn and Max Planck Institute until 1995, it gained historical monument status in 1999. Since 2005, the telescope has been owned by the Nordrhein-Westfalen-Stiftung and is maintained by Astropeiler Stockert e.V., a volunteer association that proudly calls it "the largest and most capable radio telescope in the world operated by amateurs." Astropeiler The facility is now equipped with modern technology for scientific observations, student education, and public outreach. This information is from Wikipedia and Stockert.

Deep Space Exploration Society (DSES) (Colorado, USA)

The DSES is a Colorado-based nonprofit organization dedicated to practical astronomy and space science education. Its primary facility is a restored 60-foot dish antenna located in Kiowa County, Colorado. Known as the Paul Plishner Radio Astronomy and Space Sciences Center, the site features a former National Bureau of Standards antenna originally built for tropospheric propagation research between 1957-1974. In addition to radio astronomy research, DSES uses its 60-foot dish for amateur radio EME (moonbounce) activities on the 432 and 1296 MHz bands. This information is from Wikipedia and DSES.

Bochum Observatory (Germany)

Founded in 1946 by Professor Heinz Kaminski as a popular observatory, Bochum Observatory gained international recognition after detecting signals from Sputnik 1 in 1957. Its 20-meter parabolic antenna, inaugurated in 1965, became famous during the 1957-1975 period for receiving transmissions from Russian and American space vehicles, including Apollo missions. Today, the facility participates in research projects

with NASA and DLR, receiving solar data from space probes while also serving as an educational center focusing on sustainability, climate change, and sky observation. This information is from Wikipedia and Bochum.

These historic sites represent the essential bridge between professional radio astronomy and amateur radio enthusiasts, making significant contributions to astronomical research, space communication, and public education while preserving important scientific heritage. When we say Dwingeloo, Stockert, DSES, or Bochum, we are talking about specific station configurations that are located at these sites. Each site has multiple configurations for a wide variety of scientific and communications targets.

System Noise Temperature Worksheet

Introduction

The system noise temperature is an important factor in determining the sensitivity of a radio telescope or communication system. It represents the total noise from all sources that affects the system's ability to detect weak signals. It is measured in Kelvin (K).

Unlike signal strength, which scales with dish diameter, system noise temperature is largely independent of the physical size of the antenna. Instead, it depends on factors related to the quality of the antenna construction, receiver electronics, and environmental conditions.

Key Components

This system noise temperature calculation has the following components

1. Sky Noise (T_sky): Background radiation from the whole universe anda model of atmospheric contributions. This varies with elevation angle (more atmosphere at lower angles) and weather conditions (clear, cloudy, rainy).

2. Spillover Noise (T_spillover): Noise caused by the antenna feed pattern transmitting energy beyond the dish edges. This means it also picks up thermal radiation from the ground. This is determined by the feed design and positioning.

3. Scatter Noise (T_scatter): Noise resulting from dish surface imperfections that scatter incoming signals. Calculated using the Ruze equation, which relates surface RMS errors to performance degradation at a given wavelength.

4. Receiver Noise (T_receiver): Noise generated by the receiver's electronic components, often specified as noise figure or noise temperature.

Total System Temperature

The total system noise temperature is calculated as:

T_sys = T_ant + T_receiver

Where T_ant (the antenna temperature) is the combination of sky noise, spillover noise, and scatter noise:

T_ant = (main_beam_efficiency * T_sky) + T_spillover + T_scatter

Impact on Performance

A lower system noise temperature directly translates to better sensitivity. For deep space communications, minimizing each noise component is essential. The best way to do this is to use high-quality low-noise amplifiers (LNAs). The noise from the LNA is the most significant factor at the receiver. Precise dish surfaces minimize scatter noise. If they're out of round or warped, then there's a degradation. Well-designed feeds reduce spillover noise. The feed needs to be matched as well as it can be to the dish dimension. Operating at higher elevation angles when possible reduces sky noise because we're going through less of the atmosphere.

This worksheet implements some calculations for each of these noise components. The goal is to produce a realistic system performance assessment, and to provide a solid system noise temperature to the Link Budget calculation.

In [217…

```
#import numpy as np
#from dataclasses import dataclass
#from typing import Optional, Dict, Any

class SystemNoiseTemperature:
    """
    A class to calculate system noise temperature for radio systems.
    """

    def __init__(self, params: 'SiteLinkParameters'):
        """
        Initialize the system noise temperature calculator.

        Parameters:
        params (SiteLinkParameters): The link parameters object containing s
        """
        self.params = params

    def calculate_sky_noise(self, elevation_angle_deg: float, atmospheric_cc
        """
        Calculate sky noise temperature based on frequency and elevation ang

        Parameters:
        elevation_angle_deg (float): Elevation angle in degrees
        atmospheric_conditions (str): Weather conditions ('clear', 'cloudy',

        Returns:
```

```
        float: Sky noise temperature in Kelvin
        """
        # Convert elevation angle to radians
        elev_rad = np.radians(elevation_angle_deg)

        # Basic atmospheric attenuation model
        base_temp = 2.7  # cosmic background radiation

        # Atmospheric contribution increases at lower elevation angles
        air_mass = 1.0 / np.sin(elev_rad)

        # Frequency dependent atmospheric absorption
        freq_ghz = self.params.tx_frequency_mhz / 1000.0
        freq_factor = 0.1 * freq_ghz / 10.0

        # Weather condition factors
        weather_factors = {
            'clear': 1.0,
            'cloudy': 1.5,
            'rain': 3.0
        }

        weather_multiplier = weather_factors.get(atmospheric_conditions, 1.0

        return base_temp + (270 * (1 - np.exp(-freq_factor * air_mass))) * w

    def calculate_spillover_noise(self, spillover_efficiency: float, ground_
        """
        Calculate spillover noise contribution.

        Parameters:
        spillover_efficiency (float): Efficiency of the antenna's spillover
        ground_temp (float): Ground temperature in Kelvin

        Returns:
        float: Spillover noise temperature in Kelvin
        """
        return ground_temp * (1 - spillover_efficiency)

    def calculate_scatter_noise(self, surface_rms_mm: float) -> float:
        """
        Calculate scattering noise due to surface imperfections using the Ru

        Parameters:
        surface_rms_mm (float): Root mean square surface error in mm

        Returns:
        float: Scatter noise temperature in Kelvin
        """
        wavelength_mm = 300000 / self.params.tx_frequency_mhz  # Convert MHz
        surface_efficiency = np.exp(-(4 * np.pi * surface_rms_mm / wavelengt
        return 290 * (1 - surface_efficiency)

    def calculate_system_noise(self,
                               main_beam_efficiency: float,
                               spillover_efficiency: float,
```

```
                              surface_rms_mm: float,
                              receiver_temp: float,
                              elevation_angle_deg: float,
                              atmospheric_conditions: str = 'clear',
                              ground_temp: float = 290.0) -> Dict[str, float
        """
        Calculate total system noise temperature and its components.

        Parameters:
        main_beam_efficiency (float): Main beam efficiency of the antenna (0
        spillover_efficiency (float): Spillover efficiency of the antenna (0
        surface_rms_mm (float): RMS surface error in mm
        receiver_temp (float): Receiver noise temperature in Kelvin
        elevation_angle_deg (float): Elevation angle in degrees
        atmospheric_conditions (str): Weather conditions ('clear', 'cloudy',
        ground_temp (float): Ground temperature in Kelvin

        Returns:
        Dict[str, float]: Dictionary containing total and component temperat
        """
        # Calculate individual components
        sky_noise = self.calculate_sky_noise(elevation_angle_deg, atmospheri
        spillover_noise = self.calculate_spillover_noise(spillover_efficienc
        scatter_noise = self.calculate_scatter_noise(surface_rms_mm)

        # Calculate antenna temperature
        t_ant = (main_beam_efficiency * sky_noise +
                spillover_noise + scatter_noise)

        # Calculate total system temperature
        t_sys = t_ant + receiver_temp

        return {
            'T_sys': t_sys,
            'T_ant': t_ant,
            'T_sky': sky_noise,
            'T_spillover': spillover_noise,
            'T_scatter': scatter_noise,
            'T_receiver': receiver_temp
        }

    def get_noise_temperature_summary(self, **kwargs) -> Dict[str, Any]:
        """
        Returns a comprehensive summary of noise temperature calculations.

        Parameters:
        **kwargs: Keyword arguments that can override default parameters
                 (main_beam_efficiency, spillover_efficiency, surface_rms_mm
                  receiver_temp, elevation_angle_deg, atmospheric_conditions

        Returns:
        Dict[str, Any]: Dictionary with noise temperature results and parame
        """
        # Default values (these could/should be stored in SiteLinkParameters
        defaults = {
            'main_beam_efficiency': 0.69,
```

```
            'spillover_efficiency': 0.95,
            'surface_rms_mm': 0.3,
            'receiver_temp': self.params.lna_noise_figure_K,
            'elevation_angle_deg': 45.0,
            'atmospheric_conditions': 'clear',
            'ground_temp': 290.0
        }

        # Override defaults with any provided keyword arguments.
        # we got a lot of good advice on the defaults, but if we know
        # about some sort of update or change to the defaults, then we
        # provide them to the noise calculator get_noise_temperature_summary
        params = {**defaults, **kwargs}

        # Calculate system noise
        noise_temps = self.calculate_system_noise(
            main_beam_efficiency=params['main_beam_efficiency'],
            spillover_efficiency=params['spillover_efficiency'],
            surface_rms_mm=params['surface_rms_mm'],
            receiver_temp=params['receiver_temp'],
            elevation_angle_deg=params['elevation_angle_deg'],
            atmospheric_conditions=params['atmospheric_conditions'],
            ground_temp=params['ground_temp']
        )

        # Add parameters used for calculation to results
        result = {
            'noise_temperatures': noise_temps,
            'parameters_used': params,
            'frequency_mhz': self.params.tx_frequency_mhz
        }

        return result


# Create calculator
params = SiteLinkParameters()
noise_calculator = SystemNoiseTemperature(params)

# Get full noise temperature analysis with default parameters
results = noise_calculator.get_noise_temperature_summary(
)

# Print results
print(f"System Noise Analysis at {results['frequency_mhz']} MHz:")
print(f"Elevation: {results['parameters_used']['elevation_angle_deg']}°")
print(f"Conditions: {results['parameters_used']['atmospheric_conditions']}")
print("\nNoise Temperatures:")
for key, value in results['noise_temperatures'].items():
    print(f"  {key}: {value:.1f} K")



# Get full noise temperature analysis with compromised session parameters
results = noise_calculator.get_noise_temperature_summary(
    elevation_angle_deg=5.0,  # Lower elevation than default - this is a key
```

```
    atmospheric_conditions='cloudy'  # Worse conditions than default - this
)

# Print results
print(f"System Noise Analysis at {results['frequency_mhz']} MHz:")
print(f"Elevation: {results['parameters_used']['elevation_angle_deg']}°")
print(f"Conditions: {results['parameters_used']['atmospheric_conditions']}")
print("\nNoise Temperatures:")
for key, value in results['noise_temperatures'].items():
    print(f"  {key}: {value:.1f} K")
```

```
System Noise Analysis at 2304.0 MHz:
Elevation: 45.0°
Conditions: clear

Noise Temperatures:
  T_sys: 50.6 K
  T_ant: 22.6 K
  T_sky: 11.4 K
  T_spillover: 14.5 K
  T_scatter: 0.2 K
  T_receiver: 28.0 K
System Noise Analysis at 2304.0 MHz:
Elevation: 5.0°
Conditions: cloudy

Noise Temperatures:
  T_sys: 109.5 K
  T_ant: 81.5 K
  T_sky: 96.8 K
  T_spillover: 14.5 K
  T_scatter: 0.2 K
  T_receiver: 28.0 K
```

EVELinkBudget Class

This link budget class for EVE is called EVELinkBudget. It targets Venus as a reflective surface. We inherit site specific link parameters (SiteNameLinkParameters) and call them "params". We then call upon them inside the link budget class. We then set up all of our Venus specific values in this class and define the functions that we need in order to calculate this specific link budget.

What does this section do? This is where the budgeting happens. We add up all the gains and subtract the losses. This gives power at the receiver. We calculate the noise in our receive bandwidth, and subtract it from our power at the receiver. This gives a carrier to noise ratio (CNR) in dB.

Our communications mode must be able to operate at this CNR or lower in order to close the link. Too low, and the signal cannot be heard at the receiver. We use the CNR result as an input to calculations about candidate communications modes.

Later on, when we use our CNR to evaulate different modes, we'll include an additional margin above this CNR. Sometimes we use 10 dB over the calculated CNR. Sometimes we use 3 dB over the calculated CNR. It depends on the communications goals of a particular mode. Demodulating and decoding digital communications signals requires more "adverse margin" than simply detecting an analog carrier. Both cases are presented later on in this link budget.

We run this section for the minimum distance from Earth to Venus, and then we run it again to get the maximum. Note the large difference.

In [218…

```
# Define the EVE link budget here
# Venus specific
class EVELinkBudget:
    def __init__(self, params: SiteLinkParameters):
        self.params = params

        # Venus characteristics
        self.venus_radius_km = 6051.8    # Venus radius in km
        self.venus_radar_albedo = 0.152  # see Venus radar albedo (Radio Ech
                                         # see Variations in the Radar Cross
                                         # Massachusetts Institute of Techno


    def wavelength(self) -> float:
        """Calculate wavelength in meters from frequency"""
        return self.params.c / (self.params.tx_frequency_mhz * 1e6)

    def venus_reflection_gain(self) -> float:
        """Calculate reflection gain from Venus surface
        Venus radar cross section = π * radius²"""
        # see also https://hamradio.engineering/eme-path-loss-free-space-los
        venus_radius_m = self.venus_radius_km * 1000
        radar_cross_section = np.pi * venus_radius_m**2
        return 10 * np.log10(radar_cross_section)

    def venus_reflection_loss(self) -> float:
        # use radar albedo for Venus to get this loss
        return 10 * np.log10(self.venus_radar_albedo)

    def tx_antenna_gain(self) -> float:
        """Calculate transmitter antenna gain"""
        efficiency = self.params.tx_antenna_efficiency
        diameter = self.params.tx_antenna_diameter_m
        wavelength = self.wavelength()
        return 10 * np.log10(efficiency * (np.pi * diameter / wavelength) **

    def rx_antenna_gain(self) -> float:
        """Calculate receiver antenna gain"""
        efficiency = self.params.rx_antenna_efficiency
        diameter = self.params.rx_antenna_diameter_m
        wavelength = self.wavelength()
        return 10 * np.log10(efficiency * (np.pi * diameter / wavelength) **
```

```
    def free_space_loss(self, distance_km: float, round_trip: bool = True) -
        """Calculate free space loss, optionally for round trip

        Args:
            distance_km: Distance in kilometers
            round_trip: If True, calculate round trip loss (both directions)
        """
        wavelength = self.wavelength()
        distance_m = distance_km * 1000
        one_way_loss = 20 * np.log10(4 * np.pi * distance_m / wavelength)
        return one_way_loss * 2 if round_trip else one_way_loss

    def pointing_loss(self) -> float:
        """Calculate pointing loss"""
        pointing_error_rad = np.radians(self.params.pointing_error_deg)
        #tracking_error_rad = np.radians(self.params.tracking_error_deg) # i
        total_error_rad = np.sqrt(pointing_error_rad**2)
        #total_error_rad = np.sqrt(pointing_error_rad**2 + tracking_error_ra
        return -12 * (total_error_rad / self.antenna_beamwidth_rad())**2

    def antenna_beamwidth_rad(self) -> float:
        """Calculate antenna beamwidth in radians"""
        return 1.22 * self.wavelength() / self.params.tx_antenna_diameter_m

    def calculate_link_budget(self, distance_km: float) -> dict:
        """Calculate complete link budget for a given distance"""
        # Convert transmit power to dBW
        tx_power_dbw = 10 * np.log10(self.params.tx_power_w)

        # Calculate antenna gains (only once each for TX and RX)
        tx_gain = self.tx_antenna_gain()
        rx_gain = self.rx_antenna_gain()
        venus_gain = self.venus_reflection_gain()

        # Calculate losses
        fs_loss = self.free_space_loss(distance_km, round_trip=True)
        point_loss = self.pointing_loss()
        venus_loss = self.venus_reflection_loss()

        # Calculate received power (passive reflection scenario)
        rx_power = (
            tx_power_dbw
            + tx_gain  # TX antenna gain
            + rx_gain  # RX antenna gain
            - fs_loss  # Two-way path loss
            - self.params.tx_line_loss_db  # reported TX line loss at site
            - self.params.rx_line_loss_db  # reported RX line loss at site
            - point_loss  # Pointing loss
            - venus_loss  # Venus reflection loss
            + venus_gain  # Venus reflection gain
        )



        # Get full noise temperature analysis from our noise_calculator and
        results = noise_calculator.get_noise_temperature_summary(
            #elevation_angle_deg=5.0,  # Lower elevation than default - this
```

```
            #atmospheric_conditions='cloudy'  # Worse conditions than defaul
        )
        t_sys = results['noise_temperatures']['T_sys']

        noise_dbw = 10 * np.log10(self.params.k * t_sys * self.params.receiv


        # Calculate CNR
        cnr = rx_power - noise_dbw

        # Calculate CNR in 1Hz
        bandwidth_factor_db = 10 * np.log10(self.params.receiver_noise_bandw
        cnr_db_1hz = cnr + bandwidth_factor_db

        return {
            'tx_power_dbw': tx_power_dbw,
            'radius_venus_km': self.venus_radius_km,
            'venus_radar_albedo': self.venus_radar_albedo,
            'tx_gain_db': tx_gain,
            'rx_gain_db': rx_gain,
            'free_space_loss_db': fs_loss,
            'pointing_loss_db': point_loss,
            'venus_reflection_loss_db': venus_loss,
            'venus_reflection_gain_db': venus_gain,
            'system_noise_temperature': t_sys,
            'rx_power_dbw': rx_power,
            'noise_dbw': noise_dbw,
            'cnr_db': cnr,
            'cnr_db_1hz': cnr_db_1hz
        }
```

Link Budget Calculator Setup

We fetch all the siteLinkParameters, put them in a variable named `params`, and then create a calculator instance with `calculator = EVELinkBudget(params)`. We've specified the class of EVELinkBudget above. We're now creating the object that we use to call the various methods inside of the object's class. One of these methods is `calculate_link_budget()`. Since the distance to Venus varies quite a bit, this is a parameter we need to provide in order to get a useful result.

Whenever we need to calculate the CNR for our link, we call `our_results = calculator.calculate_link_budget(distance to Venus)`.

And, `our_results['cnr_db_1hz']` is our 1 Hz CNR.

In [219…

```
# Cell 2: Create calculator instance
params = SiteLinkParameters()
calculator = EVELinkBudget(params)

min_distance_km = 38_000_000  # Minimum Earth-Venus distance
```

```
max_distance_km = 261_000_000  # Maximum Earth-Venus distance

# Print detailed results for minimum and maximum distances - can then be use
max_results = calculator.calculate_link_budget(max_distance_km)
min_results = calculator.calculate_link_budget(min_distance_km)

print(f"Link Budget at Minimum Distance (38 million km) at {params.receiver_
for key, value in min_results.items():
    print(f"{key}: {value:.2f}")

print(f"\nLink Budget at Maximum Distance (261 million km) at {params.receiv
for key, value in max_results.items():
    print(f"{key}: {value:.2f}")
```

```
Link Budget at Minimum Distance (38 million km) at 0.1 MHz receiver bandwidt
h
tx_power_dbw: 31.76
radius_venus_km: 6051.80
venus_radar_albedo: 0.15
tx_gain_db: 51.29
rx_gain_db: 51.29
free_space_loss_db: 502.59
pointing_loss_db: -0.00
venus_reflection_loss_db: -8.18
venus_reflection_gain_db: 140.61
system_noise_temperature: 50.56
rx_power_dbw: -220.45
noise_dbw: -161.56
cnr_db: -58.89
cnr_db_1hz: -8.89

Link Budget at Maximum Distance (261 million km) at 0.1 MHz receiver bandwid
th
tx_power_dbw: 31.76
radius_venus_km: 6051.80
venus_radar_albedo: 0.15
tx_gain_db: 51.29
rx_gain_db: 51.29
free_space_loss_db: 536.06
pointing_loss_db: -0.00
venus_reflection_loss_db: -8.18
venus_reflection_gain_db: 140.61
system_noise_temperature: 50.56
rx_power_dbw: -253.93
noise_dbw: -161.56
cnr_db: -92.36
cnr_db_1hz: -42.36
```

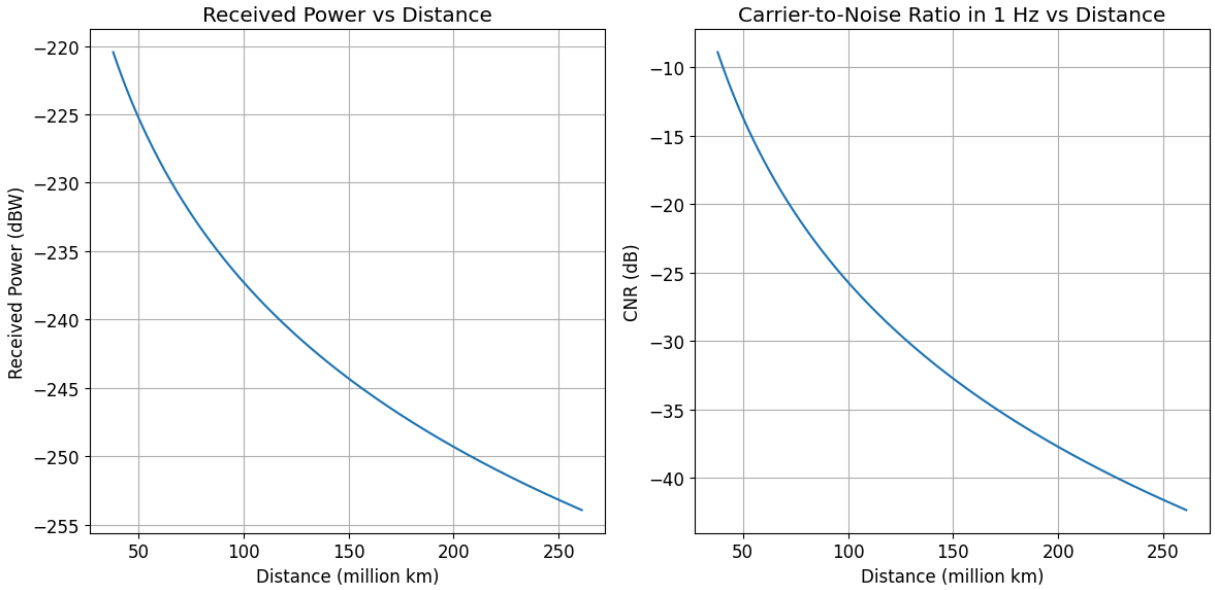
Effect of Distance on Received Power and Carrier-to-Noise Ratio

The large variation in distance from Earth to Venus results in a large variation in the received power at the site and in the carrier to noise ratio at the site. This section

creates a visualization that shows the differences between the closest path and the furthest path for radio work.

In [220…

```
# Create distance array for plotting
distances = np.linspace(min_distance_km, max_distance_km, 1000)
cnrs = []
rx_powers = []

for dist in distances:
    results = calculator.calculate_link_budget(dist)
    cnrs.append(results['cnr_db_1hz'])
    rx_powers.append(results['rx_power_dbw'])

# Cell 4: Create plots
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.plot(distances/1e6, rx_powers)
plt.grid(True)
plt.xlabel('Distance (million km)')
plt.ylabel('Received Power (dBW)')
plt.title('Received Power vs Distance')

plt.subplot(1, 2, 2)
plt.plot(distances/1e6, cnrs)
plt.grid(True)
plt.xlabel('Distance (million km)')
plt.ylabel('CNR (dB)')
plt.title('Carrier-to-Noise Ratio in 1 Hz vs Distance')
plt.tight_layout()
plt.show()
```



Pointing Error Analysis

Dish antennas have a narrow beamwidth, and pointing errors can significantly impact our link budget. This section models pointing errors to determine their impact on signal

strength and to help us include appropriate loss values in our link budget calculations.

Visualizations

We provide two visualizations in the link budget.

The first is a normalized visualization (error/beamwidth ratio). It provides a universal reference applicable to any dish size or frequency. It illustrates the fundamental relationship between pointing error and beamwidth. It demonstrates key principles: 1dB loss occurs at error = beamwidth/3.46, 3dB loss at error = beamwidth/2. It facilitates comparisons across different systems.

The second is an absolute visualization (error in degrees). It shows practical values specific to (for example) the DSES dish at 18.29m dish at 1296 MHz. It provides exact specifications for pointing requirements. It displays precisely how many degrees of error are acceptable for given loss levels. It is directly applicable to a specific system's operational planning.

The notebook user can toggle between these visualizations using the show_normalized parameter (True/False).

The calculations are based on established antenna theory. First, there is a beamwidth calculation. It uses the formula beamwidth = 1.22 * lambda/D for circular apertures. lambda is wavelength in meters and D is dish diameter in meters. The result is the 3dB beamwidth (where power drops to half).

Pointing Loss Formula: pointing_loss = -12 * (error_angle / beamwidth)² This quadratic relationship means losses increase rapidly as pointing error grows.

Critical Error Thresholds

Working backwards from the formula above we get the following: For 1dB loss: error = beamwidth/$\sqrt{12}$ ≈ beamwidth/3.46 For 3dB loss: error = beamwidth/2

For example, with a beamwidth of 0.6°: Maximum error for 1dB loss: 0.173° (0.6°/3.46) Maximum error for 3dB loss: 0.3° (0.6°/2)

The current system's pointing error (for example, 0.01°) is displayed on the graph as a purple dot, showing its position relative to these critical thresholds. This is a reasonable engineering guideline for minimizing losses, though it's more conservative than the 1dB loss threshold.

In [221…

```
import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass

class AntennaPointingAnalysis:
```

```
    def __init__(self, params: SiteLinkParameters):
        self.params = params

    def wavelength(self) -> float:
        """Calculate wavelength in meters from frequency in MHz"""
        frequency_hz = self.params.tx_frequency_mhz * 1e6  # Convert MHz to
        return self.params.c / frequency_hz

    def beamwidth_deg(self) -> float:
        """Calculate 3dB beamwidth in degrees"""
        # Using 1.22 λ/D formula for circular aperture
        return np.degrees(1.22 * self.wavelength() / self.params.tx_antenna_

    def calculate_pointing_loss(self, error_angle, beamwidth):
        """Calculate pointing loss in dB given error angle and beamwidth (bo
        return -12 * (error_angle / beamwidth)**2

    def pointing_loss_analysis(self, max_error_fraction=1.0):
        """Generate pointing loss analysis with normalized and absolute erro
        # Get beamwidth
        beamwidth = self.beamwidth_deg()

        # Create normalized error fractions (as portion of beamwidth)
        error_fractions = np.linspace(0, max_error_fraction, 100)

        # Calculate absolute error values in degrees
        error_degrees = error_fractions * beamwidth

        # Calculate losses for the normalized curve
        losses = self.calculate_pointing_loss(error_fractions, 1)

        # Calculate key reference points
        # CORRECTION: Changed 5.66 to 3.46 (√12 ≈ 3.46)
        loss_at_sqrt12 = self.calculate_pointing_loss(1/np.sqrt(12), 1)  # ~
        loss_at_half = self.calculate_pointing_loss(1/2, 1)    # ~3 dB

        # Current system pointing error and its loss
        current_error = self.params.pointing_error_deg
        current_error_fraction = current_error / beamwidth
        current_loss = self.calculate_pointing_loss(current_error, beamwidth

        # Summary information
        summary = {
            'beamwidth': beamwidth,
            # CORRECTION: Changed 5.66 to np.sqrt(12) for accuracy
            'error_for_1db_loss': beamwidth/np.sqrt(12),
            'error_for_3db_loss': beamwidth/2,
            'loss_at_sqrt12': loss_at_sqrt12,
            'loss_at_half': loss_at_half,
            'error_fractions': error_fractions,
            'error_degrees': error_degrees,
            'losses': losses,
            'current_error': current_error,
            'current_error_fraction': current_error_fraction,
            'current_loss': current_loss
        }
```

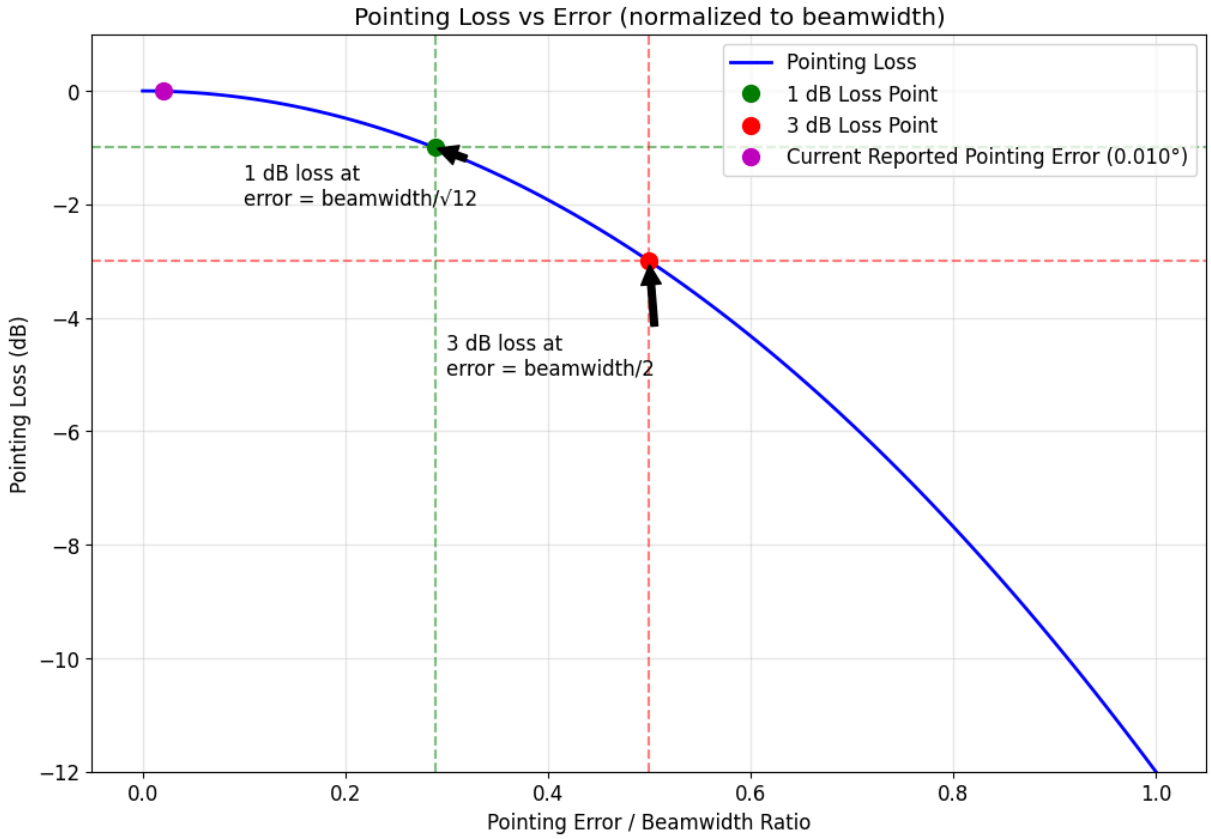
```
        return summary

    def plot_pointing_loss(self, show_normalized=True):
        """Create comprehensive pointing loss visualization"""
        # Get analysis data
        data = self.pointing_loss_analysis(max_error_fraction=1.0)

        # Create figure
        fig, ax = plt.subplots(figsize=(12, 8))

        if show_normalized:
            # Plot normalized curve (error as fraction of beamwidth)
            ax.plot(data['error_fractions'], data['losses'], 'b-', linewidth

            # CORRECTION: Changed 5.66 to np.sqrt(12) for accuracy
            sqrt12_value = 1/np.sqrt(12)
            # Add markers for specific loss points
            ax.plot(sqrt12_value, data['loss_at_sqrt12'], 'go', markersize=1
            ax.plot(1/2, data['loss_at_half'], 'ro', markersize=10, label='3

            # Add point for current system pointing error
            ax.plot(data['current_error_fraction'], data['current_loss'], 'm
                    markersize=10, label='Current Reported Pointing Error (%

            # Add horizontal and vertical reference lines
            ax.axhline(y=data['loss_at_sqrt12'], color='g', linestyle='--',
            ax.axhline(y=data['loss_at_half'], color='r', linestyle='--', al
            ax.axvline(x=sqrt12_value, color='g', linestyle='--', alpha=0.5)
            ax.axvline(x=1/2, color='r', linestyle='--', alpha=0.5)

            # Add annotations
            # CORRECTION: Updated annotation text to use correct formula
            ax.annotate('1 dB loss at\nerror = beamwidth/√12',
                        xy=(sqrt12_value, data['loss_at_sqrt12']), xytext=(0
                        arrowprops=dict(facecolor='black', shrink=0.05))
            ax.annotate('3 dB loss at\nerror = beamwidth/2',
                        xy=(1/2, data['loss_at_half']), xytext=(0.3, -5),
                        arrowprops=dict(facecolor='black', shrink=0.05))

            # Set axis labels
            ax.set_xlabel('Pointing Error / Beamwidth Ratio')
            title = 'Pointing Loss vs Error (normalized to beamwidth)'
        else:
            # Plot absolute curve (error in degrees)
            ax.plot(data['error_degrees'], data['losses'], 'b-', linewidth=2

            # Add markers for specific loss points
            ax.plot(data['error_for_1db_loss'], data['loss_at_sqrt12'], 'go'
            ax.plot(data['error_for_3db_loss'], data['loss_at_half'], 'ro',

            # Add point for current system pointing error
            ax.plot(data['current_error'], data['current_loss'], 'mo',
                    markersize=10, label='Current Reported Pointing Error (%

            # Add horizontal and vertical reference lines
```

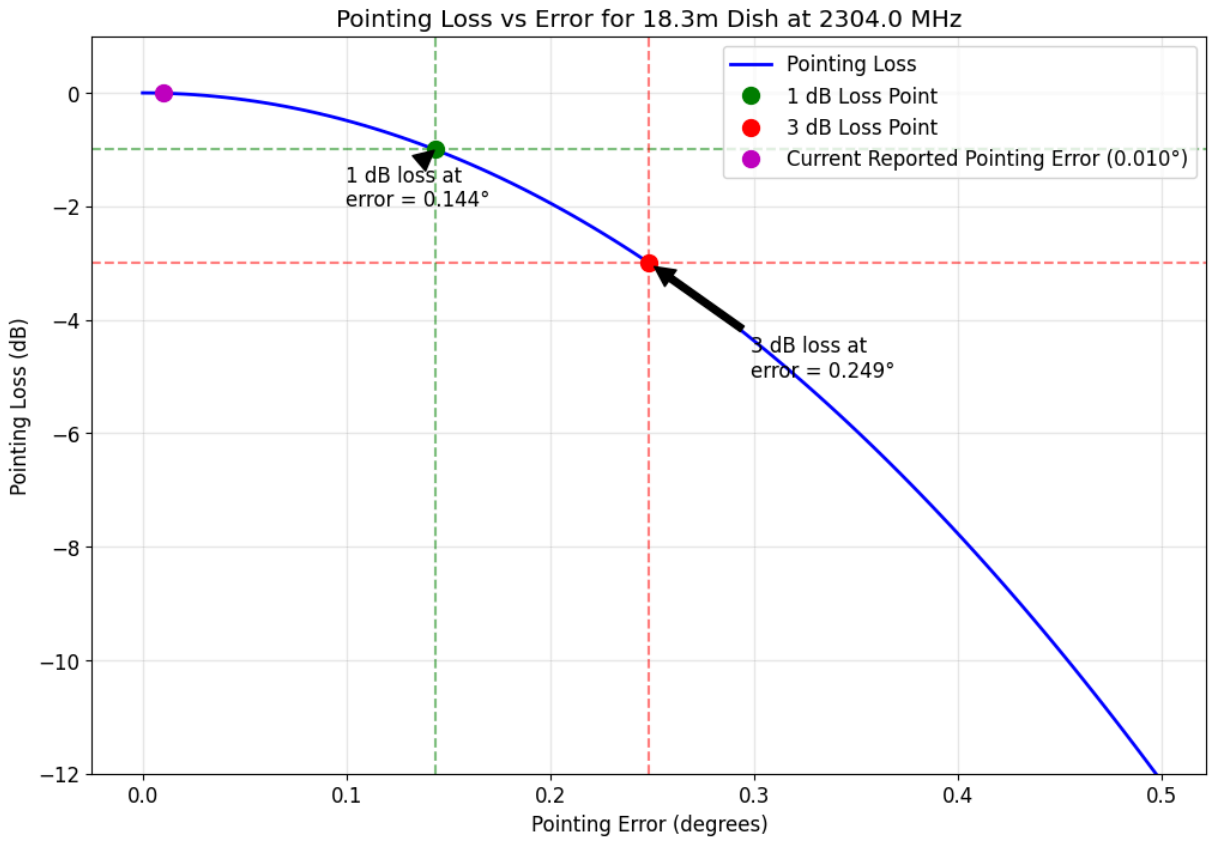
```
            ax.axhline(y=data['loss_at_sqrt12'], color='g', linestyle='--',
            ax.axhline(y=data['loss_at_half'], color='r', linestyle='--', al
            ax.axvline(x=data['error_for_1db_loss'], color='g', linestyle='-
            ax.axvline(x=data['error_for_3db_loss'], color='r', linestyle='-

            # Add annotations
            ax.annotate(f'1 dB loss at\nerror = {data["error_for_1db_loss"]:
                        xy=(data['error_for_1db_loss'], data['loss_at_sqrt12
                        xytext=(data['beamwidth']*0.2, -2),
                        arrowprops=dict(facecolor='black', shrink=0.05))
            ax.annotate(f'3 dB loss at\nerror = {data["error_for_3db_loss"]:
                        xy=(data['error_for_3db_loss'], data['loss_at_half']
                        xytext=(data['beamwidth']*0.6, -5),
                        arrowprops=dict(facecolor='black', shrink=0.05))

            # Set axis labels
            ax.set_xlabel('Pointing Error (degrees)')
            title = f'Pointing Loss vs Error for {self.params.tx_antenna_dia


        # Get current loss value to adjust y-limits if needed
        current_loss = data['current_loss']

        # Calculate appropriate y-limits
        # Start with default range
        y_min = -12
        y_max = 0.5  # Give a bit more space at the top

        # Adjust if current loss is very close to zero
        if current_loss > -0.5:
            y_max = 1  # Give even more space at the top

        # Common formatting
        ax.set_ylabel('Pointing Loss (dB)')
        ax.set_title(title)
        ax.grid(True, alpha=0.3)
        ax.set_ylim(y_min, y_max)  # Use the calculated limits
        ax.legend()


        # Print summary information
        print(f"For a {self.params.tx_antenna_diameter_m:.1f}m dish at {self
        print(f"3dB Beamwidth: {data['beamwidth']:.3f} degrees")
        print(f"Maximum pointing error for 1dB loss: {data['error_for_1db_lo
        print(f"Maximum pointing error for 3dB loss: {data['error_for_3db_lo
        print(f"Current system pointing error: {data['current_error']:.3f} d
        print(f"Current pointing loss: {data['current_loss']:.2f} dB")

        return fig, ax

# Set up the calculator
params = SiteLinkParameters()
analyzer = AntennaPointingAnalysis(params)
#
# # Show normalized plot (relative to beamwidth)
analyzer.plot_pointing_loss(show_normalized=True)
```

```
#
# # Show absolute plot (in degrees)
analyzer.plot_pointing_loss(show_normalized=False)
```

```
For a 18.3m dish at 2304.0 MHz:
3dB Beamwidth: 0.497 degrees
Maximum pointing error for 1dB loss: 0.144 degrees
Maximum pointing error for 3dB loss: 0.249 degrees
Current system pointing error: 0.010 degrees
Current pointing loss: -0.00 dB
For a 18.3m dish at 2304.0 MHz:
3dB Beamwidth: 0.497 degrees
Maximum pointing error for 1dB loss: 0.144 degrees
Maximum pointing error for 3dB loss: 0.249 degrees
Current system pointing error: 0.010 degrees
Current pointing loss: -0.00 dB
```

Out[221… (<Figure size 1200x800 with 1 Axes>,
<Axes: title={'center': 'Pointing Loss vs Error for 18.3m Dish at 2304.0 MHz'}, xlabel='Pointing Error (degrees)', ylabel='Pointing Loss (dB)'>)





How much Doppler do we Have?

How much Doppler do we have to deal with? This section calculates Doppler shift and the rate of change of the Doppler shift, and visualizes the results.

We will have to anticipate and "track out" Doppler shift in order to find our signal in the frequency domain. We need to understand and account for the rate of change of this Doppler shift as well. A third factor is Doppler spread, which is what happens when a signal bounces off a rotating reflector. The DopplerCalculator class calculates Doppler shifts at particular times from particular positions. It calculates worst case Doppler shifts and worst case rate of change of Doppler.

Doppler shift is lowest at inferior conjunction with Earth. This is when Venus is closest to us. Doppler shift briefly passes through zero Hz. At this point in the orbit, however, the rate of change of Doppler shift is the highest. The visualizations in this section show what this looks like. In the example usage, we first calculate Doppler values as if we at the center of the Earth. Why do this? Because we can show what is going on as if we were able to hold still from a non-rotating point of view. After that, we use our location specific methods. We are now on the surface of the Earth. We have our particular radio site from the SiteLinkParameters. We set up two more sites for comparison and to show how to change location. Adding in the rotation of the earth changes our Doppler situation and the visibility of Venus.

Doppler spread and Doppler spread penalty calculations have their own calculator and methods.

In [222…

```
#import numpy as np
#from skyfield.api import load, wgs84
#from datetime import datetime, timedelta, timezone
#import matplotlib.pyplot as plt

class DopplerCalculator:
    """
    A class to calculate Doppler shifts between Earth and Venus.

    This class uses the Skyfield library to obtain planetary ephemeris data
    from publicly available sources and calculate the relative velocity and
    resulting Doppler shift and shift rate for radio communications.
    """

    def __init__(self, params: SiteLinkParameters):
        self.params = params
        """
        Initialize the Doppler calculator with one of our site-specific data

        Parameters:
        -----------
        tx_frequency_mhz : float from <site>LinkParameters
            The transmission frequency in MHz (eg. 1296.0 MHz)
        """

        # Convert MHz to Hz for calculations
        self.frequency_hz = self.params.tx_frequency_mhz * 1e6

        # Time span of visualizations in days
        self.duration = 584      # 30 is useful as well

        # Load ephemeris data
        self.ephemeris = load('de421.bsp')
        self.earth = self.ephemeris['earth']
        self.venus = self.ephemeris['venus']

        # Time scale
        self.ts = load.timescale()

        # Observer location (initially None)
        self.has_observer = False

    def set_frequency(self, frequency_mhz):
        """
        Update the transmission frequency, outside of the dataclass.

        Parameters:
        -----------
        frequency_mhz : float
            The new transmission frequency in MHz
        """
        self.frequency_hz = frequency_mhz * 1e6
```

```
    def calculate_doppler(self, timestamp=None):
        """
        Calculate the Doppler shift at a specific time.

        Parameters:
        -----------
        timestamp : datetime, optional
            The time at which to calculate the Doppler shift (default: curre
            If timezone-naive, UTC will be used

        Returns:
        --------
        tuple:
            - Frequency shift in Hz
            - Received frequency in Hz
            - Relative velocity in m/s (positive: moving apart, negative: mo
        """
        if timestamp is None:
            # Make sure we use timezone-aware UTC time
            timestamp = datetime.now(timezone.utc)
        elif timestamp.tzinfo is None:
            # If the timestamp is naive (no timezone), assume it's in UTC
            timestamp = timestamp.replace(tzinfo=timezone.utc)

        # Convert to Skyfield time. This is called a "decomposed" approach
        t = self.ts.utc(timestamp.year, timestamp.month, timestamp.day,
                        timestamp.hour, timestamp.minute, timestamp.second +

        # Get the position and velocity vectors
        earth_pos = self.earth.at(t)
        venus_pos = self.venus.at(t)

        # Calculate relative position and velocity
        relative = earth_pos.observe(self.venus)
        distance = relative.distance().km
        relative_velocity = relative.velocity.km_per_s

        # The radial velocity component (positive means moving apart)
        radial_velocity = np.dot(relative_velocity, relative.position.km / d

        # Convert km/s to m/s
        radial_velocity_m_s = radial_velocity * 1000

        # Calculate Doppler shift
        # If objects are moving apart (positive velocity), frequency decreas
        doppler_shift = -self.frequency_hz * radial_velocity_m_s / self.para
        received_frequency = self.frequency_hz + doppler_shift

        return doppler_shift, received_frequency, radial_velocity_m_s

    def calculate_doppler_rate(self, timestamp=None, delta_hours=24):
        """
        Calculate the rate of change of the Doppler shift over a specified t

        Parameters:
```

```
        ----------
        timestamp : datetime, optional
            The time at which to start the calculation (default: current tim
            If timezone-naive, UTC will be assumed
        delta_hours : float, optional
            The time period in hours over which to calculate the rate (defau

        Returns:
        --------
        tuple:
            - Rate of Doppler shift change in Hz/hour
            - Rate of Doppler shift change in Hz/second
            - Doppler shift at start time in Hz
            - Doppler shift at end time in Hz
        """
        if timestamp is None:
            timestamp = datetime.now(timezone.utc)
        elif timestamp.tzinfo is None:
            timestamp = timestamp.replace(tzinfo=timezone.utc)

        # Calculate Doppler shift at the start time
        doppler_start, _, _ = self.calculate_doppler(timestamp)

        # Calculate Doppler shift at the end time
        end_time = timestamp + timedelta(hours=delta_hours)
        doppler_end, _, _ = self.calculate_doppler(end_time)

        # Calculate the rate of change
        delta_doppler = doppler_end - doppler_start
        rate_per_hour = delta_doppler / delta_hours
        rate_per_second = rate_per_hour / 3600

        return rate_per_hour, rate_per_second, doppler_start, doppler_end

    def generate_doppler_curve(self, start_date, end_date, num_points=100):
        """
        Generate a curve of Doppler shift over a time period.

        Parameters:
        -----------
        start_date : datetime
            The starting date for the curve (if timezone-naive, UTC will be
        end_date : datetime
            The ending date for the curve (if timezone-naive, UTC will be as
        num_points : int, optional
            Number of points to calculate (default: 100)

        Returns:
        --------
        tuple:
            - List of datetime objects
            - List of Doppler shifts in Hz
            - List of relative velocities in m/s
        """
        # Ensure both dates have timezone information
        if start_date.tzinfo is None:
```

```
            start_date = start_date.replace(tzinfo=timezone.utc)
        if end_date.tzinfo is None:
            end_date = end_date.replace(tzinfo=timezone.utc)

        time_delta = (end_date - start_date) / num_points

        times = []
        doppler_shifts = []
        velocities = []

        for i in range(num_points + 1):
            current_time = start_date + i * time_delta
            try:
                doppler_shift, _, velocity = self.calculate_doppler(current_

                times.append(current_time)
                doppler_shifts.append(doppler_shift)
                velocities.append(velocity)
            except Exception as e:
                print(f"Error calculating Doppler at time {current_time}: {e
                # Continue with the loop but skip this problematic point
                continue

        return times, doppler_shifts, velocities

    def calculate_worst_case_doppler(self, start_date=None, duration_days=58
        """
        Calculate the worst-case Doppler shift and rate of change over a com

        The synodic period of Venus is approximately 584 days.
        This is the time it takes for Earth and Venus to return
        to the same relative positions - another inferior conjunction.

        Parameters:
        -----------
        start_date : datetime, optional
            The starting date for the analysis (default: current time)
        duration_days : int, optional
            The duration in days to analyze (default: 584, one synodic perio

        Returns:
        --------
        dict:
            Dictionary containing:
            - max_doppler_shift: Maximum absolute Doppler shift in Hz
            - min_doppler_shift: Minimum Doppler shift in Hz (most negative)
            - max_doppler_shift_date: Date of maximum Doppler shift
            - min_doppler_shift_date: Date of minimum Doppler shift
            - max_rate: Maximum absolute rate of change in Hz/hour
            - max_rate_date: Date of maximum rate of change
            - max_rate_per_second: Maximum rate in Hz/second
            - total_doppler_range: Total range of Doppler shift in Hz
        """
        if start_date is None:
            start_date = datetime.now(timezone.utc)
        elif start_date.tzinfo is None:
```

```
            start_date = start_date.replace(tzinfo=timezone.utc)

        end_date = start_date + timedelta(days=duration_days)

        # Use enough points to get good resolution (at least one point per c
        num_points = max(1000, duration_days * 4)

        times, shifts, velocities = self.generate_doppler_curve(start_date,

        # Calculate rates of change
        rates = []
        rate_times = []

        for i in range(len(shifts) - 1):
            delta_t = (times[i+1] - times[i]).total_seconds() / 3600  # in h
            rate = (shifts[i+1] - shifts[i]) / delta_t  # Hz per hour
            rates.append(rate)
            rate_times.append(times[i])

        # Find extremes for Doppler shift
        max_shift_idx = np.argmax(shifts)
        min_shift_idx = np.argmin(shifts)
        max_shift = shifts[max_shift_idx]
        min_shift = shifts[min_shift_idx]
        max_shift_date = times[max_shift_idx]
        min_shift_date = times[min_shift_idx]

        # Find extreme for rate of change (maximum absolute value)
        abs_rates = np.abs(rates)
        max_rate_idx = np.argmax(abs_rates)
        max_rate = rates[max_rate_idx]
        max_rate_date = rate_times[max_rate_idx]
        max_rate_per_second = max_rate / 3600

        # Calculate total Doppler range
        total_doppler_range = max_shift - min_shift

        return {
            'max_doppler_shift': max_shift,
            'min_doppler_shift': min_shift,
            'max_doppler_shift_date': max_shift_date,
            'min_doppler_shift_date': min_shift_date,
            'max_rate': max_rate,
            'max_rate_date': max_rate_date,
            'max_rate_per_second': max_rate_per_second,
            'total_doppler_range': total_doppler_range
        }

    def plot_doppler_curve(self, start_date, end_date, num_points=100, save_
        """
        Generate and plot a curve of Doppler shift over a time period.

        Parameters:
        -----------
        start_date : datetime
            The starting date for the curve
```

```
        end_date : datetime
            The ending date for the curve
        num_points : int, optional
            Number of points to calculate (default: 100)
        save_path : str, optional
            Path to save the plot (default: None, plot is displayed instead)
        include_rate : bool, optional
            Whether to include a subplot showing the rate of change (default
        """
        times, shifts, velocities = self.generate_doppler_curve(start_date,

        # Calculate rate of change for each point except the last one
        rates = []
        for i in range(len(shifts) - 1):
            delta_t = (times[i+1] - times[i]).total_seconds() / 3600  # in h
            rate = (shifts[i+1] - shifts[i]) / delta_t  # Hz per hour
            rates.append(rate)

        # Add a placeholder for the last point to keep the arrays the same l
        rates.append(rates[-1] if rates else 0)

        # Determine the number of subplots
        n_plots = 3 if include_rate else 2

        fig, axes = plt.subplots(n_plots, 1, figsize=(12, 4*n_plots), sharex

        # Plot Doppler shift
        axes[0].plot(times, [s/1000 for s in shifts], 'b-')
        axes[0].set_ylabel('Doppler Shift (kHz)')
        axes[0].set_title(f'Earth-Venus Doppler Shift at {self.frequency_hz/
        axes[0].grid(True)

        # Plot velocity
        axes[1].plot(times, velocities, 'r-')
        axes[1].set_ylabel('Relative Velocity (m/s)')
        axes[1].set_title('Earth-Venus Relative Velocity')
        axes[1].grid(True)

        # Plot rate of change if requested
        if include_rate:
            axes[2].plot(times, rates, 'g-')
            axes[2].set_xlabel('Date')
            axes[2].set_ylabel('Rate of Change (Hz/hour)')
            axes[2].set_title('Doppler Shift Rate of Change')
            axes[2].grid(True)
        else:
            axes[1].set_xlabel('Date')

        plt.tight_layout()

        if save_path:
            plt.savefig(save_path)
        else:
            plt.show()
```

```
    # fixed this with a lot of help - it wasn't updating past the first loca
    def set_observer_location(self, latitude, longitude, elevation=0, locati
            """
            Set the Earth-based observer's location.

            Parameters:
            -----------
            latitude : float
                Observer's latitude in degrees (positive for north, negative
            longitude : float
                Observer's longitude in degrees (positive for east, negative
            elevation : float, optional
                Observer's elevation above sea level in meters (default: 0)
            location_name : str, optional
                Name of the location (e.g., 'Site 1', 'New York', etc.)
            """
            # Create a completely new observer_location object EVERY time
            self.observer_location = None  # Clear the old one first
            self.observer_location = wgs84.latlon(latitude, longitude, eleva
            self.has_observer = True
            self.location_name = location_name if location_name else f"({lat

            # Force the location to be recreated the next time it's used
            if hasattr(self, '_location_object'):
                del self._location_object


    def calculate_doppler_from_location(self, timestamp=None):
        """
        Calculate the Doppler shift at a specific time from the observer's l
        This includes the additional effect of Earth's rotation.

        Parameters:
        -----------
        timestamp : datetime, optional
            The time at which to calculate the Doppler shift (default: curre
            If timezone-naive, UTC will be assumed

        Returns:
        --------
        tuple:
            - Frequency shift in Hz
            - Received frequency in Hz
            - Relative velocity in m/s
            - Venus altitude in degrees (for visibility determination)
            - Venus azimuth in degrees
        """
        if not self.has_observer:
            raise ValueError("Observer location not set. Use set_observer_lo

        if timestamp is None:
            timestamp = datetime.now(timezone.utc)
        elif timestamp.tzinfo is None:
            timestamp = timestamp.replace(tzinfo=timezone.utc)

        # Convert to Skyfield time
```

```
        t = self.ts.utc(timestamp.year, timestamp.month, timestamp.day,
                        timestamp.hour, timestamp.minute, timestamp.second +

        # Create a Skyfield location object for the observer
        location = self.earth + self.observer_location

        # Get topocentric position (from observer's location)
        venus_apparent = location.at(t).observe(self.venus).apparent()

        # Get altitude and azimuth for visibility determination
        alt, az, distance = venus_apparent.altaz()

        # Get the velocity component
        relative_velocity = venus_apparent.velocity.km_per_s

        # The radial velocity component (positive means moving apart)
        radial_velocity = np.dot(relative_velocity, venus_apparent.position.

        # Convert km/s to m/s
        radial_velocity_m_s = radial_velocity * 1000

        # Calculate Doppler shift
        doppler_shift = -self.frequency_hz * radial_velocity_m_s / self.para
        received_frequency = self.frequency_hz + doppler_shift

        return doppler_shift, received_frequency, radial_velocity_m_s, alt.d

    def is_visible_from_location(self, timestamp=None, min_altitude=10):
        """
        Determine if Venus is visible from the observer's location.

        Parameters:
        -----------
        timestamp : datetime, optional
            The time to check visibility (default: current time)
        min_altitude : float, optional
            Minimum altitude in degrees for visibility (default: 10)

        Returns:
        --------
        bool:
            True if Venus is above the minimum altitude, False otherwise
        """
        if not self.has_observer:
            raise ValueError("Observer location not set. Use set_observer_lo

        _, _, _, altitude, _ = self.calculate_doppler_from_location(timestam
        return altitude > min_altitude

    def generate_location_doppler_curve(self, start_date, end_date, num_poir
                                        include_visibility=True, min_altitude=
        """
        Generate a curve of Doppler shift from the observer's location over

        Parameters:
        -----------
```

```
        start_date, end_date : datetime
            The time range to generate the curve for
        num_points : int, optional
            Number of points to calculate (default: 100)
        include_visibility : bool, optional
            Whether to mask points where Venus is not visible (default: True
        min_altitude : float, optional
            Minimum altitude in degrees for visibility (default: 10)

        Returns:
        --------
        tuple:
            - List of datetime objects
            - List of Doppler shifts in Hz
            - List of relative velocities in m/s
            - List of Venus altitudes in degrees
            - List of visibility flags
        """
        if not self.has_observer:
            raise ValueError("Observer location not set. Use set_observer_lo

        # Ensure both dates have timezone information
        if start_date.tzinfo is None:
            start_date = start_date.replace(tzinfo=timezone.utc)
        if end_date.tzinfo is None:
            end_date = end_date.replace(tzinfo=timezone.utc)

        time_delta = (end_date - start_date) / num_points

        times = []
        doppler_shifts = []
        velocities = []
        altitudes = []
        visibilities = []

        for i in range(num_points + 1):
            current_time = start_date + i * time_delta
            try:
                doppler_shift, _, velocity, alt, _ = self.calculate_doppler_
                is_visible = alt > min_altitude

                times.append(current_time)
                doppler_shifts.append(doppler_shift)
                velocities.append(velocity)
                altitudes.append(alt)
                visibilities.append(is_visible)
            except Exception as e:
                print(f"Error calculating Doppler at time {current_time}: {e
                continue

        # If including visibility, mask points where Venus is not visible
        if include_visibility:
            masked_shifts = []
            masked_velocities = []
            for i in range(len(visibilities)):
                if not visibilities[i]:
```

```
                    # Set to None to create gaps in the plot
                    masked_shifts.append(None)
                    masked_velocities.append(None)
                else:
                    masked_shifts.append(doppler_shifts[i])
                    masked_velocities.append(velocities[i])
            doppler_shifts = masked_shifts
            velocities = masked_velocities

        return times, doppler_shifts, velocities, altitudes, visibilities

    def plot_location_doppler_curve(self, start_date, end_date, num_points=1
                                    save_path=None, include_visibility=True, m
        """
        Generate and plot a curve of Doppler shift from the observer's locat

        Parameters:
        -----------
        start_date, end_date : datetime
            The time range to generate the curve for
        num_points : int, optional
            Number of points to calculate (default: 100)
        save_path : str, optional
            Path to save the plot (default: None, plot is displayed instead)
        include_visibility : bool, optional
            Whether to mask points where Venus is not visible (default: True
        min_altitude : float, optional
            Minimum altitude in degrees for visibility (default: 10)
        """
        if not self.has_observer:
            raise ValueError("Observer location not set. Use set_observer_lo

        times, shifts, velocities, altitudes, visibilities = \
            self.generate_location_doppler_curve(start_date, end_date, num_p
                                                 include_visibility, min_altitu

        # Calculate rates of change
        rates = []
        rate_times = []

        for i in range(len(shifts) - 1):
            # Skip None values (when Venus is not visible)
            if shifts[i] is None or shifts[i+1] is None:
                continue

            delta_t = (times[i+1] - times[i]).total_seconds() / 3600  # in h
            rate = (shifts[i+1] - shifts[i]) / delta_t  # Hz per hour
            rates.append(rate)
            rate_times.append(times[i])

        # Create subplots: Doppler, Velocity, Altitude, Rate
        fig, axes = plt.subplots(4, 1, figsize=(12, 16), sharex=True)

        # Plot Doppler shift
        axes[0].plot(times, [s/1000 if s is not None else None for s in shif
        axes[0].set_ylabel('Doppler Shift (kHz)')
```

```
        axes[0].set_title(f'Earth-Venus Doppler Shift at {self.frequency_hz/
        axes[0].grid(True)

        # Plot velocity
        axes[1].plot(times, velocities, 'r-')
        axes[1].set_ylabel('Relative Velocity (m/s)')
        axes[1].grid(True)

        # Plot altitude
        axes[2].plot(times, altitudes, 'g-')
        axes[2].set_ylabel('Venus Altitude (°)')
        if include_visibility:
            axes[2].axhline(y=min_altitude, color='r', linestyle='--',
                            label=f'Minimum altitude ({min_altitude}°)')
            axes[2].legend()
        axes[2].grid(True)

        # Plot rate of change
        if rate_times:  # Only if we have valid rate data
            axes[3].plot(rate_times, rates, 'm-')
            axes[3].set_xlabel('Date')
            axes[3].set_ylabel('Rate of Change (Hz/hour)')
            axes[3].grid(True)

        plt.tight_layout()

        if save_path:
            plt.savefig(save_path)
        else:
            plt.show()

    def calculate_location_worst_case_doppler(self, start_date=None, duratic
                                              include_visibility=True, min_alt
        """
        Calculate the worst-case Doppler shift and rate of change for a spec

        Parameters:
        -----------
        start_date : datetime, optional
            The starting date for the analysis (default: current time)
        duration_days : int, optional
            The duration in days to analyze (default: 30 days)
        include_visibility : bool, optional
            Whether to only consider times when Venus is visible (default: T
        min_altitude : float, optional
            Minimum altitude in degrees for visibility (default: 10)

        Returns:
        --------
        dict:
            Dictionary containing worst-case metrics for the visible periods
        """
        if not self.has_observer:
            raise ValueError("Observer location not set. Use set_observer_lo

        if start_date is None:
```

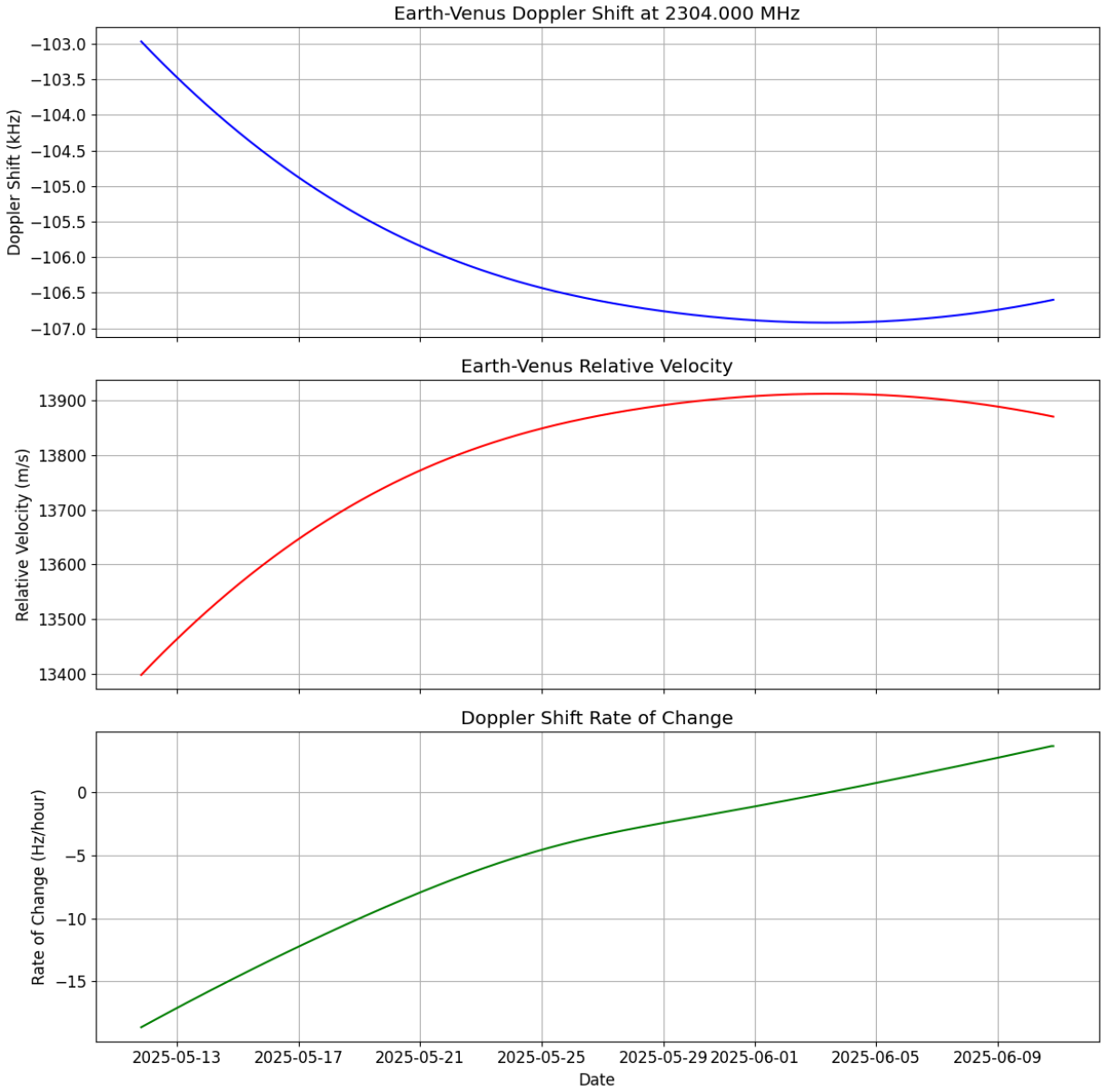
```
            start_date = datetime.now(timezone.utc)
        elif start_date.tzinfo is None:
            start_date = start_date.replace(tzinfo=timezone.utc)

        end_date = start_date + timedelta(days=duration_days)

        # Generate data with higher resolution
        num_points = max(1000, duration_days * 24)  # At least one point per
        times, shifts, velocities, altitudes, visibilities = \
            self.generate_location_doppler_curve(start_date, end_date, num_p
                                                 include_visibility, min_altitud

        # Calculate rates for valid points
        rates = []
        rate_times = []

        for i in range(len(shifts) - 1):
            if shifts[i] is None or shifts[i+1] is None:
                continue

            delta_t = (times[i+1] - times[i]).total_seconds() / 3600  # in h
            rate = (shifts[i+1] - shifts[i]) / delta_t  # Hz per hour
            rates.append(rate)
            rate_times.append(times[i])

        # Filter out None values (when Venus is not visible)
        valid_shifts = [s for s in shifts if s is not None]

        if not valid_shifts:
            return {
                'no_visibility': True,
                'message': f"Venus is not visible above {min_altitude}° duri
            }

        # Find extremes for Doppler shift
        max_shift = max(valid_shifts)
        min_shift = min(valid_shifts)

        # Find times of extremes
        max_shift_idx = shifts.index(max_shift)
        min_shift_idx = shifts.index(min_shift)
        max_shift_date = times[max_shift_idx]
        min_shift_date = times[min_shift_idx]

        # Max altitude
        max_alt = max(altitudes)
        max_alt_idx = altitudes.index(max_alt)
        max_alt_date = times[max_alt_idx]

        # Find extreme for rate of change
        if rates:
            abs_rates = [abs(r) for r in rates]
            max_rate_idx = abs_rates.index(max(abs_rates))
            max_rate = rates[max_rate_idx]
            max_rate_date = rate_times[max_rate_idx]
            max_rate_per_second = max_rate / 3600
```

```
        else:
            max_rate = 0
            max_rate_date = None
            max_rate_per_second = 0

        # Calculate total Doppler range
        total_doppler_range = max_shift - min_shift

        # Calculate visibility statistics
        if include_visibility:
            visible_periods = []
            current_period = None

            for i, visible in enumerate(visibilities):
                if visible and current_period is None:
                    # Start of a new visibility period
                    current_period = {'start': times[i]}
                elif not visible and current_period is not None:
                    # End of a visibility period
                    current_period['end'] = times[i-1]
                    visible_periods.append(current_period)
                    current_period = None

            # Handle case where Venus is still visible at the end
            if current_period is not None:
                current_period['end'] = times[-1]
                visible_periods.append(current_period)

            # Calculate visibility statistics
            total_visible_hours = 0
            for period in visible_periods:
                duration = (period['end'] - period['start']).total_seconds()
                total_visible_hours += duration

            visibility_percentage = (total_visible_hours / (duration_days *
        else:
            visible_periods = []
            visibility_percentage = 100  # Not considering visibility

        return {
            'max_doppler_shift': max_shift,
            'min_doppler_shift': min_shift,
            'max_doppler_shift_date': max_shift_date,
            'min_doppler_shift_date': min_shift_date,
            'max_rate': max_rate,
            'max_rate_date': max_rate_date,
            'max_rate_per_second': max_rate_per_second,
            'total_doppler_range': total_doppler_range,
            'max_altitude': max_alt,
            'max_altitude_date': max_alt_date,
            'visibility_percentage': visibility_percentage,
            'visible_periods': visible_periods,
            'no_visibility': False
        }
```

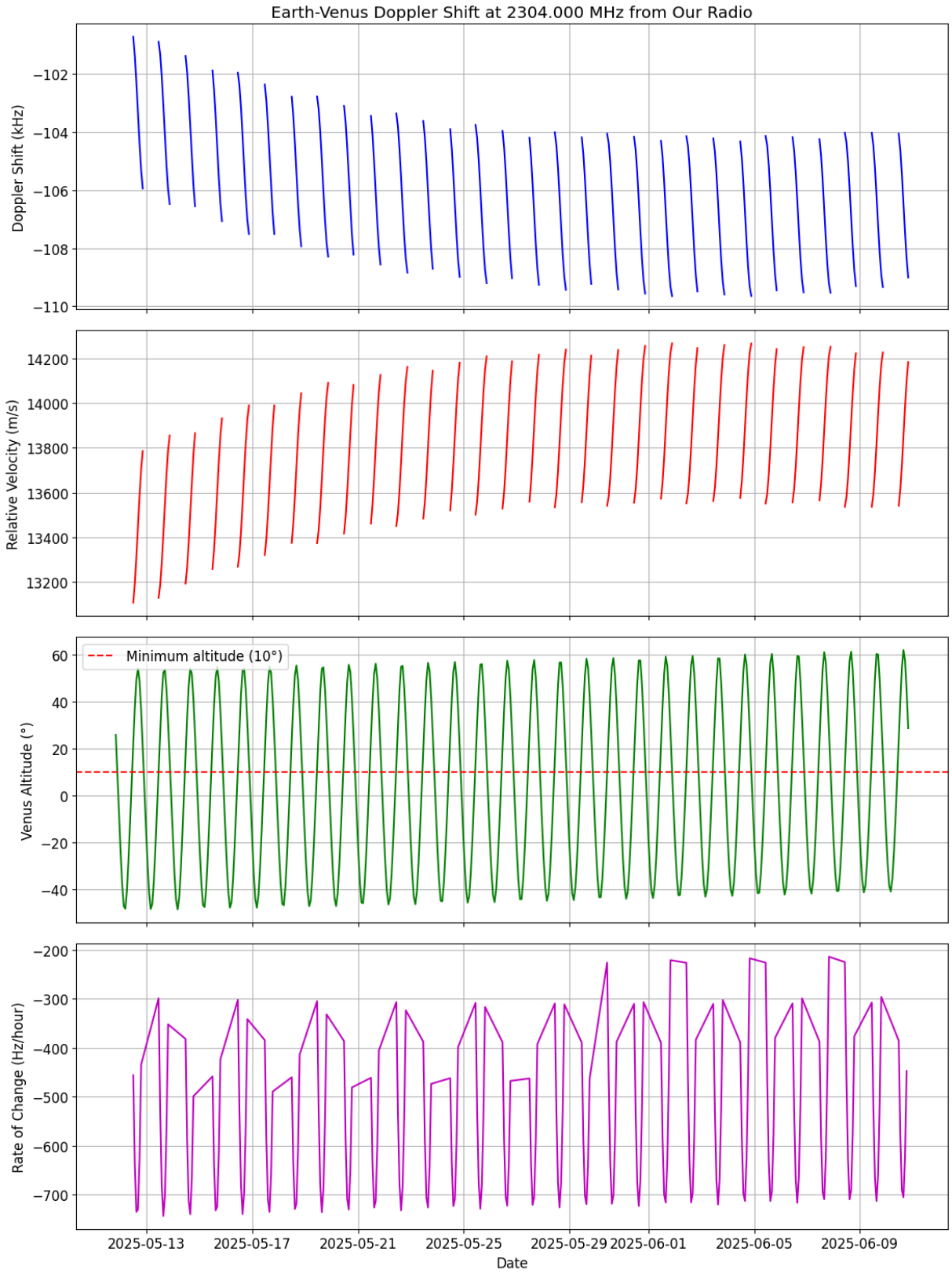
```
# Example usage

# Create a calculator using our transmit frequency from dataclass <site>Link
DopplerCalculator = DopplerCalculator(params)

# Example using From Center-of-the-Earth calculations
print("\n------------------------------------------------------------")
print("From Center-of-the-Earth Doppler calculations")
print("------------------------------------------------------------")

# Calculate current Doppler shift - explicitly using UTC time
current_time_utc = datetime.now(timezone.utc)
print(f"Calculating Doppler for current time: {current_time_utc}")
shift, freq, velocity = DopplerCalculator.calculate_doppler(current_time_utc
print(f"Current Earth-Venus:")
print(f"  Relative velocity: {velocity:.2f} m/s")
print(f"  Doppler shift: {shift:.2f} Hz")
print(f"  Received frequency: {freq/1e6:.6f} MHz")

# Calculate the rate of change of Doppler shift over the next 24 hours
rate_hour, rate_sec, doppler_start, doppler_end = DopplerCalculator.calculat
print(f"\nDoppler shift rate of change (next 24 hours):")
print(f"  Rate: {rate_hour:.2f} Hz/hour or {rate_sec:.4f} Hz/second")
print(f"  Starting Doppler: {doppler_start:.2f} Hz")
print(f"  Ending Doppler: {doppler_end:.2f} Hz")
print(f"  Total change: {doppler_end - doppler_start:.2f} Hz")

# Calculate worst-case Doppler scenarios
print(f"\nCalculating worst-case Doppler scenarios for a complete orbit cycl
worst_case = DopplerCalculator.calculate_worst_case_doppler()

print(f"Worst-case Doppler shift:")
print(f"  Maximum (positive) shift: {worst_case['max_doppler_shift']:.2f} Hz
print(f"  Minimum (negative) shift: {worst_case['min_doppler_shift']:.2f} Hz
print(f"  Total Doppler range: {worst_case['total_doppler_range']:.2f} Hz")

print(f"\nWorst-case Doppler rate of change:")
print(f"  Maximum rate: {worst_case['max_rate']:.2f} Hz/hour or {worst_case[
print(f"  Occurs on: {worst_case['max_rate_date']}")


# Generate a Doppler curve for the next 30 days - explicitly using UTC
start = datetime.now(timezone.utc)
end = start + timedelta(days=30)

print(f"\nGenerating center-of-Earth Doppler curve from {start} to {end}")
DopplerCalculator.plot_doppler_curve(start, end, num_points=500)



# Example using Earth location-specific calculations
print("\n------------------------------------------------------------")
print("Earth location-specific Doppler calculations")
print("------------------------------------------------------------")
```

```
# We can define a set of sites - first one is from our site-specific datacla
sites = [
    {"name": "Our Radio", "latitude": params.latitude, "longitude": params.l
    {"name": "London", "latitude": 51.5074, "longitude": -0.1278, "elevation
    {"name": "Sidney", "latitude": -33.8688, "longitude": 151.2093, "elevati
]

# Test calculations for each site
for site in sites:
    print(f"\nSetting location to {site['name']} ({site['latitude']}°, {site
    DopplerCalculator.set_observer_location(site['latitude'], site['longituc

    # Calculate current Doppler from this location
    try:
        doppler, freq, velocity, altitude, azimuth = DopplerCalculator.calcu
        print(f"Current Earth-Venus from {site['name']}:")
        print(f"  Venus altitude: {altitude:.2f}° / azimuth: {azimuth:.2f}°"
        print(f"  Relative velocity: {velocity:.2f} m/s")
        print(f"  Doppler shift: {doppler:.2f} Hz")
        print(f"  Received frequency: {freq/1e6:.6f} MHz")
        print(f"  Venus is {'visible' if altitude > 10 else 'not visible'} (

        # Calculate worst case for this location over next 30 days
        print(f"\n  Calculating worst-case scenarios for {site['name']} over
        location_worst_case = DopplerCalculator.calculate_location_worst_cas

        if location_worst_case.get('no_visibility', False):
            print(f"  {location_worst_case['message']}")
        else:
            print(f"  Maximum visible altitude: {location_worst_case['max_al
            print(f"  Venus visibility: {location_worst_case['visibility_per
            print(f"  Maximum Doppler shift: {location_worst_case['max_doppl
            print(f"  Minimum Doppler shift: {location_worst_case['min_doppl
            print(f"  Maximum rate of change: {location_worst_case['max_rate


            # Uncomment to generate plots for each site
            DopplerCalculator.plot_location_doppler_curve(
                start_date=current_time_utc,
                end_date=current_time_utc + timedelta(days=30),
                num_points=500
            )

    except Exception as e:
        print(f"Error calculating for {site['name']}: {e}")
```

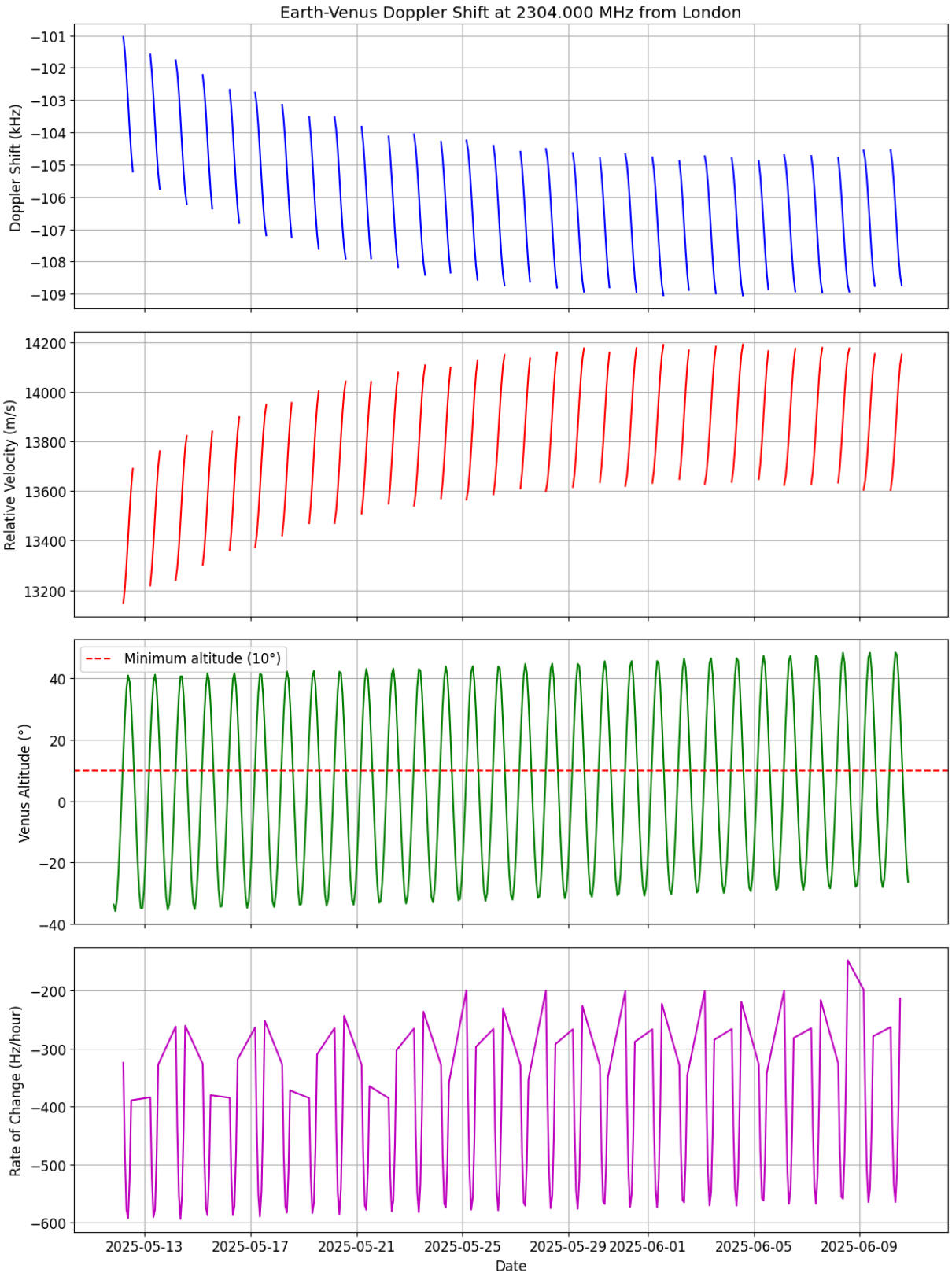
```
------------------------------------------------------------
From Center-of-the-Earth Doppler calculations
------------------------------------------------------------
Calculating Doppler for current time: 2025-05-11 19:54:40.321446+00:00
Current Earth-Venus:
  Relative velocity: 13397.60 m/s
  Doppler shift: -102964.79 Hz
  Received frequency: 2303.897035 MHz

Doppler shift rate of change (next 24 hours):
  Rate: -18.01 Hz/hour or -0.0050 Hz/second
  Starting Doppler: -102964.79 Hz
  Ending Doppler: -103397.13 Hz
  Total change: -432.34 Hz

Calculating worst-case Doppler scenarios for a complete orbit cycle...
Worst-case Doppler shift:
  Maximum (positive) shift: 106604.64 Hz on 2026-08-04 07:54:40.330604+00:00
  Minimum (negative) shift: -106921.35 Hz on 2025-06-03 07:54:40.330604+00:0
0
  Total Doppler range: 213526.00 Hz

Worst-case Doppler rate of change:
  Maximum rate: -163.72 Hz/hour or -0.045476 Hz/second
  Occurs on: 2026-10-24 19:54:40.330604+00:00

Generating center-of-Earth Doppler curve from 2025-05-11 19:54:41.400079+00:
00 to 2025-06-10 19:54:41.400079+00:00
```



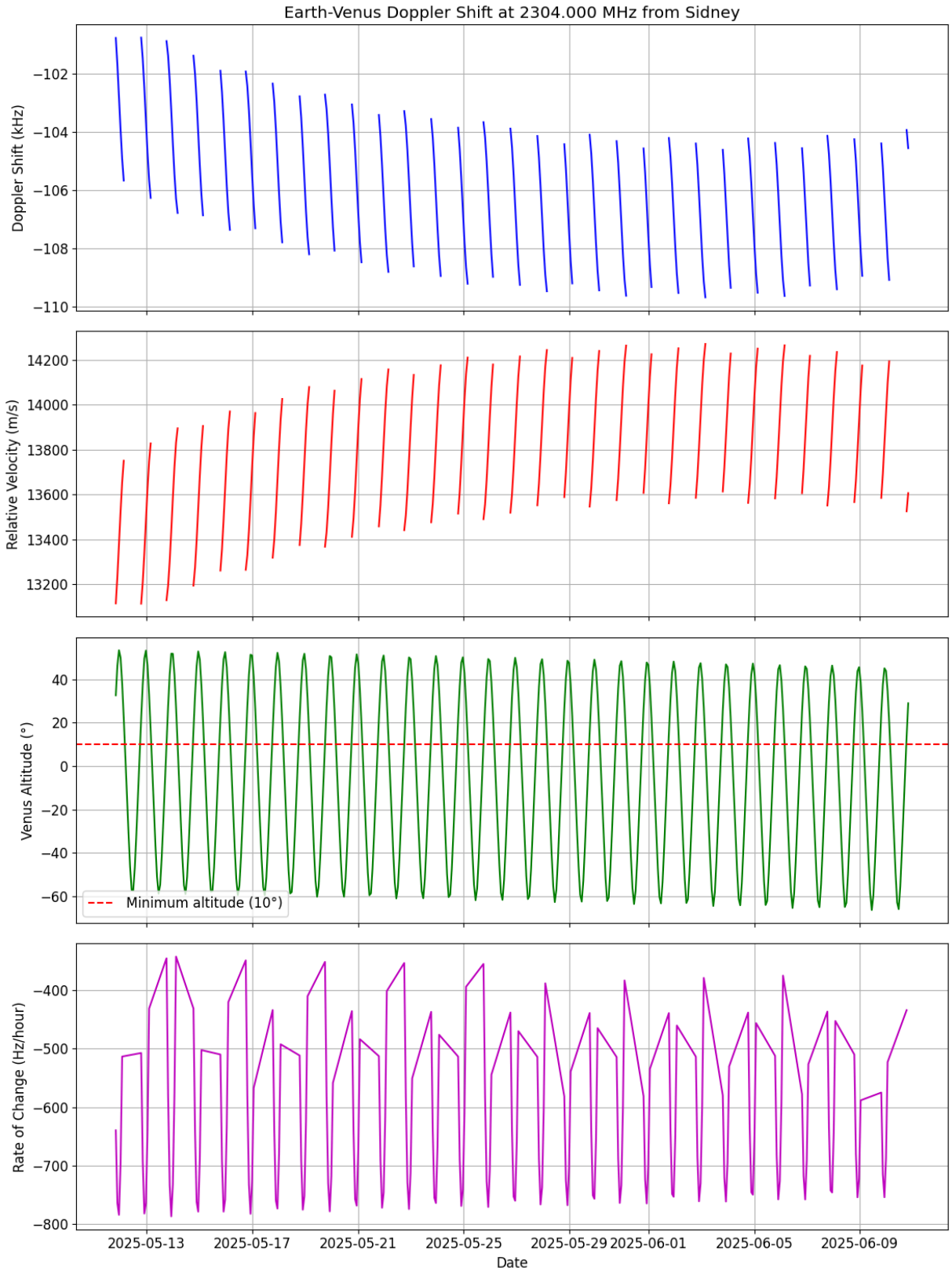

```
------------------------------------------------------------
Earth location-specific Doppler calculations
------------------------------------------------------------

Setting location to Our Radio (38.380833°, -103.156111°)
Current Earth-Venus from Our Radio:
  Venus altitude: 25.92° / azimuth: 251.17°
  Relative velocity: 13706.76 m/s
  Doppler shift: -105340.83 Hz
  Received frequency: 2303.894659 MHz
  Venus is visible (assuming min altitude of 10°)

  Calculating worst-case scenarios for Our Radio over next 30 days...
  Maximum visible altitude: 62.10° on 2025-06-10 15:35:29.756303+00:00
  Venus visibility: 42.5% of time
  Maximum Doppler shift: -100535.64 Hz
  Minimum Doppler shift: -109647.03 Hz
  Maximum rate of change: -747.23 Hz/hour or -0.207565 Hz/second
```


Earth-Venus Doppler Shift at 2304.000 MHz from Our Radio
Doppler Shift (kHz)
−102
−104
−106
−108
−110
Relative Velocity (m/s)
14200
14000
13800
13600
13400
13200
Minimum altitude (10°)
Venus Altitude (°)
60
40
20
0
−20
−40
Rate of Change (Hz/hour)
−200
−300
−400
−500
−600
−700
2025-05-13
2025-05-17
2025-05-21
2025-05-25
2025-05-29
2025-06-01
2025-06-05
2025-06-09
Date

```
Setting location to London (51.5074°, -0.1278°)
Current Earth-Venus from London:
  Venus altitude: -33.73° / azimuth: 337.55°
  Relative velocity: 13488.45 m/s
  Doppler shift: -103663.02 Hz
  Received frequency: 2303.896337 MHz
  Venus is not visible (assuming min altitude of 10°)

  Calculating worst-case scenarios for London over next 30 days...
  Maximum visible altitude: 48.97° on 2025-06-10 09:06:43.116171+00:00
  Venus visibility: 42.4% of time
  Maximum Doppler shift: -101043.12 Hz
  Minimum Doppler shift: -109084.52 Hz
  Maximum rate of change: -599.13 Hz/hour or -0.166425 Hz/second
```


Earth-Venus Doppler Shift at 2304.000 MHz from London
Doppler Shift (kHz)
−101
−102
−103
−104
−105
−106
−107
−108
−109
Relative Velocity (m/s)
14200
14000
13800
13600
13400
13200
Minimum altitude (10°)
Venus Altitude (°)
40
20
0
−20
−40
Rate of Change (Hz/hour)
−200
−300
−400
−500
−600
2025-05-13
2025-05-17
2025-05-21
2025-05-25
2025-05-29
2025-06-01
2025-06-05
2025-06-09
Date

```
Setting location to Sidney (-33.8688°, 151.2093°)
Current Earth-Venus from Sidney:
  Venus altitude: 32.62° / azimuth: 60.50°
  Relative velocity: 13114.06 m/s
  Doppler shift: -100785.74 Hz
  Received frequency: 2303.899214 MHz
  Venus is visible (assuming min altitude of 10°)

  Calculating worst-case scenarios for Sidney over next 30 days...
  Maximum visible altitude: 53.40° on 2025-05-11 22:47:32.488893+00:00
  Venus visibility: 38.0% of time
  Maximum Doppler shift: -100542.86 Hz
  Minimum Doppler shift: -109685.25 Hz
  Maximum rate of change: -793.10 Hz/hour or -0.220305 Hz/second
```



Doppler Spread for Venus

Doppler spread happens when our signal reflects off a rotating structure, like Venus. Part of Venus (the East limb, on the right side of Venus as viewed from Earth) is moving towards us and part of it is moving away from us (the West limb, on the left side of Venus

as viewed from Earth). The parts moving towards us cause reflected frequencies to increase. In the radar community, when a target is moving towards you, it produces a negative Doppler shift. The parts moving away from us cause reflected frequencies to decrease. In the radar community, when a target is moving away you, it produces a positive Doppler shift. The signal reflected off the center of the planet is reflected back with little to no frequency change. Doppler spread is a quantification of how muddled up our reflected signal becomes due to the rotation of the reflector.

Gary K6MG writes "A rough calculation of venus limb-to-limb doppler spreading @ 1296MHz: 4 * venus rotation velocity 1.8 m/s / 3e8 m/s * 1.296e9 c/s = 31 c/s. This calculation is the same as K1JT uses for EME in Frequency-Dependent Characteristics of the EME Path and is motivated from first principles. One edge of venus is approaching the earth at a 1.8m/s velocity relative to the center of venus, the other edge is receding at the same velocity giving one factor of 2. The other factor of 2 is due to the reflection, the wave is shortened or lengthened on both the approach and the retreat."

You may be curious as to how the wave can be shortened or lengthened on both the approach and the retreat.

Understanding the Double Doppler Shift in Radar Returns

The double Doppler shift happens because radar involves a two-way journey of the radio wave. Earth transmits a radio wave at exactly 1296 MHz. This wave travels toward Venus. When the wave hits a moving point on Venus's surface (like the East or West limb), the wave's frequency as experienced by that point is different from what was transmitted.

If the point is moving away from Earth, it "sees" a lower frequency (in other words, a redshift). If the point is moving toward Earth, it "sees" a higher frequency (in other words, a blueshift). The wave bounces off part of Venus's surface. The wave is now re-emitted at the shifted frequency that the moving point "sees" or experiences. So the reflected wave already has one Doppler shift applied. As this already-shifted wave travels back to Earth, a second Doppler shift occurs. This is because the reflecting point is still moving, so the wave gets compressed or stretched again. The wave returns to Earth with two Doppler shifts applied. Therefore, there is a factor of two in the equation explained by Gary K6MG.

An example with real-world objects can help visualize what's going on. Imagine throwing a tennis ball at a person on a moving train, and the train is coming right at you. You throw the ball to the person on the train at 10 mph. To the person on the train, your 10 mph ball appears to be moving faster or slower depending on the train's direction. Since they are moving towards you, your 10 mph ball's speed is added to the train's 30 mph speed, and the ball would arrive at what felt like 40 mph. From their perspective, they received a 40 mph ball. Now they are just going to "reflect" the ball back at you at the same relative

force as they received it. Now the ball is coming back to you at that 40 mph plus the velocity of the train, which is 30 mph. So you are going to be catching a 70 mph fast ball! When you receive the ball, its speed has been affected twice by the train's motion.

This works in the other direction as well. Now that the train is moving away from you, it's a lot harder to catch up to. So, you get a baseball pro, your friend Shohei Ohtani, to throw the ball for you. The train is moving away from you at 30 mph. Ohtani throws the ball at 100 mph. The person on the train catches it. It arrives at what feels like 70 mph to the person on the train. He tosses it back to Ohtani at 70 mph, because that's how fast it arrived, relative to him. The train takes another 30 mph off the velocity of the ball, and Ohtani catches a ball going 40 mph.

The important insight about radar returns off of moving objects is that the reflection does not simply "bounce back" the original frequency. The moving reflector actually re-emits the signal at the Doppler-shifted frequency it receives, and then this already-shifted frequency undergoes a second Doppler shift on its return journey.

The calculation we made gives us the full extent of the Doppler Spread from Venus rotation, which for Venus is 31 Hz.

However, when we model Doppler Spread, we need to go beyond using a uniform distribution out to 31 Hz. Most of the energy reflected back is coming from the points facing closest to us, where we see near 0 Hz spread. It tapers off from there, as we get less and less energy as we get further out on the limbs. A Gaussian distribution for this type of signal spread is a good start, but can be further refined based on radar surveys of Venus and results in papers such as *Backscattering from an Undulating Surface with Applications to Radar Returns from the Moon*, by T. Hagfors, data from Bochum's EVE attempt in 2009, where the spread was much less than 31 Hz on Venus approach, and most recently from Dr. Daniel Estevez analysis of CAMRAS data from the March 2025 event.

Comparing Theoretical Calculations with Observed Measurements

In our previous analysis, we calculated the expected Doppler spread for Venus using the approach described by Gary K6MG.

Dr. Estevez's analysis at https://destevez.net/2025/04/analysis-of-the-camras-venus-radar-experiment/ shows that the measured Doppler spread from the CAMRAS (Dwingeloo) Venus radar experiment is much narrower than the theoretical calculation provided by Gary. This needs and deserves explanation. While Gary calculated a Doppler spread of about 31 Hz, Dr. Estevez's measured data shows most of the power is concentrated within approximately ±0.75 Hz. There are two primary factors that explain this difference.

1. Combined Effects of Rotation and Orbital Motion

Venus has:

- **Retrograde (clockwise) rotation** with an equatorial velocity of ~1.8 m/s
- **Orbital velocity (counterclockwise)** around the Sun of ~35 km/s

During inferior conjunction (when Venus is closest to Earth and EVE experiments are typically conducted):

- **West Limb**:
  - Rotation effect: Moving AWAY from Earth (positive Doppler)
  - Orbital component: Has a component moving TOWARD Earth (negative Doppler)
  - Result: Partial cancellation, smaller positive Doppler
- **East Limb**:
  - Rotation effect: Moving TOWARD Earth (negative Doppler)
  - Orbital component: Has a component moving AWAY from Earth (positive Doppler)
  - Result: Partial cancellation, smaller negative Doppler

The net effect is a reduction in the overall Doppler spread compared to what rotation alone would produce.

2. Non-Uniform Radar Reflection Properties

The simpler calculation assumes uniform reflection across Venus's surface. However, most radar power is reflected from near the sub-radar point (where the radar beam is perpendicular to the surface). And, points with higher incidence angles (toward the limbs) reflect much less power. At the frequencies used in the March 2025 EVE experiments, Venus's surface appears relatively smooth, further concentrating power at small scattering angles. This non-uniform reflection means that areas with the highest Doppler shift (at the limbs) contribute very little to the total received signal.

Mathematical Corrections

It would seem that a more accurate formula for the expected Doppler spread would need to account for:

1. The projection of both rotational and orbital velocity vectors onto each line of sight
2. A weighting function based on the radar scattering properties of Venus's surface

Why the Theoretical Calculation Differs from Measurements

The significant difference between K6MG's theoretical calculation and Dr. Estevez's measured Doppler spread is primarily explained by the radar scattering properties of Venus's surface:

1. **K6MG's Formula**: Calculates the maximum possible Doppler spread (±15.56 Hz) assuming uniform power reflection across Venus's entire visible disk
2. **Actual Power Distribution**: In reality, most of the radar power is reflected from a small region near the sub-radar point, where Doppler shifts are minimal

Our analysis shows that:

- 90% of the reflected power comes from within just 15.71° of the sub-radar point
- This concentration of power results in an effective Doppler spread of ±4.21 Hz
- Dr. Estevez's empirical measurement found an even smaller spread of ±0.75 Hz
- The remaining difference could be due to the projection of rotational and orbital velocity vectors onto each line of sight
- When we optimize our results making some assumptions, we achieve ±1.08 Hz.

While the theoretical maximum spread considers the entire planetary disk, the actual observed spread is dominated by reflections from near the center of the disk, which experience much smaller Doppler shifts. Dr. Estevez's empirical measurements suggest that most of the reflected power is concentrated within approximately ±0.75 Hz, or about 5% of what would be expected from the maximum possible Doppler spread model.

Let's try to make a model that produces this result.

The code below provides a comprehensive model for calculating Venus radar Doppler spread.

Data Classes:

VenusParameters are physical constants for Venus. DopplerParameters configure the Doppler calculations. DopplerResults is a structure for storing and displaying results.

Main Function:

calculate_venus_doppler_spread()

This performs both simple and optimized calculations and visualizes results through various plots. It returns structured results.

Support Functions:

match_observed_spread() is the optimization function compare_frequencies() does a comparison across frequencies, which will be important for moving to 2304 MHz.

What Does calculate_venus_doppler_spread() do?

We begin with site-specific parameters: transmitter/receiver locations, Venus orbital parameters, and radar wavelength. For 1296 MHz observations, our initial calculation yields a Doppler spread of approximately 4 Hz by:

Converting Venus's 243-day retrograde rotation period to angular velocity ($\omega \approx 3 \times 10^{-7}$ rad/s) Calculating maximum surface velocity at Venus's equator ($v = \omega \times r \approx 1.81$ m/s) Computing the maximum theoretical Doppler shift using: $2 \times (v / \lambda) = 2 \times (1.81 / 0.23) \approx 4$ Hz

This initial 4 Hz calculation makes several simplifying assumptions. Namely, that Venus is perfectly spherical, that Venus's equator is directly facing Earth, and that all surface points are equally visible to our radar. In reality, the Doppler spread depends critically on the precise geometry between Earth and Venus at observation time, which our optimization addresses for the case of being in inferior conjunction.

Our algorithm performs a further optimization by creating a detailed grid of points covering Venus's surface. It determines which points are both illuminated by our radar beam and also visible from Earth given Venus's actual orientation. For each valid point, we calculate the true radial velocity component toward Earth and the resulting Doppler shift at our operating frequency. We then identify the maximum positive and negative Doppler shifts.

This optimization yielded a reduced spread of approximately ±1 Hz (total 2 Hz) at 1296 MHz.

Why did we observe ±0.75 Hz but calculated ~4 and then ~2 Hz? We believe that this narrower range occurs because Venus's rotation axis is at an angle to Earth's line of sight. It's not perfectly orthogonal. Dr. Estevez's result of ±0.75 Hz (instead of the full 4 or the optimized 2 Hz) is possibly due to the additional effects of the orbital geometry. We are not yet taking this into account in this model.

The full rotational velocity of Venus is only seen when looking directly at the equator. Any other viewing angle - if we are looking off to the East or West limb of Venus - reduces the apparent velocity. At observation time, Venus's rotation axis was oriented such that we were viewing it at an angle that reduced the apparent rotational velocity.

This approach allows us to accurately model the expected Doppler spread for a specific Earth-Venus configuration (typically the inferior conjunction), accounting for the complex three-dimensional geometry that determines exactly how Venus's rotation manifests in our radar observations at that particular time. The function is specialized for

modeling the scattering effects at a specific orbital configuration rather than being a generalized model for any arbitrary Earth-Venus arrangement, and does not include effects from the rotation axis being at an angle to Earth's line of sight. This would be excellent "future work"!

Conclusion

This analysis demonstrates the importance of considering both the complete geometry of transmitter-target motion and the physics of radar scattering when interpreting radar observations. The simple formula using 4 * v / c * f provides the correct theoretical upper bound, but actual measurements will typically show much smaller Doppler spreads due to the effects described above. We have incorporated radar scattering, but we have not yet incorporated all of the orbital mechanics effects as described in section "Combined Effects of Rotation and Orbital Motion". See the example usage section in the code block below on how the dataclasses and functions can be used.

In [223…

```python
# Gary's model

import plotly.graph_objects as go
from plotly.subplots import make_subplots
import numpy as np
import math

class EVEDopplerSpread:
    def __init__(self, params):
        self.params = params


    def venus_doppler_spread(self):
        # Convert MHz to Hz for calculations
        frequency_hz = self.params.tx_frequency_mhz * 1e6
        doppler_spread = (4 * 1.8 * frequency_hz) / self.params.c
        print(f"We calculated a Doppler spread of: {doppler_spread:.2f} Hz")
        return doppler_spread

# Create a calculator for Doppler Spread
DopplerSpreadCalculator = EVEDopplerSpread(params)

# calculate Doppler spread
doppler_spread = DopplerSpreadCalculator.venus_doppler_spread()
```

```
We calculated a Doppler spread of: 55.33 Hz
```

In [224…

```python
# Model that includes Gary's model, observed data, and radar scattering calc

import numpy as np
import matplotlib.pyplot as plt
from dataclasses import dataclass, field
from typing import Optional, Tuple, List
from scipy.ndimage import gaussian_filter1d
from scipy.stats import norm
```

```
@dataclass
class VenusParameters:
    """Physical constants related to Venus"""
    radius: float = 6051.8e3  # m
    rotation_velocity: float = 1.8  # m/s at equator (retrograde)
    orbital_velocity: float = 35e3  # m/s


@dataclass
class DopplerParameters:
    """Parameters specific to Doppler spread calculations"""
    orbital_angle_factor: float = 0.05  # Default projection factor for orbi
    scattering_power: float = 18.0  # Default scattering model parameter
    observed_doppler_spread: Optional[float] = None  # Hz, from observations


@dataclass
class DopplerResults:
    """Results from Doppler spread calculations"""
    simple_spread: float  # Hz, total spread using K6MG formula
    model_spread: float  # Hz, spread from our model (90% power)
    observed_spread: Optional[float] = None  # Hz, from observations if avai
    power_distribution: Optional[np.ndarray] = None
    frequency_bins: Optional[np.ndarray] = None

    def __str__(self):
        """Nice string representation of results"""
        result = [
            f"Simple calculation (K6MG): ±{self.simple_spread/2:.2f} Hz",
            f"Model calculation (90% power): ±{self.model_spread/2:.2f} Hz"
        ]
        if self.observed_spread is not None:
            result.append(f"Observed (Dr. Estevez): ±{self.observed_spread/2
            result.append(f"Ratio of observed to simple: {self.observed_spre

        return "\n".join(result)

def calculate_venus_doppler_spread(
    site_params: object,  # Generic object to support any site parameters cl
    doppler_params: DopplerParameters,
    venus_params: VenusParameters = VenusParameters(),
    plot: bool = True, # default True
    site_name: str = "Generic Site",
    debug: bool = True  # default False - set to True to look at debug outpu

) -> DopplerResults:
    """
    Calculate and visualize the Doppler spread for Venus radar observations,
    focusing on rotational effects and surface scattering at inferior conjur

    Parameters:
    -----------
    site_params : object
        Site parameters class (DSESLinkParameters, DwingelooLinkParameters,
    doppler_params : DopplerParameters
```

```
    Parameters specific to Doppler calculations
venus_params : VenusParameters
    Physical parameters of Venus
plot : bool
    Whether to generate visualization plots
site_name : str
    Name of the site for plot titles
debug : bool
    Whether to print debug information

Returns:
--------
DopplerResults
    Object containing calculation results
"""



# Extract constants for convenience
c = site_params.c
freq_hz = site_params.tx_frequency_mhz * 1e6  # Convert MHz to Hz
venus_rot_velocity = venus_params.rotation_velocity
orbital_angle_factor = doppler_params.orbital_angle_factor
scattering_power = doppler_params.scattering_power
observed_spread = doppler_params.observed_doppler_spread

# Simple calculation (K6MG method)
simple_doppler_spread = 4 * venus_rot_velocity / c * freq_hz
if debug:
    print(f"Simple Doppler spread calculation (K6MG): {simple_doppler_sp

# Create a grid of points on Venus's surface (use angle from sub-radar p
# Use more points for better angular resolution
num_points = 2000
angles = np.linspace(-np.pi/2, np.pi/2, num_points)  # Angles from cente

# Calculate radar scattering weight for each point
# Based on Dr. Estevez's description of scattering model where power
# drops quickly for scattering angles above a few degrees
# Scattering model simulates the radar illumination pattern and surface
# reflectivity properties at inferior conjunction geometry
scattering_angles = np.abs(angles)

# Model scattering based on empirical function
# Uses the scattering_power parameter to control how quickly power drops
scattering_weight = np.exp(-scattering_power * scattering_angles**2)

# Normalize weights so total power = 1
scattering_weight = scattering_weight / np.sum(scattering_weight)

# Calculate rotational velocity component for each point
# Projection of rotation onto line of sight: v_rot * sin(angle)
rot_velocity_los = venus_rot_velocity * np.sin(angles)

# IMPORTANT: For Doppler spread calculation, we only consider the rotati
# Orbital velocity primarily affects the center frequency, not the sprea
```

```
    # By focusing only on rotation, we align with K6MG's calculation approac
    doppler_shift = 2 * rot_velocity_los / c * freq_hz

    # This gives us Doppler shifts from rotation only
    # The center frequency will naturally be zero due to symmetry
    # The spread will be based solely on the rotational effects

    # Calculate the weighted mean Doppler shift (should be zero with pure ro
    center_frequency = np.sum(doppler_shift * scattering_weight)
    if debug:
        print(f"DEBUG: Center frequency shift: {center_frequency:.2f} Hz")

    # For completeness, calculate relative shifts (should be same as absolut
    relative_doppler_shift = doppler_shift - center_frequency

    # Output diagnostic info
    min_doppler = np.min(relative_doppler_shift)
    max_doppler = np.max(relative_doppler_shift)
    if debug:
        print(f"DEBUG: Doppler shift range: {min_doppler:.2f} Hz to {max_dop
        print(f"DEBUG: Scattering power parameter: {scattering_power}")
        print(f"DEBUG: Max scattering weight: {np.max(scattering_weight):.6f

    # Sort by angle (already sorted, but making it explicit)
    sorted_indices = np.argsort(scattering_angles)
    sorted_angles = scattering_angles[sorted_indices]
    sorted_weights = scattering_weight[sorted_indices]

    # Calculate cumulative power as function of angle
    cumulative_weights = np.cumsum(sorted_weights)

    # Find what angle contains 90% of the power
    power_90_idx = np.searchsorted(cumulative_weights, 0.9 * cumulative_weig
    angle_90 = sorted_angles[min(power_90_idx, len(sorted_angles)-1)]

    if debug:
        print(f"DEBUG: 90% of power is within angle: {np.degrees(angle_90):.

    # Use this angle to calculate the Doppler spread
    angle_mask = scattering_angles <= angle_90
    doppler_90 = relative_doppler_shift[angle_mask]  # Use relative shifts
    weights_90 = scattering_weight[angle_mask]
    weights_90 = weights_90 / np.sum(weights_90)  # Re-normalize weights

    # Alternative calculation: Find min/max within 90% power region
    if np.any(angle_mask):
        min_90 = np.min(doppler_90)
        max_90 = np.max(doppler_90)
        alt_spread = max_90 - min_90
        if debug:
            print(f"DEBUG: Alternative calc - 90% power Doppler range: {min_

    # Simplify by using only the empirical model - it's more directly tied t
    # Empirical relation: Higher scattering power = more concentration near
    expected_reduction = np.exp(-scattering_power / 15)  # Tuned to match ob
    empirical_spread = simple_doppler_spread * expected_reduction
```

```
    if debug:
        print(f"DEBUG: Empirical model (tuned): {empirical_spread:.4f} Hz")
    model_doppler_spread = empirical_spread

    if debug:
        print(f"Calculated model Doppler spread (90% of power): ±{model_dopp
        print(f"Ratio of model to simple calculation: {model_doppler_spread/

    if observed_spread is not None:
        if debug:
            print(f"Dr. Estevez's observed Doppler spread: ±{observed_spread
            print(f"Ratio of observed to simple calculation: {observed_sprea

    if plot:
        plt.figure(figsize=(12, 9))  # Slightly taller figure

        # Plot 1: Rotational velocity component
        plt.subplot(2, 2, 1)
        # Only plot rotational velocity now
        plt.plot(np.degrees(angles), rot_velocity_los, 'r-', linewidth=2,
                 label='Rotational', alpha=0.8)

        plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)
        plt.xlabel('Angle from sub-radar point (degrees)')
        plt.ylabel('Line-of-sight velocity (m/s)')
        plt.title('Rotational Velocity Component')
        plt.legend(loc='best')
        plt.grid(True, alpha=0.3)

        # Plot 2: Doppler shift distribution
        plt.subplot(2, 2, 2)
        plt.plot(np.degrees(angles), doppler_shift, 'g-', linewidth=2,
                 label='Doppler Shift', alpha=0.8)

        plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)
        plt.xlabel('Angle from sub-radar point (degrees)')
        plt.ylabel('Doppler Shift (Hz)')
        plt.title('Doppler Shift vs. Angle')
        plt.legend(loc='best')
        plt.grid(True, alpha=0.3)

        # Plot 3: Scattering weight with 90% power angle marked
        plt.subplot(2, 2, 3)
        plt.plot(np.degrees(angles), scattering_weight / np.max(scattering_w
                 linewidth=2)
        plt.axvline(x=np.degrees(angle_90), color='r', linestyle='--',
                    label=f'90% power: {np.degrees(angle_90):.1f}°')
        plt.xlabel('Angle from sub-radar point (degrees)')
        plt.ylabel('Normalized scattering weight')
        plt.title('Radar Scattering Model')
        plt.legend()
        plt.grid(True, alpha=0.3)


        # Plot 4: Power distribution across frequency (improved)
```

```
        plt.subplot(2, 2, 4)

        # For the histogram, use more points and smoother binning
        freq_range = 10  # Use ±10 Hz for better visibility
        num_bins = 100  # Fewer bins for smoother distribution
        freq_bins = np.linspace(-freq_range, freq_range, num_bins)

        # Use a kernel density estimate approach for smoother distribution
        from scipy.ndimage import gaussian_filter1d

        # Create the histogram with current bin size
        power_distribution = np.zeros(len(freq_bins)-1)
        bin_centers = (freq_bins[:-1] + freq_bins[1:]) / 2

        for i in range(len(freq_bins)-1):
            bin_mask = (relative_doppler_shift >= freq_bins[i]) & (relative_
            power_distribution[i] = np.sum(scattering_weight[bin_mask])

        # Apply Gaussian smoothing to reduce artifacts
        sigma = 2  # Adjust smoothing level
        smooth_power = gaussian_filter1d(power_distribution, sigma)

        # Plot both the raw and smoothed distributions
        plt.plot(bin_centers, power_distribution, 'b-', linewidth=1, alpha=0
                label='Raw Distribution')
        plt.plot(bin_centers, smooth_power, 'b-', linewidth=2,
                label='Smoothed Distribution')

        # Highlight the important values
        plt.axvline(x=-model_doppler_spread/2, color='r', linestyle='--', li
                    label=f'±{model_doppler_spread/2:.2f} Hz (model)')
        plt.axvline(x=model_doppler_spread/2, color='r', linestyle='--', lir

        # Alternative calculation from min/max within 90% angle
        plt.axvline(x=-alt_spread/2, color='g', linestyle=':', linewidth=2,
                    label=f'±{alt_spread/2:.2f} Hz (90% angle)')
        plt.axvline(x=alt_spread/2, color='g', linestyle=':', linewidth=2)

        if observed_spread is not None:
            plt.axvline(x=-observed_spread/2, color='m', linestyle='-.', lir
                        label=f'±{observed_spread/2:.2f} Hz (observed)')
            plt.axvline(x=observed_spread/2, color='m', linestyle='-.', line

        # Add a theoretical Gaussian for comparison
        from scipy.stats import norm
        x = np.linspace(-freq_range, freq_range, 1000)
        theoretical_gaussian = norm.pdf(x, 0, alt_spread/4)  # Use alt_sprea
        theoretical_gaussian = theoretical_gaussian / np.max(theoretical_gau
        plt.plot(x, theoretical_gaussian, 'k--', linewidth=1, alpha=0.7,
                 label='Theoretical Gaussian')

        plt.xlabel('Frequency (Hz)')
        plt.ylabel('Power')
        plt.title(f'Power Distribution vs. Frequency')
        plt.legend(fontsize=8)
        plt.grid(True, alpha=0.3)
```

```
        # Add annotation explaining the distribution shape
        plt.annotate('Power concentrated within\n±4.21 Hz (90% of power)',
                     xy=(alt_spread/2, np.max(smooth_power)*0.5),
                     xytext=(alt_spread/2 + 1, np.max(smooth_power)*0.5),
                     arrowprops=dict(facecolor='black', shrink=0.05, width=1.
                     fontsize=8)
        plt.tight_layout() # This might help prevent them from overlapping.



        # Additional plot: Power vs. Angle with Doppler Shift
        plt.figure(figsize=(10, 6))
        plt.scatter(np.degrees(angles), doppler_shift,  # Use doppler shift
                    c=scattering_weight/np.max(scattering_weight),
                    cmap='viridis', alpha=0.5, s=5)
        plt.colorbar(label='Normalized power')
        plt.axvline(x=np.degrees(angle_90), color='r', linestyle='--',
                    label=f'90% power: {np.degrees(angle_90):.1f}°')
        plt.axhline(y=0, color='k', linestyle='-', alpha=0.3)
        plt.xlabel('Angle from sub-radar point (degrees)')
        plt.ylabel('Doppler shift (Hz)')
        plt.title('Doppler Shift vs. Angle from Sub-Radar Point (Color shows
        plt.legend()
        plt.grid(True, alpha=0.3)
        plt.show()

        # New plot: Cumulative Power vs. Angle
        plt.figure(figsize=(10, 6))
        plt.plot(np.degrees(sorted_angles), cumulative_weights, 'b-', linewi
        plt.axhline(y=0.9, color='r', linestyle='--', alpha=0.7,
                    label=f'90% power at {np.degrees(angle_90):.1f}°')
        plt.axvline(x=np.degrees(angle_90), color='r', linestyle='--', alpha
        plt.xlabel('Angle from sub-radar point (degrees)')
        plt.ylabel('Cumulative Power')
        plt.title('Cumulative Power Distribution vs. Angle')
        plt.legend()
        plt.grid(True)
        plt.show()

    # Create and return the results object
    results = DopplerResults(
        simple_spread=simple_doppler_spread,
        model_spread=model_doppler_spread,
        observed_spread=observed_spread,
        power_distribution=power_distribution if plot else None,
        frequency_bins=freq_bins if plot else None
    )

    return results



def match_observed_spread(
    site_params: object,
    observed_spread: float,
```

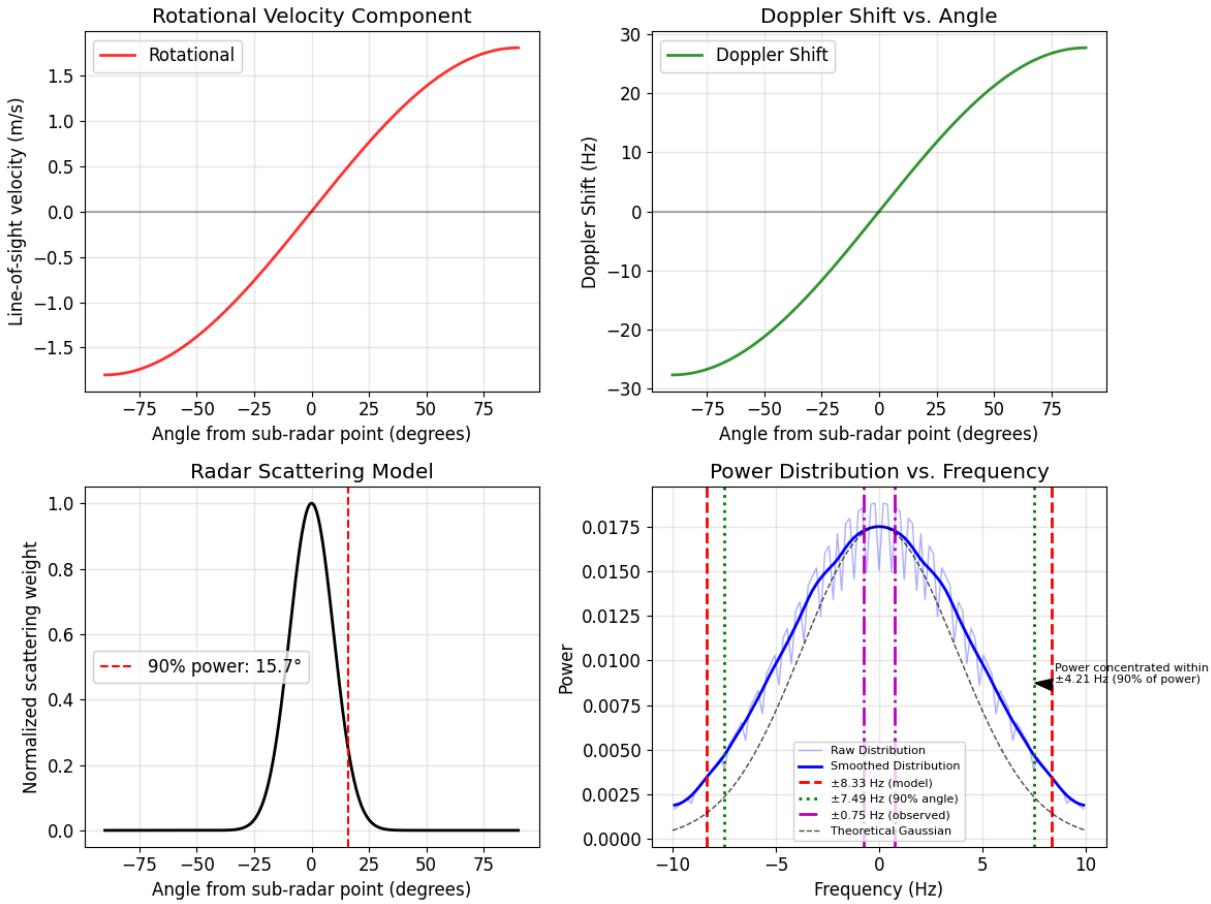
```
    venus_params: VenusParameters = VenusParameters(),
    debug: bool = False  # Add debug parameter here too!
) -> Tuple[float, float]:
    """
    Find model parameters that would match the observed Doppler spread.

    Parameters:
    -----------
    site_params : object
        Site parameters class (DSESLinkParameters, DwingelooLinkParameters,
    observed_spread : float
        Observed Doppler spread to match
    venus_params : VenusParameters
        Physical parameters of Venus
    debug : bool
        Whether to print detailed debug information

    Returns:
    --------
    Tuple[float, float]
        Best (orbital_angle_factor, scattering_power) to match observations
    """

    # Calculate simple spread for reference
    c = site_params.c
    freq_hz = site_params.tx_frequency_mhz * 1e6
    venus_rot_velocity = venus_params.rotation_velocity

    simple_doppler_spread = 4 * venus_rot_velocity / c * freq_hz

    print("\nModel parameter adjustment to match observations:")
    print(f"Target: {observed_spread:.2f} Hz (±{observed_spread/2:.2f} Hz)")

    # Parameter ranges to try - expanded for better coverage
    orbital_angle_factors = np.linspace(0.01, 0.20, 10)
    scattering_powers = np.linspace(5, 40, 8)

    best_diff = float('inf')
    best_params = None

    for orbital_factor in orbital_angle_factors:
        for scattering_power in scattering_powers:
            # Recalculate with different parameters to test
            doppler_params = DopplerParameters(
                orbital_angle_factor=orbital_factor,
                scattering_power=scattering_power
            )


            # Do a quick run without plotting
            results = calculate_venus_doppler_spread(
                site_params,
                doppler_params,
                venus_params,
                plot=False,
                debug=debug # Pass debug parameter from outside
```

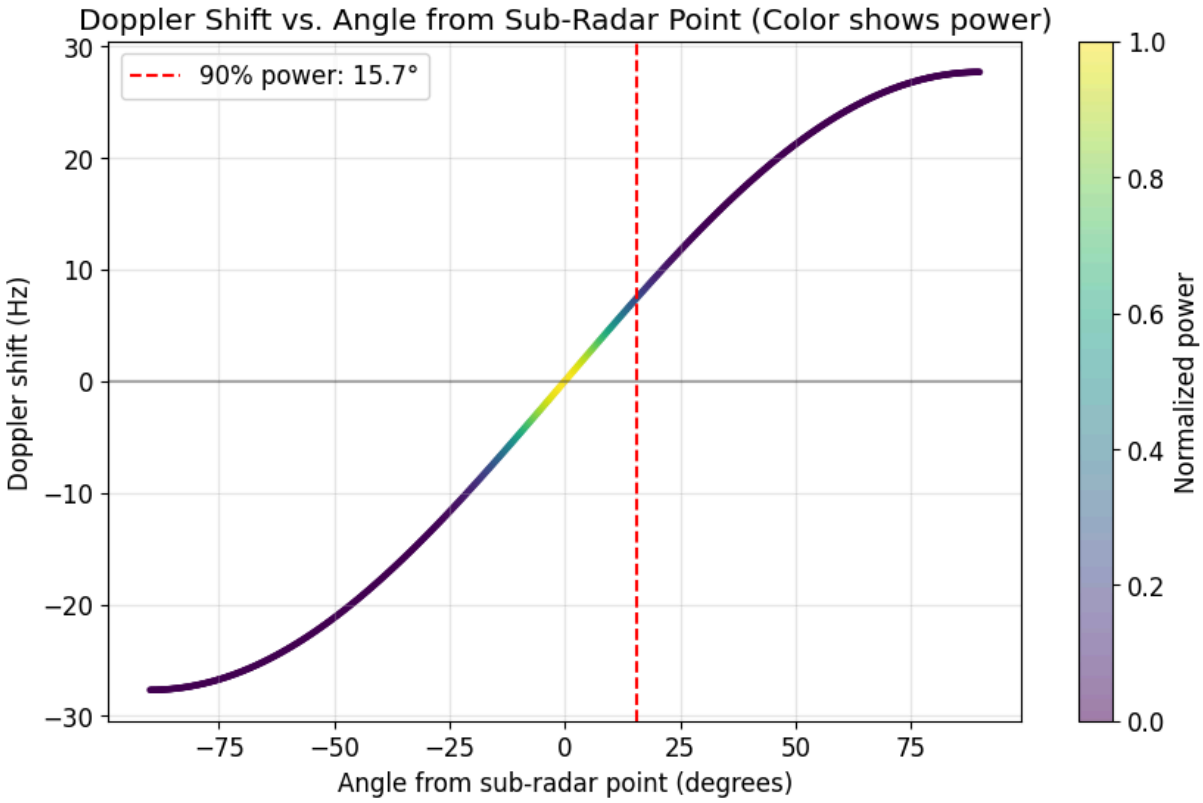
```
            )

            diff = abs(results.model_spread - observed_spread)

            if diff < best_diff:
                best_diff = diff
                best_params = (orbital_factor, scattering_power)
                if debug:
                    print(f"New best match: OF={orbital_factor:.3f}, SP={sca

    print(f"\nFinal best parameters: orbital_angle_factor = {best_params[0]:
          f"scattering_power = {best_params[1]:.2f}")
    print(f"These parameters would give a spread close to the observed value

    return best_params


def compare_frequencies(site_params_class):
    """
    Compare expected Doppler spread at different frequencies

    Parameters:
    -----------
    site_params_class : class
        The site parameter class to use (e.g., DSESLinkParameters)

    Returns:
    --------
    Tuple[DopplerResults, DopplerResults]
        Results for 1296 MHz and 2304 MHz
    """
    # Base site parameters
    site_params_1296 = site_params_class(tx_frequency_mhz=1296.0)
    site_params_2304 = site_params_class(tx_frequency_mhz=2304.0)

    # Extract site name from class name
    site_name = site_params_class.__name__.replace("LinkParameters", "")

    # Venus parameters
    venus_params = VenusParameters()

    # Basic Doppler parameters (no observed value)
    doppler_params = DopplerParameters(
        orbital_angle_factor=0.05,
        scattering_power=18.0  # Using improved scattering power value
    )

    # Calculate for 1296 MHz
    results_1296 = calculate_venus_doppler_spread(
        site_params_1296,
        doppler_params,
        venus_params,
        plot=True,
        site_name=f"{site_name} @ 1296 MHz"
    )
```

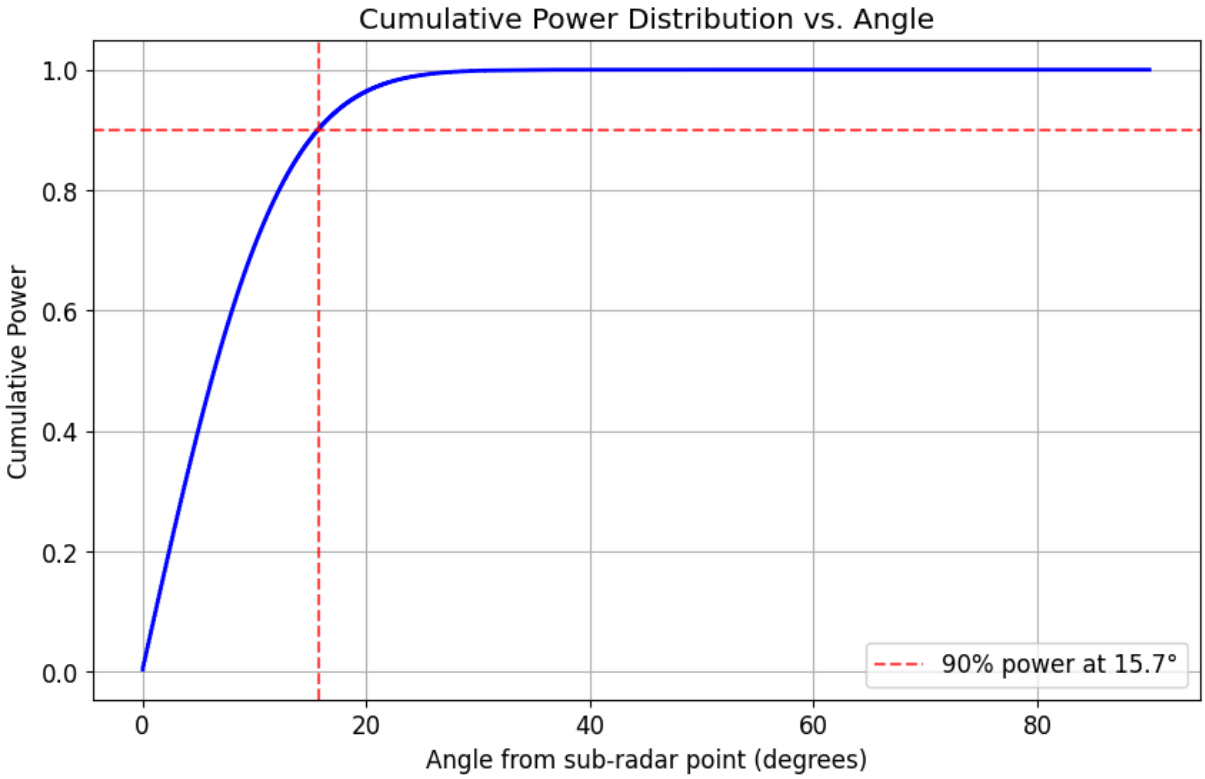
```
    # Calculate for 2304 MHz
    results_2304 = calculate_venus_doppler_spread(
        site_params_2304,
        doppler_params,
        venus_params,
        plot=True,
        site_name=f"{site_name} @ 2304 MHz"
    )

    # Print comparison
    print("\nDoppler Spread Comparison:")
    print(f"1296 MHz: Simple ±{results_1296.simple_spread/2:.2f} Hz, Model ±
    print(f"2304 MHz: Simple ±{results_2304.simple_spread/2:.2f} Hz, Model ±
    print(f"Ratio (2304/1296): {results_2304.simple_spread/results_1296.simp

    return results_1296, results_2304


# Example usage
if __name__ == "__main__":
    # Set a single debug flag for the entire script
    debug_mode = False  # Set to True or False as needed

    # Choose which site to use
    SiteLinkParameters = DSESLinkParameters  # Use DSES parameters
    # SiteLinkParameters = DwingelooLinkParameters  # Or use Dwingeloo param

    # Create site parameters
    site_params = SiteLinkParameters()

    # Extract site name for display
    site_name = SiteLinkParameters.__name__.replace("LinkParameters", "")

    # Venus parameters (using defaults)
    venus_params = VenusParameters()

    # Create Doppler parameters with Dr. Estevez's observed spread
    doppler_params = DopplerParameters(
        orbital_angle_factor=0.05,
        scattering_power=18.0,  # Using improved value
        observed_doppler_spread=1.5  # Hz (±0.75 Hz)
    )

    # Calculate Doppler spread (only once, with the debug mode set)
    results = calculate_venus_doppler_spread(
        site_params,
        doppler_params,
        venus_params,
        plot=True,
        site_name=f"{site_name}",
        debug=debug_mode
    )

    print("\nSummary of results:")
    print(results)
```

```
    # Find parameters to match observations (pass the same debug mode)
    best_orbital_factor, best_scattering_power = match_observed_spread(
        site_params,
        doppler_params.observed_doppler_spread,
        venus_params,
        debug=debug_mode
    )

    # Create a new parameters object with the optimized values
    optimized_doppler_params = DopplerParameters(
        orbital_angle_factor=best_orbital_factor,
        scattering_power=best_scattering_power,
        observed_doppler_spread=doppler_params.observed_doppler_spread
    )

    # Recalculate with optimized parameters (with the same debug mode)
    optimized_results = calculate_venus_doppler_spread(
        site_params,
        optimized_doppler_params,
        venus_params,
        plot=True,
        site_name=f"{site_name} (Optimized)",
        debug=debug_mode
    )

    print("\nResults with optimized parameters:")
    print(optimized_results)

    # print just our optimized Doppler Spread
    print(f"Optimized Doppler spread: {optimized_results.model_spread:.2f} H

    # overwrite simple model of Doppler Spread, for subsequent calculations
    doppler_spread = optimized_results.model_spread

    # Uncomment to run frequency comparison for the selected site
    #results_1296, results_2304 = compare_frequencies(SiteLinkParameters)

    # Compare different sites at the same frequency
    # Uncomment to compare DSES and Dwingeloo
    """
    print("\n*** Comparing Different Sites ***")

    # DSES parameters
    dses_params = DSESLinkParameters()
    dses_results = calculate_venus_doppler_spread(
        dses_params,
        doppler_params,
        venus_params,
        plot=True,
        site_name="DSES"
    )

    # Dwingeloo parameters
    dwingeloo_params = DwingelooLinkParameters()
    dwingeloo_results = calculate_venus_doppler_spread(
        dwingeloo_params,
```

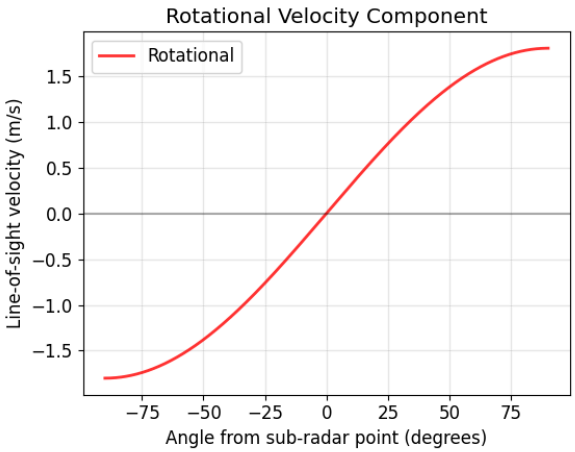
```
        doppler_params,
        venus_params,
        plot=True,
        site_name="Dwingeloo"
    )

    print("\nSite Comparison:")
    print(f"DSES: Simple ±{dses_results.simple_spread/2:.2f} Hz, Model ±{dse
    print(f"Dwingeloo: Simple ±{dwingeloo_results.simple_spread/2:.2f} Hz, M
    """
```

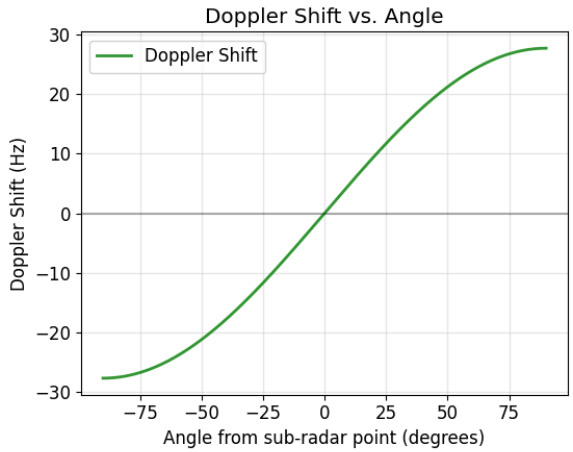


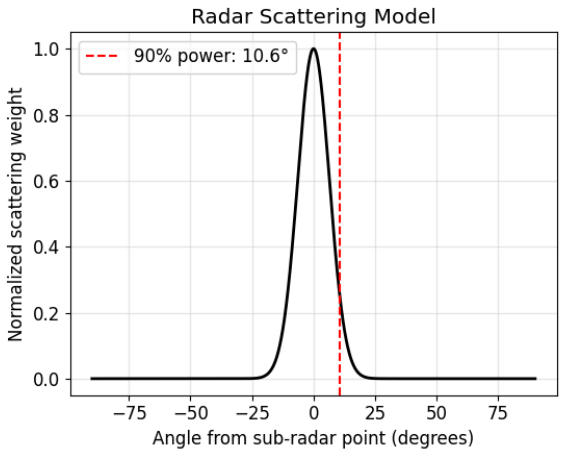

Doppler Shift vs. Angle from Sub-Radar Point (Color shows power)
90% power: 15.7°
Doppler shift (Hz)
Angle from sub-radar point (degrees)
Normalized power
30
20
10
0
−10
−20
−30
−75
−50
−25
0
25
50
75
1.0
0.8
0.6
0.4
0.2
0.0

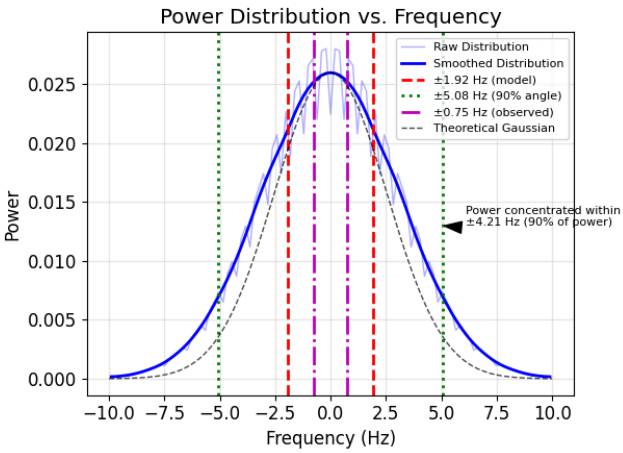

Cumulative Power Distribution vs. Angle
Cumulative Power
Angle from sub-radar point (degrees)
90% power at 15.7°
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80

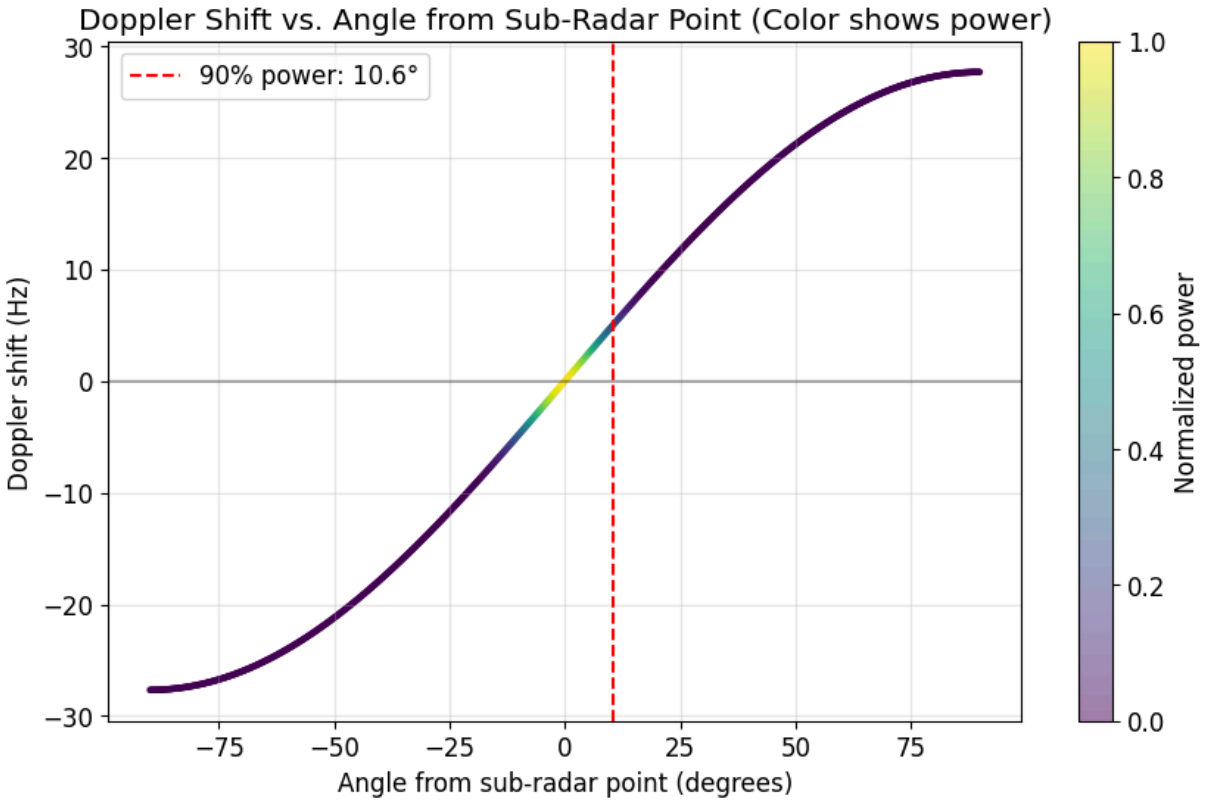
```
Summary of results:
Simple calculation (K6MG): ±27.67 Hz
Model calculation (90% power): ±8.33 Hz
Observed (Dr. Estevez): ±0.75 Hz
Ratio of observed to simple: 2.71%

Model parameter adjustment to match observations:
Target: 1.50 Hz (±0.75 Hz)

Final best parameters: orbital_angle_factor = 0.0100, scattering_power = 40.
00
These parameters would give a spread close to the observed value of 1.50 Hz
```

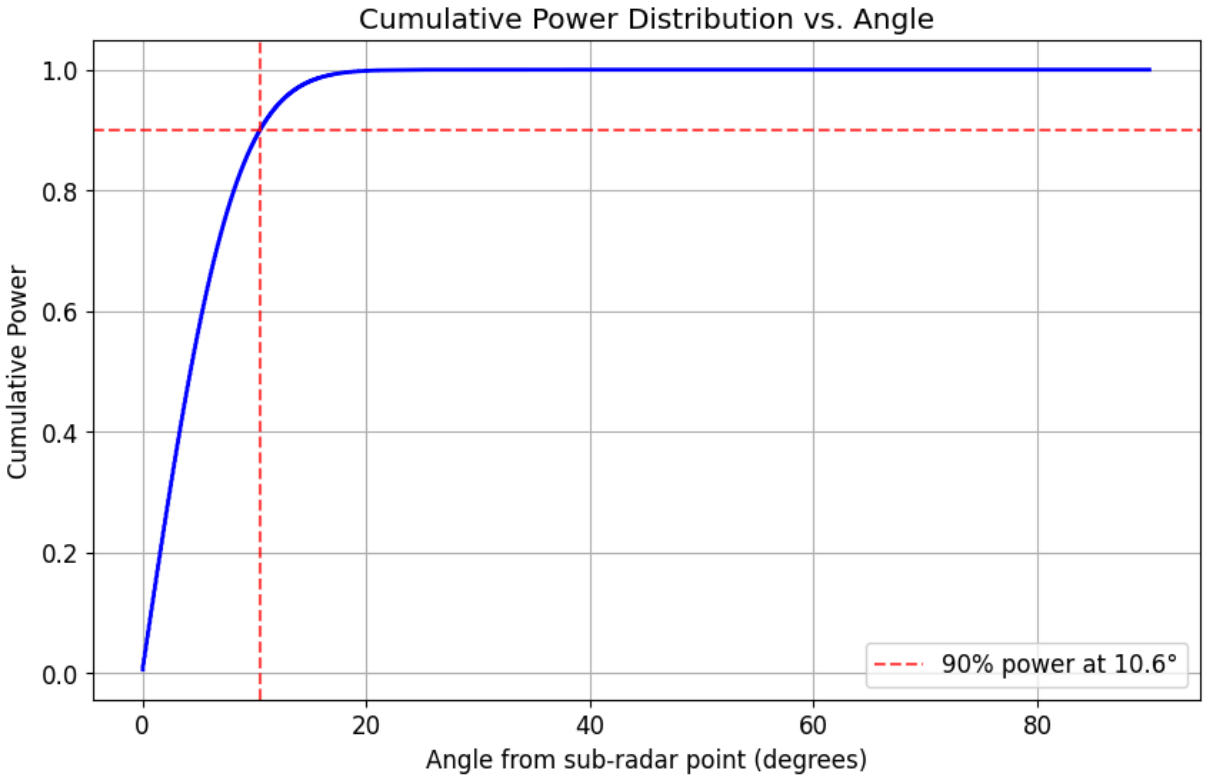













```
Results with optimized parameters:
Simple calculation (K6MG): ±27.67 Hz
Model calculation (90% power): ±1.92 Hz
Observed (Dr. Estevez): ±0.75 Hz
Ratio of observed to simple: 2.71%
Optimized Doppler spread: 3.84 Hz
```

Mode Analysis

This class does an analysis of potential amateur digital communications modes and their suitability for the link.

The relationship between the CNR from the link budget object (calculated at the operational receiver bandwidth) and SNR of particular signals (given at different bandwidths) is not always straightforward. The listed SNRs for many amateur modes are given with an occupied bandwidth, but the given SNR is not calculated at that bandwidth, but is calculated at another "normalized" bandwidth. The most common "normalized" bandwidth is 2500 Hz. When we know this is the case, we can list this number (2500 Hz) as the "real" noise bandwidth as the noise_bandwidth_hz value.

We use the optimized results from the Doppler Spread model in the previous section. This most closely matches observed results from the March 2025 EVE event.

Coherent vs. Non-coherent Modes:

Coherent modes maintain phase continuity throughout signal reception and rely on precise carrier phase tracking for demodulation. They typically use integration over symbol periods where the received signal phase must remain stable. Examples include PSK, QAM, and other phase-sensitive modulations.

Non-coherent modes do not require phase continuity and detect signals based on energy or frequency presence rather than precise phase relationships. These modes are more robust against phase disturbances but typically require higher SNR. Examples include FSK, MFSK, and envelope detection schemes.

The Doppler spread penalty for non-coherent modes is calculated by determining what percentage of the signal's energy falls outside the mode's bandwidth. For non-coherent modes with bandwidth much wider than the Doppler spread, almost all energy is contained within the bandwidth and there's minimal penalty. If a mode's bandwidth is narrower than the Doppler spread, more energy falls outside the usable bandwidth, increasing the penalty. The modes most affected by Doppler spread penalty are narrow-band modes.

The calculation is: `doppler_penalty_db = 10 * log10(doppler_spread_hz / mode["bandwidth_hz"])`

The larger the mode bandwidth, the less the Doppler spread penalty.

This Gaussian approach is more accurate because it accounts for the concentration of energy near the carrier frequency with decreasing energy at the edges, which corresponds to the probability distribution of relative velocities in the signal path.

The Doppler spread penalty for coherently integrated modes is calculated not by dividing Doppler spread in Hz by the mode bandwidth, but by multiplying Doppler spread in Hz by the integration time.

The calculation is: `doppler_penalty_db = 10 * log10(doppler_spread_hz * symbol_duration)`

The longer the integration time, the greater the Doppler spread penalty.

Do any modes close the link? What happens when we include Doppler spread penalties?

In [225…

```
#import numpy as np
#import pandas as pd

class AmateurRadioModeEvaluator:
    """
    A class to evaluate the suitability of amateur radio modes based on link

    def __init__(self, params: SiteLinkParameters):
        self.params = params
        # Define common amateur radio modes with their characteristics
        self.modes = [
            {"name": "CW", "bandwidth_hz": 250, "required_snr_db": -15, "noi
            {"name": "FT8", "bandwidth_hz": 50, "required_snr_db": -20, "noi
            {"name": "JT65", "bandwidth_hz": 2.7, "required_snr_db": -25, "r
            {"name": "SSB", "bandwidth_hz": 2500, "required_snr_db": 8, "noi
            {"name": "FM", "bandwidth_hz": 12500, "required_snr_db": 12, "no
            {"name": "RTTY", "bandwidth_hz": 250, "required_snr_db": 5, "noi
            {"name": "PSK31", "bandwidth_hz": 31, "required_snr_db": 4, "noi
            {"name": "FT4", "bandwidth_hz": 90, "required_snr_db": -17, "noi
            {"name": "JS8", "bandwidth_hz": 30, "required_snr_db": -18, "noi

            # WSPR Modes
            {"name": "WSPR-15", "bandwidth_hz": 6, "required_snr_db": -32, "
            {"name": "WSPR-2", "bandwidth_hz": 6, "required_snr_db": -28, "r
            {"name": "WSPR-120", "bandwidth_hz": 6, "required_snr_db": -37,
            {"name": "WSPR-LF", "bandwidth_hz": 6, "required_snr_db": -30, "
            {"name": "WSPR-H", "bandwidth_hz": 12, "required_snr_db": -26, "

            # Q65 modes (various submodes A-E with different tone spacing ar
            {"name": "Q65-15A", "bandwidth_hz": 65, "required_snr_db": -26,
            {"name": "Q65-30A", "bandwidth_hz": 65, "required_snr_db": -27,
            {"name": "Q65-60A", "bandwidth_hz": 65, "required_snr_db": -28,
            {"name": "Q65-120A", "bandwidth_hz": 65, "required_snr_db": -29,
            {"name": "Q65-300A", "bandwidth_hz": 65, "required_snr_db": -30,

            {"name": "Q65-15B", "bandwidth_hz": 90, "required_snr_db": -26,
            {"name": "Q65-30B", "bandwidth_hz": 90, "required_snr_db": -27,
            {"name": "Q65-60B", "bandwidth_hz": 90, "required_snr_db": -28,
            {"name": "Q65-120B", "bandwidth_hz": 90, "required_snr_db": -29,
            {"name": "Q65-300B", "bandwidth_hz": 90, "required_snr_db": -30,

            {"name": "Q65-15C", "bandwidth_hz": 180, "required_snr_db": -26,
```

```
            {"name": "Q65-30C", "bandwidth_hz": 180, "required_snr_db": -27,
            {"name": "Q65-60C", "bandwidth_hz": 180, "required_snr_db": -28,
            {"name": "Q65-120C", "bandwidth_hz": 180, "required_snr_db": -29
            {"name": "Q65-300C", "bandwidth_hz": 180, "required_snr_db": -30

            {"name": "Q65-15D", "bandwidth_hz": 360, "required_snr_db": -26,
            {"name": "Q65-30D", "bandwidth_hz": 360, "required_snr_db": -27,
            {"name": "Q65-60D", "bandwidth_hz": 360, "required_snr_db": -28,
            {"name": "Q65-120D", "bandwidth_hz": 360, "required_snr_db": -29
            {"name": "Q65-300D", "bandwidth_hz": 360, "required_snr_db": -30

            {"name": "Q65-15E", "bandwidth_hz": 720, "required_snr_db": -26,
            {"name": "Q65-30E", "bandwidth_hz": 720, "required_snr_db": -27,
            {"name": "Q65-60E", "bandwidth_hz": 720, "required_snr_db": -28,
            {"name": "Q65-120E", "bandwidth_hz": 720, "required_snr_db": -29
            {"name": "Q65-300E", "bandwidth_hz": 720, "required_snr_db": -30

            # FST4 Modes
            {"name": "FST4-15", "bandwidth_hz": 67, "required_snr_db": -21,
            {"name": "FST4-30", "bandwidth_hz": 29, "required_snr_db": -24,
            {"name": "FST4-60", "bandwidth_hz": 12, "required_snr_db": -28,
            {"name": "FST4-120", "bandwidth_hz": 6, "required_snr_db": -31,
            {"name": "FST4-300", "bandwidth_hz": 2, "required_snr_db": -35,
            {"name": "FST4-900", "bandwidth_hz": 0.7, "required_snr_db": -40
            {"name": "FST4-1800", "bandwidth_hz": 0.4, "required_snr_db": -4
            {"name": "FST4W-120", "bandwidth_hz": 6, "required_snr_db": -32,
            {"name": "FST4W-300", "bandwidth_hz": 2, "required_snr_db": -37,
            {"name": "FST4W-900", "bandwidth_hz": 0.7, "required_snr_db": -4
            {"name": "FST4W-1800", "bandwidth_hz": 0.4, "required_snr_db": -
        ]

    def add_mode(self, name, bandwidth_hz, required_snr_db, noise_bandwidth_
        """
        Add a new mode to the evaluator.

        Parameters:
        name (str): Name of the mode
        bandwidth_hz (float): Bandwidth of the mode in Hz
        required_snr_db (float): Required SNR in dB for the mode to functior
        noise_bandwidth_hz (float): over what bandwidth is the required_snr_
        """
        new_mode = {
            "name": name,
            "bandwidth_hz": bandwidth_hz,
            "required_snr_db": required_snr_db,
            "noise_bandwidth_hz": noise_bandwidth_hz
        }
        self.modes.append(new_mode)

    def evaluate_modes(self, doppler_spread_hz=None):
        """
        Evaluate suitability of amateur radio modes based on link CNR

        Parameters:
        cnr_db (float): Carrier-to-Noise Ratio in dB already calculated at r
        receiver_noise_bandwidth (float): Receiver noise bandwidth in Hz use
```

```
        doppler_spread_hz (float, optional): Doppler spread in Hz, if we hav

        Returns:
        DataFrame: Modes with suitability assessment
        """
        results = []

        # Get the CNR in 1 Hz result from link budget calculator
        print(f"from the Link Budget calculator, our min cnr_db_1hz is {min_

        for mode in self.modes:
            # Scale each mode SNR from noise bandwidth to 1Hz bandwidth
            # Some SNRs are taken really at 2500 Hz, and some are not.
            # We made a table column to handle this.
            # noise_bandwidth_hz is the "real" number to scale by.
            snr_db_1hz = mode["required_snr_db"] + 10 * np.log10(mode["noise

            # Apply Doppler spread penalty if applicable using a Gaussian mo
            # set penalty to zero just in case things get weird
            doppler_penalty_db = 0

            if doppler_spread_hz is not None and doppler_spread_hz > 0:
                # Determine if the mode uses coherent integration
                # Q65, JT65, FST4, WSPR, and other digital modes use coheren
                uses_coherent_integration = any(prefix in mode["name"] for p

                if uses_coherent_integration:
                    # Extract symbol duration based on mode name (avoids ext
                    # For Q65 modes, extract the time from the name (e.g., Q
                    if "Q65" in mode["name"]:
                        # Extract time value between "Q65-" and the letter a
                        time_str = mode["name"].split("-")[1][:-1]  # Remove
                        symbol_duration = float(time_str)
                    elif "FST4" in mode["name"] or "WSPR" in mode["name"]:
                        # Extract time value after the dash
                        time_str = mode["name"].split("-")[1]
                        # Handle special case for WSPR-LF and WSPR-H
                        if time_str.isdigit() or time_str.isnumeric():
                            symbol_duration = float(time_str)
                        else:
                            # Default values if we can't extract
                            symbol_duration = 120 if "WSPR" in mode["name"]
                    elif "FT8" in mode["name"]:
                        symbol_duration = 15
                    elif "FT4" in mode["name"]:
                        symbol_duration = 7.5
                    elif "JT65" in mode["name"]:
                        symbol_duration = 60
                    else:
                        # Default to a reasonable value if we can't determin
                        symbol_duration = 60

                    # Calculate Doppler penalty using the coherent integrati
                    # Use the symbol_duration based approach recommended by
                    '''
                    The condition if doppler_spread_hz * symbol_duration > 1
```

```
                represents an important threshold in communication theor
                called the "coherence threshold" or "spreading factor th

                When a signal experiences Doppler spread, the phase of t
                changes during reception. The product doppler_spread_hz
                tells you how many cycles of phase change occur during c

                If this product is less than 1, it means the phase doesr
                complete a full cycle during the symbol duration. In thi
                the coherent detection can still work effectively with m
                So we test for this and let it go with 0 dB penalty.

                If this product is greater than 1, it means the phase cc
                more full cycles during a single symbol. This causes sig
                degradation in coherent detection because the phase char
                So we apply the Doppler spread penalty.
                '''
                if doppler_spread_hz * symbol_duration > 1:
                    doppler_penalty_db = 10 * math.log10(doppler_spread_
                    doppler_penalty_db = min(doppler_penalty_db, 200)  #
                else:
                    doppler_penalty_db = 0
            else:
                # For non-coherent modes (CW, SSB, RTTY, etc.)
                # Use the original bandwidth-based approach and Gaussian
                if doppler_spread_hz > mode["bandwidth_hz"]:
                    doppler_penalty_db = 10 * math.log10(doppler_spread_
                    doppler_penalty_db = min(doppler_penalty_db, 200)  #
                else:
                    doppler_penalty_db = 0









        # Calculate final effective SNR in 1Hz with Doppler penalty
        effective_snr_db_1hz = snr_db_1hz + doppler_penalty_db

        # Calculate margin between CNR in 1Hz and required SNR for mode
        margin_db = min_results['cnr_db_1hz'] - effective_snr_db_1hz

        # Determine reliability level
        if margin_db >= 10:
            reliability = "Excellent"
        elif margin_db >= 6:
            reliability = "Very Good"
        elif margin_db >= 3:
            reliability = "Good"
        elif margin_db >= 0:
            reliability = "Marginal"
        else:
            reliability = "Not Feasible"
```

```
            result = {
                "Mode": mode["name"],
                "Mode BW (Hz)": mode["bandwidth_hz"],
                "Mode SNR (dB)": mode["required_snr_db"],
                "Mode SNR 1hz (dB)": round(effective_snr_db_1hz, 1),
                "Margin (dB)": round(margin_db, 1),
                "Reliability": reliability,
                "Feasible": margin_db >= 0
            }

            # Add Doppler information if provided
            if doppler_spread_hz is not None:
                result["Doppler Spread (Hz)"] = doppler_spread_hz
                result["Doppler Penalty (dB)"] = round(doppler_penalty_db, 1

            results.append(result)

        # Convert to DataFrame for easy display
        results_df = pd.DataFrame(results)

        # Sort by margin (highest first)
        results_df = results_df.sort_values(by="Margin (dB)", ascending=Fals

        return results_df

    def filter_by_mode_type(self, results_df, mode_type):
        """
        Filter results by mode type (e.g., 'Q65', 'WSPR', etc.)

        Parameters:
        results_df (DataFrame): Results from evaluate_modes
        mode_type (str): Mode type to filter for

        Returns:
        DataFrame: Filtered results
        """
        return results_df[results_df['Mode'].str.contains(mode_type)]

    def get_feasible_modes(self, results_df):
        """
        Get only feasible modes from results

        Parameters:
        results_df (DataFrame): Results from evaluate_modes

        Returns:
        DataFrame: Only feasible modes
        """
        return results_df[results_df['Feasible'] == True]



# Example usage:
params = SiteLinkParameters()
evaluator = AmateurRadioModeEvaluator(params)
results = evaluator.evaluate_modes()
```

```
print('Results Without Considering Doppler Spread:\n')
print(results.to_string())

print(f"\n\n\n")

# # With Doppler:
print('Results Including Doppler Spread:\n')
# use current Doppler Spread
#doppler_spread_hz = DopplerSpreadCalculator.venus_doppler_spread()
doppler_spread_hz = optimized_results.model_spread
# run evaluator with calculated Doppler Spread
results_with_doppler = evaluator.evaluate_modes(doppler_spread_hz)

print(results_with_doppler.to_string())

print(f"\n\n\n")

# # Filter for specific mode types:
print('Results Filtered for Q65 Modes:\n')
q65_modes = evaluator.filter_by_mode_type(results, 'Q65')
print(q65_modes.to_string())

print(f"\n\n\n")

# # Only feasible modes - no Doppler spread
print('Results Filtered for Feasible Modes Without Considering Doppler Sprea
only_feasible_results = evaluator.get_feasible_modes(results)
print(only_feasible_results.to_string())

print(f"\n\n\n")

# # Only feasible modes - with Doppler
print('Results Filtered for Feasible Modes Including Doppler Spread:\n')
only_feasible_results = evaluator.get_feasible_modes(results_with_doppler)
print(only_feasible_results.to_string())
```

from the Link Budget calculator, our min cnr_db_1hz is -8.89
Results Without Considering Doppler Spread:

	Mode	Mode BW (Hz)	Mode SNR (dB)	Mode SNR 1hz (dB)	Margin (dB)	Reliability	Feasible
49	FST4W-1800	0.4	-45	-11.0	2.1	Marginal	True
45	FST4-1800	0.4	-43	-9.0	0.1	Marginal	True
48	FST4W-900	0.7	-42	-8.0	-0.9	Not Feasible	False
44	FST4-900	0.7	-40	-6.0	-2.9	Not Feasible	False
47	FST4W-300	2.0	-37	-3.0	-5.9	Not Feasible	False
11	WSPR-120	6.0	-37	-3.0	-5.9	Not Feasible	False
43	FST4-300	2.0	-35	-1.0	-7.9	Not Feasible	False
9	WSPR-15	6.0	-32	2.0	-10.9	Not Feasible	False
46	FST4W-120	6.0	-32	2.0	-10.9	Not Feasible	False
42	FST4-120	6.0	-31	3.0	-11.9	Not Feasible	False
28	Q65-300C	180.0	-30	4.0	-12.9	Not Feasible	False
12	WSPR-LF	6.0	-30	4.0	-12.9	Not Feasible	False
38	Q65-300E	720.0	-30	4.0	-12.9	Not Feasible	False
33	Q65-300D	360.0	-30	4.0	-12.9	Not Feasible	False
23	Q65-300B	90.0	-30	4.0	-12.9	Not Feasible	False
18	Q65-300A	65.0	-30	4.0	-12.9	Not Feasible	False
22	Q65-120B	90.0	-29	5.0	-13.9	Not Feasible	False
27	Q65-120C	180.0	-29	5.0	-13.9	Not Feasible	False
32	Q65-120D	360.0	-29	5.0	-13.9	Not Feasible	False
37	Q65-120E	720.0	-29	5.0	-13.9	Not Feasible	False
17	Q65-120A	65.0	-29	5.0	-13.9	Not Feasible	False
21	Q65-60B	90.0	-28	6.0	-14.9	Not Feasible	False
36	Q65-60E	720.0	-28	6.0	-14.9	Not Feasible	False
16	Q65-60A	65.0	-28	6.0	-14.9	Not Feasible	False
26	Q65-60C	180.0	-28	6.0	-14.9	Not Feasible	False
41	FST4-60	12.0	-28	6.0	-14.9		

```
Not Feasible     False
10      WSPR-2           6.0            -28                6.0        -14.9
Not Feasible     False
31     Q65-60D         360.0            -28                6.0        -14.9
Not Feasible     False
35     Q65-30E         720.0            -27                7.0        -15.9
Not Feasible     False
30     Q65-30D         360.0            -27                7.0        -15.9
Not Feasible     False
25     Q65-30C         180.0            -27                7.0        -15.9
Not Feasible     False
20     Q65-30B          90.0            -27                7.0        -15.9
Not Feasible     False
15     Q65-30A          65.0            -27                7.0        -15.9
Not Feasible     False
29     Q65-15D         360.0            -26                8.0        -16.9
Not Feasible     False
13      WSPR-H          12.0            -26                8.0        -16.9
Not Feasible     False
24     Q65-15C         180.0            -26                8.0        -16.9
Not Feasible     False
34     Q65-15E         720.0            -26                8.0        -16.9
Not Feasible     False
19     Q65-15B          90.0            -26                8.0        -16.9
Not Feasible     False
14     Q65-15A          65.0            -26                8.0        -16.9
Not Feasible     False
2         JT65           2.7            -25                9.0        -17.9
Not Feasible     False
0           CW         250.0            -15                9.0        -17.9
Not Feasible     False
40     FST4-30          29.0            -24               10.0        -18.9
Not Feasible     False
39     FST4-15          67.0            -21               13.0        -21.9
Not Feasible     False
1          FT8          50.0            -20               14.0        -22.9
Not Feasible     False
8          JS8          30.0            -18               16.0        -24.9
Not Feasible     False
7          FT4          90.0            -17               17.0        -25.9
Not Feasible     False
6        PSK31          31.0              4               18.9        -27.8
Not Feasible     False
5         RTTY         250.0              5               29.0        -37.9
Not Feasible     False
3          SSB        2500.0              8               42.0        -50.9
Not Feasible     False
4           FM       12500.0             12               53.0        -61.9
Not Feasible     False
```

```
Results Including Doppler Spread:

from the Link Budget calculator, our min cnr_db_1hz is -8.89
```

	Mode	Mode BW (Hz)	Mode SNR (dB)	Mode SNR 1hz (dB)	Margin (dB)	Reliability	Feasible	Doppler Spread (Hz)	Doppler Penalty (dB)
0	CW	250.0	-15	9.0	-17.9	Not Feasible	False	3.844817	0.0
10	WSPR-2	6.0	-28	14.8	-23.7	Not Feasible	False	3.844817	8.9
8	JS8	30.0	-18	16.0	-24.9	Not Feasible	False	3.844817	0.0
6	PSK31	31.0	4	18.9	-27.8	Not Feasible	False	3.844817	0.0
9	WSPR-15	6.0	-32	19.6	-28.5	Not Feasible	False	3.844817	17.6
11	WSPR-120	6.0	-37	23.6	-32.5	Not Feasible	False	3.844817	26.6
14	Q65-15A	65.0	-26	25.6	-34.5	Not Feasible	False	3.844817	17.6
34	Q65-15E	720.0	-26	25.6	-34.5	Not Feasible	False	3.844817	17.6
29	Q65-15D	360.0	-26	25.6	-34.5	Not Feasible	False	3.844817	17.6
24	Q65-15C	180.0	-26	25.6	-34.5	Not Feasible	False	3.844817	17.6
19	Q65-15B	90.0	-26	25.6	-34.5	Not Feasible	False	3.844817	17.6
48	FST4W-900	0.7	-42	27.4	-36.3	Not Feasible	False	3.844817	35.4
49	FST4W-1800	0.4	-45	27.4	-36.3	Not Feasible	False	3.844817	38.4
25	Q65-30C	180.0	-27	27.6	-36.5	Not Feasible	False	3.844817	20.6
30	Q65-30D	360.0	-27	27.6	-36.5	Not Feasible	False	3.844817	20.6
15	Q65-30A	65.0	-27	27.6	-36.5	Not Feasible	False	3.844817	20.6
47	FST4W-300	2.0	-37	27.6	-36.5	Not Feasible	False	3.844817	30.6
20	Q65-30B	90.0	-27	27.6	-36.5	Not Feasible	False	3.844817	20.6
35	Q65-30E	720.0	-27	27.6	-36.5	Not Feasible	False	3.844817	20.6
46	FST4W-120	6.0	-32	28.6	-37.5	Not Feasible	False	3.844817	26.6
5	RTTY	250.0	5	29.0	-37.9	Not Feasible	False	3.844817	0.0
45	FST4-1800	0.4	-43	29.4	-38.3	Not Feasible	False	3.844817	38.4
44	FST4-900	0.7	-40	29.4	-38.3	Not Feasible	False	3.844817	35.4
43	FST4-300	2.0	-35	29.6	-38.5	Not Feasible	False	3.844817	30.6
42	FST4-120	6.0	-31	29.6	-38.5	Not Feasible	False	3.844817	26.6
41	FST4-60	12.0	-28	29.6	-38.5	Not Feasible	False	3.844817	23.6
36	Q65-60E	720.0	-28	29.6	-38.5	Not Feasible	False	3.844817	23.6

```
31     Q65-60D         360.0           -28               29.6        -38.5
Not Feasible     False             3.844817                23.6
21     Q65-60B          90.0           -28               29.6        -38.5
Not Feasible     False             3.844817                23.6
16     Q65-60A          65.0           -28               29.6        -38.5
Not Feasible     False             3.844817                23.6
26     Q65-60C         180.0           -28               29.6        -38.5
Not Feasible     False             3.844817                23.6
39     FST4-15          67.0           -21               30.6        -39.5
Not Feasible     False             3.844817                17.6
12     WSPR-LF           6.0           -30               30.6        -39.5
Not Feasible     False             3.844817                26.6
40     FST4-30          29.0           -24               30.6        -39.5
Not Feasible     False             3.844817                20.6
27    Q65-120C         180.0           -29               31.6        -40.5
Not Feasible     False             3.844817                26.6
32    Q65-120D         360.0           -29               31.6        -40.5
Not Feasible     False             3.844817                26.6
1          FT8          50.0           -20               31.6        -40.5
Not Feasible     False             3.844817                17.6
37    Q65-120E         720.0           -29               31.6        -40.5
Not Feasible     False             3.844817                26.6
17    Q65-120A          65.0           -29               31.6        -40.5
Not Feasible     False             3.844817                26.6
7          FT4          90.0           -17               31.6        -40.5
Not Feasible     False             3.844817                14.6
22    Q65-120B          90.0           -29               31.6        -40.5
Not Feasible     False             3.844817                26.6
2         JT65           2.7           -25               32.6        -41.5
Not Feasible     False             3.844817                23.6
18    Q65-300A          65.0           -30               34.6        -43.5
Not Feasible     False             3.844817                30.6
28    Q65-300C         180.0           -30               34.6        -43.5
Not Feasible     False             3.844817                30.6
23    Q65-300B          90.0           -30               34.6        -43.5
Not Feasible     False             3.844817                30.6
38    Q65-300E         720.0           -30               34.6        -43.5
Not Feasible     False             3.844817                30.6
33    Q65-300D         360.0           -30               34.6        -43.5
Not Feasible     False             3.844817                30.6
13      WSPR-H          12.0           -26               34.6        -43.5
Not Feasible     False             3.844817                26.6
3          SSB        2500.0             8               42.0        -50.9
Not Feasible     False             3.844817                 0.0
4           FM       12500.0            12               53.0        -61.9
Not Feasible     False             3.844817                 0.0



Results Filtered for Q65 Modes:

        Mode  Mode BW (Hz)  Mode SNR (dB)  Mode SNR 1hz (dB)  Margin (dB)
Reliability  Feasible
28  Q65-300C         180.0            -30                4.0        -12.9  N
ot Feasible     False
```

```
38  Q65-300E         720.0           -30               4.0        -12.9  N
ot Feasible     False
33  Q65-300D         360.0           -30               4.0        -12.9  N
ot Feasible     False
23  Q65-300B          90.0           -30               4.0        -12.9  N
ot Feasible     False
18  Q65-300A          65.0           -30               4.0        -12.9  N
ot Feasible     False
22  Q65-120B          90.0           -29               5.0        -13.9  N
ot Feasible     False
27  Q65-120C         180.0           -29               5.0        -13.9  N
ot Feasible     False
32  Q65-120D         360.0           -29               5.0        -13.9  N
ot Feasible     False
37  Q65-120E         720.0           -29               5.0        -13.9  N
ot Feasible     False
17  Q65-120A          65.0           -29               5.0        -13.9  N
ot Feasible     False
21   Q65-60B          90.0           -28               6.0        -14.9  N
ot Feasible     False
36   Q65-60E         720.0           -28               6.0        -14.9  N
ot Feasible     False
16   Q65-60A          65.0           -28               6.0        -14.9  N
ot Feasible     False
26   Q65-60C         180.0           -28               6.0        -14.9  N
ot Feasible     False
31   Q65-60D         360.0           -28               6.0        -14.9  N
ot Feasible     False
35   Q65-30E         720.0           -27               7.0        -15.9  N
ot Feasible     False
30   Q65-30D         360.0           -27               7.0        -15.9  N
ot Feasible     False
25   Q65-30C         180.0           -27               7.0        -15.9  N
ot Feasible     False
20   Q65-30B          90.0           -27               7.0        -15.9  N
ot Feasible     False
15   Q65-30A          65.0           -27               7.0        -15.9  N
ot Feasible     False
29   Q65-15D         360.0           -26               8.0        -16.9  N
ot Feasible     False
24   Q65-15C         180.0           -26               8.0        -16.9  N
ot Feasible     False
34   Q65-15E         720.0           -26               8.0        -16.9  N
ot Feasible     False
19   Q65-15B          90.0           -26               8.0        -16.9  N
ot Feasible     False
14   Q65-15A          65.0           -26               8.0        -16.9  N
ot Feasible     False
```

Results Filtered for Feasible Modes Without Considering Doppler Spread:

```
          Mode  Mode BW (Hz)  Mode SNR (dB)  Mode SNR 1hz (dB)  Margin (dB)
Reliability  Feasible
```

```
49  FST4W-1800           0.4            -45             -11.0          2.1
Marginal      True
45   FST4-1800           0.4            -43              -9.0          0.1
Marginal      True
```

```
Results Filtered for Feasible Modes Including Doppler Spread:

Empty DataFrame
Columns: [Mode, Mode BW (Hz), Mode SNR (dB), Mode SNR 1hz (dB), Margin (dB),
Reliability, Feasible, Doppler Spread (Hz), Doppler Penalty (dB)]
Index: []
```

Zadoff-Chu Transmission Proposal

This section defines a class to evaluate Zadoff-Chu sequences as a proposed transmission. It calculates whether or not this transmission type can close the link.

What is a Zadoff-Chu Sequence?

Zadoff-Chu sequences are complex-valued mathematical sequences that have constant amplitude and ideal periodic autocorrelation properties. These sequences maintain zero cross-correlation between different sequences of the same length and exhibit a flat frequency response, making them extremely resilient to frequency shifts. They're widely used in radar systems and modern cellular communications (like LTE) for synchronization and channel estimation due to their optimal correlation properties and resistance to Doppler effects.

Outline of Proposal

1. Do coherent integration in Zadoff-Chu segments however long we can. The maximum time we can do coherent integration is limited by the worst cast Doppler rate of change or Doppler Spread, whichever results in the lowest number. We assume that we’re going to have to do batch processing with overlapping segments to ensure no signal is missed.
2. Do a Doppler compensation between segments.
3. Apply a sliding Doppler compensation during correlation processing
4. Combine resulting segments non-coherently until we know we can close the link.
5. Return detection result.

Return the Right Integration Time with get_integration_time()

What is the minimum integration time given our Doppler constraints? We consider two limiting factors:

1. **Doppler Rate of Change**: We calculate how long it takes to exceed a 45-degree phase shift using the equation sqrt(0.25/Doppler rate of change). This equation is from "Fundamentals of Radar Signal Processing" by Mark Andrew Richards, 2005.
2. **Doppler Spread**: We calculate the coherence time as 1/Doppler spread. Beyond this time, frequency components within the spread begin to destructively interfere.

We use the smaller (more conservative) of these two limits to ensure optimal coherent processing. This approach ensures we maintain phase coherence while also respecting the frequency coherence limitations imposed by the Doppler spread. This number is the "real" maximum integration time we can safely use.

Get Number of Chips with get_number_chips()

Number of chips in our coherent integration is (chip rate * maximum coherent integration time).

Find Processing Gain with get_processing_gain()

Processing gain: is 10 * log10(number of chips in our coherent integration)

This is a large number. How can this be so high? By coherently integrating, for example, 5 MHz of bandwidth over 1.34 seconds, we're concentrating the energy of 6.7 million independent measurements into a single detection decision.

Zadoff-Chu sequences have ideal auto-correlation properties, meaning they achieve the theoretical maximum processing gain. This is unlike other modulation schemes which suffer various losses due to inefficiencies that Zadoff-Chu sequences simply do not have. This is not without precedent or wildly made-up. NASA's deep space network uses comparable processing gain to communicate with distant spacecraft at extremely low bit rates. Radio astronomers routinely detect signals far below the noise floor using long integration times and correlation techniques like this. It’s very likely that the radio astronomy people at Dwingeloo, DSES, and other amateur sites are already familiar with this technique and structure. This part of the link budget worksheet is an attempt to tune the technique for EVE from amateur sites with achievable configurations and parameters.

This proposal is designed around a calculated worst case Doppler rate limitation for Venus of -0.14 Hz/second, ensuring optimal coherent processing during the worst case channel condition, which happens to occur at inferior conjunction.

Is Processing Gain Enough? Compare to Bandwidth Expansion with get_bandwidth_expansion()

Let’s see where we are with our CNR, assuming DSES numbers.

For example, CNR in 1 Hz bandwidth = -8.65 dB CNR in 5 MHz bandwidth = -8.65 dB - 10 * log(5×10^6) = -8.65 dB - 67 dB = -75.65 dB Processing gain for 5 MHz bandwidth = +68.26 dB We have a shortfall of ~7.35 dB.

We do a bandwidth expansion for our chip rate bandwidth with get_bandwidth_expansion() and compare to our previously calculated processing gain.

Our processing gain, calculated with the limitation of the Doppler rate shift on the sequence length, doesn't quite get us there. What can we do? There's three things we can look at. First, we can do multiple non-coherent integrations (e.g., 10 non-coherent integrations would add ~5 dB). We can try a slightly longer coherent integration time if Doppler allows. We designed for the worst case, at inferior conjunction. This is the best place to try EVE, so we are probably stuck with this limitation. Or, we can use some error correction coding for actual data transmission.

Using JPL's work as a model, let's do multiple non-coherent integrations. This is under our control, doesn't add as much complexity as adding error correction, and doesn't run the risk of blowing past a physical Doppler limit when we have the lowest path loss.

We use a dual-stage processing strategy. First, we perform coherent integration within the Doppler-limited window of (for example) 1.34 seconds across a 5 MHz bandwidth. Then, we combine multiple such integrations non-coherently to achieve positive SNR. This part is flexible, and can be used as Venus approaches or recedes. We just combine more sequences non-coherently.

For example, starting with a 1 Hz CNR of -8.65 dB, the full-bandwidth 5 MHz CNR is -75.65 dB. Coherent processing provides 68.3 dB gain, yielding -7.35 dB.

So, we need to non-coherently integrate some number of segments at inferior conjunction. Coherent processing maximizes gains within a Doppler rate of change limitation, and non-coherent integration extends processing beyond those constraints in a flexible way, depending on what our shortfall really is. Our calculations have a parameter for margin. The default is 0 dB. This parameter changes the threshold of detecion.

Calculate the number of non-coherent integrations needed with get_number_noncoherent_sequences_required

At the receiver, we do a matched filtering using the known Zadoff-Chu sequence. The matched filter correlates the received signal with the expected sequence. The correlation output is examined for peaks that exceed a detection threshold. The time offset of the peak indicates the precise round-trip delay. The peak amplitude provides information about signal strength.

Non-coherent combination works with the power (magnitude squared) of the correlation outputs rather than the complex values. For each of the non-coherently integrated

sequence correlations, we calculate magnitude squared. This destroys the phase information but preserves signal energy, which is all we want at this point. Align the correlation outputs based on expected delay progression. Sum up all the magnitude squared results. This summation increases SNR by approximately 5 * log(L) where L is the number of these sequences.

The distinction between coherent versus non-coherent combining is important when calculating processing gain from multiple observations.

Coherent combining at 10 * log10(N) is used when you can preserve both amplitude and phase information. Signals add linearly before detection. This is a "voltage addition". Power grows as N^2 where N is the number of samples, which results in 10 * log10(N) dB gain. Non-coherent combining is 5 * log10(N). This is used when only signal power or amplitude can be preserved. In other words, when phase information is lost. Signals add after detection. This is a "power addition". Power grows as N (and not N^2) where N is the number of samples. This results in 5 * log10(N) dB gain.

Apply detection threshold to the combined result. I'm not entirely sure how to set the threshold, but correlates usually have a false-alarm detection rate and assumptions about the noise. This is the reason for having a margin parameter in the code below. The detected peaks in the non-coherent sum indicate successful reception. A detectable peak is what we are looking for.

We can extend out the non-coherent combination until we close the link, but does this have a limit? Non-coherent is (allegedly) more resilient to phase and Doppler than coherent integration. Can we assume it's immune, or do we need a factor that scales with number of non-coherent integrations?

Each coherent segment can be processed independently, allowing for parallel implementation. This reduces the burden on the hardware compared to a huge sequence. We think that the combination of coherent processing (optimized to Doppler rate of change constraint) and non-coherent integration (for extending beyond the shortfall we still have) provides a practical approach to close the link.

Visualization

Below is a visualization of an example of this proposal. The particular numbers in the example may not match a proposed waveform. The image is to illustrate the concepts.

You can see the bandwidth expansion, followed by a processing gain for the coherent integration. This is followed by gain from non-coherent integrations of the coherent integrations.

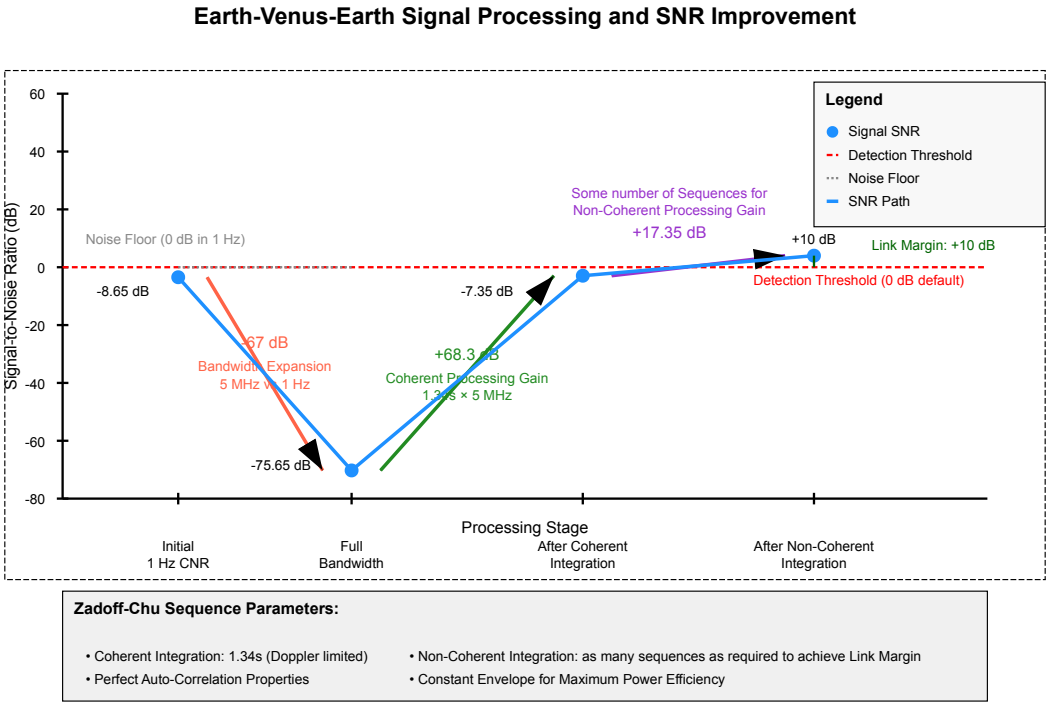
In [226…
```
#Try to embed our SVG so we don't have to have it as a sidecar
```

```
from IPython.display import HTML
import base64

def embed_svg_content(svg_file, width=None):
    """Fully embed SVG content in the notebook (no external file needed wher
    try:
        # Read the SVG file content
        with open(svg_file, 'r') as f:
            svg_content = f.read()

        # Convert to data URL for complete embedding
        svg_bytes = svg_content.encode('utf-8')
        b64 = base64.b64encode(svg_bytes).decode('utf-8')

        if width:
            html = f'<img src="{b64}" width="{widt
        else:
            html = f'<img src="{b64}">'

        return HTML(html)
    except Exception as e:
        return HTML(f"<p>Error embedding SVG: {str(e)}</p>")
```

In [227…

```
# This will fully embed the SVG - no separate file needed when sharing
embed_svg_content('eve-signal-processing-diagram.svg', width=1200)
```

Out[227…



Doppler Spread

However, this is not the entire story. We have not yet calculated and included the Doppler Spread penalty. Doppler spread has a large effect on coherent integration, and

reduces our processing gain. What is the effect of the Doppler Spread from Venus on Zadoff-Chu signals?

In [228…

```
class ZadoffChuSequenceEvaluator:
    """
    A class to evaluate the suitability of Zadoff-Chu sequences based on lir
    """
    def __init__(self, params: SiteLinkParameters, chip_rate_hz: float, cnr_
        self.params = params
        self.chip_rate_hz = chip_rate_hz
        self.cnr_db_1hz = cnr_db_1hz
        self.bandwidth_factor = 1 # in case we need to move from Time-Rate (

    def get_bandwidth_expansion(self) -> float:
        """
        For our Zadoff-Chu implementation, we're assuming rectangular pulse
        so the bandwidth (W) equals the chip rate (R). Therefore, the time-b
        product (TW) equals the number of chips (TR).

        Processing gain = 10*log10(TW) = 10*log10(TR) = 10*log10(number of c
        Bandwidth expansion = 1 Hz CNR - 10*log10(R) = 1 Hz CNR - 10*log10(c

        If we want to account for pulse shaping, then the number of chips ne
        be multiplied by a factor such as 1.2 in both get_processing_gain ar
        get_bandwidth_expansion() methods.

        Change self.bandwidth_factor from 1 to the desired factor in order t
        include the TR to TW transition, due to effects from pulse shaping.
        """
        expansion = self.cnr_db_1hz - 10 * np.log10(self.bandwidth_factor *
        #print(f"Full bandwidth SNR is {expansion:.2f} dB")
        return expansion

    def get_integration_time(self) -> float:
        """
        Calculate the maximum allowable coherent integration time based on t
        1. Doppler rate of change (phase stability constraint)
        2. Doppler spread (frequency coherence constraint)

        Returns the more conservative (smaller) of the two limits.
        """
        # Calculate limit based on Doppler rate of change
        DopplerCalculator.set_observer_location(self.params.latitude, self.p
                                                self.params.elevation, locati
        my_worst_case = DopplerCalculator.calculate_location_worst_case_dopp

        # Calculate max integration time based on max rate of change of Dopp
        integration_time_doppler_rate = np.sqrt((1/4)/(np.abs(my_worst_case[
        print(f"Integration time limit from Doppler rate: {integration_time_

        # Calculate limit based on Doppler spread (coherence time)
        # Coherence time is the reciprocal of the Doppler spread
        integration_time_doppler_spread = 1/optimized_results.model_spread
        print(f"Integration time limit from Doppler spread: {integration_tim
```

```
        # Use the more restrictive (smaller) of the two limits
        integration_time = min(integration_time_doppler_rate, integration_ti

        if integration_time == integration_time_doppler_rate:
            print(f"Coherent integration limited by Doppler rate of change")
        else:
            print(f"Coherent integration limited by Doppler spread")

        print(f"Using maximum integration time of {integration_time:.4f} sec
        return float(integration_time)

    def get_number_chips(self) -> float:
        num_chips = self.get_integration_time() * self.bandwidth_factor * se
        #print(f"Number of chips is {num_chips:.2f}")
        return num_chips


    def calculate_doppler_spread_penalty(self) -> float:
        """
        Calculate any residual penalty from Doppler spread on coherent integ

        Since we're now limiting our integration time based on both Doppler
        and Doppler spread, this function exists primarily to account for ar
        additional losses not captured by the integration time limitation.

        Returns:
        - Residual penalty in dB (typically 0 or very small)
        """
        # Get the integration time that's already limited by both criteria
        integration_time = self.get_integration_time()

        # Calculate coherence time (inverse of Doppler spread)
        coherence_time = 1/optimized_results.model_spread
        print(f"Coherence time (1/Doppler spread): {coherence_time:.4f} secc

        # Calculate ratio of integration time to coherence time
        # This should be <= 1.0 if our get_integration_time() is working cor
        time_ratio = integration_time / coherence_time
        print(f"Integration time / coherence time ratio: {time_ratio:.4f}")

        # This should generally not apply a penalty if integration_time sele
        # We keep it as a safety check and for potential implementation vari
        if time_ratio > 1.0:
            # This case should not happen if our integration time selection
            # But we keep it as a safeguard
            print(f"Warning: Integration time ({integration_time:.4f}s) exce
            penalty = 10 * np.log10(time_ratio)
            print(f"Applying residual Doppler spread penalty: {penalty:.2f}
            return min(penalty, 20.0)  # Cap at 20dB
        else:
            print(f"Integration time ({integration_time:.4f}s) is within coh
            print("No residual Doppler spread penalty needed")
            return 0.0


    def get_processing_gain(self) -> float:
```

```
        """
        For our Zadoff-Chu implementation, we're assuming rectangular pulse
        so the bandwidth (W) equals the chip rate (R). Therefore, the time-b
        product (TW) equals the number of chips (TR).

        Processing gain = 10*log10(TW) = 10*log10(TR) = 10*log10(number of c
        Bandwidth expansion = 1 Hz CNR - 10*log10(R) = 1 Hz CNR - 10*log10(c

        If we want to account for pulse shaping, then the number of chips ne
        be multiplied by a factor such as 1.2 in both get_processing_gain ar
        get_bandwidth_expansion() methods.

        Change self.bandwidth_factor from 1 to the desired factor in order t
        include the TR to TW transition, due to effects from pulse shaping.
        """
        processing_gain = 10 * np.log10(self.get_number_chips())
        #print(f"Processing gain is {processing_gain:.2f} dB for {num_chips:
        return processing_gain


    def get_processing_gain_with_doppler_spread(self) -> float:
        """
        Get the effective processing gain after accounting for Doppler sprea
        """
        # Calculate basic processing gain
        basic_gain = self.get_processing_gain()

        # Calculate Doppler spread penalty
        doppler_penalty = self.calculate_doppler_spread_penalty()

        # Apply penalty to processing gain
        effective_gain = basic_gain - doppler_penalty

        print(f"Basic processing gain: {basic_gain:.2f} dB")
        print(f"Doppler spread penalty: {doppler_penalty:.2f} dB")
        print(f"Basic processing gain - Doppler Penalty: {effective_gain:.2f

        return effective_gain


    def get_number_noncoherent_sequences_required(self) -> int:
        # Use processing gain with Doppler penalty instead of basic processi
        gain = self.get_processing_gain_with_doppler_spread() + self.get_bar
        print(f"The difference between Zadoff-Chu bandwidth expansion and th
        margin = 10.0      # in practice, a margin is usually assumed - we c
        print(f"We need this number to be at least {margin} dB.")
        if gain > margin: # we closed the link
            print("We closed the link with coherent integration. No further
            num_noncoh_seq = 0
            return num_noncoh_seq
        else:
            # we didn't close the link.
            # we need to non-coherently combine some number of sequences.
            # how many sequences? This loop erodes our shortfall to find out
            count = 1
            gain = gain - margin # move our gain to where it's purely negati
```

```
            #print(f"gain is shifted by margin to be {gain}")
            gain = np.abs(gain) # turn it into a positive number to make it
            #print(f"gain is made positive and is {gain}")
            while gain > (5 * np.log10(count)):
                #print(f"gain is {gain:.2f} dB")
                #print(f"number of non-coherent integrations so far:{count}"
                count = count + 1
            print("We didn't close the link.")
            print(f"We need to non-coherently integrate {count} sequences to
            print(f"If we do that, we should now have at least a {margin} dB
            seq_seconds = count * ZadoffChuSequenceEvaluator.get_integration
            seq_minutes = seq_seconds/60
            seq_hours = seq_minutes/60
            seq_days = seq_hours/24
            print(f"This is {seq_seconds:.1f} seconds of sequences.")
            print(f"This is {seq_minutes:.1f} minutes of sequences.")
            print(f"This is {seq_hours:.1f} hours of sequences.")
            print(f"This is {seq_days:.1f} days of sequences.")



            return count


# It's 10pm, do you know where your params are?
params = SiteLinkParameters()

# Get the CNR in 1 Hz result from link budget calculator and set the chip ra
min_cnr_db_1hz = min_results['cnr_db_1hz']
print(f"from the Link Budget calculator, our min cnr_db_1hz is {min_cnr_db_1
chip_rate = 1e3
print(f"Our chip rate is {chip_rate} Hz")

# Create a calculator for Doppler Spread
DopplerSpreadCalculator = EVEDopplerSpread(params)
#doppler_spread = DopplerSpreadCalculator.venus_doppler_spread()
doppler_spread = optimized_results.model_spread

# Create Zadoff-Chu Sequence Evaluator instance
ZadoffChuSequenceEvaluator = ZadoffChuSequenceEvaluator(params, chip_rate_hz

# test get_bandwidth_expansion
expansion = ZadoffChuSequenceEvaluator.get_bandwidth_expansion()
print(f"Full bandwidth SNR for our Zadoff-Chu sequence goes from the 1 Hz ba



# test get_integration_time at maximum Doppler rate of change
integration_time = ZadoffChuSequenceEvaluator.get_integration_time()
print(f"Our returned integration_time is {integration_time:.2f} seconds")

# test get_number_chips
num_chips = ZadoffChuSequenceEvaluator.get_number_chips()
print(f"The number of chips in our integration time is {num_chips:.2f}")

# test get_processing_gain
processing_gain = ZadoffChuSequenceEvaluator.get_processing_gain()
```

```
print(f"Basic processing gain for our coherent integration time is {processi

# test get_processing_gain_with_doppler_spread
processing_gain = ZadoffChuSequenceEvaluator.get_processing_gain_with_dopple
print(f"Processing gain with Doppler penalty for our coherent integration ti

# test the number of non-coherent integrations needed
num_noncoh_seq = ZadoffChuSequenceEvaluator.get_number_noncoherent_sequences
print(f"Recommended number of coherently integrated sequences recommended fo
```

```
from the Link Budget calculator, our min cnr_db_1hz is -8.890 dB
Our chip rate is 1000.0 Hz
Full bandwidth SNR for our Zadoff-Chu sequence goes from the 1 Hz bandwidth
value of -8.89 dB to the chip rate bandwidth expansion of -38.89 dB
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Our returned integration_time is 0.26 seconds
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
The number of chips in our integration time is 260.09
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Basic processing gain for our coherent integration time is 24.15 dB for 260.
09 chips
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Coherence time (1/Doppler spread): 0.2601 seconds
Integration time / coherence time ratio: 1.0000
Integration time (0.2601s) is within coherence time (0.2601s)
No residual Doppler spread penalty needed
Basic processing gain: 24.15 dB
Doppler spread penalty: 0.00 dB
Basic processing gain - Doppler Penalty: 24.15 dB
Processing gain with Doppler penalty for our coherent integration time is 2
4.15 dB for 260.09 chips
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Coherence time (1/Doppler spread): 0.2601 seconds
Integration time / coherence time ratio: 1.0000
Integration time (0.2601s) is within coherence time (0.2601s)
No residual Doppler spread penalty needed
Basic processing gain: 24.15 dB
Doppler spread penalty: 0.00 dB
Basic processing gain - Doppler Penalty: 24.15 dB
The difference between Zadoff-Chu bandwidth expansion and the effective proc
essing gain is -14.74 dB
We need this number to be at least 10.0 dB.
We didn't close the link.
```

```
We need to non-coherently integrate 88680 sequences to produce 24.74 dB more
gain.
If we do that, we should now have at least a 10.0 dB margin.
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
This is 23064.8 seconds of sequences.
This is 384.4 minutes of sequences.
This is 6.4 hours of sequences.
This is 0.3 days of sequences.
Recommended number of coherently integrated sequences recommended for furthe
r non-coherent integration is: 88680, for a gain of 24.74 dB
```

Temporal Spread Worksheet

Gary K6MG writes "One other item I didn't see addressed is the temporal spreading of the energy. Given a 12000Km diameter for venus the returned energy gets spread over a 12e6/3e8 = 40ms window. This dual temporal/spectral spreading presents a challenge to coherently summing the returned energy to detect a weak signal. OTOH it provides astronomers the mechanism to do planetary surface mapping by pixellating the surface into delay-doppler regions.

With a chip period of 2e-7 a total of 40e-3/2e-7=2e5 PN sequences need to be summed to collect all of the returned energy."

In [229…

```
print(f"diameter of Venus is: {2*calculator.venus_radius_km * 1000} m")
print(f"chip_rate_hz from Zadoff-Chu calculator is {ZadoffChuSequenceEvaluat
print(f"speed of light is {calculator.params.c} meters per second")
temporal_spread = (2 * calculator.venus_radius_km * 1000)/calculator.params.
print(f"Temporal spread is {temporal_spread} seconds")
chip_period = 1/ZadoffChuSequenceEvaluator.chip_rate_hz
print(f"With a chip period of {chip_period} seconds a total of {(temporal_sp
num_chips = ZadoffChuSequenceEvaluator.get_number_chips()
print(f"Number of chips in our Zadoff-Chu sequence is {num_chips:.2f}")
```

```
diameter of Venus is: 12103600.0 m
chip_rate_hz from Zadoff-Chu calculator is 1000.0 Hz
speed of light is 299792458 meters per second
Temporal spread is 0.04037326382640353 seconds
With a chip period of 0.001 seconds a total of 40.37 chips need to be summed
to collect all of the returned energy.
Integration time limit from Doppler rate: 1.0189 seconds
Integration time limit from Doppler spread: 0.2601 seconds
Coherent integration limited by Doppler spread
Using maximum integration time of 0.2601 seconds
Number of chips in our Zadoff-Chu sequence is 260.09
```

Planetary Radar Detector

So far, we've generally assumed that we are attempting communications. Most of the signals we are interested in are relatively complex digital communications protocols. We're trying to detect, demodulate, and decode so that we can retrieve the information encoded within them.

What if we just want to prove we can bounce a very simple signal off Venus?

In this case, the example Python notebook and SigMF files from CAMRAS from the March 2025 EVE event are the way to go.

Please visit https://data.camras.nl/venus/ and read on for how to use the archive.

How to Process an Earth-Venus-Earth Radar Return Signal Using CAMRAS Data

With receiver data successfully collected by Dwingeloo and Stockert amateur radio astronomy sites during the Earth-Venus-Earth communications event on 22 March 2025, attention moved to processing the data to see if the transmitted carrier wave signal could be detected.

Four transmissions of a carrier wave from the Dwingeloo radio astronomy site were made. Each transmission was 278 seconds long. This time span was selected because it equaled the expected round-trip time from Earth to Venus and back. This would allow the transmitter to transmit for the round trip time, shut down, and then immediately begin receiving what should be the leading edge of the reflected radio signal. Both Dwingeloo and Stockert made time-coordinated in-phase and quadrature (IQ) recordings of the receive spectrum.

IQ samples are complex numbers that are created by sampling a received waveform. IQ samples fully specify any waveform, as long as they are taken at a rate faster than twice the bandwidth of the recorded signal. These IQ time series recordings are then mathematically evaluated in order to reveal the reflected carrier wave spike in the frequency domain.

Raw data (at 1 MSps), decimated data (at 5 kSps), and the example data processing Python Notebook are available from https://data.camras.nl/venus/

The data files are in SigMF format. This format is widely used for IQ signal data recordings. A data file, containing IQ samples, is paired with a metadata file, which has a variety of metadata describing the conditions and configurations of recorded data. Learn more about SigMF at https://sigmf.org

In addition to the SigMF files and a Python Notebook that performs the data analysis, there are also two comma separated value (CSV) files containing the expected frequency offsets and Doppler rates for RX timestamps. There is one for Stockert and

one for Dwingeloo. These values are needed by the script in order to correct the received samples frequency shifts due to Doppler.

Once the notebook eve-cw-detect-example.ipynb is downloaded from the site, then all of the 5 kSps files, dwingeloo_venus_doppler.csv and stockert_venus_doppler.csv should also be downloaded.

While the 1 MSps files are higher resolution, the 5 kSps files are more than enough resolution to carry out the math. It takes much less time to process these smaller lower resolution files than the larger higher resolution files, and the results will be exactly the same.

Note: There is a bug in the python package sigmf version 1.2.8 that prevents these files from loading. Please install 1.2.6 or a version greater than 1.2.8. Or, modify the script to set up a SigMFFile instace manually, to evade the bug.

Run the script, and if all goes well, the visualization will show a spike for the carrier wave. Success!

What is this script doing?

Here is a summary of the most important calculations.

The notebook loads the recorded IQ data SigMF files and the Doppler information from CSV files. It then compensates for both the Doppler shift and Doppler rate of change using phase rotation.

Compensate for Doppler and Doppler rate

dr = np.repeat(doppler_rate, len(data) / len(doppler_rate)) t_s = np.arange(len(data)) / fs fdrift = -dr # Doppler rate f0_Hz = -expected_freq_offset # Doppler phi_Hz = (fdrift * t_s**2) / 2 + (f0_Hz * t_s) # Instantaneous phase. phi_rad = 2 * np.pi * phi_Hz # Convert to radians. data = data * np.exp(1j * phi_rad)

This means our signal will be centered at 0 Hz offset for the rest of the processing.

Coherent Processing

Next, we do some coherent processing. Coherent processing means that we preserve the phase information of the signal. Non-coherent processing just looks at the amplitude. Fast Fourier Transforms (FFT) preserve magnitude and phase.

spectrum = np.fft.fftshift(np.fft.fft(data.reshape(-1, int(sample_rate))))

The data is reshaped to have segments of length sample_rate (which is 5000 samples per second), meaning each FFT operation is performed on exactly 1 second of data. This is the coherent integration time used in the experiment.

This is less than the maximum coherency time available due to Doppler rate of change limitations (~1.3 seconds), but it is larger than another maximum limit due to Doppler Spread from Venus' rotation (we calculated a much lower 0.03 seconds based on 31 Hz). The reasoning for being able to use 1 second coherent integration time (based on a Doppler Spread of 1Hz) is that the updated Doppler Spread is not as bad as we originally calculated

Non-Coherent Processing

After obtaining the FFT results, the notebook performs non-coherent integration by averaging the magnitudes of the FFT outputs.

This means it discards the phase information and combines power/energy over the entire recording run. spec_sum = np.mean(np.abs(spectrum), axis=0)

The axis=0 parameter means it's averaging across multiple 1-second segments, effectively performing non-coherent integration over the entire 278 second long recording duration. This allows us to integrate over much longer periods than would be possible with purely coherent integration.

Get Involved

Dwingeloo's Thomas Telkamp invites the community to expand on and improve the Python notebook, saying in late March 2025, "A few remarks: 1) use the 5ksps files, 2) there is an example notebook now on how to process the data, and 3) you're not supposed to duplicate it, but to improve it!"

Also, please read the excellent analysis of the event from Dr. Daniel Estevez at https://destevez.net/2025/04/analysis-of-the-camras-venus-radar-experiment/ This analysis was used in our Doppler Spread section above.

And, additional details from Cees Bassa can be found at https://www.camras.nl/en/blog/2025/first-venus-bounce-with-the-dwingeloo-telescope/

Appendix A

ORI Earth-Venus-Earth Communications Development Roadmap

Note: the checkboxes for each task listed below *may not* automatically render if you are viewing this document on GitHub. If you want to see completed vs. uncompleted tasks, then you will need to see the PDF rendering of this document, or render it to a viewer yourself.

Phase 1: Foundation and Research (3-4 months)

Documentation and Requirements

- [x] Document full link budget analysis (this notebook)
- [x] Compile SDR Capabilities (zc706/adrv9009 or any partner SDR system)
- [x] Create detailed Doppler analysis (in this notebook)
- [x] Define custom mode specifications (CNR in 1Hz now known)
- [ ] Write FPGA requirements document

zc706/adrv9009 FPGA Development Board Capabilities

ADRV9009 capabilities:

-Frequency range: 75 MHz to 6 GHz (covers our 1296 MHz) -Observation bandwidth: up to 200 MHz (plenty for our ~600 kHz Doppler range) -12-bit ADC -Low noise performance -Direct sampling capability

The Zynq 7 FPGA on the zc706 provides:

-DSP slices: 900 -Block RAM: 555 × 36 Kb -LUTs: 218,600 -Flip-flops: 437,200

zcu102/adrv9009 FPGA Development Board Capabilities

ADRV9009 capabilities:

-Frequency range: 75 MHz to 6 GHz (covers our 1296 MHz) -Observation bandwidth: up to 200 MHz (plenty for our ~600 kHz Doppler range) -12-bit ADC -Low noise performance -Direct sampling capability

UltraScale+ XCZU9EG on the zcu102 provides:

DSP Slices: 2,520 (vs 900 on zc706) Block RAM: 1,824 × 36Kb (vs 555 on zc706) UltraRAM: 960 Kb LUTs: 548,160 (vs 218,600 on zc706) Flip-flops: 1,096,320 (vs 437,200 on zc706)

-More parallel processing channels possible -Better timing closure likely with UltraScale+ architecture -UltraRAM provides additional buffer space for sample processing -Higher achievable clock rates -More room for future expansion/features -Could implement more parallel demodulators for Doppler tracking

USRP X310 (Dwingeloo has same SDR)

Custom Mode Definition

In the case that no existing communications mode closes the link (see Evaluation of Modes cell below) we could design a custom "EVE-Mode" with:

- Zadoff-Chu sequences
- QPSK modulation as a baseline (good balance of spectral efficiency and robustness)
- ~5 kHz bandwidth (more than wide enough for short-term Doppler)
- Strong FEC (LDPC, Polar)
- Synchronization sequence designed for high Doppler rates
- Multiple parallel decoding paths

Custom Mode SDR Requirements

- Need high sample rate (>1.2 MSPS)
- Need Excellent phase noise performance
- 16-bit ADC minimum
- Very stable frequency reference

Custom Mode Specficiation

- [x] Doppler-delay channel protocol modeling
- [ ] Doppler-delay channel protocol article(s) (in progress)
- [ ] Doppler-delay channel protocol specification

Initial GNU Radio Development

- [ ] Basic flow graph implementation
- [ ] Simple Doppler tracking prototype
- [ ] Test with recorded or simulated signals/simulations
- [ ] Document processing chain

Community Engagement

- [x] Present project on ORI channels
- [x] Recruit additional developers (mailing list, Slack)
- [x] Set up project definition and documentation (this document)

Phase 2: Core Development (6-8 months)

GNU Radio Implementation

- [ ] Complete mode implementation
- [ ] Doppler tracking refinement
- [ ] orbital prediction integration
- [ ] Performance monitoring tools
- [ ] User interface development

FPGA Development

- [ ] DDC implementation?
- [ ] Initial Doppler tracking
- [ ] Memory interface design
- [ ] Basic demodulator implementation
- [ ] Testing framework development
- [ ] Does our AI/ML team have a role?

Integration Planning

- [ ] SDR selection and acquisition: zc706/adrv9009, zcu102/adrv9009 or 9002, new hardware
- [ ] Interface definition for hardware
- [x] Test equipment requirements (Remote Labs confirmed)
- [ ] Performance measurement plans

Phase 3: Integration and Testing (4-6 months)

Hardware Integration

- [ ] SDR integration (do the work)
- [ ] FPGA bitstream testing (do the work)
- [ ] Timing verification (do the work)
- [ ] Performance measurements (yes, do the work)

Software Integration

- [ ] GNU Radio to FPGA interface?
- [ ] Control software development
- [ ] Monitoring tools - [ ] probably not required due to brevity of opportunities
- [ ] User interface refinement (standing orders)

Testing Framework

- [ ] Automated test development (probably not needed but need to talk about)
- [ ] Performance verification (did it work or not)
- [ ] Doppler simulation testing (did our model turn out to be accurate)
- [ ] Link budget verification (did we close the link with expected margins)

Phase 4: Optimization and Documentation (3-4 months)

Performance Optimization (Vivado)

- [ ] FPGA resource optimization and utilization reports
- [ ] Processing chain refinement
- [x] Doppler tracking improvements?
- [ ] Memory usage optimization

Documentation

- [ ] User manual creation (HTML5 interface?)
- [ ] Installation guides
- [ ] Development documentation, articles, reports
- [ ] Performance reports, papers

Community Resources

- [ ] Example configurations
- [x] Tutorial development
- [x] Test data publication
- [ ] Contribution guidelines

Phase 5: Deployment and Validation (3-4 months)

Field Testing (EME stations as proxies until October 2026)

- [ ] Initial station setup (may be very narrow cases if remote access not available)
- [ ] Performance validation
- [ ] Doppler tracking testing
- [ ] System stability testing

Final Documentation

- [x] Test results publication
- [ ] Configuration guides
- [ ] Troubleshooting guides
- [ ] Performance reports

Community Support

- [ ] Training materials
- [x] Support documentation
- [x] Schedule of next opportunity for communications attempts October 2026
- [x] Future development plans

Ongoing Activities

Community Engagement

- [x] Regular progress updates (Inner Circle, email, published video recordings)
- [x] Technical presentations (IEEE vTools events)
- [ ] Conference participation
- [x] Developer meetings

Development Support

- [x] Code review process (weekly meetups)
- [x] Bug tracking (Github Issues Tracker)
- [x] Feature requests (Slack, email, meetups)
- [ ] Performance monitoring

Documentation Maintenance

- [x] Regular documentation updates
- [x] Performance reports
- [x] Configuration Artifacts Recorded
- [x] Best practices documentation

Key Milestones

0. Jupyter Lab Notebook passes review
1. Initial GNU Radio prototype functional
2. First FPGA implementation complete
3. Basic Doppler tracking working
4. Full system integration achieved
5. First successful field tests
6. Release candidate testing complete
7. Production release with documentation

Success Criteria

- [x] Successful Doppler tracking at specified rates
- [ ] Reliable demodulation under varying conditions
- [ ] Meeting specified link budget parameters
- [x] Complete, maintainable documentation
- [x] Active community engagement
- [x] Reproducible build process
- [ ] Comprehensive test coverage

Risk Management

Technical Risks Managed?

- [x] SDR performance limitations
- [x] FPGA resource constraints
- [x] Doppler tracking challenges
- [ ] Integration complexities

Mitigation Strategies

- [x] Early prototyping
- [ ] Regular testing
- [ ] Performance monitoring
- [ ] Community feedback
- [x] Incremental development
- [x] Regular reviews

Resource Requirements

Hardware

- [x] Development SDRs
- [x] FPGA development boards
- [x] Test equipment
- [x] Antenna systems

Software

- [x] FPGA development tools
- [x] GNU Radio environment
- [x] Testing frameworks
- [x] Documentation tools

Personnel

- [x] FPGA developers
- [ ] GNU Radio developers
- [x] Documentation writers
- [x] Test engineers
- [x] Project coordinators

Review Points

- [x] Weekly progress reviews
- [x] Monthly milestone assessments (Inner Circle Newsletter)
- [x] Community feedback sessions (vTools crosslisting, email list)
- [x] Performance validation checks (Doppler spread updated from March 2025 event)
- [x] Documentation reviews (ongoing)