Modeling Transit Timing Variations with the Exoplanet Python Library

Transit Timing Variations (TTVs) are an essential tool in exoplanetary science, allowing astronomers to detect the gravitational influence of additional planets in a system and characterize orbital dynamics. The following report details how to implement TTV modeling using the exoplanet Python library, with specific focus on implementing sinusoidal period variations.

Understanding Transit Timing Variations

Transit Timing Variations occur when a transiting planet's orbital period appears to change slightly over time, causing transits to occur earlier or later than would be expected from a strictly periodic orbit. These variations often manifest as periodic signals in the observed-minus-calculated (O-C) transit times and can be caused by gravitational perturbations from other planets, orbital decay, or apsidal precession.

Recent studies, such as the analysis of HAT-P-12b's TTVs, have demonstrated the effectiveness of sinusoidal models in characterizing these variations. In that study, researchers analyzed 46 light curves over 14 years using various analytical models, with the sinusoidal model providing the lowest reduced chi-squared value ($\chi^2_r = 3.2$)[1].

The Exoplanet Library's TTV Implementation

The exoplanet library provides a specialized class called `TTVOrbit` for modeling transit timing variations. This class extends the standard `KeplerianOrbit` class to account for non-uniform transit timing[2].

Core Components of TTVOrbit

The `TTVOrbit` class works by shifting the time axis to account for TTVs before solving for a standard Keplerian orbit. It identifies which times correspond to which transits by finding the closest labeled transit to each timestamp and adjusting accordingly[2].

The implementation includes a utility function `compute_expected_transit_times` that calculates the expected transit times within a dataset given:

- The minimum time of the dataset
- The maximum time of the dataset
- The orbital period
- The reference transit time[2]

Implementing a Sinusoidal TTV Model

To model sinusoidal transit timing variations, we need to combine the base functionality of the exoplanet library with custom code to implement the periodic variation. Here's how to approach this:

Step 1: Set Up Basic Parameters

First, initialize the basic orbital parameters:

```
import numpy as np
import pymc3 as pm
import exoplanet as xo
from exoplanet.orbits.ttv import TTVOrbit, compute_expected_transit_times

# Define basic orbital parameters
period = 3.213059  # Orbital period in days
t0 = 2454216.77325  # Reference transit time
```

Step 2: Define the Sinusoidal TTV Model

Following the approach used in the HAT-P-12b study, we can implement a sinusoidal TTV model as:

```
def sinusoidal_ttv_model(epochs, amplitude, frequency, phase):
    """
    Create sinusoidal TTV model similar to that used in HAT-P-12b analysis

    Parameters:
    -----------
    epochs : array-like
        The epoch numbers
    amplitude : float
        Amplitude of the TTV in days
    frequency : float
        Frequency in cycles/epoch
    phase : float
        Phase in radians

    Returns:
    --------
    ttv_values : array-like
        The TTV values in days for each epoch
    """
    return amplitude * np.sin(2 * np.pi * frequency * epochs + phase)
```

Step 3: Implement the PyMC3 Model

Now we can integrate this with exoplanet's PyMC3-based workflow:

```
with pm.Model() as model:
    # Stellar parameters
    stellar_mass = pm.Normal("stellar_mass", mu=1.0, sd=0.1)

    # Orbital parameters
    period_mean = pm.Normal("period_mean", mu=period, sd=0.0001)
    t0_mean = pm.Normal("t0_mean", mu=t0, sd=0.001)

    # TTV parameters
    amplitude = pm.Uniform("amplitude", lower=0, upper=0.01)  # in days
    frequency = pm.Uniform("frequency", lower=0.001, upper=0.1)  # cycles/epoch
    phase = pm.Uniform("phase", lower=0, upper=2*np.pi)

    # Compute expected transit times
    min_time = observed_times.min() - period
    max_time = observed_times.max() + period
    transit_times = compute_expected_transit_times(min_time, max_time, period_mean, t0_me

    # Compute expected epochs
    epochs = np.round((transit_times - t0_mean) / period_mean).astype(int)

    # Apply sinusoidal TTV model
    ttv_offsets = sinusoidal_ttv_model(epochs, amplitude, frequency, phase)

    # Create TTVOrbit with computed TTVs
    orbit = TTVOrbit(
        period=period_mean,
        t0=t0_mean,
        transit_times=transit_times + ttv_offsets,
        m_star=stellar_mass,
    )
```

Step 4: Fitting the Model to Data

To fit this model to observed transit times, we need to define a likelihood function:

```
with model:
    # Define the likelihood based on observed transit times
    # Assume observed_times and observed_uncertainties are arrays of
    # observed transit times and their uncertainties
    pm.Normal("obs",
              mu=orbit.get_transit_times(epochs),
              sd=observed_uncertainties,
              observed=observed_times)

    # Sample from the posterior
    trace = pm.sample(
        tune=1000,
        draws=1000,
        chains=4,
        cores=4,
        return_inferencedata=True
    )
```

Analysis of Sinusoidal TTVs

The approach above closely mirrors the method used in recent scientific analyses like the HAT-P-12b study, where researchers found a significant periodic signal at 0.00415 cycles/day (with a False Alarm Probability of 5.11×10^{-6}%), suggesting the influence of an additional planetary companion[1].

When implementing this for your own dataset, consider the following parameters identified in the HAT-P-12b study:

- The amplitude of the best-fit TTV (A_TTV) was found to be 2.6 minutes
- The lowest reduced chi-squared value (χ^2_r) was 3.2
- The best frequency was 0.00415 cycles/day[1]

Validating Your Model

To validate your sinusoidal TTV model, you can use:

1. Generalized Lomb-Scargle Periodogram (GLS) to identify significant periodic signals in your observed TTVs
2. Comparison of reduced chi-squared values between different models (linear, orbital decay, apsidal precession, and sinusoidal)
3. Calculation of False Alarm Probability (FAP) for identified frequencies[1]

Limitations and Considerations

When implementing sinusoidal TTV models with exoplanet, be aware of these limitations:

1. The TTVOrbit class requires that the time between neighboring transits is less than twice the orbital period[2]
2. Multiple peaks in the periodogram might have similar significance, requiring careful analysis to select the correct frequency
3. Computational complexity increases with longer observational baselines and more transits

Conclusion

The exoplanet Python library provides a flexible framework for modeling transit timing variations, including sinusoidal period changes. By extending the TTVOrbit class with custom sinusoidal functions, you can effectively model periodic TTVs and potentially identify the gravitational influence of additional planets in your system.

For complex systems, consider implementing multiple TTV models (linear, orbital decay, apsidal precession, and sinusoidal) and comparing their fit quality, as was done in the HAT-P-12b analysis. The sinusoidal model is particularly valuable when searching for the influence of additional planets, especially those in mean-motion resonances with the transiting planet.

⁂

1. https://arxiv.org/html/2503.10441v1
2. https://github.com/dfm/exoplanet/blob/master/src/exoplanet/orbits/ttv.py