perplexity

Complete this function.

Analyze lc: calculate time and depth of transit. Calculate T1, T2, T3, T4, TT and depth.
Fit a model of a transit to just this one lightcurve.
Do not use other data.
Therefore, folding lightcurves does not make sense. Do not do it.
Using a periodogram does not make sense. Do not do it.
Finding a max power does not make sendse. Do not do it.

from lightkurve import TessLightCurve
def analyze_lightcurve():
sectors = ["28"]
# sectors = ["27", "28", "29", "30", "31", "32", "33", "34", "35", "36", "37", "38", "61", "62", "63", "64", "65", "67", "68", "69"]
for sector in sectors:
file = f"../research/star_systems/TOI-4504/lightkurve/{sector}/{sector}.fits"
lc = TessLightCurve.read(file)
# analyze lc: calculate time and depth of transit. Calculate T1, T2, T3, T4, TT and depth

To address the task of analyzing a single transit light curve to determine transit parameters without folding or periodogram analysis, we develop a methodology based on numerical transit modeling and parameter optimization. The approach combines light curve preprocessing with transit model fitting using PyTransit's RoadRunnerModel, leveraging numerical optimization to derive contact times and transit depth directly from the observed flux variations.

Transit Parameter Analysis Methodology

The transit of an exoplanet across its host star follows a characteristic light curve pattern defined by four contact points (T_1-T_4) and a depth dependent on the planetary and stellar radii[1] [2]. The analysis requires accurate modeling of the transit shape, accounting for limb darkening and orbital geometry[1] [3].

Light Curve Preprocessing

1. **Outlier Removal and Detrending**:
   The raw light curve undergoes outlier removal using sigma clipping and detrending via a Savitzky-Golay filter to isolate the transit signal from systematics and noise[3].

   ```
   lc_clean = lc.remove_outliers().flatten(window_length=101)
   ```

2. **Transit Localization**:
   The approximate transit center (t_0) is identified as the time of minimum flux. A 1-day window

around t_0 isolates the transit event for focused analysis.

```
min_flux_index = np.argmin(lc_clean.flux)
t0_guess = lc_clean.time[min_flux_index].value
mask = (lc_clean.time >= t0_guess - 0.5) & (lc_clean.time <= t0_guess + 0.5)
lc_transit = lc_clean[mask]
```

Transit Model Fitting

The PyTransit RoadRunnerModel[4] evaluates transit light curves using parameters:

- **k**: Planet-to-star radius ratio (R_p/R_s)
- **ldc**: Limb darkening coefficients
- **t_0**: Central transit time
- **a**: Scaled semi-major axis (a/R_s)
- **i**: Orbital inclination
- **p**: Orbital period (fixed or fitted)

Parameter Initialization and Bounds:

```
tm = RoadRunnerModel('quadratic')
tm.set_data(time)
initial_params = [0.1, 0.5, 0.1, 10.0, np.pi/2, t0_guess, 10.0]  # k, ldc1, ldc2, a, i, t
bounds = [
    (0.01, 0.3), (0, 1), (0, 1),
    (5, 20), (np.pi/2 - 0.1, np.pi/2 + 0.1),
    (t0_guess - 0.1, t0_guess + 0.1), (1, 100)
]
```

Model Optimization:
A least-squares optimizer minimizes residuals between the model and observed flux:

```
def loss(params):
    k, ldc1, ldc2, a, i, t0, p = params
    model = tm.evaluate(k=k, ldc=[ldc1, ldc2], t0=t0, p=p, a=a, i=i)
    return np.sum((flux - model)**2)
result = minimize(loss, initial_params, method='L-BFGS-B', bounds=bounds)
best_params = result.x
```

Contact Time Derivation

High-resolution model evaluation identifies flux thresholds corresponding to contact points:

- **T_1/T_4**: Flux crosses 1 - depth/2 during ingress/egress
- **T_2/T_3**: Minimum flux points marking full planetary coverage

```
time_high_res = np.linspace(time.min(), time.max(), 10000)
model_high_res = tm.evaluate(...)
```

```
threshold = 1 - depth/2
transit_indices = np.where(model_high_res < threshold)[^0]
T1 = time_high_res[transit_indices[^0]]
T4 = time_high_res[transit_indices[-1]]
min_indices = np.where(model_high_res == model_high_res.min())[^0]
T2 = time_high_res[min_indices[^0]]
T3 = time_high_res[min_indices[-1]]
TT = T4 - T1
depth = 1 - model_high_res.min()
```

Functional Implementation

```
from lightkurve import TessLightCurve
import numpy as np
from pytransit import RoadRunnerModel
from scipy.optimize import minimize

def analyze_lightcurve():
    sectors = ["28"]
    for sector in sectors:
        file = f"../research/star_systems/TOI-4504/lightkurve/{sector}/{sector}.fits"
        lc = TessLightCurve.read(file)

        # Preprocess light curve
        lc_clean = lc.remove_outliers().flatten(window_length=101)

        # Locate transit
        min_flux_idx = np.argmin(lc_clean.flux)
        t0_guess = lc_clean.time[min_flux_idx].value
        mask = (lc_clean.time >= t0_guess - 0.5) & (lc_clean.time <= t0_guess + 0.5)
        lc_transit = lc_clean[mask]

        # Prepare data
        time = lc_transit.time.value
        flux = lc_transit.flux.value
        flux /= np.median(flux[flux > np.percentile(flux, 10)])

        # Initialize model
        tm = RoadRunnerModel('quadratic')
        tm.set_data(time)

        # Parameter setup
        params_init = [0.1, 0.5, 0.1, 10.0, np.pi/2, t0_guess, 10.0]  # k, ldc1, ldc2, a,
        bounds = [
            (0.01, 0.3), (0, 1), (0, 1),
            (5, 20), (np.pi/2 - 0.1, np.pi/2 + 0.1),
            (t0_guess - 0.1, t0_guess + 0.1), (1, 100)
        ]

        # Optimization
        def loss(params):
            k, ldc1, ldc2, a, i, t0, p = params
            model = tm.evaluate(k=k, ldc=[ldc1, ldc2], t0=t0, p=p, a=a, i=i)
            return np.sum((flux - model) ** 2)
```

```
        result = minimize(loss, params_init, method='L-BFGS-B', bounds=bounds)
        k, ldc1, ldc2, a, i, t0, p = result.x

        # High-res contact time analysis
        time_hr = np.linspace(time.min(), time.max(), 10000)
        tm.set_data(time_hr)
        model_hr = tm.evaluate(k=k, ldc=[ldc1, ldc2], t0=t0, p=p, a=a, i=i)
        depth = 1 - model_hr.min()

        transit_mask = model_hr < (1 - depth/2)
        transitions = np.where(np.diff(transit_mask.astype(int)))[^0]
        T1, T4 = time_hr[transitions[^0]], time_hr[transitions[-1]]

        min_idx = np.argmin(model_hr)
        T2 = time_hr[np.where(transit_mask)[^0][^0]]
        T3 = time_hr[np.where(transit_mask)[^0][-1]]
        TT = T4 - T1

        print(f"T1: {T1:.5f}, T2: {T2:.5f}, T3: {T3:.5f}, T4: {T4:.5f}, TT: {TT:.5f} d, D

    return
```

Analytical Considerations

1. **Limb Darkening**: Quadratic limb darkening coefficients significantly affect transit depth and shape[1] [4]. The RoadRunnerModel incorporates these through user-provided coefficients.
2. **Orbital Geometry**: The impact parameter
$$b = \frac{a}{R_*}\cos i$$
determines transit duration and contact time symmetry[2]. High inclinations (
$$i \approx 90^\circ$$
) produce near-symmetric ingress/egress.
3. **Parameter Degeneracy**: Single-transit fits exhibit degeneracy between
$$a/R_*$$
and orbital period
$$P$$
. Fixing
$$P$$
based on external data improves parameter accuracy[4].

This methodology provides a robust framework for single-transit analysis, though users should validate results with follow-up observations or prior constraints on orbital period.

⁂

1. https://www.telurium.net/PDF/The exoplanet potential from single-transit detection.pdf
2. https://rfo.org/wp-content/uploads/2024/08/Exoplanet-Transits-at-RFO.pdf
3. https://sab-astro.org.br/wp-content/uploads/2023/04/ParaProceedings-4.pdf
4. https://github.com/hpparvi/PyTransit