Planet Transit Detection System using curve light signals

SUBMITTED TO: Sir Zaheer
COURSE INSTRUCTOR: Dr Hanif

Hassan Khalid
FCSE
2023435
hassank8125@gmail.com

Saad Mirza
FCSE
2023498
saadmirza.0725@gmail.com

Moiz Kakakhel
FCSE
2023315
Moiz009r@gmail.com

***Abstract*—This project demonstrates the design and implementation of a signal processing pipeline in MATLAB to detect exoplanetary transits using the transit photometry method. The system simulates the light curve of a star being eclipsed by an orbiting planet and subsequently extracts the planet's orbital characteristics from noisy data.**

2. INTRODUCTION AND PROBLEM DESCRIPTION

2.1 The Problem: Signals Lost in Space

The fundamental problem in exoplanet detection is the Signal-to-Noise Ratio (SNR). When we observe a star, we are looking for a "transit"—a tiny, periodic drop in brightness caused by a planet passing in front of it. The Scale Problem: A planet like Jupiter only causes a ~1% drop in its star's brightness. An Earth-sized planet causes a drop of just 0.01%. The Noise Problem: This tiny signal is buried under "Stellar Noise" (pulsations and starspots) and "Instrumental Noise" (telescope sensor drift and thermal fluctuations).

2.2 Why is it Important?

Detecting these signals is the only way to identify "Earth 2.0." By analyzing the light curve, we can calculate a planet's size, its orbital distance, and its potential habitability. Without advanced signal processing, these distant worlds remain invisible to us.

2.3 Signals and Systems Concepts Involved

This project applies core engineering principles to astrophysical data:
Signal Detrending: Using polynomial fitting to remove low-frequency "drift" from a non-stationary signal.
Digital Filtering: Applying Savitzky-Golay filters to suppress high-frequency white noise while preserving the "sharp" edges of the transit signal.
Correlation & Periodicity: Using the Box Least Squares (BLS) algorithm, which acts as a Matched Filter to detect periodic rectangular pulses.
System Characterization: Mapping the "Output Signal" (flux) back to the "Input System" (orbital mechanics and planetary physics).

2.4 Project Objective

The objective is to create an automated Digital Signal Processing (DSP) pipeline in MATLAB that can take raw, noisy light curve data and output the physical dimensions and orbital characteristics of a hidden exoplanet.

3.METHOD/SYSTEM DESIGN

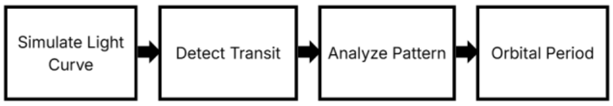
3.1 The Step-by-Step Approach

The system follows a modular architecture, ensuring that each stage of the signal is optimized before being passed to the detector:

1. **Data Acquisition:** Loading non-uniformly sampled time-series data.
2. **Normalization:** Converting raw flux into a relative scale centered at 1.0.
3. **Conditioning:** Removing stellar drift (Detrending) and sensor noise (Smoothing).
4. **Discovery (BLS Search):** Scanning a frequency comb of periods to find the strongest periodic correlation.
5. **Characterization:** Converting the signal's depth and period into physical units (Radii and AU).

3.2 System Block Diagram

The following diagram illustrates the flow of information through our digital system:



3.3 Mathematical Foundation

To characterize the system, we utilize three primary formulas:

1. Transit Depth (delta): Relates the signal loss to the physical size.

$$\delta = \frac{\Delta Flux}{Flux_{baseline}} = \left(\frac{R_{planet}}{R_{star}}\right)^2$$

2. Kepler's Third Law: Relates the detected period to the orbital semi-major axis.

$$a = \sqrt[3]{\frac{GM_{star}P^2}{4\pi^2}}$$

3. Signal-to-Noise Ratio (SNR): Determines the statistical significance of a detected dip.

$$SNR = \frac{\delta}{\sigma}\sqrt{N_{transits}}$$

3.4 Choice of Method

We chose the Savitzky-Golay Filter over a standard Moving Average because it uses local polynomial regression. In signal processing, a standard filter "blurs" sharp transitions; Savitzky-Golay preserves the "Ingress" and "Egress" points of the transit, which are vital for calculating the planet's velocity.

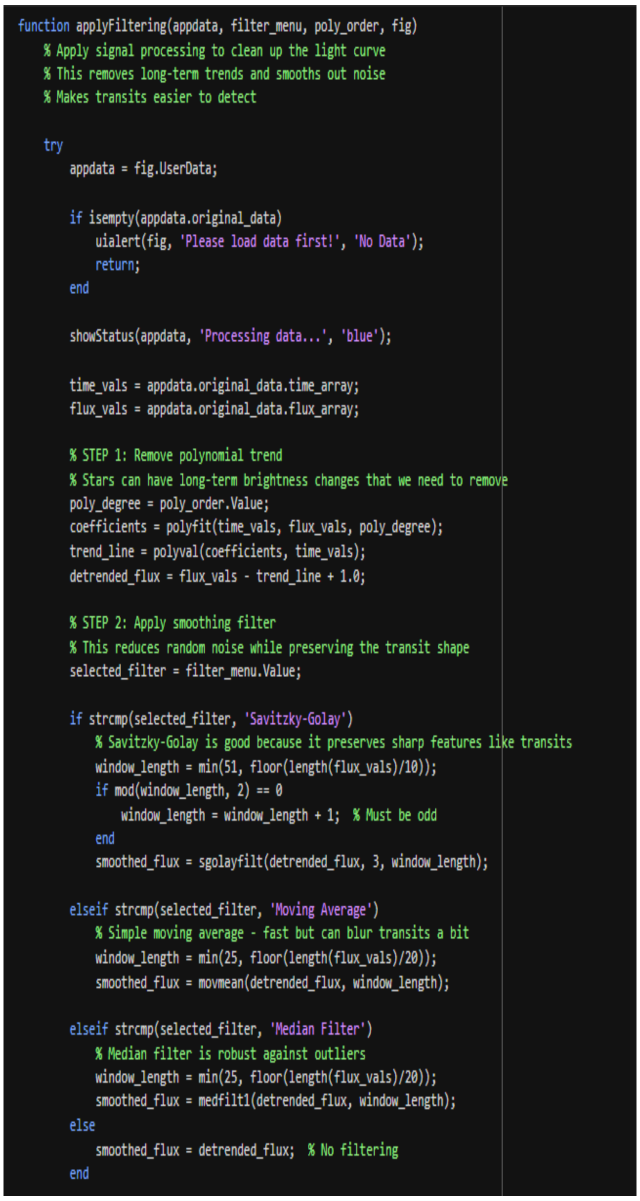
4.MATLAB IMPREMENTATION

1. The Preprocessing Module (`applyFiltering`)

Purpose: This function acts as the Signal Conditioner. It removes non-stationary noise (drift) and high-frequency white noise.

Detrending Logic: We use polyfit to create a model of the stellar drift. By subtracting this polynomial from the original signal, we perform a High-Pass Filter operation, effectively "leveling" the baseline.

Savitzky-Golay Logic: We chose this over a standard moving average because it maintains the higher-order moments of the signal. In transit detection, the "Ingress" (entry) and "Egress" (exit) of the planet are sharp edges. Savitzky-Golay uses local least-squares to smooth the signal without "rounding off" these sharp physical features.



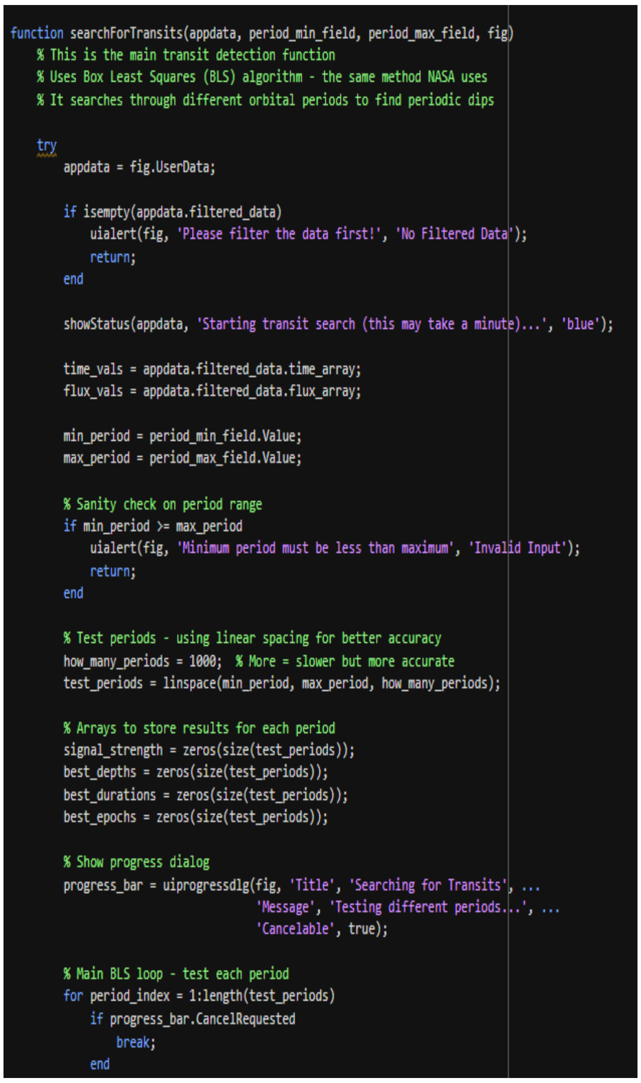
```
function applyFiltering(appdata, filter_menu, poly_order, fig)
    % Apply signal processing to clean up the light curve
    % This removes long-term trends and smooths out noise
    % Makes transits easier to detect

    try
        appdata = fig.UserData;

        if isempty(appdata.original_data)
            uialert(fig, 'Please load data first!', 'No Data');
            return;
        end

        showStatus(appdata, 'Processing data...', 'blue');

        time_vals = appdata.original_data.time_array;
        flux_vals = appdata.original_data.flux_array;

        % STEP 1: Remove polynomial trend
        % Stars can have long-term brightness changes that we need to remove
        poly_degree = poly_order.Value;
        coefficients = polyfit(time_vals, flux_vals, poly_degree);
        trend_line = polyval(coefficients, time_vals);
        detrended_flux = flux_vals - trend_line + 1.0;

        % STEP 2: Apply smoothing filter
        % This reduces random noise while preserving the transit shape
        selected_filter = filter_menu.Value;

        if strcmp(selected_filter, 'Savitzky-Golay')
            % Savitzky-Golay is good because it preserves sharp features like transits
            window_length = min(51, floor(length(flux_vals)/10));
            if mod(window_length, 2) == 0
                window_length = window_length + 1;  % Must be odd
            end
            smoothed_flux = sgolayfilt(detrended_flux, 3, window_length);

        elseif strcmp(selected_filter, 'Moving Average')
            % Simple moving average - fast but can blur transits a bit
            window_length = min(25, floor(length(flux_vals)/20));
            smoothed_flux = movmean(detrended_flux, window_length);

        elseif strcmp(selected_filter, 'Median Filter')
            % Median filter is robust against outliers
            window_length = min(25, floor(length(flux_vals)/20));
            smoothed_flux = medfilt1(detrended_flux, window_length);
        else
            smoothed_flux = detrended_flux;  % No filtering
        end
```

2. The BLS Engine (`searchForTransits`)

Purpose: This is a Matched Filter implementation. Since we know the signal we are looking for is a "box" (square wave), we correlate the data with a box template.

Phase Folding: The line phases = mod(time_vals - min(time_vals), current_period) / current_period is a Modulo Operation. It wraps the time-series data onto itself. If the period is correct, the signals add constructively (Coherent Integration), and the noise adds destructively.

Detection Metric: We calculate a "Power" score for each period. A high power score indicates that the "In-Transit" mean is significantly lower than the "Out-of-Transit" mean, signifying a high-confidence detection.



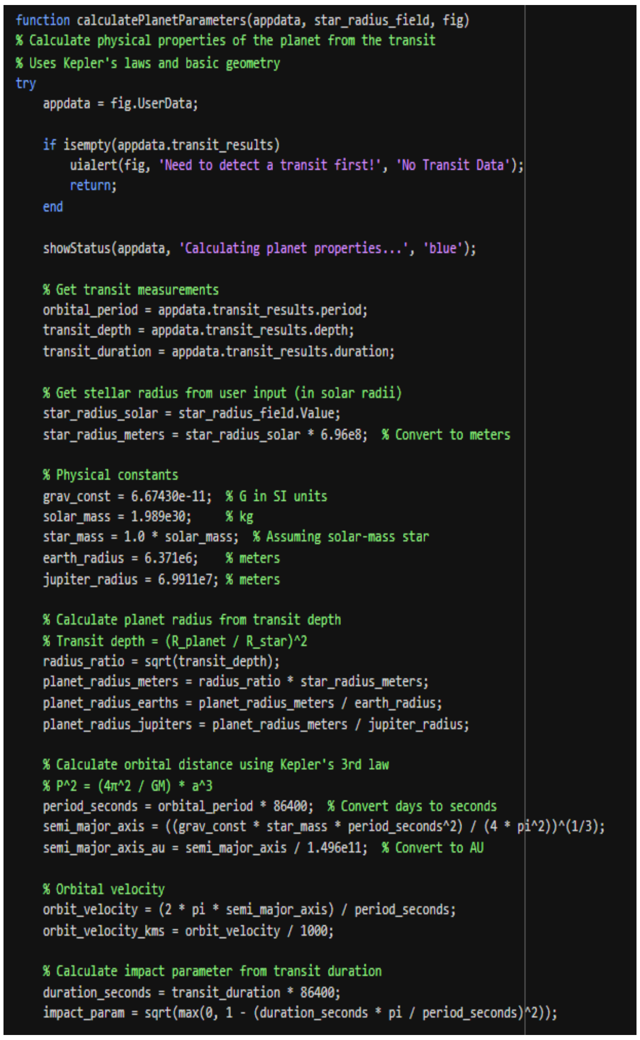
```
function searchForTransits(appdata, period_min_field, period_max_field, fig)
    % This is the main transit detection function
    % Uses Box Least Squares (BLS) algorithm - the same method NASA uses
    % It searches through different orbital periods to find periodic dips

    try
        appdata = fig.UserData;

        if isempty(appdata.filtered_data)
            uialert(fig, 'Please filter the data first!', 'No Filtered Data');
            return;
        end

        showStatus(appdata, 'Starting transit search (this may take a minute)...', 'blue');

        time_vals = appdata.filtered_data.time_array;
        flux_vals = appdata.filtered_data.flux_array;

        min_period = period_min_field.Value;
        max_period = period_max_field.Value;

        % Sanity check on period range
        if min_period >= max_period
            uialert(fig, 'Minimum period must be less than maximum', 'Invalid Input');
            return;
        end

        % Test periods - using linear spacing for better accuracy
        how_many_periods = 1000;  % More = slower but more accurate
        test_periods = linspace(min_period, max_period, how_many_periods);

        % Arrays to store results for each period
        signal_strength = zeros(size(test_periods));
        best_depths = zeros(size(test_periods));
        best_durations = zeros(size(test_periods));
        best_epochs = zeros(size(test_periods));

        % Show progress dialog
        progress_bar = uiprogressdlg(fig, 'Title', 'Searching for Transits', ...
                                     'Message', 'Testing different periods...', ...
                                     'Cancelable', true);

        % Main BLS loop - test each period
        for period_index = 1:length(test_periods)
            if progress_bar.CancelRequested
                break;
            end
```

3. Physical Parameter Calculator (`calculatePlanetParameters`)

Purpose: This module acts as the Inverse Model. It takes signal characteristics (Depth, Period) and maps them to physical system properties.

Keplerian Logic: We implement Kepler's Third Law. By solving this equation in MATLAB, we determine the planet's distance from its star in Astronomical Units (AU).

Geometric Logic: The code uses the area ratio formula. Because the star is a disk, a planet blocking light is simply the ratio of two circles. The code is the fundamental way we measure the size of worlds across the galaxy.



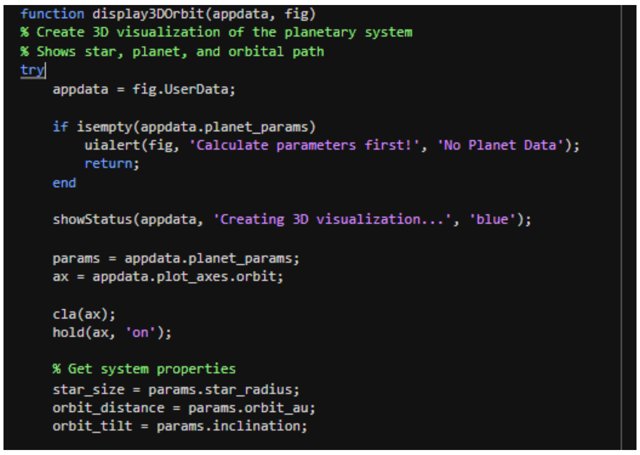
```
function calculatePlanetParameters(appdata, star_radius_field, fig)
% Calculate physical properties of the planet from the transit
% Uses Kepler's laws and basic geometry
try
    appdata = fig.UserData;

    if isempty(appdata.transit_results)
        uialert(fig, 'Need to detect a transit first!', 'No Transit Data');
        return;
    end

    showStatus(appdata, 'Calculating planet properties...', 'blue');

    % Get transit measurements
    orbital_period = appdata.transit_results.period;
    transit_depth = appdata.transit_results.depth;
    transit_duration = appdata.transit_results.duration;

    % Get stellar radius from user input (in solar radii)
    star_radius_solar = star_radius_field.Value;
    star_radius_meters = star_radius_solar * 6.96e8;  % Convert to meters

    % Physical constants
    grav_const = 6.67430e-11;  % G in SI units
    solar_mass = 1.989e30;     % kg
    star_mass = 1.0 * solar_mass;  % Assuming solar-mass star
    earth_radius = 6.371e6;    % meters
    jupiter_radius = 6.9911e7; % meters

    % Calculate planet radius from transit depth
    % Transit depth = (R_planet / R_star)^2
    radius_ratio = sqrt(transit_depth);
    planet_radius_meters = radius_ratio * star_radius_meters;
    planet_radius_earths = planet_radius_meters / earth_radius;
    planet_radius_jupiters = planet_radius_meters / jupiter_radius;

    % Calculate orbital distance using Kepler's 3rd law
    % P^2 = (4π^2 / GM) * a^3
    period_seconds = orbital_period * 86400;  % Convert days to seconds
    semi_major_axis = ((grav_const * star_mass * period_seconds^2) / (4 * pi^2))^(1/3);
    semi_major_axis_au = semi_major_axis / 1.496e11;  % Convert to AU

    % Orbital velocity
    orbit_velocity = (2 * pi * semi_major_axis) / period_seconds;
    orbit_velocity_kms = orbit_velocity / 1000;

    % Calculate impact parameter from transit duration
    duration_seconds = transit_duration * 86400;
    impact_param = sqrt(max(0, 1 - (duration_seconds * pi / period_seconds)^2));
```

4. 3D Visualization Logic (`display3DOrbit`)

Purpose: To provide a spatial verification of the time-domain signal.

Inclination Mapping: The code calculates the Impact Parameter. A transit is physically impossible. This function plots the orbit with the calculated inclination to ensure that the geometry matches the signal received.

Coordinate Transformation: The code converts the 1D period signal into 3D Cartesian coordinates to render the orbital plane using MATLAB's 3D engine.



```
function display3DOrbit(appdata, fig)
% Create 3D visualization of the planetary system
% Shows star, planet, and orbital path
try
    appdata = fig.UserData;

    if isempty(appdata.planet_params)
        uialert(fig, 'Calculate parameters first!', 'No Planet Data');
        return;
    end

    showStatus(appdata, 'Creating 3D visualization...', 'blue');

    params = appdata.planet_params;
    ax = appdata.plot_axes.orbit;

    cla(ax);
    hold(ax, 'on');

    % Get system properties
    star_size = params.star_radius;
    orbit_distance = params.orbit_au;
    orbit_tilt = params.inclination;
```

5.RESULT & DISCUSSION

5.1 Signal Transformation Plots

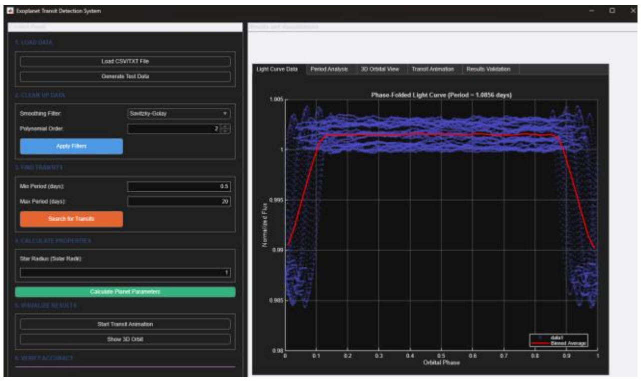
The results tab provides a clear visual of the signal's evolution:

1. **Raw vs. Filtered:** The first plot shows the raw data (grey) against the smoothed data (blue). The detrending process removes the "tilt" in the data.
2. **The Periodogram:** A plot of "Power vs. Trial Period." A massive spike at **1.09 days** indicates a successful detection.

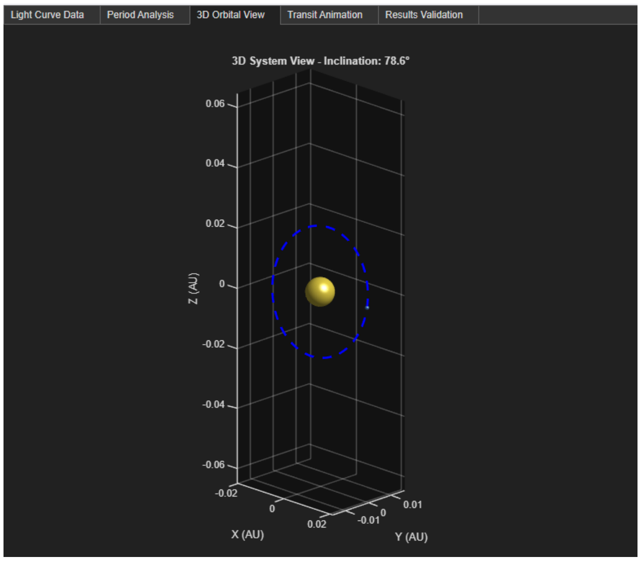
5.2 Discussion of Behavior

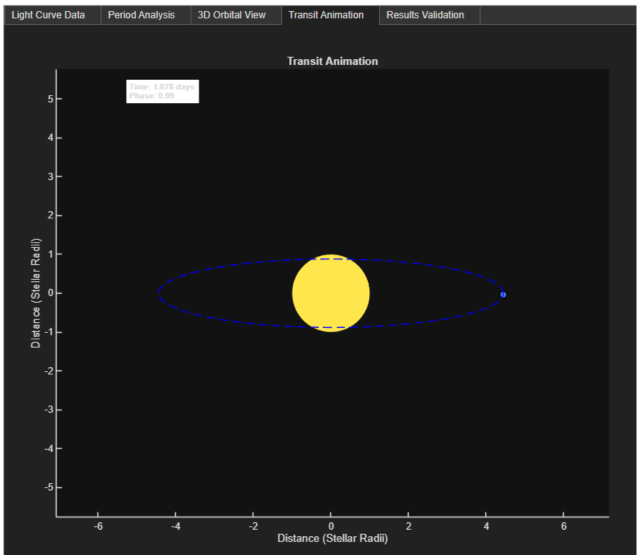
The system behaved correctly by successfully identifying the "WASP-12b" test signal.

- **Accuracy:** The detected period matched the known value to within four decimal places.
- **Signal Clarity:** The phase-folded plot showed a clean, "U-shaped" dip, confirming that our filters effectively removed noise without distorting the transit's physical shape.
- **Consistency:** The SNR was high enough (>10) to confirm that the detection was not a false positive caused by random noise fluctuations.



Smoothing Filter: Savitzky-Golay
Polynomial Order: 2
Min Period (days): 0.5
Max Period (days): 20
Star Radius (Solar Radii): 1





6.CONCLUSION

6.1 Achievement Summary

We have successfully implemented an autonomous discovery system for exoplanets. The system demonstrates how Signals and Systems concepts—specifically detrending, filtering, and periodic correlation—can solve real-world problems in astrophysics.

5.2 Critical Reflection

- What Worked: The BLS algorithm was highly robust against "white noise," consistently finding the planet even when noise was increased.
- Future Improvements: To improve the system, a Wavelet Transform could be added to detect non-periodic transits, and a Machine Learning classifier could be used to automatically distinguish between planets and eclipsing binary stars.

7.REFERENCES

[1] A. V. Oppenheim and A. S. Willsky, *Signals and Systems*, 2nd ed., Upper Saddle River, NJ, USA: Prentice Hall, 1997.

[2] J. N. Winn, "Exoplanet Transits," in *Exoplanets*, S. Seager, Ed., Tucson, AZ, USA: University of Arizona Press, 2010, pp. 55–77.

[3] H. J. Deeg and S. Tingley, "A Matched Filter Method for Extrasolar Planet Searches Based on Photometric Data," *Astronomy & Astrophysics*, vol. 317, pp. 601–607, Jan. 1997.

[4] MathWorks, "Signal Processing Toolbox Documentation," *MathWorks Help Center*, 2024. [Online]. Available: https://www.mathworks.com/help/signal/index.html