Exoplanets: Modeling the Transit Light Curve

By Maggie Ramsey, Ainsley jacquemain, Lisa Chiang

Motivations

Understand Light Curves

Modeling demonstrates what the typical light curve looks likes, helping us understand the different components that make each transit unique.

Finding the Radius

Using the information we gained from the model, we can quite accurately predict the radius and of the planet.

Gathering More Data

Knowing the radius, we can make further predictions that will help us understand what the planet would be like, and if it would be habitable.

The Steps to the Final Result

1. Making the General Model
2. Adjusting to Fit Real Data
3. Finding the Reduced Chi Squared
4. Repeat 1–3 with a Better Model

1.
Making the General Model

Making the General Model

Understand what makes up the general Model

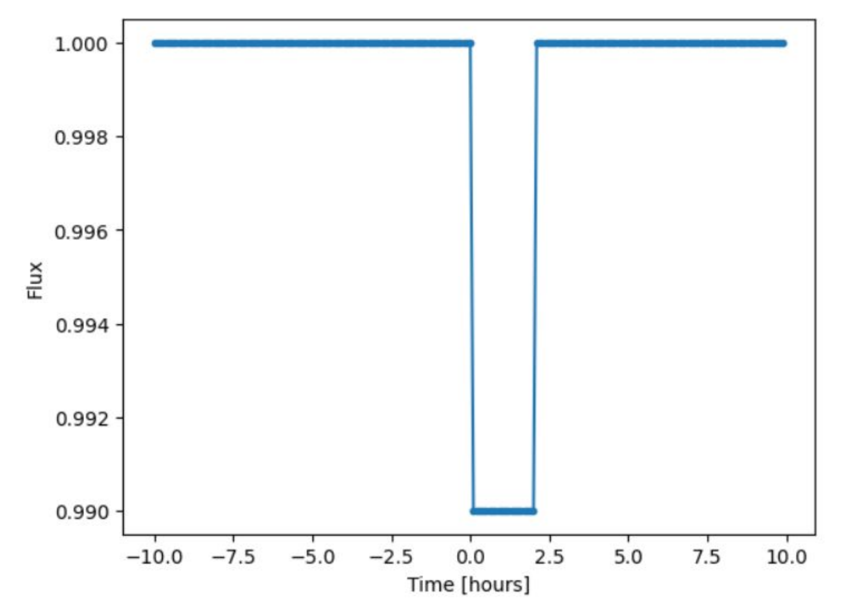
- **τ:** Changing this increases/decreases transit duration
- **T_0:** Changing this shifts the transit left/right
- **Start Time:** Changing this changes when the transit will <u>start</u>
- **End Time:** Changing this changes when the transit will <u>end</u>
- **Delta:** Changing this changes the depth/flux of light reduction

Making the General Model

Adding components to make the model function

- Assign index values to time array
- Using these values and the start/end time to determine where the flux/transit will start
- Graphing the components

```
delta = 0.01 #graphing what we just definded
flux_arr[ind] = 1.0 - delta
plt.plot(time_arr, flux_arr, linestyle = "-", marker = ".")
plt.xlabel("Time [hours]")
plt.ylabel("Flux")
plt.show()
```



Making the General Model

Turn the information into a single function to be used as a model

-
```
def generate_transit_lightcurve(time_arr, t_o, tau, delta):
  flux_arr = np.zeros(np.shape(time_arr)) + 1.0
  time_start = t_0 - tau / 2
  time_end = t_0 + tau / 2
  ind = np.where((time_arr >= time_start) & (time_arr <= time_end))
  flux_arr[ind] = 1.0 - delta
  return flux_arr
```

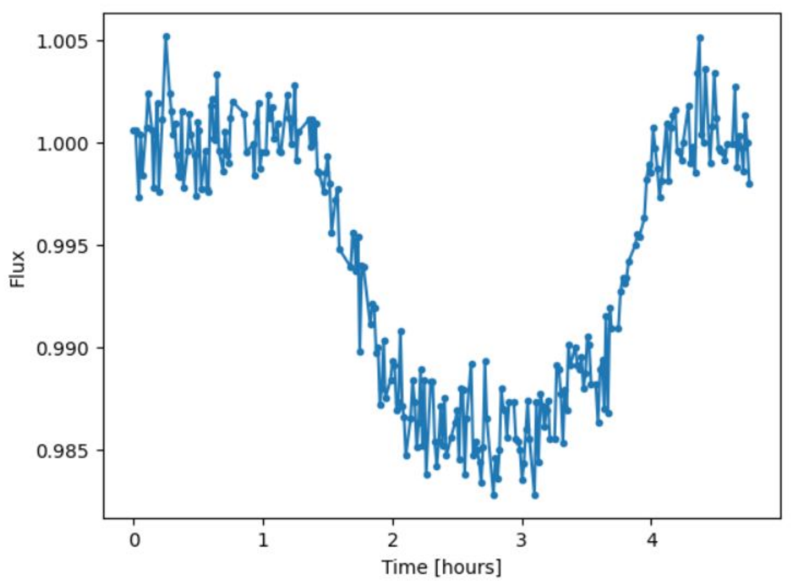
- Now we can use this model at any point as long as we define a specific **time array, T_0, τ,** and **Delta**

2.
Adjusting to Fit Real Data

Adjusting to Fit Real Data

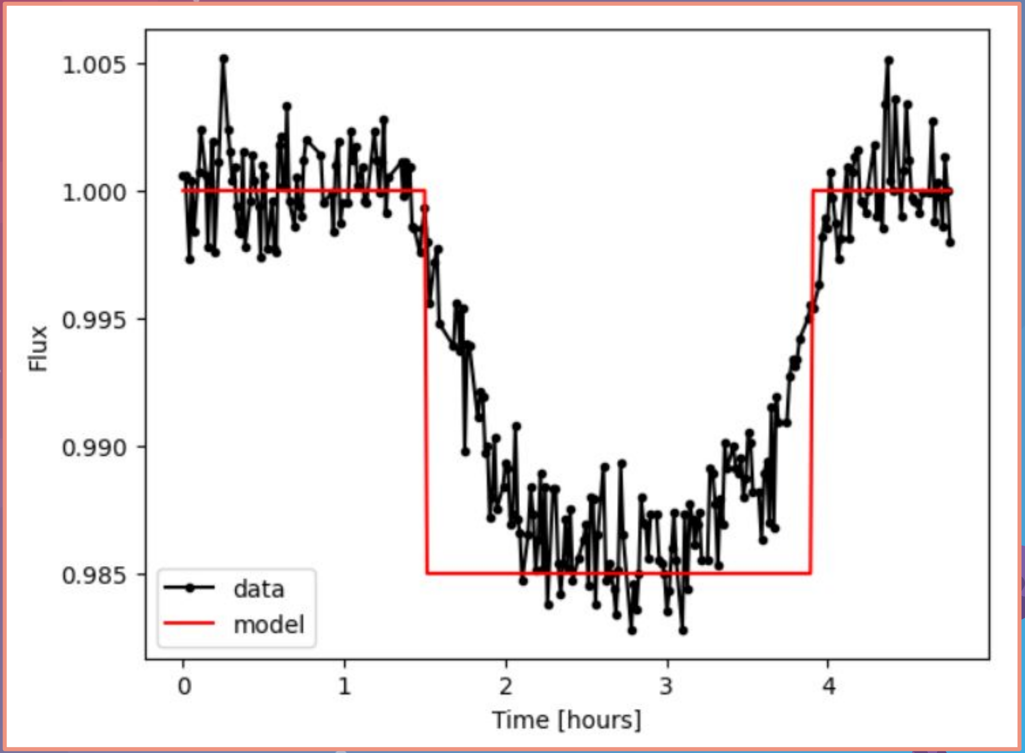
- Import data from the NASA Exoplanet Archive
    - HAT-P-1 b (Johnson et al. 2008 with 253 data points)
- Use astropy.io.ascii to read the data and make it a table
- Graph using values from the table

```
time_obs = dat["HJD"] #putting our time data in
time_obs = (time_obs - time_obs[0]) * 24.0 * u.hour #our time was in da
flux_obs = dat["Relative_Flux"] # putting our flux data in
plt.plot (time_obs, flux_obs, marker = ".")
plt.xlabel("Time [hours]")
plt.ylabel("Flux")
plt.show() #ploting the transit for HAT-P-1b
```



Adjusting to Fit Real Data

- estimating parameters for the model
  - **τ:** 2.4 hours
  - T_0: 2.7 hours
  - **Delta:** 0.015
- Overplot model



3.
Finding the Reduced Chi Squared

Finding the Reduced Chi Squared

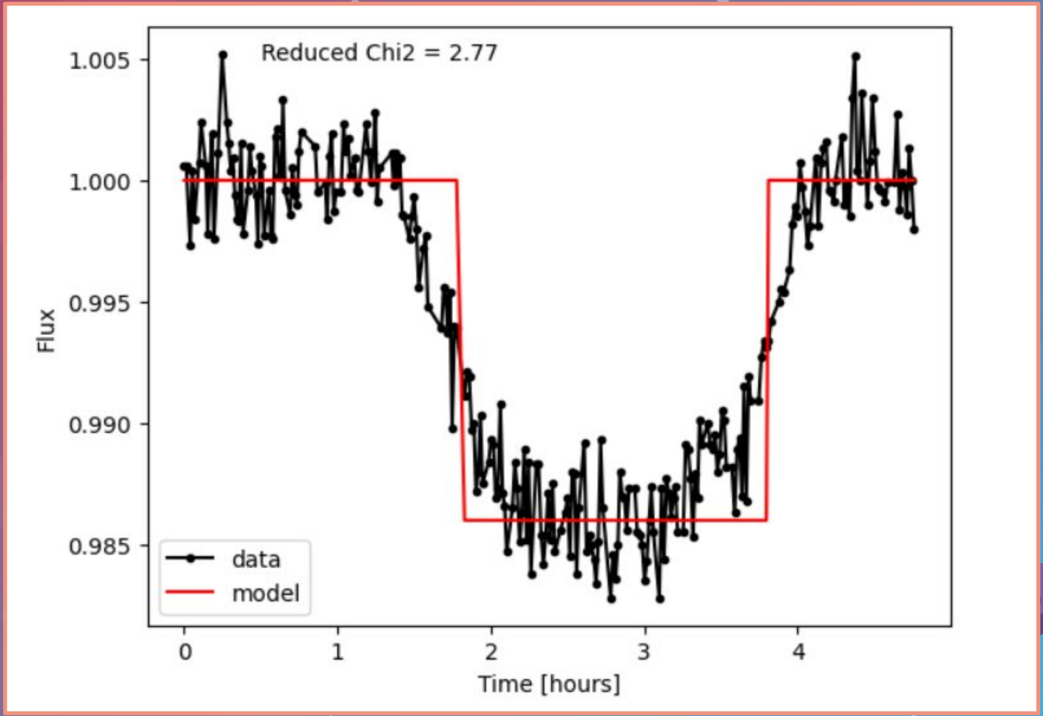
Calculating the Reduced Chi Squared

```
ind = np.where(time_obs < 1.5 * u.hour)
error = np.std(flux_obs[ind])

degree_of_freedom = len(time_obs) - 3
reduced_Chi2 = np.sum(((flux_obs - flux_model) / error)**2) / degree_of_freedom
print(reduced_Chi2)
```

```
5.4712288561723925
```

Finding the Reduced Chi Squared

- Re-adjusting Values until we get as low a Reduced Chi Squared as possible
  - **τ:** 2.8 hours
  - **T_0:** 2.0 hours
  - **Delta:** 0.014
- Re-graphing the overplotted model with new Reduced Chi Squared listed



4.
Repeat 1–3 with a Better Model

Repeat Step 1

To more accurately measure the shape of the light curve, we will define a trapezoidal model that slopes in and out of the transit

- **τ:** Changing this increases/decreases transit duration
- **T_0:** Changing this shifts the transit left/right
- **Start Time:** Changing this changes when the transit will start
- **End Time:** Changing this changes when the transit will end
- **Delta:** Changing this changes the depth/flux of light reduction
- **Ingress egress time:** Changing this changes the slope in/out of the transit

NEW!

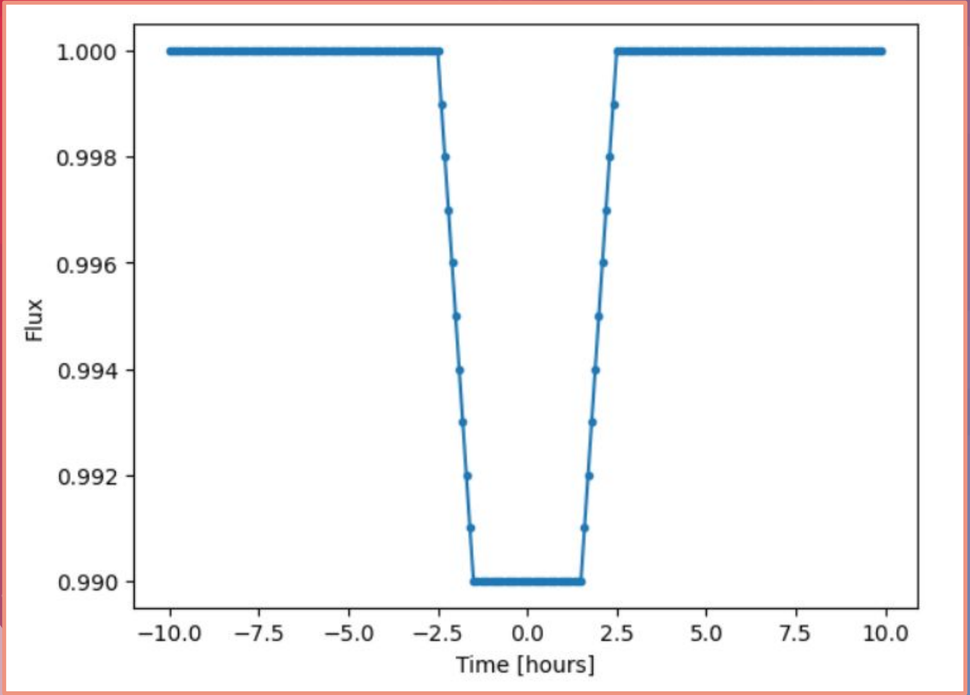
Repeat Step 1

- Defining our new Model Function

```
def trapizoid_model(time_arr, t_o, tau, delta, ingress_egress_time):
  flux_arr = np.zeros(np.shape(time_arr)) + 1.0
  time_start = t_0 - tau / 2
  time_end = t_0 + tau / 2

  #Find the indices for the transit
  transit_start = np.where(time_arr >= time_start)[0][0]
  transit_end = np.where(time_arr <= time_end)[0][-1]

  #changing the flux array for a trapezoidal shape (#making the lines down/up an adjustable V slope)
  for i in range(transit_start, transit_end + 1):
      if time_arr[i] < time_start + ingress_egress_time:
        flux_arr[i] = 1.0 - delta * (time_arr[i] - (time_start)) / ingress_egress_time
      elif time_arr[i] > time_end - ingress_egress_time:
        flux_arr[i] = 1.0 - delta * (time_end - time_arr[i]) / ingress_egress_time
      else:
          flux_arr[i] = 1.0 - delta

  return flux_arr
```

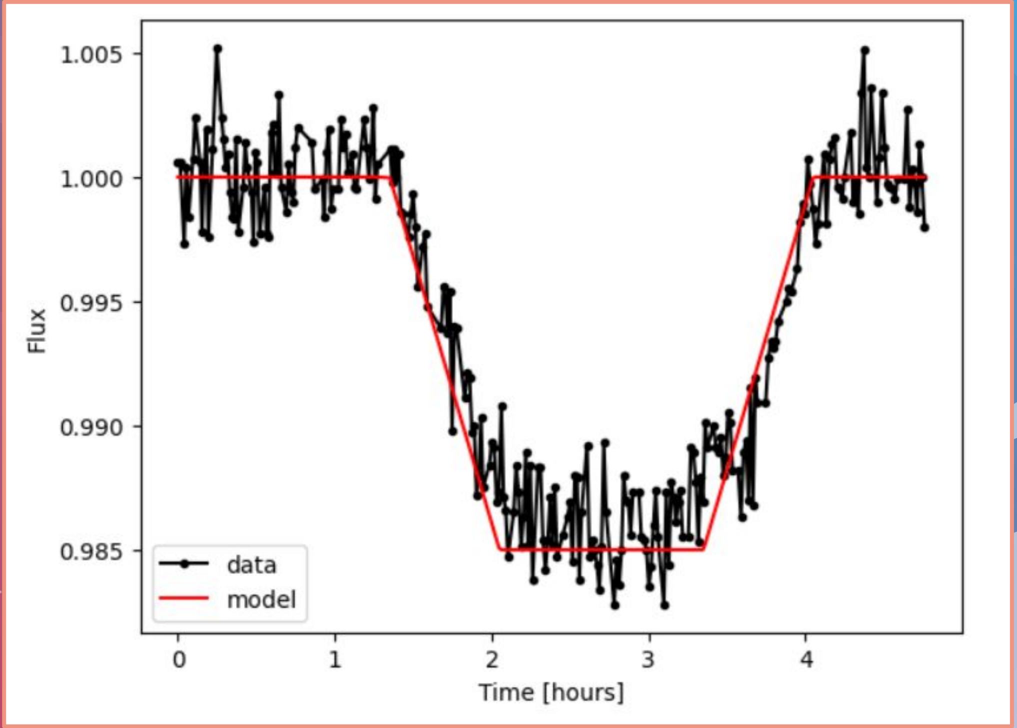
Repeat Step 1

- Graphing New Model



Repeat Step 2

- estimating parameters for the model
  - **τ:** 2.7 hours
  - **T_0:** 2.7 hours
  - **Delta:** 0.015
  - **I/E time:** 0.7 hour
- Overplot model



Repeat Step 3

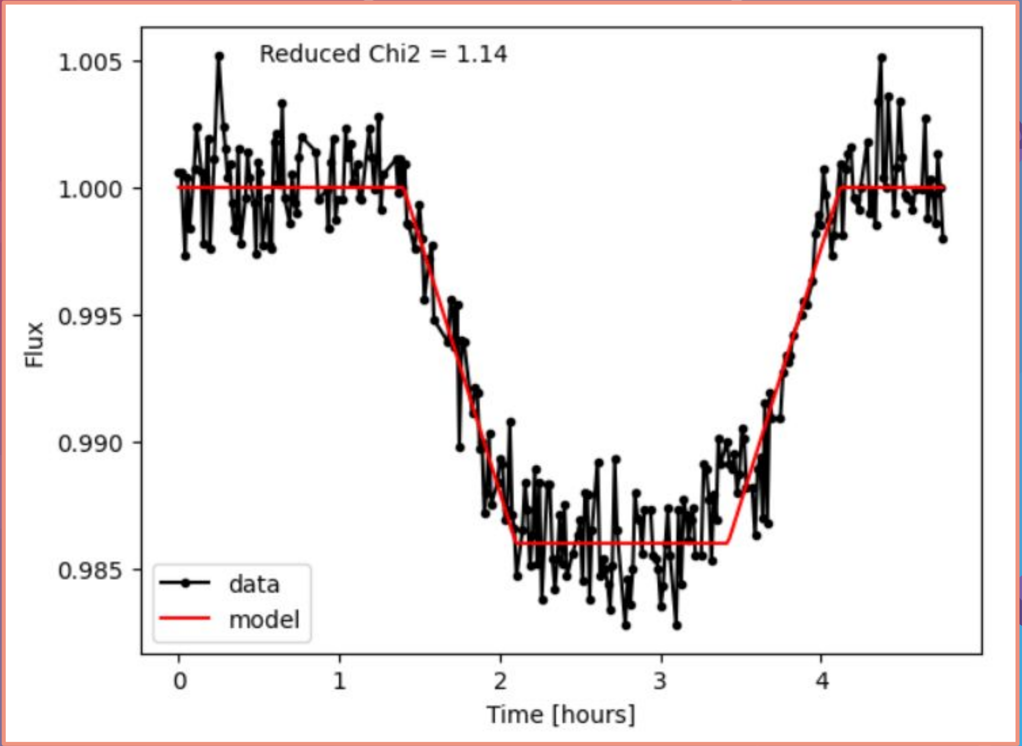
Calculating the Reduced Chi Squared

```
ind = np.where(time_obs < 1.5 * u.hour
error = np.std(flux_obs[ind])

degree_of_freedom = len(time_obs) - 3
reduced_Chi2 = np.sum(((flux_obs - flux_model2) / error)**2) / degree_of_freedom
print(reduced_Chi2)
```

```
1.5226942614463883
```

Repeat Step 3

- Re-adjusting Values until we get as low a Reduced Chi Squared as possible
  - **τ:** 2.76 hours
  - **T_0:** 2.72 hours
  - **Delta:** 0.014
  - **I/E time:** 0.7 hour
- Re-graphing the overplotted model with new Reduced Chi Squared listed



Conclusion

Box Model

This model works, and gives a fairly accurate reduced chi squared of **2.77**

Trapezoidal Model

This model work far better, giving an even more accurate reduced chi squared of **1.14**

Thank You!