In [ ]:
```
#HORIZONTAL TRANSLATION

import matplotlib.pyplot as plt
import numpy as np
from matplotlib import animation

%config InlineBackend.figure_format='retina'
%matplotlib osx

fig_animate, ax = plt.subplots()
dots = []

ax.scatter(0, 0, color="orange",marker="o",s=5000) #sun

dots.append(ax.plot([], [], linestyle='none', marker='o', markersize=2

ax.set_xlim([-10,10])
ax.set_ylim([-10,10])

data=np.linspace(-10,10,50)

def animate(z):
    dots[0][0].set_data(data[z],0)
    return dots

anim = animation.FuncAnimation(fig_animate, animate, frames=len(data),
writer = animation.writers['ffmpeg'](fps=30)
dpi=300
anim.save('dot.mp4', writer=writer,dpi=dpi)
```

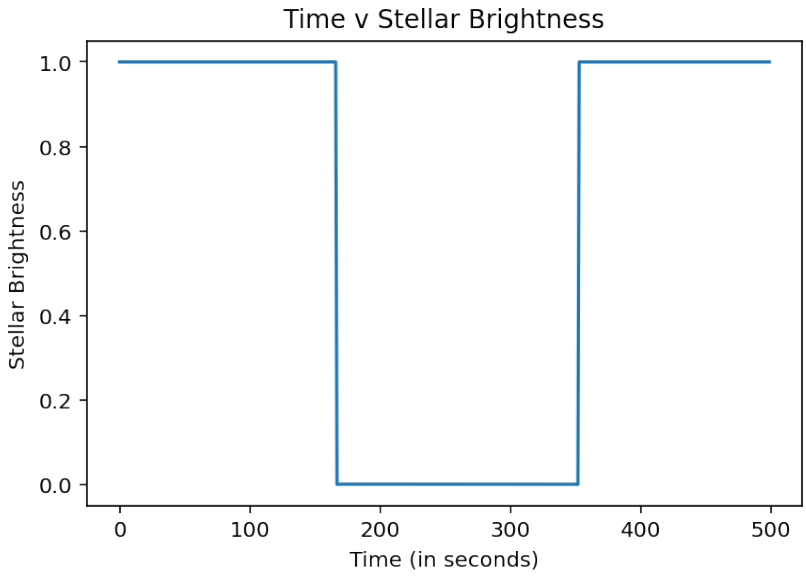
In [ ]:
```
#ORBIT

import numpy as np
import matplotlib.pyplot as plt
import scipy
import scipy.integrate
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FFMpegWriter

%config InlineBackend.figure_format

planet = 4 * 10 ** (-6)
star = 1.0
rp = np.array([1,0])
rs = np.array([0,0])
vp = np.array([0,0.75])
vs = np.array([0,0])
```

```
def ORBIT(d, t, m1, m2):
    rp = d[:2]
    vp = d[2:4]
    R = np.linalg.norm(rp)
    acceleration = m2 * (-rp) / R ** 3
    positionder = vp
    r_derivs = positionder
    v_derivs = acceleration
    derivs = np.concatenate((r_derivs, v_derivs))
    return derivs

params_0 = np.array([rp,vp])
params_0 = params_0.flatten()
time = np.linspace(0,5,500)

solve = scipy.integrate.odeint(ORBIT, params_0, time, args=(planet, st

solve_rp = solve[:,:2]

metadata = dict(title = 'orbit', artist = 'Matplotlib')
writer = FFMpegWriter(fps=100, metadata=metadata, bitrate=200000)
fig = plt.figure(dpi=200)

fig,ax = plt.subplots()
#ax.scatter(0, 0, color="orange",marker="o",s=5000)

with writer.saving(fig, "orbit.mp4", dpi=200):
    for i in range(len(time)):
        ax.clear()
        ax.scatter(solve_rp[i,0], solve_rp[i,1], color="blue", marker=
        ax.set_xlim(-1.5, 1.5)
        ax.set_ylim(-1.5, 1.5)
        plt.draw()
        plt.pause(0.01)
        writer.grab_frame()
```

In [22]:
```
#GRAPH

import numpy as np
from matplotlib import pyplot as plt
from matplotlib import animation
import scipy
import scipy.integrate

from mpl_toolkits.mplot3d import Axes3D
from matplotlib.animation import FFMpegWriter
from matplotlib import animation

%matplotlib osx
```

```
#variables
massExoplanet = 8.599968 * 10**24
massStar = 9.5473 * 10**29

rPlanet = 7454153
rStar = 3.2728 * 10**8
rOrbit = 6.46263 * 10**10

v0 = 35985.10901 #meters/second
#w = (2 * np.pi)/period #angular acc, meters/second

period = 130 * 24 * 60 * 60 #seconds
transitTime = int((2 * rStar) / v0) #8189
data = []


# -------------- plotting data ----------------- #

# intersecting area functions
def intersection_area(d, R, r):
    """Return the area of intersection of two circles.
    The circles have radii R and r, and their centres are separated by
    if d <= abs(R-r):
        # One circle is entirely enclosed in the other.
        return np.pi * min(R, r)**2
    if d >= r + R:
        # The circles don't overlap at all.
        return 0

    r2, R2, d2 = r**2, R**2, d**2
    alpha = np.arccos((d2 + r2 - R2) / (2*d*r))
    beta = np.arccos((d2 + R2 - r2) / (2*d*R))

    return (r2 * alpha + R2 * beta - 0.5 * (r2 * np.sin(2*alpha) + R2

for time in np.arange(0, period//4):
    d = abs(rOrbit - (v0 * time))
    #d = np.abs(rStar - (v0 * np.sin(w * time))) #with angular acceler

    area = intersection_area(d, rStar, rPlanet)
    depth = (np.sqrt(area/np.pi) / rStar)**2

    if depth == 0:
        data.append(1)
    else:
        data.append(depth)

%matplotlib inline
data_ = data[1770000:1820000]
data_small = data_[::100]
```

```
plt.plot(data_small)
plt.title("Time v Stellar Brightness")
plt.xlabel("Time (in seconds)")
plt.ylabel("Stellar Brightness")
plt.show()

%config InlineBackend.figure_format='retina'
%matplotlib osx

x_data = np.linspace(0, period//4, len(data_small))
y_data = data_small


# set up plot
fig,ax = plt.subplots(1,1)
ax.set_title("Time v Stellar Brightness")
ax.set_xlabel("Time (in seconds)")
ax.set_ylabel("Stellar Brightness")


# animation function.  This is called sequentially
def animate(i):
    x = x_data[:i]
    y = y_data[:i]
    ax.plot(x,y,color='blue')

# call the animator.  blit=True means only re-draw the parts that have
anim = animation.FuncAnimation(fig, animate, frames=500, interval=200)
anim.save('basic_animation.mp4', fps=60, extra_args=['-vcodec', 'libx2
```




In [ ]: