AA310 – Orbital Mechanics
Homework #3

Name: Luke Verlangieri

October 21, 2025

1 Problems

Problem 1.1 (Periods & Energies)**.**

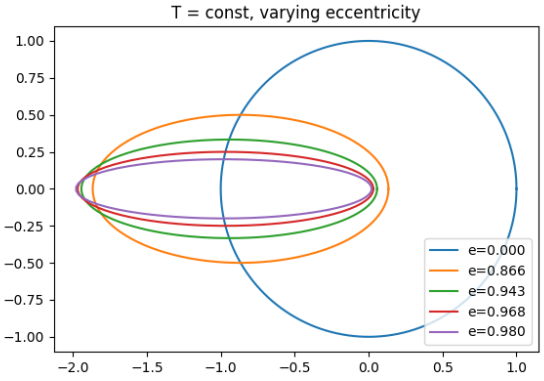
(a)

$$b = \frac{a}{n} = a\sqrt{(1-e^2)} \longrightarrow e = \sqrt{1 - \frac{1}{n^2}} \tag{1}$$

$$r = \frac{h^2}{\mu}\left(\frac{1}{1 + e\cos(\theta)}\right) = \frac{a(1-e^2)}{1 + e\cos(\theta)} \tag{2}$$

Plugging this into python for values of n 1, 2, 3, 4, 5.

n	e
1	0
2	0.866
3	0.943
4	0.968
5	0.98



(b) Here is the code I used to produce this graph

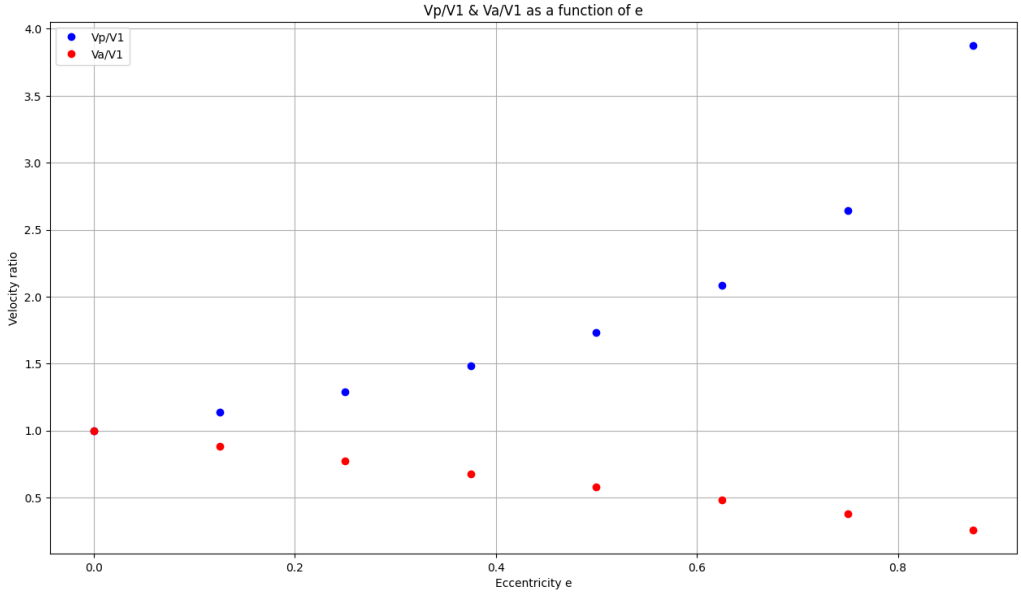
```
def part_a(n):
  return math.sqrt((1-(1/n**2)))

n_vals = [1,2,3,4,5]

# for n in n_vals:
#   print(f'For n of {n}: e = {part_a(n)}')

"""Part b"""
a = 1
theta = np.linspace(0, 2 * np.pi, 500) # domain for plot

eccentricities = [part_a(n) for n in n_vals]

fig, ax = plt.subplots()
for e in eccentricities:
    r = a * (1 - e**2) / (1 + e*np.cos(theta))
    x = r * np.cos(theta) #Add + a*e for geometric center, otherwise plts focal pt
    y = r * np.sin(theta)
    ax.plot(x, y, label=f"e={e:.3f}")

ax.set_aspect('equal')
ax.set_title("T = const, varying eccentricity")
ax.legend(loc='lower right')
plt.show()
```

(c) Comparing the generated plot to the plot in the homework pdf, it is clear to see that most drastic changes in the plots are from the focus shifting further and further right of the geometric center the greater the value of e. this is because rp gets closer and ra gets further away as the eccentricity of the orbit increases. Moreover, because of this shift rp approaches 0 and ra approaches infinity! This is interesting because it somewhat captures the shift from an orbit being elliptical vs parabolic or hyperbolic. The closer e gets to 1 the closer rp will be to the focus and the futher ra will be away from the focus.

(d) Referencing Curtis Eqn 2.81 the vis-viva equation, we find that $v = \sqrt{\mu\left(\frac{2}{r} - \frac{1}{a}\right)}$. Moreover, the equation for the velocity of a circular orbit is $v_{circ} = \sqrt{\frac{\mu}{a}}$. Plugging in the equations, $rp = a(1-e)$ and $ra = a(1+e)$ for r in the expression $\frac{v_n}{v_{circ}}$, We can resolve to find that

$$\frac{v_p}{v_1} = \sqrt{\frac{1+e}{1-e}} \quad \& \quad \frac{v_a}{v_1} = \sqrt{\frac{1-e}{1+e}}$$

Plotting this in python for the given e values...



Here is my commented code for creating this plot

```
#problem statement
r = 1
mu = 1
v_1 = math.sqrt(mu/r)

# returns a velocity ratio v_p/v_1 as a function of e
def velo_ratio_peri(e):
    return math.sqrt((1+e)/(1-e)) #resolved vis-viva of vp/v1

# returns a velocity ratio v_a/v_1 as a function of e
def velo_ratio_apo(e):
    return math.sqrt((1-e)/(1+e)) #resolved vis-viva of va/v1

#init lists
vp_over_v1 = []
va_over_v1 = []
e_vals = []

fig, ax = plt.subplots()

#create lists of points
for e in np.arange(0, 0.90001, .125): #includes 0.9
    e_vals.append(e)
    vp_over_v1.append(velo_ratio_peri(e))
    va_over_v1.append(velo_ratio_apo(e))
```

```
plt.plot(e_vals, vp_over_v1, 'bo', label='Vp/V1')
plt.plot(e_vals, va_over_v1, 'ro', label='Va/V1')
plt.legend(loc='upper left')
plt.xlabel('Eccentricity e')
plt.ylabel('Velocity ratio')
plt.title('Vp/V1 & Va/V1 as a function of e')
plt.grid(True)
plt.show()
```

(e) Consulting the comments I left above for part b and c. As e approaches 1, rp approaches 0 and ra approaches infinity. The inverse is true when looking the above graph. As e approaches 1, vp increases exponentially, and va decreases significantly. Thinking about this in terms of conservation of energy, where $\epsilon = \frac{v^2}{2} - \frac{\mu}{r}$ is the same for all orbits. The closer rp is to the focal point, the greater magnitude of the potential energy term in the energy equation, therefore the velocity must increase relative to the distance in order to maintain convseration of energy. Inversely looking at apoapsis, the potential energy term is very small because the distance is very large, therefore the velocity must also be very small in order to maintain conservation of energy. To summarize, the velocity is inversely related to the position of an orbit.

Problem 1.2 (Escaping Earth!)**.**

(a) The velocity at any point in orbit for a circle is $v_{circ} = \sqrt{\frac{\mu}{r}}$, the escape velocity for any given orbit is $v_{escape} = \sqrt{\frac{2\mu}{r_p}}$. Considering that $r_p = r$ for a circular orbit...

$$v_{escape} - v_{circ} = \sqrt{\frac{2\mu}{r_p}} - \sqrt{\frac{\mu}{r}} = \sqrt{\frac{\mu}{r}}(\sqrt{2} - 1)$$

Plugging this into python...

```
mu = planetary_data.MU_EARTH

#returns the needed change in velocity to escape orbit
#takes in r as an orbital radius
def needed_change_in_velo(r):
return math.sqrt(mu/r)*(math.sqrt(2)-1)
```

Planetary data is a file I wrote that contains data from Curtis Appendix A.

(b) Using the above function, I generated these values for the needed velocity Δv.

Orbit	Δv $\left[\frac{Km}{s}\right]$
LEO	3.1653
MEO	1.5472
GEO	1.2751

(c) Taking the speed of light to be $c = 2.998 * 10^8 \left[\frac{m}{s}\right]$ we can solve for the required radius to be...

$$v_{escape} = c = \sqrt{\frac{2\mu}{r}} \longrightarrow r = \frac{2\mu}{c^2} = 0.0088723\ m = 8.8723\ mm$$

(d) Because we start from a circular orbit, we know that we start our new orbit at periapsis after moving to the new velocity.

$$h = v_{new} r_p$$

$$r_p = \frac{h^2}{\mu}\left(\frac{1}{1+e}\right) \longrightarrow e = \frac{h^2}{\mu r_p} - 1$$

$$a = \frac{r_p}{1-e}$$

Then throwing this into python...

```
mu = planetary_data.MU_EARTH

v_new = math.sqrt(mu/orbital_radi['LEO']) + math.sqrt(mu/orbital_radi['LEO']) * (math.sqrt
r = orbital_radi['LEO']
h = v_new * r
e = ((h**2)/(mu*r)) - 1
a = r/(1-e)

state = orbital_equations_of_motion.orbital_state(a, e, mu)

orbital_equations_of_motion.print_state(state)

RAW OUTPUT: #Consider Sig figs when using these values.

    #This is the resulting orbit of the satellite after accelerating to the new velocity

    r_p (m) : 6828000.0
    r_a (m) : 18326117.84
    v_p (m/s) : 9224.32
    v_a (m/s) : 3436.83
    b (m) : 11186184.9
    h (m^2/s) : 62983663312.0
    T (s) : 14035.02
    spec_e (J/kg) : -15851189.24
```

Problem 1.3 (2025 TF)**.**

(a) From the problem statement, we are given alt = 428km and Vp = 20.88 km/s. Resolving Keplers 2nd law of eccentricity at periapsis...

$$r_p = \frac{h^2}{\mu}\left(\frac{1}{1+e}\right) \longrightarrow e = \frac{h^2}{\mu r_p} - 1$$

Writing this into python...

```
mu = planetary_data.MU_EARTH
rp = orbital_equations_of_motion.altitude_to_orbital_radius_earth(428000)
vp = 20880 # m/s
h = rp * vp
e = ((h**2)/(rp * mu)) - 1
print(f'e = {round(e,3)}')

RAW OUTPUT:

    e = 6.442
```

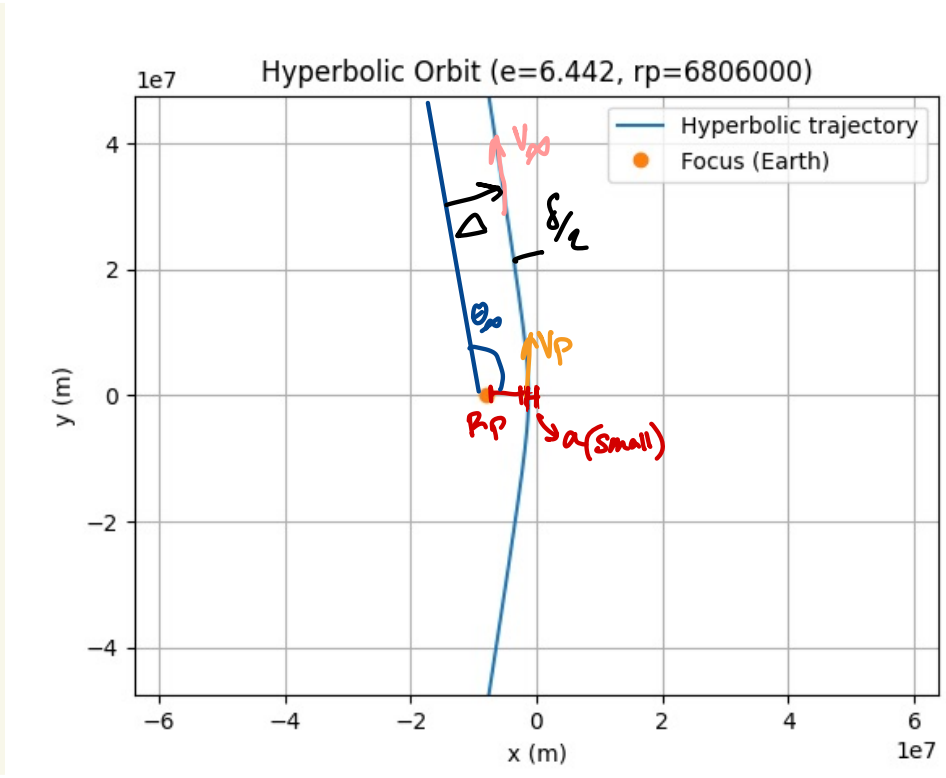
Because $e > 1$ this is a hyperbolic orbit.

(b) Here is my cited function code

```
#expects si units (rp in m)
#From lecture slide hyperbolic trajectories summary of equations
#which cites curtis equations
def hyperbolic_state(e, rp, mu):

  a = rp/(e-1)
  ra = -a*(e+1)
  delta = 2 * math.asin(1/e) * (180/math.pi) #degrees
  b = a * math.sqrt((e**2 - 1))
  theta_inf = math.acos(-1/e) * (180/math.pi) #degrees
  h = math.sqrt(mu * a * (e**2 - 1))
  beta = 180 - theta_inf
  v_inf = math.sqrt(mu/a)
  v_esc = math.sqrt(2*mu/rp)
  v_p = math.sqrt((v_esc**2) + (v_inf**2))
  epsilon = mu/(2*a)

  state = {
    'e'               : e,
    'rp (m)'          : rp,
    'ra (m)'          : ra,
    'a (m)'           : a,
    'b (m)'           : b,
    'h (m^2/s)'       : h,
    'theta inf (deg)' : theta_inf,
    'Beta (deg)'      : beta,
    'delta (deg)'     : delta,
    'V_inf (m/s)'     : v_inf,
    'V_esc (m/s)'     : v_esc,
    'V_p (m/s)'     : v_p,
    'epsilon (J/kg)'  : epsilon
  }

  return state
```

Using the summary information from the lecture on hyperbolic trajectories, here is the state my hyperbolic state function outputted for this situation

```
e : 6.44
rp (m) : 6806000
ra (m) : -9307346.74
a (m) : 1250673.37
b (m) : 7959007.59
h (m^2/s) : 142109280000.0
theta inf (deg) : 98.93
Beta (deg) : 81.07
delta (deg) : 17.86
V_inf (m/s) : 17855.15
V_esc (m/s) : 10824.42
V_p   (m/s) : 20880.0
epsilon (J/kg) : 159403203.23
```



(c)

(d) The true anomaly formula can be rewritten to find the eccentricity of the new orbit. Given $\theta_{\text{inf}} = 138°$

$$\theta_\infty = \arccos\left(\frac{-1}{e}\right) \longrightarrow e = \frac{-1}{cos(\theta_\infty)} = 1.346$$

Because r_p is the same, this can be thrown into the previously used hyperbolic orbit solving

function. This results in

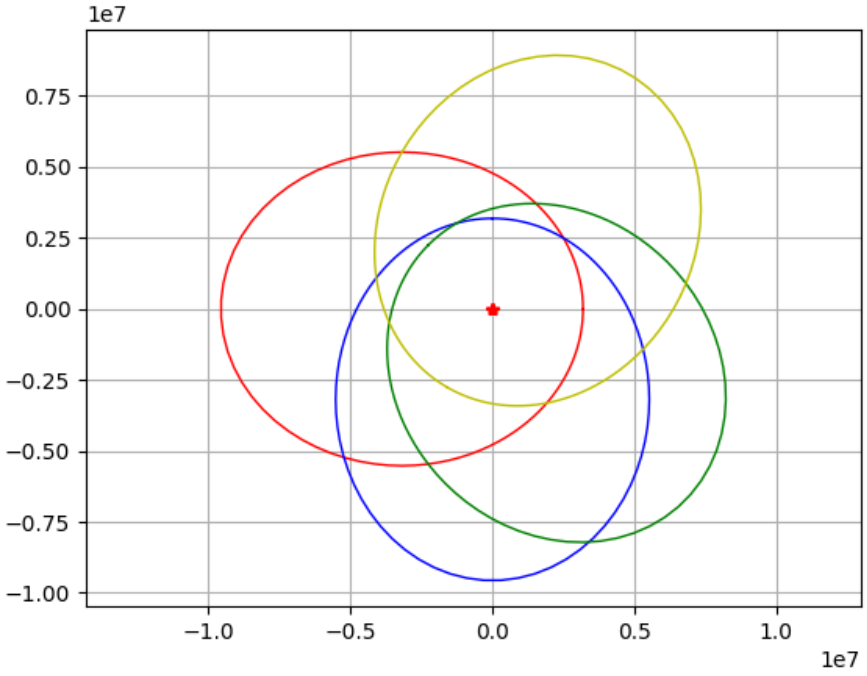
$$v_p = 11.723 \left[\frac{km}{s}\right]$$

Intuitively it makes sense for the velocity to be slower in order for the object to crash into the moon because at this slow velocity the object will have a greater shift in trajectory becuause it will be under the strongest force of gravity for longer. Therefore it will curve more as a trajectory and will hit the moon. The closer to 90 degrees the asteroid comes in at, the less time the asteroid will spend in the strongest part of earths gravity. The reverse is also true which is what we see here.

(e) Assuming its infinitely far away, the astroid will make contact with the moon at speed v_{inf}. Using the same state and the function above, $v_{inf} = 4.502 \left[\frac{Km}{s}\right]$

Problem 1.4. (Perifocal Frames)

Here is my code to solve this problem. For the plot, Red is England/Africa, Blue is India, Green is Australia, and Seattle is yellow.




```
def perifocal_plot(a, e, theta_deg, ax=None, color='b'):
    if ax is None: #create plot if no plot parameter was given
        fig, ax = plt.subplots()

    theta_rad = np.deg2rad(theta_deg)
    p_vec = np.array([np.cos(theta_rad), np.sin(theta_rad)]) #unit vector p
    q_vec = np.array([np.cos(theta_rad + np.pi/2), np.sin(theta_rad + np.pi/2)]) #unit vec

    #create data points
```

```
    theta = np.linspace(0, 2*np.pi, 100)
    r = a * (1 - e**2) / (1 + e * np.cos(theta))
    x = r * (p_vec[0]*np.cos(theta) + q_vec[0]*np.sin(theta))
    y = r * (p_vec[1]*np.cos(theta) + q_vec[1]*np.sin(theta))

    #Plot
    ax.plot(x, y, color=color)
    ax.plot(0, 0, 'r*') #set focus to be 0,0
    ax.set_aspect('equal', adjustable='datalim')
    ax.grid(True)

    return ax

""" Part 4 """

angles = [0, 90, 135, -120] #deg
colors = ['r', 'b', 'g', 'y']

a = planetary_data.RADIUS_EARTH #m

e = 0.5 #statement

fig, ax = plt.subplots()

for i, angle in enumerate(angles):
orbital_equations_of_motion.perifocal_plot(a,e,angle,ax, color=colors[i])

plt.show()
```