Astrodynamics Software and Science Enabling Toolkit (ASSET) Training

NESC Training – Flight Mechanics Tech. Discipline Team

Astrodynamics and Space Research Laboratory

Presenter: Aaron Houin
PI: Dr. Rohan Sood

The University of Alabama, Tuscaloosa AL

Training Overview

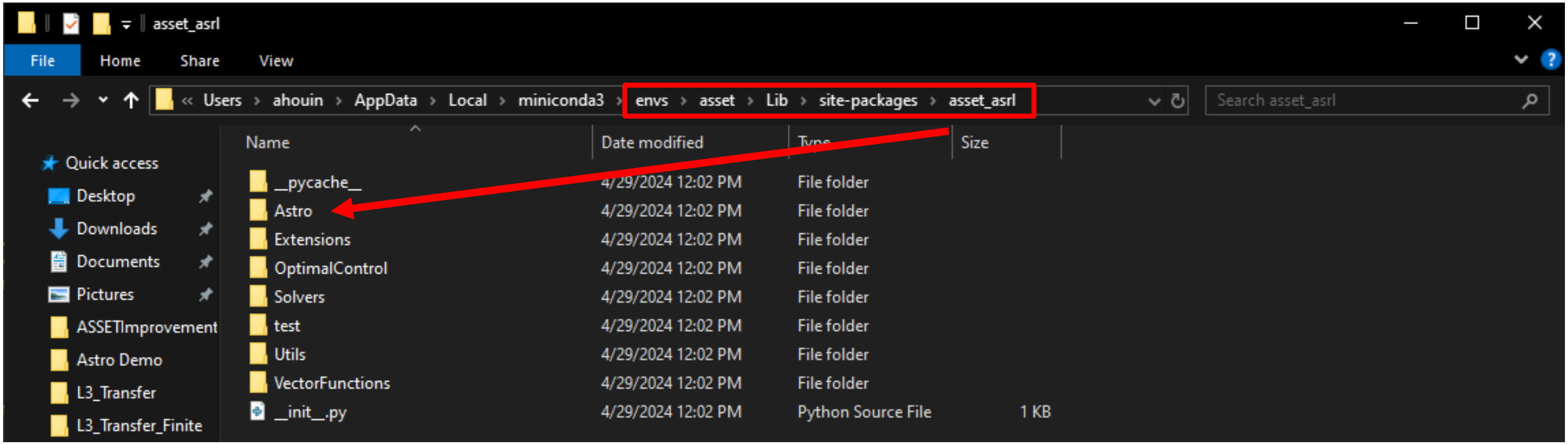
- Day 1: Introduction to ASSET's core functionality
  - Vector Functions - "Basic building blocks used in nearly all ASSET operations"
  - ODEs and Integrators - "Defining solution spaces and integrating the dynamics"
  - Phases - "Setting up optimization problems"
  - Optimal Control Problems (OCPs) - "Configuring complex, multi-phase optimization problems"

- Day 2: Using the "Astro Library" for quick astrodynamics modeling
  - Two-Body Problem Example
  - N-Body Problem Example
  - CR3BP Example
  - Ephemeris Pulsing Rotating Example

Astro Library

Where is Astro?

- Astro comes bundles with the ASSET install
  - Can be found in “Site-Packages” of wherever your ASSET virtual environment is



What is the Astro Library?

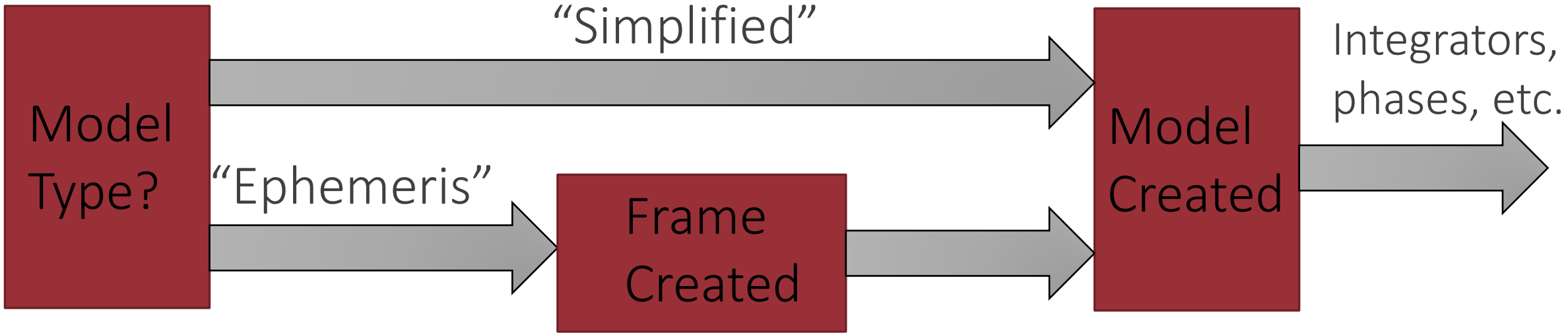
- While the core of ASSET is the underlying C++ tools and their python bindings, ASSET ships with additional functions for user convenience

- Astro is useful to quickly get models up and running, using prebuild astrodynamics frames written in Vector Function syntax

- Existing tools currently include:
  - Astrodynamical constants (gravitational parameters, units, etc.)
  - Simplified models (two-body, CR3BP, etc.)
  - Ephemeris models (N-body, Ephemeris Rotating, etc.)
  - Engines/Accelerators (low-thrust engine, solar sail, etc.)
  - Data handlers (spice wrapper, date conversion, etc.)
  - More to come!

Astro Model Types

- In general, models in Astro can be defined as either “simplified” or “ephemeris”
  - While “simplified” models can stand along, “ephemeris” models need a “frame” defined
  - “Frames” use JPL SPICE kernels on the backend to handle planetary data



Model Instantiation

- To conduct proper numerical optimization, all Astro models are nondimensionalized

- Astro models are instantiated with dimensional terms, and the nondimensionalization occurs behind the scenes
  - This is handled via “characteristic” terms for each unit type
  - An instantiated model class has lstar, vstar, and astar defined as properties (length, velocity, accel.)
  - To nondimensionalize a term, divide it by the appropriate “characteristic” value (i.e., 100 meters / lstar)
  - Redimensionalization is as simple as multiplying an ND term by the “characteristic” value

- Using the Astro Constants library helps ensure all models and dimensionalization terms use the same underlying units
  - All units in Astro.Constants are given in meters, kg, and seconds
  - For example, Astro.Constants.day = 86400.0 (seconds)

Astro Utility Functions

- On top of the models and frames provided in Astro, there are a wide range of utility functions to make analysis easier and more consistent

- These utilities include:
  - Constants
  - SPICE data management
  - Date conversion tools
  - Plotting functions

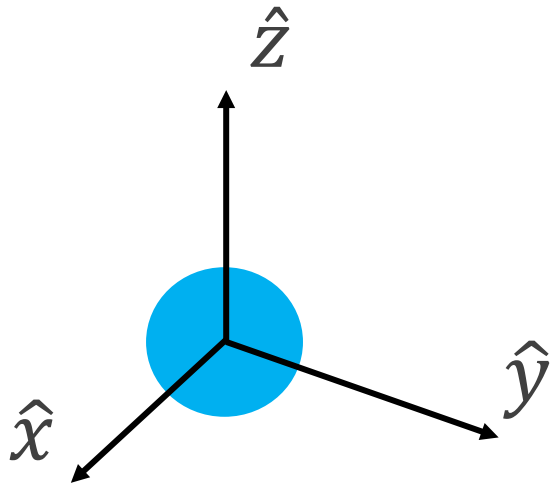
- Many of these will be demonstrated in the following examples

Astro Demo 1: Two-Body Model

Model Definition

- The simplest example of orbital mechanics is the two-body equations of motion
  - Consists of an arbitrary cartesian set of right-handed basis vectors
  - Utilizes point mass acceleration, with the attractive body at the origin

- Model doesn't require a frame, only needs gravitational parameter and lstar



Demo Overview

- This first demo will show how to instantiate and use the TwoBody model in Astro, along with how to model and optimize impulsive delta Vs

- we will use a Moon centric model, with a predefined initial condition of a suborbital trajectory

- A delta V will be applied and optimized to attain a desired semi-major axis value as part of a guided example

- Finally, you will have a chance to try and optimize a circularizing burn on your own!

Demo Imports

```
import asset_asrl as ast
from asset_asrl.VectorFunctions import Arguments as Args
import asset_asrl.Astro.Constants as c
import asset_asrl.VectorFunctions as vf
import asset_asrl.OptimalControl as oc
from asset_asrl.Astro.AstroModels import TwoBody

import json
import pickle as pkl

import numpy as np
import matplotlib.pyplot as plt
```

Model Setup

```
14    #set up constants
15    r_moon = c.RadiusMoon
16
17    #set up model
18
19    twobody = TwoBody(c.MuMoon,r_moon)
20    twobody_integ = twobody.integrator(200/twobody.tstar)
21
22    #Nondim Funcs
23    M = c.meter/twobody.lstar
24    KM = c.kilometer/twobody.lstar
25    Day = c.day/twobody.tstar
26    Hour = c.hour/twobody.tstar
27    MS = (c.meter/c.sec)/twobody.vstar
28
```

Cartesian to Keplerian Elements Vector Function

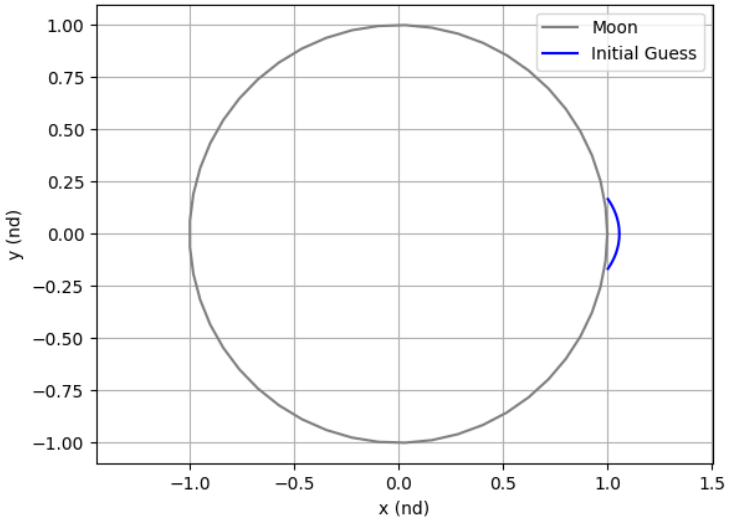
- Function definition provided in the “UserProvidedInputs.txt” file
- This definition is only meant to handle elliptical orbit states!

```
30  #Make RV2Kep
31
32  def Cart2Kep(mu,lstar,vstar):
33      '''expects nondim pos and vel in elliptical orbit, returns kep elements'''
34      args = Args(6)
35
36      r = args.head(3) * lstar
37      v = args.tail(3) * vstar
38
39      h = r.cross(v) #angular momentum
40      n = vf.cross(np.array([0,0,1]),h)
41      evect = ((v.norm()**2 - (mu/r.norm()))*r - r.dot(v)*v)/mu
42      e = evect.norm()
43
44      mechenergy = (v.norm()**2)/2 - (mu/r.norm())
45
46      a = -mu/(2*mechenergy)
47      p = a*(1-e**2)
48
49      i = vf.arccos(h[2]/h.norm()) #inclination in rad
50      raan = vf.arccos(n[0]/n.norm()) #right ascention of ascending node, in rad
51
52      w = vf.arccos(n.dot(evect)/(n.norm()*evect.norm())) #argument of periapsis, in rad
53
54      ta = vf.arccos(evect.dot(r)/(evect.norm()*r.norm()))
55
56      return vf.stack(a,e,i,raan,w,ta)
57
```

Set Up Initial Guess (Suborbital)

```
69   #Set an initial condition 1
70   theta = np.linspace(0,2*np.pi)
71   moon_x = r_moon*M*np.cos(theta)
72   moon_y = r_moon*M*np.sin(theta)
73
74   ic1 = [ 1.00280458, -0.16691993, 0.0001, 0.31806197, 0.46184659, 0., 0.]
75   ig1 = twobody_integ.integrate_dense(ic1, ic1[6] + .2*Hour, 500)
76   ig1 = np.array(ig1)
77
78   plt.figure()
79   plt.plot(moon_x,moon_y,c='grey')
80   plt.plot(ig1[:,0],ig1[:,1],c='blue')
81   plt.axis('equal')
82   plt.grid(True)
83   plt.legend(['Moon', 'Initial Guess'])
84   plt.xlabel('x (nd)')
85   plt.ylabel('y (nd)')
86   plt.show()
```

- Initial Condition provided in "UserProvidedInputs.txt"



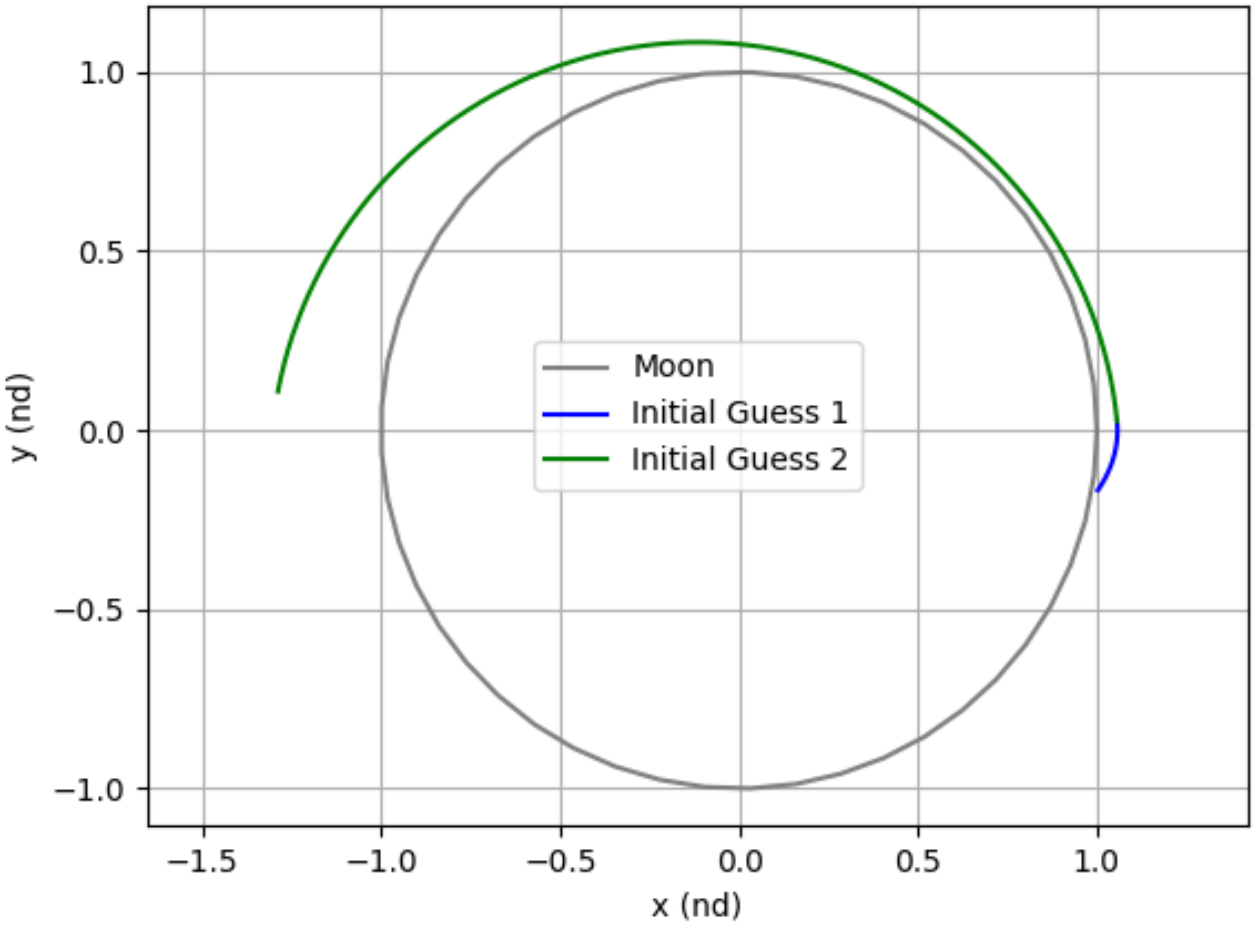
Set Up Event Detection Integration

```
88  #Event Detection
89  def ApsEventFunc():
90      args = Args(7)
91      r = args.head(3)
92      v = args.segment3(3)
93
94      return r.dot(v)
95
96  direction = 0
97  stopcode = True
98
99  apsevent = (ApsEventFunc(),direction,stopcode)
100 twobody_integ.EventTol = 1e-13
101
102 ig1_apo, eventlocs = twobody_integ.integrate_dense(ic1, ic1[6] + .2*Hour, 1000, [apsevent])
103 ig1_apo = np.array(ig1_apo)
104
```

Set Up First Burn Initial Guess

```
112   #Set up ig 2 (first burn)
113
114   ic2 = np.copy(ig1_apo[-1])
115   dv = 900*MS
116   uv = np.copy(ic2[3:6])/np.linalg.norm(ic2[3:6])
117   ic2[3:6] = ic2[3:6] + dv*uv
118
119   ig2 = twobody_integ.integrate_dense(ic2,ic2[6]+1*Hour,1000)
120   ig2 = np.array(ig2)
121
122   a,e,i,raan,w,ta = Cart2Kep(c.MuMoon,twobody.lstar,twobody.vstar).compute(ic2[0:6])
123   print(f'a = {a}\ne = {e}\ni = {np.rad2deg(i)}\nraan = {np.rad2deg(raan)}\nw = {np.rad2deg(w)}\nta = {np.rad2deg(ta)}\n')
124
125   plt.figure()
126   plt.plot(moon_x,moon_y,c='grey')
127   plt.plot(ig1_apo[:,0],ig1_apo[:,1],c='blue')
128   plt.plot(ig2[:,0],ig2[:,1],c='green')
129   plt.axis('equal')
130   plt.grid(True)
131   plt.legend(['Moon', 'Initial Guess 1', 'Initial Guess 2'])
132   plt.xlabel('x (nd)')
133   plt.ylabel('y (nd)')
134   plt.show()
```

Initial Guess 2 Should Provide:




```
a = 2066120.576974816
e = 0.1331606232096741 7
i = 0.006223915722477822
raan = 124.55419357103575
w = 163.54128440805005
ta = 37.5681591524984
```

Define Semi-Major Axis Constraint

```
137   #Write Kep Constraint
138
139   def SemiMajorConstraint(a_con,mu,lstar,vstar):
140       args = Args(6)
141       # r = args.head(3)
142       # v = args.tail(3)
143       states = args.head(6)
144
145       Kep = Cart2Kep(mu,lstar,vstar).eval(states)
146
147       return (a_con - Kep[0])/lstar
```

Configure Phases and OCP

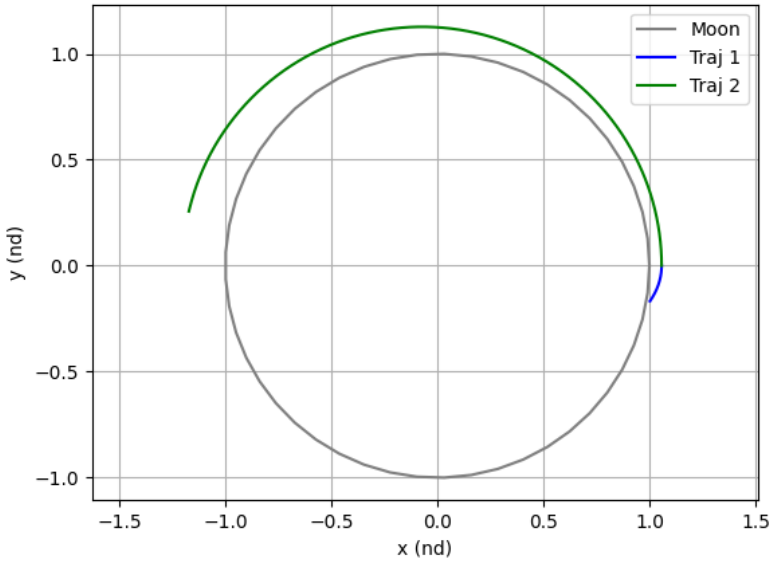
```
161  #OCP
162
163  def DVCost():
164      args = Args(6)
165      v1 = args.head(3)
166      v2 = args.tail(3)
167
168      return (v1-v2).norm()
169
170  def MoonDist(r_moon):
171      args = Args(3)
172      r = args.head(3)
173
174      return r_moon - r.norm()
175
176
177  phase1 = twobody.phase('LGL3')
178  phase1.setTraj(ig1_apo,1000)
179  phase1.addBoundaryValue('Front',[0,1,2,3,4,5,6],ic1)
180  phase1.addLowerDeltaTimeBound(.001*Hour)
181  phase1.addUpperDeltaTimeBound(2*Hour)
182
183  phase2 = twobody.phase('LGL3')
184  phase2.setTraj(ig2,1000)
185  phase2.addEqualCon('Front',SemiMajorConstraint(((r_moon+350000) + (r_moon+100000))/2,c.MuMoon,twobody.lstar,twobody.vstar),[0,1,2,3,4,5])
186  phase2.addInequalCon('Path',MoonDist((r_moon+100000)*M),[0,1,2])
187  phase2.addLowerDeltaTimeBound(.01*Hour)
188  phase2.addUpperDeltaTimeBound(1*Hour)
```

Used later for link constraints

Configure Phases and OCP (cont.)

```
190    ocp = oc.OptimalControlProblem()
191    # ocp.optimizer.set_SoeLSMode("L1")
192    # ocp.optimizer.set_OptLSMode("L1")
193    ocp.addPhase(phase1)
194    ocp.addPhase(phase2)
195
196    ocp.addForwardLinkEqualCon(phase1,phase2,[0,1,2,6]) #not vel
197    ocp.addLinkObjective(DVCost(),
198                         phase1, 'Back', [3,4,5],
199                         phase2, 'Front', [3,4,5])
200
201    ocp.addLinkInequalCon(1e-6*MS - DVCost(), #set min to be >0
202                          phase1, 'Back', [3,4,5],
203                          phase2, 'Front', [3,4,5])
204
205    ocp.solve_optimize()
206
207    traj1 = np.array(phase1.returnTraj())
208    traj2 = np.array(phase2.returnTraj())
```

```
==============================
DV1 = 864.7261195653463 m/s
```



Your Turn!

Next Challenge:

- Using what is written so far in demo 1, let's add another phase that performs an optimized circularization burn

- This new phase can be added to the end of the orbit raise burn, and both burns can be optimized together in a single ocp

A Few Hints:

- Your new trajectory arc will need the following:
  - An initial condition (recall how we added a delta V guess for ig2)
  - An initial guess -> integrate the initial condition
  - A “phase” added to the optimal control problem, along with proper constraints
  - Speaking of constraints, instead of the sma constraint you will need to define and use an eccentricity constraint

- The ocp has already solved 2 of the 3 trajectories in it, so you can use ocp.optimize() instead of solve_optimize()

- To avoid numerical issues, **constrain final eccentricity to be 0.0001, not 0!**

- I integrated/constrained the new phase to be 3 hours to show a full orbit revolution, but this is arbitrary

Exercise 1 Answer Key

Define Eccentricity Constraint

```
137  #Write Kep Constraint
138
139  def SemiMajorConstraint(a_con,mu,lstar,vstar):
140      args = Args(6)
141      # r = args.head(3)
142      # v = args.tail(3)
143      states = args.head(6)
144
145      Kep = Cart2Kep(mu,lstar,vstar).eval(states)
146
147      return (a_con - Kep[0])/lstar
148
149  def EccConstraint(e_con,mu,lstar,vstar):
150      args = Args(6)
151      # r = args.head(3)
152      # v = args.tail(3)
153      states = args.head(6)
154
155      Kep = Cart2Kep(mu,lstar,vstar).eval(states)
156
157      return (e_con - Kep[1])
158
```

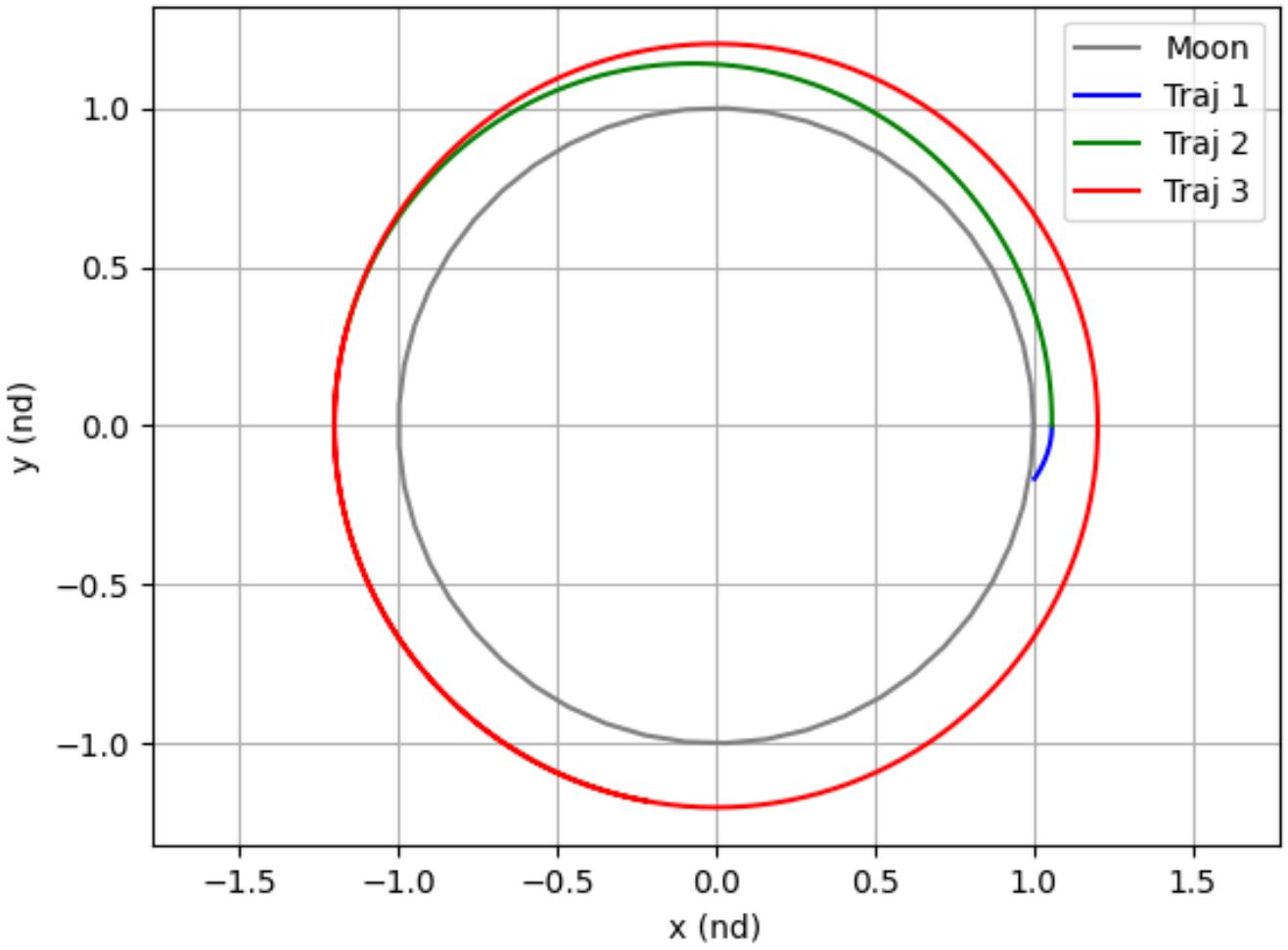
Initial Guess and Phase Creation

```
232    #Set up ig 3 (2nd burn)
233
234    ic3 = np.copy(traj2[-1])
235    dv = 150*MS
236    uv = np.copy(ic3[3:6])/np.linalg.norm(ic3[3:6])
237    ic3[3:6] = ic3[3:6] + dv*uv
238
239    ig3 = twobody_integ.integrate_dense(ic3,ic3[6]+3*Hour,1000)
240    ig3 = np.array(ig3)
241
242
243    phase3 = twobody.phase('LGL3')
244    phase3.setTraj(ig3,1000)
245    phase3.addInequalCon('Path',MoonDist((r_moon+100000)*M),[0,1,2])
246    phase3.addEqualCon('Front',EccConstraint(.0001,c.MuMoon,twobody.lstar,twobody.vstar),[0,1,2,3,4,5])
247
248    phase3.addDeltaTimeEqualCon(3*Hour)
```

Add To OCP and Optimize

```
250    ocp.addPhase(phase3)
251
252    ocp.addForwardLinkEqualCon(phase2,phase3,[0,1,2,6])
253
254    ocp.addLinkObjective(DVCost(),
255                         phase2, 'Back', [3,4,5],
256                         phase3, 'Front', [3,4,5])
257
258    ocp.addLinkInequalCon(1e-6*MS - DVCost(), #set min to be >0
259                          phase2, 'Back', [3,4,5],
260                          phase3, 'Front', [3,4,5])
261
262    ocp.optimize()
263
264    traj1 = np.copy(phase1.returnTraj())
265    traj2 = np.copy(phase2.returnTraj())
266    traj3 = np.copy(phase3.returnTraj())
```

Final Solution




```
==============================
DV1 = 864.8194149227508 m/s
DV2 = 50.67283830897026 m/s
```

Save Traj Data

```
#Save traj data in dictionary as json file for later use

data = {'traj1_x':traj1[:,0].tolist(),'traj1_y':traj1[:,1].tolist(),'traj1_z':traj1[:,2].tolist(),
        'traj1_vx':traj1[:,3].tolist(),'traj1_vy':traj1[:,4].tolist(),'traj1_vz':traj1[:,5].tolist(),'traj1_t':traj1[:,6].tolist(),
        'traj2_x':traj2[:,0].tolist(),'traj2_y':traj2[:,1].tolist(),'traj2_z':traj2[:,2].tolist(),
        'traj2_vx':traj2[:,3].tolist(),'traj2_vy':traj2[:,4].tolist(),'traj2_vz':traj2[:,5].tolist(),'traj2_t':traj2[:,6].tolist(),
        'traj3_x':traj3[:,0].tolist(),'traj3_y':traj3[:,1].tolist(),'traj3_z':traj3[:,2].tolist(),
        'traj3_vx':traj3[:,3].tolist(),'traj3_vy':traj3[:,4].tolist(),'traj3_vz':traj3[:,5].tolist(),'traj3_t':traj3[:,6].tolist(),
        }

with open('./TwoBody_Data.json','w') as file:
    json.dump(data,file)


#Alternatively, save pickle file for later use

data2 = {'traj1':traj1,'traj2':traj2,'traj3':traj3}

with open('TwoBody_Data.pkl', 'wb') as file:
        pkl.dump(data2, file)
```

Astro Demo 2: N-Body Frame

New Imports

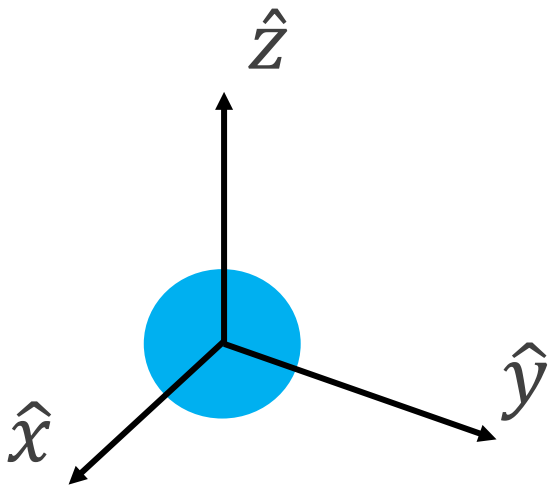
- Provide additional imports as needed for N-Body Modeling

```
import asset_asrl as ast
from asset_asrl.VectorFunctions import Arguments as Args
import asset_asrl.Astro.Constants as c
import asset_asrl.VectorFunctions as vf
import asset_asrl.OptimalControl as oc
from asset_asrl.Astro.AstroModels import NBody
from asset_asrl.Astro.AstroFrames import NBodyFrame
from asset_asrl.Astro.Date import datetime, datetime_to_jd

import json
import pickle as pkl
import spiceypy as spice

import numpy as np
import matplotlib.pyplot as plt
```

Model Definition

- The N-Body frame is a planet centric ephemeris model
  - Leverages JPL SPICE kernels for ephemerides and frame definition
  - Defaults to point mass acceleration, with J2 gravity as an option

- This Model requires a Frame



Model: an instantiated ASSET ODE that exposes ASSET Integrators and Phases
(The last example used the two-body model)

Frame: a utility class that contains important methods and properties that a Model can use
(The N-Body Frame holds the SPICE ephemerides)

Set Constants and Furnish the SPICE Kernels

```
17    #set up constants
18    r_moon = c.RadiusMoon
19    w_moon = 2*c.pi/(28*c.day)
20
21    spice.furnsh('./kernels/BasicKernel.txt')
22
```

*The “kernels” folder provided needs to be added to your working directory

Define the N-Body Frame

```
24    #set up Frame and Model
25    init_datetime = datetime(2015, 3, 14, 9, 26)
26    JD0 = datetime_to_jd(init_datetime)  #March 14, 2015, 9:26 - 3/14/15/9:26
27    JDF = JD0 + 2
28
29    frame = NBodyFrame('Moon',c.MuMoon,r_moon,JD0,JDF,3000,'J2000')
30    # frame.Add_P1_J2Effect() #J2 gravity
31    frame.AddSpiceBodies(['EARTH','SUN','JUPITER BARYCENTER'])
32
33    nbody = NBody(frame,ActiveAltBodies='All',Enable_J2=True)
34    nbody_integ = nbody.integrator(200/nbody.tstar)
35
36    #Nondim Funcs
37    M = c.meter/nbody.lstar
38    KM = c.kilometer/nbody.lstar
39    Day = c.day/nbody.tstar
40    Hour = c.hour/nbody.tstar
41    MS = (c.meter/c.sec)/nbody.vstar
```

Reuse Functions from Demo 1

- In this new script, copy over the utility vector functions we wrote for demo 1

- This includes:
  - Cart2Kep
  - SemiMajorConstraint
  - EccConstraint
  - DVCost
  - MoonDist

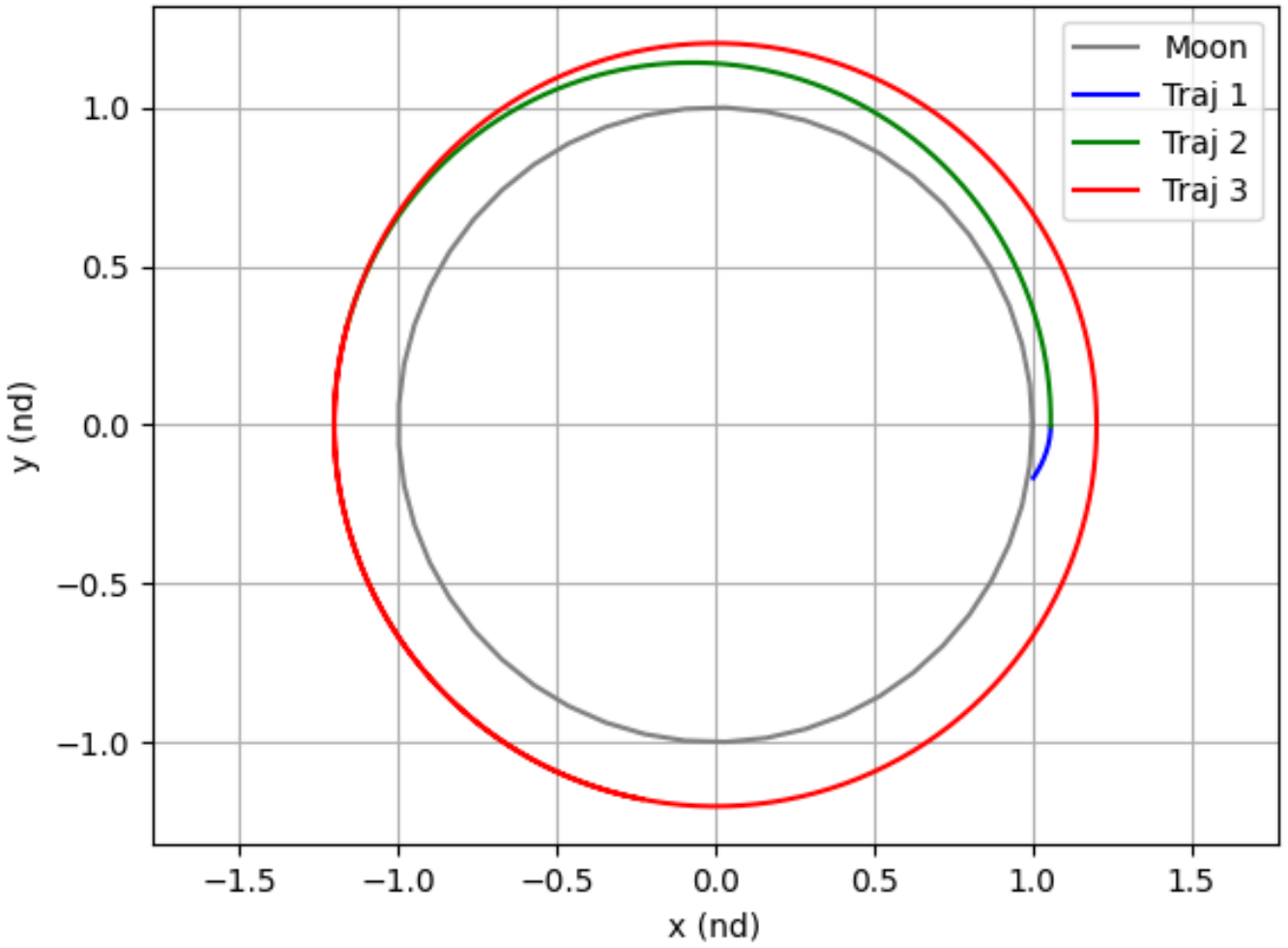
- Functions you use often can of course be saved into a utility function script or python module to be imported into projects

Next Challenge

- Now try to create the same 3 phase optimal control problem from Demo 1, this time using the N-Body Model we just defined

- Hints:
  - Remember that an ASSET "phase" is spawned from an ASSET Model

Final Solution

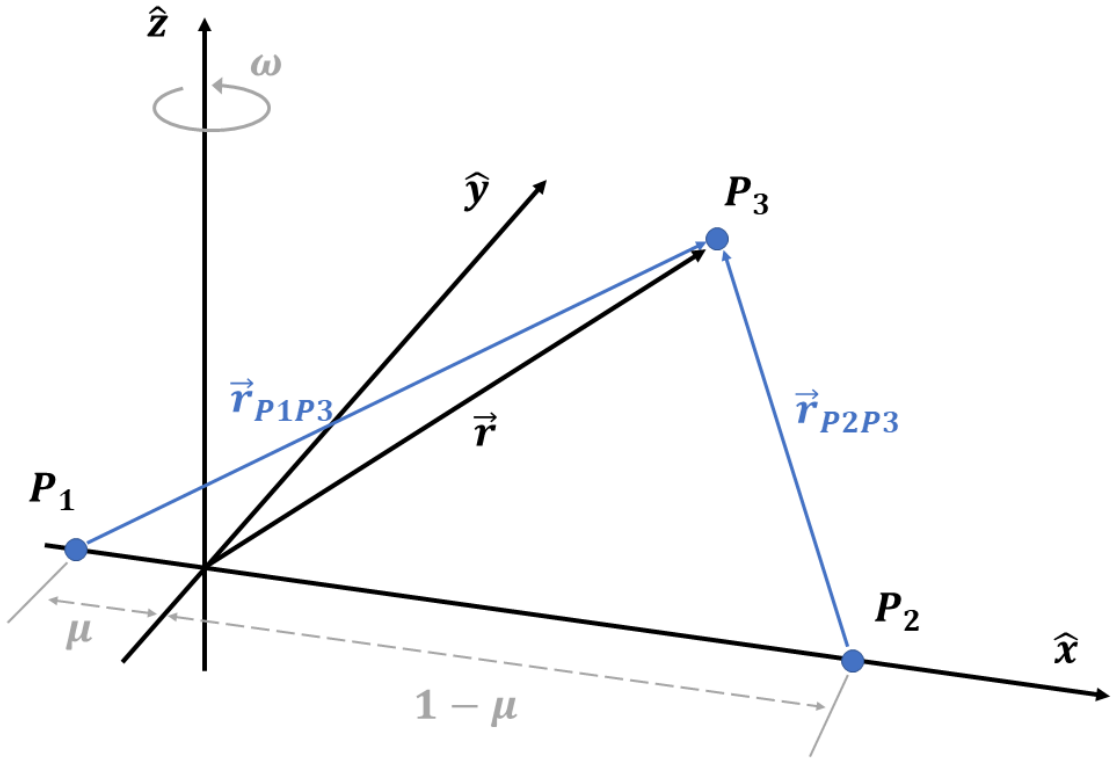



```
==========================================
DV1 = 864.6770049251543 m/s
DV2 = 50.741566572898634 m/s
```

Astro Demo 3: CR3BP Frame

Model Definition

- The Circular Restricted 3 Body Problem (CR3BP) has already been addressed in Day 1
- In the Astro Library, CR3BP is a simplified Model that doesn't require a Frame



Imports

```
1  import asset_asrl as ast
2  import asset_asrl.Astro.Constants as c
3  from asset_asrl.Astro.AstroModels import CR3BP, CR3BP_LT
4  from asset_asrl.Astro.Extensions.ThrusterModels import LowThrustAcc
5
6  import numpy as np
7  import matplotlib.pyplot as plt
8  from mpl_toolkits import mplot3d
9  import pickle as pkl
10
11 vf = ast.VectorFunctions
12 Args = vf.Arguments
```

Configure CR3BP Model and Define Constants

```
14    #Set up models
15
16    cr3bp = CR3BP(c.MuEarth,c.MuMoon,c.LD)
17    cr3bp_integ = cr3bp.integrator(200/cr3bp.tstar)
18
19
20    M = c.meter/cr3bp.lstar
21    KM = c.kilometer/cr3bp.lstar
22    MS = (c.meter/c.sec)/cr3bp.vstar
23    Hour = c.hour/cr3bp.tstar
24    Day = c.day/cr3bp.tstar
25    MSS = (c.meter/c.sec**2)/cr3bp.astar
```

Create a Low-Thrust Engine and Low-Thrust CR3BP Model

- The LowThrustAcc model from Astro.Extensions.ThrusterModels accepts either a nondimensional or a dimensional acceleration value as input
  - The other input should be set as “False”
  - The model instantiation is of the form “LowThrustAcc(<Nondimensional Acc>, <Dimensional Acc>)”
  - We use a dimensional 0.005 (m/s^2) for this example
- The “cr3bp_lt” Model is entirely independent of the base “cr3bp” Model already instantiated

```
27    lt_engine = LowThrustAcc(False,0.005)
28
29    cr3bp_lt = CR3BP_LT(c.MuEarth,c.MuMoon,c.LD,lt_engine)
```

Make a Prograde Control Law and Low-Thrust Integrator

```
31  #define control law
32
33  def ControlLaw_Prograde():
34      args = Args(3)
35      v = args.head(3)
36
37      return v.normalized()
38
39  cr3bp_lt_integ = cr3bp_lt.integrator(200/cr3bp.tstar,ControlLaw_Prograde(),[3,4,5])
```

Generate Initial Guesses

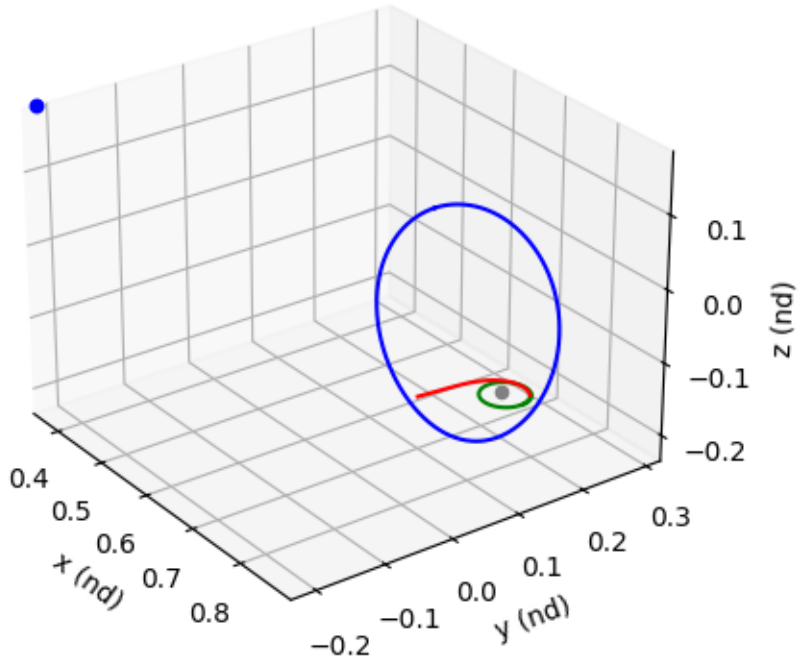
```
43   #Initial Guess Generation
44
45   halo_ic = np.array([8.6012940776697100E-1, 1.2156094011891834E-27, 1.8415292733184649E-1,
46   halo_dt = 2.3976941611197926
47
48   halo_ig = cr3bp_integ.integrate_dense(halo_ic, halo_ic[6]+halo_dt, 1000)
49   halo_ig = np.array(halo_ig)
50
51   lunar_ic = np.array([cr3bp.P2[0]+15000*KM, 0, 0, 0, 450*MS, 0, 0])
52   lunar_ig = cr3bp_integ.integrate_dense(lunar_ic, lunar_ic[6]+40*Hour, 1000)
53   lunar_ig = np.array(lunar_ig)
54
```

*The initial condition values are included in “ASSET_Training_Files/Astro Library (Day 2)/UserProvidedInputs.txt” Starting on line 72

Create Low-Thrust Arc Initial Guess and Plot

```
56   #Burn IG
57
58   burn_ic = np.concatenate([np.copy(lunar_ig[-1]), [1,0,0]])
59   burn_ig = cr3bp_lt_integ.integrate_dense(burn_ic, burn_ic[6] + 24*Hour, 1000)
60   burn_ig = np.array(burn_ig)
61
```

```
63   plt.figure()
64   ax = plt.axes(projection='3d')
65
66   ax.scatter([cr3bp.P1[0]], [cr3bp.P1[1]],[cr3bp.P1[2]],s=22,c='blue')
67   ax.scatter([cr3bp.P2[0]], [cr3bp.P2[1]],[cr3bp.P2[2]],s=20,c='grey')
68   ax.plot3D(halo_ig[:,0],halo_ig[:,1],halo_ig[:,2],c='blue')
69   ax.plot3D(lunar_ig[:,0], lunar_ig[:,1], lunar_ig[:,2], c = 'green')
70   ax.plot3D(burn_ig[:,0], burn_ig[:,1], burn_ig[:,2], c = 'red')
71
72   plt.axis('equal')
73   plt.show()
```



Create Interpolation Table for Halo Orbit Rendezvous

```
76    #Make Halo Orbit an interp table
77
78    halo_states = halo_ig[:,0:6]
79    halo_t = halo_ig[:,6]
80
81    halo_table = vf.InterpTable1D(halo_t, halo_states, kind='cubic')
82    rend_pt = 3*halo_dt/4
83    rend_state = halo_table(rend_pt)
84
```

Make a Rendezvous Constraint Vector Function

- Python inputs include the “InterpTable1D” for the target, and ASSET Arguments for “states” (position and velocity) and a “rendezvous time” that is used for InterpTable1D lookup

```
102    #Write Rendezvous constraint
103
104    def RendCon(table):
105        args = Args(7)
106        states = args.head(6)
107        rend_time = args[6]
108
109        target_states = table(rend_time)
110
111        return target_states - states
112
```

Construct Phases

```
115   #Low Lunar Orbit Phase
116   llophase = cr3bp.phase('LGL3')
117   llophase.setTraj(lunar_ig, 1000)
118   llophase.addBoundaryValue('Front',[0,1,2,3,4,5,6], lunar_ic)
119   llophase.addLowerDeltaTimeBound(30*Hour)
120   llophase.addUpperDeltaTimeBound(70*Hour)
121
122   #Low Thrust Transfer Phase
123   ltphase = cr3bp_lt.phase('LGL5')
124   ltphase.setTraj(burn_ig,1000)
125   ltphase.addLowerDeltaTimeBound(5*Hour)
126   ltphase.addDeltaTimeObjective(1.0)
127
128   #set Rendezvous Constraint via Static Parameter
129   ltphase.setStaticParams([rend_pt])
130   ltphase.addLUVarBound('StaticParams', 0, 0.0, halo_dt)
131   ltphase.addEqualCon('Back', RendCon(halo_table), [0,1,2,3,4,5],[],[0])
132
133   #Add controller settings
134   ltphase.setControlMode('HighestOrderSpline')
135   ltphase.addLUNormBound('Path',[7,8,9], 0.01, 1.0)
```

Construct Phases

```
115   #Low Lunar Orbit Phase
116   llophase = cr3bp.phase('LGL3')
117   llophase.setTraj(lunar_ig, 1000)
118   llophase.addBoundaryValue('Front',[0,1,2,3,4,5,6], lunar_ic)
119   llophase.addLowerDeltaTimeBound(30*Hour)
120   llophase.addUpperDeltaTimeBound(70*Hour)
121
122   #Low Thrust Transfer Phase
123   ltphase = cr3bp_lt.phase('LGL5')
124   ltphase.setTraj(burn_ig,1000)
125   ltphase.addLowerDeltaTimeBound(5*Hour)
126   ltphase.addDeltaTimeObjective(1.0)
127
128   #set Rendezvous Constraint via Static Parameter
129   ltphase.setStaticParams([rend_pt])
130   ltphase.addLUVarBound('StaticParams', 0, 0.0, halo_dt)
131   ltphase.addEqualCon('Back', RendCon(halo_table), [0,1,2,3,4,5],[],[0])
132
133   #Add controller settings
134   ltphase.setControlMode('HighestOrderSpline')
135   ltphase.addLUNormBound('Path',[7,8,9], 0.01, 1.0)
```

Construct Phases

```
115   #Low Lunar Orbit Phase
116   llophase = cr3bp.phase('LGL3')
117   llophase.setTraj(lunar_ig, 1000)
118   llophase.addBoundaryValue('Front',[0,1,2,3,4,5,6], lunar_ic)
119   llophase.addLowerDeltaTimeBound(30*Hour)
120   llophase.addUpperDeltaTimeBound(70*Hour)
121
122   #Low Thrust Transfer Phase
123   ltphase = cr3bp_lt.phase('LGL5')
124   ltphase.setTraj(burn_ig,1000)
125   ltphase.addLowerDeltaTimeBound(5*Hour)
126   ltphase.addDeltaTimeObjective(1.0)
127
128   #set Rendezvous Constraint via Static Parameter
129   ltphase.setStaticParams([rend_pt])
130   ltphase.addLUVarBound('StaticParams', 0, 0.0, halo_dt)
131   ltphase.addEqualCon('Back', RendCon(halo_table), [0,1,2,3,4,5],[],[0])
132
133   #Add controller settings
134   ltphase.setControlMode('HighestOrderSpline')
135   ltphase.addLUNormBound('Path',[7,8,9], 0.01, 1.0)
```

Construct Phases

```
115   #Low Lunar Orbit Phase
116   llophase = cr3bp.phase('LGL3')
117   llophase.setTraj(lunar_ig, 1000)
118   llophase.addBoundaryValue('Front',[0,1,2,3,4,5,6], lunar_ic)
119   llophase.addLowerDeltaTimeBound(30*Hour)
120   llophase.addUpperDeltaTimeBound(70*Hour)
121
122   #Low Thrust Transfer Phase
123   ltphase = cr3bp_lt.phase('LGL5')
124   ltphase.setTraj(burn_ig,1000)
125   ltphase.addLowerDeltaTimeBound(5*Hour)
126   ltphase.addDeltaTimeObjective(1.0)
127
128   #set Rendezvous Constraint via Static Parameter
129   ltphase.setStaticParams([rend_pt])
130   ltphase.addLUVarBound('StaticParams', 0, 0.0, halo_dt)
131   ltphase.addEqualCon('Back', RendCon(halo_table), [0,1,2,3,4,5],[],[0])
132
133   #Add controller settings
134   ltphase.setControlMode('HighestOrderSpline')
135   ltphase.addLUNormBound('Path',[7,8,9], 0.01, 1.0)
```

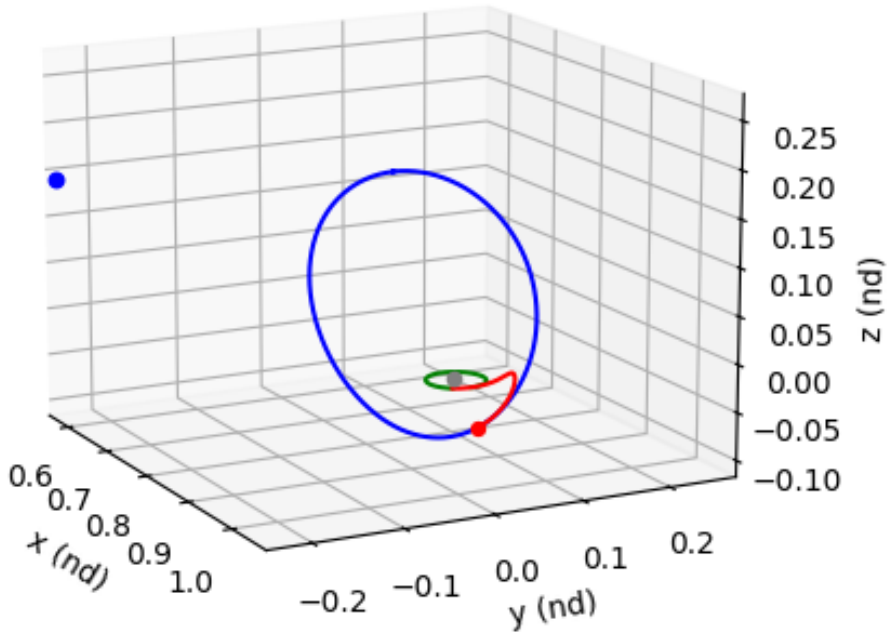
Next Challenge

- Using these phases, construct an Optimal Control Problem for a low-thrust transfer that starts on the LLO and arrives on the target halo, minimizing low-thrust burn time

- Hints:
  - During impulsive maneuvers we expected discontinuity in velocity between phases, is that still true in this case?
  - Is minimizing low-thrust burn time going to be handled as a link objective (between two phases) or handled directly by one of the phases?

Solution

```
138    ocp = ast.OptimalControl.OptimalControlProblem()
139    ocp.addPhase(llophase)
140    ocp.addPhase(ltphase)
141
142    ocp.addForwardLinkEqualCon(llophase, ltphase, [0,1,2,3,4,5,6])
143
144    ocp.solve_optimize()
145
146    llo = llophase.returnTraj()
147    llo = np.array(llo)
148    lt = ltphase.returnTraj()
149    lt = np.array(lt)
150    rend = ltphase.returnStaticParams()[0]
151    rend_state = halo_table(rend)
152
153    print(f'Transfer Duration = {(lt[-1][6] - lt[0][6])/Hour} hours')
```




```
==========================================
Transfer Duration = 51.53396346515125 hours
```

Plot and Save Final Solution

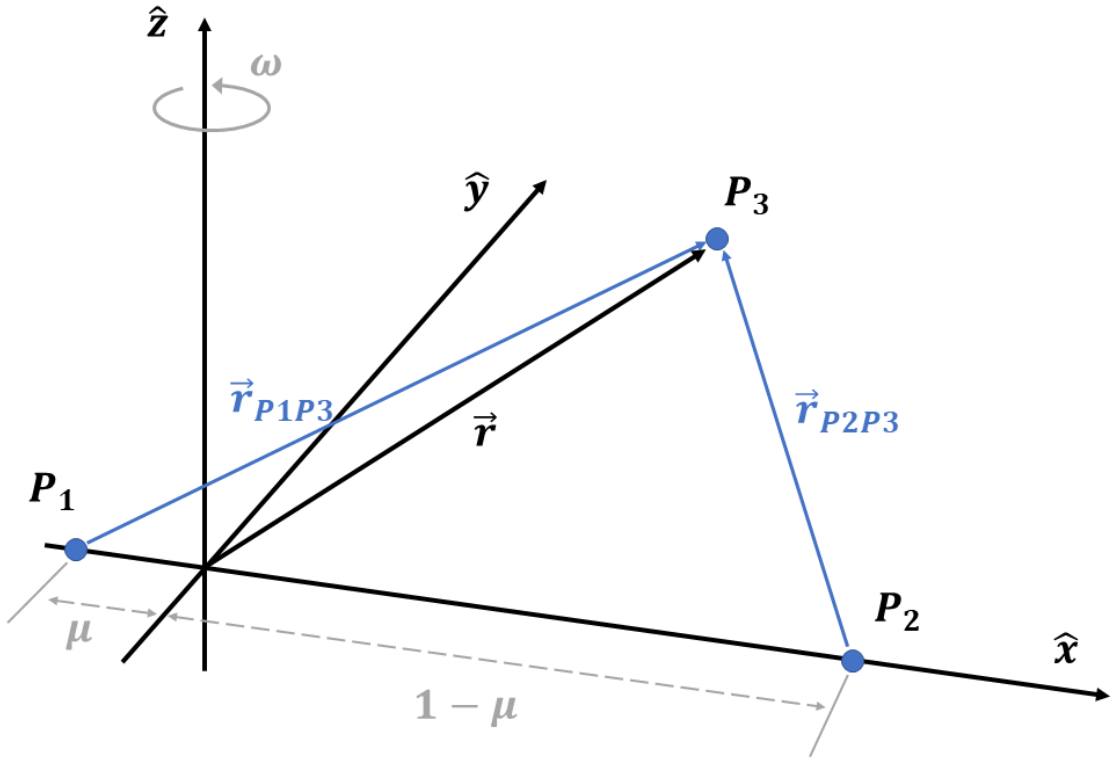
```
155    plt.figure()
156    ax = plt.axes(projection='3d')
157
158    ax.scatter([cr3bp.P1[0]], [cr3bp.P1[1]],[cr3bp.P1[2]],s=22,c='blue')
159    ax.scatter([cr3bp.P2[0]], [cr3bp.P2[1]],[cr3bp.P2[2]],s=20,c='grey')
160
161    ax.plot3D(halo_ig[:,0],halo_ig[:,1],halo_ig[:,2],c='blue')
162    ax.plot3D(llo[:,0], llo[:,1], llo[:,2], c = 'green')
163    ax.plot3D(lt[:,0], lt[:,1], lt[:,2], c = 'red')
164
165    ax.scatter([rend_state[0]], [rend_state[1]], rend_state[2], s=18, c='red')
166
167    plt.axis('equal')
168    plt.show()
```

```
171  data = {'llo':llo,'lt':lt,'rend_t':rend,'halo_t':halo_t,'halo_states':halo_states,'halo_ig':halo_ig,'halo_dt':halo_dt,'engine':lt_engine}
172
173  with open('CR3BP_Data.pkl','wb') as file:
174      pkl.dump(data,file)
```

Astro Demo 4: EPPR Frame

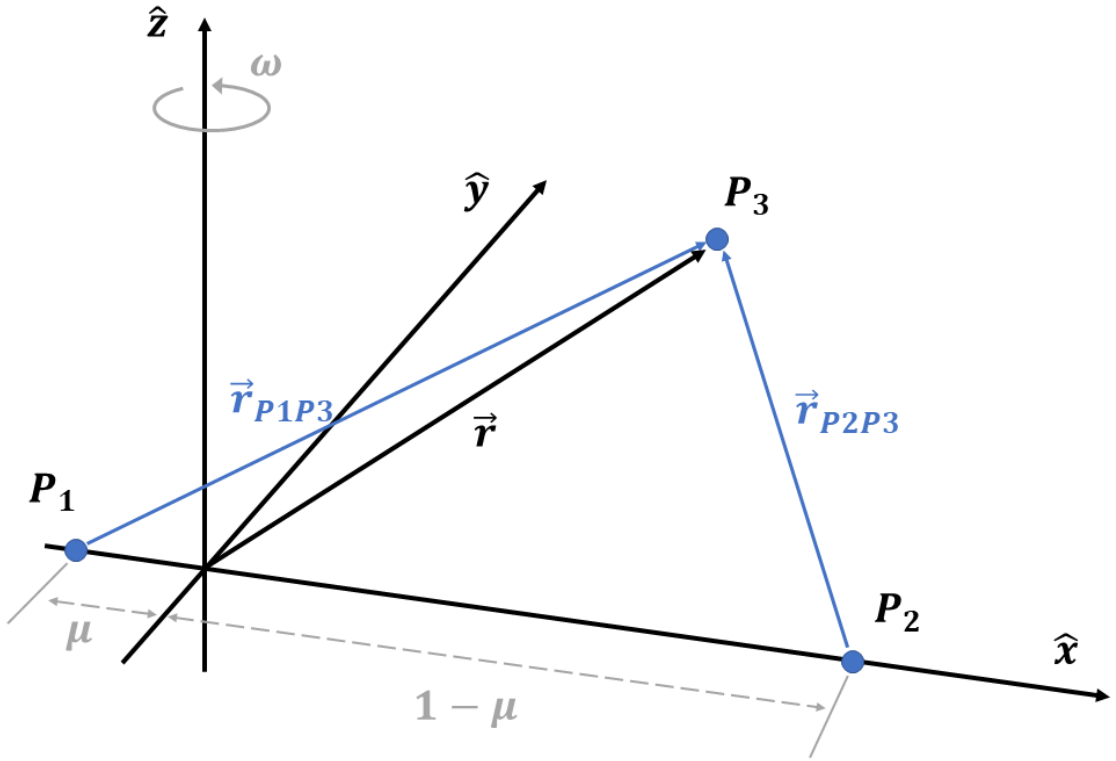
Model Definition

- The Ephemeris Pulsing Rotating (EPPR) Frame is an ephemeris Model that utilizes CR3BP basis vectors
- Due to leveraging SPICE ephemerides, the EPPR Model requires a Frame



Model Definition (cont.)

- EPPR is dynamically scaled to maintain unity distance between P1 and P2
- The Frame dynamically computes ephemeris perturbations and frame pulsing
- EPPR is **ONLY** defined for fixed, user-defined epoch ranges established at Frame creation



Imports

```
1   import asset_asrl as ast
2   import asset_asrl.OptimalControl as oc
3   import asset_asrl.Astro.Constants as c
4   from asset_asrl.Astro.AstroModels import EPPR, EPPR_LT
5   from asset_asrl.Astro.AstroFrames import EPPRFrame
6   from asset_asrl.Astro.Extensions.ThrusterModels import LowThrustAcc
7   from asset_asrl.Astro.Date import datetime, datetime_to_jd
8   import spiceypy as spice
9
10  import numpy as np
11  import matplotlib.pyplot as plt
12  from mpl_toolkits import mplot3d
13  import pickle as pkl
14
15  vf = ast.VectorFunctions
16  Args = vf.Arguments
```

Read CR3BP Trajectory Data

```
19  #Read in CR3BP data and unpack
20
21  with open('CR3BP_Data.pkl', 'rb') as file:
22      data = pkl.load(file)
23
24  halo_ig = data['halo_ig'] #halo orbit traj
25  halo_dt = data['halo_dt']
26
27  halo_states = data['halo_states']
28  halo_t = data['halo_t']
29  rend_t = data['rend_t']
30
31  halo_table = vf.InterpTable1D(halo_t, halo_states, kind='cubic')
32
33  llo = data['llo']
34  lt = data['lt']
35
36  lt_engine = data['engine']
```

Configure the Frame and Models

```
39    #Set up frame and models
40
41    spice.furnsh('./kernels/BasicKernel.txt')
42
43    init_datetime = datetime(2015, 3, 14, 9, 26)
44    JD0 = datetime_to_jd(init_datetime)  #March 14, 2015, 9:26 - 3/14/15/9:26
45    JDF = JD0 + 10
46
47    EFrame = EPPRFrame('EARTH',c.MuEarth,'MOON',c.MuMoon,c.LD,JD0,JDF,3000,'J2000')
48    EFrame.AddSpiceBodies(['SUN','JUPITER BARYCENTER'])
49    # EFrame.Add_P1_J2Effect()
50
51    eppr = EPPR(EFrame,ActiveAltBodies='All')
52    eppr_lt = EPPR_LT(EFrame,lt_engine,ActiveAltBodies='All')
53
54    Hour = c.hour/eppr.tstar
```

Next Challenge

- With the Model defined and CR3BP initial guess imported, create the optimized transfer from LLO to the target halo in the EPPR Model

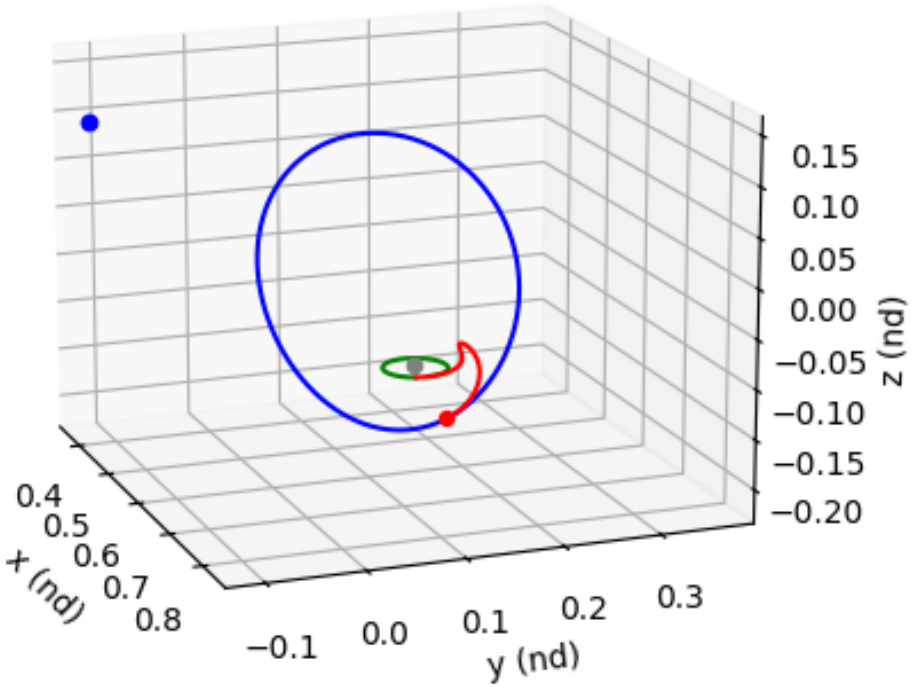
- Hints:
  - Remember, instantiated Models are used to create ASSET “phases”
  - “Phases” need initial guesses, which we have already created for this example

Solution

```
71  llophase = eppr.phase('LGL3')
72  llophase.setTraj(llo, 1000)
73  llophase.addBoundaryValue('Front',[0,1,2,3,4,5,6], llo[0])
74  llophase.addLowerDeltaTimeBound(30*Hour)
75  llophase.addUpperDeltaTimeBound(70*Hour)
76
77  ltphase = eppr_lt.phase('LGL5')
78  ltphase.setTraj(lt,1000)
79  ltphase.addLowerDeltaTimeBound(5*Hour)
80  ltphase.addDeltaTimeObjective(1.0)
81
82  #set Rendezvous Constraint via Static Parameter
83  ltphase.setStaticParams([rend_t])
84  ltphase.addLUVarBound('StaticParams', 0, 0.0, halo_dt)
85  ltphase.addEqualCon('Back', RendCon(halo_table), [0,1,2,3,4,5],[],[0])
86
87  #Add controller settings
88  ltphase.setControlMode('HighestOrderSpline')
89  ltphase.addLUNormBound('Path',[7,8,9], 0.01, 1.0)
90
91
92  ocp = oc.OptimalControlProblem()
93  ocp.addPhase(llophase)
94  ocp.addPhase(ltphase)
95
96  ocp.optimizer.set_SoeLSMode('L1')
97
98  ocp.addForwardLinkEqualCon(llophase, ltphase, [0,1,2,3,4,5,6])
99
00  ocp.solve_optimize()
```

Results

```
104   llo = llophase.returnTraj()
105   llo = np.array(llo)
106   lt = ltphase.returnTraj()
107   lt = np.array(lt)
108   rend = ltphase.returnStaticParams()[0]
109   rend_state = halo_table(rend)
110
111   print(f'Transfer Duration = {(lt[-1][6] - lt[0][6])/Hour} hours')
112
113   plt.figure()
114   ax = plt.axes(projection='3d')
115
116   ax.scatter([eppr.P1[0]], [eppr.P1[1]],[eppr.P1[2]],s=22,c='blue')
117   ax.scatter([eppr.P2[0]], [eppr.P2[1]],[eppr.P2[2]],s=20,c='grey')
118
119   ax.plot3D(halo_ig[:,0],halo_ig[:,1],halo_ig[:,2],c='blue')
120   ax.plot3D(llo[:,0], llo[:,1], llo[:,2], c = 'green')
121   ax.plot3D(lt[:,0], lt[:,1], lt[:,2], c = 'red')
122
123   ax.scatter([rend_state[0]], [rend_state[1]], rend_state[2], s=18, c='red')
124
125   plt.axis('equal')
126   plt.show()
```




```
==========================================
Transfer Duration = 65.99332500186648 hours
```

Thank you!

Contact Info:
Aaron Houin – ajhouin@crimson.ua.edu
Professor Sood – rsood@eng.ua.edu
ASSET Documentation: https://alabamaasrl.github.io/asset_asrl/