rpOML

MATLAB Orbital Mechanics Library
[C: 11JAN22] [Readme LM: 06SEP22]

*****This library is a work in progress and will be frequently updated*****

Overview

This library of MATLAB functions and scripts is designed around CU Boulder's ASEN5050, ASEN6008, and ASEN6060 courses. The intent is to catalog common astrodynamics functions and tools that are used in trajectory design and orbital mechanics. Optionally, the JPL CSPICE MICE Toolkit can be loaded through this library. Currently, rpOML is independent of CSPICE, but in future development, it is likely that astrodynamics functions from JPL's library will be utilized here. The toolkit is also useful to process JPL published constants, frames, and ephemerides. rpOML primarily consists of 2-Body orbital mechanics functions and interplanetary mission design related algorithms. The Circular Restricted 3-Body Problem (CR3BP) functions are being developed in the Fall of 2022 through the ASEN6060 Adv. Astrodynamics course.

Installation

1. Clone the following repository
2. In the download there is a file called `rpOMLstart.m` . Open this file in MATLAB and follow the commented block of code at the top to set the correct path(s).
3. **[Optional]** The JPL CSPICE MICE Toolkit can be loaded in with this library. To do so, follow the instructions in the file from step (2).

Using the Library

1. Copy/Paste `rpOMLstart.m` to your project's working directory.
2. At the top of your MATLAB script add either of the following lines to initialize rpOML:
   a. Without CSPICE MICE: `rpOMLstart();`
   b. With CSPICE MICE: `rpOMLstart('cspice',true);`

   Note: Step (2) will clear the CSPICE kernel and reinitialize it every time it is called. Consider this when attempting to use `rpOMLstart()` in functions or subroutines.

Conversions and Planetary Constants

The library has a set of celestial constants and conversion factors that can be easily imported as a structure. At the top of your script add: `c = constants();`

Celestial Constants Structure Contains:

1. au2km - 1 AU to km
2. dayyear - Days in 1 Earth Year
3. G - Gravitational Constant (km^3 s^-2 kg^-1)
4. c - Speed of Light (km/s)
5. Body Name (with subfields):
    a. mu (km^3/s^2) Gravitational Constant
    b. r (km) Body Radius
    c. sma (km) Semi-major axis w.r.t. orbiting central body
    d. t (s) Orbit Period (seconds)

A Circular Restricted 3-Body Problem set of constants can also be imported by: `c3 = constantscr3bp();`

CR3BP Constants Structure Contains:

1. System Name (with subfields):
    a. L/V/T - ND Characteristic Quantities for length, velocity, and time.
    b. mu/mu1 - ND Characteristic Mass for the secondary (system main) and primary masses respectively. Use “mu” for most computations as mu1 is trivial.
    c. b1/b2 (6x1) (6x1) arrays of ND Rot. Frame fixed locations of the primary and secondary bodies.

Numerical Orbit Propagation

Please refer to Equations of Motion (./eoms/) for a full list of functions available for numerical integration

Two Body Problem

Kepler's Equations (Combined) - Two-Body Propagation (./twobp/propKep)

```
xf = propKep(x_,dt,mu)
```

Where:

1. x_ (6x1) Initial State Vector [x;y;z;vx;vy;vz]
2. dt (1x1) Propagation Time (seconds)
3. mu (1x1) central body gravitational parameter (km^3/s^2)

ODE45 Based - Two-Body Propagation (./twobp/prop2bp)

Requires a full state structure (./create_state()). Plots results in figure

```
[t,stateX] = prop2bp(fX,dt,rcb,pltType)
```

Where:

1. fX (struct) Full state structure created from ./create_state()
2. dt (1x2) initial and final time column array (seconds)
3. rcb (1x1) Radius of central body (for plotting reasons) (km)
4. pltType String Variable with Options: 'none' - no plot or 'given' - makes plot in new figure

Direct EOMs - Two-Body Propagation

```
options = odeset('reltol',1e-8,'abstol',1e-8');
[t,x] = ode45(@(t,y) eom2BP(t,y,mu), [0:dt:tf], xi_, options)
```

Circular Restricted 3-Body Problem

Direct EOMs - Non-Dim. Rotating CR3BP Propagation

```
options = odeset('reltol',1e-8,'abstol',1e-8);
[t,x] = ode113(@(t,y) eomCR3BP(t,Y,mu3), [0:dt:tf], xi_, options)
```

Where:

1. mu3 (1x1) characteristic mass (ND)
1. dt/tf (1x1) (1x1) dt arbitrary time step, tf final time (ND)
2. xi_ (6x1) ND initial state [x;y;z;vx;vy;vz]

Direct EOMs w/ Stop Conditions - Non-Dim. Rotating CR3BP Propagation

```
options = odeset('Events',@cr3bp_Event_0Y_PdY,'reltol',1e-8,'abstol',1e-8);
[t,x] = ode113(@(t,y) eomCR3BP(t,Y,mu3), [0:dt:tf], xi_, options)
```

Where: "@cr3bp_Event_0Y_PdY()" is a stopping conditions that is y=0 (0Y) and positive dy (PdY)

Direct EOMs - CR3BP+STM N.D Propagation

```
[t,x] ode113(@(t,y) eomCR3BPwSTM(t,Y,mu3), [0:dt:tf], xi_, options)
```

Where: Y (42x1) Initial States and Partial of States.

N-Body Problem (WIP)

The function "eomNBP()" exists but is still work in progress.

Scripts/Functions Directory

*** - Scripts and functions with an asterisk after the name* indicate that they are commented and have a help description available.**

Most functions have description block comments at the top that can be called by the MATLAB console window command:

```
help <name_of_function>
```

Assumptions, warnings, TODO, inputs, outputs, references, and other data is all noted at the top of each function in this comment block. Below is a summarized table of all the scripts and functions in this library.

(Functions in blue are used more often and are especially useful)

Astrodynamics Functions

Constants, States, Frames, and Conversions (./)

File Path and Name	Description
./constants()	Function to load in planetary constants and conversions info including: Mu, radius, Orbit SMA, Orbit Period, Grav Constant, AUtoKM, and more.
./constantscr3bp()*	Constants for the CR3BP including characteristic quantities and primary/secondary body locations for many 3-body systems in our solar system
./conv_ele2state()	Converts classical orbital elements (a, e, i , o w, ta) to a cartesian state in the inertial and perifocal frames (includes the required rotation matrix)
./conv_state2ele()	Converts inertial state vector to classical orbital elements
./create_state()*	Creates a full state structure with all cartesian coordinates and orbital elements. Full orbit properties are given in the output MATLAB structure.
./dcm_iot2rth()*	Given inc., RAAN, true anomaly, and arg. of peri. (in radians) find DCM

CSPICE Wrapper Functions (./cspice/)

File Path and Name	Description
cspice_loadKernal()*	Loads specified kernel to memory (good for adtl. .bsp files)
cspice_queryState()*	Returns structure of states, times (w/ variations), frames, and other state info.
cspice_queryStateSingle()*	Same as above but for a single state only.

“Get” Functions (convenience functions and planetary states) (./)

File Path and Name	Description
./getDatePastDate()*	Given a calendar date and number of days elapsed, find the new calendar date.

./getJulianDate()*	Wrapper to MATLAB's Julian date conversion from calendar date.
./getMeeusData()*	Returns an array of planetary coefficients for the Meeus algorithm.
./getStateCircularPlanar()*	Returns Cartesian State w.r.t Time (CIRCULAR-PLANAR ORBITS ONLY)
./getStatePlanet()*	Returns MJ2000EC inertial cartesian state for a given julian date(s) vector using the Meeus algorithm and w/ NAIF JPL Ephemerides DE file support.
./getGMATmodJD()*	Returns modified Julian date specifically compatible for GMAT given a reg. JD
./getInterplanetaryLambertInfo()*	Returns text outputs and variables to detail a Lambert transfer between two planets.

Equations of Motion (./eoms/)

File Path and Name	Description
eom2BP()*	2-Body Equations of Motion (Integrate with ODE45)
eomdual2BP()*	Dual 2-Body EOMs (Integrate with ODE45)
eomNBP()*	N-Body Equations of Motion (Integrate with ODE45)
eomCR3BP()*	Circular Restricted 3-Body Problem (Integrate with ODE113)
eomCR3BPwSTM()*	CR3BP with State Transition Matrix (STM) (Integrate with ODE113)

Two-Body Problem (./twobp/)

File Path and Name	Description
getAnomalyandDt()	Gets Eccentric, Mean Anomaly, and Time Since Periapsis
kepElipAnomaly()*	Newton Iterative Process to find F and theta given Mh and e
kepHypAnomaly()*	Iterative Process to find F and theta given Mh and e
prop2bp()	Propagate state using 2-BP EOMs and optionally plot trajectory
propKepElip()*	Propagate using Kepler's Equation of an Ellipse
propKepHyp()*	Propagate using Kepler's Equation of a Hyperbola
propKep()	Integrate any conic 2BP with Kepler's Equations

Lambert's Problem (./lambert/)

File Path and Name	Description
lambert0rev()*	Universal variable Lambert algorithm for Type I and II trajectories
lambertNrev()*	Computes N revolution Lambert Transfer given Lambert rev. type.
lambertgeo()*	Geometric Lambert's Algorithm

	(considerably slower but more intuitive than the Universal variable algorithm)
lambert_getc2c3()*	Gets the C2 and C3 constants for the universal variable algorithm
lambert_getPsiNRev()*	Compute Bounds on Psi given 'n>0' Number of Revolutions
lambert_getTOFfromPsi()*	Finds the associated time of flight given a value of the universal variable
lambert_getRevfromType()*	Computes rev number given Lambert trajectory type (max rev=4, type=9/10)
plotLambertTransfer()	[GEO ALG. ONLY] Plots geometric lambert solver's answer

Circular Restricted 3-Body Problem (./cr3bp/)

File Path and Name	Description
cr3bp_computeJacobiConstant()*	Computes the Jacobi Constant of a ND Rot. State
cr3bp_convertRotNDToInertial()*	Converts Rot. ND state and times to Inertial Dim quantities
cr3bp_plotsystem()*	Plots the ND Rot (and optionally the Dim Inertial) system and can include actual body sizes (ND Rot plot) Plots additionally can include zero-velocity contours and Lagrange pts.
cr3bp_plotsystemNoState()*	Plot CR3BP system only (without s/c traj) and additionally can include Jacobi constant for ZVCs.
cr3bp_jacobiZVC()*	Plots contours for Zero-Velocity Curves given a Jacobi energy level
cr3bp_Event_0Y()*	MATLAB odeset function to terminate ODE113 at the first y=0 crossing.
cr3bp_Event_0Y_PdY()*	MATLAB odeset function to terminate ODE113 at the first y=0 crossing and with dy>0.
cr3bp_Event_0dY()*	MATLAB odeset function to terminate ODE113 at the first dy=0 crossing.
cr3bp_yaxstopcond()	(not required) Does what "cr3bp_Event_0Y" does.
cr3bp_periodicOrbitX0()*	Adjusts initial condition state vector to achieve a periodic orbit.
cr3bp_dstm()*	Returns the Jacobian Matrix (dSTM/dY) procedurally generated fn().
cr3bp_getSTMfromY()*	Returns STM (6x6) given integration output and index (i)
cr3bp_computeLagrangePoints()*	Returns updated CR3BP system parameters structure with L1-L5 locations and a check to make sure the L1-L3 point computations have zero component in the JU/Jx function (should be 0 to machine accuracy)

Mission Design Functions

Launch Vehicle Performance

File Path and Name	Description
./lvperformance()*	Finds mass delivered for various LVs given a departure C3.

Porkchop Plotting (./porkchop/)

File Path and Name	Description
pkchp_porkchop()	Iteratively solves the Lambert's problem for Type I and II trajectories to create C3, arrival Vinf, and TOF contour plots
pkchp_plt()	Plots result from previous function. See example: "ex4_porkchopplot1"

Planar Flybys (./fly2bp/)

File Path and Name	Description
flybyCompute2D()*	Solves the planar post-flyby vinf and v2 computations
flybyPlot2D()*	Plots resulting structure from previous function

Resonant Orbits (./resonantorbits/)

File Path and Name	Description
resonantOrbitCalc()*	Computes a resonant orbit V-Infinity vectors and radii of close approaches.
resonantOrbitCheck()*	Plots the resulting structure from the previous function to verify it.

Broad Search Queries (./broadsearch/)

File Path and Name	Description
broadsearch()*	Computes multi-flyby sequence search (no resonant orbits) of planets
broadsearch_patchLegs()	[Not standalone] Patches two leg sets of Lambert solutions
broadsearch_plot()*	[Not standalone] Plots results from broadsearch()
broadsearch_plotTraj()*	[Not standalone] Plots intermediate integrated trajectory results
broadsearch_sequencename()*	[Not standalone] Generates string for planets' names.
broadsearchtesting.m	Script to test broadsearch() [example 9 is better though]

B-Plane Targeting (./bplane/)

File Path and Name	Description
bplaneBRBTfromRV()*	Returns the B-Plane BR and BT values only from a SV and CB mu
bplanefromRV()*	Returns a structure of B-Plane parameters from a SV and CB mu
bplanefromVi1Vi2()*	Returns a structure of B-Plane params. given an in/outgoing V-inf vectors
bplaneTCA()*	Returns linearized time of flight given state vector and CB mu
bplane_computeXYTCM()*	Returns maneuver DV (dv_z=0) & state for TCM given desired BP targets
bplane_computeXYZTCM()*	Returns maneuver DV given desired BP targets
bplane_getJacobian()*	Returns Jacobian matrix (3x3)
bplane_getSTR()*	Gets the S, T, and R frame vectors in Inertial coordinates
bplane_getSTRDCM()*	Returns the STR→Inertial Directional Cosine Matrix (DCM)

Example Scripts (./examples/)

File Name	Description
ex1_statesandconstants.m*	Introduction to 2-body orbital dynamics state creation and propagation using integration or Kepler's method
[WIP] ex2_nbp.m	N-Body Problem simulation example
ex3_lambert_transferTypes.m*	This script is intended to show how the transfer time of flight varies between two bodies as a function of the universal variable for Hyperbolic, Parabolic, and Elliptical Type I-IV trajectories
ex4_porkchopplot1.m*	Ballistic Type I and II porkchop plot between Earth and Mars (example 1)
ex5_porkchopplot2.m*	Ballistic Type I and II porkchop plot between Earth and Mars (example 2)
ex6_launchvehicleperf.m*	This script is intended to show how to use the launch vehicle performance function
ex7_LambertMultiRev.m*	Demonstrates capability of multi-revolution Lambert arcs from Earth-Venus
ex8_BPlaneJacobianLinear.m*	Demonstrates acceptable values for perturbation (DV) magnitude to construct a B-Plane Jacobian matrix that's in the linear region (acceptable)
ex9_BroadSearch.m*	Shows how to use the broad search functions for a EVEEJ sequence search
ex10_PeriodicOrbitConstruction_CR3BP.m*	Single shooting method to find a periodic orbit given some initial condition
ex11_cr3bpPlotting.m*	Shows various plotting capabilities of cr3bp_plotSystem() and cr3bp_plotSystemNoState()

Code Verification Scripts/Data (./codeverif/)

File Name	Description
test_lambert0rev.m*	Verify 0 Revolution Lambert Algorithm
test_planetarystates.m*	Verify Meeus Algorithm to compute planetary inertial states in MJ200EC
test_planetdata()*	Test cases data to verify planetary states
test_bpXYZtcm.m test_bpXYZ_tcm_gmat.script test_bpXYZtcm_gmat.JPG	MATLAB file designs maneuver using B-Plane parameters for corrections GMAT script verifies maneuver and delivery conditions JPG is from GMAT showing propagated final state
test_propkep.m	Tests "universal" Kepler propagator against Lambert solution