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

ODEs and Integrators

ODEs

- ASSET enables optimal control and integration of systems of ordinary differential equations (ODEs)
- In ASSET ODEs are special vector functions of the form
  - State variables: $\mathbf{x}$
    - Ex: Position,velocity, mass
  - Time variable : t
  - Control variables: $\mathbf{u}$
    - Ex: Thrust direction, angle of attack
  - Parameter variables: $\mathbf{p}$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{y}) = \mathbf{f}(\mathbf{x}, t, \mathbf{u}, \mathbf{p})$$

ODEArguments

- Optimal control module has special `ODEArguments` class for defining ODEs.

```
import numpy as np
from asset_asrl import VectorFunctions as vf
from asset_asrl import OptimalControl as oc

from asset_asrl.VectorFunctions import Arguments as Args
from asset_asrl.OptimalControl import ODEArguments as ODEArgs
```

  - Provide # of States, Controls(if any), Params(if any)

- Additional members for accessing specific components
  - `.XVec(),.XVar(i)`
  - `.TVar()`
  - `.UVec(),.UVar(i)`
  - `.PVec(),.PVar(i)`

```
XVars = 6
UVars = 3
PVars = 0

XtU = oc.ODEArguments(XVars,UVars,PVars) # [ [R,V] , t, [U],[]]

# no need to specify PVars if there aren't any, same would go for UVars
# if there were no control variables
XtU = oc.ODEArguments(XVars,UVars)

# Index state,control or parameter vectors
XVec = XtU.head(6)
XVec = XtU.XVec()

UVec = XtU.segment(XVars+1,UVars)
UVec = XtU.UVec()

r0,r1,r2,v0,v1,v2 = XVec.tolist()
u0,u1,u2 = UVec.tolist()

# If we had ode parameters
#PVec = XtU.segment(XVars+1+UVars,PVars)
#PVec = XtU.PVec()

R,V  = XVec.tolist([(0,3),(3,3)])
U = UVec

### Index specific elements
t  = XtU.TVar()   # same as XtU[XVars]

v1 = V[1]
v1 = XVec[4]
v1 = XtU.XVar(4) # XtU.UVar(i) is same as XtU[i]

u0 = UVec[0]
u0 = XtU.UVar(0) # XtU.UVar(i) is same as XtU[XVars+i]
```

Defining ODEs

- Defined by inheriting `oc.ODEBase` class
- ODEs give us access to ASSET's main features
  - Integrators
  - Phases

$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z] = [\mathbf{r}, \mathbf{v}]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{|\mathbf{r}|^3}\mathbf{r} \end{bmatrix}$$

```
class TwoBody(oc.ODEBase):
    def __init__(self,mu):

        XVars = 6
        UVars = 0
        PVars = 0

        Xt = oc.ODEArguments(XVars,UVars,PVars)
        Xt = oc.ODEArguments(XVars,UVars)
        Xt = oc.ODEArguments(XVars) ## Can drop ODE doesent have UVars,PVars


        R,V  = Xt.XVec().tolist([(0,3),(3,3)])

        G = -mu*R.normalized_power3()

        Rdot = V
        Vdot = G

        ode = vf.stack(Rdot,Vdot)


        super().__init__(ode,XVars)
        #super().__init__(ode,XVars,UVars)
        #super().__init__(ode,XVars,UVars,PVars)
```

Example: Simple Two-Body Low Thrust

$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z] = [\mathbf{r}, \mathbf{v}]$$
$$\mathbf{u} = [u_x, u_y, u_z]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{|\mathbf{r}|^3}\mathbf{r} + \alpha\mathbf{u} \end{bmatrix}$$

```
class TwoBodyLT(oc.ODEBase):

    def __init__(self,mu,alpha):

        XVars = 6
        UVars = 3

        XtU = oc.ODEArguments(XVars,UVars)

        R,V  = XtU.XVec().tolist([(0,3),(3,3)])
        U = XtU.UVec() ## Throttle vector |U|<=1


        G = -mu*R.normalized_power3()
        Acc = G + U*alpha

        Rdot = V
        Vdot = Acc

        ode = vf.stack(Rdot,Vdot)

        super().__init__(ode,XVars,UVars)
```

Practice: CR3BP

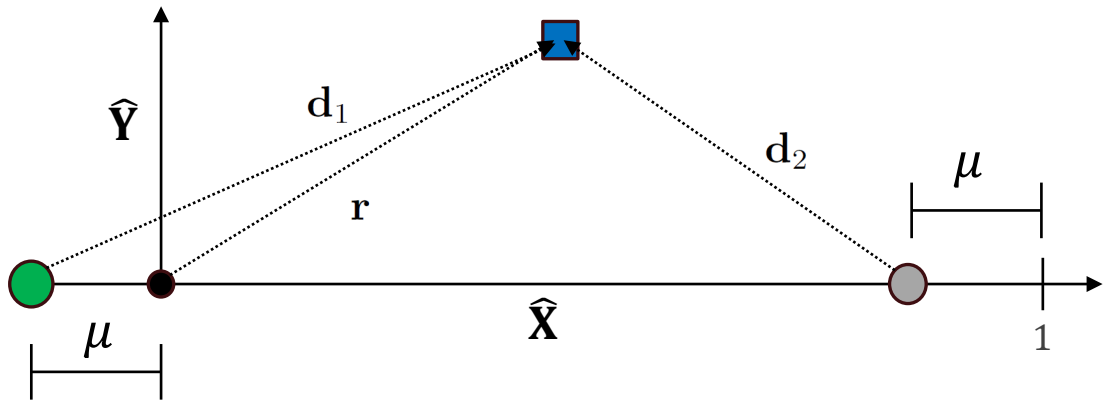
- Turn the CR3BP EOMs into an ODE

$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z] = [\mathbf{r}, \mathbf{v}]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{1-\mu}{d_1^3}\mathbf{d}_1 - \frac{\mu}{d_2^3}\mathbf{d}_2 + \mathbf{q} \end{bmatrix}$$

where:

$$\mathbf{d}_1 = \mathbf{r} - [-\mu, 0, 0]$$
$$\mathbf{d}_2 = \mathbf{r} - [1-\mu, 0, 0]$$
$$\mathbf{q} = [2v_y + r_x, -2v_x + r_y, 0]$$



Practice: CR3BP

- Turn the CR3BP EOMs into an ODE

$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z] = [\mathbf{r}, \mathbf{v}]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{1-\mu}{d_1^3}\mathbf{d}_1 - \frac{\mu}{d_2^3}\mathbf{d}_2 + \mathbf{q} \end{bmatrix}$$

where:

$$\mathbf{d}_1 = \mathbf{r} - [-\mu, 0, 0]^T$$
$$\mathbf{d}_2 = \mathbf{r} - [1-\mu, 0, 0]^T$$
$$\mathbf{q} = [2v_y + r_x, -2v_x + r_y, 0]^T$$

```
class CR3BP(oc.ODEBase):

    def __init__(self, mu):

        Xt = ODEArgs(6)

        ## Xt=[R,V,t]
        R = Xt.head(3)
        V = Xt.segment(3,3)

        # P1,P2 positions
        P1loc = np.array([-mu,0,0])
        P2loc = np.array([1.0-mu,0,0])

        # Relative Position Vectors
        D1 = R-P1loc
        D2 = R-P2loc

        # Planar elements of r,v
        rx = R[0]
        ry = R[1]
        vx = V[0]
        vy = V[1]

        # Gravity terms
        G1 = (mu-1)*D1.normalized_power3()
        G2 = -mu*D2.normalized_power3()

        # Rotating Frame Terms,pad 0 to bottom of result for sum
        Q = vf.stack(2*vy+rx,-2*vx+ry,0.0)

        # Total Acceleration
        Acc = vf.sum([G1,G2,Q])

        ode = vf.stack(V,Acc)

        super().__init__(ode,6)
```

Practice: Two-Body finite burn model with RTN throttle vector

- Implement Two Body finite burn model with RTN throttle vector as an ODE

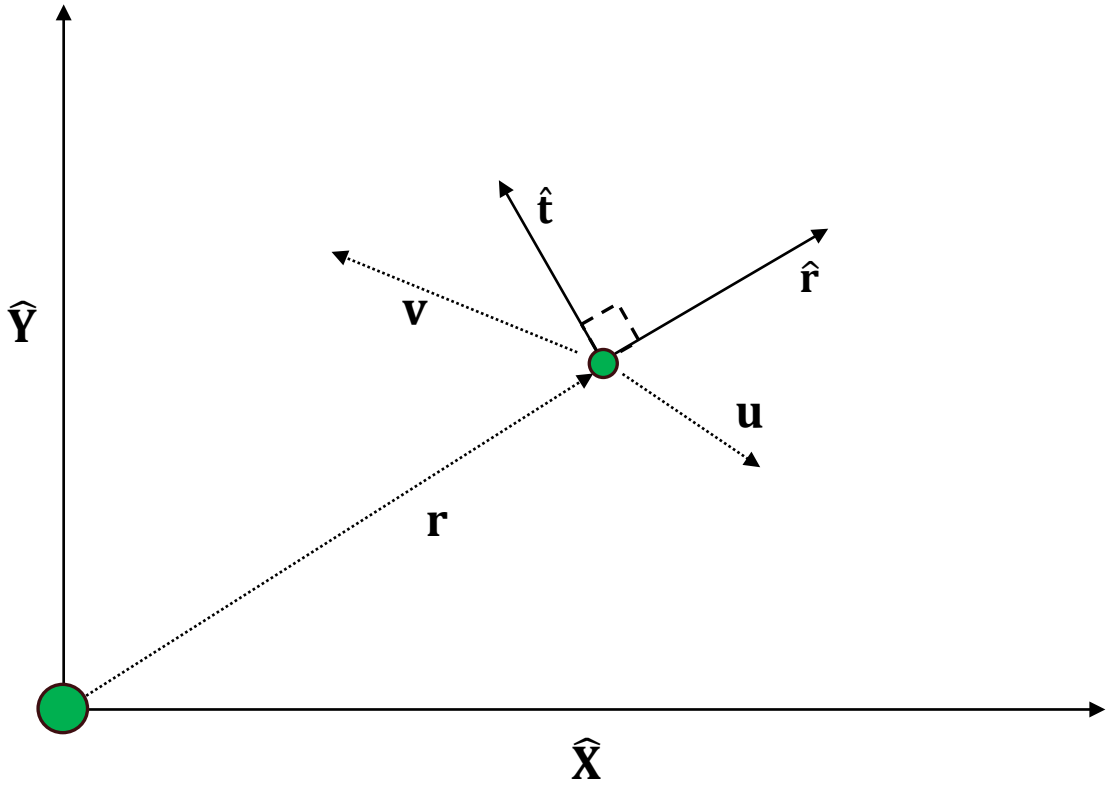
$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z, m] = [\mathbf{r}, \mathbf{v}, m]$$

$$\mathbf{u} = [u_r, u_t, u_n]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{|\mathbf{r}|^3}\mathbf{r} + \frac{\tau}{m}\mathbf{M}\mathbf{u} \\ -|\mathbf{u}|\frac{\tau}{g_0 I_{sp}} \end{bmatrix}$$

where:

$$\mathbf{M} = [\hat{\mathbf{r}}; \hat{\mathbf{t}}; \hat{\mathbf{n}}]$$
$$\mathbf{n} = \mathbf{r} \times \mathbf{v}$$
$$\mathbf{t} = \mathbf{n} \times \mathbf{r}$$



Practice: Two-Body finite burn model with RTN throttle vector

- Implement Two-Body finite burn model with RTN throttle vector as an ODE

$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z, m] = [\mathbf{r}, \mathbf{v}, m]$$

$$\mathbf{u} = [u_r, u_t, u_n]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{|\mathbf{r}|^3}\mathbf{r} + \frac{\tau}{m}\mathbf{M}\mathbf{u} \\ -|\mathbf{u}|\frac{\tau}{g_0 I_{sp}} \end{bmatrix}$$

where:

$$\mathbf{M} = [\hat{\mathbf{r}}; \hat{\mathbf{t}}; \hat{\mathbf{n}}]$$
$$\mathbf{n} = \mathbf{r} \times \mathbf{v}$$
$$\mathbf{t} = \mathbf{n} \times \mathbf{r}$$

```
class FBRTNModel(oc.ODEBase):

    def __init__(self,mu,tau,Isp,g0 = 9.81):

        XVars = 7
        UVars = 3

        XtU = oc.ODEArguments(XVars,UVars)

        R,V,m  = XtU.XVec().tolist([(0,3),(3,3),(6,1)])
        U = XtU.UVec() ## Throttle vector |U|<=1

        RTNBasisFunc = RTNBasis()
        RTNCoeffs = RTNBasisFunc(R,V)
        M = vf.ColMatrix(RTNCoeffs,3,3)


        Acc = -mu*R.normalized_power3() + (tau/m)*(M*U)
        mdot =  -(tau/(g0*Isp))*U.norm()

        Rdot = V
        Vdot = Acc

        ode = vf.stack(Rdot,Vdot,mdot)

        super().__init__(ode,XVars,UVars)
```

Next Steps

- ODEs give us access to two of the main interfaces of the library
  - Integrators: Solve initial value problems, control laws
  - Phases: Solve boundary value problems, optimal control problems

```
ode = TwoBody(1.0)

integ = ode.integrator(.1)

phase = ode.phase("LGL3")
```

Integrators

- Utility for solving initial value problems for an ODE
- Useful for:
  - Pure propagation tasks (e.g. dispersion analysis)
  - Generating initial guesses
  - Validating optimizer solutions

$$\text{solve} : \mathbf{x}(t) \in [t_0, t_f]$$

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, t, \mathbf{u}, \mathbf{p})$$
$$\mathbf{x}(t_0) = \mathbf{x}_0$$
$$\mathbf{u}(t) = \mathbf{g}(\mathbf{x}, t, \mathbf{p})$$
$$\mathbf{p}(t) = \mathbf{p}_0$$

Integrators: Initialization

- Create integrator from existing ODE object
  - Must specify default stepsize
  - Optionally specify integration method (default:DOPRI87)
- Modify step sizes if inappropriate

```
TBode = TwoBody(1.0)

DefStepSize = .1

#Initialization
TBInteg = TBode.integrator("DOPRI87",DefStepSize)
TBInteg = TBode.integrator("DOPRI54",DefStepSize)
TBInteg = TBode.integrator(DefStepSize)  # Default is DOPRI87


print("Default StepSize    = DefStepSize = "      , TBInteg.DefStepSize)
print("Default MinStepSize = DefStepSize/10000 = ", TBInteg.MinStepSize)
print("Default MaxStepSize = DefStepSize*10000 = ", TBInteg.MaxStepSize)

MinStepSize = 1e-6
MaxStepSize = 1.0

## Set def,min, and max step sizes manually
TBInteg.setStepSizes(DefStepSize,MinStepSize,MaxStepSize)
```

```
Default StepSize    = DefStepSize =  0.1
Default MinStepSize = DefStepSize/10000 =  1e-05
Default MaxStepSize = DefStepSize*10000 =  1000.0
```

Integrators: Tolerancing

- Integrators are adaptive error-controlled integration
  - Error controlled per step, not globally
- **YOU** must choose tolerances appropriately
  - Defaults suited for non-dimensional systems

$$\epsilon_{abs} = |\mathbf{x} - \mathbf{x}_{est}|_\infty$$
$$\epsilon_{rel} = |(\mathbf{x} - \mathbf{x}_{est})/\mathbf{x}|_\infty$$

```
TBode = TwoBody(1.0)
DefStepSize = .1
TBInteg = TBode.integrator(DefStepSize)  # Default is DOPRI87

print("Default AbsTols = [1.0-12...] = ",TBInteg.getAbsTols())
print("Default RelTols = [0.0,  ...] = ",TBInteg.getRelTols())

AbsTol      = 1.0e-13
RelTol      = 0

# Set tolerances uniformly for all state variables
TBInteg.setAbsTol(AbsTol)
TBInteg.setRelTol(RelTol)

AbsTols = np.array([1,1,1,3,3,3])*1.0e-13
RelTols = np.array([0,0,0,1,1,1])*1.0e-9
# Set tolerances individually for each state variables
TBInteg.setAbsTols(AbsTols)
TBInteg.setRelTols(RelTols)
```

```
Default AbsTols =  [1.e-12 1.e-12 1.e-12 1.e-12 1.e-12 1.e-12]
Default RelTols =  [0. 0. 0. 0. 0. 0.]
```

Integrators: Basic Usage

- .integrate(X0t0U0P,tf)->XftfU0P
  - Returns just the final state
- .integrate_dense(X0t0U0P,tf)->[X0t0U0P,…,XftfU0P]
  - Returns exact intermediate steps
- .integrate_dense(X0t0U0P,tf,N)->[X0t0U0P,…,XftfU0P]
  - Returns N linearly spaced intermediate steps

```
TBode = TwoBody(1.0)
DefStepSize = .1
TBInteg = TBode.integrator(DefStepSize)

r  = 1.0
v  = 1.1
t0 = 0.0
tf = 10.0

X0t0 = np.zeros((7))
X0t0[0]=r
X0t0[4]=v
X0t0[6]=t0

# Just the final full-state
Xftf = TBInteg.integrate(X0t0,tf)

TrajExact  = TBInteg.integrate_dense(X0t0,tf)

N  = 100

TrajInterpN = TBInteg.integrate_dense(X0t0,tf,N)
```

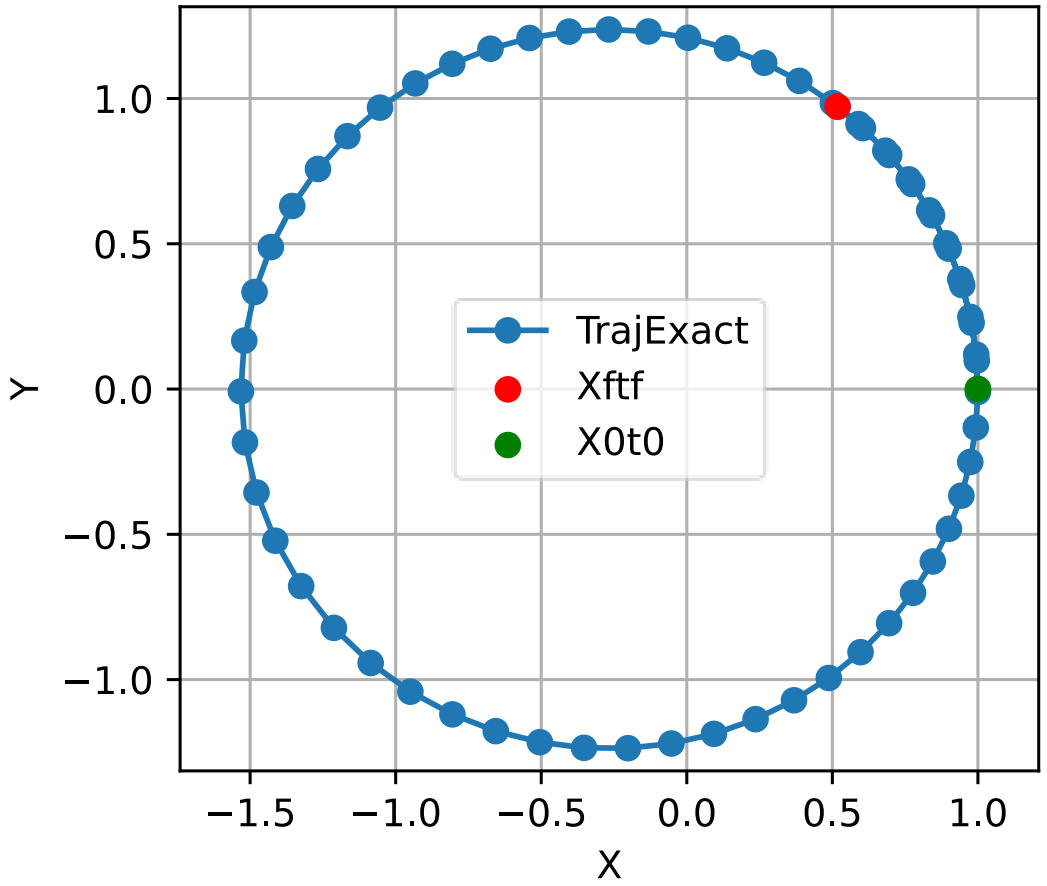
Integrators: Basic Usage

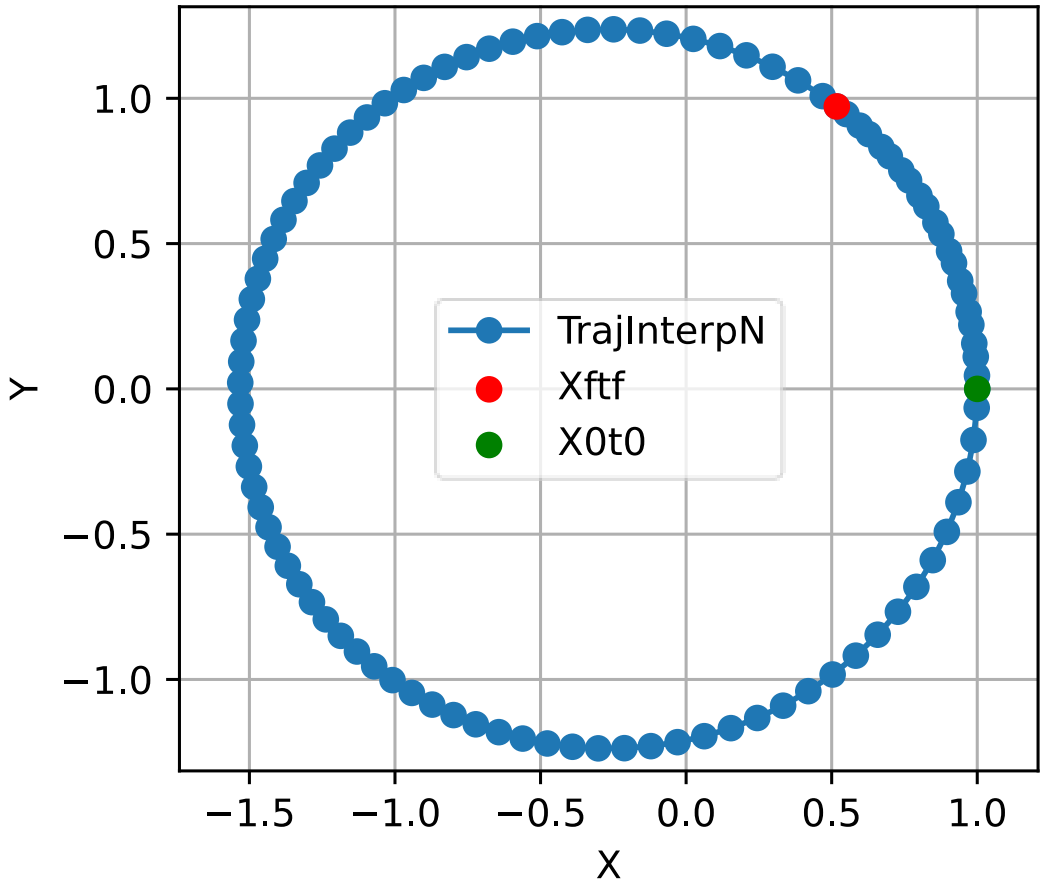
- `.integrate(X0t0U0P,tf)->XftfU0P`
  - Returns just the final state
- `.integrate_dense(X0t0U0P,tf)->[X0t0U0P,…,XftfU0P]`
  - Returns exact intermediate steps
- `.integrate_dense(X0t0U0P,tf,N)->[X0t0U0P,…,XftfU0P]`
  - Returns N linearly spaced intermediate steps

- **`tf` is the final time, not the delta time**

```
TBode = TwoBody(1.0)
DefStepSize = .1
TBInteg = TBode.integrator(DefStepSize)

r  = 1.0
v  = 1.1
t0 = 0.0
tf = 10.0

X0t0 = np.zeros((7))
X0t0[0]=r
X0t0[4]=v
X0t0[6]=t0

# Just the final full-state
Xftf = TBInteg.integrate(X0t0,tf)

TrajExact  = TBInteg.integrate_dense(X0t0,tf)

N  = 100

TrajInterpN = TBInteg.integrate_dense(X0t0,tf,N)
```

Integrators: Basic Usage





Integrators: Event Detection

- Integration can be stopped by one or more events

$$f(\mathbf{y}) = f(\mathbf{x}, t, \mathbf{u}, \mathbf{p}) = 0$$

- Event Format: `(eventfunction,direction,stopcode)`
  - `eventfunction: an Asset scalar function`
  - `direction: (-1,0,1)`
  - `stopcode: (0,1,2 … )`
- `Pass list of events to integration`
  - `ex: .integrate(X0,t0,[event1,…])->Xftf,EventLocs`

```
def ApseFunc():
    R,V = oc.ODEArguments(6).tolist([(0,3),(3,3)])
    return R.dot(V)

direction = -1
stopcode = False
ApoApseEvent  = (ApseFunc(),direction,stopcode)

direction = 1
stopcode = False
PeriApseEvent  = (ApseFunc(),direction,stopcode)

direction = 0
stopcode  = 2  # Stop after finding 2 apses
AllApseEvent  = (ApseFunc(),direction,stopcode)
```

$$f(\mathbf{y}) = \mathbf{r} \cdot \mathbf{v} = 0$$

Integrators: Event Detection

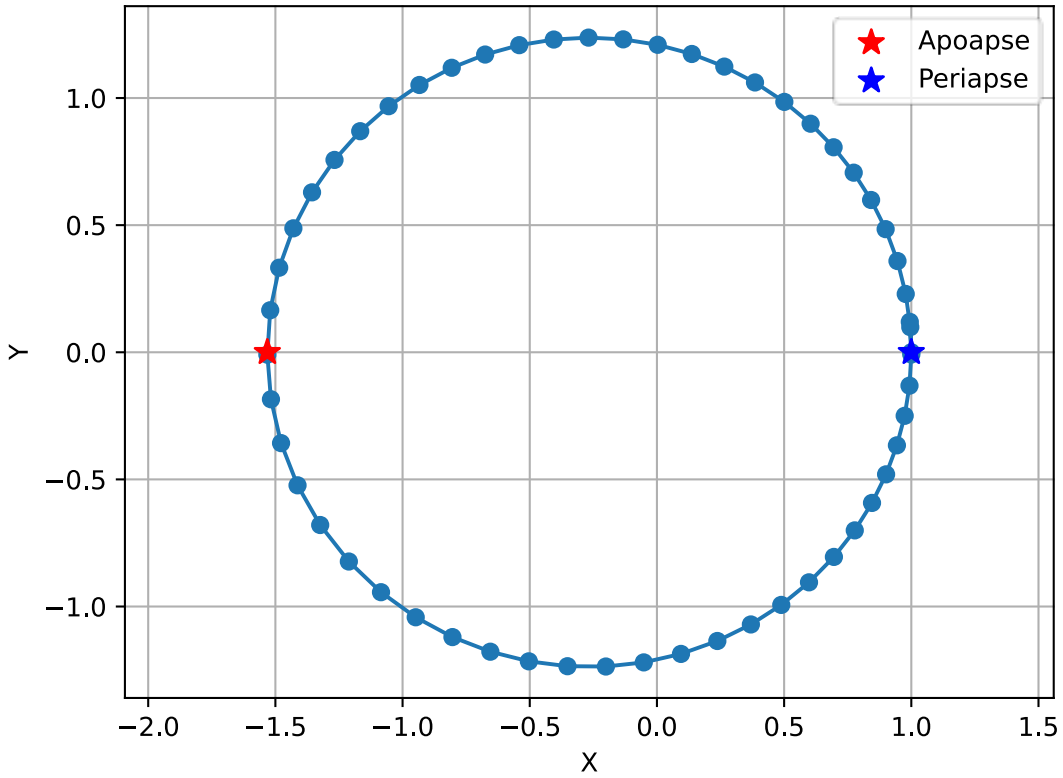
- Integration can be stopped by one or more events

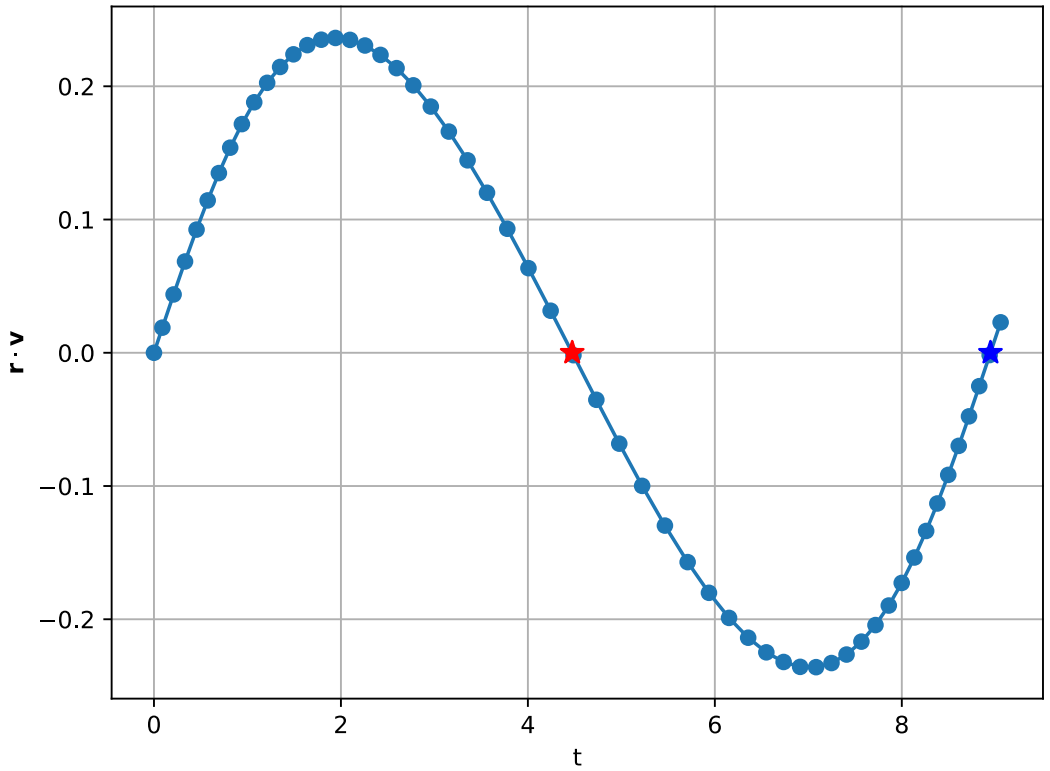
$$f(\mathbf{y}) = f(\mathbf{x}, t, \mathbf{u}, \mathbf{p}) = 0$$

- Event Format: `(eventfunction,direction,stopcode)`
  - `eventfunction: an Asset scalar function`
  - `direction: (-1,0,1)`
  - `stopcode: (0,1,2 … )`
- `Pass list of events to integration`
  - `ex: .integrate(X0,t0,[event1,…])->Xftf,EventLocs`

```
r  = 1.0
v  = 1.1
t0 = 0.0
tf = 100.0
N  = 1000

X0t0 = np.zeros((7))
X0t0[0]=r
X0t0[4]=v
X0t0[6]=t0

Events = [ApoApseEvent,PeriApseEvent,AllApseEvent]

## Just append event list to any normal call
Xftf, EventLocs = TBInteg.integrate(X0t0,tf,Events)

TrajExact, EventLocs  = TBInteg.integrate_dense(X0t0,tf,Events)

TrajInterpN, EventLocs  = TBInteg.integrate_dense(X0t0,tf,N,Events)

#EventLocs[i] will be empty if the event was not detected

ApoApseEventLocs  = EventLocs[0]
ApoApse =ApoApseEventLocs[0]

PeriApseEventLocs = EventLocs[1]
PeriApse =PeriApseEventLocs[0]

# Or
AllApseEventLocs  = EventLocs[2]
ApoApse  = AllApseEventLocs[0]
PeriApse = AllApseEventLocs[1]
```

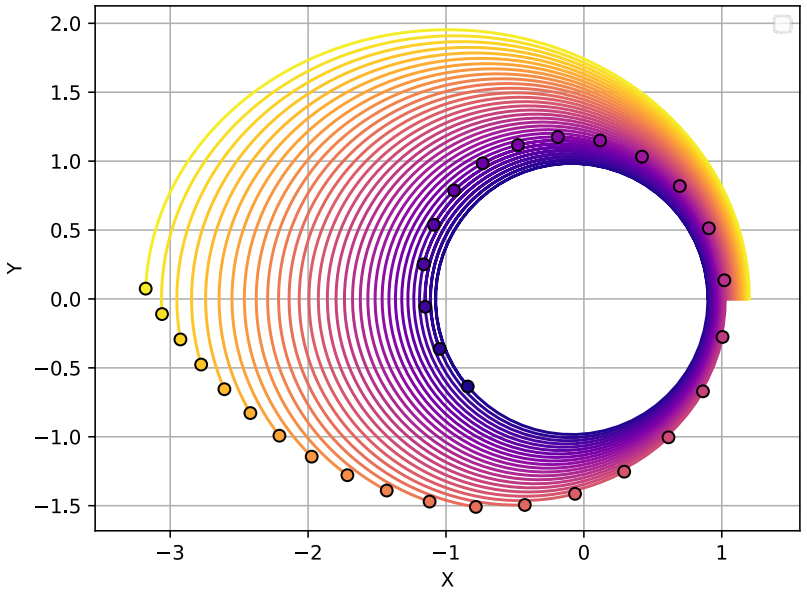
Integrators: Event Detection





Integrators: Parallelism

- `###_parallel(…)` versions of all integrate methods
- Pass in lists of initial states, final times, etc.
- Returns list of outputs corresponding to each input
- Set number of threads based on your computer




```
ntrajs = 32
tfs = [tf]*ntrajs
Ns  = [N]*ntrajs

rs = np.linspace(.9,1.2,ntrajs)

v  = 1.1
t0 = 0.0

X0t0s = []
for r in rs:
    X0t0 = np.zeros((7))
    X0t0[0]=r
    X0t0[4]=v
    X0t0[6]=t0
    X0t0s.append(X0t0)

nthreads = 16

Xftfs = TBInteg.integrate_parallel(X0t0s,tfs,nthreads)

Trajs  = TBInteg.integrate_dense_parallel(X0t0s,tfs,nthreads)

Trajs = TBInteg.integrate_dense_parallel(X0t0s,tfs,Ns,nthreads)
```

Integrators: Controls and Control Laws

- By default integration uses constant controls
- Add control law to update dynamically

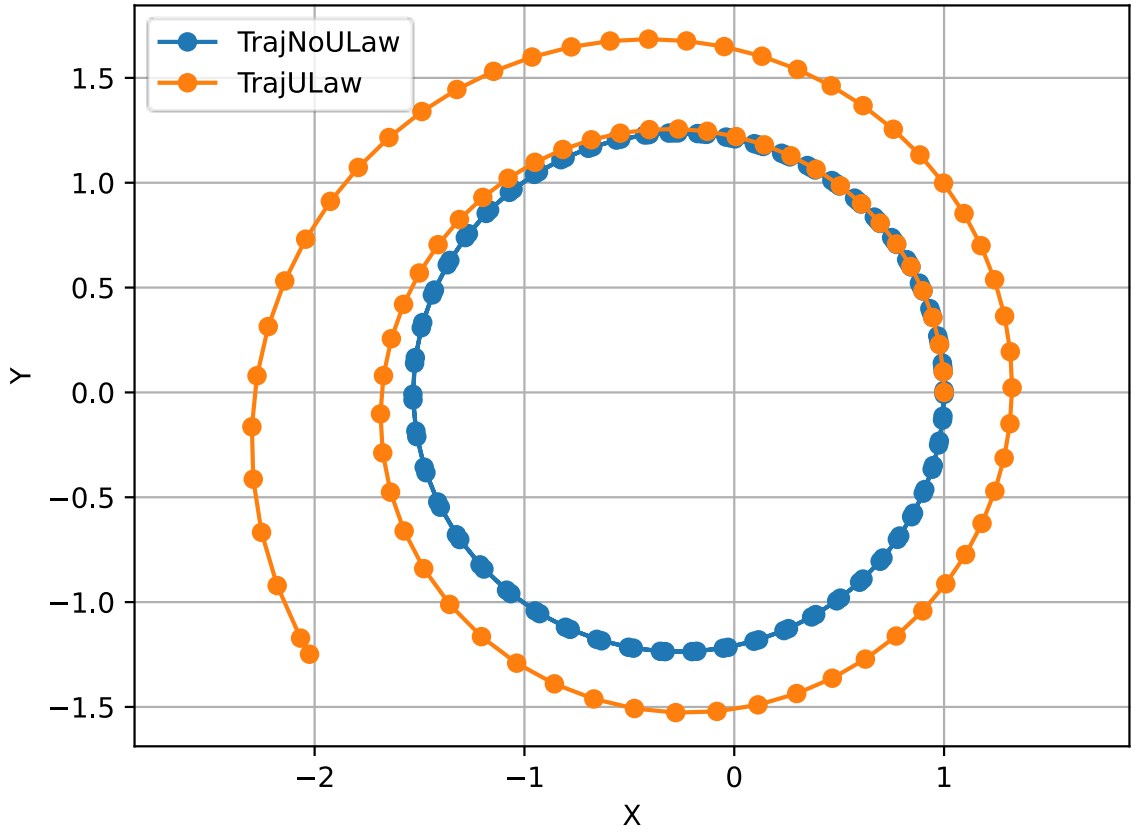
$$\mathbf{u} = \mathbf{f}(\mathbf{x}, t, \mathbf{p})$$

- Write control law as vector function
- Pass to integrator along with input variable indices
- Example: prograde throttle

$$\mathbf{u} = \hat{\mathbf{v}}$$

```
def ULaw(throttle):
    V = Args(3)
    return V.normalized()*throttle

ode = TwoBodyLTODE(1,.01)

integNoUlaw = ode.integrator("DOPRI87",.1)
integULaw   = ode.integrator("DOPRI87",.1,ULaw(1.0),[3,4,5])

r  = 1.0
v  = 1.1
t0 = 0.0
tf = 20.0

X0t0U0 = np.zeros((10))
X0t0U0[0]=r
X0t0U0[4]=v
X0t0U0[6]=t0

TrajNoULaw = integNoUlaw.integrate_dense(X0t0U0,tf)
TrajULaw   = integULaw.integrate_dense(X0t0U0,tf)
```

Integrators: Controls and Control Laws




```
def ULaw(throttle):
    V = Args(3)
    return V.normalized()*throttle

ode = TwoBodyLTODE(1,.01)

integNoUlaw = ode.integrator("DOPRI87",.1)
integULaw   = ode.integrator("DOPRI87",.1,ULaw(1.0),[3,4,5])

r  = 1.0
v  = 1.1
t0 = 0.0
tf = 20.0

X0t0U0 = np.zeros((10))
X0t0U0[0]=r
X0t0U0[4]=v
X0t0U0[6]=t0

TrajNoULaw = integNoUlaw.integrate_dense(X0t0U0,tf)
TrajULaw   = integULaw.integrate_dense(X0t0U0,tf)
```

Practice: Tangential Control Law

- Integrate Two-Body Low thrust ODE with tangential thrust control law, compare with prograde

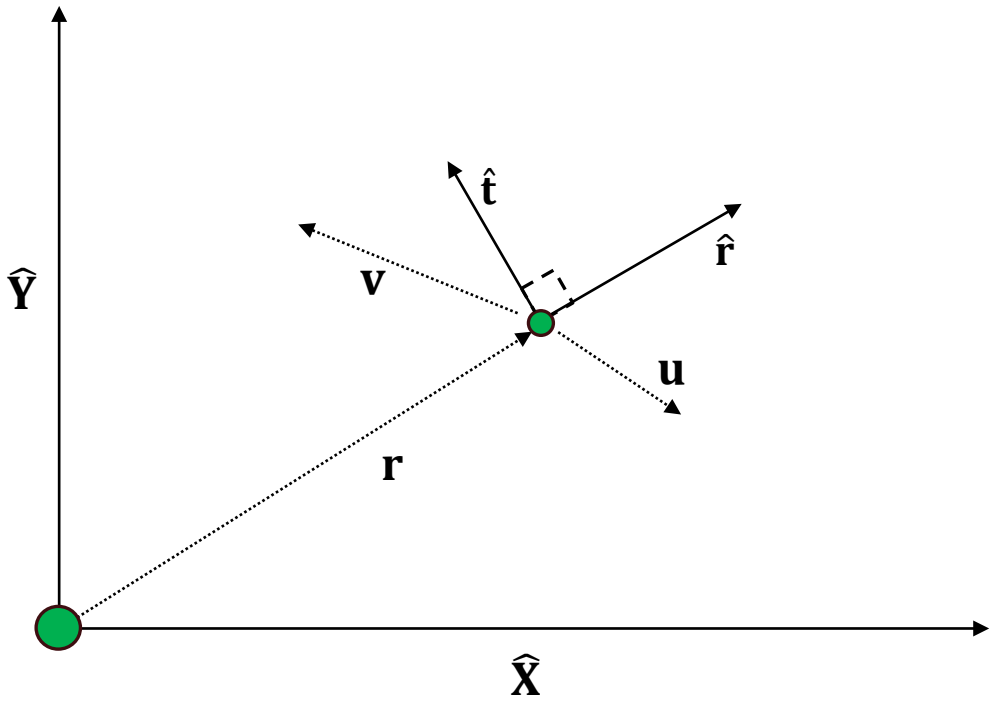
$$\mathbf{u} = \hat{\mathbf{t}}$$

where:

$$\mathbf{n} = \mathbf{r} \times \mathbf{v}$$
$$\mathbf{t} = \mathbf{n} \times \mathbf{r}$$

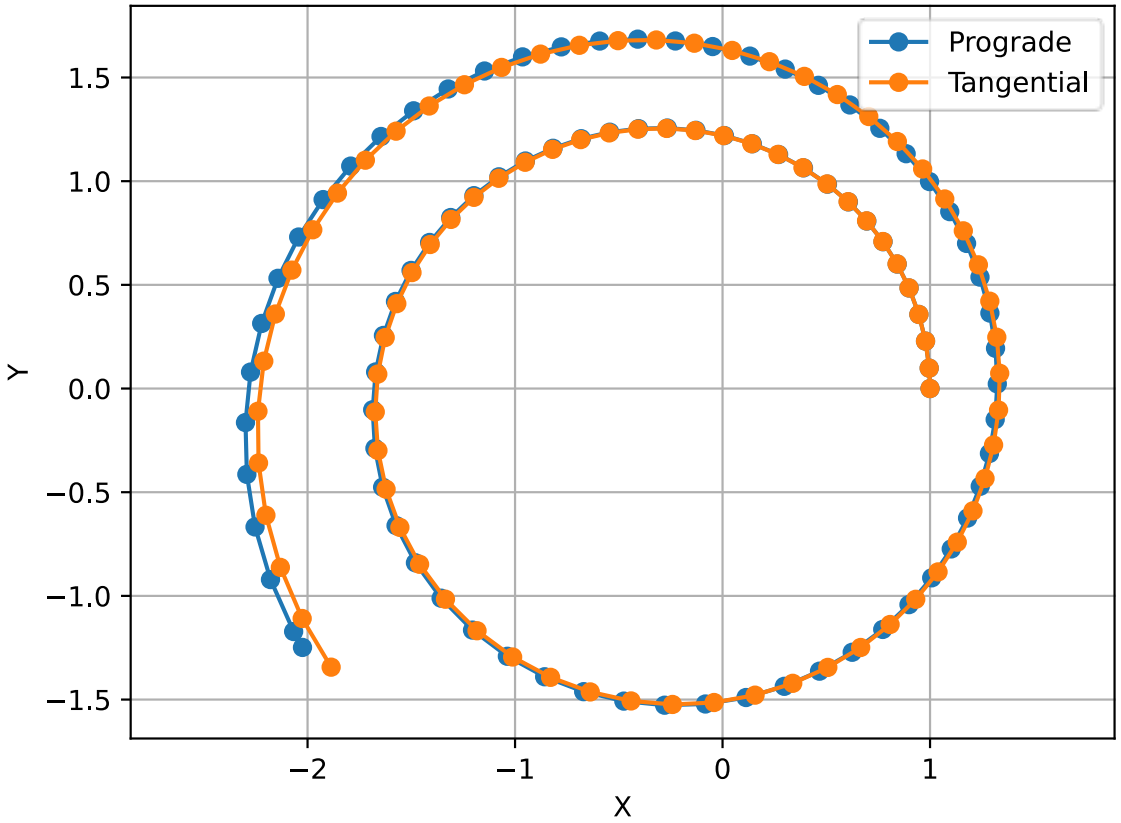
$$\mathbf{x} = [r_x, r_y, r_z, v_x, v_y, v_z] = [\mathbf{r}, \mathbf{v}]$$
$$\mathbf{u} = [u_x, u_y, u_z]$$

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{v} \\ -\frac{\mu}{|\mathbf{r}|^3}\mathbf{r} + \alpha\mathbf{u} \end{bmatrix}$$



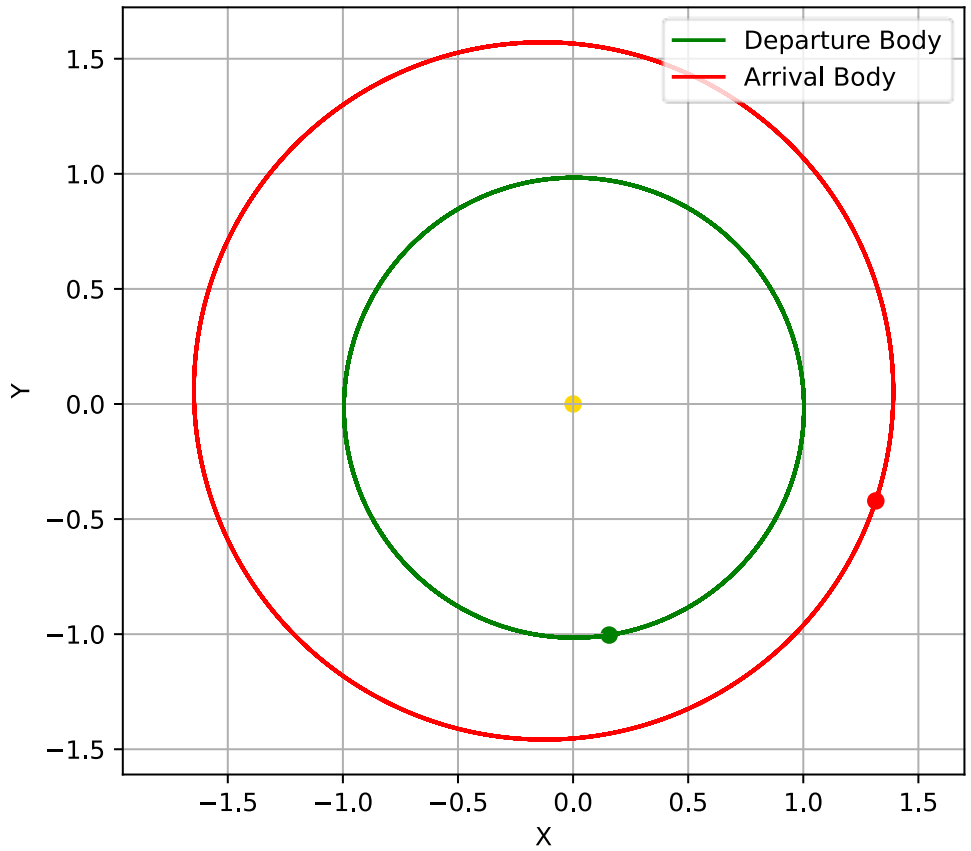
Practice: Tangential Control Law

```
def TanLaw(throttle):
    R,V = Args(6).tolist([(0,3),(3,3)])
    Rhat = R.normalized()
    Nhat = R.cross(V).normalized()
    That = Nhat.cross(Rhat).normalized()
    return That*throttle


integTanLaw   = ode.integrator("DOPRI87",.1,TanLaw(1.0),[0,1,2,3,4,5])
TrajTanLaw = integTanLaw.integrate_dense(X0t0U0,tf)
```



Practical Example

- Goal: Find an initial guess for low-thrust transfer between two bodies
- Requires:
  - `TwoBody`
  - `TwoBodyLTODE`
  - `TanLaw`



Practical Example

```
tmax = 100.0 ## Max epoch time
mu = 1.0
alpha = .02  ## low-thrust accelleration
ig_throttle = 0.5  ## throttle level for our ig

## Initial conditions of departare arrival bodies at initial epoch
dep_ig =[ 1.567e-01, -1.004e+00,  4.690e-05, 9.722e-01,1.505e-01, -2.510e-05,  0]
arr_ig= [ 1.315, -0.421, -0.0411,  0.279,  0.844,   0.0108,.0]

tbode = TwoBody(mu)
tbint = tbode.integrator(.1)

ltode = TwoBodyLT(mu, alpha)
ltig_int = ltode.integrator(.1,TanLaw(ig_throttle),range(0,6))

#########################################################################

## Integrate out ics to generate body data
dep_body_traj = tbint.integrate_dense(dep_ig,tmax,20000)
arr_body_traj = tbint.integrate_dense(arr_ig,tmax,20000)

## Load into table for vector functions
dep_body_tab = vf.InterpTable1D(dep_body_traj,tvar = 6)
arr_body_tab = vf.InterpTable1D(arr_body_traj,tvar = 6)
#########################################################################
```

Practical Example

- Integrate all ICs in parallel
- Stop integration when we cross target radius

$$f(\mathbf{y}) = |\mathbf{r}| - |\mathbf{r}_{arr}(t)\ | = 0$$

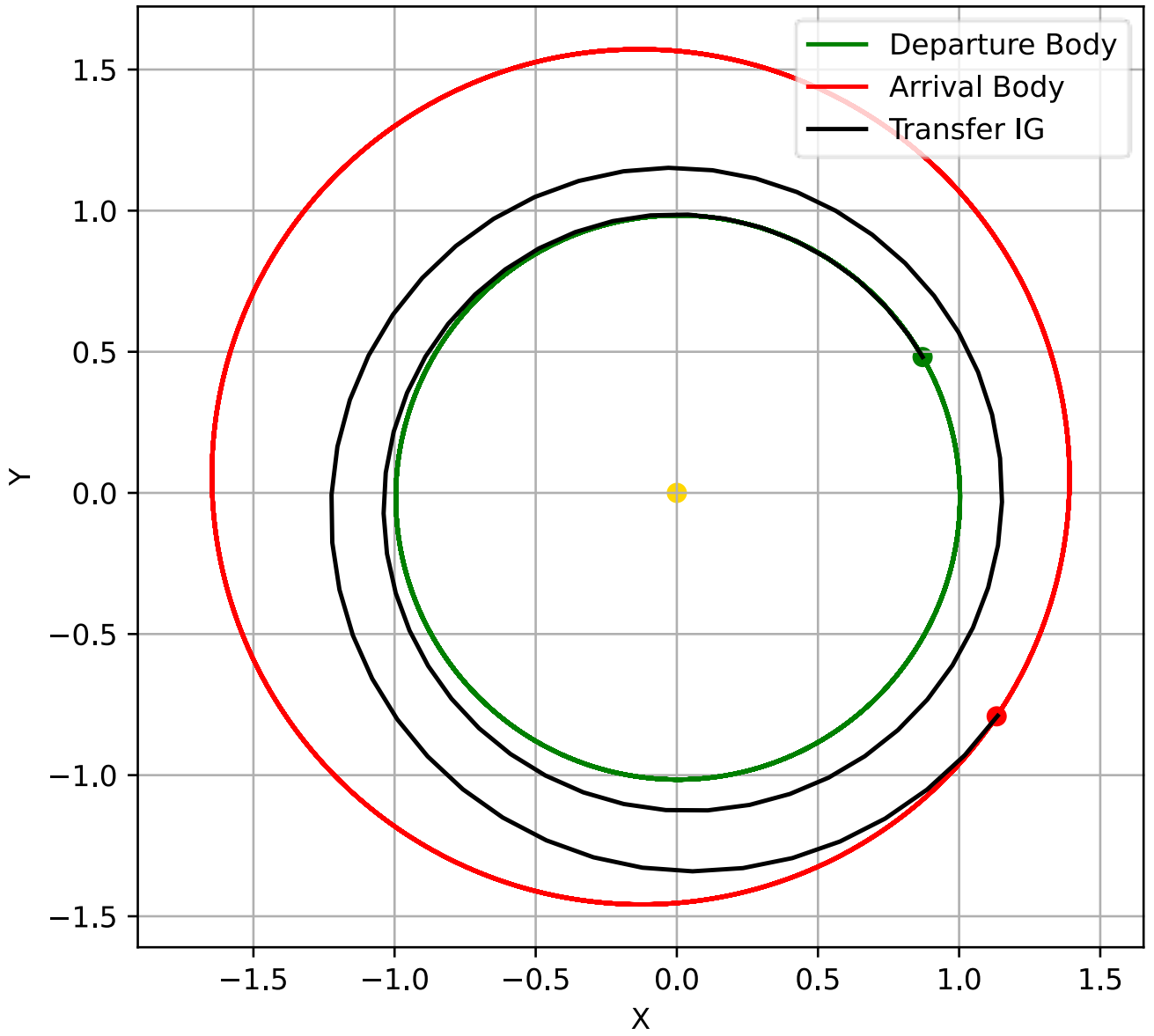
- Sort results to find closest approach

```
t0s = np.linspace(2*np.pi,8*np.pi,1000)

X0t0U0s = []

for t0 in t0s:
    X0t0U0 = np.zeros(10)
    X0t0U0[0:6] = dep_body_tab(t0)
    X0t0U0[6]=t0
    X0t0U0s.append(X0t0U0)

tfs = [tmax]*len(t0s)

nthreads = 8
trajs_events = ltig_int.integrate_dense_parallel(X0t0U0s,tfs,
                                                 [(CrossEvent(arr_body_tab),0,True)],
                                                 nthreads)

trajs = [t[0] for t in trajs_events]

def keyfunc(traj):
    Rf = traj[-1][0:3]
    Rftarg = arr_body_tab(traj[-1][6])[0:3]
    return np.linalg.norm(Rf-Rftarg)

trajs.sort(key=keyfunc)

traj = trajs[0]
```

```
def CrossEvent(targtab):
    XtU = ODEArgs(6,3)
    R = XtU.XVec().head(3)
    t = XtU.TVar()
    Rtarg = targtab(t).head(3)
    return R.norm()-Rtarg.norm()
```

Practical Example

- Obtain near rendezvous in < 100 ms



Conclusion

- See tutorial online for more in-depth details:

Tutorials - ASSET 0.4.0 documentation (alabamaasrl.github.io)