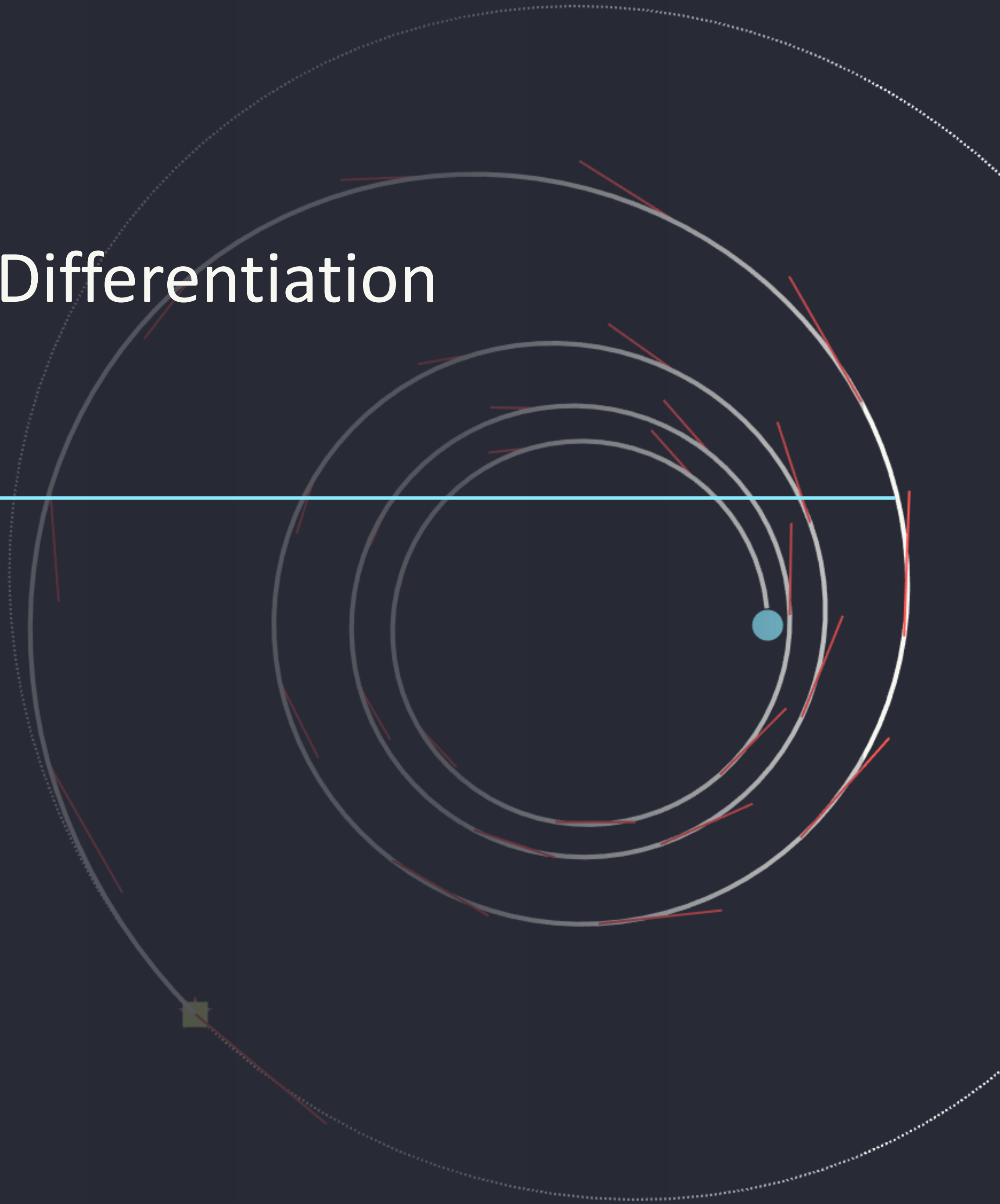
Direct Optimization using Automatic Differentiation

ADiGator Package



Burton Yale

ASE387P-6: Optimal Spacecraft Trajectories

Dr. Ryan Russell

December 11th, 2021

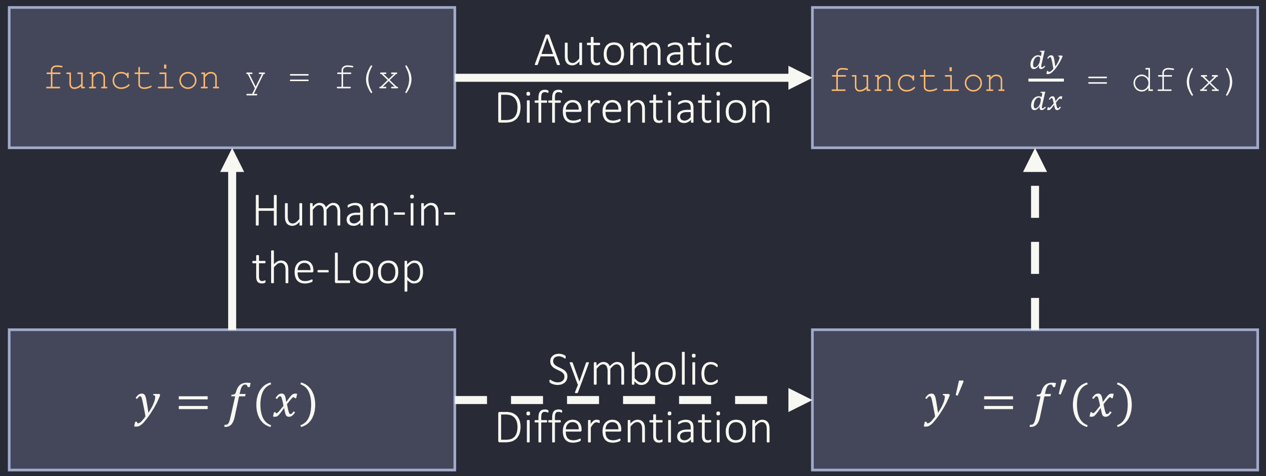
What is Automatic Differentiation?

Automatic Differentiation (AD):

- Differentiation of written code to find derivatives
- Streamlines process of finding derivatives/updating existing derivatives

Forward and Reverse Derivatives:

- Forward derivatives: $y' = f'(x)$
- Reverse derivatives: $x' = f^{(-1)'}(y)$



How AD Works: Function Overloading

- Creates new variable type
  - For `ADiGator` this is `cada()`
- Function overloading adds new methods for all functions given this variable type
- These variables are used to track operators through a function.

```
% Overloaded unary math array operations
methods

  function y = abs(x)
    % CADA overloaded ABS function
    global ADIGATOR
    if ADIGATOR.OPTIONS.COMPLEX
        y = sqrt(real(x).^2 + imag(x).^2);
    else
        y = cadaunarymath(x,1,'abs');
    end
  end

  function y = acos(x)
    % CADA overloaded ACOS function
    y = cadaunarymath(x,0,'acos');
  end

end
```

How AD Works: Successive Chain-Rules

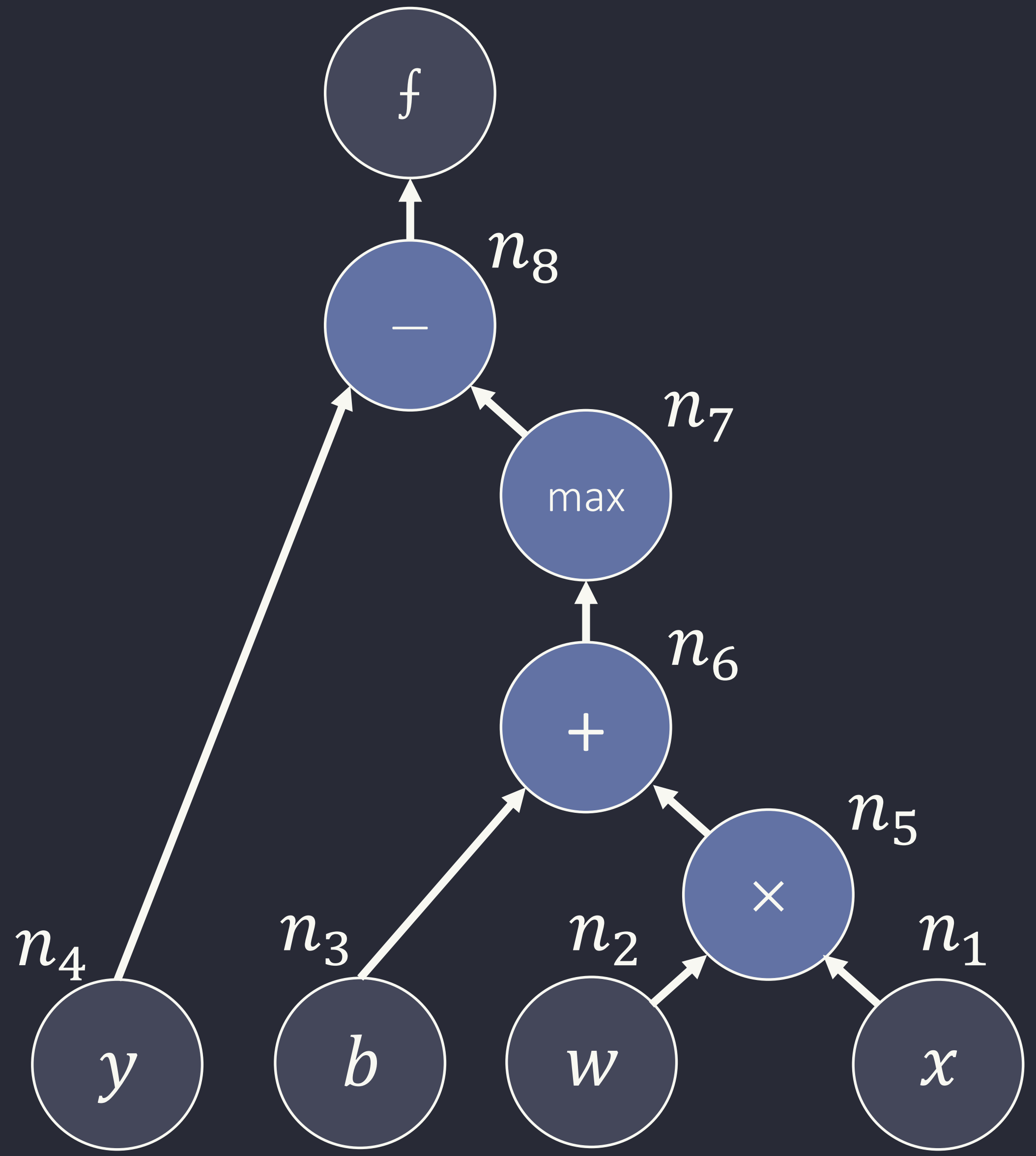
$$f(y, w, x, b) = y - \max(0, w \cdot x + b)$$

- Taking `cada()` outputs, the function calls are converted into binary trees of math operators

- Every node is used to chain derivatives from the root to variable differentiated.

- Reverse modes switch direction of chain rules

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial n_8} \times \frac{\partial n_8}{\partial n_7} \times \frac{\partial n_7}{\partial n_6} \times \frac{\partial n_6}{\partial n_5} \times \frac{\partial n_5}{\partial n_1}$$

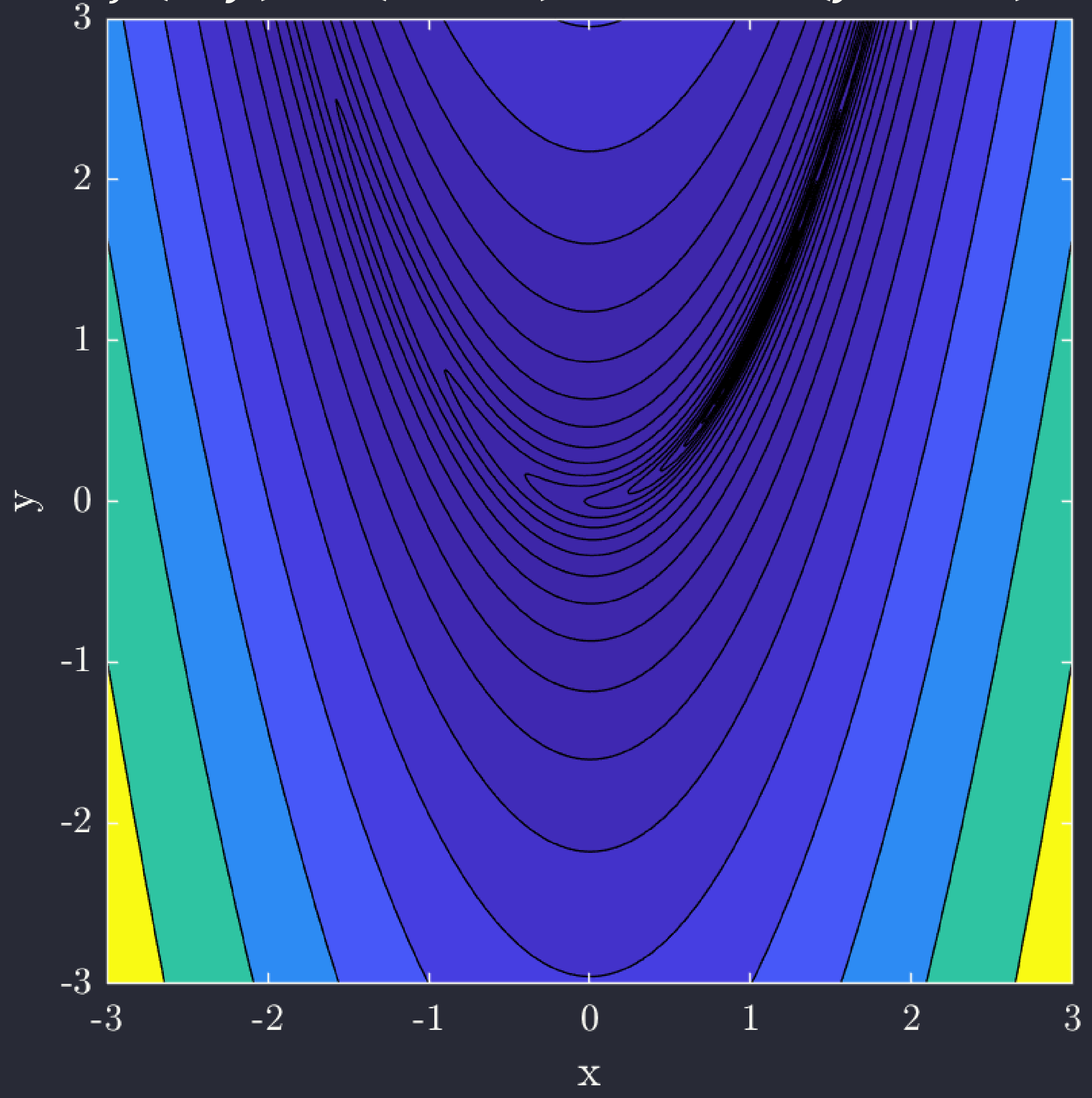


Rosenbrock: Applying AD to Simple Functions

- To evaluate the effectiveness of AD, its ability to calculate derivatives will be compared against the following
  - Symbolically Derived
  - Finite Difference Stencils
  - Complex Differentiation

- Generation time for Jacobian and Hessian scripts:
  - Jacobian: 0.5 seconds
  - Hessian: 2.2 seconds

$$f(x,y) = (1-x)^2 + 100 \cdot (y - x^2)^2$$



Rosenbrock: AD vs Symbolics vs Finite Difference

Error vs Symbolic Solution:

- AD proved to be as accurate as both Complex and Symbolic differentiation

Run Time vs Symbolic Solution

- AD solutions on average 10x slower than symbolic counterparts
- Finite Difference (FD) 10x slower still to get comparable error values for Hessians

Table 1: Error and calculation times for the Jacobian of the Rosenbrock Function

Methods	|Error|	Avg. Exe Time
Symbolics	0	2.0850e-06 sec
Complex	0	2.7643e-06 sec
FD 3pt	4.0047e-08	2.3589e-05 sec
FD 5pt	3.9738e-11	2.2243e-05 sec
FD 7pt	2.5925e-12	2.4901e-05 sec
AD	0	2.8025e-05 sec

Table 2: Error and calculation times for the Hessian of the Rosenbrock Function

Methods	|Error|	Avg. Exe Time
Symbolics	0	2.4233e-06 sec
FD* 3pt	3.7045e-08	7.3599e-05 sec
FD* 5pt	6.5612e-11	1.0546e-04 sec
FD* 7pt	9.2149e-11	1.4798e-04 sec
AD	0	2.2535e-05 sec

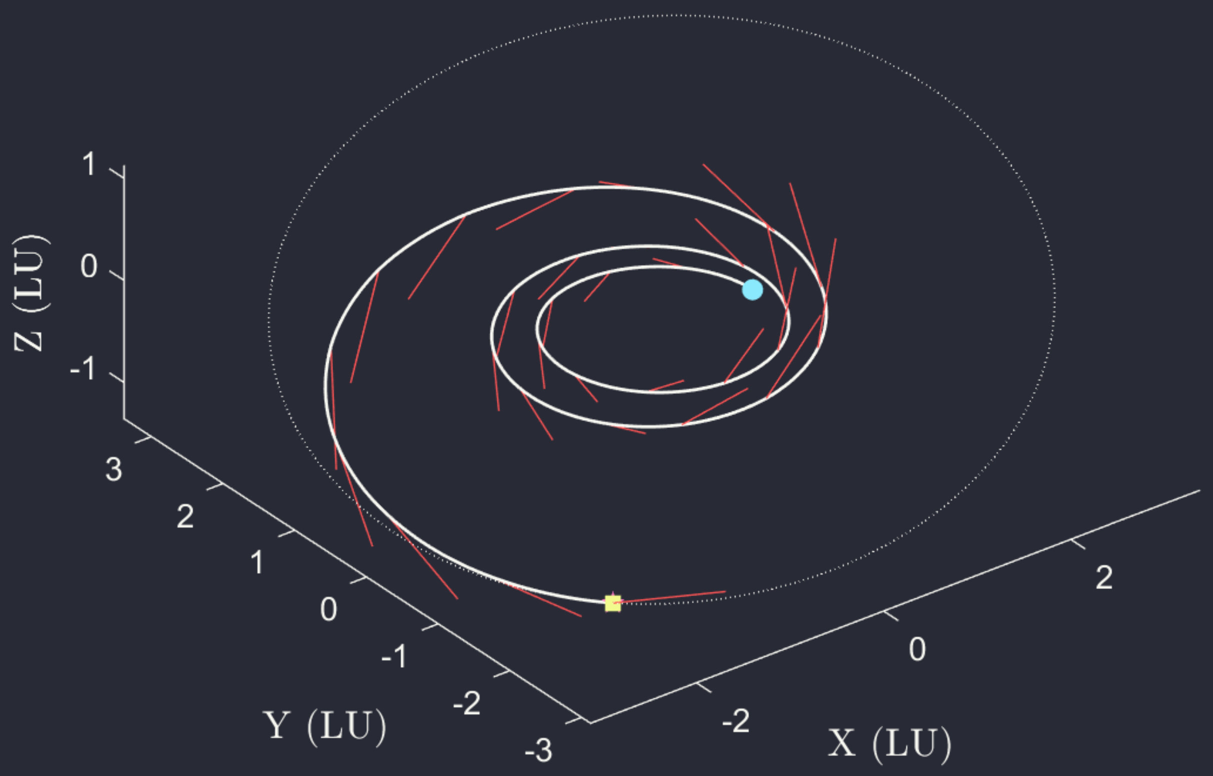
*: Jacobian calculated with complex difference. That Jacobian is then finite differenced

Direct Optimization: Problem Setup

- AD was used to generate Jacobian and Hessian files of combined objective-constraint cost function.

- Results will be compared against the same algorithm using Finite Differencing for derivative information

- FD algorithm will also use FORTRAN libraries for Kepler propagation and parallel processing for derivative calculation



Direct Optimization: Function Generation

<u>Setup</u>:

- Create MATLAB function file
- Define all inputs and derivative variables to ADiGator

<u>Limitations</u>:

- Inputs can only be Doubles
- Certain functions note supported
  - ODE Solvers
  - Switch Statements
  - Variable array lengths (!)

<u>Complexity Growth</u>:

- Jacobian File: 2200 lines
- Hessian File: 7800 lines

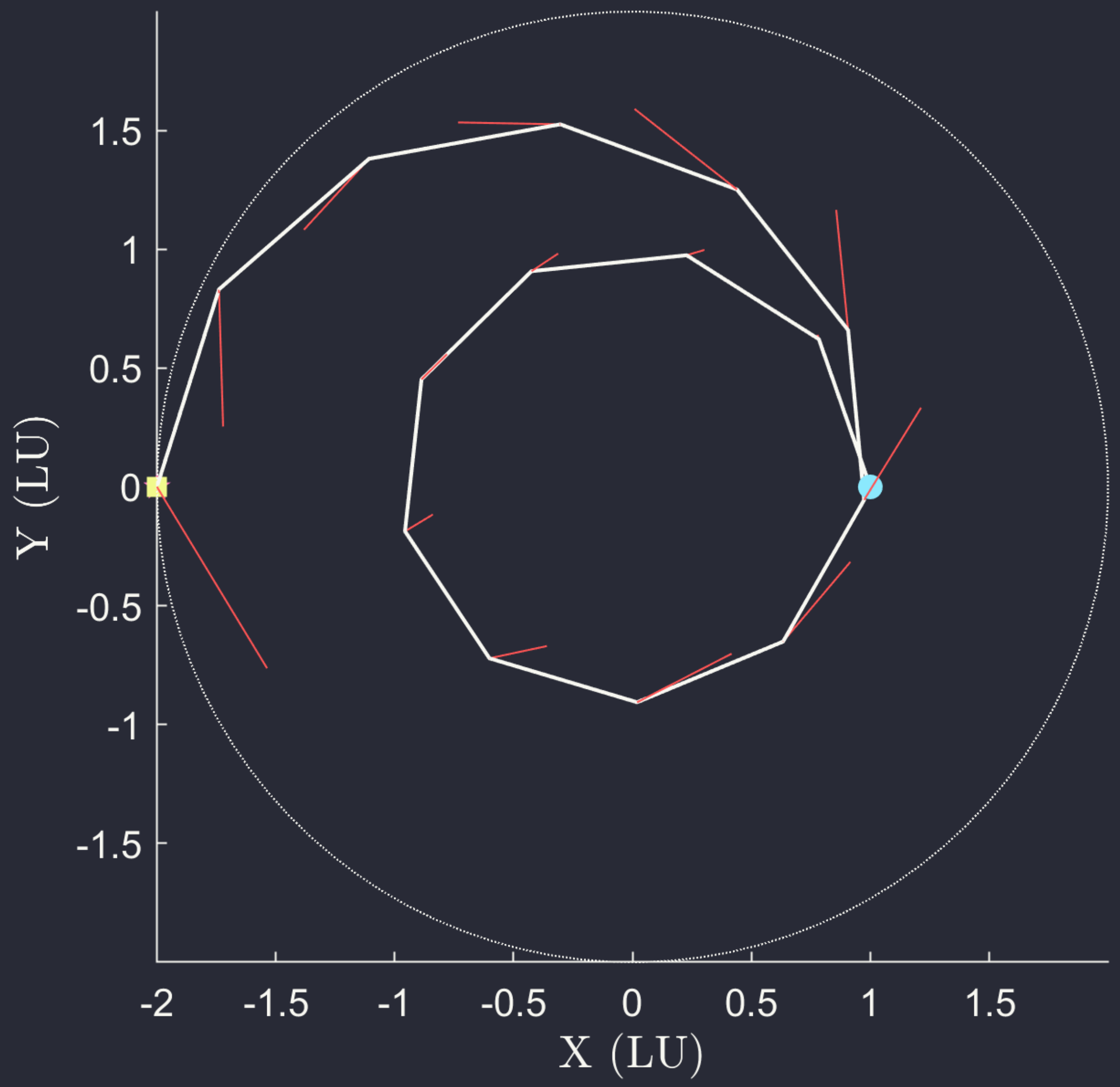
Listing 1: Setup used to create Jacobian and Hessian functions for Direct Optimizer

```
gx0 = adigatorCreateDerivInput([3*numsegs+1,1]);
ps0 = adigatorCreateAuxInput([6 1]);
plam = adigatorCreateAuxInput([sum(conLen) 1]);
pp = adigatorCreateAuxInput([sum(conLen) 1]);
pweights = adigatorCreateAuxInput([4 1]);
ponOff = adigatorCreateAuxInput([4 1]);
prtarg = adigatorCreateAuxInput([3 1]);
pvtarg = adigatorCreateAuxInput([3 1]);
ptofTarg = adigatorCreateAuxInput([1 1]);
pm0 = adigatorCreateAuxInput([1 1]);
pTbnds = adigatorCreateAuxInput([2 1]);
pg0IspInv = adigatorCreateAuxInput([1 1]);
pnumsegs = adigatorCreateAuxInput([1 1]);
pOptType = adigatorCreateAuxInput([1 1]);
pkhomo = adigatorCreateAuxInput([1 1]);
pmu = adigatorCreateAuxInput([1 1]);
adigatorGenJacFile('cost.m',{gx0, ps0, plam, ...
   pp, eqLen, ineqLen, conLen, pweights, ...
   ponOff, prtarg, pvtarg, ptofTarg, ...
   pm0, pTbnds, pg0IspInv, ...
   numsegs, pOptType, pkhomo, pmu})
```

Direct Optimization: 15 Segment – HW4 Problem

<u>Run Settings</u>:

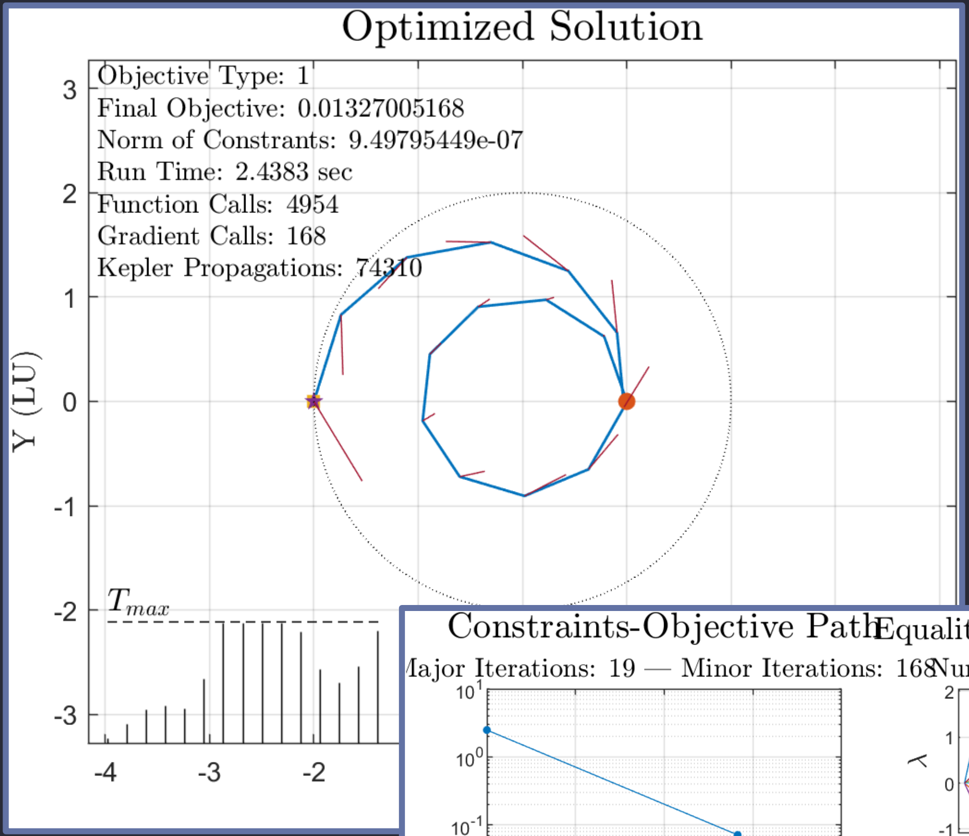
- $r^* = [-2, 0, 0]$
- $v^* = \left[0, -\sqrt{1/2}, 0\right]$
- $\delta UV = 30°$
- $ToF = 12\ TU$
- Number of Segments: 15
- Objective: $\phi = \sum_i^{N_{seg}} \|\Delta v_i\|^2$

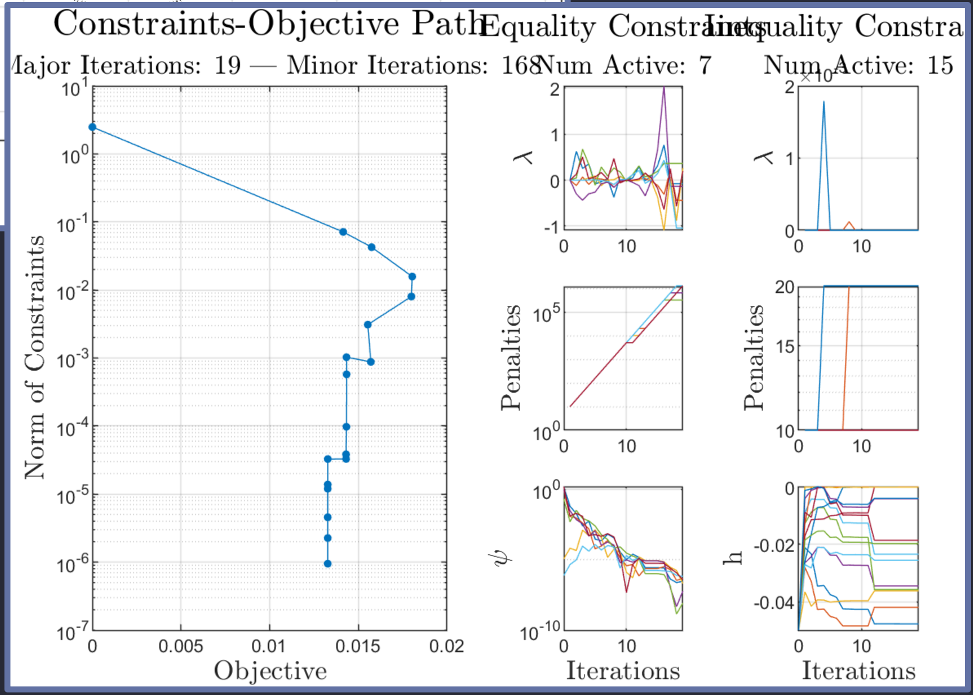


Direct Optimization: 15 Segment – HW4 Problem

Results:

Category	AD	FD
Run Time	2.4283 sec	8.747 sec
Final Obj.	0.01327005	0.01296366
Final Const	9.4979e-07	1.2728e-06
Func Calls	4954	16561
Grad Calls	168	134
Kepler Prop	74310	497370



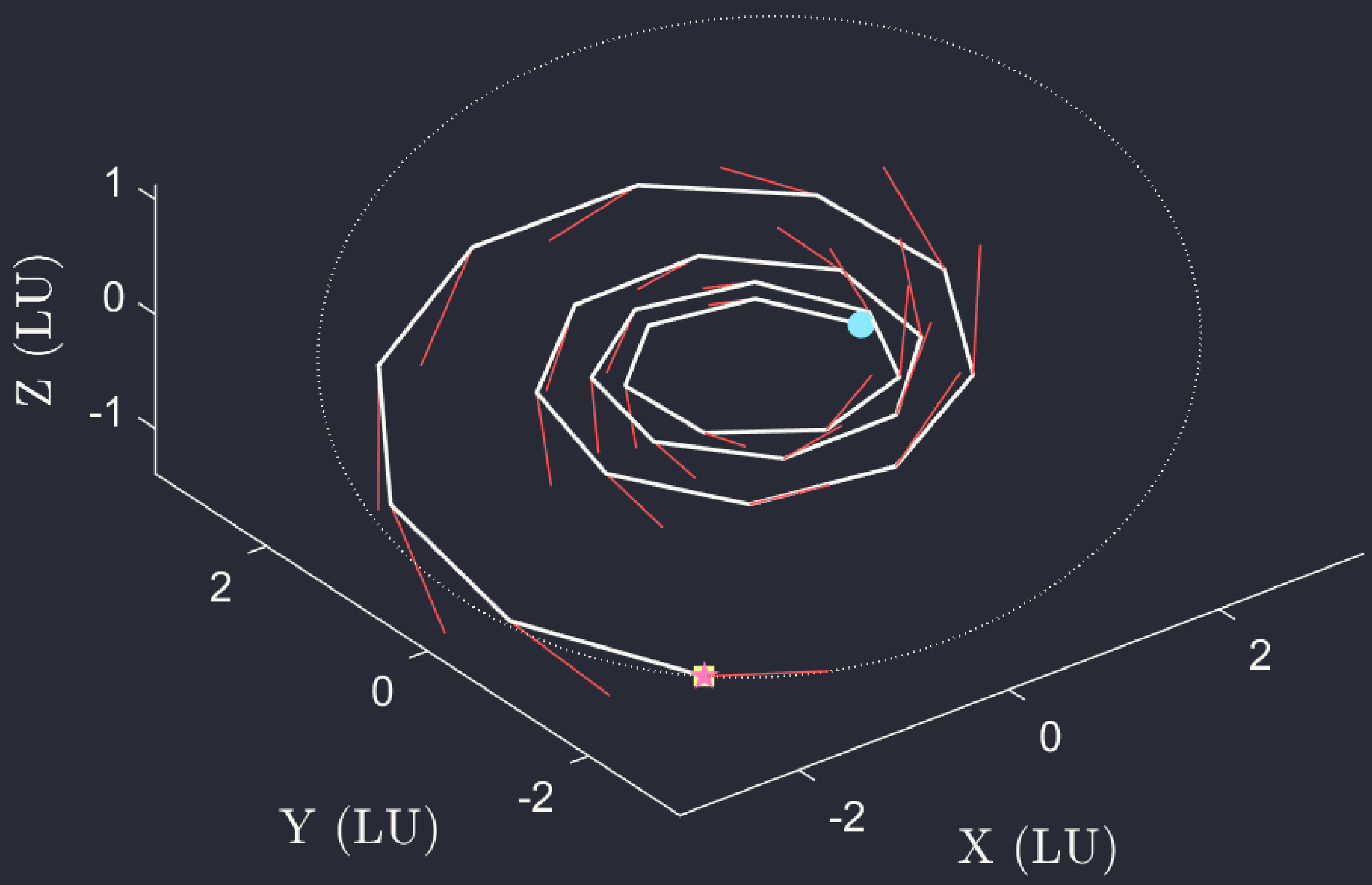


AD Direct Optimization Results

Direct Optimization: 30 Segment – Multi-Rev

Run Settings:

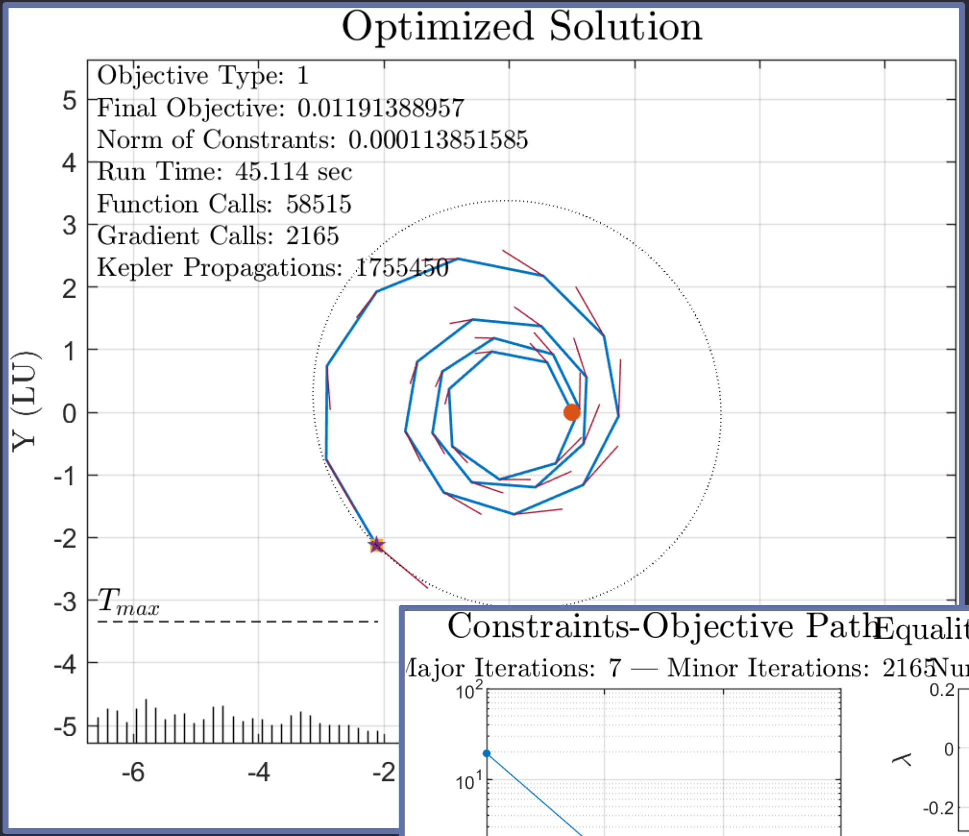
- $r^* = [3\cos\theta, 3\sin\theta, -0.5]$
- $v^* = \sqrt{\frac{\mu}{\|r^*\|}}[-\sin\theta, \cos\theta, 0]$
- $\theta = 225°$
- $\delta UV = 50°$
- $ToF = 45\, TU$
- Number of Segments: 30
- Objective: $\phi = \sum_i^{N_{seg}} \|\Delta v_i\|^2$

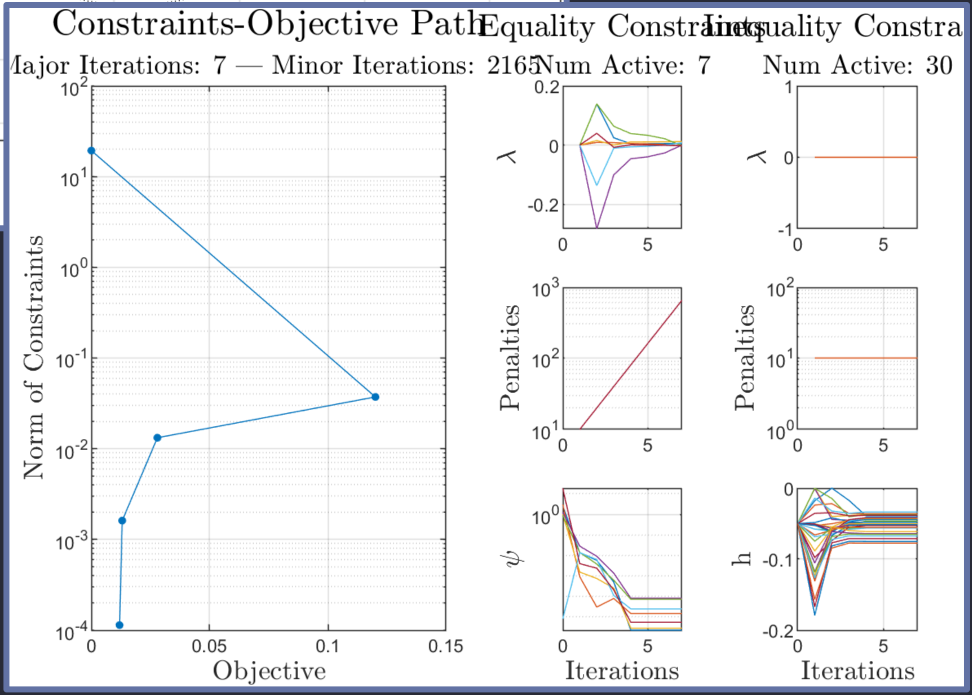


Direct Optimization: 30 Segment – Multi-Rev

Results:

Category	AD	FD
Run Time	45.11 sec	DNF cutoff at 10 min
Final Obj.	0.011913	DNE 9e-9 at cutoff
Final Const	1.1385e-4	DNE 0.063 at cutoff
Func Calls	58515	912624
Grad Calls	2165	4224
Kepler Prop	1755450	54758640

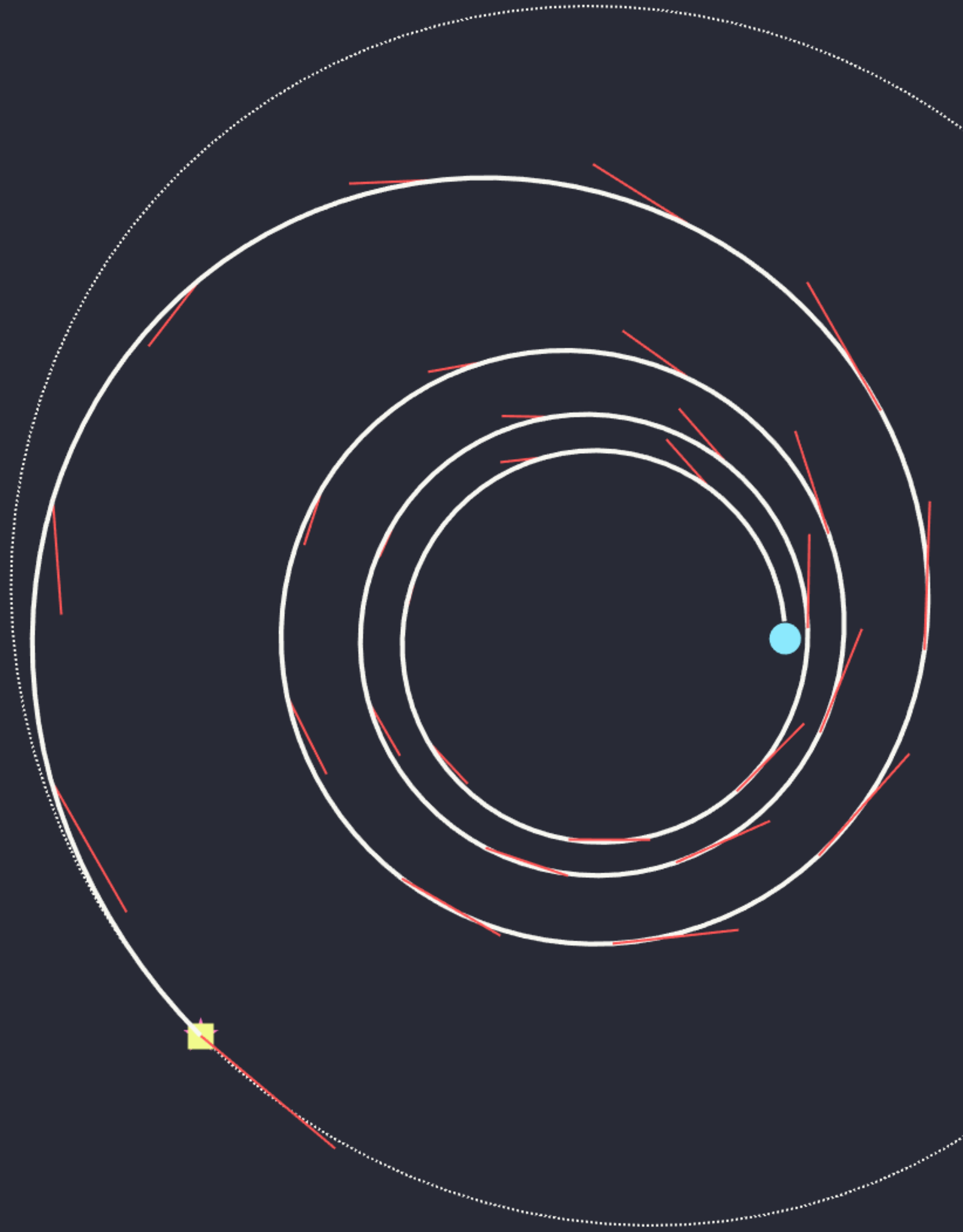




AD Direct Optimization Results

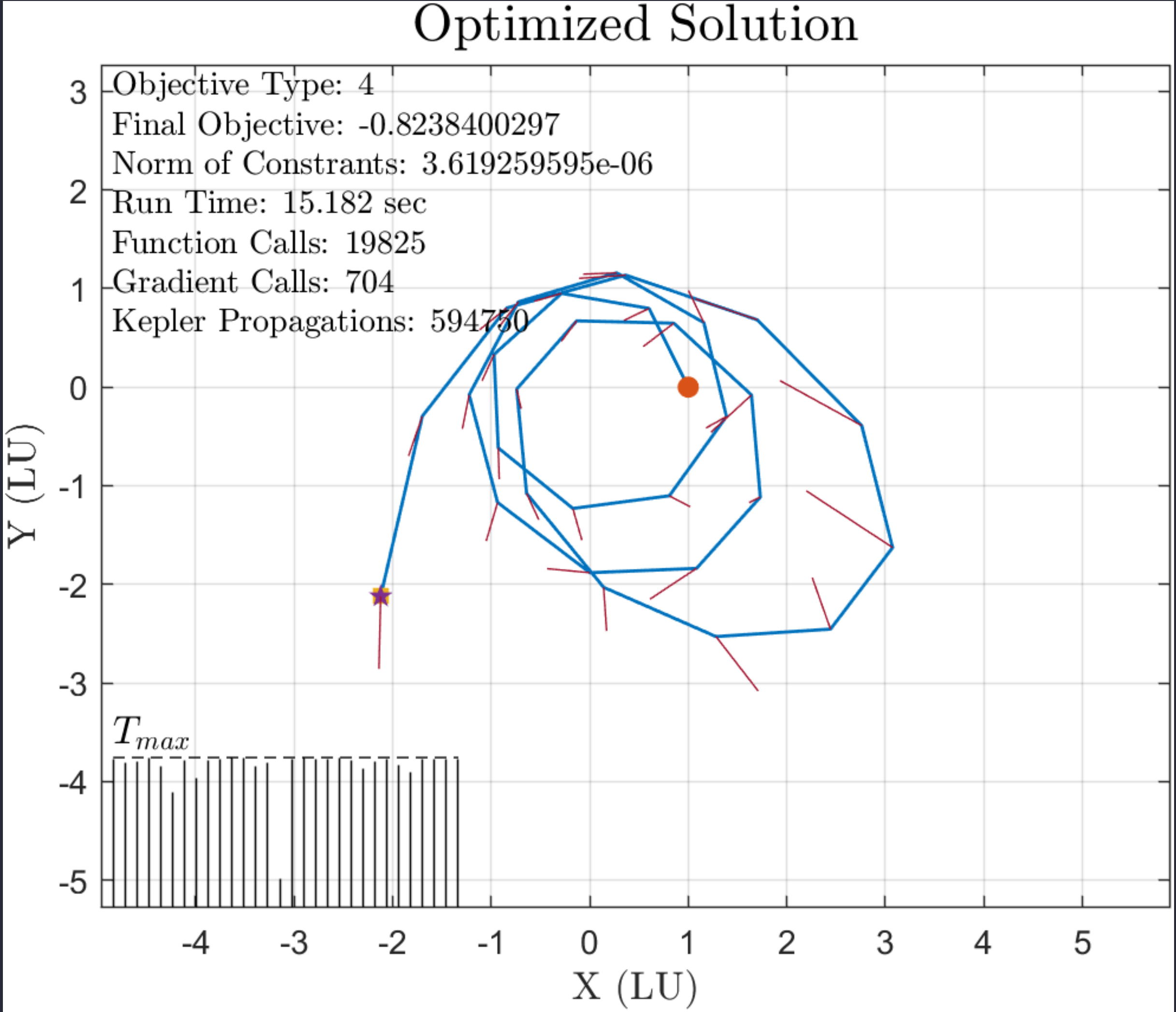
Overall Results

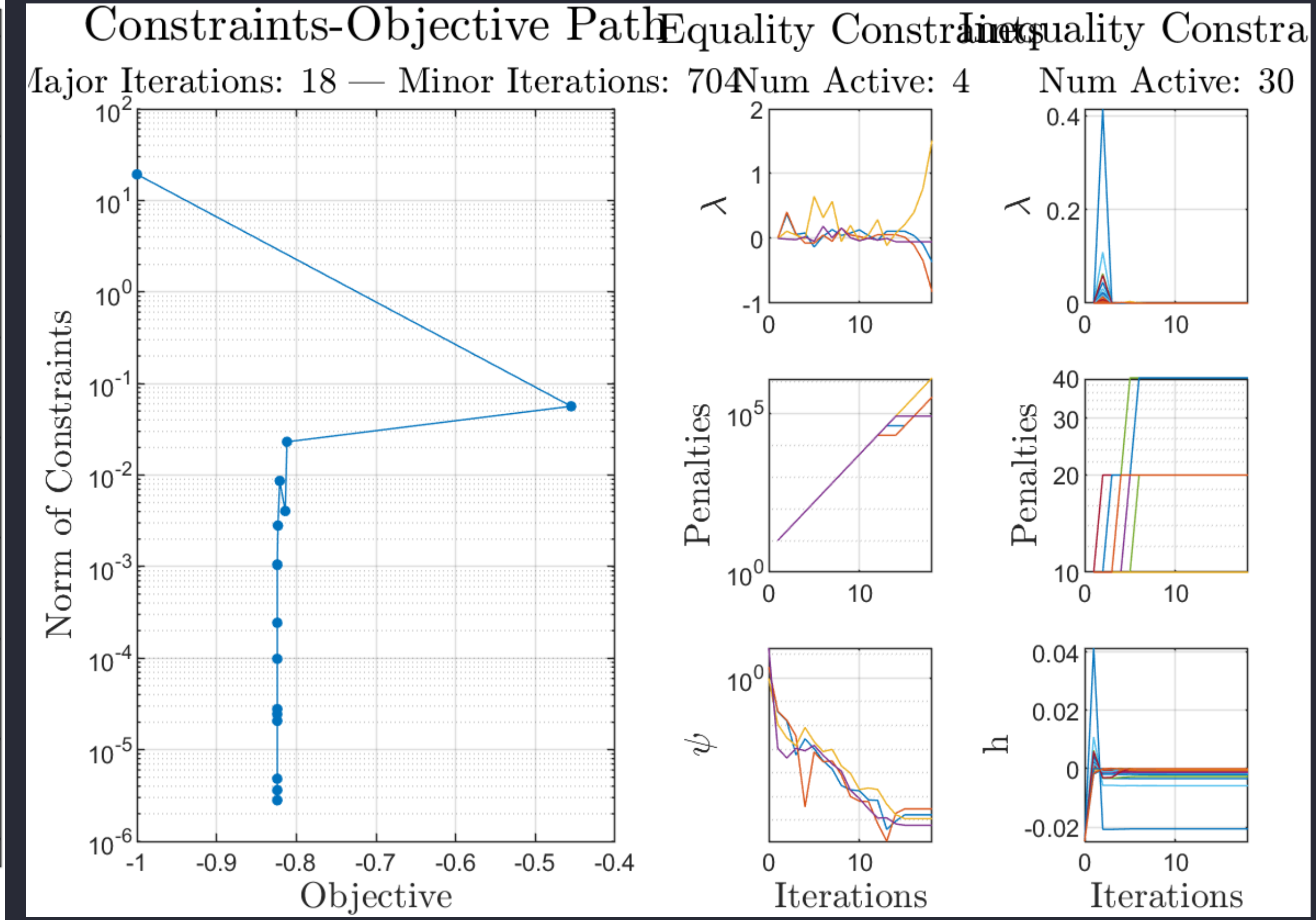
- AD has proven to be a robust method of calculating derivatives, more accurately than FD methods (apart from complex differentiation)
- This pure MATLAB approach provided a greater than 4x speed improvement over similar code using parallel processing and FORTRAN libraries
- AD's limitations can make it impossible to implement in edge cases, but similar applies to helpful methods such as complex differentiation



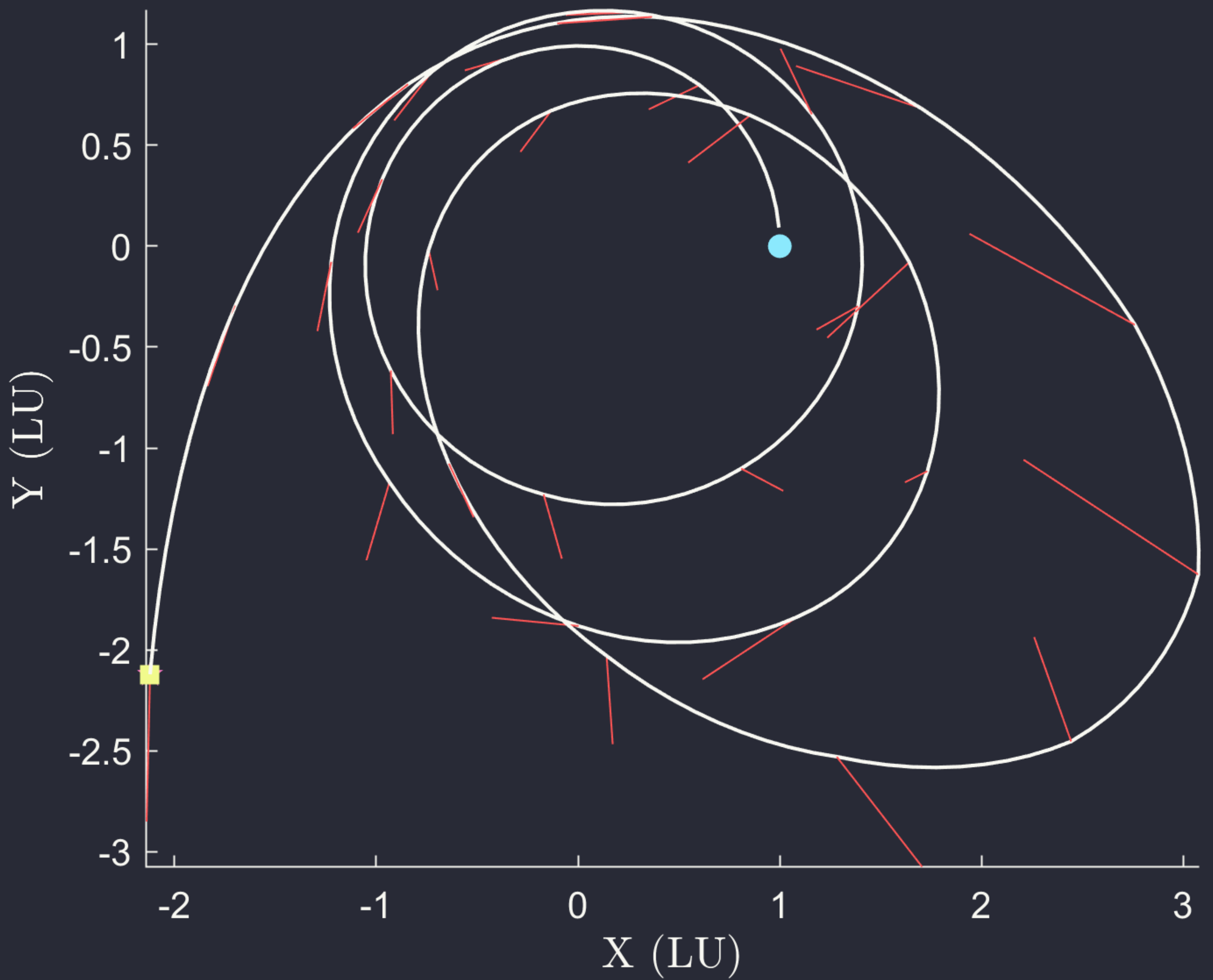
Additional Results Appendix

30 Segment – Multi-Rev: Maximizing v_f





30 Segment – Multi-Rev: Maximizing v_f



30 Segment – Multi-Rev: Minimizing ToF