Extremal Field Map User Guide

Robyn Woollands & John Junkins
Texas A&M University
(robyn.woollands@gmail.com)

May 2016

Introduction

The Extremal Field Map (EFM) tool is was developed for trajectory design studies to allow optimal two-impulse orbit transfers to be determined for objects orbiting in the vicinity of a large body (i.e. two-body or Keplerian dynamics). The EFM tool makes use of the Battin/Prussing approach [1,2] for solving the elliptic Keplerian Lambert problem. Parabolic and hyperbolic orbits are not considered in this algorithm. The EFM tool is coded in MATLAB and offers the user an interactive plotting feature for investigating the ΔV cost for various orbit transfers. A version of this tool also exists in C/C++ and this accommodates user specified high fidelity force models for solving the perturbed Lambert problem to high precision. This tool is named the Unified Lambert Tool (ULT) and is implemented in serial as well as parallel using Message Passing Interface. More details on both EFM and ULT can be found in references [3,4].

Generating Extremal Field Maps

To generate an EFM run the following command in the MATLAB command window from inside the EFM directory. An example input file (EFM_example1.m) is given for generating an EFM between two low Earth orbits. The run time may vary from seconds up to a few minutes and depends on the size and resolution of the EFM specified by the user. During the run some information is printed to screen to inform the user of progress towards completion.

```
>> EFM_generate('EFM_example1')
```

To use the interactive plotting tool run the following command in the MATLAB command window from inside the EFM directory. It may take a few seconds for the data to load before the graphical user interface (GUI) is displayed. More details regarding the GUI are given below (see Figure 1).

```
>> EFM_plot('EFM_example1')
```

The GUI consists of six figures. The top three are (from left to right) the number of mathematically possible solutions, the number of physically feasible solutions and the ΔV cost EFM. When the window opens only these three figures will be visible. The user may click/select on any point in each of these figures and this will result in the corresponding trajectories for that data point being plotted in the bottom three figures. The bottom three figures are (from left to right) the

mathematically possible transfers corresponding to the selected data point, the physically feasible transfers corresponding to the selected data point, and the minimum ΔV cost of these transfers from the ΔV cost EFM. The trajectory corresponding to the EFM global minimum (magenta circle/cross) is also shown (magenta) on the bottom right figure in the GUI.

Note that the legends for the figures titled "Possible Transfer Trajectories" and "Feasible Transfer Trajectories" show the number of revolutions (e.g. N0, N1 etc.) and the branch (U=upper, L=lower). For example, N2U is a transfer trajectory that spans a true anomaly angle between 0 and 6pi (2 and a fraction orbits) and resides on the upper branch. More details on the number of revolutions and the upper/lower branches are discussed by Prussing [2].

Each time a new point is selected a black/white circle/cross marker appears on all of the top plots and remains for the duration that the GUI is running. The trajectories corresponding to the most recently selected point are always displayed in the bottom three plots i.e. they are updated every time a new point is selected in the EFMs. The minimum ΔV (km/s) for a particular selected point is displayed in the legend in the bottom right figure. Cntrl C in the command window or closing the GUI ends the program.

Example 1 – EFM between two planar LEO orbits

Below is an example file that the user can edit to set the departure and arrival orbit elements. The GUI corresponding to this example test case shown in Figure 1.

```
% EFM_example1.m
% AUTHOR:            Robyn Woollands (robyn.woollands@gmail.com)
% DATE WRITTEN:      May, 2016
% LAST MODIFIED:     May, 2016
% AFFILIATION:       Department of Aerospace Engineering, Texas A & M University, College Station,
TX
%
% DESCRIPTION:       User input for generating Keplerian extremal field maps between two orbits.

clearvars -except filename
close all

consts;                         % Load Constants

setplot = 0;                    % Output figures: NO = 0 (recommended) & YES = 1 (for debugging).

tol     = 1e-10;                % Set tolerance

% Departure Orbit Elements
DepA    = 10000;                % Semimajoe Axis (km)
DepE    = 0.05;                 % Eccentricity
DepI    = 0*pi/180;             % Inclination (rad)
DepOm   = 0*pi/180;             % Right Ascension of Ascending Node (rad)
DepW    = 0*pi/180;             % Argument of Perigee (rad)
DepM0   = 0*pi/180;             % Mean Anomaly (rad)

% Arrival Orbit Elements
ArrA    = 15000;                % Semimajor Axis (km)
ArrE    = 0.1;                  % Eccentricity
ArrI    = 0*pi/180;             % Inclination (rad)
ArrOm   = 0*pi/180;             % Right Ascension of Ascending Node (rad)
ArrW    = 0*pi/180;             % Argument of Perigee (rad)
ArrM0   = 0*pi/180;             % Mean Anomaly (rad)
```

```
% Extremal Field Map Specifics
DimH    = 6;                % Horizontal dimension of EFM (Number of departure orbit periods)
DimV    = 6;                % Vertical dimension of EFM (Number of departure orbit periods)
EFMres  = 150;              % EFM resolution. Number of points along x axis.
```

Example 2 – EFM between two inclined LEO orbits

Below is an example file that the user can edit to set the departure and arrival orbit elements. The only different between these two examples is that the arrival orbit is inclined 40 degrees with respect to the departure orbit. The GUI corresponding to this example test case shown in Figure 2.

```
% EFM_example2.m
% AUTHOR:           Robyn Woollands (robyn.woollands@gmail.com)
% DATE WRITTEN:     May, 2016
% LAST MODIFIED:    May, 2016
% AFFILIATION:      Department of Aerospace Engineering, Texas A & M University, College Station,
TX
%
% DESCRIPTION:      User input for generating Keplerian extremal field maps between two orbits.

clearvars -except filename
close all

consts;                     % Load Constants

setplot = 0;                % Output figures: NO = 0 (recommended) & YES = 1 (for debugging).

tol     = 1e-10;            % Set tolerance

% Departure Orbit Elements
DepA    = 10000;            % Semimajoe Axis (km)
DepE    = 0.05;             % Eccentricity
DepI    = 0*pi/180;         % Inclination (rad)
DepOm   = 0*pi/180;         % Right Ascension of Ascending Node (rad)
DepW    = 0*pi/180;         % Argument of Perigee (rad)
DepM0   = 0*pi/180;         % Mean Anomaly (rad)

% Arrival Orbit Elements
ArrA    = 15000;            % Semimajor Axis (km)
ArrE    = 0.1;              % Eccentricity
ArrI    = 40*pi/180;        % Inclination (rad)
ArrOm   = 0*pi/180;         % Right Ascension of Ascending Node (rad)
ArrW    = 0*pi/180;         % Argument of Perigee (rad)
ArrM0   = 0*pi/180;         % Mean Anomaly (rad)

% Extremal Field Map Specifics
DimH    = 6;                % Horizontal dimension of EFM (Number of departure orbit periods)
DimV    = 6;                % Vertical dimension of EFM (Number of departure orbit periods)
EFMres  = 150;              % EFM resolution. Number of points along x axis.
```

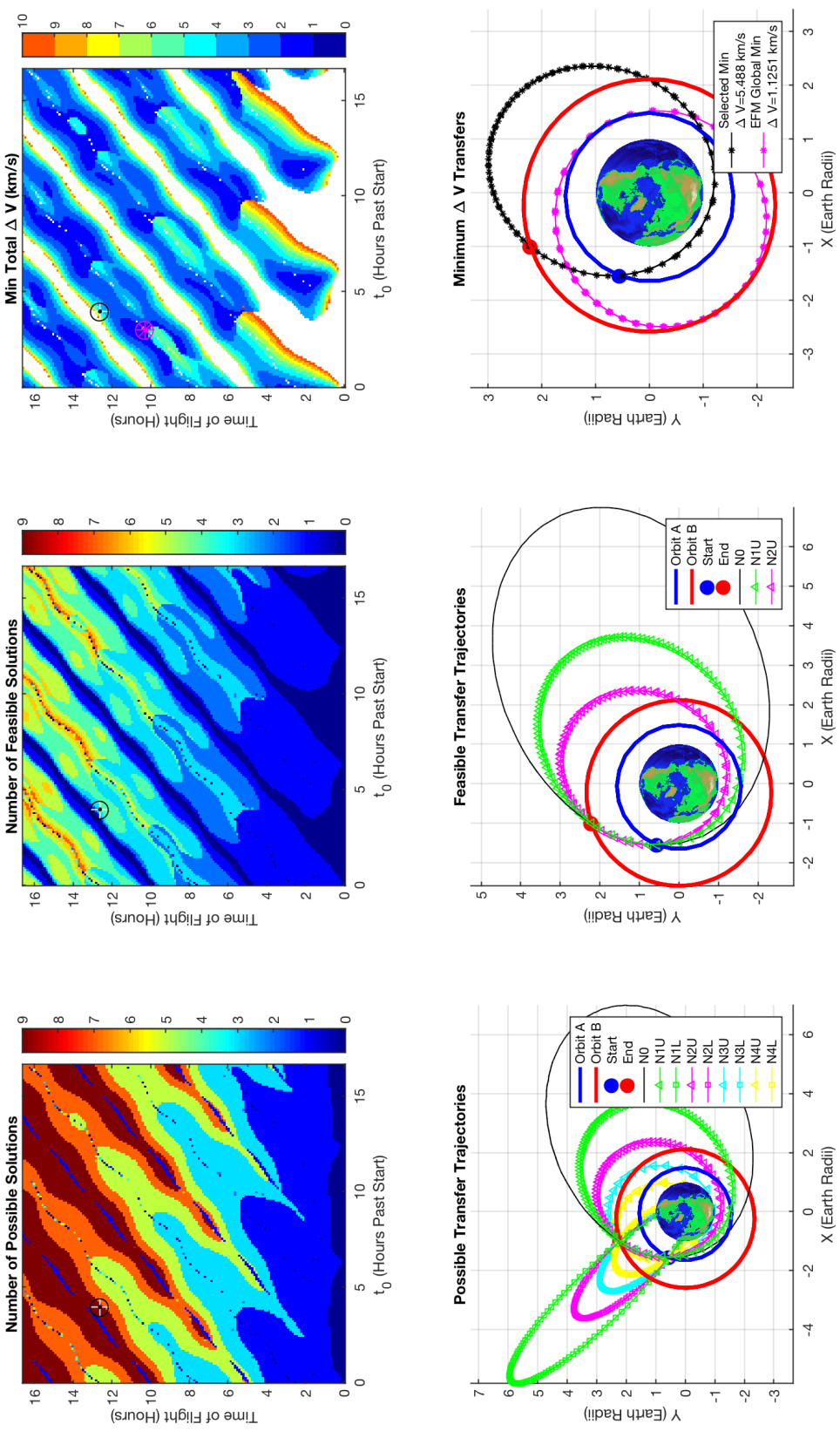

Figure 1: Interactive Plotting Window for example 1.

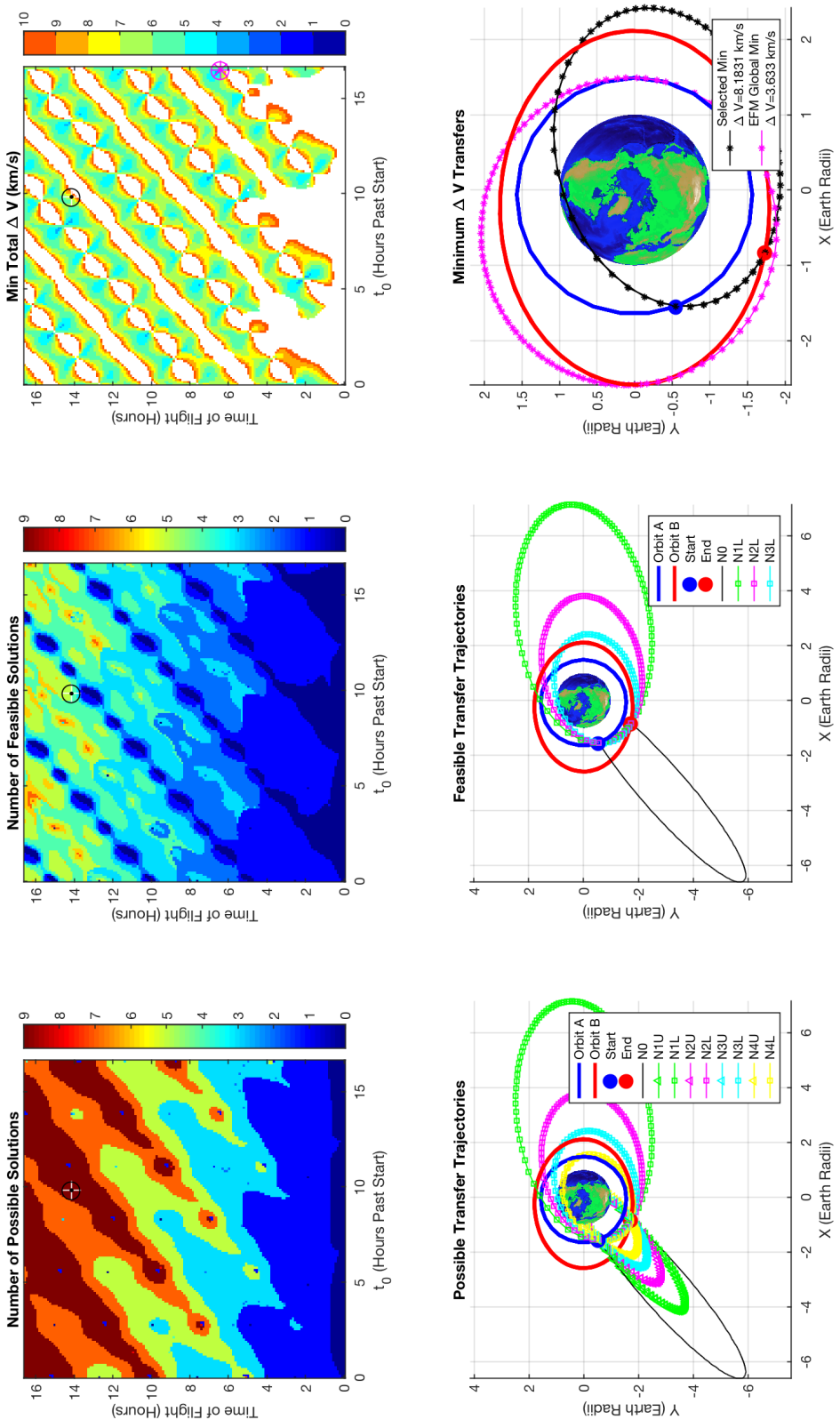

Figure 2: Interactive Plotting Window for example 2.

Code Structure & Functions

EFM_generate.m

All computations are anchored back to this main function. The function loops through the transfer "time-of-flight" (y-axis) and "start time" (x-axis) for computing all the orbit transfer information that is required for generating the EFM. All other functions (except EFM_plot.m) are called from this function or from sub-functions within this function.

consts.m

Loads some astrodynamics constants and also specifies the canonical distance and time units. All the computations are done using canonical units.

generate_orbits.m

This function takes the information specified by the user in ***EFM_example1.m*** and generates the departure and arrival orbits and the points along the orbit that will produce an EFM with the user specified resolution.

> ***Sub-functions*** *(these are also used elsewhere in the code)*
>
> ***elm2rv.m***
>
> This function converts classical orbit elements to Cartesian position and velocity.
>
> ***FnG.m***
>
> The function computes the analytical F&G solution.
>
> ***kepler.m***
>
> This function solves Keopler's equation.

initialize.m

Initializes some variables for setting up the solution to Lambert's problem.

t_min_energy.m

This function computes the time-of-flight for the minimum energy transfer orbit.

t_parabolic.m

This function computes the time-of-flight for a parabolic orbit, which must be less than the desired transfer time for a solution to be computed.

n_max_newton.m

Determines the maximum number of possible revolutions over which Lambert's problem can be solved considering the user specified parameters. This function uses Newton's method, as specified by Prussing's paper [2]. If Newton's method fails then the bisection method (n_max_bisection.m) is used as a second attempt for solving the problem.

n_max_bisection.m

Determines the maximum number of possible revolutions over which Lambert's problem can be solved considering the user specified parameters. The bisection method is used only if Newton's method (n_max_newton.m) fails to converge.

a_transfer_newton.m
This function solves for the semimajor axis of the transfer orbit using Newton's method, as outlined in Prussing's paper [2]. If Newton's method fails then the bisection method (a_transfer_bisection.m) is used as a second attempt for solving the problem.

a_transfer_bisection.m
This function solves for the semimajor axis of the transfer orbit using the bisection method. This method is only utilized if Newton's method (a_transfer_newton.m) failed to converge.

velocity.m
This function computes the initial and final velocity of the transfer orbit.

retrograde.m
This function computes the z-component of the angular momentum vector. If it is negative then the orbit transfer was retrograde and the algorithm reruns the case using "theta = 2pi – theta" (see [2] for the definition of theta). Both the prograde and retrograde transfer orbits are save and the one that requires the minimum ΔV is plotted in the EFM.

feasibility.m
This function checks each simulated transfer to determine if the transfer is physically possible. For example, trajectories that pass through the Earth are infeasible and are discarded.

plot0_inform_user.m, plot1_inform_user, plot2_inform_user.m, plot3_inform_user.m
These files are mainly used for debugging and allow the trajectories to be plotted for each new transfer time while EFM_generate.m is being run. These dramatically increase the computation and should only be used for debugging small sections of an EFM. The default is "setplot = 0" in EFM_example1.m input file.

store_data.m
This file build data arrays as new data becomes available during the run.

store_data.m
This file save the data arrays at the end of the run.

EFM_plot.m
This loads the data saved from EFM_generate and displays the GUI to allow the use to study different orbit transfers.

References

1. Battin, R., “An introduction to the mechanics and methods of astrodynamics”, AIAA education series, Restin, Virginia, 1999.

2. Ochoa, S., Prussing, J., “Multiple revolution solutions to Lambert’s problem”, AAS/AIAA spaceflight mechanics meeting, Kissimmee, 1992.

3. Woollands, R., Read, J., Hernandez, K., Probe, A., Junkins, J., “Unified Lambert Tool for Massively Parallel Applications in Space Situational Awareness”, Journal of Astronautical Sciences, accepted 2017.

4. Woollands, R., “Regularization and Computational Methods of Precise Solution of Perturbed Orbit Transfer Problems”, PhD Dissertation, Texas A&M University, College Station, Texas, 2016.