Lagrange MFD v1.5 Documentation

Based on the original concept and algorithms of Keith "Keithth G" Gelling
Credits to Dimitris "dgatsolis" Gatsoulis, Brian "Brian J" Jones, ZT "Nicholas Kang" Kang for testing support.
Credit to Szymon "enjo" Ender for ModuleMessagingExt co-developent, for MFDButton framework, and for awesome technical advice over the years.

Revision History

V1.0 April 13th 2017

- Original Release

V1.1 June 18th 2017 – Feature release:

- LP focus mode … allowing you to focus at the LP.
- Focus center lock mode … locks the focus point in the center of the MFD
- Focus scale lock mode … locks the current scale across all three PRJ axes
- Focus rot lock mode … allows display in a rotational frame of reference, with the Major and Minor entities aligned with each other. (I.e. the Moon hangs on the same horizontal level as Earth, rather than rotating).
- Orbit Plot is now variable, from 10 to 10000.

V1.5 November 19th 2017 – Significant feature release for Halo and Lissajous and Halo orbit visualizations

- Added Plot/No Plot toggles for Sun, Earth, Moon (so you can remove orbits if you wish)
- Fixed bug in rotating frame visualization in other planes than X-Y
- Allow Zoom in Scale-Lock mode (does a one-off reset of the scale to the new zoom)
- Added ability to rotate the visualization of the orbit to study the orbit from different angles

Introduction to Lagrange Points

Lagrange points (also called Lagrangian points, or libration points, or just LP), are interesting points of gravitational balance between a major body (e.g. Sun, Earth) and an orbiting minor body (e.g. Earth around the sun, Moon around the Earth). There are five Lagrangian points for each Major-Minor system. For example, in this diagram, we see the Sun-Earth Lagrange points:

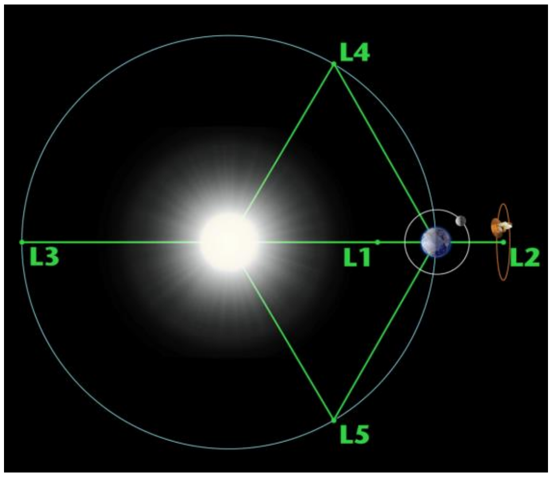


Diagram 1: From Space.com, credit NASA / WMAP

Sun-Earth L1 (SEL1) is a point in a lower orbit than Earth, and Sun-Earth L2 (SEL2) is a point at a correspondingly higher orbit than Earth. Ordinarily, for vessels to fly in these orbits, they would have a different orbital period than the Earth, to offset the different gravitational force from the Sun. At SEL1 and SEL2, the effect of the Earth's gravity is to slightly reduce the net Sun-Earth gravity at SEL1, and to slightly increase it at SEL2. The effect is to get to a balance where a vessel can have the same orbital period as the Earth – i.e. stay aligned with the Earth either in front of Earth or behind Earth. SEL1 is currently home to the Solar and Heliospheric Observatory (SOHO), whilst SEL2 is a great place for deep space telescopes, including the European Space Agency's Gaia space observatory now, and the new James Webb Space Telescope will join Gaia in 2018.

Sun-Earth L3 (SEL3) is a technical Lagrange point, without much practical use. It orbits 180 degrees ahead of Earth, and benefits from a tiny additional gravitational pull from the Earth to allow it to be marginally further out than Earth's orbit, but practically identical.

SEL1, SEL2, and SEL3 are dynamically unstable points, meaning that for vessels at these locations, course corrections are needed to stay on station. A relatively small delta-Velocity (e.g. 1 m/s) will take the vessel away from the Lagrange point.

Sun-Earth L4 and L5 (SEL4, SEL5) are points at 60 degrees ahead and behind of Earth's orbital position. The effect of the Earth's gravity on these points is to act like a stabilizer, making SEL4 and SEL5 somewhat gravitationally stable. In these locations, a small perturbation away from the Lagrange point generates a gravitational force that acts to correct the error. As a result, small "Trojan" asteroids gather at these points, trapped by this celestial gravity phenomenon.

SEL4 and SEL5 are interesting points to see our Sun from multiple angles. The Solar Terrestrial Relations Observatories, STEREO A and B are doing this today, at these 2 points.

Introduction to Lagrange MFD

Would you like to be able to fly to these Lagrange points, and explore the intricate and chaotic orbits near to these points, then I present to you: Lagrange MFD.

If you are used to seeing elliptical or circular orbits around regular gravity fields, then the orbits around these Lagrangian points will be a great surprise to you, as they look truly bizarre. (E.g. loops and swirls, with chaotic turns away from the Lagrange point, spiraling towards e.g. the Earth or the Moon).

Lagrange MFD has five modes:

1. The Orbit mode, which is used to plot the trajectories of the vessel, the celestial bodies, and the Lagrange point.
2. The Encounter mode, showing detailed analysis of current and predicted Lagrange encounter points
3. The Plan mode, used to determine the effect of a future maneuver for our vessel.
4. The Autopilot mode, for accurate burn alignment and execution, and to remain at the LP once arrived.
5. The S4I mode, named after the "4th Order Symplectic Integrator" which forms the core of the prediction engine in Lagrange MFD. This mode allows you to adjust the duration and resolution of the trajectory prediction, and to generate various diagnostic debugs to allow further Excel or Matlab analysis of the predictions.

Installation

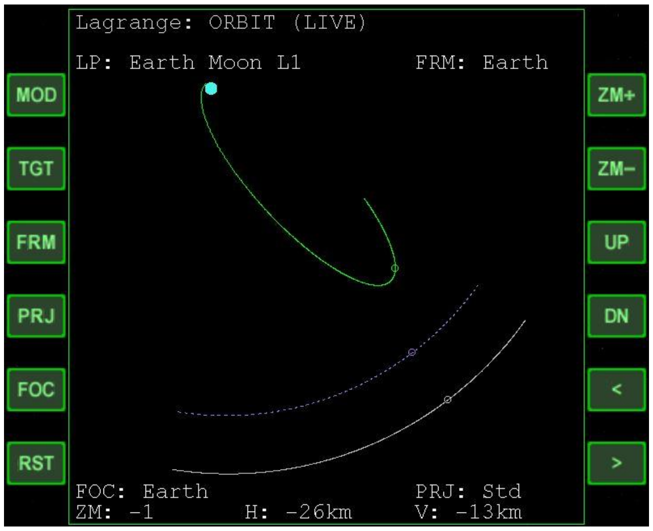
This package is designed to run on Orbiter 2016, on Windows 7, Windows 8 or Windows 10. Prior to installing this package, you *must* install ModuleMessagingExt for Orbiter 2016 from here: http://www.orbithangar.com/searchid.php?ID=6966. If you ignore this dependency, expect to see an error 126 in your Orbiter.log, and Lagrange MFD will not start up.

Unzip this package over the top of your Orbiter 2016 installation, so that the files go in the right directories and locations (e.g. Lagrange.dll goes in your Orbiter 2016's Modules\Plugin directory). Once installed, ensure that the Lagrange module is enabled in the Modules tab in your Orbiter launchpad (i.e. when you run Orbiter.exe or Orbiter_ng.exe, click Modules, and then enable Lagrange, before clicking the Launch Orbiter button).

Discussion of the Mode Screens

1. Orbit Mode

Lagrange MFD Orbit mode shows the live or planned trajectories of the vessel and the celestial bodies in the selected Lagrangian system.



- **Orbit (LIVE) or (PLAN)**

The title to this mode indicates whether the planning mode is active or not.

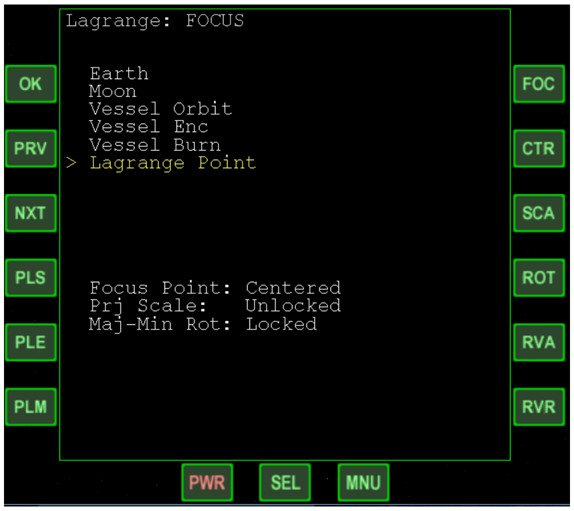
- **Display elements**

By default (user-overrideable in the Colors.cfg file), the MFD displays the Sun in yellow, Earth in pale blue, Moon in grey, the Lagrange point track in dashed purple, the live vessel orbit in green, and the planned vessel orbit in red. Small circles on the trajectory lines indicate the point of closest encounter between the vessel and the LP. If the celestial body is larger than the circle, then the circle will fill in and will expand to indicate the actual size of the body.

- **MOD = Mode Select**
  - Cycles through five mode screens: Orbit, Encounter, Plan, Autopilot, and S4I.
- **TGT = Target Select**
  - This is where you select the LP of interest, from 10 pre-defined choices (EML1-5, SEL1-5).
  - There is a cheat mode called “PMT” (Put Me There) on the Target Select entry screen. Use this to be warped directly to the Lagrange point, if you want to quickly see what it looks like.
- **FRM = Frame of Reference Select**
  - Defines the frame of reference for calculation of orientation (i.e. prograde, outward, plane change). Note that in close proximity to the Earth-Moon LPs, the default Orbit MFD reference defaults to Sun, where you may prefer it to be Earth or Moon. This selection affects the burn orientation and planning, so should not be changed during set-up of a burn and especially not mid-burn.
- **PRJ = Projection Select**
  - Selects the axis projection for the 3D plot. The default view is the usual top-down view of the ecliptic plane, e.g. showing a typical near-circular orbit for the Earth around the Sun. Click PRJ to cycle through top-down, and two sideways-on views of the orbital environment. Notice, for example, the inclination of the lunar orbit with respect to the Earth (or the Earth orbit with respect to the Moon if you prefer).
- **FOC = Focus Select**
  - See the next section below.
- **RST = Reset Pan/Zoom**
  - Resets the view of the orbit, to fit all items into the view screen again.
- **ZM+ / ZM- = Zoom Display**
  - Increases or decreases the zoom factor for the orbit displays
- **UP/DN/</> = Pan Display**
  - Pans the orbit display up/down/left/right similarly to Map MFD
  - Note: these buttons have continuous action (i.e. you can press and hold to scroll the screen)

Note: if the vessel’s predicted orbit results in a close-proximity approach to one of the celestial bodies (i.e. Sun, Earth, Moon), or is predicted to enter Earth’s atmosphere, or is predicted to impact Sun or Moon, then there will be a warning on the bottom of this screen (as well as other modes).

1.1 FOC = Focus Select



```
Lagrange: FOCUS

   Earth
   Moon
   Vessel Orbit
   Vessel Enc
   Vessel Burn
 > Lagrange Point



   Focus Point: Centered
   Prj Scale:   Unlocked
   Maj-Min Rot: Locked
```

This option controls the focus of the Orbit Mode graphics. Select using PRV/NXT, and press OK when complete. Selecting one of these freezes this point on the plot, and everything else rotates around it. There are 4 "lock modes":

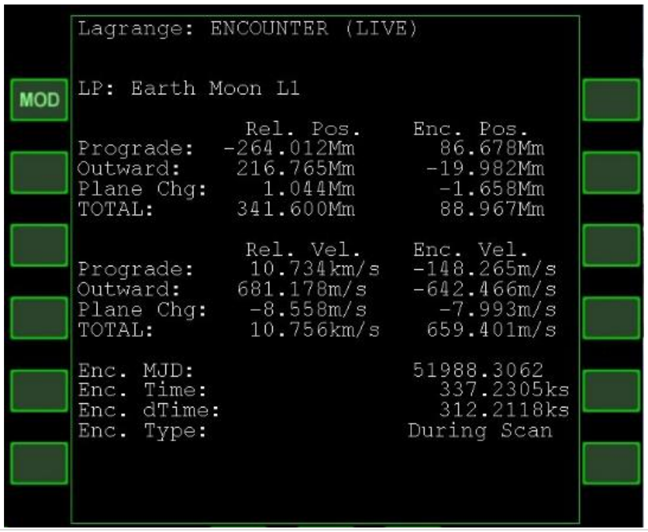
- **FOC:** locks the focus point position on the screen
- **CTR:** locks and centers the focus point position on the screen
- **SCA:** locks the zoom-scale to be the same across each projection view
- **ROT:** locks the rotation of the major and minor entities, to give you a static view of the Lagrange system in a rotating reference plane. This view is really useful to see your vessel's orbit around a Lagrange Point.

There are three Plot Controls: **PLS**, **PLE**, and **PLM**. These switch on and off the display of the Sub, Earth, or Moon plots, respectively. This is useful if you prefer to focus on just the vessel orbit, or the LP orbit, without being distracted by the perturbations of the Earth-Moon orbits.

If you are in a rotating reference plane (i.e. ROT locked mode), the display will show the major body at the origin, the minor body on the +Z axis, and the relative motion of the minor body on the -X axis. If you select the **RVA** button, you can enter three angles to further visualize the motion about the rotating reference plane. The three numbers represent the degrees/second rotation about the X axis, Y axis and Z-axis, in any combination you prefer. The **RVR** button resets the visualization back to the default view for your projection axis.

2. Encounter Mode

Lagrange Encounter Mode shows a detailed readout of your current relative position and velocity to the Lagrange point, and the predicted closest encounter. The three axes (Prograde, Outward, and Plane Change) are the same as used by TransX, and are determined by the FRM setting on the Orbit mode (i.e. axes with respect to the Sun, the Moon, or the Earth).



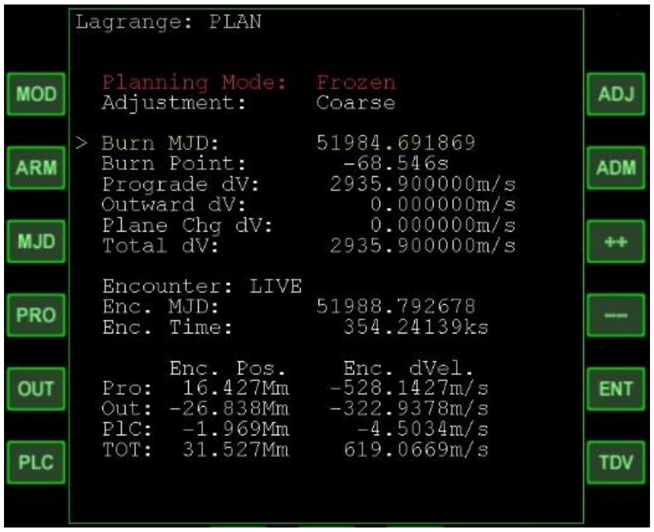
```
Lagrange: ENCOUNTER (LIVE)

LP: Earth Moon L1

                Rel. Pos.       Enc. Pos.
Prograde:   -264.012Mm          86.678Mm
Outward:     216.765Mm         -19.982Mm
Plane Chg:     1.044Mm          -1.658Mm
TOTAL:       341.600Mm          88.967Mm

                Rel. Vel.       Enc. Vel.
Prograde:     10.734km/s     -148.265m/s
Outward:     681.178m/s      -642.466m/s
Plane Chg:    -8.558m/s        -7.993m/s
TOTAL:        10.756km/s      659.401m/s

Enc. MJD:                      51988.3062
Enc. Time:                      337.2305ks
Enc. dTime:                     312.2118ks
Enc. Type:                     During Scan
```

No further interaction is required with this display, as it's purely informational.

In the lower portion of this screen, you can see the encounter time. The time is given in three formats: the Modified Julian Date (MJD in the top right of the Orbiter main window), the simulation time (Sim in the top right of the Orbiter main window), and delta-Time (i.e. countdown timer). You can also see the encounter type, which will say “Before Scan”, “During Scan”, or “After Scan”. The S4I scanner is looking for a minimum on the encounter distance curve. If it does not find one, it simply looks at the distance at the start and the end of the scan to make a presumption whether the encounter was before (i.e. first scan distance was less than the last scan distance), or after (i.e. vice versa).

3. Plan Mode

Lagrange Plan Mode represents the core maneuver planning system in this MFD. Use this mode to create hypothetical burns, and look at Orbit MFD to see the effect of the predicted burn. The display shows the current mode and parameters of the burn, and the effect on the hypothetical encounter point. The yellow highlighted line indicates the currently active parameter for adjustment,



```
Lagrange: PLAN

  Planning Mode:   Frozen
  Adjustment:      Coarse

> Burn MJD:        51984.691869
  Burn Point:        -68.546s
  Prograde dV:      2935.900000m/s
  Outward dV:          0.000000m/s
  Plane Chg dV:        0.000000m/s
  Total dV:         2935.900000m/s

  Encounter: LIVE
  Enc. MJD:        51988.792678
  Enc. Time:         354.24139ks

         Enc. Pos.      Enc. dVel.
  Pro:   16.427Mm    -528.1427m/s
  Out:  -26.838Mm    -322.9378m/s
  PlC:   -1.969Mm      -4.5034m/s
  TOT:   31.527Mm     619.0669m/s
```

- **ARM = Arm/ Disarm the Burn planning mode**
  - Enables or disables the planning mode. When plan is disarmed, the Orbit and Encounter displays show live predictions. When the burn is armed, the Orbit and Encounter displays show planned predictions. From T-2.5s to a burn until the burn is complete, the planning mode is frozen (as shown here), in order to not change the parameters during a burn. When the mode is frozen, the displays revert to the live view, so you can watch the progression of the live encounter point during the burn.

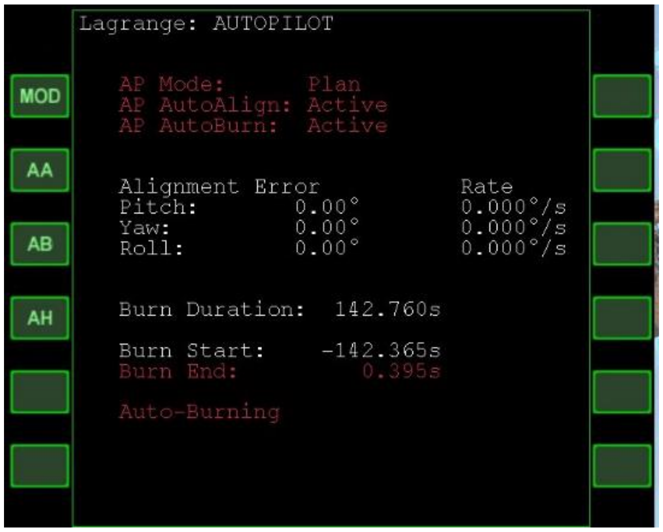
- **MJD = Select Burn MJD Parameter**
  - This selects the Burn MJD parameter for adjustment. (The Burn Point automatically adjusts according to the Burn MJD point).
- **PRO = Select Prograde Parameter**
  - This selects the Prograde parameter for adjustment
- **OUT = Select Outward Parameter**
  - This selects the Outward parameter for adjustment
- **PLC = Select Plane Change Parameter**
  - This selects the Plane Change parameter for adjustment
- **TDV = Select Total DV Parameter**
  - This selects the Total DV parameter for adjustment. Click once to select Total DV. If you set a specific Total DV, then the current burn vector is scaled to achieve the desired total DV. If the burn vector is currently zero, then the total DV is all applied to the prograde axis.
  - If you click TDV again, whilst Total DV is highlighted, it toggles a lock on and off for the Total DV. Having it unlocked is the usual way that the input works (i.e. each axis can be set independently). If the total DV is locked, then for each adjustment of another axis, the other two axes are adjusted to make the total DV stay the same. This is useful where you are happy with the magnitude of the burn, and you are just moving the angle of the burn around. Note that there are times when the lock on total DV cannot be honored: for example, you lock the total DV at 5 m/s, with 3 m/s of outward, and 4 m/s of plane change (remember your Pythagorean Theorem?), and then start incrementing prograde from 0 m/s upwards;

when the Prograde gets to 5 m/s, the other two axes are now at 0 m/s. Continuing to increase the prograde will then increase the total DV as well.

- **ADJ/ADM = Select Adjustment Sensitivity**
  - These two settings work like TransX, setting the increments for change from Rough to Ultra. Note that the settings are remembered per-parameter (e.g. if you are adjusting MJD at Ultra and prograde at Coarse levels, then these are remembered as you select MJD and PRO and back again).
- **++/-- = Increment/Decrement Parameter**
  - These two settings increment or decrement the parameter. They are continuous buttons, so you can press and hold to achieve a smooth change in the parameters. Use ADJ/ADM to adjust the speed of the change to the parameter.
- **ENT = Enter Plane Change Parameter**
  - This allows you to enter a specific value for a parameter.
  - Hint: press TDV, then ENT, then enter 0, then RETURN to reset all the burn dV parameters on one entry.
  -

4. Autopilot Mode

The Lagrange Autopilot mode engages three different types of autopilot: the "AA" auto-alignment autopilot (to point to the desired burn vector), the "AB" auto-burn autopilot (to execute the timed burn), and the "AH" auto-hold autopilot (to hold station when within 1000km of the LP.



```
Lagrange: AUTOPILOT

AP Mode:        Plan
AP AutoAlign:   Active
AP AutoBurn:    Active

Alignment Error            Rate
Pitch:          0.00°      0.000°/s
Yaw:            0.00°      0.000°/s
Roll:           0.00°      0.000°/s

Burn Duration:   142.760s

Burn Start:     -142.365s
Burn End:          0.395s

Auto-Burning
```

- **AA = Arm/Disarm Alignment AP**
  - Can only be armed if Plan mode is armed or frozen

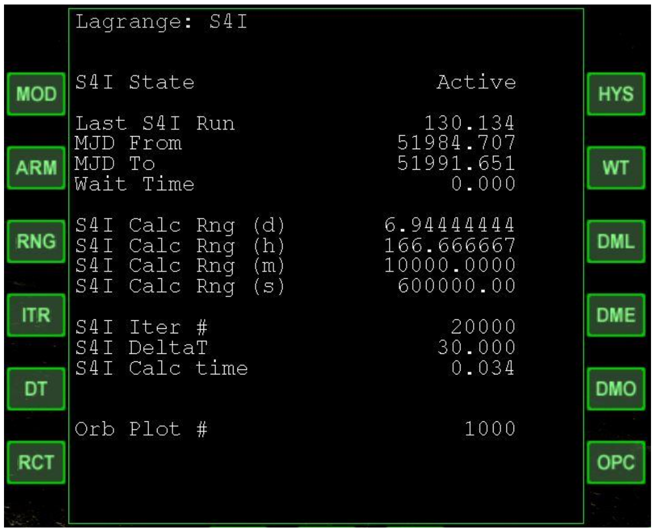
- **AB = Arm/Disarm Burn AP**
  - Can only be armed if Plan mode is armed or frozen
  - Automatically enables AA if AB is enabled

- **AH = Arm/Disarm LP Hold AP**
  - Can only be armed is Plan mode is disarmed and we are within 1000km of the target Lagrange point.

The colors of the display highlight impending actions (e.g. start and end of the burn).

5. S4I Mode

The Lagrange S4I mode controls the internal prediction engine for Lagrange MFD. Its name comes from the underlying physics: it implements a "4th order Symplectic Integrator". See later for a discussion of this integrator. For day to day usage, you do not need to know any of the internal details.



```
Lagrange: S4I

S4I State                  Active

Last S4I Run              130.134
MJD From                51984.707
MJD To                  51991.651
Wait Time                   0.000

S4I Calc Rng (d)       6.94444444
S4I Calc Rng (h)       166.666667
S4I Calc Rng (m)       10000.0000
S4I Calc Rng (s)        600000.00

S4I Iter #                  20000
S4I DeltaT                 30.000
S4I Calc time               0.034

Orb Plot #                   1000
```

- **ARM = Arm/Disarm S4I Integrator**
  - This will freeze or unfreeze the prediction engine. If you freeze it, then the encounter points and the plots will remain as of the last completed calculation. This may be occasionally useful if you do not want the encounter to be updated.

- **RNG = Set Calculation Time Duration Range**
  - This sets the duration of the calculation. You can enter it in days, hours, minutes or seconds (e.g. 4.5d is 4½ days, 90.3m is 90.3 minutes). The calculation range is the product of ITR and DT (i.e. RNG = ITR x DT); adjusting RNG inversely adjusts DT.

- **ITR = Set Iteration Count**
  - This sets the number of iterations for the prediction engine, which also determines how long each prediction calculation will take to run. The default (20,000) is a good balance of responsive updates and general accuracy. You can set this number up to 100,000, depending on the memory configuration and CPU performance of your system. Beyond 100,000 is sometimes possible, but is not recommended, as it runs into memory issues. (See RNG for the relation between ITR and RNG).
- **DT = Set Delta-Time per Iteration Step**
  - This sets the time step for each iteration of the integrator. Typically, this is 10 – 60 seconds. Make this value smaller to achieve a more accurate prediction, or if the Orbit plot is jumping around wildly. (This is a sure indication that you have a very precise orbit with the Lagrange Point, but a large DT, so on successive iteration runs, you are getting wildly different results. With a lower DT, you will reduce the differences between each run.) (See RNG for the relation between DT and RNG).
- **RCT = Set Requested Calculation Time**
  - This sets a requested calculation time for the integrator, based on a linear adjustment to the iterations to achieve the desired multiplier change to the calculation time. (E.g. if you have a calculation time of 1.00 sec, and you desire 0.25 sec, then it will reduce the iteration steps by 4 times.) Note that it also changes the step DT to preserve the calculation range – i.e. for a 4x faster calculation, it does 4x less iterations, but at a 4x longer step delta-time.
  - Note that the actual calculation time is not precisely linear, but if you enter the requested calc time again, you will get successively closer to your desired time (e.g. you will be within 10%-20% on the first entry, and within 1%-2% on the second entry.)
- **HYS = Set Hysteresis Distance**
  - When you have two encounter points (e.g. at +1,000 seconds and +4,500 seconds) which are nearly identical distances, then the integrator can jump between the two points on every calculation run. To minimize this effect, you can either set the calculation range to not scan the second point, or you can use this setting to create a hysteresis effect. “Hysteresis” means that the integrator will only jump to a

new lower distance encounter point once the new point is a lower distance by at least the hysteresis value. This acts to damp down the jumpiness of the engine in these edge-cases.

- **WT = Set Wait Time**
  - This causes the S4I integrator to wait for this wait time setting (in seconds) between each integration calculation. Usually set this at 0.0, unless you want to see an intermittent update e.g. each 10 seconds.
- **OPC = Set Orbit Plot Count**
  - This sets the number of plot lines in the orbit plot, between 10 and 10,000. The default of 1,000 gives a smooth curve except in extreme duration ranges and orbits around Lagrange Points. Set this number higher to make smoother plot lines.

The remaining three buttons are for technical analysis only:

- **DML = Dump the S4I integration log**
  - Creates SNAP.csv, and S4I.csv in the Config\MFD\Lagrange\Diags directrory.
  - SNAP.csv is a snapshot of the state vectors of the entities in the integrator.
  - S4I.csv is a summary dump of a single run of the integrator, showing for each index, the absolute position ("Q") in X,Y,Z dimensions, for Earth, Moon, Sun, Vessel, and the same for the absolute velocity ("P"). The I columns are the instantaneous impulse from a planned burn. The LP columns are the predicted Lagrange point Q/P, and the Vrlp columns are the vessel relative to the LP calculations. Yes … it is doing all these calculations (and many more) on every calculation pass!!

- **DME = Dump the S4I encounter log**
  - Creates ENC.csv in the Config\MFD\Lagrange\Diags directory.
  - ENC.csv shows the local minimum hunting algorithm diagnostic data. The algorithm looks for three successive points that have a higher / lower / higher encounter distance (i.e. local minimum). Once found, it chops the iteration DT successively to do a 2x resolution pass, then a 4x resolution, etc until the determined pass scans over the encounter point at a very small resolution). The columns indicate the vessel index causing the encounter, the three S4I index steps we are scanning, the times and distances for each, and the trend. You will notice the effect of diving into an encounter (with the chop value going 1, 2, 4, 8 … 1024, 2048, etc.) and then resetting back to chop of 1 once the best encounter point is found.

- **DML = Dump the S4I orbit plot log**
  - Creates OrbPlot.csv in the Config\MFD\Lagrange\Diags directory.
  - OrbPlot.csv is a snapshot of the orbit display calculations, showing the lines being plotted, with their color settings, then the details of each plot. Each curve on the Orbit display is by default a 1,000 element poly-line (i.e. it looks smooth, but it's actually a set of straight lines). Each one refers to a specific timestamp from the S4I integrator (e.g. picking 1,000 from the 20,000 S4I plot steps), determining the Focus X and Y point, vessel, LP, Major, Minor, and any Other line. The low and high X and Y limits are determined as this log is generated, and this drives the final calculations of the plot position for each of

the different size MFD's (e.g. you could have an ExtMFD that needs a big plot, versus a panel MFD with a small plot).

Color Configuration Settings

You can adjust the colors for Lagrange.cfg by editing the Colors.cfg file in the Config\MFD\Lagrange directory. For full instructions, look at the comment lines in the top of this file. You can define a custom set of color names, with your preferred Red/Green/Blue selection. For example, use http://rapidtables.com/web/color/RGB_Color.htm to see the effects of different RGB settings. Once your colors are defined, then set the colors and the line types for each plot (Sun, Earth, Moon, etc). Finally, set the default, highlight, and warning colors for the text on each display.

Known Bugs and Items Planned for later versions

1. The fuel usage is a little heavy on the Hold AP. In a future version, I would like to reduce this somewhat.
2. The Hold AP occasionally overshoots the 1,000 km radius around the LP, and then disables itself. Please help it out if it seems to be headed away from the LP. Once it is going in the right direction, it should settle down.
3. Off-plane burns – there is some inaccuracy in the burn engine. For example, if you are doing a 700 m/s outward burn, and a -300 m/s plane change burn, then you may end up 100 m/s off from your expected burn. This is due to the current implementation of the burn engine. In a future version, I will replace the instantaneous calculation with a continuous calculation across the burn duration, which will make this calculation more accurate.
4. Burn targeting – if you change warp whilst the auto-align is mid-alignment, then occasionally it leaves the alignment slightly wrong (e.g. 0.03 degrees misaligned). Give it a manual nudge using ROT thrusters and it will re-hunt for the target.
5. Burn auto-trim – there is no burn auto-trim, meaning that any alignment errors or burn duration errors will nt be automatically nulled out post-burn. You are required to trim it yourself, using small amounts of LIN thrust as needed.
6. Request: collar points for the encounter engine. This is an idea to select an encounter range as a percentage of the calculation range (e.g. 20% - 28%), with tick marks on the orbit plots to indicate the applicable range.
7. Request: a rotating reference frame, for orbital analysis. In this frame, the major-minor axis would be aligned as two fixed points on the Orbit plots, by rotating about the focus point until the major/minor points are horizontally aligned across the middle of the MFD display.
8. Request: ability to plot the current live and the predicted orbit for a vessel at the same time (e.g. show the green and the red lines).

Technical coding discussion

This section is of interest only to developers who are interested in the technical implementation of this MFD. For end-users, most of this will not make any sense at all, and that's ok!

Please see the source at https://github.com/ADSWNJ/LagrangeMFD/ You will always find the latest mainline and future feature branches up here.

The architecture of LagrangeMFD features an ephemeral drawing and display layer as usual for MFD's – i.e. Orbiter.exe destroys your MFD on screen view change or resizing of the ExtMFD. This MFD implements a three-layer persistence core to get around this: a Local Core (per-vessel, per-MFD position), a Vessel Core (per-vessel calculations), and a Global Core (containing the whole Lagrange Universe simulation). You will see everwhere regerences to LC->, VC->, and GC-> to refer to these cores.

The Button Handling functions (MFDButtonPage) are from Enjo's awesome library. You set up each page in Lagrange_Buttons.cpp, then handle the callbacks in Lagrange_ButtonHandling.cpp and Lagrange_DialogFunc.cpp, and all the underlying mastery is left to Enjo!

Pretty early on in the development of this app, I realized that the calc engine to implement the really intricate S4I integration would take a decent amount of time to run, enough that it would cripple the responsiveness of the UI. To get around this, I used the new standard threading library in C++ 11 to implement a background worker thread. The general concept was to have a pair of buffers – active and a working – where the display would work from the active buffer, whilst the worker thread updated the working buffer. On each synchronization point, the background thread freezes on a mutex, and the main thread swaps over the buffers and pushes across any key updates (e.g. new vessels, changed parameters). For items accessed either from the active side or the working thread side, I use the new std::atomic variables for these. See the implementation in Lagrange_Universe.cpp threadCtrlMain(), and threadCtrlWorker() functions. If I were rewriting it, I would have made these into pointers rather than array references everywhere, just to make it look a bit prettier.

I built the autopilot to be portable. I've built autopilots in the past that required complex pre-calibration passes, and were sensitive to vessel weight (e.g. burning off fuel, or full vs empty cargo). For this one, I wrote a dynamically self-calibrating "One Axis" autopilot, with the intention that it can be pulled out and reused in other applications. The Lagrange_AP.cpp manages the six degrees of AP control (X, Y, Z, P, Y, R) via six independent OneAxis_AP objects. The result was very nice. It almost feels unnatural sometimes how aggressively this AP can latch onto a target: it is constantly trying to solve for zero displacement at zero velocity, to end up with a sharp latch onto target.

The S4I Engine

This is a series of essays from Keith "Keithth G" Gelling posted to the Orbiter-Forum in July and August 2016, that served as the entire foundation for this MFD. Keith and I worked together for over 6 months on the technical implementation, and his encouragement and patience with explaining these more complex physics ideas was amazing. I reproduce them here so that they are preserved for future development needs.

Building a simple ephemeris generating tool for short-run mission planning - Part 1

(c) Keith Gelling July 21, 2016

In Orbiter, to do anything useful, you need to know the future trajectory of your spacecraft so that you can ensure that you are 'on target'. Usually, the standard Orbiter MFDs of TransX and IMFD's Map program will give you the information that you want to achieve whatever you mission goal you have set yourself. But sometimes - either because these MFDs don't quite give you the information that you want, or because they don't quite do it with the accuracy that you would like - you may find you need to build your own trajectory planning tool.

An example

A case in point: recently, I was considering the technique for getting out to the Earth-Moon L1 Lagrange point (see Lagrange points/halo orbits? Are there any methods?). This Lagrange point is a point in space at which the gravitational pull of the Earth and Moon more or less balance each other and so, if you park a spacecraft there, then with minimal effort, it will tend to stay at the L1 Lagrange point. However, neither TransX nor IMFD recognise the Earth-Moon Lagrange points as valid 'targets' and so don't provide any information about how close your trajectory will take you to these points, nor do they provide any information about your speed when you get there. Without this information, it is difficult to plan a manoeuvre that 'targets' the Lagrange points.

However, if you know the position of the Earth and Moon at any time in the future you can calculate where the Lagrange points are. And, by using an n-body integrator that takes into account the motion of the spacecraft, the Earth, the Moon and the Sun, you can work out (with considerable accuracy) where your planned trajectory will take you, and how far you will be from a Lagrange point at any time along that trajectory.

But how does one calculate the position of the Earth and Moon so as to be able to calculate the position of the Lagrange point?

Orbiter's internal ephemeris engine

Well, to calculate the position of the Earth and the Moon, you can rely upon the ephemeris system built into Orbiter. This is probably the most accurate way to calculate the position of the Earth and the Moon since it's the same calculation that Orbiter uses, but accessing that information can be tricky - and slow. The calculation of the position of these bodies using Orbiter's ephemeris system requires the summation of a few thousand sine and cosine terms. For a computer, calculating sines and cosines is a slow mathematical operation - and to calculate thousands of sine and cosine terms is very slow. Every time you want to know the position of these bodies, you have to repeat this long-winded calculation.

2-body physics engines

You could also rely on more straightforward Keplerian, 2-body physics. This is essentially what TransX and IMFD's Course program do in making their trajectory projections. But if one is trying to get to a Lagrange point, this is precisely where the gravitational forces of two bodies (the Earth and Moon) more or less cancel out, so you would expect that standard 2-body physics may not work well if you are close to that point. Close to the Lagrange points, and when moving slowly relative to those Lagrange points, a full n-body integration is more or less essential.

So what to do?

The cheap and cheerful solution

One way out of this dilemma is to build your own 'cheap and cheerful' (but still very accurate) short-run ephemeris engine which accurately calculates the position of the Earth and the Moon (and even your spacecraft) for around a week into the future - based solely on a snapshot of the current positions and velocities of the Earth, Moon and Sun.

(What kind of accuracies are we talking about here for an ephemeris engine of this kind? For the Earth and Moon, one can expect accuracies of a few centimetres for 1 hour into the future; a few metres one day into the future; and a few hundred metres a week into the future. For most Earth-Moon missions this provides ample accuracy.)

So, what does it take to build this 'cheap and cheerful' (but accurate) short-run ephemeris engine? Well, you need:

an ability to take a 'snapshot' of the position and velocity of the Earth, Moon and Sun;

an n-body integrator that 'co-integrates' the motion of the spacecraft, the Earth, the Moon and the Sun from their starting positions to, say, a week into the future; and

a tool that analyses these trajectories to allow you to calculate, say, your periapsis altitude at the Earth or the Moon; or your closest approach to L1.

In this context co-integration means to use an n-body integration that calculates the positions of all the bodies at the same time. An example of n-body integrator is a 4th-order symplectic integrator described here Building a 4th order symplectic integrator. So, by co-integrating the motion of all the bodies, taking into account all of the (Newtonian) gravitational forces acting on all of the bodies, a co-integrator gets around the need for an external ephemeris generator by building its own ephemeris 'on the fly'.

Of course, you could write an MFD to do all of this, but assuming that you don't want to write an MFD, then the simple, but slightly more manual way to do this is to:

write a short Lua script that takes the snapshots of position and velocities and dumps the information to a text file;

write an excel spreadsheet which performs the co-integration of the spacecraft Earth, Moon and Sun using the 4th order symplectic integrator;

on the same spreadsheet, assume that the spacecraft executes a manoeuvre at some point in time in its trajectory and the recalculate what its new trajectory will be; and

using all this information, and while still in the spreadsheet - calculate the information that you want (e.g., closest approach to L1, time of arrival at L1, periapsis altitude at the Moon, and so on).

Coming soon

Now, I don't have time right now to go through how to do all of this in detail - but each of the steps is quite straightforward. Progressively, I shall add to this thread detailed notes on how to implement each of these components. By the time I'm done, I should have:

a template Lua script for extracting snapshot information from Orbiter;

constructed a symplectic integrator in a spreadsheet (which will be available for download) that calculates the future trajectories in the Earth-Moon system (although the idea can easily be extended to other plant/moon systems); and

a short video to describe how to use the Lua script and spreadsheet to do some 'real' mission planning.

Building a simple ephemeris generating tool for short-run mission planning - Part 2

(c) Keith Gelling July 24, 2016

Well, I've found a little time this weekend to work on this little project, so time for an addition the preceding post. First a quick recap:

The goal here is to develop a spreadsheet-based, but highly accurate, co-integration scheme to help getting to difficult to reach places - such as the L1 and L2 Lagrange points of the Earth-Moon system. In many respects, this spreadsheet tool has the same type of functionality as IMFD's Map program but loses fidelity over time-scales of much more than a week or so. But over that one week, it can be very accurate.

A couple of things are required to implement this:

* the ability to take a snapshot of current state vectors of the certain bodies (in this case the Earth, Moon and Sun) as well as that of the focus vessel;

* feeding this snapshot information into a fourth-order symplectic integrator and using that symplectic integrator to help with planning a manoeuvre to take the focus vessel to the Lagrange points; and

* translating the outputs of the symplectic integrator into something that tools such as TransX and IMFD can understand so that the manoeuvre can be executed in Orbiter.

This post concentrates on taking the data snapshot. I have also finished building the spreadsheet-based integrator so I should be in a position to post that in the near future.

Why use Microsoft Excel to build the integrator?

Readers may be wondering why I want to build a high-fidelity numerical integrator in Microsoft Excel. Microsoft Excel is good for many things - budget planning, financial analysis and so on - but it isn't renown for being a cutting-edge astrodynamics planning program. And that's because it isn't. It is, however, ubiquitous. Almost everyone will have some passing familiarity with using Excel and most people have access to it - whereas C/C++ (and Python) programming skills are less common. In a way, this is an attempt to demonstrate that building these planning tools of this kind is something that most people have both the requisite skills and tools to implement.

Interfacing with Excel

Anyone that has worked with Orbiter for any length of time will appreciate that Orbiter does not natively talk to Excel. But in order to use Excel, we need a 'data bridge' that allows us to transfer relevant information from Obiter to Excel. The easiest way of building this bridge is by way of an intermediate text file: Orbiter is instructed to produce a text file with the relevant information from its internal tracking of the position and velocity of objects; and Excel is then instructed to pick up this text file and extract the relevant information from it. A simple process. Sometimes tedious to apply, but always robust.

Producing the data file

The easiest way of getting Orbiter to produce the relevant data is by way of a Lua script. This is a short Lua program that is run in Orbiter by way of the Lua Console Window. The Lua script that I am using to produce the bridge text file is as follows:

Code:

```
oapi.set_pause(true)

mjd = oapi.get_simmjd()

-- get the handle for the focus spacecraft
               ves     = vessel.get_focushandle()

-- get the handles for the Earth, the Moon and the Sun
               earth   = oapi.get_objhandle("Earth")
               moon    = oapi.get_objhandle( "Moon")
               sun     = oapi.get_objhandle( "Sun")

-- get the current location of the Earth
               q_ear   = oapi.get_globalpos(earth)
               p_ear   = oapi.get_globalvel(earth)

-- get the current location of the Moon
               q_mon   = oapi.get_globalpos(moon)
               p_mon   = oapi.get_globalvel(moon)
```

```
-- get the current location of the Moon
               q_sun   = oapi.get_globalpos(sun)
               p_sun   = oapi.get_globalvel(sun)

-- get the current location of the Vessel
               q_ves    = oapi.get_globalpos(ves)
               p_ves   = oapi.get_globalvel(ves)

io.write("-- MJD \n")
io.write(mjd, "\n")
io.write("\n")

io.write("-- State vectors of the Earth \n")
io.write(q_ear.x,"  ", q_ear.z, "  ", q_ear.y,  "\n")
io.write(p_ear.x,"  ", p_ear.z, "  ", p_ear.y,  "\n")
io.write("\n")

io.write("-- State vectors of the Moon \n")
io.write(q_mon.x,"  ", q_mon.z, "  ", q_mon.y,  "\n")
io.write(p_mon.x,"  ", p_mon.z, "  ", p_mon.y,  "\n")
io.write("\n")

io.write("-- State vectors of the Sun \n")
io.write(q_sun.x,"  ", q_sun.z, "  ", q_sun.y,  "\n")
io.write(p_sun.x,"  ", p_sun.z, "  ", p_sun.y,  "\n")
io.write("\n")

io.write("-- State vectors of the Vessel \n")
io.write(q_ves.x,"  ", q_ves.z, "  ", q_ves.y,  "\n")
io.write(p_ves.x,"  ", p_ves.z, "  ", p_ves.y,  "\n")
io.write("\n")

oapi.set_pause(false)

io.close(file)
```

This file does the following things:

* it opens a file called 'test.lua' for output

* it temporarily pauses the Orbiter engine. (At the end of the script it starts the Orbiter engine again. This stopping and starting happens so quickly that it isn't discernible.)

* it gets the 'handles' for the focus vessel, the Earth, the Moon and the Sun

* with the integration engine pauses, it then takes a snapshot of the MJD of the snapshot, and the position and velocities of the focus vessel, the Earth, the Moon and the Sun

* it then prints this information out to the bridge file 'test.lua'

* it restarts Orbiter's integration engine

* and, to be tidy, it then closes the bridge file 'test.lua'

Simple, but effective.

Now, you might be wondering: why does the Lua script need the position and velocities of the Sun, Earth and Moon as well as that of the focus vessel? The integration scheme that the spreadsheet will implement is a 'co-integration' scheme. The position of the (dominant) gravitating bodies is needed to know the forces acting on the focus vessel at any point in the future. And rather than carry around the computational overhead of a separate ephemeris generator for these bodies, the integration scheme calculates this ephemeris information 'on the fly' from the their snapshot positions and velocities. This type of integration scheme produces high accuracy in the near-term but does not perform well over long time-scales.

Running the Lua script

The Lua script file needs to be placed in the "Script" directory located under the Orbiter 2010 root directory with a ".lua" suffix. To run the script, load up Orbiter 2010 and open the Lua Console Window. (To do this you will have to enable the module.) Then, in the Console Window type "run 'XXXXX' " where XXXX is the name of the script without the '.lua' suffix. That should be sufficient.

The output of the Lua script

A sample of the snapshot output file is shown below:

Code:

```
-- MJD
51987.250220876

-- State vectors of the Earth
-149508665534.05  2635582735.6114  21269682.320874
-1162.176500679  -29901.58660315  0.78700526732017

-- State vectors of the Moon
-149339150764.12  2268244892.3456  13600127.396682
-281.13300397464  -29510.988872506  -82.803656389856

-- State vectors of the Sun
-598586284.83708  -775318010.03728  21094585.025077
14.440446183745  -5.1243342754206  -0.32898982217866

-- State vectors of the Vessel
-149506285915.75  2633449799.5878  27193356.262695
5897.4800025109  -30415.309360392  -3025.5718902154
```

This output file contains 25 useful numbers:

* the MJD of the snapshot

* four times six numbers that capture the X-Y-Z components of the position and velocities of the Sun, Earth, Moon and the focus vessel.

This information is given in Orbiter's global coordinate system and, more or less, the origin of that coordinate system is located at the Solar System Barycentre (i.e., the Centre of Mass of the Solar System). Positions are give in metres, and velocities are given in m/s.

Nest step

The next step is to feed this data into the Excel spreadsheet integrator - but this will be the subject of the next post.

Building a simple ephemeris generating tool for short-run mission planning - Part 3

(c) Keith Gelling July 26, 2016

This is the third post in a short series setting out how to use Microsoft Excel to build a high-fidelity trajectory integration engine with a view to using that engine to target, say, the Lagrange points of the Earth-Moon system. Microsoft Excel has been chosen in this instance because most people have some familiarity with, and access to, Excel and so should be able to build and implement these tools themselves - should they so wish to do so.

In the previous post (above), a data bridge was constructed that extracted the relevant information from Orbiter and posted that information to a text file. The next step ini the process is to load the data into the integration engine spreadsheet so as to project spacecraft trajectories into the future. The heart of that spreadsheet, though, is the Visual Basic User-Defined-Function (aka 'a macro') that performs a single time-step of the symplectic integrator. This not focuses on that User-Defined-Function which is presented in the code block below for the four-bodyEarth/Moon/Sun/Vessel system:

Code:

```
Public Function S4INTEGRATOR(QP As Object) As Variant

    Dim QX_E, QY_E, QZ_E, PX_E, PY_E, PZ_E As Double
    Dim QX_M, QY_M, QZ_M, PX_M, PY_M, PZ_M As Double
    Dim QX_S, QY_S, QZ_S, PX_S, PY_S, PZ_S As Double
    Dim QX_V, QY_V, QZ_V, PX_V, PY_V, PZ_V As Double

    Dim PX_I, PY_I, PZ_I As Double

    Dim dt, w0, w1 As Double

    dt = 30
    w0 = 1.35120719195966
    w1 = -1.70241438391932


    QX_E = QP(1): QY_E = QP(2): QZ_E = QP(3)
    QX_M = QP(4): QY_M = QP(5): QZ_M = QP(6)
    QX_S = QP(7): QY_S = QP(8): QZ_S = QP(9)
    QX_V = QP(10): QY_V = QP(11): QZ_V = QP(12)

    PX_E = QP(13): PY_E = QP(14): PZ_E = QP(15)
    PX_M = QP(16): PY_M = QP(17): PZ_M = QP(18)
    PX_S = QP(19): PY_S = QP(20): PZ_S = QP(21)
    PX_V = QP(22): PY_V = QP(23): PZ_V = QP(24)
    PX_I = QP(25): PY_I = QP(26): PZ_I = QP(27)
    ' Add in the impulse
    PX_V = PX_V + PX_I
    PY_V = PY_V + PY_I
    PZ_V = PZ_V + PZ_I
    ' Step 1 - A
    QX_E = QX_E + 0.5 * w0 * dt * PX_E
    QY_E = QY_E + 0.5 * w0 * dt * PY_E
    QZ_E = QZ_E + 0.5 * w0 * dt * PZ_E

    QX_M = QX_M + 0.5 * w0 * dt * PX_M
    QY_M = QY_M + 0.5 * w0 * dt * PY_M
    QZ_M = QZ_M + 0.5 * w0 * dt * PZ_M

    QX_S = QX_S + 0.5 * w0 * dt * PX_S
    QY_S = QY_S + 0.5 * w0 * dt * PY_S
    QZ_S = QZ_S + 0.5 * w0 * dt * PZ_S
```

```
QX_V = QX_V + 0.5 * w0 * dt * PX_V
QY_V = QY_V + 0.5 * w0 * dt * PY_V
QZ_V = QZ_V + 0.5 * w0 * dt * PZ_V
' Step 1 - B
PX_E = PX_E + w0 * dt * FX_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_E = PY_E + w0 * dt * FY_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_E = PZ_E + w0 * dt * FZ_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_M = PX_M + w0 * dt * FX_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_M = PY_M + w0 * dt * FY_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_M = PZ_M + w0 * dt * FZ_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_S = PX_S + w0 * dt * FX_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_S = PY_S + w0 * dt * FY_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_S = PZ_S + w0 * dt * FZ_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_V = PX_V + w0 * dt * FX_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
PY_V = PY_V + w0 * dt * FY_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
PZ_V = PZ_V + w0 * dt * FZ_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)

' Step 1 - C
QX_E = QX_E + 0.5 * w0 * dt * PX_E
QY_E = QY_E + 0.5 * w0 * dt * PY_E
QZ_E = QZ_E + 0.5 * w0 * dt * PZ_E

QX_M = QX_M + 0.5 * w0 * dt * PX_M
QY_M = QY_M + 0.5 * w0 * dt * PY_M
QZ_M = QZ_M + 0.5 * w0 * dt * PZ_M

QX_S = QX_S + 0.5 * w0 * dt * PX_S
QY_S = QY_S + 0.5 * w0 * dt * PY_S
QZ_S = QZ_S + 0.5 * w0 * dt * PZ_S

QX_V = QX_V + 0.5 * w0 * dt * PX_V
QY_V = QY_V + 0.5 * w0 * dt * PY_V
QZ_V = QZ_V + 0.5 * w0 * dt * PZ_V
' Step 2 - A
QX_E = QX_E + 0.5 * w1 * dt * PX_E
QY_E = QY_E + 0.5 * w1 * dt * PY_E
QZ_E = QZ_E + 0.5 * w1 * dt * PZ_E

QX_M = QX_M + 0.5 * w1 * dt * PX_M
QY_M = QY_M + 0.5 * w1 * dt * PY_M
QZ_M = QZ_M + 0.5 * w1 * dt * PZ_M

QX_S = QX_S + 0.5 * w1 * dt * PX_S
QY_S = QY_S + 0.5 * w1 * dt * PY_S
QZ_S = QZ_S + 0.5 * w1 * dt * PZ_S

QX_V = QX_V + 0.5 * w1 * dt * PX_V
QY_V = QY_V + 0.5 * w1 * dt * PY_V
QZ_V = QZ_V + 0.5 * w1 * dt * PZ_V

' Step 2 - B
PX_E = PX_E + w1 * dt * FX_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_E = PY_E + w1 * dt * FY_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_E = PZ_E + w1 * dt * FZ_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_M = PX_M + w1 * dt * FX_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_M = PY_M + w1 * dt * FY_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_M = PZ_M + w1 * dt * FZ_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_S = PX_S + w1 * dt * FX_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
```

```
PY_S = PY_S + w1 * dt * FY_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_S = PZ_S + w1 * dt * FZ_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_V = PX_V + w1 * dt * FX_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
PY_V = PY_V + w1 * dt * FY_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
PZ_V = PZ_V + w1 * dt * FZ_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
' Step 2 - C
QX_E = QX_E + 0.5 * w1 * dt * PX_E
QY_E = QY_E + 0.5 * w1 * dt * PY_E
QZ_E = QZ_E + 0.5 * w1 * dt * PZ_E

QX_M = QX_M + 0.5 * w1 * dt * PX_M
QY_M = QY_M + 0.5 * w1 * dt * PY_M
QZ_M = QZ_M + 0.5 * w1 * dt * PZ_M

QX_S = QX_S + 0.5 * w1 * dt * PX_S
QY_S = QY_S + 0.5 * w1 * dt * PY_S
QZ_S = QZ_S + 0.5 * w1 * dt * PZ_S

QX_V = QX_V + 0.5 * w1 * dt * PX_V
QY_V = QY_V + 0.5 * w1 * dt * PY_V
QZ_V = QZ_V + 0.5 * w1 * dt * PZ_V
' Step 3 - A
QX_E = QX_E + 0.5 * w0 * dt * PX_E
QY_E = QY_E + 0.5 * w0 * dt * PY_E
QZ_E = QZ_E + 0.5 * w0 * dt * PZ_E

QX_M = QX_M + 0.5 * w0 * dt * PX_M
QY_M = QY_M + 0.5 * w0 * dt * PY_M
QZ_M = QZ_M + 0.5 * w0 * dt * PZ_M

QX_S = QX_S + 0.5 * w0 * dt * PX_S
QY_S = QY_S + 0.5 * w0 * dt * PY_S
QZ_S = QZ_S + 0.5 * w0 * dt * PZ_S

QX_V = QX_V + 0.5 * w0 * dt * PX_V
QY_V = QY_V + 0.5 * w0 * dt * PY_V
QZ_V = QZ_V + 0.5 * w0 * dt * PZ_V
' Step 3 - B
PX_E = PX_E + w0 * dt * FX_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_E = PY_E + w0 * dt * FY_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_E = PZ_E + w0 * dt * FZ_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_M = PX_M + w0 * dt * FX_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_M = PY_M + w0 * dt * FY_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_M = PZ_M + w0 * dt * FZ_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_S = PX_S + w0 * dt * FX_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PY_S = PY_S + w0 * dt * FY_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
PZ_S = PZ_S + w0 * dt * FZ_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

PX_V = PX_V + w0 * dt * FX_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
PY_V = PY_V + w0 * dt * FY_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
PZ_V = PZ_V + w0 * dt * FZ_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
' Step 3 - C
QX_E = QX_E + 0.5 * w0 * dt * PX_E
QY_E = QY_E + 0.5 * w0 * dt * PY_E
QZ_E = QZ_E + 0.5 * w0 * dt * PZ_E

QX_M = QX_M + 0.5 * w0 * dt * PX_M
QY_M = QY_M + 0.5 * w0 * dt * PY_M
QZ_M = QZ_M + 0.5 * w0 * dt * PZ_M

QX_S = QX_S + 0.5 * w0 * dt * PX_S
```

```
        QY_S = QY_S + 0.5 * w0 * dt * PY_S
        QZ_S = QZ_S + 0.5 * w0 * dt * PZ_S

        QX_V = QX_V + 0.5 * w0 * dt * PX_V
        QY_V = QY_V + 0.5 * w0 * dt * PY_V
        QZ_V = QZ_V + 0.5 * w0 * dt * PZ_V

        ' Return the results
        Test = Array(QX_E, QY_E, QZ_E, _
                     QX_M, QY_M, QZ_M, _
                     QX_S, QY_S, QZ_S, _
                     QX_V, QY_V, QZ_V, _
                     PX_E, PY_E, PZ_E, _
                     PX_M, PY_M, PZ_M, _
                     PX_S, PY_S, PZ_S, _
                     PX_V, PY_V, PZ_V)

    End Function
    Private Function FX_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_E - QX_M) * (QX_E - QX_M) + (QY_E - QY_M) * (QY_E - QY_M) + (QZ_E - QZ_M) * (QZ_E -
    QZ_M)) ^ 1.5
        temp2 = ((QX_E - QX_S) * (QX_E - QX_S) + (QY_E - QY_S) * (QY_E - QY_S) + (QZ_E - QZ_S) * (QZ_E -
    QZ_S)) ^ 1.5
        FX_E = -mu_M * (QX_E - QX_M) / temp1 - mu_S * (QX_E - QX_S) / temp2

    End Function
    Private Function FY_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_E - QX_M) * (QX_E - QX_M) + (QY_E - QY_M) * (QY_E - QY_M) + (QZ_E - QZ_M) * (QZ_E -
    QZ_M)) ^ 1.5
        temp2 = ((QX_E - QX_S) * (QX_E - QX_S) + (QY_E - QY_S) * (QY_E - QY_S) + (QZ_E - QZ_S) * (QZ_E -
    QZ_S)) ^ 1.5
        FY_E = -mu_M * (QY_E - QY_M) / temp1 - mu_S * (QY_E - QY_S) / temp2

    End Function
    Private Function FZ_E(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_E - QX_M) * (QX_E - QX_M) + (QY_E - QY_M) * (QY_E - QY_M) + (QZ_E - QZ_M) * (QZ_E -
    QZ_M)) ^ 1.5
        temp2 = ((QX_E - QX_S) * (QX_E - QX_S) + (QY_E - QY_S) * (QY_E - QY_S) + (QZ_E - QZ_S) * (QZ_E -
    QZ_S)) ^ 1.5
        FZ_E = -mu_M * (QZ_E - QZ_M) / temp1 - mu_S * (QZ_E - QZ_S) / temp2

    End Function
    Private Function FX_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)
        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_M - QX_E) * (QX_M - QX_E) + (QY_M - QY_E) * (QY_M - QY_E) + (QZ_M - QZ_E) * (QZ_M -
    QZ_E)) ^ 1.5
        temp2 = ((QX_M - QX_S) * (QX_M - QX_S) + (QY_M - QY_S) * (QY_M - QY_S) + (QZ_M - QZ_S) * (QZ_M -
    QZ_S)) ^ 1.5
        FX_M = -mu_E * (QX_M - QX_E) / temp1 - mu_S * (QX_M - QX_S) / temp2

    End Function
    Private Function FY_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_M - QX_E) * (QX_M - QX_E) + (QY_M - QY_E) * (QY_M - QY_E) + (QZ_M - QZ_E) * (QZ_M -
    QZ_E)) ^ 1.5
        temp2 = ((QX_M - QX_S) * (QX_M - QX_S) + (QY_M - QY_S) * (QY_M - QY_S) + (QZ_M - QZ_S) * (QZ_M -
    QZ_S)) ^ 1.5
        FY_M = -mu_E * (QY_M - QY_E) / temp1 - mu_S * (QY_M - QY_S) / temp2

    End Function
    Private Function FZ_M(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_M - QX_E) * (QX_M - QX_E) + (QY_M - QY_E) * (QY_M - QY_E) + (QZ_M - QZ_E) * (QZ_M -
    QZ_E)) ^ 1.5
        temp2 = ((QX_M - QX_S) * (QX_M - QX_S) + (QY_M - QY_S) * (QY_M - QY_S) + (QZ_M - QZ_S) * (QZ_M -
    QZ_S)) ^ 1.5
        FZ_M = -mu_E * (QZ_M - QZ_E) / temp1 - mu_S * (QZ_M - QZ_S) / temp2

    End Function
    Private Function FX_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_S - QX_E) * (QX_S - QX_E) + (QY_S - QY_E) * (QY_S - QY_E) + (QZ_S - QZ_E) * (QZ_S -
    QZ_E)) ^ 1.5
        temp2 = ((QX_S - QX_M) * (QX_S - QX_M) + (QY_S - QY_M) * (QY_S - QY_M) + (QZ_S - QZ_M) * (QZ_S -
    QZ_M)) ^ 1.5
        FX_S = -mu_E * (QX_S - QX_E) / temp1 - mu_M * (QX_S - QX_M) / temp2

    End Function
    Private Function FY_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_S - QX_E) * (QX_S - QX_E) + (QY_S - QY_E) * (QY_S - QY_E) + (QZ_S - QZ_E) * (QZ_S -
    QZ_E)) ^ 1.5
        temp2 = ((QX_S - QX_M) * (QX_S - QX_M) + (QY_S - QY_M) * (QY_S - QY_M) + (QZ_S - QZ_M) * (QZ_S -
    QZ_M)) ^ 1.5
        FY_S = -mu_E * (QY_S - QY_E) / temp1 - mu_M * (QY_S - QY_M) / temp2

    End Function
    Private Function FZ_S(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S)

        Dim mu_E, mu_M, temp1, temp2 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_S - QX_E) * (QX_S - QX_E) + (QY_S - QY_E) * (QY_S - QY_E) + (QZ_S - QZ_E) * (QZ_S -
    QZ_E)) ^ 1.5
        temp2 = ((QX_S - QX_M) * (QX_S - QX_M) + (QY_S - QY_M) * (QY_S - QY_M) + (QZ_S - QZ_M) * (QZ_S -
    QZ_M)) ^ 1.5
        FZ_S = -mu_E * (QZ_S - QZ_E) / temp1 - mu_M * (QZ_S - QZ_M) / temp2

    End Function
    Private Function FX_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)

        Dim mu_E, mu_M, temp1, temp2, temp3 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_V - QX_E) * (QX_V - QX_E) + (QY_V - QY_E) * (QY_V - QY_E) + (QZ_V - QZ_E) * (QZ_V -
    QZ_E)) ^ 1.5
        temp2 = ((QX_V - QX_M) * (QX_V - QX_M) + (QY_V - QY_M) * (QY_V - QY_M) + (QZ_V - QZ_M) * (QZ_V -
    QZ_M)) ^ 1.5
        temp3 = ((QX_V - QX_S) * (QX_V - QX_S) + (QY_V - QY_S) * (QY_V - QY_S) + (QZ_V - QZ_S) * (QZ_V -
    QZ_S)) ^ 1.5

        FX_V = -mu_E * (QX_V - QX_E) / temp1 - mu_M * (QX_V - QX_M) / temp2 - mu_S * (QX_V - QX_S) / temp3


    End Function
    Private Function FY_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)

        Dim mu_E, mu_M, temp1, temp2, temp3 As Double

        mu_E = 398600440157821#
        mu_M = 4902794935300#
        mu_S = 1.32712440018E+20

        temp1 = ((QX_V - QX_E) * (QX_V - QX_E) + (QY_V - QY_E) * (QY_V - QY_E) + (QZ_V - QZ_E) * (QZ_V -
    QZ_E)) ^ 1.5
        temp2 = ((QX_V - QX_M) * (QX_V - QX_M) + (QY_V - QY_M) * (QY_V - QY_M) + (QZ_V - QZ_M) * (QZ_V -
    QZ_M)) ^ 1.5
        temp3 = ((QX_V - QX_S) * (QX_V - QX_S) + (QY_V - QY_S) * (QY_V - QY_S) + (QZ_V - QZ_S) * (QZ_V -
    QZ_S)) ^ 1.5

        FY_V = -mu_E * (QY_V - QY_E) / temp1 - mu_M * (QY_V - QY_M) / temp2 - mu_S * (QY_V - QY_S) / temp3


    End Function
    Private Function FZ_V(QX_E, QY_E, QZ_E, QX_M, QY_M, QZ_M, QX_S, QY_S, QZ_S, QX_V, QY_V, QZ_V)
```

```
    Dim mu_E, mu_M, temp1, temp2, temp3 As Double

    mu_E = 398600440157821#
    mu_M = 4902794935300#
    mu_S = 1.32712440018E+20

    temp1 = ((QX_V - QX_E) * (QX_V - QX_E) + (QY_V - QY_E) * (QY_V - QY_E) + (QZ_V - QZ_E) * (QZ_V -
QZ_E)) ^ 1.5
    temp2 = ((QX_V - QX_M) * (QX_V - QX_M) + (QY_V - QY_M) * (QY_V - QY_M) + (QZ_V - QZ_M) * (QZ_V -
QZ_M)) ^ 1.5
    temp3 = ((QX_V - QX_S) * (QX_V - QX_S) + (QY_V - QY_S) * (QY_V - QY_S) + (QZ_V - QZ_S) * (QZ_V -
QZ_S)) ^ 1.5

    FZ_V = -mu_E * (QZ_V - QZ_E) / temp1 - mu_M * (QZ_V - QZ_M) / temp2 - mu_S * (QZ_V - QZ_S) / temp3

End Function
```

A brief description of the macro

OK, so what does this macro do? Basically this is a function that takes an Excel data range as an input and returns a vector-valued result. The input are the starting positions and velocities of the Earth, Moon, Sun and Vessel; and the outputs are the updated positions and velocities of the same. Repeated application of this function produces the integrated trajectory.

Inputs

The macro assumes that the input data range is a contiguous set of 27 numbers. That's quite a few - so here is a description of them;

* the first three are the starting X, Y and Z coordinates of the Earth's position. For the first time-step these values come from the Lua script described in the previous post. For subsequent time-steps, the input is (in the main) the outputs from the previous time-step;

* the next three are the starting X Y, Z coordinates of the Moon's position;

* the next three are the starting X,Y, Z coordinates of the Sun's position; and

* the next three are the starting X,Y,Z coordinates of the Vessel's position.

So far, that makes 12 input numbers. Now for the next 12:

* the next three are the starting X, Y, Z coordinates of the Earth's velocity;

* the next three are the starting X, Y, Z coordinates of the Moon's velocity;

* the next three are the starting X, Y, Z coordinates of the Sun's velocity;

* the next three are the starting X, Y, Z coordinates of the Vessel's velocity.

In total the positions and velocity inputs make 24 data inputs. What about the remaining three? Well, these are discretionary user inputs that define the X-Y-Z co-ordinates of a velocity impulse (e.g., an impulsive manoeuvre that takes place at the start of the time-step. It is here that one would enter an optional user-defined impulsive burn. If no burn is planned for that time-step, then the user must set these values to zero.

The integration process

So, having specified a data range, the first thing that the integration macro does is to pars the input data into these 27 separate values. Here a notation we use a shorthand notation to specify the various values. For example, 'QY_S' is shorthand for the y-coordinate of the Sun's position; and 'PX_V' is shorthand for the x-coordinate of the Vessel's velocity.

Having parsed the inputs, the macro then runs the symplectic integration step. In this macro, the time-step is fixed to a 30 second update which means that the output of the macro is the position and velocity of the Earth, Moon, Sun and Vessel 30 seconds after the timestamp of the input data. For most purposes, this afford sufficient precision.

In executing the integration step, the macro needs to know the gravitational forces acting on the four bodies of our system. To calculate these, 12 force functions have been defined. These calculate the gravitational forces assuming spherical symmetry. In this integrator, non-spherical gravity sources have been ignored, although if one wanted to be pedantic, one could include those in the force functions as well.

Outputs

The output of the macro is an array with 24 values. These are the updated positions and velocities of the four bodies. And the order that they appear in the vector is the same as in the input data range.

What about the remaining three impulsive manoeuvre components of the input data range? These are not updated by the macro. Being discretionary, they need to be specified by the user.

Next steps

The heart of the integration engine is precisely the single-step macro set out above. In the next step, I'll present the macro which contains the working integration engine and trajectory planner.

Building a simple ephemeris generating tool for short-run mission planning - Part 4

(c) Keith Gelling July 26, 2016

Well, here it is: the first iteration of the Excel speadsheet-based integration engine (download from https://www.dropbox.com/s/56qfw8zex9...S4INT.xls?dl=0).

Believe it or, this is a usable, precision trajectory planning tool. It is accurate (some might even say very accurate), but as with all things Excel, it doesn't calculate quickly, so some patience is required to use properly. But still, it shows how these sort of tools can be built and if one wishes a more responsive tool, then one would have to transcribe this know-how into a 'proper' computer language - e.g., C/C++.

This is the first iteration of the spreadsheet, largely as a demonstration (and because I have limited time at the moment), and this version's core function is planning Trans Lunar Injection (TLI) burns. Of course, there are many ways of using Orbiter that achieve much the same - although aside from IMFD's Map Program, none I suspect will be as accurate. The second iteration of this spreadsheet, however, O plan to make more useful. It will incorporate the location of the Earth-Moon L1/L2 Lagrange points. This will make it a precision tool for planning Trans Earth-Moon L1/L2 injection burns from Low Earth Orbit - as well as subsequent mid-course corrections so as to achieve near pin-point L1/L2 targeting.

First iteration

The first iteration spreadsheet has two worksheets - a control sheet; and a integration sheet. On the control sheet (screen image below), there is a grey area into which the output of the Lua script that takes position and velocity snapshots is pasted into and then parsed using Excel's text editor. (Yes, a very manual process, I know.)

On the right of the control sheet, is a graph showing the trajectories of the spacecraft and Moon over the next seven days (after the date of the snapshot). Earth is in the centre of the image; and trajectories are projected into the plane of the ecliptic. This isn't useful for precision work, but it does help clarify what is going on.

And on the bottom left of the control sheet, the spreadsheet allows input of one impulse manoeuvre described in the usual TransX components of 'prograde', 'outward' and 'plane'. At the moment, the limitation is that this impulsive manoeuvre must take place on the dates of the integration time-steps - spaced at 30 second intervals. This date limitation reflects more my programming convenience than a limitation of the underlying method. In this area of the worksheet,the closest approach to the Moon (and date of closest approach) is reported. This information is used to refine the impulsive manoeuvre. For anyone familiar with TransX, this should be a familiar procedure.

The second sheet, contains the integrator itself. This sheet picks up the snapshot information from the control sheet and then uses the integration macro to integrate forward for 20,000 time-steps. Since each time-step is 30 seconds, that works out as a total integration period of 6.94 days - i.e., a shade under one week. Since each time-step means updating 24 worksheet cells (12 that define the position of the Earth, Moon, Sun and Vessel; and 12 that define the velocities of the Earth, Moon, Sun and Vessel), this means that the integrator calculates half a million spreadsheet cells. It is this extensive calculation effort that give the integrator its accuracy, but also its poor performance

On this sheet, a few ancillary calculations are performed allow lunar closest approach to be calculated, and to provide the graphical trajectory information for the first sheet.

Second iteration

The next iteration of this spreadsheet will do much the same as this, except that it will contain targeting information for the Lagrange points, L1 and L2.

A couple of caveats

A spreadsheet isn't the natural way to do these trajectory calculations. This is due largely to its sluggishness in performing the calculations. To avoid some frustration in use, you may wish to make sure you have access to a reasonably modern (and powerful - i.e., multicore) laptop or desktop - and, if you can, turn multi-threading 'on' in Excel.

Next steps

The next obvious step is to provide some instruction in use of this targeting tool. This will be by way of a shortish video. But not today, I'm afraid. And after that I'll provide a video demonstration of the use of the tool to get to the Moon. Then after that, I'll repeat the whole exercise for the second iteration spreadsheet - and targeting Lagrange points.

Building a simple ephemeris generating tool for short-run mission planning - Part 5

(c) Keith Gelling Aug 9, 2016

So far, a spreadsheet-based trajectory planning tool has been developed that incorporates an accurate fourth-order symplectic integrator along with some rudimentary visualisation and targeting information for getting to the Moon. However, the original purpose of this too development was to develop a workable (albeit sluggish) tool with which to target the Lagrange points of the Earth-Moon system.

To be able to do that, the spreadsheet tool needs to have a way of working out where the L1 and L2 Lagrange points are at any point in time. Fortunately, the basic theory of this calculation has already been worked out and described in this Forum. See A reference orbit for the Earth-Moon Lagrange points (Elliptic Restricted Three-Body. That thread sets out the calculations needed to work out the location of the Lagrange points if the position of the Earth and the Moon are known. And since the spreadsheet symplectic integrator of this thread calculates the position of the Earth and the Moon as part of its calculations, this information can be fed into the Lagrange point calculation via another Visual Basic user-defined function.

And here is the Visual Basic user-defined function for the L1 Lagrange point:

Code:

```
Public Function L1Test(QE As Object, QM As Object, PE As Object, PM As Object) As Variant

    Dim QX_E, QY_E, QZ_E, PX_E, PY_E, PZ_E As Double
    Dim QX_M, QY_M, QZ_M, PX_M, PY_M, PZ_M As Double

    QX_E = QE(1): QY_E = QE(2): QZ_E = QE(3)
    QX_M = QM(1): QY_M = QM(2): QZ_M = QM(3)

    PX_E = PE(1): PY_E = PE(2): PZ_E = PE(3)
    PX_M = PM(1): PY_M = PM(2): PZ_M = PM(3)


    Dim GM1, GM2, GM, MU1, MU2, alpha As Double

    GM1 = 398600440157821#
    GM2 = 4902794935300#
    GM = GM1 + GM2
    MU1 = GM1 / GM
    MU2 = GM2 / GM
    alpha = 0.83691519487206
    Dim COMX, COMY, COMZ As Double
    Dim COVX, COVY, COVZ As Double

    COMX = MU1 * QX_E + MU2 * QX_M
    COMY = MU1 * QY_E + MU2 * QY_M
    COMZ = MU1 * QZ_E + MU2 * QZ_M

    COVX = MU1 * PX_E + MU2 * PX_M
    COVY = MU1 * PY_E + MU2 * PY_M
    COVZ = MU1 * PZ_E + MU2 * PZ_M

    Dim RX, RY, RZ As Double
    Dim VX, VY, VZ As Double

    RX = QX_M - QX_E
    RY = QY_M - QY_E
    RZ = QZ_M - QZ_E

    VX = PX_M - PX_E
    VY = PY_M - PY_E
    VZ = PZ_M - PZ_E
```

```
Dim vsq, rln, rv, eX, eY, eZ, ecc, a, nu As Double

vsq = VX * VX + VY * VY + VZ * VZ
rln = Sqr(RX * RX + RY * RY + RZ * RZ)
rv = RX * VX + RY * VY + RZ * VZ

eX = RX * vsq / GM - VX * rv / GM - RX / rln
eY = RY * vsq / GM - VY * rv / GM - RY / rln
eZ = RZ * vsq / GM - VZ * rv / GM - RZ / rln
ecc = Sqr(eX * eX + eY * eY + eZ * eZ)
a = GM / (2# * GM / rln - vsq)
nu = Application.Acos((eX * RX + eY * RY + eZ * RZ) / ecc / rln)
If rv < 0 Then nu = 2# * Application.Pi() - nu


Dim xhat_X, xhat_Y, xhat_Z As Double
Dim yhat_X, yhat_Y, yhat_Z As Double
Dim zhat_X, zhat_Y, zhat_Z As Double
Dim temp As Double

xhat_X = eX / ecc
xhat_Y = eY / ecc
xhat_Z = eZ / ecc

zhat_X = RY * VZ - RZ * VY
zhat_Y = RZ * VX - RX * VZ
zhat_Z = RX * VY - RY * VX

temp = Sqr(zhat_X * zhat_X + zhat_Y * zhat_Y + zhat_Z * zhat_Z)

zhat_X = zhat_X / temp
zhat_Y = zhat_Y / temp
zhat_Z = zhat_Z / temp

yhat_X = zhat_Y * xhat_Z - zhat_Z * xhat_Y
yhat_Y = zhat_Z * xhat_X - zhat_X * xhat_Z
yhat_Z = zhat_X * xhat_Y - zhat_Y * xhat_X


Dim k1, k2, cnu, snu, k3 As Double

k1 = a * (1# - ecc * ecc)
k2 = Sqr(GM / k1)
cnu = Cos(nu)
snu = Sin(nu)
k3 = 1# + ecc * cnu


Dim qx, qy, px, py As Double

qx = alpha * k1 * cnu / k3
qy = alpha * k1 * snu / k3
px = -alpha * snu * k2
py = alpha * (ecc + cnu) * k2


Dim L1q_X, L1q_Y, L1q_Z As Double
Dim L1p_X, L1p_Y, L1p_Z As Double

L1q_X = qx * xhat_X + qy * yhat_X + COMX
L1q_Y = qx * xhat_Y + qy * yhat_Y + COMY
L1q_Z = qx * xhat_Z + qy * yhat_Z + COMZ '
```

```
    L1p_X = px * xhat_X + py * yhat_X + COVX
    L1p_Y = px * xhat_Y + py * yhat_Y + COVY
    L1p_Z = px * xhat_Z + py * yhat_Z + COVZ

    L1Test = Array(L1q_X, L1q_Y, L1q_Z, L1p_X, L1p_Y, L1p_Z)

End Function
```

The inputs to this macro are the positions of the Earth and the Moon; and the velocities of the Earth and the Moon. Each of these inputs is a 3-vector, i.e., a list of three numbers describing the X, Y and Z coordinates of the position or velocity.

The outputs of the macro is a list of six number: the first three specify the position of the L1 point; and the second thee numbers the velocity of that point in Orbiter's inertial global coordinate system.

The user-defined for the L2 Lagrange point is very similar to that for the L1 Lagrange point. In fact, aside from a few label changes, the only material change is a change in one coefficient. So, for completeness, here is the VBA user-defined function for the L2 Lagrange point:

Code:

```
Public Function L2Test(QE As Object, QM As Object, PE As Object, PM As Object) As Variant

    Dim QX_E, QY_E, QZ_E, PX_E, PY_E, PZ_E As Double
    Dim QX_M, QY_M, QZ_M, PX_M, PY_M, PZ_M As Double

    QX_E = QE(1): QY_E = QE(2): QZ_E = QE(3)
    QX_M = QM(1): QY_M = QM(2): QZ_M = QM(3)
    PX_E = PE(1): PY_E = PE(2): PZ_E = PE(3)
    PX_M = PM(1): PY_M = PM(2): PZ_M = PM(3)

    Dim GM1, GM2, GM, MU1, MU2, alpha As Double

    GM1 = 398600440157821#
    GM2 = 4902794935300#
    GM = GM1 + GM2
    MU1 = GM1 / GM
    MU2 = GM2 / GM
    alpha = 1.15568211143362

    Dim COMX, COMY, COMZ As Double
    Dim COVX, COVY, COVZ As Double

    COMX = MU1 * QX_E + MU2 * QX_M
    COMY = MU1 * QY_E + MU2 * QY_M
    COMZ = MU1 * QZ_E + MU2 * QZ_M
    COVX = MU1 * PX_E + MU2 * PX_M
    COVY = MU1 * PY_E + MU2 * PY_M
    COVZ = MU1 * PZ_E + MU2 * PZ_M

    Dim RX, RY, RZ As Double
    Dim VX, VY, VZ As Double

    RX = QX_M - QX_E
    RY = QY_M - QY_E
    RZ = QZ_M - QZ_E
```

```
VX = PX_M - PX_E
VY = PY_M - PY_E
VZ = PZ_M - PZ_E

Dim vsq, rln, rv, eX, eY, eZ, ecc, a, nu As Double

vsq = VX * VX + VY * VY + VZ * VZ
rln = Sqr(RX * RX + RY * RY + RZ * RZ)
rv = RX * VX + RY * VY + RZ * VZ

eX = RX * vsq / GM - VX * rv / GM - RX / rln
eY = RY * vsq / GM - VY * rv / GM - RY / rln
eZ = RZ * vsq / GM - VZ * rv / GM - RZ / rln
ecc = Sqr(eX * eX + eY * eY + eZ * eZ)
a = GM / (2# * GM / rln - vsq)
nu = Application.Acos((eX * RX + eY * RY + eZ * RZ) / ecc / rln)
If rv < 0 Then nu = 2# * Application.Pi() - nu

Dim xhat_X, xhat_Y, xhat_Z As Double
Dim yhat_X, yhat_Y, yhat_Z As Double
Dim zhat_X, zhat_Y, zhat_Z As Double

Dim temp As Double

xhat_X = eX / ecc
xhat_Y = eY / ecc
xhat_Z = eZ / ecc

zhat_X = RY * VZ - RZ * VY
zhat_Y = RZ * VX - RX * VZ
zhat_Z = RX * VY - RY * VX

temp = Sqr(zhat_X * zhat_X + zhat_Y * zhat_Y + zhat_Z * zhat_Z)

zhat_X = zhat_X / temp
zhat_Y = zhat_Y / temp
zhat_Z = zhat_Z / temp

yhat_X = zhat_Y * xhat_Z - zhat_Z * xhat_Y
yhat_Y = zhat_Z * xhat_X - zhat_X * xhat_Z
yhat_Z = zhat_X * xhat_Y - zhat_Y * xhat_X



Dim k1, k2, cnu, snu, k3 As Double

k1 = a * (1# - ecc * ecc)
k2 = Sqr(GM / k1)
cnu = Cos(nu)
snu = Sin(nu)
k3 = 1# + ecc * cnu

Dim qx, qy, px, py As Double

qx = alpha * k1 * cnu / k3
qy = alpha * k1 * snu / k3
px = -alpha * snu * k2
py = alpha * (ecc + cnu) * k2

Dim L2q_X, L2q_Y, L2q_Z As Double
Dim L2p_X, L2p_Y, L2p_Z As Double
```

```
    L2q_X = qx * xhat_X + qy * yhat_X + COMX
    L2q_Y = qx * xhat_Y + qy * yhat_Y + COMY
    L2q_Z = qx * xhat_Z + qy * yhat_Z + COMZ

    L2p_X = px * xhat_X + py * yhat_X + COVX
    L2p_Y = px * xhat_Y + py * yhat_Y + COVY
    L2p_Z = px * xhat_Z + py * yhat_Z + COVZ

    L2Test = Array(L2q_X, L2q_Y, L2q_Z, L2p_X, L2p_Y, L2p_Z)

End Function
```

In much the same way, one can set up user-defined functions for the other Lagrange points L3, L4 and L5. But most of the time, users are likely to be more interested in L1 and L2.

I have created two targeting spreadsheets which anyone should be able to download from Dropbox. The first is set up to target the Earth-Moon L1 Lagrange point. The link to the spreadsheet is S4INT-EML1.xls. And the second is set up to target the Earth-Moon L2 Lagrange point. The link to the second spreadsheet is S4INT-EML2.xls. (Yes, 'boogabooga', I'll get around to putting these on Orbit Hangar - but not just yet.)

As with the earlier spreadsheet designed to get out to the Moon only, these spreadsheets provide rudimentary techniques for setting up a manoeuvre to target the relevant Lagrange point from Low Earth Orbit. As before, the spreadsheet tool may be basic (and slow!) but it is very accurate. Each spreadsheet also contains information about the MJD of closest approach to the relevant Lagrange point and the speed relative to the Lagrange point at the point of closet approach. This information allows one to null out one's velocity upon arrival at the Lagrange point. With these tools, it should be possible (with a little effort and patience) to target L1 and L2 with high precision.

As with the earlier spreadsheet, I'll put together a short video series showcasing use of one of the spreadsheets (the L1 version in fact). Just to make life interesting I will park one Deltaglider at the L1 Lagrange point, and take a second Deltaglider from Low Earth Orbit to rendezvous with it. Although I haven't done this yet, I fully expect to arrive at L1 with a few hundred metres of the first Deltaglider.

Again, as with the earlier spreadsheet targeting tool, it can do with some enterprising developer converting the VBA code into a faster and more robust C/C++ MFD form....

I had the privilege to be that "enterprising developer" Keith mentioned in his last sentence above!

Credits:

- The whole concept and all the early testing with me: Keith "Keithth G" Gelling. I hope you get to see this work and that it brings you as much satisfaction using it as it did for me coding it. Missing you!
- The extended Alpha test team: Dimitris "dgatsolis" Gatsoulis, Brian "Brian J" Jones, ZT "Nicholas Kang" Kang. Thanks for the constant feedback on the weekly alphas, and your detailed tests to get it to where it is today.
- Code advice and co-developer of ModuleMessagingExt: Szymon "Enjo" Ender, and for Szymon's Button Handling framework. Thanks for all the encouragement. Hope you get to see this on your Linux hybrid environment in due

course.