When learning orbital mechanics, it can sometimes be difficult to understand certain subjects without a visual aid. Topics such as orbital elements, interplanetary travel, and multi-body perturbations can be difficult to visualize and model without the assistance of computational software. This guide is designed to assist your knowledge and understanding of these complicated topics with the use of the helpful tools built in to FreeFlyer®.

FreeFlyer® is a space simulation program designed to visualize and model various scenarios including, but not limited to, spacecraft propagation, spacecraft maneuvering, coverage and contact analysis, interplanetary analysis, and generating various visual aids. FreeFlyer® also has an intuitive and flexible scripting capability which allows the user to use FreeFlyer® in any way needed.
# Table of Contents

Ch 1 **Getting Started Guide** 5
  1.1 Installing FreeFlyer ................................................................. 7
  1.2 Navigating FreeFlyer ............................................................... 9
  1.3 Starting Your First Mission ..................................................... 14

Ch 2 **Orbital Elements Tutorial** 30
  2.1 Orbit Shapes and Sizes ............................................................ 32
  2.2 Orbit Orientation ................................................................. 39

Ch 3 **Spacecraft Attitude** 49
  3.1 Attitude State Representations ............................................... 51
  3.2 Attitude Reference Frames .................................................... 69
  3.3 Mission Constraints on Spacecraft Attitude ............................. 83

Ch 4 **Maneuvering Tutorial** 92
  4.1 Hohmann Transfer ............................................................... 94
  4.2 Bi-Elliptic Transfer ............................................................... 100
  4.3 Phasing Maneuver ............................................................... 108
  4.4 Plane Change Maneuver ........................................................ 115

Ch 5 **Interplanetary Topics** 120
  5.1 Interplanetary Hohmann Transfer .......................................... 122
  5.2 Patched Conics Transfer ....................................................... 131
  5.3 Gravity Assist ................................................................. 148
  5.4 The B-Plane ................................................................. 154

Ch 6 **Targeting Tutorial** 162
  6.1 Targeting Concepts .............................................................. 164
  6.2 Inclination Change ............................................................. 166
  6.3 B-Plane Targeting ............................................................. 172

Ch 7 **Real World Modeling** 190
  7.1 Multi-Body Effects ............................................................. 192
  7.2 J2 Perturbation ................................................................. 196
  7.3 Atmospheric Modeling ........................................................ 202
  7.4 Solar Radiation Pressure ...................................................... 206
CHAPTER 1

Getting Started Guide
FreeFlyer® is a space simulation program designed to simulate and model various scenarios including, but not limited to spacecraft propagation, spacecraft maneuvering, coverage and contact analysis, interplanetary analysis, and generating plots, charts, and videos. Also, FreeFlyer has a scripting capability which allows the user to use the tool in whatever way they desire.

This Getting Started Guide is designed to help you get familiar with the basics of FreeFlyer and help you construct your first Mission Plan. If you follow this guide from start to finish, you should have no problem setting up your first mission.

This guide is divided into three sections:

1. Installing FreeFlyer
2. Navigating FreeFlyer
3. Starting Your First Mission
1.1 - Installing FreeFlyer

To download and install FreeFlyer, register for an account with a.i. solutions with your full name and .edu email address and sign in at:

https://ai-solutions.com/freeflyer-login/

NOTE: A copy of FreeFlyer may be available to you from your school or professor. In that event, registration is not required.

Students are eligible for a free Educational License for FreeFlyer.

Once signed in, go to the "Downloads" page to download and install the most recent version of FreeFlyer. NOTE: We only offer a headless version of FreeFlyer for Linux. For the purpose of this guide, we do not recommend that you install this version.

FreeFlyer Downloads

<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
<th>Date Uploaded</th>
</tr>
</thead>
<tbody>
<tr>
<td>FreeFlyer_7.4.0_Help_Linux.zip</td>
<td>77.50M</td>
<td>November 29, 2018</td>
</tr>
<tr>
<td>FreeFlyer_7.4.0_Help_Windows.zip</td>
<td>49.15M</td>
<td>November 29, 2018</td>
</tr>
<tr>
<td>FreeFlyer_7.4.0_Installer_Linux_RedHat6_64-Bit.zip</td>
<td>227.25M</td>
<td>November 29, 2018</td>
</tr>
<tr>
<td>FreeFlyer_7.4.0_Installer_Linux_RedHat7_64-Bit.zip</td>
<td>259.96M</td>
<td>November 29, 2018</td>
</tr>
<tr>
<td>FreeFlyer_7.4.0_Installer_Windows_64-Bit.zip</td>
<td>666.89M</td>
<td>November 29, 2018</td>
</tr>
<tr>
<td>FreeFlyer_Extensions_SDK.zip</td>
<td>2.61M</td>
<td>November 29, 2018</td>
</tr>
</tbody>
</table>

FreeFlyer Download Page

When your download is complete, run the FreeFlyer_X.X.X.X_Installer.msi installer. The install wizard will walk you through installing FreeFlyer. Note that if your system does not yet have the .NET 4.7.2 framework installed, you will be prompted to install .NET 4.7.2 first; the installer is available in the same folder as the FreeFlyer installer and is called NET472.exe.

When you open FreeFlyer for the first time, it will prompt you for a license.
In order to obtain your license file, copy the information given to you and email it to fflicense@ai-solutions.com. You should receive an email with your license file within 1 business day.

If you are connecting to a network license server at your school, you do not need to request a license file. Choose "Connect to License Server" in the Registration Method dropdown, and then type in the server information given to you.

If you requested a license file (and are not using a license server) and have received it, click "Browse" to select that file, and then click "Register License File". Once you have done that, you can begin using FreeFlyer!

**See Also**

- [Getting Started Guide](#)
- [Next Topic: Navigating FreeFlyer](#)
1.2 - **Navigating FreeFlyer**

This section will discuss how to navigate the various sections of FreeFlyer.

FreeFlyer has four main screens:

1. **The Home Screen**
2. **The Control Screen**
3. **The Output Screen**
4. **The Notes Screen**

---

**The Home Screen**

1. **The Menu Bar**
   - Open, Close, and Save Mission Files as well as change user preferences, manipulate workspaces, and find help and documentation on questions you may have.

2. **Quick Tasks**
   - Create New Mission Plans, browse existing Sample Mission Plans, or use the Mission Plan Wizard to automatically generate a Mission Plan for simple analysis problems.

3. **Learn About FreeFlyer**
   - Read up on the added features of the latest update, and also view video tutorials.
4. **Recent Files**  
Open recently modified Mission Plans.

5. **Demonstration Mission Plans**  
Mission Plans that demonstrate FreeFlyer’s advanced analysis capabilities.

6. **Sample Mission Plans with New Features**  
Mission Plans that demonstrate the latest features added to FreeFlyer.

7. **Featured Mission Plan Wizards**  
Initiates a ‘wizard’ which creates a Mission Plan to solve a simple analysis.

8. **Contact Information**  
Contact information for the FreeFlyer team for license requests, sales information, and technical support.

9. **Tip of the Day**  
Provides a random tip regarding a functionality of FreeFlyer.

---

### The Control Screen

1. **Navigation Bar**

   - ![New Mission Plan (CTRL + N)](image)
   - ![Syntax Check (CTRL + Q)](image)
Open Mission Plan (CTRL + O)  Run Mission (CTRL + R)
Save, Save As (CTRL+S), (CTRL + SHIFT + S)  Record Output
Close Mission Plan (CTRL + SHIFT + C)  Pause (CTRL + P)
Undo (CTRL + U)  Stop (CTRL + T)
Redo (CTRL + R)

Controls the Speed of the Mission Plan Execution

NOTE: Mission Plans created using the "New Mission Plan" icon or CTRL + N will be created in nanosecond precision mode, which stores all time values to nanosecond-level precision. You can use the drop-down arrow next to the "New Mission Plan" icon to create a Mission Plan in millisecond precision mode instead, which will store all time values at millisecond-level precision. Nanosecond mode and millisecond mode use different syntax in many cases. This guide is written for nanosecond mode syntax, and we recommend that you create all Mission Plans in nanosecond mode by clicking the "New Mission Plan" icon or using the CTRL + N shortcut.

2. **Object Browser**
Add Objects to the Mission Plan and organize them by category.
NOTE: Objects created through a FreeForm script editor will not appear here.

3. **Script Elements**
List of all commands and flow control elements. You can drop and drag these onto the Mission Sequence, or directly into a FreeForm script editor.

4. **Mission Sequence**
The sequence of how the Mission Plan will run. You can comment out sections of code by clicking the "//" column next to the block you wish to comment out. Also, you can set breakpoints by clicking the red dot column next to the block you wish to set a breakpoint.

5. **Status Bar**
Shows the status of the mission as "Running", "Paused", "Stopped", etc. Also displays the run duration of the Mission Plan execution.

---

The Output Screen
The Output Screen

1. **Workspace**
   A new workspace will appear every time you run the Mission Plan. You can easily compare output by looking at different workspaces.

2. **Output Properties**
   Contains modifiable properties of output windows (View Windows, Plot Windows, Map Windows, etc.) to help post-process data to include in presentations.

The Notes Screen
The Notes Screen

The Notes Screen provides a space for you to capture notes and comments regarding your Mission Plan.

See Also

- [Getting Started Guide](#)
- [Next Topic: Starting Your First Mission](#)
- [Previous Topic: Installing FreeFlyer](#)
1.3 - Starting Your First Mission

In this section, we will start our very first mission! For this scenario, we have the Keplerian Elements of two different spacecraft. We want to compare the speeds of those spacecraft during a 2 day period.

The topics discussed in this tutorial are the following:

1. Adding Spacecraft
2. Adding Other Objects
3. Writing the Mission Plan - Drag and Drop
4. Writing the Mission Plan - FreeForm
5. The Report Command
6. Running the Mission
7. Adding Tanks and Thrusters
8. Adding Impulsive Burns
9. Using Impulsive Burns

Let's make the Mission Plan!

- To start, on the Home Screen click "Create a New Mission Plan"

Adding Spacecraft

- Right Click in the Object Browser and Press
  - Add → Spacecraft

- Double Click "Spacecraft1"
- Change the Element Type from "Cartesian" to "Keplerian"
- Enter the orbital parameters for Spacecraft 1:
  - A: 7100 km
  - E: 0.05
- $I$: 0 deg
- RAAN: 0 deg
- $W$: 0 deg
- TA: 0 deg

**Spacecraft Orbit Editor**

- Click on “Propagator” in the Left Hand side of the Spacecraft Window under “Motion Model”
- Change the Step Size from 300s to 100s
FreeFlyer has the ability to modify the Force Model used on the Spacecraft. Here you can change the gravitational model and atmospheric model used in the simulation. You can also change the bodies used in force calculations, as well as simulate the effects of Solar Radiation Pressure. However, our simulation does not require us to change any of these options.

- Under "Primary Options,” click on “Physical Properties”

In this section, you can change the Spacecraft’s Aerodynamic properties as well as its Mass and Moment of Inertia. These properties are especially critical when using Low Earth Orbiters over longer periods of time, or when your Spacecraft uses thrusters.

- Under "Dry Inertial Properties”, change the Mass to 200 kg.
Click “Ok” to save your changes and close the Spacecraft Editor.

We need to add the Second Spacecraft in as well. This time, we will make sure it has a different tail color.

- Right Click the Object Browser and Press
  - Add → Spacecraft
- Double Click “Spacecraft2”

This time, we can use the Orbit Wizard to create a Spacecraft with a pre-defined orbit.

- Click on “Orbit Wizard” at the bottom right of the Spacecraft editor
- Choose “Molniya” from the “Orbit Type” dropdown

- Click “Next”
- Leave all the default values
- Click “Finish”
- Click on “Propagator” in the Left Hand side of the Spacecraft editor
- Change the Step Size from 300s to 100s
- Click on Visualization on the left-hand side
- Change the tail color to green
- Go to the 3D Model dropdown and select “TDRS” (This will change the appearance of the Spacecraft)
Great! We have created our Spacecraft. Now we need to add in a few more objects through the Object Browser.

**Adding Other Objects**

First, we will add in one of the more important Objects: a ViewWindow. This will allow us to view our Spacecraft orbiting a 3D globe. If we so desired, we could plot its ground track on a 2D map as well. Let’s add in the ViewWindow:

- Right Click the Object Browser
  - Add → Output → ViewWindow
- Double Click “ViewWindow1”
- Check both boxes for Spacecraft 1 and 2 in the “Available Objects” so that they are visible in the ViewWindow.
- We want the names of the Spacecraft visible as well
  - Click on “Spacecraft” to select both Spacecraft objects
    - **Note:** Selecting "Spacecraft" will select all the Spacecraft objects, but you can also select individual Spacecraft by simply clicking on their name, or multiple Spacecraft with CTRL + click
  - Check the "Show Name" box on the right hand side so that both Spacecraft will have their names visible

• Click “Ok”
Now, we want to modify one of the viewpoints of the ViewWindow so we have a good view of both Spacecraft. Also, we’d like to have a chase viewpoint of Spacecraft2.

- In the left-hand side of the ViewWindow editor, Click on “Viewpoints”
- For Reference Frame, click “Inertial”
- To add the second viewpoint, Click “Create”
- Change the name to “ChaseSC2”
- Change the title to “Chase SC2”
- Change source to “Spacecraft2” and click “Copy to Target” to set the Target as Spacecraft2 as well
- Change the Radius to 3000 km

Click “Ok” to save your changes and close the ViewWindow Editor

Now that we have our ViewWindow all set up, we still need one more thing – a PlotWindow. This will allow us to graph the velocities of the Spacecraft.
Right Click the Object Browser
  - Add → Output → PlotWindow
Double click "PlotWindow1"
Click on the Y-Axis dropdown menu
Under "Choose a property or method", select “VMag”
  - To navigate more quickly, you may press the first letter of what you’re looking for to jump to the beginning of that alphabetical section
  - VMag is the magnitude of the velocity vector, which is what we are looking for

![PlotWindow Editor](image)

- Click “More” so we can add another line to the plot
- Click on the second dropdown that just appeared
- Under “Choose an object”, select “Spacecraft2”
- Under “Choose a property or method”, select “VMag”
- Click “Ok”

Now that we have all the objects we need for our Mission Plan, let’s move on to the actual Mission Sequence.

**Writing the Mission Plan - Drag and Drop**
The easiest way you can create your Mission Sequence is to drag and drop Script Elements onto the Mission Sequence. To start, we will need to put in a while loop that will last us 2 days. You will likely use while loops in most missions you create.

- Drag and drop “While...End” from the Script Elements on the right-hand side into the Mission Sequence portion of your window.

  ![Scripting Elements and While Loop inside Mission Sequence](image)

  - Double Click the first line which should read “While (Spacecraft1.ElapsedTime < TIMESPAN(1 days));”

The Mission Sequence will loop through until the condition inside the parenthesis is met. For our mission, we want it to run for 2 days.

  - Click on the dropdown that currently reads “TIMESPAN(1 days)”
  - Under "Enter a time span value", delete "1" and enter "2"

We can continue dragging and dropping elements into the Mission Sequence, but for practice we are going to write the rest of the mission in FreeForm, FreeFlyer’s scripting editor.

**Writing the Mission Plan - FreeForm**

- Drag and drop a “FreeForm” from the “Script Elements” section on the right-hand side into the While loop.
- Double-click "FreeForm: FreeForm"
- So we know what this script does, let's change the FreeForm label to “Step and Update”

  ![FreeForm Label](image)
FreeFlyer script allows us to bypass the limitations and constraints of doing things through the GUI, and also allows us to write complex math functions or logic control that are otherwise not built in to FreeFlyer. Anything that can be done through the Object Browser and Drag-and-Drop can also be done through a FreeForm script editor.

To start off, we need to step both of the Spacecraft forward one step (Remember when we set the step size to 100s?) First, we will step forward Spacecraft1. To do this, we will write:

```
Step Spacecraft1;
```

To step Spacecraft2, we will do so with what is called an "Epoch Sync". This ensures that both Spacecraft will be simulated to the same points in time, syncing their epochs. This is always a good practice when dealing with multiple Spacecraft. To do this, we will write:

```
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
```

Now, we need to update the PlotWindow so that the current Spacecraft speed is written to the plot. To do this, we will write:

```
Update PlotWindow1;
```

Next, we need to update the ViewWindow to show our Spacecraft at its current state. To do this, we will write:

```
Update ViewWindow1;
```

Overall, your script should look something like this:

```
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
Update PlotWindow1;
Update ViewWindow1;
```

For more on FreeForm syntax and usage, go to

```
Help → Show Help Contents → Guides → FreeFlyer Scripting → Syntax and the FreeForm Script Editor
```

For help on other parts of FreeFlyer that you may have questions about, you can right-click on the element in question and click "Go To Help File." This will bring you to the FreeFlyer Help File and give you information about the element you right-clicked.
The Report Command

Report commands can be used to print text or numerical data to a data table. For this scenario, we will print the time, spacecraft altitude, and spacecraft velocity for each spacecraft.

- Click on the “Mission Sequence” tab to go back to the Mission Sequence
- Drag and drop a FreeForm script editor underneath the “Step and Update” FreeForm we just created (still inside the While loop)
- Double-click the FreeForm
- Change the name label to “Report Time, Alt, and Velocity” so we know what this script does

To report the data, we need to write the following:

```
Report Spacecraft1.EpochText, Spacecraft1.Height, Spacecraft1.VMag;
Report Spacecraft2.EpochText, Spacecraft2.Height, Spacecraft2.VMag;
```

You’ll notice that as soon as you press the period key after an object, FreeFlyer will give you a scrollable list of properties (‘P’) and methods (‘M’) applicable to that object along with a description of that property or method. The auto complete logic allows you to scroll and either double-click or press enter to have FreeFlyer automatically type the property or method you’ve chosen.

Your Mission Sequence should now look like this:
Running the Mission

- Press the play button on the navigation bar
  - This will automatically run a Syntax Check, and then start the mission
- Four windows should pop up: The ViewWindow, PlotWindow, and two Report Windows

- Double Click the top of the MissionView Window to maximize it
- Click and hold the left mouse button to rotate around the scenario
- Click and hold the right mouse button to zoom in and out
- As stated before, to modify any of the output properties, you can modify the components in the Output Properties Section on the Right Hand side.
- To change the viewpoint to the chase viewpoint we set up earlier, in “Output Properties” change the Viewpoint to “Chase SC2”

If you wish to switch to other full screen outputs, on the bar just below the Workspace tabs, you’ll see each output at the top.

- Switch to the Velocity Plot by clicking on the “Plot” tab below the workspace tabs
Plot Tab

- To zoom in on a section of the plot, click and drag to highlight the section of where you would like to zoom in.
- To zoom out, right-click the plot.
- You can modify the output properties of the plot in the “Output Properties Section” on the Right Hand Side.
- If you need to export the plot, click on the button labeled “Export”
  - You can export it as an image file, or a data file

Once the Mission Plan finishes running and comes to a stop, try to answer the following questions:

Which Spacecraft has the fastest top speed?

Which Spacecraft shows the least change in speed during its orbit?

Looking at the graph, about how long is Spacecraft2’s orbital period?

Adding Tanks and Thrusters

Let’s add something in to complicate the problem. Let’s assume that 8 hours into the scenario, Spacecraft1 speeds up 1 km/s in the direction it is traveling. How do we model this?

- Click on the “Control” button on the Navigation Bar to return to the Mission Sequence
- Double-click on Spacecraft1
- Under “Subsystems” click on “Tanks”
- Click “Create”
- Click “Edit Tank” to see the properties

We just added a tank to Spacecraft1. In the Tank editor, we can see the different properties you can edit.

- Change the Total Tank Volume to 0.25 m^3
- Change the Fuel Mass to 200 kg
Click “Ok” to close the Tank editor
Under “Subsystems” in the Spacecraft editor, click on “Thrusters”
Click “Create”

Click “Edit Thruster” to see the properties

We just added a Thruster object to Spacecraft1. In the Thruster editor, we can see the various properties that can be changed. We will not make any modifications for this example.

Click “Ok” to close the Thruster editor
Click “Ok” to close the Spacecraft editor

Adding Impulsive Burns
In order to use an ImpulsiveBurn, you must create it as an object in the Object Browser. Once created, any Spacecraft can maneuver using that ImpulsiveBurn.

- Right-click on the Object Browser
  - Add → Spacecraft Related → Impulsive Burn
- Double-click on "ImpulsiveBurn1"
- Change the "Attitude System" to "VNB" (This is important so that the burn direction is correct)
- Leave the default burn magnitudes, which should be:
  - Velocity: 1km/s
  - Normal: 0km/s
  - Binormal: 0km/s

![ImpulsiveBurn Editor](image)

- Click "Ok" to close the editor

**Using Impulsive Burns**

We've stated that this burn occurs at about 8 hours into the mission. We need to edit our while loop to adjust for this.

- Double-click on the while loop in the Mission Sequence
- Click on the "TIMESPAN(2 days)" dropdown
- Change the "Enter a time span value" field currently set at "2" to "8"
- In the "Choose the time span units" dropdown, select hours
- Click "Ok"

This makes it so the Spacecraft will propagate normally, before the impulsive burn occurs. As soon as 8 hours has passed, the Mission Sequence will exit the loop, and we will perform the burn.
After the while loop in the Mission Sequence, drag and drop "Maneuver" from the "Script Elements" section on the right hand side.

- Double-click on that line (Should be Line 5)
- Uncheck the box that says “Create Report?”
- Click “Ok”

Once line 5 executes, we still need the mission to continue. To do this, we need to put in a while loop for the remaining 40 hours of the mission.

- Drag and Drop “While…End” from the "Script Elements" section onto the bottom of the Mission Sequence
- Double-click on the loop
- Click on the "TIMESPAN(1 days)" dropdown
- Change the "Enter a time span value" field currently set at "1" to "40"
- In the "Choose the time span units" dropdown, select hours
- Click “Ok”

Inside the while loop, we want to do the same thing we were doing in the previous while loop. Instead of putting in a new FreeForm script editor and rewriting it, we can clone the “Step and Update” script we wrote.

- Right-click on “FreeForm: Step and Update”
- Click on “Clone Selected”
- Drag the newly cloned FreeForm script editor into the second while loop

Your Mission Sequence should now look like this:

Now our mission is ready to run! This time, Spacecraft1 will change its orbit 8 hours into the mission.

- Click the Play button on the Navigation Bar

Now that we have successfully run our mission with the impulsive burn, try to answer these questions:

**Look at the plot. How does Spacecraft1’s speed compare Pre-Burn to immediately Post-Burn?**

**What happened to Spacecraft1’s overall variance in velocity?**
We hope that the rest of this guide is helpful in your understanding of orbital mechanics and spacecraft flight dynamics. Remember to utilize the help file if you have any questions about FreeFlyer! You may do this by pressing F1, or right clicking the element you have questions about and clicking "Go To Help File." If the help file couldn't quite answer your question, you may email techsupport@ai-solutions.com. Also, be sure to check out http://www.ai-solutions.com for internship and career opportunities!

**See Also**

- [Getting Started Guide](#)
- [Previous Topic: Navigating FreeFlyer](#)
In this chapter, we will discuss the six classical orbital elements, otherwise known as "Keplerian elements." We will learn about how to manipulate an orbit's shape, size, and orientation in respect to its central body.

This chapter will cover the following topics:

1. Orbit Shapes and Sizes
2. Orbit Orientation
2.1 - Orbit Shapes and Sizes

In this section, we will discuss the various aspects of an orbit that make up its shape and size. Two of the six Keplerian elements will be discussed. They are:

1. Semi-Major Axis ('a')
2. Eccentricity ('e')

Semi-Major Axis

The Semi-Major Axis (referred to as 'SMA' or 'a') is the distance from the center of an ellipse to the longer end of the ellipse. In a circle, the SMA is simply the radius.

The semi-major axis determines various properties of the orbit such as orbital energy and orbital period. As the semi-major axis increases, so does the orbital energy and the orbital period.

Problem:
We have three spacecraft orbiting at three different semi-major axes. All three spacecraft orbit in a circular, equatorial orbit. Simulate these spacecraft using FreeFlyer and output each Spacecraft object's orbital period.

- Spacecraft 1 - 7,000 km
- Spacecraft 2 - 15,000 km
- Spacecraft 3 - 42,164 km

Let's begin our scenario. Open up a new Mission Plan and label it "SMAperiod.MissionPlan".

Adding in Spacecraft

- Create a new Spacecraft object by right-clicking the Object Browser on the left-hand side of the Control Screen
- Double-click "Spacecraft1" to open the Spacecraft editor
• Change the element type to "Keplerian"

![Element Type Dropdown in the Spacecraft Editor]

• Put in the following orbital parameters:
  o A: 7000 km
  o E: 0
  o I: 0 deg
  o RAAN: 0 deg
  o W: 0 deg
  o TA: 0 deg

• Click "Ok" to close the Spacecraft editor

• Right-click "Spacecraft1"
• Click "Clone"
Cloning a Spacecraft

- Double-click "Spacecraft1_Copy1"
- In the top right, rename the Spacecraft to "Spacecraft2"
  
  Note: You can also quickly change a Spacecraft's name by clicking on it in the Object Browser and pressing "F2"
- Change A to 15000 km
- On the left-hand side, click on "Visualization"
- Change the tail color to green
- Click "Ok" to close the Spacecraft editor

- Right-click "Spacecraft2"
- Click "Clone"
- Double-click "Spacecraft2_Copy1"
- Rename the Spacecraft to "Spacecraft3"
- Change A to 42164 km
- Change the tail color to yellow

Adding a ViewWindow

- Create a new ViewWindow by right-clicking the Object Browser
- Double-click "ViewWindow1" to open the ViewWindow editor
- Check each Spacecraft in the "Available Objects" section
- Click on "Spacecraft" to select the group of all three Spacecraft, and check "Show Name"
- Change the "History Mode" to "Unlimited" (all three Spacecraft should still be selected, so this will change the History Mode for all Spacecraft)
- On the left-hand side, navigate to "Viewpoints"
- Click on the "Default" viewpoint
- Change the reference frame to "Inertial"
- Click "Ok" to close the ViewWindow editor

Building the Mission Sequence

- From the Script Elements browser, drag and drop a "While...End" loop into the Mission Sequence
- Change the while loop stopping condition to "Spacecraft1.ElapsedTime < TIMESPAN(2 days)"
- Drag and drop two "Step" commands into the while loop
- The first Step command should say "Step Spacecraft1"
- Double-click the second Step command
- Change the "What to Step" to "Spacecraft2"
- Check the "Step to Condition?" box to step to a condition
- Change the first drop down to "Spacecraft2.Epoch"
- Change the middle dropdown menu from "<" to "=="
- Click on the dropdown that says "0"
- Change "Number" to "Object/Property Method"
- Change the last dropdown to "Epoch"
- The statement should now say "Spacecraft2.Epoch == Spacecraft1.Epoch"
Step Command Editor

- Click "Ok" to close the editor
- Clone this command by right-clicking the "Step Spacecraft2" command and click "Clone Selected"
- Change the parameters of the new Step command to say "Step Spacecraft3 to Spacecraft3.Epoch == Spacecraft1.Epoch"
- Drag and drop an Update command into the while loop after the Step commands

Now that we have set up the "Step" and "Update" commands, let's add a FreeForm script editor to report each Spacecraft object's orbital period.

- From the Script Elements browser, drag and drop a FreeForm script editor into the while loop after the "Update" command
- Double-click the FreeForm script editor to open it
- Rename it to "Report Orbital Periods"

This script will be a simple one. It should read as follows:

```
```

Now our Mission Plan is ready to run! Your Mission Sequence should look something like this:
Save your progress and then run to execute the Mission Plan. Then, try to answer these questions:

Which Spacecraft has the shortest period?

Which Spacecraft has the longest period?

What is the significance of Spacecraft3’s period?

**Eccentricity**

Orbital eccentricity is defined as a ratio of the distance between foci to the entire length of the major axis (2x the semi-major axis). Different values of eccentricities determine what kind of orbit exists.

<table>
<thead>
<tr>
<th>Eccentricity</th>
<th>Type of Orbit</th>
</tr>
</thead>
<tbody>
<tr>
<td>e = 0</td>
<td>Circular</td>
</tr>
<tr>
<td>0 &lt; e &lt; 1</td>
<td>Elliptical</td>
</tr>
<tr>
<td>e = 1</td>
<td>Parabolic</td>
</tr>
<tr>
<td>e &gt; 1</td>
<td>Hyperbolic</td>
</tr>
</tbody>
</table>

Remember, the eccentricity of an orbit does not affect a Spacecraft object’s orbital period or orbital energy. It simply defines the shape of the orbit.

**Problem:**

We have three spacecraft with the same SMA, but different eccentricities. Use FreeFlyer to plot the Orbital Velocities of these spacecraft and compare them.

For this problem, let's edit the "SMAPeriod.MissionPlan" file that we just made and save it under a different name.

- Open the "SMAPeriod.MissionPlan" file
Click the "Save As" button (or press CTRL + Shift + S)
Save the Mission Plan as "Eccentricity.MissionPlan"

We are going to edit each Spacecraft object’s Keplerian elements. Edit the semi-major axis and eccentricity to the following:

- **Spacecraft1** - A: 30,000 km; E: 0.05;
- **Spacecraft2** - A: 30,000 km; E: 0.25;
- **Spacecraft3** - A: 30,000 km; E: 0.7;

Once you have changed the Keplerian elements of each Spacecraft, it’s time to move on to plotting the Spacecraft velocities. Also, since we do not need to report the orbital periods anymore, let’s delete that FreeForm script editor:

- Delete "FreeForm: Report Orbital Periods" from the Mission Sequence
- Drag and drop a new FreeForm script editor into the while loop after the "Update" command
- Double-click the FreeForm script editor
- Rename it to "Plot Speeds"

For this FreeForm script editor, the code is relatively simple as well:

```plaintext
Plot Spacecraft1.ElapsedTime.ToHours, Spacecraft1.VMag, Spacecraft2.VMag,
Spacecraft3.VMag;
```

Your Mission Sequence should look something like this:

```
Mission Sequence Example

<table>
<thead>
<tr>
<th>#</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>While (Spacecraft1.ElapsedTime &lt; TIMESSPAN(2 days));</td>
</tr>
<tr>
<td>2</td>
<td>Step Spacecraft1;</td>
</tr>
<tr>
<td>3</td>
<td>Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);</td>
</tr>
<tr>
<td>4</td>
<td>Step Spacecraft3 to (Spacecraft3.Epoch == Spacecraft1.Epoch);</td>
</tr>
<tr>
<td>5</td>
<td>Update ViewWindow;</td>
</tr>
<tr>
<td>6</td>
<td>FreeForm: Plot Speeds</td>
</tr>
<tr>
<td>7</td>
<td>End;</td>
</tr>
</tbody>
</table>
```

Save the Mission Plan, and then click run. After the Mission Sequence has concluded, try and answer these questions:

*Which Spacecraft had the highest maximum velocity? Which Spacecraft had the lowest?*

*Run the Mission Plan again and look for the peak in velocity. Where in the Spacecraft's orbit did this occur?*

*Looking at the graph, how does the orbital period of each Spacecraft compare?*
What is the relationship of the orbital eccentricity and the variance in velocity?

See Also

- Orbital Elements Tutorial
- Next Topic: Orbit Orientation
2.2 - **Orbit Orientation**

We've learned how to change the shape and size of an orbit. Now we must learn the various aspects of orienting an orbit.

In this section, we will discuss and demonstrate the other four Keplerian elements. They are as follows:

1. **Inclination** ('i')
2. **Right Ascension of the Ascending Node** ('RAAN' or 'Ω')
3. **Argument of Periapsis** ('w' or 'ω')
4. **True Anomaly** ('TA' or 'ν')

A good visualization for reference is shown below:

**Inclination**

Inclination is the "tilt" of an orbit. It is described as the angle of the orbit plane above the equatorial plane. The most important thing to note about inclination is that it determines the latitudes covered by the orbit ground track.
Problem:
We have four satellites with different inclinations. We want to determine what areas of the Earth each will fly over. Model and simulate this in FreeFlyer.

- Open a new Mission Plan
- Save it as "Inclination.MissionPlan"

Adding in Spacecraft

- Create four Spacecraft objects with the following attributes:

<table>
<thead>
<tr>
<th>Spacecraft Name</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>RAAN</th>
<th>W</th>
<th>TA</th>
<th>Tail Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacecraft1</td>
<td>7000 km</td>
<td>0</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Red</td>
</tr>
<tr>
<td>Spacecraft2</td>
<td>7000 km</td>
<td>0</td>
<td>45 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Green</td>
</tr>
<tr>
<td>Spacecraft3</td>
<td>7000 km</td>
<td>0</td>
<td>90 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Yellow</td>
</tr>
<tr>
<td>Spacecraft4</td>
<td>7000 km</td>
<td>0</td>
<td>160 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Blue</td>
</tr>
</tbody>
</table>
Adding a ViewWindow

- Add a ViewWindow object through the Object Browser
- Open the ViewWindow editor by double-clicking "ViewWindow1"
- Check each Spacecraft in the "Available Objects"
- Click on "Spacecraft" to select the group of all four Spacecraft, and check "Show Name"
- Change the "History Mode" to "Unlimited" (this will change the History Mode for all four selected Spacecraft)
- Click on "Viewpoints" on the left-hand side of the editor
- Click on the "Default" viewpoint
- Change the viewpoint type from "3D View" to "2D Map"

![Viewpoint Editor]

- For the Reference Frame, check "Body Fixed"
- Click "Ok" to close the editor

Building the Mission Sequence

- Drag and drop a "While...End" loop into the Mission Sequence
- Drag and drop a FreeForm script editor into the while loop
- Double-click the FreeForm script editor
- Change the name of the FreeForm script editor to "Step and Update"

In this FreeForm script editor, we are going to Step all Spacecraft making sure to keep the epochs synced and then update the ViewWindow. This will be very similar to what we did in the Getting Started Guide. The script will look like this:

```plaintext
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
Step Spacecraft3 to (Spacecraft3.Epoch == Spacecraft1.Epoch);
Step Spacecraft4 to (Spacecraft4.Epoch == Spacecraft1.Epoch);
Update ViewWindow1;
```
Your Mission Sequence should look something like this:

```
Mission Sequence
// # Content
1 While (Spacecraft1.ElapsedTime < TIMESPAN(1 days));
2 FreeForm: Step and Update
3 End;
```

Mission Sequence Example

Save and run the Mission Plan, then try to answer these questions:

- Which Spacecraft sees the most of the Earth over time?
- Which Spacecraft sees the least of the Earth over time?
- What happens to the direction of an orbit if its inclination is greater than 90 degrees?

### Right Ascension of the Ascending Node (RAAN)

The Right Ascension of the Ascending Node (RAAN) is the longitude of the point where the spacecraft crosses the equatorial plane moving from south to north. The descending node is where the spacecraft drops through the equatorial plane, moving from the Northern Hemisphere to the Southern Hemisphere. Manipulating the RAAN can be thought of as moving the orbit around a swivel, rotating it around the axis normal to the reference plane (in this case, the equatorial plane).

We saw in the previous example that it takes several orbits for an inclined satellite to map out the Earth. What if we wanted to do this faster? If we add in more spacecraft and manipulate the RAAN, more of the Earth is covered more quickly.

**Problem:**

Add in a constellation of 4 satellites with a variance in RAAN. Also add in a ground station in Australia as our communications station. Compare the revisit times of one satellite versus an entire constellation of satellites. Model and simulate this in FreeFlyer.

- Open a new Mission Plan
- Save it as "RAANRevisit.MissionPlan"

### Adding in Spacecraft

- Create four Spacecraft with the following attributes:

<table>
<thead>
<tr>
<th>Spacecraft Name</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>RAAN</th>
<th>W</th>
<th>TA</th>
<th>Tail Color</th>
</tr>
</thead>
</table>

© 2019 a.i. solutions, Inc
<table>
<thead>
<tr>
<th>Spacecraft1</th>
<th>7000 km</th>
<th>0</th>
<th>45 deg</th>
<th>0 deg</th>
<th>0 deg</th>
<th>0 deg</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacecraft2</td>
<td>7000 km</td>
<td>0</td>
<td>45 deg</td>
<td>90 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Green</td>
</tr>
<tr>
<td>Spacecraft3</td>
<td>7000 km</td>
<td>0</td>
<td>45 deg</td>
<td>180 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Yellow</td>
</tr>
<tr>
<td>Spacecraft4</td>
<td>7000 km</td>
<td>0</td>
<td>45 deg</td>
<td>270 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Blue</td>
</tr>
</tbody>
</table>

Adding a GroundStation

- Add a preset Canberra GroundStation object through the Object Browser
  - Add → Object Preset → GroundStation → Canberra

Adding the ViewWindows

- Add a ViewWindow object through the Object Browser
- Open the ViewWindow editor by double-clicking "ViewWindow1"
- Check each Spacecraft in the "Available Objects"
- Click the "+" next to "GroundStation" in the "Available Objects" to expand the list of GroundStation objects and check "Canberra"
- Click on "Canberra" (the name, not the checkbox) to select it, and check "Show Name"
- Click on "Spacecraft" to select the group of all four Spacecraft, and check "Show Name"
- Change the "History Mode" to "Unlimited" (this will change the History Mode for all four selected Spacecraft)
- Click on "Viewpoints" on the left-hand side of the editor
- Click on the "Default" viewpoint
- Change the reference frame to "Inertial"
- Click "Ok" to close the editor
- Add another ViewWindow object through the Object Browser
- Open the ViewWindow editor by double-clicking "ViewWindow2"
- Check each Spacecraft in the "Available Objects"
- Click the "+" next to "GroundStation" in the "Available Objects" to expand the list of GroundStation objects and check "Canberra"
- Click on "Canberra" (the name, not the checkbox) to select it, and check "Show Name"
- Click on "Spacecraft" to select the group of all four Spacecraft, and check "Show Name"
- Click on "Viewpoints" on the left-hand side of the editor
- Click on the "Default" viewpoint
- Change the viewpoint type from "3D View" to "2D Map"
- For the reference frame, check "Body Fixed"
- Click "Ok" to close the editor

Building the Mission Sequence

- Drag and drop a "While...End" loop into the Mission Sequence
- Drag and drop a FreeForm script editor in the while loop
- Double-click the FreeForm script editor
- Change the name of the FreeForm script editor to "Step, Update, and Report"
This script will look almost exactly like the Inclination Mission Plan's script except for two things: we need to add a second Update command so both the 3D and 2D views are shown, and we need to report our contact times. The script should look like this:

```plaintext
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
Step Spacecraft3 to (Spacecraft3.Epoch == Spacecraft1.Epoch);
Step Spacecraft4 to (Spacecraft4.Epoch == Spacecraft1.Epoch);
Update ViewWindow1;
Update ViewWindow2;
```

Now, we need to add in our reporting. Remember, our goal is to compare the contact times of a single Spacecraft versus the contact time of any Spacecraft. To start, we'll write the logic to report the times Spacecraft1 sees Canberra. To do this, we write:

```plaintext
// Report if SC1 sees Canberra
If(Spacecraft1.Contact(Canberra)) then;
   Report "SC1 Contact: " + Spacecraft1.Epoch.ConvertToCalendarDate();
End;
```

Now, let's have the script report the time when any Spacecraft sees Canberra. To do this, we write:

```plaintext
// Report if any SC sees Canberra
If(Spacecraft1.Contact(Canberra) or Spacecraft2.Contact(Canberra) or Spacecraft3.Contact(Canberra) or Spacecraft4.Contact(Canberra)) then;
   Report "Constellation Contact: " + Spacecraft1.Epoch.ConvertToCalendarDate();
End;
```

Your Mission Sequence should look something like this:

```markdown
<table>
<thead>
<tr>
<th>Mission Sequence Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
```

Save and run the Mission Plan, then try and answer these questions:

**How many times did Spacecraft1 contact Canberra during our simulation period?**

**How many times did the entire constellation contact Canberra during our simulation period?**
Argument of Periapsis

The Argument of Periapsis (w) is a description of where the periapsis of the orbit is located relative to the ascending node. Perfectly circular orbits do not have a definitive argument of periapsis, as there is no difference in orbit radius. However, in the real world there is no such thing as a perfectly circular orbit. Realistically, all orbits we encounter have some degree of eccentricity to them. Therefore, all orbits we encounter also have an argument of periapsis.

When we manipulate the argument of periapsis, we are swiveling the orbit around the axis normal to the orbital plane. We are essentially controlling where the periapsis occurs.

For eccentric orbits, a spacecraft will have the most dwell time over the Earth at its apoapsis. Controlling the argument of periapsis would control where the periapsis occurs, and therefore where the apoapsis occurs, giving us control of where over Earth a communications satellite would have its dwell time.

Problem:
We have 4 inclined eccentric orbits. We must figure out which is best for communications over a Communications Station in Alaska.

- Open a new Mission Plan
- Save it as "ArgPeriComms.MissionPlan"

Adding in Spacecraft

- Create four Spacecraft with the following attributes:

<table>
<thead>
<tr>
<th>Spacecraft Name</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>RAAN</th>
<th>W</th>
<th>TA</th>
<th>Tail Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacecraft1</td>
<td>26610.25 km</td>
<td>0.75</td>
<td>65 deg</td>
<td>135 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Red</td>
</tr>
<tr>
<td>Spacecraft2</td>
<td>26610.25 km</td>
<td>0.75</td>
<td>65 deg</td>
<td>135 deg</td>
<td>90 deg</td>
<td>0 deg</td>
<td>Green</td>
</tr>
<tr>
<td>Spacecraft3</td>
<td>26610.25 km</td>
<td>0.75</td>
<td>65 deg</td>
<td>135 deg</td>
<td>180 deg</td>
<td>0 deg</td>
<td>Yellow</td>
</tr>
<tr>
<td>Spacecraft4</td>
<td>26610.25 km</td>
<td>0.75</td>
<td>65 deg</td>
<td>135 deg</td>
<td>270 deg</td>
<td>0 deg</td>
<td>Blue</td>
</tr>
</tbody>
</table>

Adding the ViewWindows

- Add a ViewWindow object through the Object Browser
- Open the ViewWindow editor by double-clicking "ViewWindow1"
Check each Spacecraft in the "Available Objects"
Click on "Spacecraft" to select the group of all four Spacecraft, and check "Show Name"
Change the "History Mode" to "Unlimited" (this will change the History Mode for all four selected Spacecraft)
Click on "Viewpoints" on the left-hand side of the editor
Click on the "Default" viewpoint
Change the reference frame to "Inertial"
Click "Ok" to close the editor
Add a second ViewWindow object through the Object Browser
Open the ViewWindow editor by double-clicking "ViewWindow2"
Check each Spacecraft in the "Available Objects"
Click on "Spacecraft" to select the group of all four Spacecraft, and check "Show Name"
Click on "Viewpoints" on the left-hand side of the editor
Click on the "Default" viewpoint
Change the viewpoint type from "3D View" to "2D Map"
For the reference frame, check "Body Fixed"
Click "Ok" to close the editor

Building the Mission Sequence

From the Script Elements browser, drag and drop a "While...End" loop into the Mission Sequence
Drag and drop a FreeForm script editor into the while loop
Double-click the FreeForm script editor
Change the name of the FreeForm script editor to "Step and Update"

This FreeForm script editor will look exactly like the FreeForm script editor in the RAAN Mission Plan, minus the reporting:

```plaintext
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
Step Spacecraft3 to (Spacecraft3.Epoch == Spacecraft1.Epoch);
Step Spacecraft4 to (Spacecraft4.Epoch == Spacecraft1.Epoch);
Update ViewWindow1;
Update ViewWindow2;
```

Save and run the Mission Plan, then try and answer these questions:

*Which Spacecraft had the best orbit for communications to a Ground Station in Alaska?*

*What orbital parameters would you need for an orbit with the same size and shape, but with an apoapsis over the North Pole?*

**True Anomaly**
The True Anomaly (TA) is the last Keplerian orbital element. It doesn't necessarily have anything to do with the orientation or shape of an orbit itself. However, it does have something to do with the location of a spacecraft in its orbit. The true anomaly is the angle between the Spacecraft object's current position vector and orbit periapsis vector at a given moment. When we define it in the Spacecraft object editor, we are stating where the Spacecraft will start in its orbit.

Using this concept, we can place multiple spacecraft in the same orbit spaced apart in true anomaly to reduce revisit time. You could even use this as a part of a communications network to increase your coverage.

**Problem:**
We have a set of equatorial satellites in low Earth orbit (LEO) that have a primary mission of observing the Amazon river basin. Demonstrate the advantages of multiple spacecraft in one orbit for this mission using FreeFlyer.

- Open a New Mission Plan
- Save it as "TARevisit.MissionPlan"

**Adding in Spacecraft**

- Create four Spacecraft with the following attributes:

<table>
<thead>
<tr>
<th>Spacecraft Name</th>
<th>A</th>
<th>E</th>
<th>I</th>
<th>RAAN</th>
<th>W</th>
<th>TA</th>
<th>Tail Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spacecraft1</td>
<td>7000 km</td>
<td>0</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>Red</td>
</tr>
<tr>
<td>Spacecraft2</td>
<td>7000 km</td>
<td>0</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>90 deg</td>
<td>Green</td>
</tr>
<tr>
<td>Spacecraft3</td>
<td>7000 km</td>
<td>0</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>180 deg</td>
<td>Yellow</td>
</tr>
<tr>
<td>Spacecraft4</td>
<td>7000 km</td>
<td>0</td>
<td>0 deg</td>
<td>0 deg</td>
<td>0 deg</td>
<td>270 deg</td>
<td>Blue</td>
</tr>
</tbody>
</table>

**Adding the ViewWindow**

- Add a ViewWindow object through the Object Browser
- Open the ViewWindow editor by double-clicking "ViewWindow1"
- Check each Spacecraft in the "Available Objects"
- Click on "Spacecraft" to select the group of all four Spacecraft, and check "Show Name"
- Change the "History Mode" to "Unlimited" (this will change the History Mode for all four selected Spacecraft)
- Click on "Viewpoints" on the left-hand side of the editor
- Click on the "Default" viewpoint
- Change the viewpoint type from "3D View" to "2D Map"
- For the reference frame, check "Body Fixed"
- Click "Ok" to close the editor

**Building the Mission Sequence**
From the Script Elements browser, drag and drop a "While...End" loop into the Mission Sequence
Drag and drop a FreeForm script editor into the while loop
Double-click the FreeForm script editor
Change the name of the FreeForm script editor to "Step and Update"

This FreeForm script editor will look exactly like the FreeForm script editor in the RAAN Mission Plan:

```plaintext
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
Step Spacecraft3 to (Spacecraft3.Epoch == Spacecraft1.Epoch);
Step Spacecraft4 to (Spacecraft4.Epoch == Spacecraft1.Epoch);
Update ViewWindow1;
```

Save and run the Mission Plan, then try and answer these questions:

* Approximately how long does it take Spacecraft1 to revisit the Amazon?

* With more Spacecraft, how long does it take for any Spacecraft in the constellation to revisit the Amazon?

See Also

* Orbital Elements Tutorial
* Previous Topic: Orbit Shapes and Sizes
CHAPTER 3

Spacecraft Attitude
In this chapter, we will discuss spacecraft attitude, the various attitude representations, and reference frames. We will take a look at the physical significance of these concepts, and apply them in FreeFlyer in order to visualize them better.

This chapter will cover the following topics:

1. Attitude State Representations
2. Attitude Reference Frames
3. Mission Constraints on Spacecraft Attitude
3.1 - Attitude State Representations

A spacecraft's attitude is defined as its orientation in space, and the motion of a rigid spacecraft is defined by its position, velocity, attitude, and attitude motion. You previously learned in this guide the role position and velocity play into a spacecraft's motion; that is the position and velocity describe the inputs to compute the translational motion of the center of mass of the spacecraft (orbit). Attitude and attitude motion describe the rotational motion of the spacecraft about its center of mass. Spacecraft Attitude plays an important role in high-fidelity mission design; now we are going to look at the basics and model them in FreeFlyer.

There are different ways of numerically representing the attitude state of a spacecraft. The most common representations use Euler angles, quaternions, or a direction cosine matrix ("attitude matrix" in FreeFlyer). In FreeFlyer, these are referred to as the spacecraft's "attitude system." The Attitude System specifies the method FreeFlyer uses to orient a spacecraft with respect to the chosen reference frame.

In this section, we will discuss:

1. Euler Angles
2. Calculating the Direction Cosine Matrix and Quaternions
3. Modeling the Direction Cosine Matrix and Quaternions

Euler Angles

Euler's rotation theorem states that any finite rotation of a rigid body can be expressed as a rotation through some angle about some fixed axis. Euler angle rotation is denoted by rotation angles $\phi$, $\theta$, $\psi$ about coordinate axes $i$, $j$, and $k$ respectively. The $i$-$j$-$k$ Euler angle rotation means that the first rotation by angle $\phi$ is about the $i$-axis, then $\theta$ about the $j$-axis, and finally $\psi$ about the $k$-axis. There are 12 different sequences for Euler angle rotation, and they are broken down into two types.

Type 1 - In this case, each rotation takes place about 3 different axes. This type of rotation experiences a singularity at $\theta = \pm 90^\circ$. A singularity is the loss of one degree of freedom in three-dimensional space. At these angles of $\theta$, both $\phi$ and $\psi$ have the exact same effect.

Type 2 - For this sequence, the first rotation occurs about some axis, then the second rotation occurs about another axis, and finally the third rotation occurs about the same axis as the first rotation. This rotation experiences a singularity at $\theta = 0^\circ$ and $180^\circ$. Thus at these values of $\theta$, both $\phi$ and $\psi$ have the exact same effect.
Let's take a look at some of these sequences and model them using FreeFlyer.

### Problem:
A start-up company wants to launch a remote sensing satellite to study Earth. They've hired you to demonstrate to them the use of Euler Angles to manipulate a spacecraft's attitude so they can learn how to align their sensor properly.

- Create a new mission plan and save it as "EulerAngles.MissionPlan"

### Adding in Spacecraft
- Add a Spacecraft object from the Object Browser on the left hand side of the control screen
- Double-click on Spacecraft1
- In the Orbit section of the Spacecraft Editor navigate to "Element Type"
- Click the dropdown and select "Keplerian"
- Give the Spacecraft the following Keplerian elements:
  - A: 8000 km
  - E: 0
  - I: 30 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg
Right click Spacecraft1 and rename it to "scConstantAttitude".

Click on "Attitude" in the left hand side of the Spacecraft Editor under "Primary Options", and give scConstantAttitude the following properties:

- Reference Frame: LVLH - Earth Pointing
- Attitude System: Euler Angles
- Rotation Sequence: 1-2-3
- Euler Angles and Rates: 0
Click "Propagator" on the left hand side and change the step size to 3 seconds.

Click "Ok" to close the spacecraft editor.

Right-click scConstantAttitude and clone it three times.

Double-click "scConstantAttitude_Copy1" and rename it "scRoll"

Under scRoll attitude tab, change the "Euler Rate 1" to 0.08 deg/s

Double-click "scConstantAttitude_Copy2" and rename it "scPitch"

Under scPitch attitude tab, change the "Euler Rate 2" to 0.08 deg/s

Double-click "scConstantAttitude_Copy3" and rename it "scYaw"

Under scYaw attitude tab, change the "Euler Rate 3" to 0.08 deg/s

Adding in multiple ViewWindow objects

Add a ViewWindow object through the Object Browser.

Open the ViewWindow editor by double-clicking "ViewWindow1".

In the top right change the name to "viewConstantAttitude".

Check scConstantAttitude under the "Available Objects" section.

Under scConstantAttitude "Object Options" section:

- Check Show Name
- Uncheck Show History
- Check Show Axes

Navigate over to the Viewpoints tab on the left-hand side.

Make sure the Reference Frame is checked as "body-fixed".

In the "Source" drop-down, change the source to scConstantAttitude.
Click "Copy to Target"

Change the following properties of the Default Viewpoint:
- Right Ascension: 12 deg
- Declination: -135 deg
- Radius: 1250 km

Click "Ok" to close the ViewWindow editor

Right-click "viewConstantAttitude" and clone it three times

Double-click "viewConstantAttitude_Copy1" and rename it "viewRoll"
In the "Available Objects" section:
- Keep scConstantAttitude checked
- Check scRoll

In the "Object Options" section for scConstantAttitude uncheck "Show Object"
In the "Object Options" section for scRoll:
- Check Show Name
- Uncheck Show History
- Check Show Axes

Double-click "viewConstantAttitude_Copy2" and rename it "viewPitch"
In the "Available Objects" section:
- Keep scConstantAttitude checked
- Check scPitch

In the "Object Options" section for scConstantAttitude uncheck "Show Object"
In the "Object Options" section for scPitch:
- Check Show Name
- Uncheck Show History
- Check Show Axes
- Double-click "viewConstantAttitude_Copy3" and rename it "viewYaw"
- In the "Available Objects" section:
  - Keep scConstantAttitude checked
  - Check scYaw
- In the "Object Options" section for scConstantAttitude uncheck "Show Object"
- In the "Object Options" section for scYaw:
  - Check Show Name
  - Uncheck Show History
  - Check Show Axes

Example for ViewWindow viewRoll
Building the Mission Sequence

Before building the Mission Sequence, make sure your Object Browser looks like this:

- From the Script Elements browser, drag and drop a "While...End" loop into the Mission Sequence
- Change the while loop stopping condition to "scConstantAttitude.ElapsedTime < TIMESPAN(1.5 hours)"
- From the Script Elements browser, drag and drop a FreeForm script editor inside the while loop
- Double click the FreeForm script editor and change its name to "Step and Update"

The following script is relatively simple, we just need to step our spacecraft, synchronize their epochs, and update each ViewWindow. It should look something like this:

```plaintext
// Step each Spacecraft and sync their Epochs
Step scConstantAttitude;
Step scRoll to (scRoll.Epoch == scConstantAttitude.Epoch);
Step scPitch to (scPitch.Epoch == scConstantAttitude.Epoch);
Step scYaw to (scYaw.Epoch == scConstantAttitude.Epoch);

// Update each ViewWindow
Update viewConstantAttitude;
Update viewRoll;
Update viewPitch;
Update viewYaw;
```
Now our Mission Plan is ready to run! Your Mission Sequence should look something like this:

```
<table>
<thead>
<tr>
<th>Mission Sequence</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>While (scConstantAttitude.ElapsedTime &lt; TIMESPAN(1.5 ho)</td>
</tr>
<tr>
<td>2</td>
<td>FreeForm: Step and Update</td>
</tr>
<tr>
<td>3</td>
<td>End;</td>
</tr>
</tbody>
</table>
```

Save your progress and then run to execute the Mission Plan. Then, try to answer these questions:

**What is the significance of Roll, Pitch, and Yaw?**

*Run the mission plan again, but change scYaw’s Rotation Sequence to 3-2-1. Which two Spacecraft now have identical attitude motion?*

**Bonus: Modeling a Singularity**

Let’s talk more about the singularity we mentioned at the beginning of this section. This singularity, also known as Gimbal Lock, is the loss of one degree of freedom in 3D space. Therefore, two of the rotations will have the exact same effect on the body for certain values of \( \theta \) that we discussed earlier. Let’s take a look at Type-1 sequences and model a singularity in FreeFlyer.

Keep "EulerAngles.MissionPlan" open

**Changing a Spacecraft**

- Double-click on scRoll to change its properties
- Under the Attitude tab:
  - Keep the 1-2-3 sequence
  - Euler Angle 1: 0 deg
  - Euler Angle 2: 90 deg
  - Euler Angle 3: 0 deg
  - Euler Rate 1: 0.08 deg/s
  - Euler Rate 2: 0 deg/s
  - Euler Rate 3: 0.08 deg/s
Notice how scRoll is pitched upwards by 90° but only rotating about one axis. Why is this? We made sure that both Euler Angle 1 and Euler Angle 3 are both rotating. This is the singularity we discussed earlier. Since θ is rotated 90°, both φ and ψ have the same effect. That is why it appears scRoll is rotating twice as fast as the other spacecraft, because two 0.08 deg/s rotation rates are applied in the same direction due to the singularity.

### Calculating the Direction Cosine Matrix and Quaternions

There are other ways of representing a spacecraft’s attitude. Here we will discuss the significance of the Direction Cosine Matrix along with Quaternions, and the math behind calculating these values for a given attitude state. Starting with a given set of Euler Angles, we will perform our own calculations in order to analyze the same spacecraft attitude in three different representations.

#### Problem:

We are performing the initial attitude analysis on a future satellite project. Our goal is to take the planned Euler Angle rotations and perform hand calculations of the Direction Cosine Matrix and Quaternion. We need to verify our hand calculations are correct, and visualize the spacecraft’s attitude in FreeFlyer. Our spacecraft will have a 3-1-2 sequence with rotations: φ = 20 degrees, θ = 35 degrees, and ψ = 45 degrees.

### Direction Cosine Matrix

Given an orthogonal, right handed triad $u$, $v$, and $w$ of unit vectors fixed in the body frame, such that $u \times v = w$, we can specify the components of $u$, $v$, and $w$ along the three axes of the body frame so that the body frame is fixed completely with respect to the reference frame. This sets up a 3x3 matrix with 9 parameters, $A$, called the attitude matrix:

$$A = \begin{bmatrix} u_1 & u_2 & u_3 \\ v_1 & v_2 & v_3 \\ w_1 & w_2 & w_3 \end{bmatrix}$$

**Attitude Matrix**
Each of these elements is the cosine of the angle between the body frame unit vector and the reference axis. For example, $u_2$ is the cosine of the angle between $u$ and the reference 2 axis. Because of this, $A$ is often referred to as the direction cosine matrix. The direction cosine matrix, or DCM, is a coordinate transformation that maps vectors from the reference frame to the body frame. Some properties of the DCM:

$$A^T = A^{-1}$$  

Properties of the DCM

These properties mean that the DCM is a real orthogonal matrix; therefore the transpose of $A$ maps vectors from the body frame back to the reference frame. It is possible to calculate the direction cosine matrix from Euler Angles. Each individual Euler Angle Rotation about a reference axis has its own DCM. For an arbitrary angle $\phi$, the Euler rotations about reference axes 1, 2, and 3 are:

$$A_1(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}$$

$$A_2(\phi) = \begin{bmatrix} \cos\phi & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\phi & 0 & \cos\phi \end{bmatrix}$$

$$A_3(\phi) = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Direction Cosine Matrices for Individual Rotations

The direction cosine matrix for the overall rotation sequence is the matrix product of each of the individual rotation matrices shown above, with the first rotation matrix on the right and the last rotation matrix on the left as shown by this equation:

$$A_{ijk}(\phi_1, \phi_2, \phi_3) = A_k(\phi_3)A_j(\phi_2)A_i(\phi_1)$$

Equation for Calculating DCM

We are going to calculate the DCM for a 3-1-2 Euler Sequence. Following the equation above, for a 3-1-2 sequence we have:

Note: Every Euler angle rotation sequence has its own DCM; this is the DCM for a 3-1-2 rotation.

$$A_{312}(\phi, \theta, \psi) = A_2(\psi)A_1(\theta)A_3(\phi) =$$

$$\begin{bmatrix} 
\cos\phi\cos\psi - \sin\theta\sin\phi\sin\psi & \cos\psi\sin\phi + \sin\theta\sin\psi\cos\phi & -\cos\theta\sin\psi \\
-\cos\theta\sin\phi & \cos\theta\cos\phi & \sin\theta \\
\sin\psi\cos\phi + \sin\theta\cos\psi\sin\phi & \sin\psi\sin\phi - \sin\theta\cos\psi\cos\phi & \cos\theta\cos\psi 
\end{bmatrix}$$

Direction Cosine Matrix for a 3-1-2 Euler Sequence

Our Spacecraft is going to have rotations of $\phi = 20$ degrees, $\theta = 35$ degrees, and $\psi = 45$ degrees. Performing hand calculations by plugging in the values into the matrix above we get:

$$\begin{bmatrix} 
.52574656 & .62296509 & -.57922796 \\
-.280166499 & .769751131 & .5735764 \\
.8031794 & -.13927557 & .57922796 
\end{bmatrix}$$

Results of Hand Calculations

We will use these values to model our Spacecraft's attitude in the next section of this tutorial. For now, let's talk about quaternions.
Quaternions

The DCM can also be parameterized in terms of Euler symmetric parameters. These four parameters are not independent, but satisfy the constraint equation:

\[ q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \]

\[ \text{Constraint Equation} \]

Euler symmetric parameters make up the elements of the quaternion providing a redundant, non-singular attitude description that is well suited to describe arbitrary, large rotations. Quaternions are made up of four parameters: the three components of a vector and a scalar. The vector component of the quaternion describes an axis between a reference frame and the body frame of the vehicle, and the scalar component gives the rotation about that axis. The components of a quaternion are:

\[ q \equiv \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \equiv \left[ \begin{array}{c} q \\ \end{array} \right] \]

\[ \text{Quaternion Representation} \]

Also represented by:

\[ q \equiv q_4 + iq_1 + jq_2 + kq_3 \]

\[ \text{Quaternion Representation} \]

Where i, j, and k are the hyperimaginary numbers satisfying the conditions:

\[ i^2 = j^2 = k^2 = -1 \]

\[ ij = -ji = k \]

\[ jk = -kj = i \]

\[ ki = -ik = j \]

\[ \text{Hyperimaginary Numbers} \]

The quantity, \( q_4 \), is the real or scalar part of the quaternion, and \( iq_1 + jq_2 + kq_3 \) is the imaginary or vector part. Now let's calculate the Quaternion from the DCM we previously calculated.

The components of a quaternion with respect to a DCM can be calculated by the following equations:

**NOTE:** There is a sign ambiguity in the calculations, because changing the signs of all the components of the quaternion simultaneously does not affect the DCM. There are 4 possible ways of calculating the components of the quaternion. All methods are mathematically equivalent, but error can be reduced by avoiding calculations in which the component of the quaternion appearing in the denominator is close to zero. For our analysis, we will just use the following equations.
Plugging in the elements of the DCM we previously calculated, the components of the 3-1-2 sequence should be:

\[
q_4 = \pm \frac{1}{2} \left( 1 + A_{11} + A_{22} + A_{33} \right)^{\frac{1}{2}}
\]

\[
q_1 = \frac{1}{4q_4} (A_{23} - A_{32})
\]

\[
q_2 = \frac{1}{4q_4} (A_{31} - A_{13})
\]

\[
q_3 = \frac{1}{4q_4} (A_{12} - A_{21})
\]

**Quaternion calculations from DCM**

Of course we can also use FreeFlyer script to calculate these values, and compare them to the values stored in the Spacecraft object. The following script breakdown demonstrates the manual calculations for a 3-1-2 rotation sequence:

- Create the rotation angles as Variables, and assign them to the Spacecraft object’s AttitudeAngles property

```
// Calculations for a 3-1-2 rotation
Variable phi;
Variable theta;
Variable psi;

// Assign Euler Angles
phi = Spacecraft1.AttitudeAngles[0];
theta = Spacecraft1.AttitudeAngles[1];
psi = Spacecraft1.AttitudeAngles[2];
```

- Create a matrix and enter each element from the DCM for a 3-1-2 sequence in variable form that we found earlier

```
Matrix dcm312;

// Calculate the DCM by hand
dcm312 = [
    [cos(rad(psi))*cos(rad(phi)) - sin(rad(psi))*sin(rad(theta))*sin(rad(phi)),
    [cos(rad(psi))*sin(rad(phi)) + sin(rad(psi))*sin(rad(theta))*cos(rad(phi))],
    [sin(rad(psi))*cos(rad(theta))],
    [sin(rad(psi))*sin(rad(theta))]
]
```
cos(rad(psi)) * sin(rad(phi)) + sin(rad(theta)) * sin(rad(phi)) * sin(rad(theta)),
-cos(rad(theta)) * sin(rad(psi)); -cos(rad(theta)) * sin(rad(phi)),
-cos(rad(theta)) * cos(rad(phi)), 
-sin(rad(psi)) * cos(rad(phi)) + cos(rad(psi)) * sin(rad(theta)) * sin(rad(phi)),
sin(rad(psi)) * sin(rad(phi)) - sin(rad(theta)) * cos(rad(psi)) * cos(rad(phi)),
sin(rad(psi)) * cos(rad(theta)))];

- Calculate the quaternion by creating an array and assigning each element to the four components of the quaternion and the necessary equations to solve for them
- Create a Report command so that we can compare our calculations and equations to the values that FreeFlyer has calculated

```plaintext
Array q[4];

Calculate the quaternion by hand
q[3] = .5*sqrt((1+ dcm312[0,0] + dcm312[1,1] + dcm312[2,2]));
q[0] = 1/(4*q[3])*(dcm312[1,2] - dcm312[2,1]);
q[1] = 1/(4*q[3])*(dcm312[2,0] - dcm312[0,2]);
q[2] = 1/(4*q[3])*(dcm312[0,1] - dcm312[1,0]);

Report dcm312, Spacecraft1.AttitudeMatrix, q, Spacecraft1.Quaternion;
```

Now we'll compare these hand-calculated values to FreeFlyer's built in conversions.

**Modeling the Direction Cosine Matrix and Quaternions**

We discussed how to find the DCM and Quaternion by hand - now let's take a closer look at their physical meaning by modeling them in FreeFlyer. Keep your hand calculations close as we will be using them in the following Mission Plan!

Create a new Mission Plan and save it as "ModelQuaternionDCM.MissionPlan"

**Adding in Spacecraft**

- Create a Spacecraft with the following Keplerian elements:
  - A: 15000 km
  - E: 0.4
  - I: 15 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg
- Rename the Spacecraft to 'scEuler'.
- Click on "Attitude" in the left hand side of the Spacecraft Window under "Primary Options", and give scEuler the following properties:
  - Reference Frame: LVLH - Earth Pointing
  - Attitude System: Euler Angles
  - Rotation Sequence: 3-1-2
- Euler Angle 1: 20
- Euler Angle 2: 35
- Euler Angle 3: 45

- Click on "Propagator" on the left hand side and change the step size to 0.5 seconds.
- Click "Ok" to save your changes and close the Spacecraft Editor.

- Right-click on scEuler and clone it twice.

- Double click on "scEuler_Copy1" and rename it "scDCM"
- Click on "Attitude" on the left hand side of the Spacecraft Editor
- Under the "Attitude System" dropdown, select "Attitude Matrix"
- Fill in the Matrix with the values we calculated by hand
- Click "Ok" to save your changes and close the Spacecraft Editor.

```
<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>.52374656</td>
<td>.62296509</td>
<td>-.57922796</td>
</tr>
<tr>
<td>-.280166499</td>
<td>.769751131</td>
<td>.5735764</td>
</tr>
<tr>
<td>.8031794</td>
<td>-.13927557</td>
<td>.57922796</td>
</tr>
</tbody>
</table>
```

- Double click on "scEuler_Copy2" and rename it "scQuaternion"
- Click on "Attitude" on the left hand side of the Spacecraft Editor
- Under the "Attitude System" dropdown, select "Quaternion"
- Fill in the components of the quaternion with the values we calculated by hand
- Click "Ok" to save your changes and close the Spacecraft Editor.
• Right click on "scQuaternion" and clone it
• Rename the clone to "scRotating"
• Click on "Attitude" on the left hand side of the Spacecraft Editor
• Give scRotating the following rotation rates:
  o wu: 1.006 deg/s
  o wv: 0.008 deg/s
  o ww: 0.06 deg/s
• Click “Ok” to save your changes and close the Spacecraft Editor

Adding a ViewWindow

• Add a ViewWindow through the object browser and rename it "threeDView"
• Under "Available Objects" check all four spacecraft
• In the “Object Options” for all four spacecraft:
  o Check Show Name
  o Check Show History
  o Check Show Axes
• Navigate to Viewpoints
• Make the following Changes to the Default ViewPoint
  o In the "Source" dropdown scroll down and click scEuler, then click "Copy to Target"
  o Right Ascension: 315 deg
  o Declination: -55 deg
  o Radius: 930 km
• Click “Ok” to save your changes and close the ViewWindow Editor
Adding a PlotWindow

- Add a PlotWindow through the object browser and rename it "plotQuaternion"
- Click on the X-Axis dropdown menu
- Under "Choose an object", click the dropdown and select scRotating
- Under the "Choose a property or method", click the dropdown and select "ElapsedTime"
- Click on the Y-Axis dropdown menu
- Under "Choose and object", click the dropdown and select scRotating
- Under "Choose a property or method", select "Quaternion"
- Leave the "0" under "Choose an index value"
- Click "More" 3 times to add the other components of the quaternion
- Click on the second Y-Axis dropdown
- Under "Choose and object", click the dropdown and select scRotating
- Under "Choose a property or method", select "Quaternion"
- Enter "1" in the "Choose an index value" box
- Repeat this for the other components of the quaternion
- Your PlotWindow Editor should look like this:

![PlotWindow Editor Example](image)

- Click "Ok" to save your changes and close the PlotWindow Editor
Building the Mission Sequence

Before building the Mission Sequence, make sure your Object Browser looks like this:

- From the Script Elements browser on the right hand side of the control screen, drag and drop a "While...End" loop into the Mission Sequence.
- Change the while loop stopping condition to "scEuler.ElapsedTime < TIMESPAN(30 minutes)"
- From the Script Elements browser, drag and drop a FreeForm script editor inside the while loop.
- Double click the FreeForm script editor and change its name to "Step and Update".
- We will also be adding a Watch command to create a WatchWindow that will report the value of scRotating's quaternion components at every time step.

The following script is relatively simple, we just need to step our spacecraft, synchronize their epochs, add the Watch command, and update the other outputs:

```plaintext
// Step Spacecraft and Synch Epochs
Step scEuler;
Step scQuaternion to (scQuaternion.Epoch == scEuler.Epoch);
Step scDCM to (scDCM.Epoch == scEuler.Epoch);
Step scRotating to (scRotating.Epoch == scEuler.Epoch);

// Watch command
Watch scRotating.Quaternion[0], scRotating.Quaternion[1],
    scRotating.Quaternion[2], scRotating.Quaternion[3];

// Update Outputs
```
Also, let's verify that our hand calculations from earlier were correct:

- From the Script Elements browser, drag and drop a FreeForm script editor before the while loop
- Double click the FreeForm script editor and change its name to "Check Calculations"

The following script will report the DCM and Quaternion of scEuler, which means FreeFlyer is calculating the DCM and Quaternion from the Euler Angles that we input into scEuler:

```
Report scEuler.AttitudeMatrix, scEuler.Quaternion;
```

Now the Mission Plan is ready to run! It should look something like this:

```
// Mission Sequence

<table>
<thead>
<tr>
<th>#</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FreeForm: Check Calculations</td>
</tr>
<tr>
<td>2</td>
<td>While (scEuler.ElapsedTime &lt; TIMESSPAN(30 minutes));</td>
</tr>
<tr>
<td>3</td>
<td>FreeForm: Step and Update</td>
</tr>
<tr>
<td>4</td>
<td>End;</td>
</tr>
</tbody>
</table>

Mission Sequence Example
```

Save your progress and then run to execute the Mission Plan. Then, try to answer these questions:

- **In the threeDView output - why does it appear that there is a single rotating spacecraft and a single stationary spacecraft?**

- **Did your hand calculated values match what FreeFlyer calculated?**

- **Inspect the generated WatchWindow and the plotQuaternion plot - do you notice something significant about the components of the quaternion at all times?**

**See Also**

- [Spacecraft Attitude](#)
- [Next Topic: Attitude Reference Frames](#)
3.2 - Attitude Reference Frames

At the heart of attitude representation is the attitude reference frame. The Attitude Reference frame defines the coordinate system that the spacecraft's attitude is referenced to. Converting between the spacecraft body frame and the reference frame is how we manipulated the spacecraft attitude in the previous section. Different frames of reference are used for different applications.

In this section we will discuss:

1. Available Reference Frames
2. LVLH vs Other Reference Frames

Available Reference Frames

We will be discussing these attitude reference frames that come built in with FreeFlyer:

1. LVLH - Earth Pointing
2. VNB
3. MJ2000 Earth Equator
4. Geodetic

For the following descriptions of the attitude reference frames, let $\mathbf{v}$ be the velocity vector, $\mathbf{r}$ be the position vector, and $\mathbf{r} \times \mathbf{v} = \mathbf{h}$ be the angular momentum vector.

NOTE: For each of the following reference frames, if a reference frame is noted as a rotating reference frame, it will rotate once per orbit.
LVLH - Earth Pointing

LVLH (Local Vertical, Local Horizontal) - Earth Pointing is the default reference frame in FreeFlyer. It is a rotating reference frame that is described by:

- Z-Axis: Oriented in the direction of -r (points to center of Earth) - Local Vertical
- Y-Axis: Negative to the orbit normal, or in the direction of -h
- X-Axis: Perpendicular to Y and Z, forming a right-handed coordinate system - Local Horizontal
- Origin: Center of the Spacecraft
VNB

The VNB (Velocity, Normal, Binormal) is a rotating reference frame that is described by:

- **X-Axis:** Vector oriented in the direction of \( \mathbf{v} \)
- **Y-axis:** Vector oriented in the direction of the orbit normal, or in the direction of \( \mathbf{h} \)
- **Z-Axis:** Oriented in the direction of \( \mathbf{r} \), forming a right-handed coordinate system
- **Origin:** Center of the Spacecraft
MJ2000 Earth Equator

The MJ2000 (Mean of Julian Date 2000) reference frame is an inertial reference frame (non-rotating) that is described by:

- Z-Axis: Vector normal to the mean equatorial plane at Julian year 2000.0, pointing towards the Northern Hemisphere
- X-Axis: Vector pointing from the center of Earth to the mean vernal equinox at Julian year 2000.0
- Y-Axis: Vector perpendicular to the x- and z- axes, forming a right-handed coordinate system
- Origin: Center of the Earth
Geodetic

The Geodetic reference frame is a rotating reference frame that is described by:

- Z-Axis: Vector normal to the Earth’s surface in the direction of $-r$
- Y-axis: Negative to the orbit normal, or in the direction of $-h$
- X-Axis: Cross product of $-h \times -r$, oriented in the direction of $v$
- Origin: Center of the Spacecraft

LVLH vs Other Reference Frames

The LVLH - Earth Pointing Reference Frame is the default attitude reference frame in FreeFlyer. We are going to take a look at the differences between LVLH and a few of the other attitude reference frames, and show those differences in FreeFlyer.

LVLH vs Geodetic

The Geodetic frame is nearly the same as the LVLH frame. The difference is that the Geodetic frame's Z-axis is normal to the Earth's surface, not pointing to the center of the Earth like in the LVLH frame. The difference in these two frames is due to the shape of the Earth. The Earth is not a perfect sphere, it's an oblate spheroid, which you will get more into later in this guide. Therefore, the only time these frames line up are directly over the equator,
and the north and south poles. By creating a spacecraft in FreeFlyer, and setting it's attitude reference frame to Geodetic, we can plot the spacecraft's attitude angles with respect to the LVLH frame. This shows that the frames are very similar, but due to the shape of the Earth, there are small differences in the attitude with respect to the LVLH frame.

![Geodetic Attitude Angles wrt LVLH Frame](image)

**LVLH vs MJ2000**

Here we are going to take a short look at the difference between the LVLH attitude reference frame and the MJ2000 attitude reference frame.

**Problem:**

An important part of spacecraft operations is remote sensing, or the study of some quantity from a distance using a sensor on a spacecraft. Given a spacecraft model, we know our sensor is aligned along the Z-axis of the spacecraft body. Compare the results of attaching the same sensor to the same spacecraft, but in different reference frames.

- Create a new Mission Plan and save it as "LVLHvsMJ2000.MissionPlan"

**Adding in Spacecraft**

- Add in two spacecraft objects through the object browser
- Double-click Spacecraft1 to bring up the editor window
- Rename it "scLVLH"
- Change the step size to 3 seconds
- On the left hand side navigate to Subsystems → Sensors
- Click "Create" to add a sensor to scLVLH. This creates a sensor pointed along the spacecraft Z-axis
- Click "Ok" to close the object editor and save your changes
- Click "Ok" again to save all changes and close the Spacecraft Editor
• Double-click Spacecraft2 to bring up the editor window
• Rename it to scMJ200
• Change the spacecraft's tail color to blue
• On the left hand side click "Attitude" → "Reference Frame" dropdown → Mean of J2000 Earth Equator
• On the left hand side navigate to Subsystems → Sensors
• Click "Create" to add a sensor to scMJ2000
• Click on Sensor1 in the Sensors page
• Click "Edit Sensor" under "Sensor1 - Properties"
• On the left hand side click visualization, and change the color to blue to match the spacecraft
• Click "Ok" to close the object editor and save your changes

Adding a ViewWindow

• Add in a ViewWindow through the object browser and open the ViewWindow Editor
• Under "Available Objects" check both spacecraft
• In the "Object Options" for both spacecraft:
  o Check Show Name
• Navigate over to Viewpoints on the left hand side
• Make the following changes to the default viewpoint
  o Under the "Source" dropdown change it to scLVLH
  o Click "Copy to target"
  o Right Ascension: 85 deg
  o Declination: -140 deg
  o Radius: 3000 km

Building the Mission Sequence

Before building the Mission Sequence, make sure your Object Browser looks like this:
Drag and drop a "While...End" loop into the Mission Sequence from the Script Elements
Drag and drop a "FreeForm" segment inside the while loop and rename it "Step and Update"
Open the FreeForm and type the following simple script.

```
Step scLVLH;
Step scMJ2000 to (scMJ2000.Epoch == scLVLH.Epoch);
Update ViewWindow1;
```

Now our Mission Plan is ready to run! Your Mission Sequence should look something like this:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>//</td>
<td>#</td>
<td>FreeForm: Mission Plan Description and Setup</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>While (scLVLH.ElapsedTime &lt; TIMESSPAN(1 days));</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>FreeForm: Step and Update</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>End;</td>
</tr>
</tbody>
</table>

Run the mission plan and try to answer these questions:

Which attitude reference frame should be used if it's important to keep a sensor mounted to the Spacecraft body pointed directly toward the center of the Earth?

Which attitude reference frame should be used when pointing a camera mounted on the Spacecraft body at a distant star or galaxy?

**LVLH vs VNB**

In an eccentric orbit, the difference between these two frames will be a pitch rotation that is equal to the orbital flight path angle. For a circular orbit, the LVLH and VNB frames will be the same. Here we are going to take a look at this in FreeFlyer.

The flight path angle is defined as the angle measure from the local horizontal to the velocity vector. When the spacecraft is in orbit from perigee to apogee (outbound) the flight path angle is positive, and when the spacecraft is in orbit from apogee to perigee (inbound) the flight path angle is negative. At perigee and apogee the flight path angle is 0°, and it has a max value of 90°. The equation for solving the flight path angle is given by:

\[ \phi = \cos^{-1}
\left(\frac{1 + ecov}{\sqrt{1 + e^2 + 2ecov}}\right) \]

\[
\begin{align*}
\phi &= \text{Flight Path Angle} \\
e &= \text{Eccentricity} \\
v &= \text{True Anomaly} \\
\text{Flight Path Angle Equation}
\end{align*}
\]
Problem:
Your company is launching a satellite in a highly elliptical orbit. You're tasked with the investigation of two different attitude reference frames: LVLH and VNB - so that the company can decide the most cost affective way of maintaining their desired attitude.

Create a new Mission Plan and save it as "LVLHvsVNB.MissionPlan"

NOTE: This Mission Plan uses vectors for visualization and some calculations, but extensive knowledge of using vectors in FreeFlyer is not needed.

Adding in Spacecraft

- Create a Spacecraft with the following Keplerian elements:
  - A: 18000 km
  - E: 0.6
  - I: 15 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg
- In the top right corner of the object editor window, change the name to "scLVLH"
- Click "Propagator" on the left hand side and change the time step to 30 seconds
- Click "Ok" to close the editor and save your changes
- Right click scLVLH and clone it
- Rename the clone "scVNB"
- On the left hand side of the Spacecraft Editor click "Attitude"
- In the "Reference Frame" dropdown, select "VNB"
- Click "Ok" to close the editor and save your changes

Adding in ViewWindow Objects

- Add two ViewWindow Objects from the Object Browser
- Double click on ViewWindow1 to edit it, and rename it "closeUpView"
- Under "Available Objects" check both spacecraft
- In the "Object Options" for both spacecraft:
  - Check Show Name
- Navigate to "Viewpoints" on the left hand side
- Make the following changes to the Default Viewpoint
  - Source: scLVLH
  - Click "Copy to Target"
  - Right Ascension: 45 deg
  - Declination: -130 deg
  - Radius: 1750 km
- Click "Ok" to close out of the ViewWindow editor and save your changes

- Double click on ViewWindow2 to edit it, and rename it "orbitView"
- Under "Available Objects" check both spacecraft
- Under "Object Options", check the following for both spacecraft:
  - Show Name
  - Show History
  - History Mode: Unlimited
- Navigate to "Viewpoints" on the left hand side
- Make the following changes to the Default Viewpoint
  - Reference Frame: Inertial
  - Source: Earth
  - Click "Copy to Target/Tail Ref."
  - Right Ascension: 270 deg
  - Declination: 90 deg
  - Radius: 75000 km
- Click "Ok" to close out of the ViewWindow editor and save your changes

**Adding a PlotWindow**

- Add a PlotWindow Object from the object browser
- Double click on PlotWindow1 to edit it, and rename it "deltaAngles"
- Under the "X-Axis" dropdown, select the "Choose an Object" dropdown and then select scVNB
- Make sure the "Choose a property or method" drop down is still "ElapsedTime"
- Click "More" on the right hand side to add one more series
- Click on the first series drop down
- Under the "Choose an object" drop down select scVNB
- Under the "Choose a property or method" drop down, scroll down and find "FPAngle". This is the orbital flight path angle
- Click on the next series drop down
- Under the "Choose an object" drop down select scVNB
- Under the "Choose a property or method" drop down, scroll down and find "AttitudeLVLH"
- In the "Enter an index value" box, type in the number 2. This is the scVNB's pitch rotation with respect to the LVLH frame
- Click "Ok" to close out of the PlotWindow editor and save your changes
Building the Mission Sequence

Before building the Mission Sequence, make sure your Object Browser looks like this:

- Drag and drop a FreeForm script editor from the script elements into the mission sequence
- Rename this FreeForm "Vectors"
- Copy and paste the following script in this FreeForm, or follow this script breakdown to boost your knowledge of using Vectors in FreeFlyer

NOTE: These vectors will represent the Axes of the two spacecraft, and this script adjusts the color of the spacecraft and vectors to match.

- Create the 3 axes of each spacecraft as Vectors, and use the BuildVector method to set them as the axes of the two spacecraft
- Use the DrawMethod property to draw the vectors as arrows instead of lines when visualizing them in Outputs
• scLVLH will have yellow axes, so we do not need to change the color because the default vector color is yellow
• scVNB will have blue axes - use the ColorTools object to change the color to "CornflowerBlue"

```csharp
// Create vectors for both spacecraft
Vector XaxisLVLH;
Vector YaxisLVLH;
Vector ZaxisLVLH;
Vector XaxisVNB;
Vector YaxisVNB;
Vector ZaxisVNB;

// Build scLVLH vectors and draw them as arrows
XaxisLVLH.BuildVector(4, scLVLH, 1); // X-body-axis vector
YaxisLVLH.BuildVector(4, scLVLH, 2); // Y-body-axis vector
ZaxisLVLH.BuildVector(4, scLVLH, 3); // Z-body-axis vector
XaxisLVLH.DrawMethod = 1;
YaxisLVLH.DrawMethod = 1;
ZaxisLVLH.DrawMethod = 1;

// Build scVNB vectors and draw them as arrows
XaxisVNB.BuildVector(4, scVNB, 1); // X-body-axis vector
YaxisVNB.BuildVector(4, scVNB, 2); // Y-body-axis vector
ZaxisVNB.BuildVector(4, scVNB, 3); // Z-body-axis vector
XaxisVNB.Color=ColorTools.CornflowerBlue;
YaxisVNB.Color=ColorTools.CornflowerBlue;
ZaxisVNB.Color=ColorTools.CornflowerBlue;
XaxisVNB.DrawMethod = 1;
YaxisVNB.DrawMethod = 1;
ZaxisVNB.DrawMethod = 1;
```

• Add both vectors to the ViewWindow's we created and show their names - this must be done in script since we did not define the Vectors in the GUI
• Match the color of the Spacecraft objects to their respective vectors

```csharp
// Add scLVLH vectors to the closeUpView and turn on vector names
closeUpView.AddObject(XaxisLVLH);
closeUpView.SetShowName("XaxisLVLH", 1);
closeUpView.AddObject(YaxisLVLH);
closeUpView.SetShowName("YaxisLVLH", 1);
closeUpView.AddObject(ZaxisLVLH);
closeUpView.SetShowName("ZaxisLVLH", 1);

// Add scVNB vectors the the closeUpView and turn on vector names
closeUpView.AddObject(XaxisVNB);
closeUpView.SetShowName("XaxisVNB", 1);
closeUpView.AddObject(YaxisVNB);
closeUpView.SetShowName("YaxisVNB", 1);
closeUpView.AddObject(ZaxisVNB);
closeUpView.SetShowName("ZaxisVNB", 1);
```
Drag and drop a "While...End" after "Vectors"
Change the while loop stopping condition to "scLVLH.ElapsedTime < TIMESSPAN(14 hours)"
Drag and drop a FreeForm inside the While loop, and rename it "Step, Report, Update"

The following script is very simple: we will step our spacecraft, sync their epochs, update the vector epochs, and update the outputs. Also, we need to report certain values to verify that the difference in these two frames is a change in pitch equal to the flight path angle at any given time. We can report scVNB's third Euler Angle with respect to the LVLH frame, as well as the flight path angle, and the angle between the two spacecraft's X-axes through the vectors we created earlier:

```csharp
// Step sc and sync epochs
Step scLVLH;
Step scVNB to (scVNB.Epoch == scLVLH.Epoch);

// Update Vector epochs
XaxisLVLH.Epoch = scLVLH.Epoch;
YaxisLVLH.Epoch = scLVLH.Epoch;
ZaxisLVLH.Epoch = scLVLH.Epoch;
XaxisVNB.Epoch = scLVLH.Epoch;  // The epochs can be set to scLVLH because
YaxisVNB.Epoch = scLVLH.Epoch;  // scVNB's epoch is synced with scLVLH
ZaxisVNB.Epoch = scLVLH.Epoch;

// Report and update outputs
Report scVNB.AttitudeLVLH[2], scVNB.FPAngle, XaxisLVLH.VertexAngle(XaxisVNB);
Update deltaAngles;
Update closeUpView;
Update orbitView;
```

Now your mission plan is ready to run! The mission sequence should look like this:

```
Mission Sequence Example

1  FreeForm: Vectors
2  While (scLVLH.ElapsedTime < TIMESSPAN(14 hours));
    FreeForm: Step, Report, Update
3  End;
```

Save your progress and run to execute the Mission Plan. Then try to answer the following questions:

When are the X-axes of the two spacecraft oriented in the exact same direction?
What is the significance of the values on the plot?

Run the mission plan again but set the two spacecraft in circular orbits ($e=0$). What do you notice about these two frames? Why do you think that is happening?

We can also set a spacecraft’s attitude reference frame to a custom reference frame using the CoordinateSystem object in FreeFlyer, and we will get more into this in the next tutorial.

See Also

- [Spacecraft Attitude](#)
- [Next Topic: Mission Constraints on Spacecraft Attitude](#)
3.3 - Mission Constraints on Spacecraft Attitude

The most important part of mission design is the mission itself. There are hundreds of spacecraft and satellite missions, and each has varying requirements. Sometimes we want to point our spacecraft towards the Sun, or point to the ground for communications, or point to the Earth or to space for science.

In FreeFlyer, it is possible to design your own custom attitude reference frame with the use of Vector and CoordinateSystem objects. We can choose a primary vector that we want to point the spacecraft body toward, and then a secondary vector that will define the rotation about the primary vector. This is also referred to as "align and constrain".

In this section we will discuss:

1. Custom Mission Attitude Frame

Custom Mission Attitude Frame

In the previous sections we have discussed different ways to represent a spacecraft's attitude, and how those representations can change with the variety of reference frames. Now, let's put it all together with a mission design problem.

Problem:
A GEO satellite is equipped with 40% efficient solar panels that have a total area of 6.5m² aligned with the Z-axis. Its mission requires that it constantly looks to the Canberra ground station in Australia with a sensor along the X-axis, and that it must have at least 1000W of power supplied at all times. We must create a custom attitude reference frame that meets these requirements.

Create a new Mission Plan and save it as "CustomMissionFrame.MissionPlan"

Adding in a Spacecraft with a Sensor

- Create a Spacecraft with the following Keplerian elements:
  - A: 42164 km
  - E: 0
  - I: 30 deg
  - RAAN: 210 deg
  - W: 15 deg
  - TA: 0 deg
- Change the Spacecraft tail color to green
- Change the step size to 30 seconds
- On the left hand side of the editor window under "Subsystems" click "Sensors"
- Click "Create" to create a sensor for the Spacecraft, then click "Edit"
- On the "Sensor Properties" page, change the "Cone Half Angle" to 2.5 degrees
Under the "Orientation wrt Spacecraft BCS" click the "System" dropdown, and select "Unit Vector"

In the "NX" box enter 1, and in the "NZ" box enter 0

On the left hand side of the Object Editor click "Visualization"

On that page, change the color to green to match the Spacecraft

In the "Projection Height" box enter 42164 km

Click "Ok" to save your changes and click "Ok" again to save your changes and close the Spacecraft Editor

Add in a GroundStation through the object browser and double-click the GroundStation to edit it

On the "Properties" Page where it says "GroundStation Setup" click "Station Geodetics File"

In the "Station Data" box click the "Station Acronym" dropdown and select "DS45 - Canberra Australia"

Navigate to "Visualization" on the left hand side and change the color to blue

Click "Ok" to save your changes and close the GroundStation Editor
Adding in Vector, Variable, and CoordinateSystem Objects

- In the Object Browser click "Add"
- Under the "Variable" section that has multiple options:
  - Add in 6 Variable, 3 Vector, and 1 CoordinateSystem Objects and name them the following:
    - Vectors:
      - scToCanberra
      - scToSun
      - solarPanelVec
    - Variables:
      - meanSunDist
      - panelAngle
      - panelArea
      - panelEfficiency
      - panelPower
      - solarFlux
    - Coordinate System:
      - missionFrame

NOTE: After adding an object you can hit CTRL+SHIFT+Y to continue adding the most recently added object type, also you can use F2 as a shortcut to rename objects
Adding in Multiple ViewWindow Objects

- Add two ViewWindow Objects from the Object Browser
- Double click on ViewWindow1 to edit it, and rename it "earthView"
- Check Spacecraft1
- Under "Object Options", check the following:
  - Show Name
  - Show History
  - History Mode: Unlimited
- Check Canberra
- Under "Object Options", check the following
  - Show Name
- Check Vectors scToCanberra and scToSun
- Navigate to "Viewpoints" on the left hand side
- Make the following changes to the Default Viewpoint
  - Reference Frame: Inertial
  - Source: Earth
Click "Copy to Target/Tail Ref."
- Right Ascension: 130 deg
- Declination: 30 deg
- Radius: 95000 km
- Click "Ok" to close out of the ViewWindow editor and save your changes

Double click on ViewWindow to edit it, and rename it "scView"
- Check Spacecraft1
- Under "Object Options", check the following:
  - Show Name
- Check Canberra
- Under "Object Options", check the following:
  - Show Name
- Check Vectors scToCanberra and scToSun
- Navigate to "Viewpoints" on the left hand side
- Make the following changes to the Default Viewpoint:
  - Reference Frame: Inertial
  - Source: Spacecraft1
  - Click "Copy to Target/Tail Ref."
  - Right Ascension: 200 deg
  - Declination: 0 deg
  - Radius: 800 km
- Click "Ok" to close out of the ViewWindow editor and save your changes

**Adding in a PlotWindow**

- Add a PlotWindow Object from the object browser
- Double click on PlotWindow1 to edit it, and rename it "powerPlot"
- Click the "Y-Axis" dropdown, and in the "Choose which type of item to use" dropdown select "Object"
- Click the "Choose an object:" dropdown and choose panelPower
- Click "Ok" to close the PlotWindow editor and save your changes

Prior to building the Mission Sequence we must set our Spacecraft's attitude reference frame to the custom one we just created

- Double click Spacecraft1 to edit it
- On the left hand side, go into the "Attitude" options
- In the "Reference Frame" dropdown select "Custom Coordinate System"
- A new dropdown will appear called "Coordinate System", click it and select our custom CoordinateSystem "missionFrame"
- Click "Ok" to close the Spacecraft editor and save your changes
Building the Mission Sequence

- Drag and drop a FreeForm script editor from the script elements into the mission sequence
- Rename this FreeForm "Initialize Variables"

First, we are going to build our vectors. The BuildVector method in FreeFlyer allows us to define a vector in many ways. First we must select the vector type we want; in this case we will use Object-to-Object and Body Axis vectors. For an Object-to-Object vector we must define the origin object, and then the reference object that the vector will point to. For a Body Axis vector, we will choose our Spacecraft, and then which body axis we want, in this case the Z-Axis for the solar panel normal vector. Then, we will use those two vectors to build our custom CoordinateSystem. We will define the CoordinateSystem X-axis as the vector that points from our Spacecraft to the GroundStation, then we will define the CoordinateSystem Z-axis to be constrained toward the vector that points from our Spacecraft to the Sun. Finally, we will input values for a few of our variables.

Type the following script into this FreeForm:

NOTE: Take a look at a few of the different possible overloads for Vector and CoordinateSystem objects to see the many options FreeFlyer offers.

```csharp
// Vectors
scToSun.BuildVector(9, Spacecraft1, Sun); // Object-to-Object vector
scToCanberra.BuildVector(9, Spacecraft1, Canberra); // Object-to-Object vector
solarPanelVec.BuildVector(4, Spacecraft1, 3); // Z-body-axis vector
```
Here we are going to calculate the power from our solar panels. The equation for that calculation is:

\[ P = \eta \cdot S_o \cdot A \cdot \cos \alpha \cdot \frac{R}{R_o} \]

\[ \eta = \text{Solar Panel Efficiency} \]
\[ S_o = \text{Solar Constant} \]
\[ A = \text{Solar Panel Area} \]
\[ \alpha = \text{Solar Panel Angle} \]
\[ R_o = \text{Spacecraft Sun Distance} \]
\[ \frac{R}{R_o} = \text{Spacecraft Mean Sun Distance} \]

Solar Power Equation

It is important to note that the panel angle must be between 0 and 90 degrees for the panels to receive any sunlight. We will address this in the following FreeForm with an "If" statement. Write the following script in this FreeForm:

```c
// Solar Power Calculation

// Define Panel Angle so it updates with time
panelAngle = solarPanelVec.VertexAngle(scToSun); // deg

// The Panel Angle must be between 0 and 90 degrees for the solar panels to be in contact with sunlight
If (0 < solarPanelVec.VertexAngle(scToSun) and solarPanelVec.VertexAngle(scToSun) < 90) then;
    panelPower = panelEfficiency*solarFlux*panelArea*cos(rad(panelAngle))*Spacecraft1.SCSunDistance/meanSunDist; // W
Else;
```
Drag and drop another FreeForm script editor after our calculations
Name it "Step and Update"

Here we are going to Step our Spacecraft, sync Epochs, and Update our outputs. Type the following simple script into the FreeForm:

```
// Step and Update

// Sync Vector Epochs with Spacecraft Epoch
scToCanberra.Epoch = Spacecraft1.Epoch;
scToSun.Epoch = Spacecraft1.Epoch;
solarPanelVec.Epoch = Spacecraft1.Epoch;

// Update Outputs
Update earthView;
Update scView;
Update powerPlot;

// Step Spacecraft
Step Spacecraft1;
```

Now your mission plan is ready to run! The mission sequence should look like this:

<table>
<thead>
<tr>
<th>Mission Sequence</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FreeForm: Initialize Variables</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>While (Spacecraft1.ElapsedTime &lt; TIMESSPAN(1 days));</td>
</tr>
<tr>
<td>4</td>
<td>FreeForm: Solar Power Calculation</td>
</tr>
<tr>
<td>5</td>
<td>FreeForm: Step and Update</td>
</tr>
</tbody>
</table>

Save your progress and run to execute the Mission Plan. Then try and answer the following questions:

Take a look at the graph. What are the minimum and maximum power values? Do they meet the mission requirements?

Notice the scView ViewWindow, what is the significance of the scToSun Vector?

Run the mission plan again but set the Spacecraft attitude to LVLH - Earth Pointing, do we still meet mission requirements?
See Also

- Spacecraft Attitude
- Previous Topic: Attitude Reference Frames
In this chapter, we will discuss various maneuvers spacecraft perform to change their orbits. We will discuss the math behind these maneuvers and then apply these concepts using FreeFlyer.

This chapter will cover the following topics:

1. Hohmann Transfer
2. Bi-Elliptic Transfer
3. Phasing Maneuver
4. Plane Change Maneuver
4.1 - Hohmann Transfer

Hohmann transfers are typically the most efficient transfer a spacecraft can make to change the size of an orbit. For simple Hohmann calculations, you must assume circular starting and target orbits - and they must be coplanar!

In the above diagram, you see a good depiction of a Hohmann transfer. In part 1 (the green orbit), the satellite is in a "parking orbit" which is a Low Earth Orbit that is achieved shortly after launch. In part 2 (the yellow orbit), a maneuver is performed, increasing the velocity of the satellite until its orbit is an ellipse with an apogee at the target orbit's semi-major axis. This part is called the transfer trajectory. Once the spacecraft reaches the apoapsis of that trajectory, it performs an orbital insertion burn. This increases the velocity, matching the orbit to its target circular orbit.

In this section, we will discuss:

1. Calculating Hohmann Transfers
2. Hohmann Transfer - Earth Centered

Calculating Hohmann Transfers

We'll discuss how to calculate the amount of $\Delta v$ required to perform a Hohmann transfer.

**Problem:**
Our spacecraft has a SMA of 7,000 km and is in a circular orbit. We wish to put it at a 20,000 km SMA circular orbit. Calculate the total amount of $\Delta v$ required to transfer to the new orbit using a Hohmann transfer.
Every calculation will be based around the Vis-Viva Equation:

\[
\nu = \sqrt{\frac{2}{r} - \frac{1}{a}}
\]

Vis-Viva Equation

\(v\) = Velocity  
\(\mu\) = Standard Gravitational Parameter of the Central Body (398600.442 km\(^3\)/s\(^2\) for Earth)  
\(r\) = Radius from Earth  
\(a\) = Orbit Semi-Major Axis

The first step we must take is finding the velocity of the parking orbit. If we use the variables \(r = 7000\) km, \(a = 7000\) km, and the standard gravitational parameter of Earth, we can find \(v\).

\[v_{park} = 7.546\text{ km/s}\]

Next, we must find the orbital characteristics of the transfer orbit. First, we must find the semi-major axis of the transfer orbit. To do this, we can take the average of the semi-major axes of the target orbit and the parking orbit.

\[a = \frac{a_{Target} + a_{Parking}}{2} = \frac{20,000 + 7,000}{2} = 13,500\text{ km}\]

Now, we can find the velocity at periapsis of this transfer orbit. In this case \(r = 7000\) km, and \(a = 13,500\) km. We plug these into the Vis-Viva equation to get:

\[v_{transfer\_peri} = 9.185\text{ km/s}\]

Then, we can calculate the \(\Delta v\) of the first maneuver:

\[\Delta v_1 = v_{transfer\_peri} - v_{park} = 9.185\text{ km/s} - 7.546\text{ km/s} = 1.639\text{ km/s}\]

This first burn will put our Spacecraft into its transfer orbit. Next, we need to calculate the speed at the transfer orbit’s apoapsis. For this calculation, \(r = 20,000\) km, and \(a = 13,500\) km. We plug these into the Vis-Viva equation to get:

\[v_{transfer\_apo} = 3.215\text{ km/s}\]

Now, we must calculate the velocity of the target orbit. For the variables, \(r = 20,000\) km, and \(a = 20,000\) km. We plug these into the Vis-Viva equation to get:

\[v_{target} = 4.464\text{ km/s}\]

Now, we can calculate the \(\Delta v\) for the insertion burn, and finally the total \(\Delta v\):

\[\Delta v_2 = v_{target} - v_{transfer\_apo} = 4.464\text{ km/s} - 3.215\text{ km/s} = 1.249\text{ km/s}\]

\[\Sigma \Delta v = \Delta v_1 + \Delta v_2 = 2.888\text{ km/s}\]
We can also create a Mission Plan to calculate this for us.

**Hohmann Transfer - Earth Centered**

In this section, we will write a Mission Plan that will not only visualize a Hohmann transfer, but calculate it for us as well.

- Create a new Mission Plan and save it as "HohmannEarthCentered.MissionPlan"

**Building the Mission Sequence**

- Drag and drop a FreeForm script editor in the Mission Sequence
- Double Click on the FreeForm script editor
- Rename it "User Input"

In this FreeForm script editor, we will have a space that the user can define the parking orbit SMA and the target orbit SMA. To do this, we will write:

```plaintext
// User input for Hohmann Calculation
Variable parkingSMA = 7000;
Variable targetSMA = 20000;
```

Now, we will move on to calculating the Hohmann transfer.

- In the Mission Sequence, drag and drop a second FreeForm script editor after the "User Input" FreeForm
- Double-click the FreeForm script editor
- Rename it "Hohmann Calculations"

In this FreeForm script editor, we will calculate everything needed for the Hohmann transfer using the same steps as the steps taken in Calculating Hohmann Transfers. First, we will calculate the parking orbit velocity, then the transfer semi-major axis, the velocity of the transfer, and the Δv required for that maneuver. To do this, we will write:

```plaintext
// Parking Speed Orbit
Variable vPark = sqrt(Earth.Mu * ( (2/parkingSMA) - (1/parkingSMA) ));

// Semi-Major Axis of the transfer trajectory
Variable transfSMA = (targetSMA + parkingSMA)/2;

// Velocity at periapsis of the transfer trajectory
Variable vTransfPeri = sqrt(Earth.Mu * ( (2/parkingSMA) - (1/transfSMA) ));

// Delta V of the first maneuver
Variable dV1 = vTransfPeri - vPark;
```
Next, we need to calculate the transfer trajectory velocity at apoapsis, the target orbit velocity, the magnitude of the second $\Delta v$, and the total $\Delta v$. To do this, we write:

```plaintext
// Velocity at apoapsis of the transfer trajectory
Variable vTransfApog = sqrt(Earth.Mu * ( (2/targetSMA) - (1/transfSMA) ));

// Velocity of the target orbit
Variable vTarget = sqrt(Earth.Mu * ( (2/targetSMA) - (1/targetSMA) ));

// Delta V of the second Maneuver
Variable dV2 = vTarget - vTransfApog;

// Total Delta V required
Variable totalDV = dV1 + dV2;
```

All the calculations for the Hohmann transfer have been performed at this point. Now, we need to move onto visualizing the Hohmann transfer.

**Adding a Spacecraft**

- Create a Spacecraft object through the Object Browser
- Give the Spacecraft the following Keplerian elements:
  - A: 7000 km
  - E: 0
  - I: 0 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg

So that we can ensure the Spacecraft SMA is the same as the one the user defined, double-click the "Hohmann Calculations" FreeForm script editor and add the following statement to the bottom:

```plaintext
// Assigns the Parking SMA to the spacecraft
Spacecraft1.A = parkingSMA;
```

**Adding the ViewWindow**

- Right-click the Object Browser and add a ViewWindow object
- In the ViewWindow editor, make sure that Spacecraft1 is checked under "Available Objects"
- Check "Show Name" for Spacecraft1 as well
- For the history mode, change it to "Unlimited" (this will help us visualize it better)
- Go to "Viewpoints"
- Change the reference frame to "Inertial"
- Press "Ok" to close the ViewWindow editor

**Adding ImpulsiveBurns**
• Right-click on the Object Browser to create an ImpulsiveBurn
  o  Add → Spacecraft Related → ImpulsiveBurn
• Double-click on "ImpulsiveBurn1"
• Change the attitude system to "VNB". This configures the ImpulsiveBurn such that the primary burn
direction is in the Velocity direction.
• Right-click "ImpulsiveBurn1" and Clone it
• Rename the cloned object to "ImpulsiveBurn2"

Building the Mission Sequence

• Drag and drop a While loop at the end of the Mission Sequence
• Change the While loop argument to "Spacecraft1.ElapsedTime < TIMESPAN(2 hours)"
• Drag and drop a Step command inside of the While loop
• Drag and drop an Update command inside of the While loop after the Step command
• After the While loop, add a FreeForm script editor
• Open the FreeForm script editor and rename it to "Perform Maneuver 1"

In this FreeForm, we will change the color of the Spacecraft tail, perform the first maneuver, then step to the
Spacecraft object's apoapsis. To do this, we write:

```freeflyer
// Changes color of SC tail
Spacecraft1.Color = ColorTools.Yellow;

// Assigns the calculated delta v value to the Impulsive Burn
ImpulsiveBurn1.BurnDirection[0] = dV1;

Maneuver Spacecraft1 using ImpulsiveBurn1;

// Steps the Spacecraft to apoapsis and visualizes the Spacecraft
WhileStepping Spacecraft1 to (Spacecraft1.OrbitApoapsis);
  Update ViewWindow1;
End;
```

• Drag and drop another FreeForm script editor into the Mission Sequence after "Perform Maneuver 1".
  Name this "Perform Maneuver 2".

In this FreeForm, we will change the color of the Spacecraft again and perform the second maneuver. Then, we'd
like to report the magnitude of each maneuver, and the total ∆v of the transfer. To do this, we write:

```freeflyer
// Changes the color of SC tail again
Spacecraft1.Color = ColorTools.Lime;

// Sets the calculated delta v to the Impulsive Burn
ImpulsiveBurn2.BurnDirection[0] = dV2;
```
Maneuver Spacecraft1 using ImpulsiveBurn2;

// Reports the delta v values
Report dV1, dV2, totalDV;

We are done writing script for this mission. Now let's add in another final While loop to visualize the rest of the mission.

- In the Mission Sequence, drag and drop a While loop at the end of the sequence
- Drag and drop a Step command inside of the While loop
- Drag and drop an Update command inside of the While loop after the Step command

Your Mission Sequence should look something like this:

<table>
<thead>
<tr>
<th>Mission Sequence Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>9</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>12</td>
</tr>
</tbody>
</table>

Save and run your Mission Plan, then try to answer these questions:

Were the delta-v calculations close to the ones we calculated by hand?

Change the parking orbit to a SMA of 9000 km. Run the simulation again. Does the amount of required Δv increase or decrease with larger parking orbits?

See Also

- Maneuvering Tutorial
- Next Topic: Bi-Elliptic Transfer
4.2 - Bi-Elliptic Transfer

In the previous section covering Hohmann transfers, we stated that Hohmann transfers are typically the most efficient orbit transfers. However, this raises the question: When is it not?

The diagram below depicts a bi-elliptic transfer. In the bi-elliptic transfer, the first transfer is a highly eccentric orbit with an apoapsis higher than the target orbit radius. Once the spacecraft has reached apoapsis, it performs a burn raising its periapsis to the height of its target orbit. Finally, once it reaches periapsis, it performs an orbital insertion burn to put it into a circular orbit.

If you look at the graph below, you can see the relative efficiencies of a Hohmann transfer versus a bi-elliptic transfer. The graph below graphs the amount of \( \Delta v \) required to get to a target orbit using a bi-elliptic transfer. The x-axis shows the ratio of the target orbit to the parking orbit. The y-axis is an arbitrary value of \( \Delta v \) to show the relative efficiencies of the bi-elliptical transfer. The different lines indicate the \( \Delta v \) required if the intermediate orbit is x times bigger than the target orbit. If you'll notice, the black line is 1x, which is the same thing as a Hohmann transfer.

Another thing to note: At an orbit ratio of approximately 11.94, the Hohmann transfer represented by the black line, loses its spot as the most efficient transfer. However, not all the lines are below the black line at this point. This means that when the target orbit to parking orbit ratio is above 11.94, it may be more efficient to perform a bi-elliptic transfer. However, once it reaches a ratio of approximately 15.58, all bi-elliptic transfers are more efficient.
**Bi-Elliptic Transfer ∆v Requirements**

The table below illustrates the most efficient transfer based on the orbit ratio:

<table>
<thead>
<tr>
<th>Orbit Ratio</th>
<th>Most Efficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Orbit Ratio &lt; 11.94</td>
<td>Hohmann</td>
</tr>
<tr>
<td>11.94 &lt; Orbit Ratio &lt; 15.58</td>
<td>Depends on size of Bi-Elliptic Transfer</td>
</tr>
<tr>
<td>Orbit Ratio &gt; 15.58</td>
<td>Bi-Elliptic</td>
</tr>
</tbody>
</table>

**Modeling the Bi-Elliptic Transfer**

**Problem:**
A Spacecraft object's parking orbit is at a radius of 7,000 km. The target orbit has a semi-major axis of 105,000 km. Model a bi-elliptic transfer with an intermediate transfer orbit that is 2x the semi-major axis of the target orbit.
Create a new Mission Plan and save it as "BiEllipticTransfer.MissionPlan"

**Adding a Spacecraft**

- Create a Spacecraft with the following Keplerian elements
  - A: 7000 km
  - E: 0
  - I: 0 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg
- In the Spacecraft object’s force model, uncheck the “Moon” and “Sun” boxes. (This is to make sure no unwanted perturbations occur)
- Click "Ok" to close the Spacecraft editor

**Adding ImpulsiveBurns**

- Create an ImpulsiveBurn object through the Object Browser
- Double-click on "ImpulsiveBurn1"
- Change the attitude system to "VNB"
- Press "Ok" to close the editor
- Right-click "ImpulsiveBurn1" and clone it twice
- Rename the two clones "ImpulsiveBurn2" and "ImpulsiveBurn3"
- Create a ViewWindow through the Object Browser
- Double-click "ViewWindow1"
- Check "Spacecraft1" in the "Available Objects" list
- Check the "Show Name" box for Spacecraft1
- Change the history mode to "Unlimited" (This will allow for better visualization)
- Go to the "Viewpoints" section on the left-hand side
- On the default view, change the reference frame to "Inertial"
- Change the declination value to 90 deg
- Change the radius value to 700,000 km
- Press "Ok" to close the ViewWindow editor

**Building the Mission Sequence**

- Drag and drop a FreeForm script editor onto the Mission Sequence
- Double-click the FreeForm script editor
- Rename the script to "User Input"

In this section, we want the user to define the parking orbit, the target orbit, and the magnitude of the bi-elliptic transfer. Also, we need to make sure that the parking orbit the user has defined actually gets assigned to the Spacecraft. To do this, we write:
Now, we need a FreeForm script editor to calculate all the maneuvers needed for the bi-elliptic transfer.

- In the Mission Sequence, drag and drop another FreeForm script editor after the "UserInput" FreeForm script editor
- Double-click the FreeForm script
- Rename the FreeForm script editor to "Calculate Bi-Elliptic Transfer"

In this FreeForm, there are several calculations we need to do to solve the bi-elliptic transfer problem. Ultimately, this FreeForm script editor will calculate the ∆v required for each of the maneuvers, then assign those values to each ImpulsiveBurn object. However, we will take this FreeForm step by step so it doesn't get too complicated. First, let's try and solve the ∆v of the first maneuver. To do this, we will be using the Vis-Viva equation, just like we did in the Hohmann transfer section. We will need to calculate the SMA of the first transfer orbit, the speed at periapsis of that orbit, the speed of the parking orbit, and the difference between the two. To do this, we write:

```plaintext
// Calculations for Maneuver 1
// SMA of the first transfer orbit
Variable transfSMA1 = (targetSMA * transferMagnitude + parkingSMA)/2;

// Speed at periapsis for the first transfer orbit
Variable transfPeri1 = sqrt(Earth.Mu * (2/parkingSMA) - (1/transfSMA1) );

// Delta V for the first maneuver
Variable dV1 = transfPeri1 - Spacecraft1.VMag;
```

Next, we need to calculate the second maneuver. There are a few more things we need to calculate this time. We have the SMA of the first transfer orbit, but we still need the radius of the apoapsis of that first orbit. To calculate this, we write:

```plaintext
// Calculations for Maneuver 2
// Radius of Apoapsis of the first transfer orbit Apoapsis
Variable radiusApo1 = transfSMA1 * 2 - parkingSMA;
```

We have the radius of apoapsis of the first transfer orbit. Still, we need the SMA of the second transfer orbit. To calculate this, let's look at the diagram below:
Based on this diagram, we can derive a very simple equation to solve for second transfer orbit SMA:

$$2a_{\text{transfer}_2} = 2a_{\text{transfer}_1} + (a_{\text{target}} - a_{\text{park}})$$

$$a_{\text{transfer}_2} = \frac{2a_{\text{transfer}_1} + (a_{\text{target}} - a_{\text{park}})}{2}$$

Now, we can write a statement calculating exactly this:

```c
// SMA of the second transfer Orbit
Variable transfSMA2 = (2 * transfSMA1 + (targetSMA - parkingSMA))/2;
```

Next, we need to calculate the velocity of the Spacecraft at the apoapsis of the first transfer orbit, the velocity it needs to match the second transfer orbit, and the Δv. To calculate this, we write:
Now that we have calculated the first two maneuvers, we need to calculate the third and final maneuver - the orbital insertion burn. To do this, we need to calculate the velocity of the Spacecraft at the periapsis of the second transfer orbit, the velocity it needs to be to match the target orbit, and the Δv. To calculate this, we write:

```plaintext
// Speed at apoapsis for the first transfer orbit
Variable transfApog1 = sqrt(Earth.Mu * ( (2/radiusApog1) - (1/transfSMA1) ));

// Speed at apoapsis for the second transfer orbit
Variable transfApog2 = sqrt(Earth.Mu * ( (2/radiusApog1) - (1/transfSMA2) ));

// Delta V for the second maneuver
Variable dV2 = transfApog2 - transfApog1;
```

At this point, we have calculated the values for all three maneuvers and we can assign these values to the ImpulsiveBurn objects. To do this, we write:

```plaintext
// Assigns Delta V values to each Maneuver
ImpulsiveBurn1.BurnDirection[0] = dV1;
ImpulsiveBurn2.BurnDirection[0] = dV2;
ImpulsiveBurn3.BurnDirection[0] = dV3;
```

Now, let's move onto the rest of the Mission Sequence.

- In the Mission Sequence, drag and drop a while loop after both FreeForm script editors
- Change the Argument inside the while loop to "(Spacecraft1.ElapsedTime < TIMESPAN(4 hours))"
- Inside the while loop, drag and drop a "Step" command
- Inside the while loop, drag and drop an "Update" command after the "Step" command
- Outside after the while loop, drag and drop three FreeForm script editors after the while loop
- Rename them "Perform Maneuver 1", "Perform Maneuver 2", and "Perform Maneuver 3"

Now, we will go into each of these scripts and have the Spacecraft perform the maneuvers.
To start, let's go into the "Perform Maneuver 1" Script. In this script, we will change the Spacecraft tail color, Maneuver the Spacecraft, then step the Spacecraft to its orbit apoapsis. To do this, we write:

```csharp
// Changes the tail color for the first transfer orbit
Spacecraft1.Color = ColorTools.Yellow;
Maneuver Spacecraft1 using ImpulsiveBurn1;

// Steps the spacecraft until its orbit apoapsis
WhileStepping Spacecraft1 to (Spacecraft1.OrbitApoapsis);
    Update ViewWindow1;
End;
```

Now let's go into the second maneuver. Open the "Perform Maneuver 2" Script. In this script, we will do the same as before, but we will perform "ImpulsiveBurn2" instead and step the Spacecraft to its periapsis. To do this, we write:

```csharp
// Changes the tail color for the second transfer orbit
Spacecraft1.Color = ColorTools.Lime;
Maneuver Spacecraft1 using ImpulsiveBurn2;

// Steps the spacecraft to its periapsis
WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
    Update ViewWindow1;
End;
```

Lastly, let's go into the "Perform Maneuver 3" FreeForm script editor. In this script, we will change the tail color, perform the maneuver, and then report the values of each maneuver, as well as the total required ∆v. To do this, we write:

```csharp
// Changes the tail color for the target orbit
Maneuver Spacecraft1 using ImpulsiveBurn3;

// Calculates the total Delta V used for the Bi-Elliptical transfer
Variable totalDV = dV1 + dV2 + abs(dV3);

// Reports each Maneuver value and the total amount of Delta V used
Report dV1, dV2, dV3, totalDV;
```

Note that when we calculated "totalDV", we had to use the absolute value of "dV3". This is because for our third maneuver, we need to turn the Spacecraft around and burn in the opposite direction of our velocity vector. Because of this, the value is negative and needs to be positive when calculating the total ∆v.

Let's head back to the Mission Sequence to finish the Mission Plan.
Below the "Perform Maneuver 3" FreeForm script editor, add a while loop
For the while loop argument, change it so it says ":(Spacecraft1.ElapsedTime < TIMESPAN(5 days))"
Inside the while loop, add a "Step" command
Inside the while loop, add an "Update" command after the "Step" command

Your Mission Sequence should look something like this:

Save and Run your Mission Plan to see your results. After running this Mission Plan, answer the following questions:

How did FreeFlyer's calculations compare to your own calculations by hand?

How much Δv did each maneuver cost? How much total Δv was used?

Open the Hohmann transfer Mission Plan you just built. Run the mission with the same parking and target SMAs as this problem. How does the total Δv compare to the bi-elliptic's total Δv?

Go back to the Bi-Elliptic Transfer Mission Plan. In the user input, change the "transferMagnitude" to 5. Did the total amount of Δv increase or decrease?

See Also

- Maneuvering Tutorial
- Next Topic: Phasing Maneuver
- Previous Topic: Hohmann Transfer
4.3 - Phasing Maneuver

In this section, we will discuss phasing maneuvers.

Phasing maneuvers are maneuvers that change the size of the original orbit in order to meet the original orbit at a different point in time. If the spacecraft needs to rendezvous with another spacecraft behind it, it would speed up to increase its period just enough to intercept the other spacecraft. If it needed to catch up to a spacecraft in front of it, the spacecraft would slow down to decrease its period. The maneuver has two burns. The first burn occurs in its original orbit. This shrinks or expands the orbit from its original state. Once the spacecraft completes the orbit and returns to its original burn point, it then performs its second maneuver, matching its original orbit. See the diagram below to see how this works:

![Phasing Maneuver Diagram](image)

What can this be used for? In the above diagram, a shuttle uses a phasing maneuver to rendezvous with a satellite that is $\Phi$ degrees behind it. Typically when attempting a rendezvous with another spacecraft, phasing maneuvers are used to either slow down or catch up to the target. Also, communications and weather satellites in a geosynchronous orbit that need to move to a different observation area will use a phasing maneuver to move to a different area.

Also, one important thing to note is that sometimes when a spacecraft wants to save fuel, it might make a phasing orbit closer to its original orbit and orbit it multiple times before performing its orbital insertion burn. Once it meets up with its target, it will perform its orbital insertion burn.

The rest of this section will cover the following:

1. Calculating Phasing Maneuvers
2. Modeling a Phasing Maneuver

Calculating Phasing Maneuvers

Calculating a phasing maneuver is relatively simple. Think of it as a Hohmann transfer to and from its own orbit. But how do we calculate it by hand?

<table>
<thead>
<tr>
<th>Problem:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Say we have a geosynchronous satellite with a SMA of 42,164 km, and a true anomaly of 25 degrees. It is currently sitting above the 285 degree longitude line (75° W) observing the DC Metro area. However, this satellite is being re-purposed and needs to be put over the 140 degree longitude line over Japan. How much Δv is required to make a phasing maneuver to move this satellite to its new location? How long will it take to move it there?</td>
</tr>
</tbody>
</table>

Let's take this step by step.

1. **Determine how far ahead/behind the target is in time**

   First, let's see how far behind Japan is in degrees:

   \[ \theta = 285 - 140 = 145^\circ \]

   Since the Earth has a rotational period of 24 hours, we can calculate the time behind Japan is:

   \[ \text{Time behind} = \left(\frac{145^\circ}{360^\circ}\right) \times 24 \text{ hr} = 34,800 \text{ seconds} \]

2. **Determine the period needed for the new orbit**

   Now that we have the time behind, we need to find the period of the new orbit. Since Japan is behind our current position in orbit, we need to slow down to it. This means that we will be expanding our orbit, increasing our SMA, and therefore increasing our period. This means that the new orbit needs to be the "Time Behind" + Our current period.

   \[ \text{New Period} = \text{Time behind} + \text{GEO Period} = 34,800 \text{ s} + 86,164 \text{ s} = 120,964 \text{ seconds} \]

3. **Calculate the SMA of the new orbit**

   We need to use the following formula to calculate the new SMA:

   \[
   a = \sqrt[3]{\frac{\mu \times \left(\frac{T}{2\pi}\right)^2}{\mu}}
   \]

   **Semi-Major Axis as a Function of Orbital Period**

   With this, we calculate:
a = 52,836.718 km

4. **Calculate the ∆v needed**

The final step of this process is to calculate the current velocity of our spacecraft and the velocity needed to create the correct phasing orbit. To do this, we will use the Vis-Viva Equation.

Current Speed: \( r_{\text{circ}} = 42,164 \text{ km}, \ a_{\text{circ}} = 42,164 \text{ km} \)

\[ v_1 = 3.075 \text{ km/s} \]

Phasing Speed: \( r_{\text{peri}} = 42,164 \text{ km}, \ a_{\text{phasing}} = 52,864.416 \text{ km} \)

\[ v_2 = 3.371 \text{ km/s} \]

∆v:

\[ ∆v = 0.296 \text{ km/s} \]

Keep in mind, this ∆v value is only for one burn. However, the second burn will be identical to the first in magnitude, but in the opposite direction. If we double this value, we will have the total ∆v required for this mission.

**Modeling a Phasing Maneuver**

Let’s use FreeFlyer to test our result.

- Open a new Mission Plan and save it as "GEOPhasing.MissionPlan"

**Adding a Spacecraft**

- Create a Spacecraft with the following Keplerian elements
  - A: 42,164 km
  - E: 0
  - I: 0 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 25 deg
- In the Spacecraft object’s force model, uncheck the "Moon" and "Sun" boxes (this is to make sure no unwanted perturbations occur)

**Adding a Spacecraft Sensor**

- Click "Sensors" on the left-hand side of the Spacecraft editor
Sensors Section in the Spacecraft Editor

- Click "Create" to create a Sensor
- Click "Edit Sensor" to open the Sensor editor

Create and Edit a Sensor

- Change the "Cone Half Angle" to 10 degrees

Sensor Cone Half Angle

- Click "Ok" to close the Sensor editor
- Click "Ok" to close the Spacecraft editor

Adding an ImpulsiveBurn

- Create an ImpulsiveBurn object in the Object Browser
- Open the ImpulsiveBurn editor
- Change the attitude system to "VNB"
Enter the $\Delta v$ value you calculated earlier into the "Velocity" box of the "Delta-V Directions and Magnitudes" section

Click "Ok" to close the editor

**Adding ViewWindows for Output**

- Create a ViewWindow object in the Object Browser
- Open the ViewWindow editor
- Check "Spacecraft1" in the "Available Objects" section
- Check "Show Name" for Spacecraft1
- Change the history mode to "Unlimited"
- Go into "Viewpoints" on the left-hand side
- Change the reference frame to "Inertial"
- Change the declination to 90 deg
- Change the radius to 155,000 km
- Click "Ok" to close the editor

- Clone "ViewWindow1"
- Double-click "ViewWindow1_Copy1"
- Rename it to "ViewWindow2"
- Go into "Viewpoints" on the left-hand side
- Change the type to "2D Map"
- Change the reference frame to "Body Fixed"
- Press "Ok" to close the editor

**Building the Mission Sequence**

- Drag and drop a "While...End" loop into the Mission Sequence
- Drag and drop a "Step" command inside that While loop
- Drag and drop an "Update" command inside the While loop after the "Step" command
- Drag and drop another "Update" command after the previous one
  - Modify this "Update" command to update ViewWindow2
- Drag and drop a FreeForm script editor outside of the end of the loop
- Rename the script to "Perform Maneuver 1"

In this FreeForm, we will be changing the color of Spacecraft1, performing the maneuver, then propagating for one full orbit. However, we can't just use "Step Spacecraft1 to (Spacecraft1.OrbitPeriapsis)" since we are already at periapsis. To get around this, we will step the Spacecraft once, and then step to periapsis. To do this we write:

```plaintext
// Changes the spacecraft color for the maneuver
Spacecraft1.Color = ColorTools.Lime;

Maneuver Spacecraft1 using ImpulsiveBurn1;

// Steps the spacecraft, then step it to its periapsis
Step Spacecraft1;
Update ViewWindow1;
```
Now our Spacecraft will perform the phasing maneuver, and orbit the Earth once.

- In the Mission Sequence, drag and drop another FreeForm script editor at the end of the Mission Sequence
- Rename this FreeForm to "Perform Maneuver 2"

In this script, we will be changing the Spacecraft color and then perform the maneuver. This maneuver is exactly the same in the first maneuver, but in the reverse direction. To adjust this, we can multiply that component by -1. To do this, we write:

```plaintext
// Changes the spacecraft color back to its original color

// Reverses the direction of ImpulsiveBurn1
ImpulsiveBurn1.BurnDirection[0] *= -1;

Maneuver Spacecraft1 using ImpulsiveBurn1;
```

Next, we need to step the Spacecraft for a day to make sure that we are over our target.

- Drag and drop a "While...End" loop into the Mission Sequence after "FreeForm: Perform Maneuver 2"
- Drag and drop a "Step" command inside that loop
- Drag and drop two "Update" commands after the "Step" command in that loop
  - Modify the second "Update" command to update ViewWindow2

Your Mission Sequence should look something like this:
Save and run your Mission Plan, then try to answer these questions:

*Did you end up over your target? Check your sensor view and the line of longitude your Spacecraft is over.*

*Calculate the $\Delta v$ values for orbiting a phasing orbit twice. Try putting in those values for the burn. Make sure you adjust the code so the Spacecraft will orbit twice. Was it successful? How much less total $\Delta v$ was used?*

**See Also**

- Maneuvering Tutorial
- Next Topic: Plane Change Maneuver
- Previous Topic: Bi-Elliptic Maneuver
4.4 - Plane Change Maneuver

Plane change maneuvers are known as one of the more costly maneuvers for spacecraft to perform. However, the math for them is relatively simple. In fact, you only need to have a basic understanding of trigonometry and vector math to calculate the Δv needed in a plane change.

Say we wanted to make an inclination change. If we burned normal to the orbital plane, we would be adding Δv in the y direction of the VNB attitude system. However, this would make our resultant vector larger than what we started with, thus changing the shape of the orbit. See the diagrams below:
Our resultant velocity is the hypotenuse in this triangle. Because of that, it will definitely be larger than our original velocity. With this, we are not only changing the inclination of the orbit, but we are changing the eccentricity. How can we burn in such a way that the resultant velocity vector has the same magnitude as our original velocity and not change the eccentricity of the orbit? The trick is to tilt the $\Delta v$ vector backwards. Doing this, we will get an isosceles triangle like the diagram below:

![Diagram of Inclination Change Burn]

With this, we can break down the $\Delta v$ vector into two components: the normal component, and the negative velocity component. Using trigonometry, we can calculate these values quite easily. The formulas for the the $\Delta v$ are as follows:

\[ \theta = \text{Inclination} \]
\[ \Delta v_n = \Delta v \text{ Normal Component} \]
\[ \Delta v_{nv} = \Delta v \text{ Negative Velocity Component} \]
\[ v_{\text{orig}} = \text{Original Velocity} \]
\[ v_{\text{res}} = \text{Resultant Velocity} \]
\[ v_{\text{orig}} = v_{\text{res}} \]
\[ \Delta v_n = v_{\text{orig}} \cdot \sin(\theta) \]
\[ \Delta v_{nv} = v_{\text{orig}} \cdot (1 - \cos(\theta)) \]

Now that we have worked out the math, let's try writing a Mission Plan that calculates and performs a plane change.
Modeling Plane Change Maneuvers

Problem:
We have a spacecraft in a circular, equatorial orbit with a SMA of 7200 km. How much Δv would we need to change the inclination to 30° without changing the shape of the orbit?

We can build our own calculator for this in FreeFlyer. But first, let's add in the necessary objects in the Object Browser.

- Open a new Mission Plan and save it as "CircularPlaneChange.MissionPlan"

Adding a Spacecraft

- Create a new Spacecraft with the following orbital elements:
  - A: 7200 km
  - E: 0
  - I: 0 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg

- Create a new ImpulsiveBurn object
- Open the ImpulsiveBurn editor
- Change the attitude system to "VNB"
- Click "Ok" to close the editor

Adding the ViewWindow

- Create a new ViewWindow object
- Open the ViewWindow editor
- Check “Spacecraft1” in the "Available Objects" section
- Check "Show Name"
- Change the history mode to "Unlimited"
- Go into "Viewpoints" on the left-hand side
- Change the reference frame to "Inertial"
- Click "Ok" to close the editor

Building the Mission Sequence

- Drag and drop a FreeForm script editor into the Mission Sequence
- Name this "Calculate Plane Change"

In this script, we will assign an inclination angle, calculate the Δv required, and assign it to the ImpulsiveBurn object. To do this, we write:
// Inclination to change to in degrees
Variable theta = 30;
theta = rad(theta);

// Calculate the plane change
Variable norm = Spacecraft1.VMag * sin(theta);
Variable negVel = Spacecraft1.VMag * (1 - cos(theta));

// Assigns the values to the burn
ImpulsiveBurn1.BurnDirection = { -negVel, norm, 0};

Remember, we are using the VNB attitude system. This is why we need to put a negative sign in front of the "negVel" variable, as VNB's x-component is the positive direction of the velocity vector.

Now, let's propagate the Spacecraft for a day so we can visualize its original orbit.

- In the Mission Sequence, drag and drop a "While...End" loop
- Inside that loop, drag and drop a "Step" command
- Drag and drop an "Update" command after the "Step" command inside the loop
- Drag and drop a FreeForm script editor below the While loop
- Rename this to "Perform Plane Change"

In this FreeForm, we need to change the Spacecraft object's color, maneuver the Spacecraft, and report the total Δv used for the plane change. To do this, we write:

```
// Change the spacecraft tail color
Spacecraft1.Color = ColorTools.Lime;

Maneuver Spacecraft1 using ImpulsiveBurn1;

// Reports the total Delta V used for the plane change
Report ImpulsiveBurn1.BurnDirection.Norm();
```

Now, we need to propagate the Spacecraft for one day to visualize the new orbit.

- In the Mission Sequence, drag and drop another "While...End" loop
- Inside the loop, drag and drop a "Step" command
- Drag and drop an "Update" command after the "Step" command inside the loop

Your Mission Sequence should look something like this:
Save and run your Mission Plan, then try to answer these questions:

*Try solving this problem by hand. How do your calculations compare to FreeFlyer’s calculations?*

*Change Spacecraft1’s SMA to 30,000 km and run the Mission Plan again. How does the amount of Δv needed change for slower Spacecraft?*

*If you were to use a bi-elliptic transfer and wanted to perform a plane change as well, where in the transfer would be the most efficient position to perform a plane change?*

**See Also**

- Maneuvering Tutorial
- Previous Topic: Phasing Maneuver
CHAPTER 5

Interplanetary Topics
In this chapter, we will discuss various aspects of interplanetary travel. We will take a look at the math behind these concepts and attempt to apply them in FreeFlyer in order to visualize them better.

This chapter will cover the following topics:

1. Interplanetary Hohmann Transfer
2. Patched Conics Transfer
3. Gravity Assist
4. The B-Plane
5.1 - Interplanetary Hohmann Transfer

Hohmann transfers are not just for Earth orbiting spacecraft - they can also be used for interplanetary transfers. Calculating an interplanetary Hohmann transfer is very similar to calculating a Hohmann transfer for an Earth orbiting spacecraft. The only difference we have is that we have one more thing to calculate: The necessary phase angle for the two planets.

In this section, we will cover:

1. Calculating an Interplanetary Hohmann Transfer
2. Modeling an Interplanetary Hohmann Transfer

Calculating an Interplanetary Hohmann Transfer

Calculating the ∆v required for an interplanetary Hohmann transfer is exactly like how we did it in the Hohmann Transfer tutorial. Our "parking" orbit SMA is actually our departure planet's SMA about the Sun. Our "target" orbit SMA is the arrival planet's SMA about the Sun.

However, like the Hohmann Transfer tutorial, we must assume that the two planets are both circular and co-planar. Since this definitely isn't the case with any of our solar system's planets in the real world, these calculations only present a conceptual idea of the amount of ∆v required for an interplanetary transfer.

One more thing we need to do in addition to the ∆v calculations is calculating the necessary phase angle between the planets. The planets need to be at a certain position relative to each other so that when the interplanetary spacecraft reaches the other side of the Hohmann transfer, the arrival planet is there as well. The phase angle 'Φ' is shown here:
You can calculate the phase angle using the following formula:

\[
\phi = 180^\circ - \frac{1}{2} \cdot T_{\text{Hoh}} \cdot \dot{\theta}_{\text{Target}}
\]

\[T_{\text{Hoh}} = \text{Period of the Hohmann transfer orbit}\]
\[\dot{\theta}_{\text{Target}} = \text{Angular velocity of the target planet}\]

For this formula, you need the period of the Hohmann transfer, and the angular velocity of the target planet. What we are essentially doing is finding how many degrees the target planet will travel during the time of the Hohmann transfer, which is half of the Hohmann transfer period. To calculate the period of the Hohmann transfer and the angular velocity of the target orbit, we need the following formulas:

\[T_{\text{Hoh}} = 2\pi \cdot \sqrt{\frac{a_{\text{Hoh}}^3}{\mu}}\]
\[\dot{\theta}_{\text{Target}} = \frac{360^\circ}{2\pi} \cdot \sqrt{\frac{\mu}{a_{\text{Target}}^3}}\]

\[a_{\text{Hoh}} = \text{SMA of the Hohmann transfer}\]
\[a_{\text{Target}} = \text{SMA of the target planet}\]
\[\mu = \text{Standard Gravitational Parameter of the central body}\]

It is important to note that the formula for the angular velocity is only true when dealing with a circular orbit. Because our interplanetary Hohmann transfer assumes a perfectly circular orbit for both planets, we can use this formula.

**Modeling an Interplanetary Hohmann Transfer**

When calculating Hohmann transfers, we must first assume that both orbits are circular. In the real world, the orbits of Earth and Mars are not circular. So to model an interplanetary Hohmann transfer, we will be using Spacecraft in heliocentric circular orbits with the same SMA as the planets they are representing. Because basic interplanetary Hohmann transfers only rely on the gravity of the central body, we do not need to model the departure and arrival planets' gravities in our problem.

**Problem:**
Assume that Earth and Mars are in circular orbits around the Sun at 1 AU and 1.524 AU, respectively. How much ∆v is required to perform a Hohmann transfer to Mars? How many days would this transfer take?

- Create a new Mission Plan and save it as "InterplanetaryHohmann.MissionPlan"

**Adding in Spacecraft**

- Create a Spacecraft with the following elements:
  - Central Body: "Sun"
  - Reference Frame: "Mean of J2000 Earth Ecliptic"
  - A: 149,597,871 km (This is 1 AU)
  - E: 0
  - I: 0 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 0 deg

  **NOTE:** Remember that you need to change the Element Type to "Keplerian" to access these elements

- Rename the Spacecraft to "InterplanetarySC"
- Click on the "Force Model" on the left-hand side
- Uncheck the "Earth" and "Moon" boxes
- Click on "Propagator" on the left-hand side
- Change the step size to 1 day
- Click "Ok" to close the editor

- Clone "InterplanetarySC"
- Rename the clone to "MarsSC" (this Spacecraft will represent Mars)
- Change A to 227,987,155 km (This is 1.524 AU)
- Click on "Visualization" on the left-hand side
- Change the tail color to green
- Click "Ok" to close the editor

**Adding in the ViewWindow**

- Create a ViewWindow through the Object Browser
- Double click on "ViewWindow1" to open the editor
- Check each Spacecraft in the "Available Objects"
- Click on "Spacecraft" in the "Available Objects" to select both Spacecraft, then check "Show Name"
- Change the history mode to "Unlimited" (for both Spacecraft)

Since we won't be needing to show the real Earth and the real Mars, let's hide them from the ViewWindow.

- Click on the "Solar System" section on the left-hand side
- Click on "Earth"
- Uncheck "Show Object" in "Object Options"
- Click on "Mars"
• Uncheck "Show Object" in "Object Options"

Now we can continue with the rest of the settings for the ViewWindow.

• Click on "Viewpoints" on the left-hand side
• Change the reference frame to "Inertial"
• Change the Source to "Sun"
• Click "Copy to Target/Tail Ref."
• In "Source Offsets", change the radius to 500,000,000 km
• Click "Ok" to close the editor

Adding an ImpulsiveBurn

• Create an ImpulsiveBurn object through the Object Browser
• Double-click on "ImpulsiveBurn1" to open the editor
• Change the attitude system to "VNB"
• Click "Ok" to close the editor

Building the Mission Sequence

To start, we'll propagate the entire solar system for a while so we can see each planet's orbit better.

• Drag and drop a while loop into the Mission Sequence
• Change the while loop argument to "(InterplanetarySC.ElapsedTime < TIMESPAN(500 days))"
• Drag and drop a FreeForm script editor inside that while loop
• Open the script editor and rename it to "Step and Update"
In this script, we will step both Spacecraft with an epoch sync, and update the ViewWindow. To do this, we write:

```plaintext
// Steps both spacecraft with an epoch sync
Step InterplanetarySC;
Step MarsSC to (MarsSC.Epoch == InterplanetarySC.Epoch);

// Updates the ViewWindow
Update ViewWindow1;
```

Let's go back to the Mission Sequence.

- Drag and drop a FreeForm script editor after the while loop
- Open the script editor and rename it to "Calculate Hohmann Delta V"

In this FreeForm script editor, we will calculate the necessary ∆v needed and assign it to the ImpulsiveBurn object we created. To do this, we write:

```plaintext
// SMAs of the departure and arrival planets
Variable startingOrbit = InterplanetarySC.A;
Variable arrivalOrbit = MarsSC.A;

// SMA of the Hohmann transfer
Variable transfSMA = (startingOrbit + arrivalOrbit)/2;

// Velocity of the Hohmann transfer at Periapsis
Variable vTransfPeri = sqrt(Sun.Mu * ((2/startOrbit) - (1/transfSMA)));

// Delta V for the Hohmann transfer
Variable dV1 = vTransfPeri - InterplanetarySC.VMag;
ImpulsiveBurn1.BurnDirection[0] = dV1;
```

Next, we need to calculate the phase angle. Let's add another FreeForm script editor to the Mission Sequence.

- Drag and drop a FreeForm script editor after the "Calculate Hohmann Delta V" FreeForm
- Open the script editor and rename it to "Calculate Phase Angle"

In this script, we need to calculate the necessary phase angle for the Hohmann transfer. To do this, we can use the formulas given in the Calculating an Interplanetary Hohmann Transfer section. We will need to write:

```plaintext
Variable Pi = acos(-1);

// Period of the Hohmann transfer
Variable THoch = 2 * Pi * sqrt(transfSMA^3/Sun.Mu);

// Angular Velocity of the Target Planet
Variable angVelTarget = (360/(2 * Pi)) * sqrt(Sun.Mu/(arrivalOrbit^3));
```
Now that we've calculated the phase angle, we should try and calculate another very helpful thing: the next epoch at which this phase angle occurs. To do this, we will need to calculate two things: the current phase angle, and the phase angular velocity (the rate at which the phase angle changes).

The current phase angle is pretty easy to calculate. If we take the position vectors of each Spacecraft and use the "VertexAngle" method, we can calculate the angle between the two.

```csharp
// Current Phase Angle
Variable currentPhaseAngle = InterplanetarySC.Position.VertexAngle(MarsSC.Position);
```

However, this method will not return a value greater than 180 degrees. If Earth is ahead of Mars, we need to add 180 degrees to the phase angle. To do this, we can take the z component of the cross product of InterplanetarySC.Position and MarsSC.Position, and check to see if it's negative. If it is, that means we need to add 180 degrees. To do this, we write:

```csharp
// If Earth is in front of Mars, add 180 degrees to the current phase angle
If(InterplanetarySC.Position.CrossProduct(MarsSC.Position)[2] < 0) then;
    currentPhaseAngle += 180;
End;
```

Now, we need to calculate the phase angular velocity. In this scenario, this will simply be the difference between Earth's angular velocity and Mars's angular velocity. To calculate this, we write:

```csharp
// Starting Planet Angular Velocity
Variable angVelStarting = (360/(2 * Pi)) * sqrt(Sun.Mu/(startingOrbit^3));

// Phase Angular Velocity
Variable angVelPhase = angVelStarting - angVelTarget;
```

Now that we have the phase angular velocity, we can calculate how long it will take until we've reached our departure position. To calculate this, we take the difference of our current phase angle, and our departure phase angle. If we divide this difference by the phase angular velocity, we will have the amount of time (in seconds) until we've reached our departure position. Then, we can add that to our current epoch to calculate the departure epoch. To do this, we write:

```csharp
// Time until Departure
Variable timeTilDep = (currentPhaseAngle - phaseAngle)/angVelPhase;

// Departure Epoch
TimeSpan departureEpoch = InterplanetarySC.Epoch + TimeSpan.FromSeconds(timeTilDep);
```
We have done all the necessary calculations for our first maneuver. Now, we need to step to the departure date, maneuver, then step to the arrival date. Let's go back to the Mission Sequence.

- Drag and drop a FreeForm script editor after the "Calculate Phase Angle" FreeForm
- Open the script editor and rename it to "Step to Departure, Maneuver, Step to Arrival"

In this script, we will step to the departure epoch, maneuver the Spacecraft, change the Spacecraft tail color for a better visualization, calculate the arrival epoch, and step to the arrival epoch. To do this, we write:

```plaintext
// Steps to the departure time
While (InterplanetarySC.Epoch < departureEpoch);
    Step InterplanetarySC;
    Step MarsSC;
    Update ViewWindow1;
End;

// Maneuvers the spacecraft for the Hohmann transfer
Maneuver InterplanetarySC using ImpulsiveBurn1;

// Changes the tail color of the spacecraft
InterplanetarySC.Color = ColorTools.Cyan;

// Arrival Epoch
TimeSpan arrivalEpoch = InterplanetarySC.Epoch + TimeSpan.FromSeconds((1/2) * (THoh));

// Step to Arrival
While (InterplanetarySC.Epoch < arrivalEpoch);
    Step InterplanetarySC;
    Step MarsSC;
    Update ViewWindow1;
End;
```

The last thing we need to do for this transfer is to match our speed with our target. Let's go back to the Mission Sequence.

- Drag and drop a FreeForm script editor after the "Step to Departure, Maneuver, Step to Arrival" FreeForm
- Open the script editor and rename it to "Orbit Matching Maneuver"

In this script, we will need to calculate speed of Mars's orbit, calculate the ∆v required to match the orbit, maneuver the spacecraft, then propagate for 300 days to visualize this change. To do this, we write:

```plaintext
// Velocity of Mars orbit
Variable vMarsOrbit = sqrt(Sun.Mu * ((2/arrivalOrbit) - (1/arrivalOrbit)));
```
// Delta V required for maneuver
Variable dV2 = vMarsOrbit - InterplanetarySC.VMag;

ImpulsiveBurn1.BurnDirection[0] = dV2;

Maneuver InterplanetarySC using ImpulsiveBurn1;

// Propagates SC for 300 days
While(InterplanetarySC.ElapsedTime < TIMESPAN(300 days));
    Step InterplanetarySC;
    Step MarsSC to (MarsSC.Epoch == InterplanetarySC.Epoch);
    Update ViewWindow1;
End;

One more thing we need to add to the script - the thing we've been looking for all along! We need to report the Δv, and the time of flight in days. For the time of flight, we can simply take the difference of the arrival epoch and the departure epoch as these are measured in days. To report these values, we write:

Report (dV1 + abs(dV2)), (arrivalEpoch - departureEpoch).ToDays();

Your Mission Sequence should look something like this:

<table>
<thead>
<tr>
<th>Mission Sequence Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>7</td>
</tr>
</tbody>
</table>

Save and run your Mission Plan, then try and answer these questions:

How much total Δv was required for the transfer?

How many days did the transfer take?

Try reporting the distance between the two spacecraft at the time where InterplanetarySC "meets" MarsSC (right before the orbit matching maneuver). To do this, you can add the command "Report InterplanetarySC.RadialSeparation(MarsSC)" right before the command to perform the second maneuver. About how far apart were the Spacecraft?
See Also

- Interplanetary Topics
- Next Topic: Patched Conics Transfer
5.2 - Patched Conics Transfer

When we solved the interplanetary Hohmann problem, we found the $\Delta v$ needed for a spacecraft in an Earth orbit to rendezvous and match a Mars orbit. However, this model is incredibly simplified. In real life, a spacecraft would have to start in a parking orbit around Earth, perform a maneuver, and escape Earth's sphere of influence first. Furthermore, a mission designer would also need to consider the effects of Mars's gravity during the final portion of the problem.

Let's take the problem of finding the necessary $\Delta v$ to get a spacecraft from Earth to Mars and model it more realistically using the patched conics method. In patched conics, you break down the problem from one multi-body problem to multiple two-body problems. Each body has a "Sphere of Influence" (SOI), or a spherical region around the body where the gravity of that body dominates any spacecraft. When a spacecraft exits that body's SOI, we can then ignore that body's gravity for our calculations, and use a different body instead. So to model an Earth to Mars mission using patched conics, we'd start out calculating things with Earth as the central body. Once the spacecraft exits Earth's SOI, we then ignore Earth and assume the Sun as the central body for the majority of the trajectory. Once we approach Mars and enter Mars's SOI, we then use Mars as the central body. This makes interplanetary missions easier to model and will be more accurate than the Hohmann transfer method as we are considering the gravitational effects of Earth and Mars. The formula for the radius of the Sphere of Influence is as follows:

$$R_{SOI} = \left(\frac{\mu_p}{\mu_s}\right)^{\frac{5}{2}} \cdot R_p$$

- $R_{SOI} = SOI$ Radius
- $\mu_p =$ Gravitational Parameter of the Planet
- $\mu_s =$ Gravitational Parameter of the Planet's Sun
- $R_p =$ Radius of the Planet

In this section, we will discuss:

1. Calculating a Patched Conics Problem
2. Modeling a Patched Conics Problem

**Calculating a Patched Conics Problem**

We can split the patched conics problem into three sections:

1. Elliptical Transfer Orbit
2. Hyperbolic Planetary Departure
3. Hyperbolic Planetary Arrival

**Elliptical Transfer Orbit**
Say we wanted to calculate a transfer from Earth to Mars using patched conics. The first step is to solve the interplanetary Hohmann transfer problem. The only difference is that this time, we do not need the $\Delta v$ of the Hohmann transfer (you'll see why soon). What's most important about this part is that we get the phase angle $\Phi$, and the closest epoch at which this phase angle occurs. For a reminder on how to determine these, refer to the Interplanetary Hohmann Transfer tutorial.

**Hyperbolic Planetary Departure**

The second step is to calculate the necessary maneuver from a parking orbit around Earth to enter into the Hohmann transfer orbit to Mars. To do this, we need a hyperbolic escape trajectory that is parallel to the Earth's velocity vector. In order to escape Earth's gravitational influence, the spacecraft needs to arrive at the sphere of influence with a "hyperbolic excess velocity" $v_\infty$ greater than zero.
To enter the Hohmann transfer orbit, $v_\infty$ must be the correct amount to place the spacecraft on the desired elliptical trajectory. The required $v_\infty$ is equal to the difference between Earth's orbital velocity around the Sun and the velocity the spacecraft needs to have at the periapsis of the transfer orbit:

$$v_\infty = \left| v_{Trans_E} - v_E \right|$$

$v_\infty = \text{Hyperbolic Excess Speed of Departure Hyperbola}$
$v_{Trans_E} = \text{Velocity of Transfer Orbit at Earth (Periapsis)}$
$v_E = \text{Orbital Velocity of Earth}$

Since we are modeling Earth's orbit as perfectly circular, we can determine Earth's orbital velocity simply from the circular velocity equation:
We can determine $v_{\text{trans}}$ from the Vis-Viva equation, since the spacecraft's position with respect to the Sun at the periapsis of the transfer orbit is simply equal to Earth's orbit radius:

$$v_{\text{trans}} = \sqrt{\frac{\mu_{\text{Sun}}}{r}} \left( \frac{2}{r} - \frac{1}{a_{\text{trans}}} \right)$$

$v_{\text{trans}}$ = Velocity of Transfer Orbit at Earth
$\mu_{\text{Sun}}$ = Gravitational Parameter of the Sun
$r$ = Radius of Earth's Orbit
$a_{\text{trans}}$ = Semi – Major Axis of Transfer Orbit

Plugging these in to our original equation for $v_{\infty}$ and performing a bit of algebra yields:

$$v_{\infty} = \sqrt{\frac{\mu_{\text{Sun}}}{r}} \left( \sqrt{2 - \frac{r}{a_{\text{trans}}}} - 1 \right)$$

$v_{\infty}$ = Hyperbolic Excess Speed of Departure Hyperbola
$\mu_{\text{Sun}}$ = Gravitational Parameter of the Sun
$r$ = Radius of Earth's Orbit around Sun
$a_{\text{trans}}$ = Semi – Major Axis of Transfer Orbit

Now that we have $v_{\infty}$, we need to solve for what the velocity should be at the periapsis of that hyperbolic escape trajectory, $v_p$. But in order to do that, we must first find the 'a' of the hyperbolic orbit. In the Vis-Viva equation, our 'r' will be the radius of the SOI, and our velocity will be $v_{\infty}$. If we solve for 'a', we will have the following formula:
Now that we have the hyperbolic semi-major axis, we can solve for what our velocity at our periapsis ($v_p$) needs to be. Once again, we can use the Vis-Viva equation.

$$a_{hyp} = \left(\frac{2}{r_{EarthSOI}} - \frac{v_\infty^2}{\mu_{Earth}}\right)^{-1}$$

$a_{hyp} = Semi – Major Axis of Escape Hyperbola$

$r_{EarthSOI} = Radius of Earth's SOI$

$v_\infty = Hyperbolic Excess Speed$

$\mu_{Earth} = Standard Gravitational Parameter of Earth$

Now that we have the hyperbolic semi-major axis, we can solve for what our velocity at our periapsis ($v_p$) needs to be. Once again, we can use the Vis-Viva equation.

$$v_p = \sqrt{\mu_{Earth} \left(\frac{2}{r_p} - \frac{1}{a_{hyp}}\right)}$$

$v_p = Velocity at Periapsis of the Hyperbola$

$\mu_{Earth} = Standard Gravitational Parameter of Earth$

$r_p = Radius of the Parking Orbit$

$a_{hyp} = Semi – Major Axis of Escape Hyperbola$

Then, our $\Delta v$ is simply the difference between $v_p$ and our parking orbit velocity:

$$\Delta v_1 = \left| v_p - \sqrt{\frac{\mu_{Earth}}{r_p}} \right|$$

$\Delta v_1 = Magnitude of First Burn$

$v_p = Velocity at Periapsis of the Hyperbola$

$\mu_{Earth} = Gravitational Parameter of Earth$

$r_p = Radius of the Parking Orbit$

We have now solved for the magnitude of our maneuver to depart Earth! Still, we need to know where to burn. In the hyperbolic escape trajectory diagram, you'll notice there is an angle '$\beta$'. This is the angle between the apse line of the escape hyperbola, and the line parallel to the planet's velocity vector. The formula for this angle is as follows:
We now have all the necessary information for the Earth centered portion of our patched conics problem. After departing Earth for the Hohmann transfer orbit, we just propagate our Spacecraft as it orbits the Sun until it reaches Mars's SOI. Let's move on to the third, and final portion - the planetary arrival.

**Hyperbolic Planetary Arrival**

Once we enter the target planet's SOI, we can then assume the central body to be the target planet. Our arrival trajectory should be hyperbolic like the diagram below:

If we wish to achieve an orbit around the target planet, it is most efficient to perform an orbital insertion burn at the periapsis of the hyperbolic trajectory. This $\Delta v$ will be negative as we are slowing the Spacecraft down. How much $\Delta v$ would we need? Really, it all depends on what you would like your target orbit to be. For our example, we will simply burn into a circular orbit with a radius equal to the periapsis distance of the hyperbolic arrival trajectory. Therefore, the velocity that we need to achieve with our burn is equal to the circular velocity for an orbit around
Mars at our given radius, and the magnitude of our second burn is simply the difference between the spacecraft's velocity and the circular velocity:

\[
\Delta v_2 = \left| v_p - \frac{\mu_{\text{Mars}}}{\sqrt{r_p}} \right|
\]

\[\Delta v_2 = \text{Magnitude of First Burn} \]
\[v_p = \text{Velocity at Periapsis of the Arrival Hyperbola} \]
\[\mu_{\text{Mars}} = \text{Gravitational Parameter of Mars} \]
\[r_p = \text{Radius of the Arrival Parking Orbit/Periapsis} \]

Note that it is possible to target a specific radius for our final orbit; however, for the purposes of this guide, simply using the periapsis distance upon arrival will suffice. Mission designers could adjust the orbit after the circular insertion with further maneuvers. For more information regarding aiming for a specific periapsis radius, consult "Orbital Mechanics for Engineering Students" by Howard Curtis.

Now we have all of the necessary calculations for a patched conics transfer.

**Modeling a Patched Conics Transfer**

---

**Problem:**
Assume that Earth and Mars are in circular orbits around the Sun at 1 AU and 1.524 AU respectively. Model an interplanetary mission from Earth to Mars from a 7000 km SMA parking orbit around Earth all the way to a final orbit around Mars.

For this tutorial, we will be writing the Mission Plan almost entirely in FreeForm. We also need to create CelestialObjects that will be perfectly circular and coplanar versions of the real planets, as our equations require circular, coplanar orbits. We will refer to these CelestialObjects as "idealEarth" and "idealMars".

- Create a new Mission Plan and save it as "InterplanetaryPatchedConics.MissionPlan"
- Create a Spacecraft through the Object Browser (Don't worry about modifying any of the elements. We will be doing that later)
- Create an ImpulsiveBurn object through the Object Browser
- Double-click on "ImpulsiveBurn1" to open the editor
- Change the attitude system to "VNB"
- Click "Ok" to close the editor
- Right click on "ImpulsiveBurn1" and select "Clone"
- Rename "ImpulsiveBurn1_Copy1" to "ImpulsiveBurn2"
Creating Ideal Planets

- Drag and drop a FreeForm script editor into the Mission Sequence
- Rename this FreeForm script editor to “Create Ideal Planets”

In order to create the ideal planets and set up the ViewWindows through script, you need a more advanced understanding of FreeFlyer. As this knowledge is beyond the scope of this guide, we recommend that you copy and paste the given example code whenever we say "Copy and Paste the following code into the FreeForm script editor."

- Copy and Paste the following code into the FreeForm script editor

```csharp
// Creates the Ideal Planets
CelestialObject idealEarth;
CelestialObject idealMars;

// Configures the ideal Earth CelestialObject
idealEarth.CentralBody = "Sun";
idealEarth.Epoch = Earth.Epoch;
idealEarth.Mu = Earth.Mu;
idealEarth.Flattening = 0;

// Gets the Earth Cartesian Elements
Array RealPos = Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
Array RealVel = Earth.GetVelocityAtEpoch(Spacecraft1.Epoch);

// Assigns the Cartesian state and converts them to a Keplerian State
Array CarState = {RealPos[0], RealPos[1], RealPos[2], RealVel[0], RealVel[1],
RealVel[2]};
Array KepState = ElementConvert(1, 2, CarState, Sun.Mu);
KepState[0] = 149600000; // Reassigns SMA to approx 1 AU
KepState[1] = 0; // Reassigns Eccentricity to 0
KepState[2] = 23.5; // Reassigns Inclination to 23.5 deg
KepState[3] = 0; // Reassigns RAAN to 0 deg

// Converts the ideal Keplerian state back to Cartesian
CarState = ElementConvert(2, 1, KepState, Sun.Mu);
idealEarth.SetState(CarState);
idealEarth.Epoch = Spacecraft1.Epoch;

// Configures the ideal Mars CelestialObject
idealMars.CentralBody = "Sun";
idealMars.Epoch = Mars.Epoch;
```
idealMars.Mu = Mars.Mu;
idealMars.Flattening = 0;

// Gets the Mars Cartesian Elements
RealPos = Mars.GetPositionAtEpoch(Spacecraft1.Epoch);
RealVel = Mars.GetVelocityAtEpoch(Spacecraft1.Epoch);

// Assigns the Cartesian State and converts them to a Keplerian State
CarState = {RealPos[0], RealPos[1], RealPos[2], RealVel[0], RealVel[1], RealVel[2]};
KepState = ElementConvert(1, 2, CarState, Sun.Mu);

KepState[0] = 227920000;  // Reassigns SMA to approx 1.524 AU
KepState[1] = 0;          // Reassigns Eccentricity to 0
KepState[2] = 23.5;       // Reassigns Inclination to 23.5 deg
KepState[3] = 0;          // Reassigns RAAN to 0 deg

// Converts the ideal Keplerian State back to Cartesian
CarState = ElementConvert(2, 1, KepState, Sun.Mu);

idealMars.SetState(CarState);
idealMars.Epoch = Spacecraft1.Epoch;

Configuring the ViewWindow

- Go back to the Mission Sequence
- Drag and drop another FreeForm script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Configure ViewWindow"
- Copy and Paste the following code into the FreeForm script editor:

```plaintext
ViewWindow ViewWindow1({idealEarth, idealMars, Spacecraft1});

// Set up the default view
Viewpoint defaultView = ViewWindow1.CurrentViewpoint;

defaultView.ThreeDView.Source = Sun.ObjectId;
defaultView.ThreeDView.Target = Sun.ObjectId;
defaultView.ThreeDView.TailReference = Sun.ObjectId;
defaultView.ThreeDView.ReferenceFrame = "inertial";
defaultView.ThreeDView.Radius = 600000000;
defaultView.ThreeDView.Declination = 90;

ViewWindow1.CurrentViewpoint = defaultView;

// Set History Mode
ViewWindow1.SetHistoryMode(idealEarth.ObjectId, 1);
ViewWindow1.SetHistoryMode(idealMars.ObjectId, 1);
ViewWindow1.SetShowName(idealEarth.ObjectId, 1);
```
Setting Up the Spacecraft

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Set Up Spacecraft"

In this FreeForm script editor, we will be editing the Spacecraft elements and ForceModels through FreeForm instead of how we've typically been modifying Spacecraft. We will set the central body, the Keplerian elements, and the force model for our Spacecraft. We can do this by writing the following:

```csharp
// Set up Spacecraft in Parking orbit around Earth
Spacecraft1.CentralBody = idealEarth.DeclaredName;

// Keplerian elements for Spacecraft1
Spacecraft1.A = 7000;
Spacecraft1.E = 0;
Spacecraft1.I = 23.5;
Spacecraft1.RAAN = 0;
Spacecraft1.W = 0;
Spacecraft1.TA = 0;

// Add ideal planets to Spacecraft Force Model
(Spacecraft1.Propagator AsType RK89).ForceModel>AddForce(idealEarth);
(Spacecraft1.Propagator AsType RK89).ForceModel>AddForce(idealMars);

One more thing we need to do is turn off all forces except for our "idealEarth". During our mission, we will be turning on and off various forces depending on which SOI we are in. But for now, we just need the "idealEarth" forces on. We can do this by adjusting the "UseBodyForce" array of the force model. Each element in the array represents a Solar System body. The bodies, in order, are as follows: Mercury, Venus, Earth, The Moon, Mars, Jupiter, Saturn, Uranus, Neptune, Pluto, the Sun, and then any other custom celestial bodies. If the element has a value of 1, the gravity of the corresponding body is turned on. If it is 0, it is turned off. To manipulate this, we will write:

```csharp
// Turn off all forces except "idealEarth"
(Spacecraft1.Propagator AsType RK89).ForceModel.UseBodyForce =
{0,0,0,0,0,0,0,0,0,0,0,0,0, /*Sun*/ 0, /*idealEarth*/ 1, /*idealMars*/ 0};
```
Calculating the Phase Angle

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Calculate Phase Angle"

In this script, we need to calculate what the phase angle needs to be, and at which epoch will that occur. First, we need to calculate the period of the Hohmann transfer orbit and the angular velocities of the planets. To do this, we write:

```plaintext
Variable Pi = acos(-1);

// Set up Variables for the problem
Variable rEarth = idealEarth.GetPositionAtEpoch(Spacecraft1.Epoch).Norm;
Variable rMars = idealMars.GetPositionAtEpoch(Spacecraft1.Epoch).Norm;

// Semi-major axis of transfer orbit
Variable aTrans = (rEarth + rMars)/2;

// Period of the Hohmann transfer orbit
Variable THoh = 2*Pi*sqrt(aTrans^3/Sun.Mu);

// Angular Velocities of the planets
Variable angVelMars = (360/(2*Pi))*sqrt(Sun.Mu/rMars^3);
Variable angVelEarth = (360/(2*Pi))*sqrt(Sun.Mu/rEarth^3);
```

Now that we have the Hohman transfer orbit period and the angular velocities of the planets, we can calculate the phase angle, and the phase angular velocity. To do this, we write:

```plaintext
// Phase angle and Phase Angular Velocity
Variable phaseAngle = 180 - (1/2) * THoh * angVelMars;
Variable angVelPhase = angVelEarth - angVelMars;
```

Next, we need to calculate what the phase angle between the planets currently is, and the difference between this and the desired phase angle. Calculating this is just like what we did in the Interplanetary Hohmann Transfer tutorial. To do this, we write:

```plaintext
// Calculates the current phase angle
Variable curPhaseAngle =

// Determine the difference between the current and desired phase angle
Variable angleDiff = curPhaseAngle - phaseAngle;
If (angleDiff < 0) then;
    angleDiff = 360 - abs(angleDiff); // Ensures our phasing time will be in the future
End;
```
Now we need to calculate how long it will be until the next opportunity, and the epoch of the next opportunity. To do this, we write:

```csharp
// Calculates the epoch where the planets have the proper phase angle
Variable timeToPhase = angleDiff/angVelPhase;
TimeSpan phaseEpoch = Spacecraft1.Epoch + TimeSpan.FromSeconds(timeToPhase);
```

### Stepping to Planetary Phasing

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Step to Planetary Phasing"

In this FreeForm script editor, we will propagate the solar system to the "phaseEpoch" we calculated in the previous FreeForm script editor, and change the viewpoint to an Earth-centered view. To do this, we write:

```csharp
// Set the Propagator Step Size to 1 day
Spacecraft1.Propagator.StepSize = TIMESPA(1 days);

WhileStepping Spacecraft1 to (Spacecraft1.Epoch == phaseEpoch);
    Update ViewWindow1;
End;

// Change Viewpoint to Earth Centered
ViewWindow1.SetHistoryMode(Spacecraft1.ObjectId, 1);
```

### Calculating the Departure Maneuver

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Calculate Departure Maneuver"

In this FreeForm script, we will be calculating $v_o$ and $v_p$ to find the $\Delta v$ for the escape trajectory. First, we need to calculate the radius of the sphere of influence for each planet. To do this, we write:

```csharp
// Calculates the Sphere of Influence for the problem
Variable EarthSOI = (idealEarth.Mu/Sun.Mu)^(2/5) * rEarth;
Variable MarsSOI = (idealMars.Mu/Sun.Mu)^(2/5) * rMars;
```

Next, we need to perform the calculations to find $v_o$, $v_p$, and $\Delta v$, using the equations we derived earlier.
Calculating Beta Angle and Maneuver Time

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Calculate Beta Angle and Maneuver Time"

In this script, we will be calculating the $\beta$ angle so we can figure out when is the appropriate time for our Spacecraft to maneuver. First, let's calculate $\beta$ using the formula given earlier. To do this, we write:

```plaintext
// Beta Angle
Variable beta = deg(acos(1/(1 + ((Spacecraft1.A * vInfEarth^2)/(idealEarth.Mu)))));
```

If we look at the diagram in the "Planetary Departure" section from earlier, $\beta$ is actually the angle away from Earth's negative velocity vector. So when our Spacecraft is $(180 - \beta)$ degrees away from Earth's positive velocity vector, we need to maneuver the Spacecraft. First, let's calculate the current angle between the Spacecraft's position and the Earth's velocity vector.

One thing we need to remember is that if we're starting in a prograde parking orbit (for this example, we are), we need to maneuver on the dark side of Earth. Another thing to remember is that there are two points in our orbit where we are $(180 - \beta)$ degrees away from Earth's positive velocity vector. Because of this, we could be at the correct angle but the wrong side of the planet. To check which side we're on, we can check the z component of the cross product vector. If it is negative, we are on the sunny side of Earth and we need to adjust our angle. If it is positive, we are on the dark side of Earth and we don't need to worry about any corrections. To do this, we write:

```plaintext
// Current angle of spacecraft position to the idealEarth Velocity Vector

If(Spacecraft1.Position.CrossProduct(idealEarth.GetVelocityAtEpoch(Spacecraft1.Epoch))[2] < 0) then;
    currentAngle = 360 - currentAngle;
End;
```
Next, we need to calculate how many degrees we need to travel before performing the maneuver. We do this by subtracting 180 degrees from our current angle and adding $\beta$. However, if our calculations concluded that we were on the sunny side of Earth, we need to add 360 degrees to our current angle. To do this, we write:

```plaintext
Variable degreesToTravel = currentAngle - 180 + beta;
If(degreesToTravel < 0) then;
    degreesToTravel += 360;
End;
```

Now we need to calculate the angular velocity of the spacecraft’s orbit, and the epoch at which we need to perform the maneuver. To do this, we write:

```plaintext
// Angular Velocity of the spacecraft
Variable angVelSC = (360/(2*Pi)) * sqrt(idealEarth.Mu/Spacecraft1.A^3);

// Calculates when the maneuver should be performed
Variable timeToManeuver = degreesToTravel / angVelSC;
TimeSpan maneuverEpoch = Spacecraft1.Epoch + TimeSpan.FromSeconds(timeToManeuver);
```

**First Maneuver and Stepping to the SOI**

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script editor to "Perform First Maneuver, Step to Earth SOI"

In this script, we need to step to the maneuver epoch (which corresponds to the correct beta angle), maneuver, and then step to the edge of Earth’s SOI. First, we need to change the propagator size from 1 day to approximately 600 seconds, then step to the maneuver epoch. We need to make our step size shorter because we are going from observing a heliocentric orbit to observing an Earth centered orbit. To do this, we write:

```plaintext
// Changes the Step Size to 600s
Spacecraft1.Propagator.StepSize = TIMESPAN(600 seconds);

WhileStepping Spacecraft1 to (Spacecraft1.Epoch == maneuverEpoch);
    Update ViewWindow1;
End;
```

Now that we are at the proper maneuvering time, we need to maneuver the Spacecraft. To visualize this better, we will also change the tail color. To do this, we write:

```plaintext
Maneuver Spacecraft1 using ImpulsiveBurn1;
```
Next, we need to propagate the Spacecraft to the edge of Earth's SOI. Once we do that, we need to change the Spacecraft's central body to the Sun, turn off the gravity of "idealEarth", and turn on the gravity of the Sun. To do this, we write:

```plaintext
// Steps the spacecraft to the edge of the SOI
WhileStepping Spacecraft1 to (Spacecraft1.Radius >= EarthSOI);
  Update ViewWindow1;
End;

Spacecraft1.CentralBody = "Sun";

// Turn off all forces except "Sun"
(Spacecraft1.Propagator AsType RK89).ForceModel.UseBodyForce =
{0,0,0,0,0,0,0,0,0,0, /*Sun*/ 1, /*idealEarth*/ 0, /*idealMars*/ 0};
```

Stepping to the Mars SOI

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Propagate to Mars SOI"

In this FreeForm script editor, we will change the view to be Sun centered, change the step size to 1 day, and then step until we're inside the Mars SOI. To do this, we write:

```plaintext
// Change Viewpoint to Sun Centered

Spacecraft1.Propagator.StepSize = TIMESPAN(1 days);

// Change CentralBody to idealMars to use the Radius property to determine SOI entrance
Spacecraft1.CentralBody = idealMars.DeclaredName;

WhileStepping Spacecraft1 to (Spacecraft1.Radius <= MarsSOI);
  Update ViewWindow1;
End;
```

At this point, we are inside the SOI of "idealMars." We now need to turn on the gravity of "idealMars." To do this, we write:

```plaintext
// Turn off all forces except "idealMars"
(Spacecraft1.Propagator AsType RK89).ForceModel.UseBodyForce =
{0,0,0,0,0,0,0,0,0,0, /*Sun*/ 0, /*idealEarth*/ 0, /*idealMars*/ 1};
```
Performing the Intercept Maneuver

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Calculate and Perform Intercept Maneuver"

In this FreeForm script editor, we will be changing the viewpoint, changing the step size, propagating to the hyperbolic periapsis, and then performing an orbital insertion burn. First, let's change the step size, viewpoint, and propagate to the orbit periapsis. To do this, we write:

```plaintext
// Change the step size to 6 hours
Spacecraft1.Propagator.StepSize = TIMESPAN(6 hours);

// Change the viewpoint to Mars centered

WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
  Update ViewWindow1;
End;
```

Now that we are at the periapsis of the approach hyperbola, we need to perform the orbital insertion burn to enter our desired circular orbit. We will use the formula determined earlier for the second burn to calculate our ∆v. To do this, we write:

```plaintext
// Required burn to enter a circular orbit
Variable dv2 = abs(Spacecraft1.VMag - sqrt(idealMars.Mu/Spacecraft1.Radius));
ImpulsiveBurn2.BurnDirection[0] = -dv2;
```

Notice that we are using the VMag property here to access the Spacecraft's velocity at periapsis. Also notice that we assign the burn value to be negative ∆v, since we need to slow down (burn opposite to the current velocity direction) to achieve our circular orbit.

Now that we have our maneuver set up, the next step is to perform the maneuver, and then propagate for 100 days to see if we've achieved an orbit. Also, let's report the amount of ∆v needed for this mission. To do this, we write:

```plaintext
// Perform the orbital insertion burn
Spacecraft1.Color = ColorTools.Lime;
Maneuver Spacecraft1 using ImpulsiveBurn2;

WhileStepping Spacecraft1 to (Spacecraft1.ElapsedTime == TIMESPAN(100 days));
  Update ViewWindow1;
End;
```
// Report Delta V Values
Report dv1, dv2, (dv1 + dv2);

Your Mission Sequence should look something like this:

<table>
<thead>
<tr>
<th>#</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FreeForm: Create Ideal Planets</td>
</tr>
<tr>
<td>2</td>
<td>FreeForm: Configure ViewWindow</td>
</tr>
<tr>
<td>3</td>
<td>FreeForm: Set Up Spacecraft</td>
</tr>
<tr>
<td>4</td>
<td>FreeForm: Calculate Phase Angle</td>
</tr>
<tr>
<td>5</td>
<td>FreeForm: Step to Planetary Phasing</td>
</tr>
<tr>
<td>6</td>
<td>FreeForm: Calculate Departure Maneuver</td>
</tr>
<tr>
<td>7</td>
<td>FreeForm: Calculate Beta Angle and Maneuver Time</td>
</tr>
<tr>
<td>8</td>
<td>FreeForm: Perform First Maneuver, Step to Earth SOI</td>
</tr>
<tr>
<td>9</td>
<td>FreeForm: Propagate to Mars SOI</td>
</tr>
<tr>
<td>10</td>
<td>FreeForm: Calculate and Perform Intercept Maneuver</td>
</tr>
</tbody>
</table>

Our mission is now complete!

NOTE: It is important that you do not manually change the view during the mission. This will override the automatic view changing written in our script.

Save and run your Mission Plan, and try to answer these questions:

- How much ∆v was required for the first maneuver? The second?

- How much ∆v was used total?

- What was the periapsis distance (from the Spacecraft to the center of Mars) of the arrival hyperbolic trajectory?

See Also

- Interplanetary Topics
- Next Topic: Gravity Assist
- Previous Topic: Interplanetary Hohmann Transfer
5.3 - Gravity Assist

In the Patched Conics Transfer tutorial, we had a spacecraft enter Mars's SOI and perform an orbital insertion burn. But what would happen if we let the spacecraft continue on its hyperbolic trajectory? Doing this is called a "Gravity Assist" or "Planetary Flyby."

During a gravity assist, a spacecraft will fly right near another celestial body (typically a planet) and have their flight path be redirected. Also, the spacecraft will leave the SOI with a different velocity than what it entered with. But where does this "extra energy" come from?

When a spacecraft gets really close to a planet during a hyperbolic trajectory, the planet pulls the spacecraft along and swings it around either speeding it up or slowing it down. When the spacecraft enters the SOI the velocity becomes the spacecraft's velocity vector in reference to the Sun subtracted by the planet's velocity vector. After the spacecraft's flyby, once it exits the SOI, you then add the planet's velocity vector to its velocity in reference to the planet to get the velocity in reference to the Sun. Because the spacecraft exits the SOI in a different direction, there is a change in the magnitude of velocity.

As it turns out, this "free velocity" isn't so free. According to the conservation of energy, the kinetic energy gained by the spacecraft had to be lost somewhere. In actuality, this gravity assist is a conservation of momentum problem. Momentum gets transferred from the planet to the spacecraft. Because the planet's mass is far larger than the spacecraft, the change in velocity is extremely large for the spacecraft, but miniscule for the planet. So, every time a planet is used for a gravity assist, a minute amount of planetary velocity is lost.

In this section, we will discuss:

1. Calculating a Gravity Assist
2. Modeling a Gravity Assist

Calculating a Gravity Assist

Calculating a gravity assist involves understanding hyperbolic orbits and vector math.

The entering heliocentric velocity vector is actually the sum of the entering hyperbolic excess velocity vector and the planet's velocity vector.

\[
\vec{V}_1 = \vec{V}_P + \vec{v}_{\infty 1}
\]

\[
\vec{V}_1 = \text{Entering Heliocentric Velocity Vector of the Spacecraft}
\vec{V}_P = \text{Velocity Vector of the Planet}
\vec{v}_{\infty 1} = \text{Entering Hyperbolic Excess Velocity Vector of the Spacecraft}
\]

The exiting heliocentric velocity vector is the sum of the exiting hyperbolic excess velocity vector and the planet's velocity vector.
The $\Delta v$ from the gravity assist is simply this:

$$\vec{V}_2 = \vec{V}_p + \vec{v}_\infty 2$$

If we substitute the definitions for $\vec{V}_2$ and $\vec{V}_1$, then we get the following:

$$\Delta \vec{V} = \vec{V}_2 - \vec{V}_1$$

The total $\Delta v$ from a gravity assist is the difference between the exiting hyperbolic excess velocity vector, and the entering hyperbolic excess velocity vector. You may ask: "If a hyperbola has the same speeds at the entering and exiting points of the SOI, how is there any $\Delta v$ at all?" Yes, it is true that these two vectors have the same magnitude, but not the same direction. So, the $\Delta v$ gained from a gravity assist really depends on the change of direction. If we enter and exit the SOI travelling the same direction, there really isn't any $\Delta v$ gained. However, if we fly closer to the planet, the gravity will bend our path more significantly, increasing our $\Delta v$.

**Modeling a Gravity Assist**

Let's attempt to model a gravity assist in FreeFlyer.

**Problem:**
Two similar spacecraft enter Mars's sphere of influence with the same velocity. One will pass in front of Mars, while the other passes behind Mars. Which spacecraft will end up having the larger orbit around the sun?

<table>
<thead>
<tr>
<th></th>
<th>FrontSideSC</th>
<th>BackSideSC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Central Body</td>
<td>Mars</td>
<td>Mars</td>
</tr>
<tr>
<td>$a$</td>
<td>-6088.425 km</td>
<td>-6088.425 km</td>
</tr>
<tr>
<td>$e$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$i$</td>
<td>180</td>
<td>0</td>
</tr>
<tr>
<td>$\Omega$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\omega$</td>
<td>195</td>
<td>45</td>
</tr>
<tr>
<td>$TA$</td>
<td>240.976</td>
<td>240.976</td>
</tr>
<tr>
<td>Tail Color</td>
<td>Red</td>
<td>Green</td>
</tr>
</tbody>
</table>

- Create a new Mission Plan and save it as "GravityAssist.MissionPlan"

**Adding in Spacecraft**

- Add two Spacecraft through the object browser
- Rename them to the names at the top of the table
- Configure each spacecraft to the elements in the table provided below the problem statement
- In each Spacecraft's force model, check the "Mars" box, and uncheck the "Earth" and "Moon" boxes

- Create a "Vector" object in the Object Browser (we will use this to visualize the direction of the planet's velocity vector)
  - Right-click the Object Browser
  - Add \rightarrow Variables \rightarrow Vector

**Adding the ViewWindow**

- Create a ViewWindow through the Object Browser
- Double-click "ViewWindow1" to open the editor
- Check "FrontSideSC", "BackSideSC", and "Vector1" (you will need to expand the list of "Vector" objects in the "Available Objects" to check "Vector1")
- Check "Show Name" for "FrontSideSC" and "BackSideSC"
- Change the History Mode to "Unlimited" for "FrontSideSC" and "BackSideSC"
- Expand the CelestialObject group and select "Mars"
- Check "Show History" for "Mars", and change the History Mode to "Unlimited"
- Go into "Viewpoints" on the left-hand side of the editor
- Change the reference frame to "Inertial"
- Change the source, target, and tail reference to "Mars"
- Change the radius to 300,000 km
- Click on the "Create" button to create a new viewpoint
- Change the name of this new viewpoint to "SolarView"
- Change the title of this viewpoint to "Solar View"
- Change the reference frame to "Inertial"
- Change the source, target, and tail reference to "Sun"
- Change the radius to 500,000,000 km
- Click "Ok" to close the editor

**Building the Mission Sequence**

- Drag and drop a FreeForm script editor onto the Mission Sequence
- Rename the script to "Set Up Vector"

Whenever you create a Vector object, you must configure it through a FreeForm script editor. In this script, we will set the vector epoch, tell it to draw itself as an arrow, build the vector, and make it visible. To do this, we write:

```plaintext
// Sets up the vector to display Mars's velocity vector
Vector1.Epoch = FrontSideSC.Epoch;
Vector1.DrawMethod = 1;
Vector1.BuildVector(7, Mars);
Vector1.Active = 1;
```

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
• Rename this FreeForm script to "Record Initial Data"

In this script, we will save the heliocentric velocity and the hyperbolic excess velocity vectors. To save the heliocentric vector, we must temporarily reassign both Spacecraft's central bodies to the Sun. Then we can save the vector to an array. To do this, we write:

```plaintext
// Changes the central body to Sun to get the heliocentric velocity
FrontSideSC.CentralBody = "Sun";
BackSideSC.CentralBody = "Sun";

Array frontV1 = FrontSideSC.Velocity;
Array backV1 = BackSideSC.Velocity;
```

Next, we need to reassign the central body to Mars for both Spacecraft. Then we can record the hyperbolic excess speed at entrance to the SOI. To do this, we write:

```plaintext
// Changes the central body back to Mars to get the excess hyperbolic speed
FrontSideSC.CentralBody = "Mars";
BackSideSC.CentralBody = "Mars";

Array frontVInf1 = FrontSideSC.Velocity;
Array backVInf1 = BackSideSC.Velocity;
```

• Go back to the Mission Sequence
• Drag and drop another Freeform script editor at the bottom of the Mission Sequence
• Rename this FreeForm script to "Propagate Through SOI"

In this script, we will step both Spacecraft until they fly past Mars and reach the edge of the sphere of influence, and record the new velocity data. First, let's calculate the SOI and step to the edge of it. To do this, we write:

```plaintext
// Sphere of Influence of Mars

While(FrontSideSC.Radius < MarsSOI);
    Step FrontSideSC;
    Step BackSideSC to (BackSideSC.Epoch == FrontSideSC.Epoch);
    Update ViewWindow1;
End;
```

Next, we need to record the velocity data so we can calculate the ∆v values for each Spacecraft. Also, we can measure the change in direction of each Spacecraft's heliocentric velocity vector. When we record this data, we can record the hyperbolic excess speed, change the central body to the Sun, and then record the heliocentric velocity vectors. To do this, we write:
The final thing we need to do in this script is calculating and reporting the numbers.

If we want to find the magnitude of the $\Delta v$, we need to subtract the hyperbolic excess speed of the Spacecraft at entrance from the hyperbolic excess speed at exit. One important thing to note is that we need the vector difference between the two before we take the norm. If we subtracted the norm of both velocities, we would get a calculation of '0'.

For the change in direction, we need to take the vertex angles of the heliocentric velocities at entrance and exit of the SOI. If we took the vertex angle between the hyperbolic excess velocity vectors, our change in direction would not be accurate.

One last thing we need to do after reporting the calculations - we need to pause the scenario so that the user can change their 3D view to the "Solar View" that we created earlier. To do this, we write:

```plaintext
// Calculations
Variable dVFront = (frontVInf2 - frontVInf1).Norm;
Variable dVBack = (backVInf2 - backVInf1).Norm;

Variable dThetaFront = frontV2.VertexAngle(frontV1);
Variable dThetaBack = backV2.VertexAngle(backV1);

Report dVFront, dThetaFront, dVBack, dThetaBack;
Pause;
```

- Go back to the Mission Sequence
- Drag and drop another Freeform script editor at the bottom of the Mission Sequence
- Rename this FreeForm script to "Propagate Solar System"

In this last script, we are going to change the propagator step size to 1 day (86,400 s), then step the solar system for 1000 days to visualize the new orbits. To do this, we write:

```plaintext
// Change the step size to 1 day
FrontSideSC.Propagator.StepSize = TIMESPAN(1 days);
BackSideSC.Propagator.StepSize = TIMESPAN(1 days);

While (FrontSideSC.ElapsedTime < TIMESPAN(1000 days));
```
Step FrontSideSC;
Step BackSideSC to (BackSideSC.Epoch == FrontSideSC.Epoch);
Update ViewWindow1;
End;

Your Mission Sequence should look something like this:

<table>
<thead>
<tr>
<th>Mission Sequence</th>
<th>1 - Set Up Vector</th>
<th>2 - Record Initial Data</th>
<th>3 - Propagate Through</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FreeForm: Set Up Vector</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FreeForm: Record Initial Data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FreeForm: Propagate Through SOI</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FreeForm: Propagate Solar System</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Mission Sequence Example

Save and run your Mission Plan. Once the window reports your values, make sure to change your 3D View to the "Solar View". You can do this by selecting the "MissionView" window, and changing the "View Point" dropdown to "Solar View". Once you do that, you can click play on the navigation bar.

Once you've done that, try and answer these questions:

*Which Spacecraft had the bigger heliocentric orbit?*

*In what direction in reference to Mars's velocity vector did that Spacecraft exit the SOI?*

*How does the magnitude of each Spacecraft's $\Delta v$ compare? Why are their orbits so drastically different?*

See Also

- Interplanetary Topics
- Next Topic: The B-Plane
- Previous Topic: Patched Conics Transfer
5.4 - The B-Plane

When making planetary flybys or rendezvous, many missions will use what is called "B-Plane Targeting." This allows the spacecraft to have a specific hyperbolic trajectory past the planet. This is useful when attempting to fly past a specific geographical location on a planet, or making the capture orbit a specific inclination.

But what exactly is the B-plane? The B-plane is a plane that is orthogonal to the hyperbolic trajectory plane and the initial hyperbolic excess velocity vector. This is typically described with a vector, B, and an angle, Θ, to help define the hyperbolic trajectory in 3D space. Also, there are three orthogonal unit vectors R, S, and T, to help calculate these values.

In this section, we will discuss:

1. Calculating the B-Plane
2. Modeling the B-Plane

Calculating the B-Plane

If we have the velocity vector and the radius vector at the orbit periapsis, we can solve for the B vector, and the angle Θ.

The B vector points to where the asymptote of the hyperbolic trajectory pierces the B-plane. It may be easier to visualize this as where the spacecraft would pierce the B-plane if it ignored the planet's gravity. The angle 'Θ' is the angle between the 'B' vector, and the 'T' unit vector. The 'T' unit vector is, along with the R unit vector, an axis of the B-Plane.

To find B and Θ, we must find a few unit vectors to make our calculations easier.
The first unit vector is 'S'. This is the direction of the hyperbolic excess velocity at the entry of the sphere of influence.

The second unit vector is 'T'. This is the orthogonal axis to the 'S' unit vector, and the normal of the planet. It also typically lies on the ecliptic plane of the solar system.

The third unit vector is 'R'. This is simply the cross product of 'S' and 'T'.

But if all we’re given is an initial state vector, how can we calculate these things? The first step is to calculate the angular momentum vector and the eccentricity vector.

\[ \hat{h} = \frac{\vec{r} \times \vec{v}}{|\vec{r} \times \vec{v}|} \]

\[ \hat{h} = \text{Angular Momentum Unit Vector} \]
\[ \vec{r} = \text{Radius Vector} \]
\[ \vec{v} = \text{Velocity Vector} \]

\[ \vec{e} = \frac{1}{\mu} \left( |\vec{v}|^2 \vec{r} - (\vec{r} \cdot \vec{v}) \vec{v} \right) - \frac{\vec{r}}{|\vec{r}|} \]

\[ \vec{e} = \text{Eccentricity Vector} \]
\[ \mu = \text{Gravitational Parameter of the Central Body} \]

Next, we need to calculate our semi-major axis, and our beta angle. This is simply:

\[ \beta = \cos^{-1} \left( \frac{1}{e} \right) \]

Next, we need to calculate the direction of the 'S' unit vector. Remember that the 'S' unit vector describes the direction of the entering asymptote. To derive the math for it, let's look at a plot of the hyperbolic plane.
We know that our 'S' unit vector is in the direction of the entering asymptote. We also know the half-angle, 'β' of the asymptotes. With this, we can break down the 'S' unit vector into two components - one in the 'e' unit vector direction, and one in the 'h x e' unit vector direction (the h vector is coming out of the page in this diagram). Doing so, we get this formula for the 'S' unit vector.

\[ \hat{S} = \cos(β) \hat{e} + \sin(β) (\hat{h} \times \hat{e}) \]

We still have two more unit vectors to solve for. Next, let's try to find the 'T' unit vector. If we define the 'N' unit vector as the direction of the z-axis of the reference frame, 'N' will be \(<0, 0, 1>\). Then, we can find the 'T' unit vector.

\[ \hat{N} = <0, 0, 1> \]

\[ \hat{T} = \frac{\hat{S} \times \hat{N}}{|\hat{S} \times \hat{N}|} \]

To complete the 'S' and 'T' unit vectors, we must now find the 'R' unit vector. This one is the cross product of the 'S' and 'T' unit vector.
Now that we have our unit vectors, let's try to find what we've been looking all along: the 'B' vector and 'Θ'. To find the magnitude of the 'B' vector, we need the semi-major axis and the eccentricity of the hyperbolic trajectory. The magnitude of the 'B' vector can be found in the formula below. Once we have that, we need the direction of the 'B' vector. This should be the cross product of the 'S' unit vector and the 'h' unit vector. Now that we have the magnitude and the unit vector, if we multiply them together, we will have our 'B' vector.

\[ \hat{R} = \hat{S} \times \hat{T} \]

\[ B = |a| \sqrt{e^2 - 1} \]

\[ \hat{B} = \hat{S} \times \hat{h} \]

\[ \vec{B} = B \hat{B} \]

Our angle 'Θ' is simply the vertex angle of the 'B' vector and the 'T' unit vector. To calculate 'Θ', we use the following formula:

\[ \theta = \cos^{-1} \left( \frac{\vec{B} \cdot \hat{T}}{|\vec{B}| \cdot |\hat{T}|} \right) \]

**Modeling the B-Plane**

**Problem:**
A spacecraft has a state vector of <-299761, -440886, -308712> km and <1.343, 1.899, 1.329> km/s around Mars in the MJ2000 reference frame. Calculate the 'B' vector and 'Θ' for this spacecraft.

Luckily, FreeFlyer already has tools built in to calculate these things for us. However for this Mission Plan, we will not only calculate these parameters, but visualize them as well.

- Create a New Mission Plan and save it as "BPlaneVisualization.MissionPlan"

**Adding a Spacecraft**

- Add a Spacecraft through the Object Browser
- Change the Spacecraft central body to "Mars"
- Give the Spacecraft the Cartesian elements given in the problem statement.
- In the Spacecraft's force model, make sure that Mars and the Sun are the only two bodies checked
Adding a Vector

- Create a Vector in the Object Browser
  - Right-click the Object Browser
  - Add → Variables → Vector
- Rename this Vector to "B" by right-clicking the Vector and selecting "Rename"
- Create another Vector and name it "S"
- Create another Vector and name it "T"
- Create another Vector and name it "R"

Adding the ViewWindow

- Create a ViewWindow through the Object Browser
- Double-click "ViewWindow1" to open the editor
- Check "Spacecraft1" and all four Vector objects
- Check "Show Name" for all the objects
- Change the history mode to "Unlimited" for "Spacecraft1"
- Go into "Viewpoints" on the left-hand side of the editor
- Change the reference frame to "Inertial"
- Change the source, target, and tail reference to "Mars"
- Click "Ok" to close the editor

Building the Mission Sequence

- Drag and drop a FreeForm script editor onto the Mission Sequence
- Rename the script to "Calculate Vectors"

In this script, we will be using the formulas from Calculating the B-Plane to calculate the values of the Vector objects that we've created. To start off, we need to record the Spacecraft's state vector. To do this, we write:

```csharp
// Initial state vector
Array rVect = Spacecraft1.Position;
Array vVect = Spacecraft1.Velocity;
```

Next, we can start calculating the 'h' unit vector and the 'e' vector. One important thing to note is that ".Norm" will return the magnitude of the vector array and ".Normalized" will return the unit vector. When we calculate the 'h' unit vector, what we are asking for is the unit vector of the cross product of 'r' and 'v'. To calculate these values, we write:

```csharp
// Calculations
Array hUnit = rVect.CrossProduct(vVect).Normalized;
```
Next, we can calculate the angle 'β' and all of the other unit vectors. To do this, we write:

```
Variable beta = acos(1/eVect.Norm);

Array sUnit = cos(beta) * eVect.Normalized + sin(beta) *
    (hUnit.CrossProduct(eVect.Normalized));
Array nUnit = {0,0,1};
Array tUnit = sUnit.CrossProduct(nUnit).Normalized;
Array rUnit = sUnit.CrossProduct(tUnit);
```

Last, but not least, we need to calculate the 'B' vector. To do this, we need to first calculate the 'B' unit vector. Then we multiply the 'B' unit vector by the magnitude, which we can pull directly from Spacecraft1 to get the 'B' vector. Do to this, we write:

```
Array bUnit = sUnit.CrossProduct(hUnit);
Variable bMag = Spacecraft1.BPlaneBMag(Mars);
Array bVect = bMag * bUnit;
Variable bTheta = Spacecraft1.BPlaneTheta(Mars);
```

Now that we have calculated all of the vectors, let's work on setting up the vectors

- Drag and drop a FreeForm script editor at the bottom of the Mission Sequence
- Rename the script to "Initialize Vectors"

In this script, we will be setting up our vectors. To do this, we will be matching the epochs, assigning the elements, building the vectors, changing the draw method, setting the colors, making them active, and offsetting the vectors so their origins lie in the center of Mars. To do this, we write:

```
// Set up the vector epochs
B.Epoch = Spacecraft1.Epoch;
S.Epoch = Spacecraft1.Epoch;
T.Epoch = Spacecraft1.Epoch;
R.Epoch = Spacecraft1.Epoch;

// Set up the vector elements
B.Element = bVect;
S.Element = sUnit;
T.Element = tUnit;
R.Element = rUnit;

// Build all the vectors
B.BuildVector(0);
S.BuildVector(0);
T.BuildVector(0);
R.BuildVector(0);

// Set to draw as arrows
B.DrawMethod = 1;
S.DrawMethod = 1;
T.DrawMethod = 1;
R.DrawMethod = 1;

// Set the vector colors
B.Color = ColorTools.Yellow;
S.Color = ColorTools.Red;
T.Color = ColorTools.Cyan;
R.Color = ColorTools.Green;

// Set all the vectors as visible
B.Active = 1;
S.Active = 1;
T.Active = 1;
R.Active = 1;

// Set the position of the vectors
B.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
S.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
T.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
R.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);

Let's move back to the Mission Sequence

- Drag and drop a while loop at the bottom of the Mission Sequence
- Change the while loop argument to "(Spacecraft1.ElapsedTime < TIMESSPAN(6 days))"
- Drag and drop a FreeForm script editor inside the while loop
- Rename the script to "Step, Move, and Update"

In this script, we will step the Spacecraft, move the vectors so their origins rest at the center of Mars, and then update the ViewWindow. To do this, we write:

// Steps the spacecraft forward
Step Spacecraft1;

// Moves the position of the vectors
B.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
S.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
T.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
R.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
Lastly, let's report the 'B' vector, and angle.

- Drag and drop a FreeForm script editor at the bottom of the Mission Sequence
- Rename the script to "Report Calculations"

In this script, we will report the 'B' vector, and the angle. To do this, we write:

```plaintext
// Updates the ViewWindow
Update ViewWindow1;
```

```plaintext
// Updates the ViewWindow
Update ViewWindow1;
```

```plaintext
// Reports the B vector and angle
Report bVect, bTheta;
```

Your Mission Sequence should look something like this:

```
// Calculate B vector and angle
Report bVect, bTheta;
```

Save and run your Mission Plan and try to answer the following questions:

What were the Cartesian elements of the 'B' vector?

What was the angle between the 'B' vector and the 'T' unit vector?

Try calculating the 'B' magnitude by hand. How does it compare to FreeFlyer's output?

We explained earlier that the 'B' vector points to where the Spacecraft would pierce the B-plane if it ignored the planet's gravity. Set 'B.DrawMethod' equal to 0 in the "Initialize Vectors" FreeForm and then turn off Mars's gravity in the Spacecraft Force Model. Run the Mission Plan again to check this claim.

See Also

- Interplanetary Topics
- Previous Topic: Gravity Assist
CHAPTER 6

Targeting Tutorial
In this chapter, we will discuss the mathematical process of targeting. We will then use Targeting loops in FreeFlyer to determine the necessary magnitude and orientation of impulsive burns to achieve various goals.

This chapter will cover the following topics:

1. Targeting Concepts
2. Inclination Change
3. B-plane Targeting
6.1 - Targeting Concepts

Targeting is a useful tool for any time that you want to achieve a specific goal, but are uncertain of exactly how to do it. For example, you might use targeting to determine the magnitude of a Δv maneuver to change some orbital parameter by a specified amount, or to determine a point in an orbit from which to depart to rendezvous with a satellite. FreeFlyer uses the differential corrector method or "shooting method" to achieve this.

Targeting using the differential corrector method works by taking some parameter which affects the result of some function, and varying the parameter until the function outputs a specific desired result. You begin with some "seed" value for the parameter, calculate the result for that seed, and then apply some perturbation to the seed to change the result. So if the parameter were increased, and the result was farther from the goal than the initial result, the targeting function would decrease the parameter on the next iteration. This process continues until the desired result is achieved within some tolerance. We can call the changing parameter a "Vary" and the result an "Achieve". An illustration of this process is shown below.

![One-Dimensional Case of the Shooting Method](image)

**Final Goal: To Achieve 5**
- Initial Guess = 0, Achieve = 7
- Perturbation = 0.1, Achieve = 9
- Goal is 5, so next guess is -0.1

For more information on this process, see the "Targeting" page of the Maneuvering and Targeting Guide in the FreeFlyer Help File.

The mechanism for the targeting process in FreeFlyer is called a Targeting loop, which consists of the following components:
- **Target** - command which begins the Targeting loop
- **Iterate** - command which specifies the objects that are restored to their initial states after each iteration of the loop
- **Vary** - command which specifies the parameters that are adjusted with each iteration
  - Requires a seed value and a perturbation value
- **Achieve** - command which defines the goals of the targeting sequence
  - Requires a goal value and a tolerance value
- **End** - command to close the targeting loop

An unlimited number of Iterate, Vary, and Achieve commands can be present in any single Targeting loop, although for best results, we recommend having an equal number of Vary and Achieve commands. In the Default Targeting mode, which is what we will discuss here, you cannot have a Vary statement after an Achieve statement, and Targeting loops cannot exist inside If statements, For loops, While loops, etc. If this functionality is essential to your Mission Plan, you would need to use the Runtime Setup Targeting feature, which is out of scope for this tutorial. If you are interested, more information can be found in the FreeFlyer Help File.

If you need to access specific properties of your Targeting loop, you can use a DifferentialCorrector object. A DifferentialCorrector object allows you to set the maximum number of iterations for a Targeting loop (the default value is 35) and provides access to the status of the Targeting loop during execution. We will not use a DifferentialCorrector object in this tutorial, but more information can be found in the FreeFlyer Help File if you are interested.

**See Also**

- [Targeting Tutorial](#)
- [Next Topic: Inclination Change](#)
6.2 - Inclination Change

In the final section of the Maneuvering tutorial, we discussed the math behind a plane change (or inclination change) maneuver. We discovered that we would need to execute a burn that was mostly normal to our velocity, but with a small component in the negative velocity direction (see the Plane Change Maneuver tutorial to review). We determined the following formulas for the ∆v components:

\[ \theta = \text{Inclination} \]
\[ \Delta v_n = \Delta v \text{ Normal Component} \]
\[ \Delta v_{nv} = \Delta v \text{ Negative Velocity Component} \]
\[ v_{orig} = \text{Original Velocity} \]
\[ v_{res} = \text{Resultant Velocity} \]
\[ v_{orig} = v_{res} \]

\[ \Delta v_n = v_{orig} \cdot \sin(\theta) \]
\[ \Delta v_{nv} = v_{orig} \cdot (1 - \cos(\theta)) \]

Where \( \Delta v_n \) is the normal component and \( \Delta v_{nv} \) is the negative velocity component. In the previous plane change example, we used FreeFlyer to calculate the ∆v components directly using these equations. This time, we will execute a similar mission plan, but use a Targeting loop to determine the ∆v components (instead of the equations above). Then, we can compare the results that FreeFlyer achieves to our analytic calculations.

Use Targeting to Model an Inclination Change

**Problem:**
We have a spacecraft in a circular, equatorial orbit with a SMA of 10000 km. How much ∆v would we need to change the inclination to 35º without changing the shape of the orbit?

- Create a new Mission Plan and save it as "TargetingInclinationChange.MissionPlan"

**Adding a Spacecraft, ImpulsiveBurn, and Variable**

- Create a new Spacecraft with the following Keplerian orbital elements:
  - A: 10000 km
  - E: 0
  - I: 0 deg
  - RAAN: 0 deg
  - W: 0 deg
• TA: 0 deg
• Create a new ImpulsiveBurn object
• Open the ImpulsiveBurn editor
• Change the attitude system to "VNB"
• Click "Ok" to close the editor
• Create a new Variable object
• Open the Variable editor and rename it "initialVelocity"
• Click "Ok" to close the editor

Adding the ViewWindow

• Create a new ViewWindow object
• Open the ViewWindow editor
• Check "Spacecraft1" in the "Available Objects" section
• Check "Show Name"
• Change the history mode to "Unlimited"
• Go into "Viewpoints" on the left-hand side
• Change the reference frame to "Inertial"
• Click "Ok" to close the editor

Building the Mission Sequence

• Drag and drop a FreeForm script editor into the Mission Sequence
• Name this "Set up Console and Initial Conditions"

In this script, we will use the Console window to report the spacecraft's initial Keplerian orbital elements, and save the initial velocity. To do this, we double-click to open the FreeForm and write:

```csharp
// Set up Console
Console.DockMode = 3;
Console.WordWrap = 1;
Console.Dimension = 35;
Console.Show();

// Report initial spacecraft elements
Report "Initial Spacecraft Elements: " to Console;
Report "E: ", Spacecraft1.E to Console;
Report "I: ", Spacecraft1.I to Console;
Report "RAAN: ", Spacecraft1.RAAN to Console;

// Save the initial velocity (circular velocity for the current SMA)
initialVelocity = Spacecraft1.VMag;
```

We need to save the spacecraft in its initial state before executing the targeting loop.
In the Mission Sequence, drag and drop a "Save" command after the FreeForm
Double click the command to open the editor and change the "SaveName" placeholder to "initial state"
Click "Ok" to close the editor

Now, we will create the targeting loop, which will vary the V and N components of ImpulsiveBurn1 to achieve a new spacecraft inclination of 35 degrees and the same initial velocity (meaning we will remain in a circular orbit of the same semi-major axis).

In the Mission Sequence, drag and drop a "Target...End" script element after the "Save" command
Double click the command to open the Targeting editor
In the "Initial Setup" tab, select Spacecraft1 as the object to reset at the start of each iteration of the targeting loop
Navigate to the "Choose Parameters to Vary" tab and click "Add Vary" to add to the list of components to vary in each iteration:
  - In the "Parameter to Vary" box, use the drop-down menu to select "Object Property/Method" (under "Choose which type of item to use"), "ImpulsiveBurn1" (under "Choose an object") and "BurnDirection" (under "Choose a property or method")
    - The index value in the last box should be 0, indicating that you are accessing the first element of the BurnDirection array; this represents the component in the V direction of the VNB frame
  - Select "Add Vary" again and perform the same process, but use enter an index value of "1" to represent the second component of the BurnDirection array, the N component in the VNB frame
    - For this one, in the "Seed Value" box, enter 1 km/s since we know that most of the Δv will be in the N direction

Navigate to the "Choose Conditions to Target" tab and click "Add Achieve" to indicate the goals of our targeter:
  - In the "Parameter to Achieve" box, use the drop-down menu to select "Spacecraft1" (under "Choose an object") and "I" (under "Choose a property or method")
    - Set the "Goal Value" field to be 35 degrees (our desired new inclination)
    - Set the "Tolerance (+/-)" field to be 0.01 degrees
  - Select "Add Achieve" again, and in the "Parameter to Achieve" box, use the drop-down menu to select "Spacecraft1" (under "Choose an object") and "VMag" (under "Choose a property or method")
    - In the "Goal Value" field’s drop-down menu, select "Object" under "Choose which type of item to use" and select "initialVelocity" under "Choose an object"
Set the "Tolerance (+/-)" field to be 0.001 km/s
Note: look at the "Script" section in the Targeting editor to see how you could write the Targeting loop in FreeFlyer script
- Click "Ok" to close the editor
- In the Mission Sequence, drag and drop a "Maneuver" command inside the Targeting loop
  - Double click the command to open the Maneuver editor
  - Uncheck the "Create Report?" box and click "Ok" to close the editor

We have now completed the targeting loop, but inside the loop we performed a Maneuver. We must restore the Spacecraft to its initial state before visualizing the initial orbit.

- In the Mission Sequence, drag and drop a "Restore" command after the Targeting loop
- Double click the command to open the editor and change the "SaveName" placeholder to "initial state"
- Click "Ok" to close the editor

At this point, your Mission Sequence should look something like this:

Now, we want to visualize the initial orbit and the final orbit that results from our burn. To do this, we will propagate the Spacecraft for one day in its initial orbit, then maneuver using the ImpulsiveBurn determined by our targeter, then continue to propagate in the new orbit. We will also report the new Keplerian elements and the burn components to the Console window.

- In the Mission Sequence, drag and drop a "While...End" loop after the "Restore" command
- Inside the loop, drag and drop a "Step" command
- Also inside the loop, drag and drop an "Update" command
- After the "While...End" loop, drag and drop a "Maneuver" command
  - Double click the command to open the Maneuver editor
  - Uncheck the "Create Report?" box and click "Ok" to close the editor
- Drag and drop a FreeForm script editor into the Mission Sequence
- Name this "Report New Orbital Elements and Burn Components"

In this script, we will change the Spacecraft's color (to differentiate from the initial orbit) and report the post-burn Keplerian elements, as well as the burn components and total Δv magnitude. To do this, we write:

```csharp
// Change the Spacecraft color
Spacecraft1.Color = ColorTools.Yellow;
```
// Report the Keplerian elements after the burn
Report "Post-burn Spacecraft Elements: " to Console;
Report " E: ", Spacecraft1.E to Console;
Report " I: ", Spacecraft1.I to Console;
Report " RAAN: ", Spacecraft1.RAAN to Console;

// Report the burn attributes
Report "Burn: " to Console;
Report " V: ", ImpulsiveBurn1.BurnDirection[0] to Console;

Now we just need to propagate the Spacecraft for a day in its new orbit:

- In the Mission Sequence, drag and drop a "While...End" loop
- Inside the loop, drag and drop a "Step" command
- Also inside the loop, drag and drop an "Update" command

Your Mission Sequence should now look something like this:

![Mission Sequence](image)

Save and Run your Mission Plan, then try to answer these questions:
Try solving this problem by hand. How do your calculations compare to FreeFlyer’s calculations?

What other “Achieves” could you have set in your Targeting loop? Try Achieving Spacecraft1.A = 10000 km (your initial SMA) and Spacecraft1.E = 0 (your initial eccentricity) instead of Achieving Spacecraft1.VMag = initialVelocity. Can you produce the same result?

If you were changing the inclination from any orbit other than an equatorial one, where in the orbit would you need to perform the burn? How would you modify your code to account for this additional constraint?

BONUS: try writing your Targeting loop in FreeFlyer script instead of using the drag-and-drop method. Recall that the Targeting editor displays a sample of what the FreeFlyer script should look like.

See Also

- Targeting Tutorial
- Previous Topic: Targeting Concepts
- Next Topic: B-plane Targeting
6.3 - B-Plane Targeting

In the Interplanetary Topics tutorial, we discussed how the B-plane can be used by mission planners to fly by a specific location on a planet, or target a capture orbit of a specific inclination. As a quick review, we recall that the B-plane is defined as orthogonal to the hyperbolic trajectory plane, and is described by three unit vectors R, S, and T, where S is parallel to the hyperbolic excess velocity, T lies on the ecliptic plane of the solar system, and R completes the right-handed set. R and T form the unit vectors of the B-plane itself.

We also define the B vector, which points from the center of the planetary body for which the B-plane is defined to the point at which the incoming asymptote of a spacecraft's hyperbolic trajectory pierces the B-plane. Finally, we define an angle Θ, which is the angle between the B vector and T vector.

For more information on what these vectors mean and how to calculate them, see The B-Plane in Interplanetary Topics. We will be reusing some methods from the aforementioned section for calculating and visualizing the B-plane in this tutorial.

The B-Plane in FreeFlyer

In the B-plane tutorial in the Interplanetary Topics chapter, we calculated the B-plane vectors manually in FreeFlyer in order to visualize them and develop an understanding of what they mean. However, FreeFlyer has the capability to calculate many of these parameters for us! We can use built-in methods to calculate where a spacecraft's hyperbolic trajectory pierces the B-plane. These methods are:

- BPlaneBMag - returns the distance of the piercing point from the origin of the B-plane (the center of the specified CelestialObject) [km]
- BPlaneTheta - returns the angle in the B-plane between the B and T vectors [deg]
- BPlaneBdotT - returns the projection of the piercing point onto the T vector [km]
- BPlaneBdotR - returns the projection of the piercing point onto the R vector [km]

We will use these methods to target a specific point on Mars's B-plane, a technique which is often used to achieve a desired orbit or flyby location.

Targeting Mars's B-Plane

### Problem:

Scientists want to investigate the presence of liquid water on Mars's polar ice caps. We have a spacecraft flying on a hyperbolic trajectory towards Mars. Using two impulsive burns, we want to insert the spacecraft into a circular, polar orbit to map the poles and gather scientific data. Use targeting loops to determine the magnitudes and components required of the burns to achieve our goal.

This mission plan will include Targeting loops and B-plane configuration as well as a lot of visualization aspects for convenience. Although you should be sure to understand the code in the tutorial which applies to Targeting and the B-plane, feel free to skim over the visualization aspects, unless they are particularly interesting to you.
Create a New Mission Plan and save it as "TargetingMarsBPlane.MissionPlan"

Adding Spacecraft Objects

- Create a new Spacecraft and change its Central Body to Mars
- Set the following orbital elements (remember to change the "Element Type" field to "Keplerian"):
  - A: -6000 km
  - E: 2
  - I: 35 deg
  - RAAN: 0 deg
  - W: 0 deg
  - TA: 243 deg
- In the "Force Model" tab, select the Mars checkbox under "Available Bodies". This will include Mars's gravity in the Spacecraft's Force Model
- Click "Ok" to close the Editor

- Create another new Spacecraft named "MarsCenter"
- Change the Spacecraft's Central Body to Mars
  Note: This Spacecraft is being used solely to visualize the B-plane. Its position will be at the center of Mars, and its Proximity Zones will represent the trajectory plane (of Spacecraft1) and Mars's B-plane. An understanding of Proximity Zones is NOT required for this tutorial.
- Navigate to the "Proximity Zones" tab (under "Subsystems"). We will create two Proximity Zones to visualize B-plane properties.
  - Click "Create" to create your first Proximity Zone and give it the Name "TrajectoryPlane", then click "Edit ProximityZone"
  - Under "Size", set X = 20000 km, Y = 20000 km, and Z = 5 km
  - Under "Orientation wrt Reference Frame", set the "Offsets applied to" field to "Mean of Earth J2000", and set "Euler Angle 2" to 45
  - Navigate to the "Visualization" tab and select "Translucent" (rather than "Wire Frame") and change the Tick Scale to 100000 km. Change the Color to a light blue (this step is not necessary, but recommended for consistency), and change the Opacity to 0.3
  - Click "Ok" to close the Editor
  - Your recently created Proximity Zone "TrajectoryPlane" should be selected. Click "Clone" to create an exact copy of this Proximity Zone, and rename it "BPlane", then click "Edit ProximityZone"
  - The current properties of the BPlane Proximity Zone are exact copies of the TrajectoryPlane Proximity Zone. Change "Euler Angle 2" to 0
  - Navigate to the "Visualization" tab and change the Color to yellow (or another color which is different from the TrajectoryPlane color)
  - The Editor displays your Proximity Zones in a window. At this point, it should look something like this (you can zoom around within the window):
Note that eventually, we will position the MarsCenter spacecraft so that it is at the center of Mars (as well as its associated Proximity Zones). We will do this later, in FreeFlyer script.

- Click "Ok" to close the Proximity Zone Editor
- Click "Ok" to close the Spacecraft Editor

### Adding ImpulsiveBurn Objects

- Create a new ImpulsiveBurn object
- Open the ImpulsiveBurn editor
- Change the attitude system to "VNB"
- Click "Ok" to close the editor
- Repeat the process for a second ImpulsiveBurn object

### Adding CoordinateSystem and Vector Objects

- Create a new CoordinateSystem Object and name it "BPlane" (You can right-click to Rename)
- Create five vector objects and name them "B", "R", "S", "T", and "X"

### Adding ViewWindows

- Create a new ViewWindow Object
- Open the ViewWindow Editor
- Under "Available Objects", check Spacecraft1
- Change the history mode for Spacecraft1 to "Unlimited"
- Navigate to the "Viewpoints" tab
- Fill in the Default Viewpoint with the following properties:
  - Name: DistanceView
  - Title: DistanceView
  - Type: 3D View
  - Reference Frame: Inertial
  - Source: Mars
  - Target: Mars
  - Tail Reference: Mars
  - Right Ascension: 209 deg
  - Declination: -17 deg
  - Radius: 309900 km
  - Field of View: 45 deg
- Click "Ok" to close the editor

- Create another ViewWindow Object
- Under "Available Objects", check Spacecraft1, MarsCenter, and Vectors B, R, S, and T
- Select Spacecraft1 and change the "History Mode" to "Unlimited"
- For Spacecraft1, B, R, S, and T, check the "Show Name" box
- Navigate to the "Viewpoints" tab
- Fill in the Default Viewpoint with the following properties:
  - Name: Default
  - Title: Default
  - Type: 3D View
  - Reference Frame: Inertial
  - Source: Mars
  - Target: Mars
  - Tail Reference: Mars
  - Right Ascension: 296 deg
  - Declination: -17 deg
  - Radius: 30000 km
  - Field of View: 45 deg
- Click "Ok" to close the editor

At this point, your Object Browser should look something like this:
Now we are ready to begin writing our Mission Sequence. Since this Mission Plan is fairly complicated, we will be using FreeFlyer script in the FreeForm script editor rather than the "drag-and-drop" method to execute our commands. Using Freeflyer script provides more flexibility and allows you to access more Properties of Objects than you could otherwise. In instances where the script is outside the scope of this tutorial (such as for visualization purposes), we will suggest that you copy and paste the example code provided.

**Setting up the Console**

- Drag and drop a FreeForm script editor into the Mission Sequence, and double click to open it
- Rename the FreeForm "Set up Console"
- Copy and paste the following code into the FreeForm script editor (this will simply set up the Console window for convenient reports at various times in the mission, and display Mars with a SurfaceLayer)

```plaintext
```
```csharp
Console.DockMode = 3;
Console.Dimension = 50;
Console.WordWrap = 1;
Console.Show();

// Set up Mars visualization
Mars.Globe.SurfaceLayer.UseDaytimeImage = 1;
```

**Targeting the B-Plane**

- Go back to the Mission Sequence
- Drag and drop another FreeForm script editor at the bottom of the Mission Sequence
- Rename this FreeForm to "Target B-Plane at Mars's South Pole"

In this script, we will implement our first Targeting loop, to intersect Mars's B-plane at a radius of 8000 km and angle theta of 115 degrees (we use 115 because Mars has an axial tilt of 25 degrees - if there were no tilt, we would want to intersect the B-plane at an angle of 90 degrees to achieve a polar orbit).

First, we will report the current action to the Console and save the Spacecraft's initial state so that we can retrieve it later. We will also change the spacecraft's color for visualization purposes, and increase the StepSize to make the simulation run faster. Finally, we will create a variable called "peri_time" to store the time that the spacecraft will reach periapsis - this is useful so that we can propagate the orbit within the targeting loop and see the actual targeting process. To do this, copy and paste the following code into the FreeForm script editor:

```csharp
// Update the Console on what is happening
Report "Targeting the B-Plane at Mars's south pole..." to Console;
Report "" to Console;

// Save the Spacecraft's current state so that we can go back and propagate from the beginning
Save Spacecraft1 as "initial state";

// Set the Spacecraft color
Spacecraft1.Color = ColorTools.Aqua;

// Increase the step size to speed up simulation
Spacecraft1.Propagator.StepSize = TIMESSPAN(500 seconds);
```

Next, we will create the Targeting loop by typing:

```csharp
Target;
End;
```
The next portions of our code will exist inside of the Targeting loop (before the "End" statement). Try to construct the Targeting loop yourself by following the instructions! Example code is included after the instructions for reference.

- The commands that you will need to use are
  - "Iterate" - indicates to the loop which objects to restore to their initial properties at each iteration
  - "Vary" - indicates which objects to change at each iteration, requires a seed and perturbation (i.e. "Vary ObjectName = seed + perturbation;")
  - "Maneuver" - applies the ImpulsiveBurn to the Spacecraft
  - "Achieve" - indicates the goals of the targeting loop, as well as their tolerances (i.e. "Achieve ObjectName = goal +/- tolerance;")

- Use the Iterate command to tell the Targeting loop to restore Spacecraft1 after each iteration
- Vary all three directions (V, N, and B) of ImpulsiveBurn1, accessed through the BurnDirection property (ImpulsiveBurn1.BurnDirection[0] is the first direction, V. Use indices 1 and 2 to access the N and B directions). Use a seed value of 0.1 km/s and a perturbation of 0.001 km/s for all three directions
- Use the Maneuver command to apply an impulsive ∆v to Spacecraft1 using ImpulsiveBurn1
- Achieve a B-plane magnitude of 8000 km with a tolerance of 5 km, and an angle of 115 degrees with a tolerance of 0.5 degrees (recall the B-plane methods in FreeFlyer that we discussed earlier!)
  - We must target an angle of 115 degrees for a polar orbit because Mars has an axial tilt of 25 degrees, and the T vector points along the ecliptic plane, so we must account for the tilt in our targeting
- After you have attempted to construct the Targeting loop on your own, check your code against the following:

```
Target;
Iterate Spacecraft1;

// Allow all three burn direction magnitudes to vary
Vary ImpulsiveBurn1.BurnDirection[0] = 0.1 + 0.001;
Vary ImpulsiveBurn1.BurnDirection[1] = 0.1 + 0.001;
Vary ImpulsiveBurn1.BurnDirection[2] = 0.1 + 0.001;

// Perform the calculated maneuver
Maneuver Spacecraft1 using ImpulsiveBurn1;

// Attempt to achieve a B vector magnitude of 8000 km and an angle between B and T of 115 degrees
Achieve Spacecraft1.BPlaneBMag(Mars) = 8000 +/- 5;
Achieve Spacecraft1.BPlaneTheta(Mars) = (90+25) +/- 0.5;
End;
```

For this tutorial, it will be useful to visualize the targeting process. To do this, we will update the ViewWindow while stepping the Spacecraft to the periapsis of the orbit created by maneuvering using ImpulsiveBurn1. By doing this inside of the Targeting loop, we can visualize the resulting orbit from each iteration of the Targeter. Inside of the Targeting loop (but after the Achieve statements), write:
Our Targeting loop is now complete! The final step is to restore the Spacecraft to its initial state, which we do after the end of the Targeting loop by typing:

```
Restore Spacecraft1 from "initial state";
```

Double check that your code (from the beginning of the Targeting loop) looks like this:

```
Target;
  Iterate Spacecraft1;
  // Allow all three burn direction magnitudes to vary
  Vary ImpulsiveBurn1.BurnDirection[0] = 0.1 + 0.001;
  Vary ImpulsiveBurn1.BurnDirection[1] = 0.1 + 0.001;
  Vary ImpulsiveBurn1.BurnDirection[2] = 0.1 + 0.001;
  // Perform the calculated maneuver
  Maneuver Spacecraft1 using ImpulsiveBurn1;
  // Attempt to achieve a B vector magnitude of 8000 km and an angle between B and T of 115 degrees
  Achieve Spacecraft1.BPlaneBMag(Mars) = 8000 +/- 5;
  Achieve Spacecraft1.BPlaneTheta(Mars) = (90+25) +/- 0.5;
  // Propagate to periapsis
  WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
  Update ViewWindow1;
  End;
  ViewWindow1.InsertLineBreak();
End;
Restore Spacecraft1 from "initial state";
```

We are now finished with our first Targeting loop! If you’d like, you can save and run your Mission Plan at this point to see how the Spacecraft’s trajectory changes with each iteration of the Targeter. If you do run your Mission Plan, you should see something like this:
When the simulation runs on your screen, you should be able to see how the trajectory changes and approaches the goal each time the Targeter varies the magnitude of various components of the initial burn. Click "Control" at the top of the screen to return to your script.

Resetting the View and Propagating the Converged Solution Trajectory

- Go back to the Mission Sequence
- Drag and drop another FreeForm script editor at the bottom of the Mission Sequence
- Rename this FreeForm to "Reset Viewpoint"

This script is mostly for visualization purposes. We will clear all the iterations of the Targeter from the ViewWindow and adjust the viewpoint so that we can zoom along with the spacecraft as it approaches Mars on the trajectory determined by the Targeting loop. We will propagate this solution from the start to the point of periapsis, at which point we will enter our second Targeting loop (in the next section). Copy and paste the example code below into your FreeForm:

```plaintext
// Clear targeting tails
ViewWindow1.ResetTails();
Spacecraft1.Color = ColorTools.Aqua;
Spacecraft1.Propagator.StepSize = TIMESPAN(50 seconds); // Slow the simulation down

// Apply the ImpulsiveBurn determined by the Targeter
Maneuver Spacecraft1 using ImpulsiveBurn1;
Save Spacecraft1 as "temp_save";
Step Spacecraft1 to (Spacecraft1.OrbitPeriapsis(Mars));
```
periEpoch = Spacecraft1.Epoch; // Save time of periapsis
Restore Spacecraft1 from "temp_save";

// Count the number of steps to determine the zoom rate
Variable count = 0;
WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
    count += 1;
End;
Restore Spacecraft1 from "temp_save";

// Zoom in with the Spacecraft
ViewWindow1.SetShowName(Spacecraft1.ObjectId, 1);
Variable deltaRA = abs(209-236)/count;
Variable deltaDec = abs((-17) - (-29))/count;
Variable deltaZoom = (309900 - 30000)/count;
Variable i = 1;

WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
    ViewWindow1.CurrentViewpoint.ThreeDView.Radius = 309900-(deltaZoom*i);
    ViewWindow1.CurrentViewpoint.ThreeDView.RightAscension = 236 -
(deltaRA*i);
(deltaDec*i);
    Update ViewWindow1;
    i += 1;
End;

// Now, the spacecraft should be at periapsis

Targeting a Circular, Polar Orbit

- Go back to the Mission Sequence
- Drag and drop another FreeForm script editor at the bottom of the Mission Sequence
- Rename this FreeForm to "Target a circular polar orbit"

Here, we will use another Targeting loop to insert our spacecraft into a circular orbit. First, we will report the current action to the Console. Then, we need to step the Spacecraft to exactly the time of periapsis - although we used a While loop to step it to periapsis in the previous section, our Spacecraft's StepSize property may not have allowed the Spacecraft to step to the exactly correct time, so it is a good practice to use a "Step to" command to ensure we have everything set up correctly. Copy and paste the following code into the FreeForm script editor:

// Update the Console on what is happening
Console.CurrentTextColor = ColorTools.Yellow;
Report "Targeting a circular orbit..." to Console;
Report "" to Console;
We are now ready to create our second Targeting loop. Try using what you learned from the first targeting loop to construct this one yourself!

- Use the Iterate command to tell the Targeting loop to restore Spacecraft1 after each iteration
- Vary just the first component of ImpulsiveBurn2 (the V component) with a seed of -2 km/s and a perturbation of 0.001 km/s
- Use the Maneuver command to apply an impulsive Δv to Spacecraft1 using ImpulsiveBurn2
- Achieve an eccentricity for Spacecraft 1 of 0 with a tolerance of 0.0001

The targeting aspect of the loop is now complete, but again we would like to visualize how the Targeter works to refine the solution to a circular orbit. To do this, we will step Spacecraft1 for a full period of its (now elliptical) orbit for each iteration. Copy and paste the following code inside your Targeting loop:

```csharp
// Visualize the solution from each iteration of the targeting loop
Step Spacecraft1; // Must step so that we are not already at periapsis
Update ViewWindow1;
WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
    Update ViewWindow1;
End;
ViewWindow1.InsertLineBreak();
```

Finally, restore the Spacecraft to its initial state after the end of the Targeting loop by typing:

```csharp
Restore Spacecraft1 from "initial state";
```

Your code (from the beginning of the Targeting loop) should look like this:

```csharp
Target;
    Iterate Spacecraft1;
    // Vary just the burn in the velocity direction
    Vary ImpulsiveBurn2.BurnDirection[0] = -2 + 0.001;
    // Perform the calculated maneuver
    Maneuver Spacecraft1 using ImpulsiveBurn2;
```
// Achieve a circular orbit
Achieve Spacecraft1.E = 0 +/- 0.0001;

// Visualize the solution for each iteration of the targeting loop
Step Spacecraft1;
Update ViewWindow1;

WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);
    Update ViewWindow1;
End;

ViewWindow1.InsertLineBreak();
End;

// Restore the Spacecraft to its beginning state
Restore Spacecraft1 from "initial state";

If you'd like, you can save and run your Mission Plan at this point to see how the Spacecraft's trajectory changes with each iteration of the Targeter. If you do run your Mission Plan, you should have something like this at the end:

When the simulation runs on your screen, you should first see the iterations of the first targeting loop that you wrote (in light blue), then you should zoom in to a closer view to see the solution trajectory produced by the first ImpulsiveBurn, then the iterations of the second targeting loop that you wrote (in yellow). We now have solutions for the two burns! All that is left is to visualize the B-plane and view the final trajectory. Click "Control" at the top of the screen to return to your script.

Visualizing the B-Plane
• Go back to the Mission Sequence
• Drag and drop another FreeForm script editor at the bottom of the Mission Sequence
• Rename this FreeForm to "Visualize B-Plane"

Much of this script comes from the earlier tutorial The B-Plane in the Interplanetary Topics chapter. Refer to that section if you would like to review B-plane concepts. Copy and paste the following code into the FreeForm script editor:

```plaintext
Maneuver Spacecraft1 using ImpulsiveBurn1;

// Initial state vector
Array rVect = Spacecraft1.Position;
Array vVect = Spacecraft1.Velocity;

// Calculations
Array hUnit = rVect.CrossProduct(vVect).Normalized;
Variable beta = acos(1/eVect.Norm);

Array sUnit = cos(beta) * eVect.Normalized + sin(beta) * (hUnit.CrossProduct(eVect.Normalized));
Array nUnit = {0,0,1};
Array tUnit = sUnit.CrossProduct(nUnit).Normalized;
Array rUnit = sUnit.CrossProduct(tUnit);

Array bUnit = sUnit.CrossProduct(hUnit);
Variable bMag = Spacecraft1.BPlaneBMag(Mars);

Array bVect = bMag * bUnit;
Variable bTheta = Spacecraft1.BPlaneTheta(Mars);

// Set up the vector epochs
B.Epoch = Spacecraft1.Epoch;
S.Epoch = Spacecraft1.Epoch;
T.Epoch = Spacecraft1.Epoch;
R.Epoch = Spacecraft1.Epoch;

// Set up the vector elements
B.Element = bVect;
S.Element = sUnit;
T.Element = tUnit;
R.Element = rUnit;

// Build all the vectors
B.BuildVector(0);
S.BuildVector(0);
T.BuildVector(0);
R.BuildVector(0);
```

© 2019 a.i. solutions, Inc
// Set to draw as arrows
B.DrawMethod = 1;
S.DrawMethod = 1;
T.DrawMethod = 1;
R.DrawMethod = 1;

// Set the vector colors
B.Color = ColorTools.Yellow;
S.Color = ColorTools.Red;
T.Color = ColorTools.Cyan;
R.Color = ColorTools.Green;

// Set all the vectors as visible
B.Active = 1;
S.Active = 1;
T.Active = 1;
R.Active = 1;

// Set the position of the vectors
B.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
S.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
T.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
R.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
Earth.GetPositionAtEpoch(Spacecraft1.Epoch);

X.Element = {1000, 0, 0};

Restore Spacecraft1 from "initial state";

Propagating the Solution Trajectory

- Go back to the Mission Sequence
- Drag and drop another FreeForm script editor at the bottom of the Mission Sequence
- Rename this FreeForm to "Propagate Spacecraft"

Here, we will propagate and visualize the final solution trajectory of our Spacecraft, executing the two burns determined by our Targeting loops. First, we will report the current action to the Console and set up some visualization aspects. Copy and paste the following code into the FreeForm script editor:

// Update the Console on what is happening
Console.CurrentTextColor = ColorTools.Magenta;
Report "Propagating the Spacecraft's final trajectory..." to Console;
Report "" to Console;
Spacecraft1.Color = ColorTools.Magenta;
Spacecraft1.Propagator.StepSize = TIMESPAN(50 seconds);

// Set up View
deltaZoom = (377800-54000)/count;
i = 1;
OutputLayout.SetWindowState(ViewWindow1.ID, 1);
OutputLayout.SetWindowState(ViewWindow2.ID, 2);
OutputLayout.ApplyUpdates();
Update ViewWindow2;

Next, we will begin our actual propagation. First, we maneuver Spacecraft1 using ImpulsiveBurn1. Then, we will propagate the Spacecraft until it reaches periapsis, updating the B-plane and our ViewWindow each step of the way. Copy and paste the following code into the FreeForm script editor:

Maneuver Spacecraft1 using ImpulsiveBurn1;

// Propagate the spacecraft until it reaches periapsis
WhileStepping Spacecraft1 to (Spacecraft1.OrbitPeriapsis);

    // Set the MarsCenter Spacecraft at the center of Mars
    MarsCenter.Epoch = Spacecraft1.Epoch;
    MarsCenter.Position = {0, 0, 0};

    // The TrajectoryPlane ProximityZone visualizes the orbit plane
    MarsCenter.ProximityZones[0].Orientation[0] = Spacecraft1.RAAN;
    MarsCenter.ProximityZones[0].Orientation[1] = Spacecraft1.I;

    // The BPlane ProximityZone visualizes the B-Plane
    BPlane.Epoch = Spacecraft1.Epoch;
    BPlane.BuildCoordinateSystem(3, S, 1, X);
    MarsCenter.ProximityZones[1].SetOrientation(BPlane);

    // Moves the position of the vectors
    B.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
                    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
    S.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
                    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
    T.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
                    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
    R.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) -
                    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);

    // Zoom in closer to Mars
    ViewWindow2.CurrentViewpoint.ThreeDView.Radius = 377800 -
    (deltaZoom*i);
i += 1;
Update ViewWindow2;
Now, we can move on to the second piece of the trajectory: insertion into a circular, polar orbit. Now that the Spacecraft has been stepped to periapsis, we will maneuver the Spacecraft using ImpulsiveBurn2. Finally, we will propagate the Spacecraft for half a day in its new circular orbit, continuing to update the B-plane and our ViewWindow. Copy and paste the following code into the FreeForm script editor:

```plaintext
// Perform the second Maneuver
Maneuver Spacecraft1 using ImpulsiveBurn2;

// Propagate the spacecraft in its circular polar orbit for half a day
While (Spacecraft1.ElapsedTime < TIMESPAN(0.5 days));
    Step Spacecraft1;

    // Set the MarsCenter Spacecraft at the center of Mars
    MarsCenter.Epoch = Spacecraft1.Epoch;
    MarsCenter.Position = (0, 0, 0);

    // The TrajectoryPlane ProximityZone visualizes the orbit plane
    MarsCenter.ProximityZones[0].Orientation[0] = Spacecraft1.RAAN;
    MarsCenter.ProximityZones[0].Orientation[1] = Spacecraft1.I;

    // The BPlane ProximityZone visualizes the B-Plane
    BPlane.Epoch = Spacecraft1.Epoch;
    BPlane.BuildCoordinateSystem(3, S, 1, X);
    MarsCenter.ProximityZones[1].SetOrientation(BPlane);

    // Moves the position of the vectors
    B.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - 
    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
    S.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - 
    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
    T.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - 
    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);
    R.VisualOffset = Mars.GetPositionAtEpoch(Spacecraft1.Epoch) - 
    Earth.GetPositionAtEpoch(Spacecraft1.Epoch);

    Update ViewWindow2;
End;
```

Spacecraft1 has now completed its full targeted trajectory! The remaining code simply sets up the ViewWindows in a convenient way, then reports relevant information to the Console, including the targeted components of both burns, the B-plane properties, and the Spacecraft's final orbital properties. Copy and paste the following code into the FreeForm script editor:

```plaintext
OutputLayout.SetWindowState(ViewWindow1.ID, 0);
```
Go back to the Mission Sequence. You should have six FreeForm blocks like this:

<table>
<thead>
<tr>
<th>Mission Sequence</th>
<th>1 - Set up Console</th>
<th>2 - Target B-Plane at Mars’s South Pole</th>
<th>3 - Reset Viewpoint</th>
<th>4 - Final trajectory view</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FreeForm: Set up Console</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>FreeForm: Target B-Plane at Mars’s South Pole</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>FreeForm: Reset Viewpoint</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>FreeForm: Target a circular polar orbit</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>FreeForm: Visualize the B-Plane</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>FreeForm: Propagate Spacecraft</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Save and run your Mission Plan. You should see a simulation of targeting the first and second burns, then a new window depicting the final trajectory and visualizing the trajectory plane and B-plane. When the run finishes, you should have two windows side by side, one showing the end of the second targeting sequence (on the left) and the other showing the final trajectory (on the right). Use your mouse to zoom in and pan around both views. Notice how the trajectory plane in blue aligns perfectly with the Spacecraft’s hyperbolic trajectory, and how the B-plane in yellow is exactly perpendicular to the trajectory plane. Also notice how the Spacecraft passes directly over both poles - you should be able to see Mars’s polar ice caps. After examining your results, try to answer these questions:

The Console window should have reported the angle between B and T, as well as the magnitude of the B vector. Are they within the tolerances that we set for our goals in the first Targeting loop? In the final trajectory view, does the angle between B and T look like it could be 115 degrees, as we expect?
Why do we only need to burn in the V direction for the second impulsive burn? Try varying the other
directions as well in the second Targeting loop, using a seed value of 0 km/s. Do they make a significant
contribution to the total Δv if we allow them to vary?

See Also

- Targeting Tutorial
- Previous Topic: Inclination Change
CHAPTER 7

Real World Modeling
This chapter will discuss the perturbations that commonly apply in the real world. We will examine perturbing forces both large and small and discover the importance of incorporating these forces into your model.

This chapter will cover the following topics:

1. Multi-Body Effects
2. J2 Perturbation
3. Atmospheric Modeling
4. Solar Radiation Pressure
7.1 - Multi-Body Effects

In the "Orbital Elements" chapter, we learned about the six Keplerian elements that describe an orbit: the semi-major axis, eccentricity, inclination, right ascension of the ascending node, argument of perigee, and true anomaly. These six elements were sufficient enough to describe the entire orbit of the spacecraft. However, these Keplerian elements assume a relatively simple model of gravity.

Keplerian orbits are defined based on two assumptions. The first is that there are only two bodies in the model - the central body, and the satellite. The second is that both bodies are modeled as point masses - each body has a uniform gravitational field around a single point at the very center of the body.

When dealing with the real world, however, the Earth's gravity is not the only force acting upon our orbiting spacecraft. In fact, depending upon where our spacecraft is in space, bodies such as the Moon, the Sun, and even Jupiter can change our orbits in small ways (and sometimes big ways). To increase the fidelity of our analyses, we can add more perturbing forces to our calculations. When we are calculating our orbit with two main gravitational bodies and our spacecraft (three bodies in total), we call that a "Three Body Problem". When three bodies aren't enough and we add more, we call it a "Multi-Body" Problem. However, when dealing with three body and multi-body problems, it is very difficult to calculate by hand. So, the best tool to use for these calculations is a spaceflight simulator, like FreeFlyer.

Modeling a Two Body, Three Body, and Multi-Body Problem

Problem:
We have a spacecraft orbiting the Earth in a Highly Elliptical Orbit (HEO). What does the orbit look like in a two body model (Earth and Spacecraft), a three body problem (Earth, Sun, and Spacecraft), and a multi-body problem (Earth, Sun, Moon, and Spacecraft)? The initial Keplerian elements for the spacecraft are as follows:

- A: 290,521.192 km
- E: 0.9759
- I: 25.7786 deg
- RAAN: 348.5809 deg
- W: 344.1243 deg
- TA: 0 deg
- Epoch: "Jan 01 2010 03:10:00.000"

Add a new Mission Plan and save it as "MultiBodyProblem.MissionPlan"

Adding in Spacecraft

- Create a Spacecraft with the following Keplerian elements
  - A: 290521.192 km
  - E: 0.9759
  - I: 25.7786 deg
- RAAN: 348.5809 deg
- W: 344.1243 deg
- TA: 0 deg

- Be sure to change the Spacecraft epoch to "Jan 01 2010 03:10:00.000"

- Click on "Force Model" on the left-hand side of the Spacecraft editor
- Uncheck the "Moon" and "Sun" boxes

- Change the Spacecraft name to "TwoBodySC"
- Click "Ok" to close the editor

- Right-click "TwoBodySC" and clone it
- Double-click "TwoBodySC_Copy1"
- In the Spacecraft object's force model, check the "Sun" box
- In "Visualization", change the tail color to green
- Change the name to "ThreeBodySC"
- Click "Ok" to close the editor

- Right-click "TwoBodySC" and clone it
- Double-click "TwoBodySC_Copy1"
- In the Spacecraft object's force model, check the "Sun" and the "Moon" box
- In "Visualization", change the tail color to yellow
- Change the Spacecraft object's name to "MultiBodySC"
- Click "Ok" to close the editor
Adding the ViewWindow

- Create a ViewWindow through the Object Browser
- Double-click "ViewWindow1"
- Check "TwoBodySC", "ThreeBodySC", and "MultiBodySC" under the "Available Objects"
- Check "Show Name" for all three Spacecraft
- Change the history mode to "Unlimited" for all three Spacecraft
- Click on "Viewpoints" on the left-hand side
- Change the reference frame to "Inertial"
- Change the radius to 660000 km
- Click "Ok" to close the editor

Building the Mission Sequence

- Drag and drop a While loop onto the Mission Sequence
- Change the While loop argument to "(TwoBodySC.ElapsedTime < TIMESPAN(20 days))"
- Drag and drop a FreeForm script editor inside the while loop
- Double-click on the FreeForm script editor
- Change the name to "Step and Update"

In this FreeForm, we will step all Spacecraft, making sure to keep the epochs synced, and update the ViewWindow. To do this, we write:

```plaintext
// Step each spacecraft with an epoch sync
Step TwoBodySC;
Step ThreeBodySC to (ThreeBodySC.Epoch == TwoBodySC.Epoch);
Step MultiBodySC to (MultiBodySC.Epoch == TwoBodySC.Epoch);

// Updates the ViewWindow
Update ViewWindow1;
```

Your Mission Sequence should look something like this:

![Mission Sequence Example](image)

Save and run your Mission Plan, then try and answer these questions:

Which Spacecraft object's orbit did not change from the original ellipse?
What happened to “ThreeBodySC”? What do you think caused this to happen?

What is the shape of the orbit of "MultiBodySC"?

Which Spacecraft most closely simulates real life? How does this show you the usefulness of Spacecraft modeling software?

See Also

- Real World Modeling
- Next Topic: J2 Perturbation
7.2 - J2 Perturbation

Long ago, people thought the Earth was flat. Many believed that you could sail to the edge of the Earth and fall off. We have long since dismissed this idea as we now know the Earth is a sphere. Right?

Actually, the Earth really isn't a sphere - it is an oblate spheroid. Because of the rotation of the Earth on its axis, centrifugal force bulges the equator. In fact, the radius at the Earth's equator is about 21 km larger than the radius at the poles. See the below diagram to understand what we're talking about:

Now that we understand that our Earth is really more oblate than spherical, we need to ask ourselves some questions. How does this affect our orbits? There is a perturbing force based on this oblate Earth called "J2 Perturbations." But where does the term "J2" come from? The term J2 comes from an infinite series mathematical equation that describes the perturbational effects of oblateness on the gravity of a planet. The coefficients of each term in this series is described as \( J_k \), of which \( J_2, J_3, \) and \( J_4 \) are called "zonal coefficients." However, J2 is over 1000 times larger than the rest and has the strongest perturbing factor on orbits.

The two main orbital elements affected by J2 Perturbations are the Right Ascension of the Ascending Node (\( \Omega \)) and the Argument of Perigee (\( \omega \)). If we were to model the Earth as a perfect sphere with a uniform gravitational field, the RAAN and argument of perigee would not change. But since our Earth is not really a perfect sphere, it is important that we account for this perturbation.

J2 perturbations will move the RAAN over time at a constant rate depending on the orbit's size, shape, and inclination. Using this property of J2 perturbations, we can manipulate our orbit so that the RAAN changes at a rate of 360 degrees per year, keeping the orbit in the same orientation with respect to the Sun. This is called a "Sun-Synchronous Orbit". However, if we did not account for J2, would have the red orbit in the picture below:
The green orbit on the other hand, does account for J2 and keeps the same orientation with respect to the Sun. The reason this is possible is because the orbit is designed to have the RAAN change 360 degrees per year. The formula for the change of RAAN over time is the formula below:
In this section, we will discuss:

1. Calculating a Sun-Synchronous Orbit
2. Modeling a Sun-Synchronous Orbit

### Calculating a Sun-Synchronous Orbit

In this section, we will discuss:

1. Calculating a Sun-Synchronous Orbit
2. Modeling a Sun-Synchronous Orbit

#### Calculating a Sun-Synchronous Orbit

**Problem:**

We have a circular orbit with an SMA of 7000 km that we wish to make Sun-Synchronous. What inclination do we need for this to occur?

First, we need to calculate exactly what the nodal precession rate is. We know it needs to be 360 degrees per year, but we need to convert it to radians per seconds.

\[
\frac{d\Omega}{dt} = \text{Nodal Precession} \left(\frac{\text{rad}}{s}\right)
\]

\[
J_2 = J_2 \text{ Constant} \left(1.08262668 \times 10^{-3}\right)
\]

\[
R_E = \text{Radius of the Earth} \left(6378 \text{ km}\right)
\]

\[
a = \text{Semi-Major Axis}
\]

\[
e = \text{Eccentricity}
\]

\[
\mu_E = \text{Standard Gravitational Parameter of Earth} \left(398,600.44189 \frac{\text{km}^3}{\text{s}^2}\right)
\]

\[
i = \text{Orbit Inclination} \left(\text{rad}\right)
\]

\[
\frac{d\Omega}{dt} = \frac{-3}{2} \cdot J_2 \cdot \left(\frac{R_E}{a(1-e^2)}\right)^2 \cdot \frac{\mu_E}{a^3} \cdot \cos(i)
\]
This formula looks rather complex. However, we have all of the necessary variables so let's plug in our variables and calculate the result.

\[
i = \cos^{-1}\left(\frac{-2}{3} \cdot \frac{d\Omega}{dt} \cdot \frac{1}{J_2} \cdot \left(\frac{a(1-e^2)}{R_E}\right)^2 \cdot \sqrt{\frac{a^3}{\mu_E}}\right)
\]

\[
\begin{align*}
a & = 7000 \text{ km} \\
e & = 0 \\
d\Omega/dt & = 0.19910213e-6 \text{ rad/s} \\
J_2 & = 1.08262668e-3 \\
R_E & = 6378.1363 \text{ km} \\
\mu_E & = 398600.442 \text{ km}^3/\text{s}^2
\end{align*}
\]

\[
i = 1.7082196041 \text{ rad} = 97.87377 \text{ deg}
\]

**Modeling a Sun-Synchronous Orbit**

Let's model the orbit whose inclination we just calculated in FreeFlyer. For demonstration purposes, we will add in a Spacecraft identical to the Sun-Synchronous Spacecraft we've calculated, but simplify the force model to a point mass and see if there is any nodal precession.

- Open a new Mission Plan and save it as "J2Perturbation.MissionPlan"

**Adding in Spacecraft**

- Create a new Spacecraft with the following Keplerian elements:
  - A: 7000 km
  - E: 0
  - I: 97.87377 deg
  - RAAN: 100 deg
  - W: 0 deg
  - TA: 0 deg
- To speed up simulation time, we will change the propagator. Go to the "Propagator" section on the left-hand side of the Spacecraft object editor
- Change the Integrator Type to "Bulirsch-Stoer VOP"
- Make sure the step mode is set to "Variable Step Size"
real world modeling

© 2019 a.i. solutions, Inc

propagator settings in the spacecraft editor

- Click "Ok" to close the editor
- Right-click "Spacecraft1" and click "Clone"
- Open the newly cloned Spacecraft
- Rename it to "Spacecraft2"
- Go into "Force Model" on the left-hand side
- Change the "Field Type" to "Point Mass"

force model settings in the spacecraft editor

- Press "Ok" to close the editor

adding a plotwindow

- Create a PlotWindow through the object browser
- Double-click "PlotWindow1" to open the editor
- Change the y-axis to "Spacecraft1.RAAN"
- Click "More" to add another line to the plot
- Change the new dropdown to "Spacecraft2.RAAN"
- Click "Ok" to close the editor

building the mission sequence
• Drag and drop a "While...End" loop onto the Mission Sequence
• Change the argument inside the while loop to "(Spacecraft1.ElapsedTime < TIMESPAN(500 days))"

• Drag and drop a FreeForm script editor inside the loop
• Change the name of the FreeForm to "Step and Update"

In this script, we will be stepping both Spacecraft with an epoch sync and updating the plot window. To do this, we write:

```plaintext
// Step both spacecraft with an epoch sync
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);
Update PlotWindow1;
```

Your Mission Sequence should look something like this:

![Mission Sequence Example]

Save and run your Mission Plan, then try and answer these questions:

*Look at the RAAN for Spacecraft1. The output should be like a saw wave. What is the period of this plot?*

*Did Spacecraft2's RAAN change? Why or why not?*

**See Also**

- Interplanetary Topics
- Next Topic: Atmospheric Modeling
- Previous Topic: Multi-Body Effects
7.3 - Atmospheric Modeling

Where does “space” officially begin? What is considered “inside the atmosphere”? A common standard is that “space” begins at a height of 100 km. However, this does not mean that there isn't any atmosphere above 100 km. There is actually a very small amount. It may be very miniscule, but it is enough to make a difference for LEO spacecraft over several years.

This small amount of atmosphere adds another force to be considered when modeling LEO spacecraft - Atmospheric Drag. The formula for this force is:

\[ F_D = -\frac{1}{2} \rho v_{rel}^2 C_D A \]

- \( F_D \) = Force of Drag
- \( C_D \) = Spacecraft coefficient of drag
- \( A \) = Frontal Area
- \( \rho \) = Atmospheric Density
- \( v_{rel} \) = Relative velocity in reference to the object

The primary characteristic of this formula that you should pay attention to is that the force of drag is proportional to the frontal surface area. The bigger the spacecraft, the more drag force. So something like the International Space Station will feel a lot of drag.

However, if very large things feel a lot of drag, why isn't the ISS crashing into Earth as we speak? The answer is twofold. First, think about Newton's second law of motion. Just because one object feels more force than another does not mean that it will decelerate faster. Because the ISS is so massive (estimated 450,000 kg), the overall deceleration is very small. However, there still is definitely a noticeable deceleration of the ISS over long periods of time. The second reason the ISS isn't currently crashing into the Earth is because the ISS (like many other LEO spacecraft) will perform maneuvers to counteract the force of drag to stay in the same orbit.

Observing Atmospheric Perturbations

Problem:
The International Space Station is in a Low Earth Orbit. ISS Crew Members gently release a CubeSat from the ISS, giving the CubeSat the same orbit. If neither spacecraft performs any maneuvers, which will re-enter the atmosphere first? The elements and characteristics for the spacecraft are as follows:

<table>
<thead>
<tr>
<th></th>
<th>ISS</th>
<th>CubeSat</th>
</tr>
</thead>
</table>
© 2019 a.i. solutions, Inc
Create a new Mission Plan and save it as "ISSCubeSatLifetime.MissionPlan"

Adding in Spacecraft

- Create a new Spacecraft and name it ISS
- Give the ISS the Keplerian elements listed in the table above
- Go into "Physical Properties" on the left-hand side
- Change the drag area to 4000 m^2
- Change the mass to 450,000 kg

![Physical Properties Settings in the Spacecraft Editor](image)

- Go into "Propagator" on the left-hand side
- Change the integrator type to "Bulirsch-Stoer VOP" (This will greatly improve the computation time)
- Go to "Force Model" on the left-hand side
- Change the atmospheric density model to "Jacchia Roberts" (This allows for more accurate drag calculations)
Since most of the elements and characteristics are the same between the ISS and the CubeSat, it will be easiest to clone the ISS and adjust the necessary components.

- Clone the ISS by right-clicking it and clicking "Clone"
- Open the clone and rename it to "CubeSat"
- Go into "Physical Properties" on the left-hand side
- Change the drag area to 0.01 m^2
- Change the mass to 1.333 kg
- Click "Ok" to close the editor

Adding a PlotWindow

- Create a PlotWindow through the Object Browser
- Double-click "PlotWindow1" to open the editor
- For the "Y-Axis" drop down, change it to "ISS.Height"
- Click "More" to add another line to plot
- Change the new dropdown to "CubeSat.Height"
- Click "Ok" to close the editor

Building the Mission Sequence

- Drag and drop a "While...End" loop onto the Mission Sequence
- Change the while loop argument to "(ISS.ElapsedTime < TIMESPAN(600 days))"

- Drag and drop a FreeForm script editor inside the loop
- Change the name of the script to "Step, Update, and Check"

For this Mission Plan, we will assume that any Spacecraft below 170 km has officially "re-entered". Although many people consider 100 km to be the unofficial atmosphere/space boundary, the Spacecraft will drop very quickly after it has fallen below 170 km.

In this script, we will step both Spacecraft with an epoch sync, update the plot, and check to see if either of the Spacecraft are below 170 km. Once a Spacecraft falls below 170 km, we can stop the program entirely. To do this, we write:
// Step both Spacecraft with an epoch sync
Step ISS;
Step CubeSat to (CubeSat.Epoch == ISS.Epoch);

Update PlotWindow1;

// Checks to see if either Spacecraft gets too low
If(ISS.Height < 170 or CubeSat.Height < 170) then;
   Stop;
End;

Your Mission Sequence should look something like this:

The Mission Sequence is complete! Before you run your Mission Plan, please note that this may take a while to run. Run your mission plan, then try and answer these questions:

   Which Spacecraft re-entered first?

   About how long did it take for the first Spacecraft to re-enter?

See Also

- [Real World Modeling](#)
- [Next Topic: Solar Radiation Pressure](#)
- [Previous Topic: J2 Perturbation](#)
7.4 - Solar Radiation Pressure

In space, there is another very small force that we don't really think about but definitely can make a difference. This perturbation is called "Solar Radiation Pressure" or "SRP" for short. SRP revolves around the idea that electromagnetic waves are massless, but exhibit mass-like properties. The photons in light emitted from the Sun move at the speed of light and have momentum. Because they have momentum, when they hit other objects, they transfer momentum to the that object, giving it a boost in velocity.

However, the momentum these photons carry is extremely small. Thus, the perturbing force isn't really observed in LEO spacecraft. The best time to observe this perturbation is when a spacecraft is big, light, and moving relatively slow. A basic formula for calculating the force due to SRP is:

\[ F_{SRP} = \frac{S}{c} \cdot C_R \cdot A_s \]

\( F_{SRP} \) = SRP Force
\( \frac{S}{c} \) = Shadow function (0 if in shadow, 1 if not)
\( S \) = Solar Constant \( \left( 1367 \frac{W}{m^2} \right) \)
\( c \) = Speed of light \( \left( 300,000 \frac{km}{s} \right) \)
\( C_R \) = Coefficient of Reflectivity
\( A_s \) = Absorbing Area

Observing SRP Perturbation

**Problem:**
We have a lightweight spacecraft orbiting in a highly elliptical orbit (HEO). Propagate the spacecraft over 30 days with the effects of SRP, along with a "control" spacecraft that is identical to the first spacecraft, but does not have SRP modeled in its propagation.

- Open a new Mission Plan and save it as "SRPPerturbation.MissionPlan"

Adding in Spacecraft
• Create a new Spacecraft with the following Keplerian elements:
  o A: 42,110 km
  o E: 0.829
  o I: 0 deg
  o RAAN: 0 deg
  o W: 180 deg
  o TA: 0 deg
• Go into "Physical Properties" on the left-hand side
• Change the SRP Area to 10 m^2
• Change the mass to 100 kg

[Image of FreeFlyer spacecraft editor settings]

• Go into "Propagator" on the left-hand side
• Change the step mode to "Variable Step Size"
• Go into "Force Model" on the left-hand side
• Uncheck the box that says "Moon" under "Available Bodies" (this ensures we don't encounter unwanted perturbations)
• Check the box that says "Solar Radiation Pressure"

[Image of Solar Radiation Pressure checkbox]

• Click "Ok" to close the editor
• Right-click "Spacecraft1" and click "Clone"
• Double-click the clone to open the editor
• Rename the Spacecraft to "Spacecraft2"
• Go into "Visualization" on the left-hand side
• Change the tail color to green
• Go into "Force Model" on the left-hand side
Uncheck the "Solar Radiation Pressure" box
Click "Ok" to close the editor

Adding the ViewWindow

- Create a ViewWindow through the Object Browser
- Double Click "ViewWindow1"
- Check "Spacecraft1" and "Spacecraft2" in the Available Objects list
- Change the history mode to "Unlimited" for each Spacecraft
- Go to the "Viewpoints" section on the left-hand side
- On the default view, change the Reference Frame to "Inertial"
- Press "Ok" to close the ViewWindow editor

Building the Mission Sequence

- Drag and drop a "While...End" loop onto the Mission Sequence
- Change the while loop argument to "(Spacecraft1.ElapsedTime < TIMESPAN(30 days))"
- Drag and drop a FreeForm script editor inside the while loop
- Rename this FreeForm to "Step, Plot, and Update"

In this FreeForm, we will step each Spacecraft with an epoch sync, plot the distance between the Spacecraft, and update the view window. To do this, we write:

```csharp
// Steps each spacecraft with an epoch sync
Step Spacecraft1;
Step Spacecraft2 to (Spacecraft2.Epoch == Spacecraft1.Epoch);

// Plots the distance between each spacecraft
Plot Spacecraft1.ElapsedTime, Spacecraft1.RadialSeparation(Spacecraft2);

// Updates the ViewWindow
Update ViewWindow1;
```

Your Mission Sequence should look something like this:

<table>
<thead>
<tr>
<th>Mission Sequence</th>
<th>2 - Step, Plot, and Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
<td>Content</td>
</tr>
<tr>
<td>1</td>
<td>While (Spacecraft1.ElapsedTime &lt; TIMESPAN(30 days));</td>
</tr>
<tr>
<td>2</td>
<td>FreeForm: Step, Plot, and Update</td>
</tr>
<tr>
<td>3</td>
<td>End;</td>
</tr>
</tbody>
</table>

Save and run your Mission Plan and try to answer the following questions:

How far did the SRP satellite get away from the "control" satellite?
Go back into the Mission Sequence. To check to see if SRP was the main perturbing factor, uncheck the SRP box for Spacecraft1 in the force model. Run the mission again. What is the distance between Spacecraft now?

See Also

- Real World Modeling
- Previous Topic: Atmospheric Modeling
CHAPTER 8

Answers
Getting Started Guide

Starting Your First Mission

Running the Mission

1. Spacecraft2
2. Spacecraft1
3. 0.5 Days

Using Impulsive Burns

1. Spacecraft1 has a higher velocity right after the burn
2. Spacecraft1's variance in velocity increased

Orbital Elements Tutorial

Orbit Shapes and Sizes

Semi-Major Axis

1. Spacecraft1
2. Spacecraft3
3. Spacecraft3's period is approximately 24 hours

Eccentricity

1. Spacecraft3 and Spacecraft3
2. The peak in velocity occurred at the periapsis of the orbit
3. All of the Spacecrafts' periods are the same
4. The higher the eccentricity, the more variance in velocity

Orbit Orientation

Inclination

1. Spacecraft3
2. Spacecraft1
3. The Spacecraft travels in a retrograde orbit
Right Ascension of the Ascending Node (RAAN)

1. 12 times
2. 15 times

Argument of Perigee

1. Spacecraft4
2. The inclination needs to be 90 degrees, and the argument of perigee needs to be 270 degrees

True Anomaly

1. Approximately 100 minutes
2. Approximately 25 minutes

Spacecraft Attitude

Attitude State Representations

Euler Angles

1. Roll is a rotation about the spacecraft body X-axis, Pitch is a rotation about the spacecraft body Y-axis, Yaw is a rotation about the spacecraft body Z-axis
2. scYaw now has the same attitude motion as scRoll

Modeling the Direction Cosine Matrix and Quaternions

1. If hand calculations were correct, scEuler, scDCM, and scQuaternion were all set to the same attitude using different methods. Therefore, they should appear as one spacecraft
2. Rounding at 5 significant figures, answers should be well within 1% of actual values
3. At any given time, the components of the quaternion satisfy the constraint equation, and q₄ never becomes a negative value

Attitude Reference Frames

LVLH vs Other Reference Frames

1. LVLH
2. MJ2000

3. At apogee and perigee
4. At any time the Flight Path Angle and the Euler Angle together equal 180 degrees, showing that the change in Pitch is equivalent to the Flight Path Angle
5. X-axes are always parallel, this is because the velocity vector on a circle is always tangent to the circle at any given time, making it the same vector as the Local Horizontal

Mission Constraints on Spacecraft Attitude

Custom Mission Attitude Frame

1. Minimum is approximately 1045W - Maximum is approximately 3495W - Yes this meets the requirements
2. scToSun defines the rotation about the primary scToCanberra Vector
3. No, our sensor points toward space and we do not receive enough power

Maneuvering Tutorial

Hohmann Transfer

Hohmann Transfer - Earth Centered

3. Answers should be within 1 m/s accuracy
4. Decrease

Bi-Elliptic Transfer

Modeling the Bi-Elliptic Transfer

4. Answers should be within 1 m/s accuracy
5. $\Delta v_1 = 2.952 \text{ km/s}, \Delta v_2 = 0.775 \text{ km/s}, \Delta v_3 = -0.301 \text{ km/s}, \Sigma \Delta v = 4.029 \text{ km/s}$
6. The Hohmann transfer used 0.017 km/s more
7. Decrease

Phasing Maneuver

Modeling a Phasing Maneuver

1. Final output should look like this:
2. It used a total of 0.246 km/s less

Plane Change Maneuver

Modeling Plane Change Maneuvers

1. Answers should be within 1 m/s accuracy
2. Plane change maneuvers require less ∆v for slower spacecraft
3. At the apoapsis of the intermediate transfer orbit

Interplanetary Topics

Interplanetary Hohmann Transfer

Modeling an Interplanetary Hohmann Transfer

1. 5.596 km/s
2. 259.32 days
3. Approximately 14.24 km

Patched Conics Transfer

Modeling a Patched Conics Transfer

1. Δv₁ = 3.485 km/s, Δv₂ = -2.254 km/s
2. $\Sigma \Delta v = 5.739 \text{ km/s}$
3. 280869.54 km

Gravity Assist

Modeling a Gravity Assist

1. BackSideSC
2. The Spacecraft exited in the same general direction as the Mars velocity vector
3. The magnitudes are within 0.040 km/s of each other. Their orbits are so drastically different because they had different exit directions in reference to the planet's velocity.

The B-Plane

Modeling the B-Plane

1. $<9142.492, -4264.283, -3147.362>$ km
2. 4.725 degrees
3. Answer should be within 1 meter

Targeting Tutorial

Inclination Change

Use Targeting to Model an Inclination Change

1. Calculations by hand should match with FreeFlyer's calculations. You can solve for the initial velocity using the circular velocity formula. Solutions:
   - $\Delta v_n$: 1.1417 km/s - notice that FreeFlyer represents this as negative, because it is opposite the velocity direction
   - $\Delta v_i$: 3.6213 km/s
   - Total $\Delta v$: 3.797 km/s
2. Achieving goals for SMA and eccentricity instead of circular velocity should yield the same results.
3. Coming from a non-equatorial orbit would require the burn to be performed at either the ascending or descending node. We would need to step our spacecraft to one of these positions in its orbit before executing the targeting loop in order to achieve a proper inclination change.
4. The Targeting loop in FreeForm script should look like this:

```plaintext
Target;

Iterate Spacecraft1;

// Elements we can vary to achieve the desired inclination
Vary ImpulsiveBurn1.BurnDirection[0] = 0 + 0.001;
Vary ImpulsiveBurn1.BurnDirection[1] = 1 + 0.001;
```
// Perform the maneuver
Maneuver Spacecraft1 using ImpulsiveBurn1;

// Goals to achieve with the maneuver
Achieve Spacecraft1.I = 35 +/- 0.01;
Achieve Spacecraft1.VMag = initialVelocity +/- 0.001;
End;

B-Plane Targeting

Targeting Mars's B-Plane

1. Angle between B and T: 114.923 degrees. B magnitude: 8004.149 km. Both are within their respective tolerances.
2. We only burn in the V direction because our B-plane targeting means we are already in the correct plane to achieve a polar orbit. We just need to slow down so that we do not fly by Mars. If we allow the other two burn directions to vary, their magnitudes are negligible.

Real World Modeling

Multi-Body Effects

Modeling Two Body, Three Body, and a Multi-Body Problem

1. TwoBodySC
2. ThreeBodySC crashed into the Earth. The Sun perturbed its orbit resulting in a trajectory impacting the Earth.
3. The shape of the orbit is a "figure 8"
4. MultiBodySC; It shows that sometimes there are forces that cannot be factored in easily without assistance from computers.

J2 Perturbation

Modeling a Sun-Synchronous Orbit

1. Approximately 365 days
2. No because the force model does not factor in the effects of the oblate Earth

Atmospheric Modeling

Observing Atmospheric Perturbation

1. The ISS
2. Approximately 508 days

**Solar Radiation Pressure**

Observing SRP Perturbation

1. Approximately 215 km
2. Approximately 0.000 km

**Image Sources**


[Bi-Elliptic Transfer Δv Requirements] Retrieved July 6, 2015 from [http://i.imgur.com/oXbUN33.png](http://i.imgur.com/oXbUN33.png)
