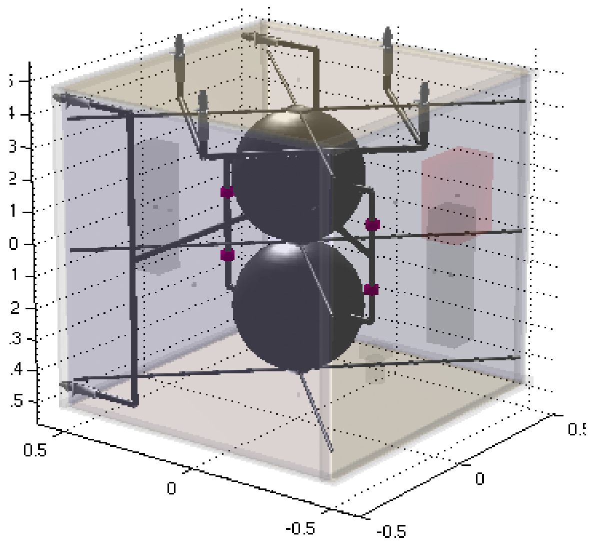
# CubeSat Toolbox

User's Guide

Release 2019.1



CubeSat Toolbox

a member of the Spacecraft Control Toolbox product family

January 14, 2020

**Princeton Satellite Systems, Inc.**
6 Market St. Suite 926
Plainsboro, New Jersey 08536

Technical Support/Sales/Info: http://www.psatellite.com

# Contents

# List of Figures

CHAPTER 1

# INTRODUCTION

This chapter shows you how to install the CubeSat Toolbox and how it is organized.

## 1.1 Organization

The CubeSat Toolbox is composed of MATLAB m-files and mat-files, organized into a set of modules by subject. It is essentially a library of functions for analyzing spacecraft and missions. The CubeSat toolbox is a subset of the Spacecraft Control Toolbox, which supports a set of scripts for analyzing mission planning, attitude control, and simulation of nano satellites. The Spacecraft Control Toolbox is composed of a set of modules including *CubeSat* , and that organization is preserved in the CubeSat Toolbox.

There is a substantial set of software which the Spacecraft Control Toolbox shares with the Aircraft Control Toolbox, and this software is in a module called *Common*. The core spacecraft analysis functions are in *SC* along with an animation tool in *Plotting*. These modules with *CubeSat* are referred to together as the *Core* toolbox. All of the Spacecraft Control Toolbox modules are described in the following table, including the add-on modules which are purchased separately.

**Table 1.1:** Spacecraft Control Toolbox Modules

| **Module** | | **Function** |
|---|---|---|
| CubeSat | | CubeSat and nanonsatellite modeling |
| Common | | Coordinate transformations, math, control, CAD tools, time conversions, graphics and general utilities |
| Plotting | | Plotting tools for complex simulations including animation |
| SC | | Attitude dynamics, pointing budgets, basic orbit dynamics, environment, sample CAD models, ephemeris, sensor and actuator modeling. |
| SCPro | | Additional high-fidelity models for environment, sensors, actuators. |
| AttitudeControl | | In-depth attitude control system design examples, including the hypothetical geosynchronous satellite ComStar |
| Estimation | | Attitude and orbit estimation. Stellar attitude determination and Kalman filtering. |
| Imaging | | Image processing functions |
| Link | | Basic RF and optical link analysis. |
| Orbit | | Orbit mechanics, maneuver planning, fuel budgets, and high-fidelity simulation. |
| Power | | Basic power modeling |
| Propulsion | | Electric and chemical propulsion. Launch vehicle analysis. |
| Thermal | | Basic thermal modeling |
| Launch Vehicle | Add-On | Launch vehicle design |
| Formation Flying | Add-On | Formation flying control |
| Fusion Propulsion | Add-On | Fusion propulsion analysis |

| | | |
|---|---|---|
| SAAD | Add-On | Spin axis attitude determination |
| Solar Sail | Add-On | Solar sail control and mission analysis |

The *Formation Flying*, *Solar Sail*, and *SAAD* add-on modules have their own user's guides. These modules can be purchased separately but they require the Professional Edition of the toolbox.

## 1.2 Requirements

MATLAB 7.0 at a minimum is required to run all of the functions. Most of the functions will run on previous versions but we are no longer supporting them.

## 1.3 Installation

The preferred method of delivering the toolbox is a download from the Princeton Satellite Systems website. Put the folder extracted from the archive anywhere on your computer. There is no "installer" application to do the copying for you. We will refer to the folder containing your modules as PSSToolboxes. If you later purchase an add-on module, you would simply add it to this folder.

All you need to do now is to set the MATLAB path to include the folders in PSSToolboxes. We recommend using the supplied function `PSSSetPaths.m` instead of MATLAB's path utility. From the MATLAB prompt, cd to your PSSToolboxes folder and then run `PSSSetPaths`. For example:

```
>> cd /Users/me/PSSToolboxes
>> PSSSetPaths
```

This will set all of the paths for the duration of the session, with the option of saving the new path for future sessions.

## 1.4 Getting Started

The first two functions that you should try are `DemoPSS` and `FileHelp`. Each toolbox or module has a Demos folder and a function `DemoPSS`. Do not move or remove this function from any of your modules! `DemoPSS.m` looks for other `DemoPSS` functions to determine where the demos are in the folders so it can display them in the `DemoPSS` GUI.

The `FileHelp` function provides a graphical interface to the MATLAB function headers. You can peruse the functions by folder to get a quick sense of your new product's capabilities and search the function names and headers for keywords. `FileHelp` and `DemoPSS` provide the best way to get an overview of the Spacecraft Control Toolbox. Both are described in more detail in the next chapter.

Demos and Functions can also be browsed by MATLAB's built-in help system. sThis allows for the same searching and browsing capabilities as `DemoPSS` and `FileHelp`.

CHAPTER 2

# GETTING HELP

This chapter shows you how to use the help systems built into PSS Toolboxes. There are several sources of help. Our toolboxes are now integrated into MATLAB's built-in help browser. Then, there is the MATLAB command line help which prints help comments for individual files and lists the contents of folders. Then, there are special help utilities built into the PSS toolboxes: one is the file help function, the second is the demo functions and the third is the graphical user interface help system. Additionally, you can submit technical support questions directly to our engineers via email.

## 2.1 MATLAB's Built-in Help System

### 2.1.1 Basic Information and Function Help

Our toolbox information can now be found in the MATLAB help system. To access this capability, simply open the MATLAB help system. As long as the toolbox is in the MATLAB path, it will appear in the contents pane. In more recent versions of MATLAB, you need to navigate to Supplemental Software from the main window, as shown in Figure 2.1 on the following page. The index page of the SCT documentation is shown in Figure 2.2 on page 5.

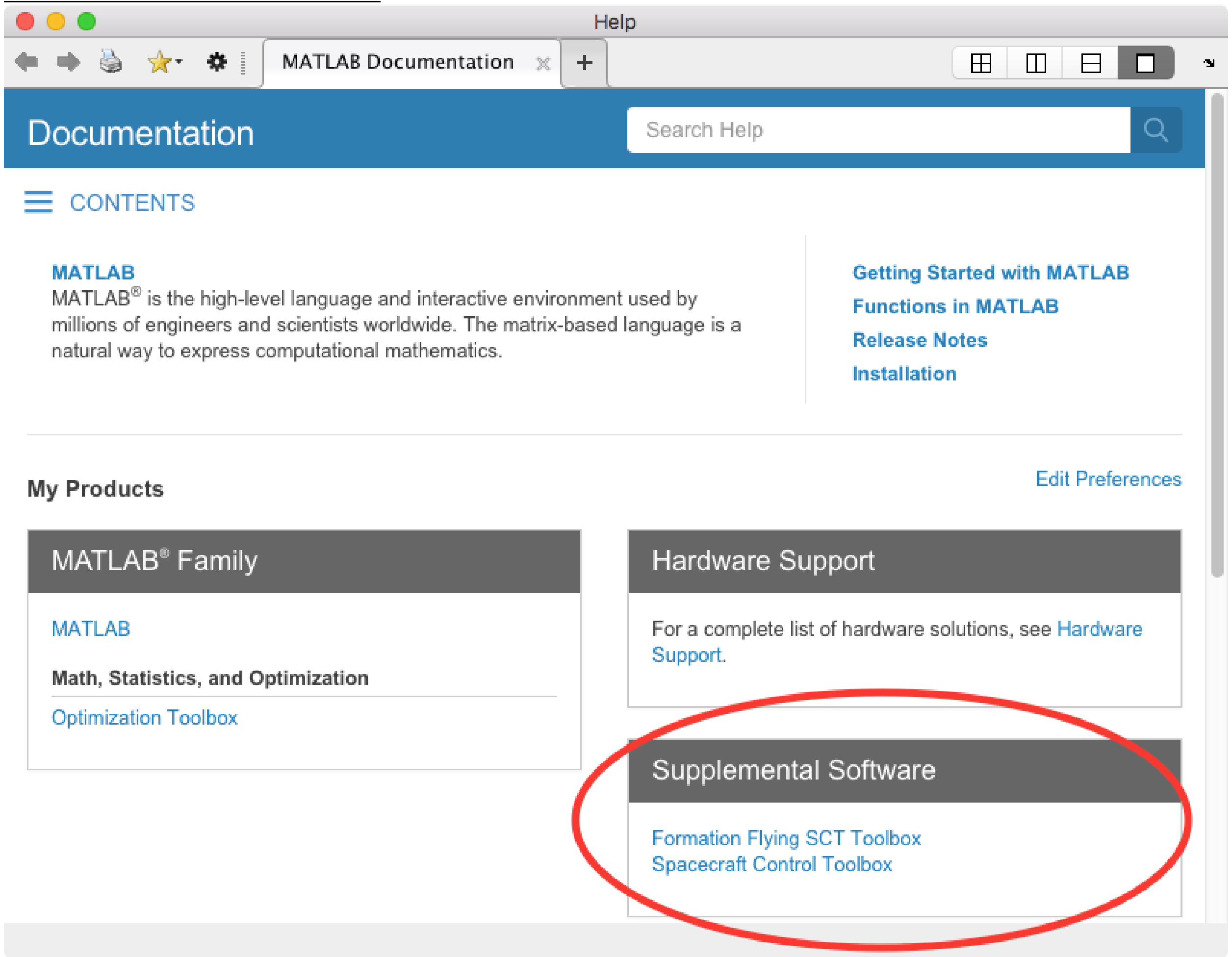
The help window from R2011b and earlier is depicted in Figure 2.3 on page 5.

This contains a lot information on the toolbox. It also allows you to search for functions as you would if you were searching for functions in the MATLAB root.

### 2.1.2 Published Demos

Another feature that has been added to the MATLAB help structure is the access to all of the toolbox demos. Every single demo is now listed, according to module and the folder. These can be found under the *Other Demos* or *Examples* portion of the Contents Pane. Each demo has its own webpage that goes through it step by step showing exactly what the script is doing and which functions it is calling. From each individual demo webpage you can also run the script to view the output, or open it in the editor. Note that you might want to save any changes to the demo under a new file name so that you can always have the original. Below is an example of demo page displayed in MATLAB help that shows where to find the toolbox demos as well as the the hierarchal structure used for browsing the demos.

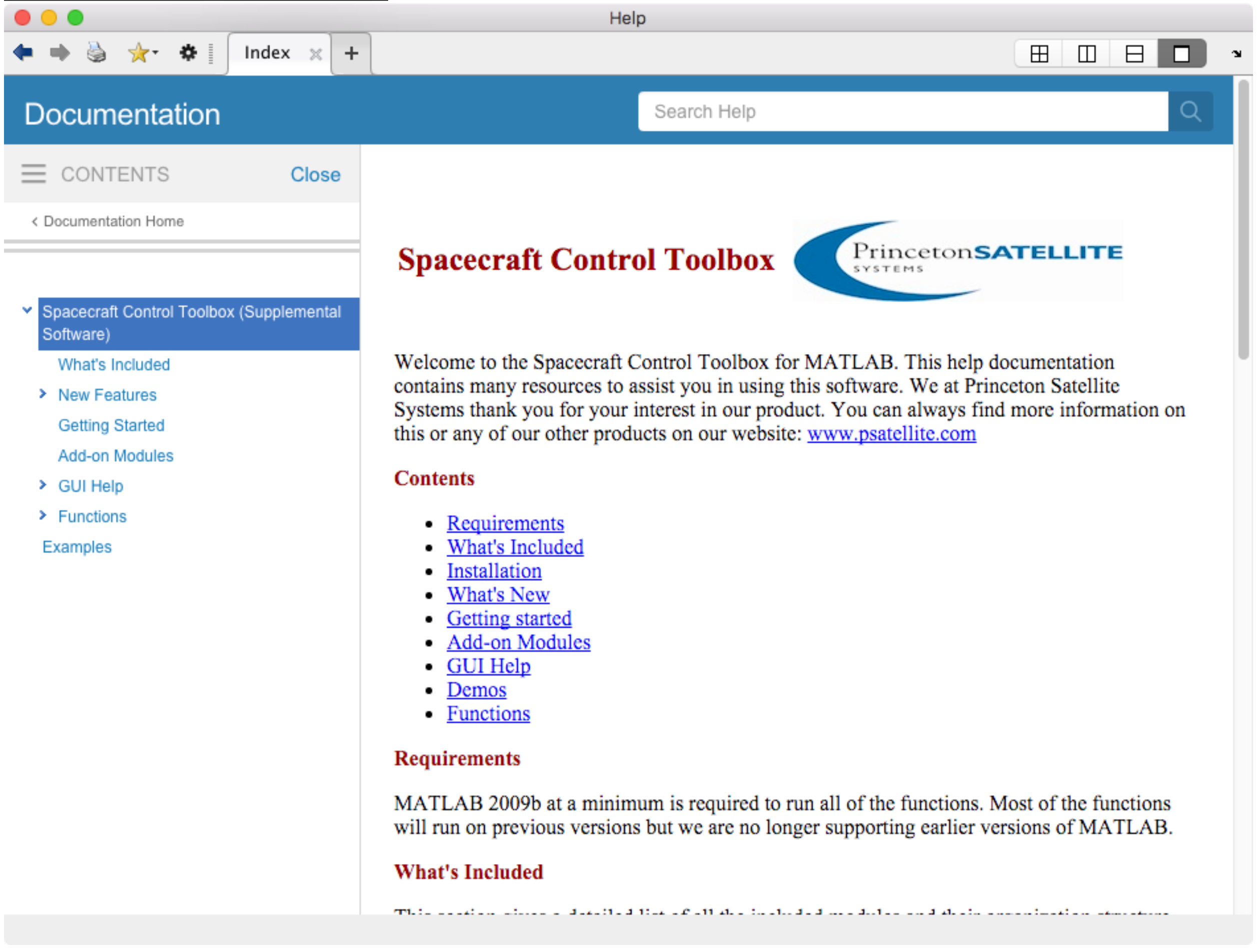
**Figure 2.1:** MATLAB Help - Supplemental Software



**MATLAB**
MATLAB® is the high-level language and interactive environment used by millions of engineers and scientists worldwide. The matrix-based language is a natural way to express computational mathematics.

For a complete list of hardware solutions, see Hardware Support.

**Figure 2.2:** Toolbox Documentation Main Page, R2016b



**Spacecraft Control Toolbox**

Welcome to the Spacecraft Control Toolbox for MATLAB. This help documentation contains many resources to assist you in using this software. We at Princeton Satellite Systems thank you for your interest in our product. You can always find more information on this or any of our other products on our website: www.psatellite.com

**Contents**

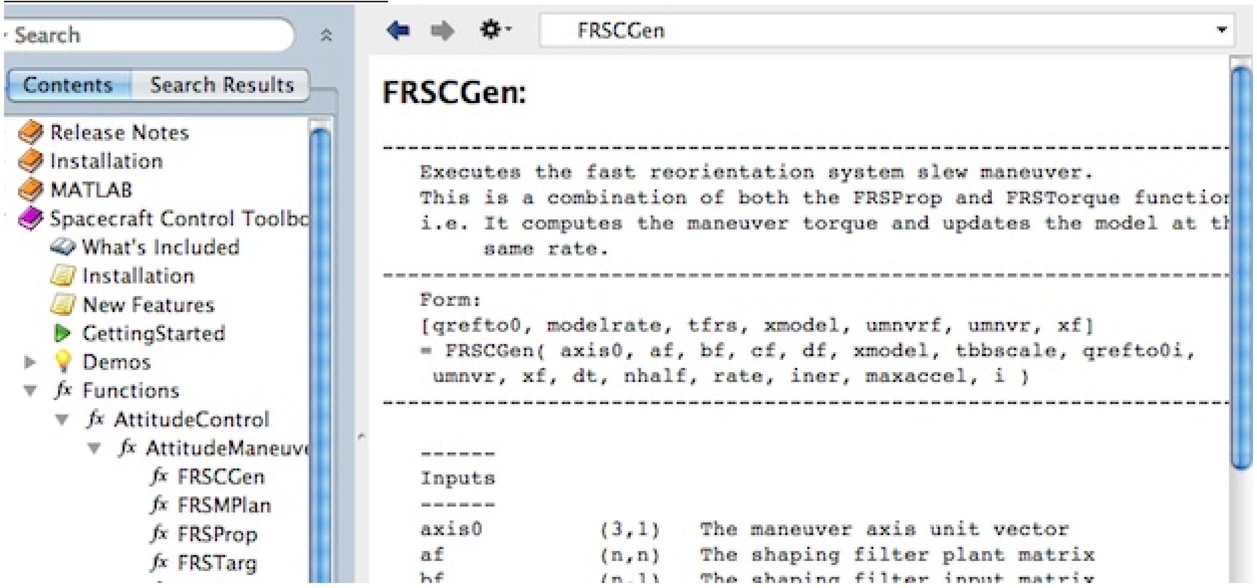
- Requirements
- What's Included
- Installation
- What's New
- Getting started
- Add-on Modules
- GUI Help
- Demos
- Functions

**Requirements**

MATLAB 2009b at a minimum is required to run all of the functions. Most of the functions will run on previous versions but we are no longer supporting earlier versions of MATLAB.

**What's Included**

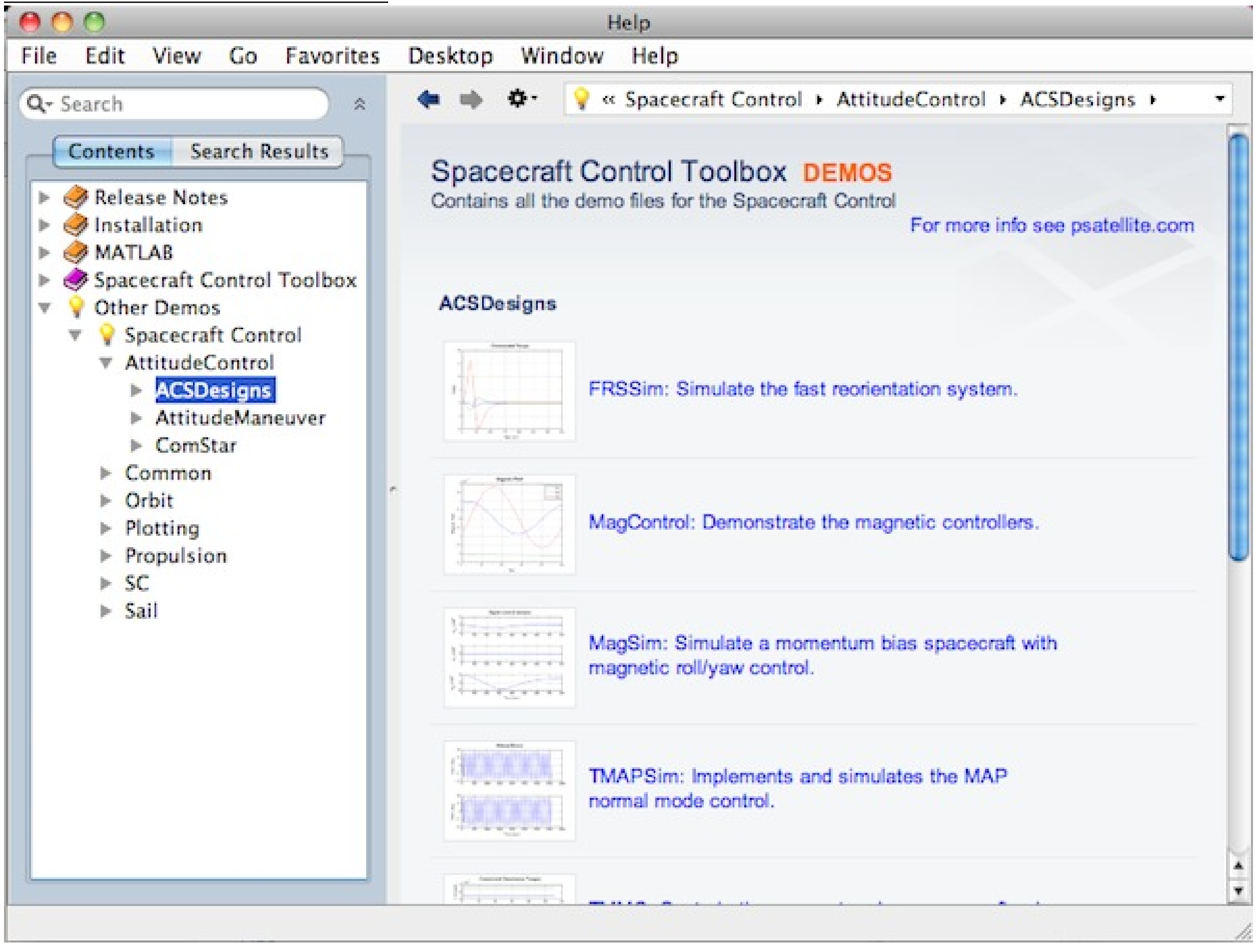
**Figure 2.3:** Toolbox Documentation, R2011b



**FRSCGen:**

```
-----------------------------------------------------------------------
   Executes the fast reorientation system slew maneuver.
   This is a combination of both the FRSProp and FRSTorque functio
   i.e. It computes the maneuver torque and updates the model at th
        same rate.
-----------------------------------------------------------------------
   Form:
   [qrefto0, modelrate, tfrs, xmodel, umnvrf, umnvr, xf]
   = FRSCGen( axis0, af, bf, cf, df, xmodel, tbbscale, qrefto0i,
    umnvr, xf, dt, nhalf, rate, iner, maxaccel, i )
-----------------------------------------------------------------------

   ------
   Inputs
   ------
   axis0          (3,1)    The maneuver axis unit vector
   af             (n,n)    The shaping filter plant matrix
```

**Figure 2.4:** Toolbox Demos



Spacecraft Control Toolbox DEMOS

Contains all the demo files for the Spacecraft Control

For more info see psatellite.com

ACSDesigns

FRSSim: Simulate the fast reorientation system.

MagControl: Demonstrate the magnetic controllers.

MagSim: Simulate a momentum bias spacecraft with magnetic roll/yaw control.

TMAPSim: Implements and simulates the MAP normal mode control.

## 2.2 Command Line Help

You can get help for any function by typing

```
>> help functionName
```

For example, if you type

```
>> help C2DZOH
```

you will see the following displayed in your MATLAB command window:

```
 Create a discrete time system using a zero order hold.

   Create a discrete time system from a continuous system
   assuming a zero-order-hold at the input.

   Given
   .
   x = ax + bu

   Find f and g where

   x(k+1) = fx(k) + gu(k)

-------------------------------------------------------------------------
   Form:
   [f, g] = C2DZOH( a, b, T )
-------------------------------------------------------------------------

   ------
   Inputs
   ------
   a             (n,n)  Continuous plant matrix
   b             (n,m)  Input matrix
   T             (1,1)  Time step

   -------
   Outputs
   -------
   f             (n,n)  Discrete plant matrix
   g             (n,m)  Discrete input matrix

-------------------------------------------------------------------------
   References: Van Loan, C.F., Computing Integrals Involving the Matrix
               Exponential, IEEE Transactions on Automatic Control
               Vol. AC-23, No. 3, June 1978, pp. 395-404.
-------------------------------------------------------------------------
```

All PSS functions have the standard header format shown above. Keep in mind that you can find out which folder a function resides in using the MATLAB command `which`, i.e.

```
>> which C2DZOH
Core/Common/Control/C2DZOH.m
```

When you want more information about a folder of interest, you can get a list of the contents in any directory by using the `help` command with a folder name. The returned list of files is organized alphabetically. For example,

```
>> help Atmosphere
  Common/Atmosphere
  S
     SimpAtm - Simplified atmosphere model.
     StdAtm  - Computes atmospheric density based on the standard atmosphere model.
```

If there is a folder with the same name in a Demos directory, the demos will be listed separately. For example,

```
>> help Plugins

  Common/Plugins

  T
     Telemetry                        - Generates a set of telemetry pages.
     TelemetryOffline                 - This plots telemetry files previously saved
                                    by Telemetry.
     TelemetryPlot                    - Plot real time in a single window.
     TimePlugIn                       - Create a time GUI plug in.

  Common/Demos/Plugins

  T
     TelemetryDemo                    - Demonstrate the Telemetry function.
```

In the case of a demo, the command line help will provide a link to the published HTML for that demo, if any exists. Any functions referenced on a See also line will have dynamic links, which will show the help for that function.

```
>> help FFSimDemo Demonstrate the use of FFSim to analyze disturbance effects.
------------------------------------------------------------------------ See also
FFSim, FFSimPlotter, Goals2DeltaElem -----------------------------------------------
-------------------------------------------------------------------------- Copyright
(c) 2009 Princeton Satellite Systems, Inc.  All rights reserved.  ------------------------------
Since version 8.  ------------------------------------------------------------------------
Published output in the Help browser showdemo FFSimDemo
```

To see the entire contents of a file at the command line, use `type`.

Command line help also works with higher level directories, for instance if you ask for help on the Common directory, you will get a list of all the subdirectories.

```
>> help Common
  PSS Toolbox Folder Common
  Version 2019.1      23-Dec-2019

  Directories:
  CommonData
  ComponentModels
  Control
  Database
  FileUtils
  GUIs
  General
  Graphics
  Help
  MassProperties
  Materials
  Plugins
  Quaternion
  Time
  Transform
```

The function `ver` lists the current version of all your installed toolboxes. Each Core module that you have installed will be listed separately. For instance,

```
->> ver
--------------------------------------------------------------------------------
MATLAB Version: 9.6.0.1072779 (R2019a)
MATLAB License Number: 346509
Operating System: Mac OS X  Version: 10.15.2 Build: 19C57
Java Version: Java 1.8.0_181-b13 with Oracle Corporation Java HotSpot(TM) 64-Bit
    Server VM mixed mode
--------------------------------------------------------------------------------
MATLAB                                                Version 9.6         (R2019a
    )
Deep Learning Toolbox                                 Version 12.1        (R2019a
    )
Image Acquisition Toolbox                             Version 6.0         (R2019a
    )
Image Processing Toolbox                              Version 10.4        (R2019a
    )
Instrument Control Toolbox                            Version 4.0         (R2019a
    )
Optimization Toolbox                                  Version 8.3         (R2019a
    )
PSS Toolbox Folder AerospaceUtils                     Version 2019.1
PSS Toolbox Folder Common                             Version 2019.1
PSS Toolbox Folder CubeSat                            Version 2019.1
PSS Toolbox Folder Electrical                         Version 2019.1
PSS Toolbox Folder FormationFlying                    Version 2019.1
PSS Toolbox Folder Link                               Version 2019.1
PSS Toolbox Folder Math                               Version 2019.1
PSS Toolbox Folder Orbit                              Version 2019.1
PSS Toolbox Folder Plotting                           Version 2019.1
PSS Toolbox Folder SC                                 Version 2019.1
PSS Toolbox Folder SCPro                              Version 2019.1
```

## 2.3 FileHelp

### 2.3.1 Introduction

When you type

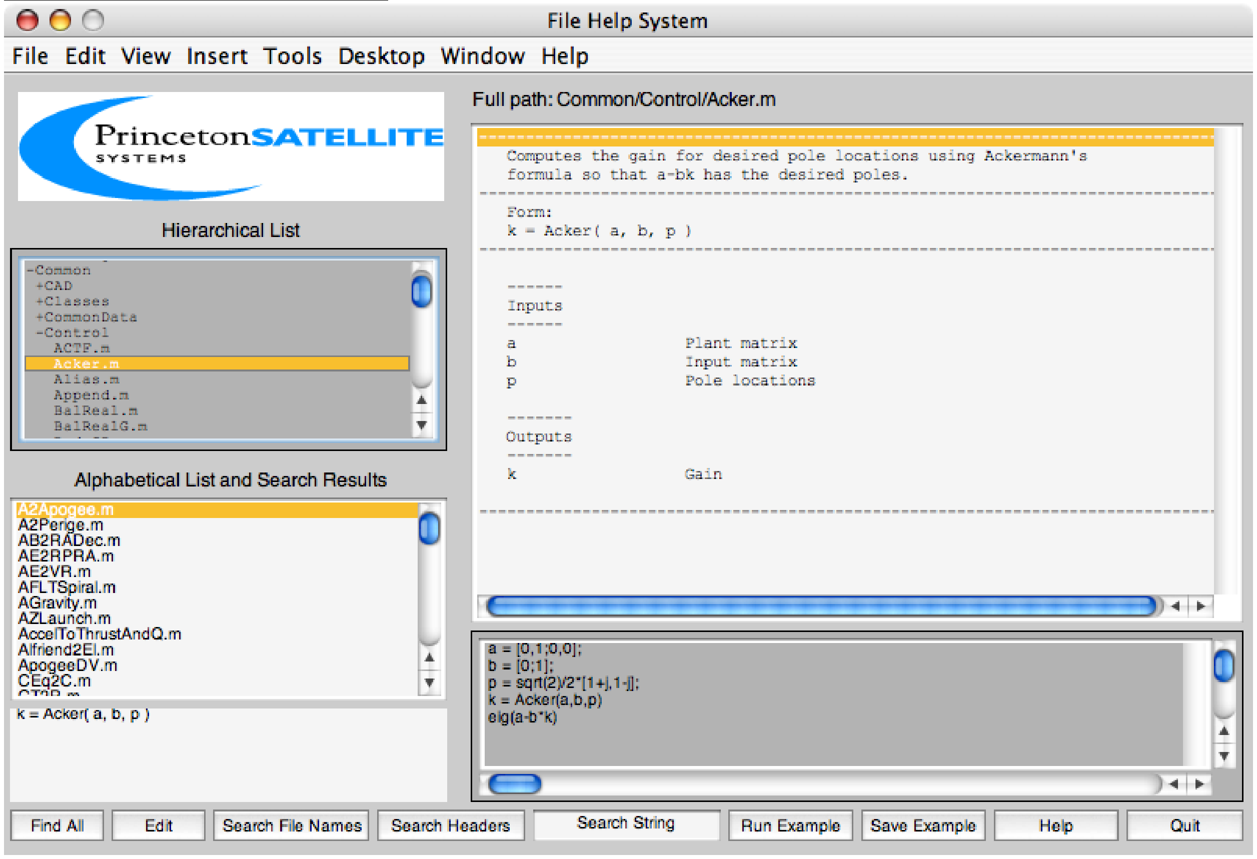
```
FileHelp
```

the FileHelp GUI appears, Figure 2.5 on the next page.

There are five main panes in the window. On the left hand side is a display of all functions in the toolbox arranged in the same hierarchy as the PSSToolboxes folder. Scripts, including most of the demos, are not included. Below the hierarchical list is a list in alphabetical order by module. On the right-hand-side is the header display pane. Immediately below the header display is the editable example pane. To its left is a template for the function. You can cut and paste the template into your own functions.

The buttons along the bottom provide additional controls along with the search feature. Select the "Search String" text and replace it with your own text, for example "sun". Then click either the Search File Names button or Search Headers.

**Figure 2.5:** The file help GUI



Full path: Common/Control/Acker.m

```
-Common
 +CAD
 +Classes
 +CommonData
 -Control
   ACTF.m
   Acker.m
   Alias.m
   Append.m
   BalReal.m
   BalRealG.m
```

```
A2Apogee.m
A2Perige.m
AB2RADec.m
AE2RPRA.m
AE2VR.m
AFLTSpiral.m
AGravity.m
AZLaunch.m
AccelToThrustAndQ.m
Alfriend2El.m
ApogeeDV.m
CEq2C.m
```

```
k = Acker( a, b, p )
```

```
-------------------------------------------------------------------------------
  Computes the gain for desired pole locations using Ackermann's
  formula so that a-bk has the desired poles.
-------------------------------------------------------------------------------
  Form:
  k = Acker( a, b, p )
-------------------------------------------------------------------------------

  ------
  Inputs
  ------
  a                 Plant matrix
  b                 Input matrix
  p                 Pole locations

  -------
  Outputs
  -------
  k                 Gain

-------------------------------------------------------------------------------
```

```
a = [0,1;0,0];
b = [0;1];
p = sqrt(2)/2*[1+j,1-j];
k = Acker(a,b,p)
eig(a-b*k)
```

### 2.3.2 The List Pane

If you click a file in the alphabetical or hierarchical lists, the header will appear in the header pane. This is the same header that is in the file. The headers are extracted from a .mat file so changes you make will not be reflected in the file. In the hierarchical list, any name with a + or - sign is a folder. Click on the folders until you reach the file you would like. When you click a file, the header and template will appear.

### 2.3.3 Edit Button

This opens the MATLAB edit window for the function selected in the list.

### 2.3.4 The Example Pane

This pane gives an example for the function displayed. Not all functions have examples. The edit display has scroll bars. You can edit the example, create new examples and save them using the buttons below the display. To run an example, push the Run Example button. You can include comments in the example by using the percent symbol.

### 2.3.5 Run Example Button

Run the example in the display. Some of the examples are just the name of the function. These are functions with built-in demos. Results will appear either in separate figure windows or in the MATLAB Command Window.

### 2.3.6 Save Example Button

Save the example in the edit window. Pushing this button only saves it in the temporary memory used by the GUI. You can save the example permanently when you Quit.

### 2.3.7 Help Button

Opens the on-line help system.

### 2.3.8 Quit

Quit the GUI. If you have edited an example, it will ask you whether you want to save the example before you quit.

## 2.4 Searching in File Help

### 2.4.1 Search File Names Button

Type in a function name in the edit box and push the button called Search File Names.

### 2.4.2 Find All Button

Find All returns to the original list of the functions. This is used after one of the search options has been used.

### 2.4.3 Search Headers Button

Search headers for a string. This function looks for exact, but not case sensitive, matches. The file display displays all matches. A progress bar gives you an indication of time remaining in the search.

### 2.4.4 Search String Edit Box

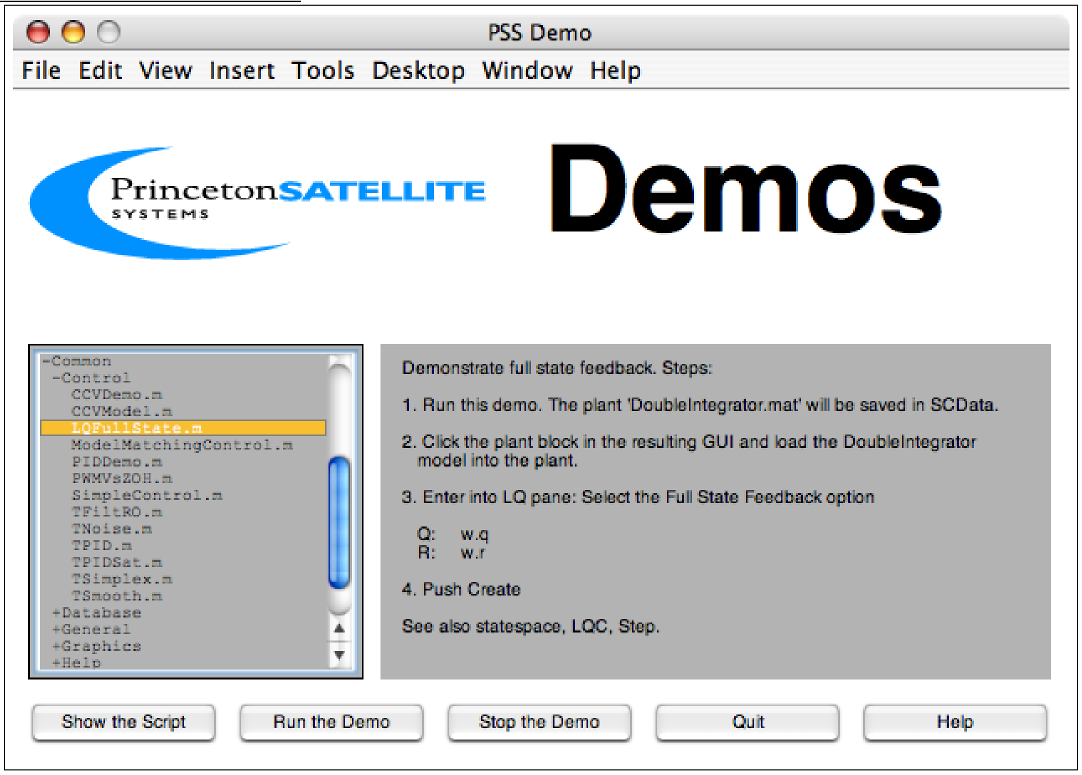
This is the search string. Spaces will be matched so if you type "attitude control" it will not match "attitude  control" (with two spaces.)

## 2.5 DemoPSS

If you type `DemoPSS` you will see the GUI in Figure 2.6. This predates MATLAB's built-in help feature and provides an easy way to run the scripts provided in the toolbox. The list on the left-hand-side is hierarchical and the top level follows the organization of your toolbox modules. Most folders in your modules have matching folders in Demos with scripts that demonstrate the functions. The GUI checks to see which directories are in the same directory as `DemoPSS` and lists all directories and files. This allows you to add your own directories and demo files.

Click on the first name to open the directory. The + sign changes to - and the list changes. Figure 2.6 shows the Common/Control folder in the core toolbox. The hierarchical menu shows the highest level folders.

**Figure 2.6:** The demo GUI


PSS Demo
File Edit View Insert Tools Desktop Window Help
PrincetonSATELLITE SYSTEMS
Demos
Show the Script
Run the Demo
Stop the Demo
Quit
Help

```
-Common
 -Control
   CCVDemo.m
   CCVModel.m
   LQFullState.m
   ModelMatchingControl.m
   PIDDemo.m
   PWMVsZOH.m
   SimpleControl.m
   TFiltRO.m
   TNoise.m
   TPID.m
   TPIDSat.m
   TSimplex.m
   TSmooth.m
 +Database
 +General
 +Graphics
 +Help
```

Demonstrate full state feedback. Steps:

1. Run this demo. The plant 'DoubleIntegrator.mat' will be saved in SCData.
2. Click the plant block in the resulting GUI and load the DoubleIntegrator model into the plant.
3. Enter into LQ pane: Select the Full State Feedback option

   Q: w.q
   R: w.r

4. Push Create

See also statespace, LQC, Step.

Your own demos will appear if they are put in any of the Demos folders. If you would like to look at, or edit, the script, push Show the Script.

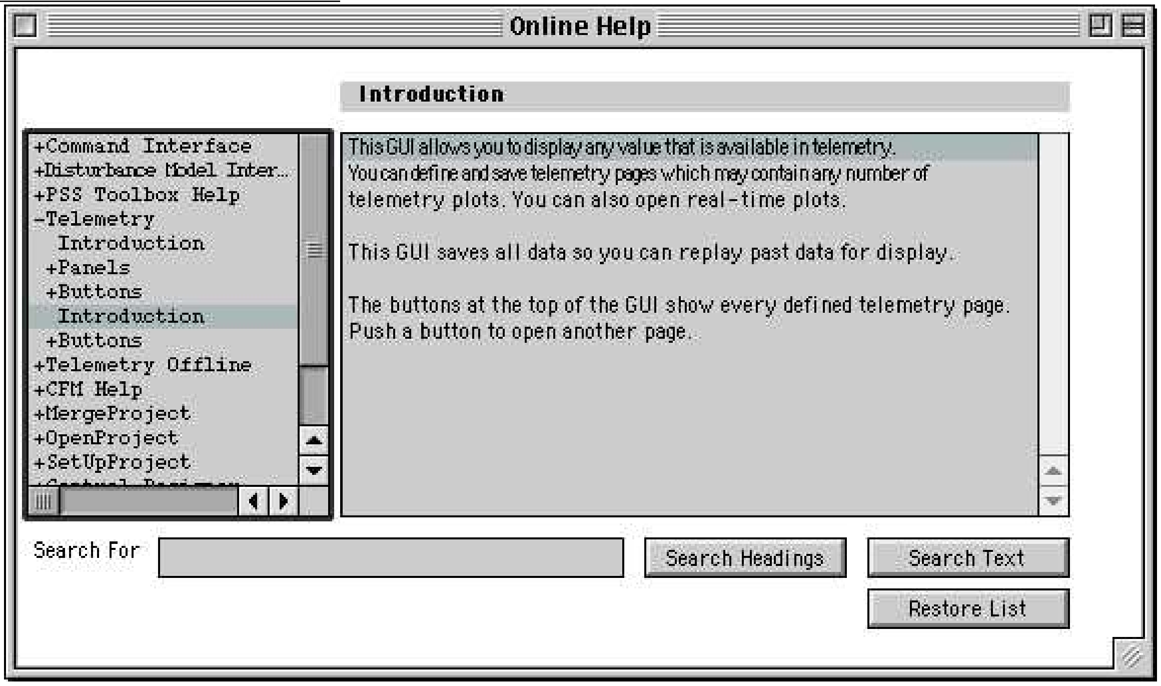
You can also access the published version of the demos using MATLAB's help system. On recent versions this is access by selecting Supplemental Software from the main Help window, and then selecting Examples.

## 2.6 Graphical User Interface Help

Each graphical user interface (GUI) has a help button. If you hit the help button a new GUI will appear. You can access on-line help about any of the GUIs through this display. It is separate from the file help GUI described above. The same help is also available in HTML through the MATLAB help window.

The `HelpSystem` display is hierarchical. Any list item with a + or - in front is a help heading with multiple subtopics. + means the heading item is closed, - means it is open. Clicking on a heading name toggles it open or closed. Figure 2.7 shows the display with the Telemetry help expanded. If you click on a topic in the list you will get a text display in the right-hand pane. You can either search the headings or the text by entering a text string into the Search For edit box

**Figure 2.7:** On-line Help



This GUI allows you to display any value that is available in telemetry. You can define and save telemetry pages which may contain any number of telemetry plots. You can also open real-time plots.

This GUI saves all data so you can replay past data for display.

The buttons at the top of the GUI show every defined telemetry page. Push a button to open another page.

and hitting the appropriate button. Restore List restores the list window to its previous configuration.

## 2.7 Technical Support

Contact support@psatellite.com for free email technical support. We are happy to add functions and demos for our customers when asked.

CHAPTER 3

# BASIC FUNCTIONS

This chapter shows you how to use a sampling of the most basic CubeSat Control Toolbox functions.

## 3.1 Introduction

The CubeSat Control Toolbox is composed of several thousand MATLAB files. The functions cover attitude control and dynamics, computer aided design, orbit dynamics and kinematics, ephemeris, actuator and sensor modeling, and thermal and mathematics operations. Most of the functions can be used individually although some are rarely called except by other toolbox functions.

This chapter will review some basic features built into the CCT functions and highlight some examples from the folder that you will use most frequently.

## 3.2 Function Features

### 3.2.1 Introduction

Functions have several features that are helpful to understand. Features that are available in the functions are listed in Table 3.1.

**Table 3.1:** Features in Spacecraft Control Toolbox functions

| Features |
|---|
| Built-in demos |
| Default parameters |
| Built-in plotting |
| Error checking |
| Variable inputs |

These are illustrated in the examples given below.

### 3.2.2 Built-in demos

Many functions have built in demos. A function with a built-in demo requires no inputs and produces a plot or other output for a range of input parameters to give you a feel for the function.

An example of a function with a built-in demo is `AnimQ`. It creates an example array of quaternions and animates the resulting body axes. The inputs for the built-in demo are generally specified near the top of the function so it is easy to check for one by looking at the code. Plots are produced at the bottom of a function.

### 3.2.3 Default parameters

Most functions have default parameters. There are two ways to get default parameters. If you pass an empty matrix, i.e.

```
[]
```

as a parameter the function will use a default parameter if defaults are available. This is only necessary if you wish to use a default for one parameter and input the value for the next input. For example, `EarthRot` takes a date in Julian centuries as the first parameter and a flag for equation of the equinoxes as the second parameter. If you look in the function, you will see that the default is to use the current date.

```
>> EarthRte
ans =
     7.292115855392341e-05
```

You should never hesitate to look in functions to see what defaults are available and what the values are. Defaults are always treated at the top of the function just under the header. Remember that the unix command `type` works in MATLAB to display a function's contents, for instance

```
function w = EarthRte( jD )

%% Computes the mean earth rate.
%   See also MSidDay.
%
%--------------------------------------------------------------------------
%   Form:
%   w = EarthRte( jD )
%--------------------------------------------------------------------------
%
%   ------
%   Inputs
%   ------
%   jD                (1,1)    Julian date (day)
%
%   -------
%   Outputs
%   -------
%   w                 (1,1)     Mean earth rate (rad/sec)
%
%--------------------------------------------------------------------------
%   References: The Astronomical Almanac for the Year 1993,
%               U.S. Government, Printing Office, 1993, p. B6.
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
%   Copyright (c) 1993 Princeton Satellite Systems, Inc.
%   All rights reserved.
%--------------------------------------------------------------------------
%   Since version 1.
%--------------------------------------------------------------------------

if( nargin == 0 )
  jD = Date2JD;
```

```
end

w = 2*pi/(86400*MSidDay(jD));

%---------------------------------------
% $Date: 2017-05-11 14:11:01 -0400 (Thu, 11 May 2017) $
% $Revision: 44563 $
```

where we have excerpted the code creating the default input.

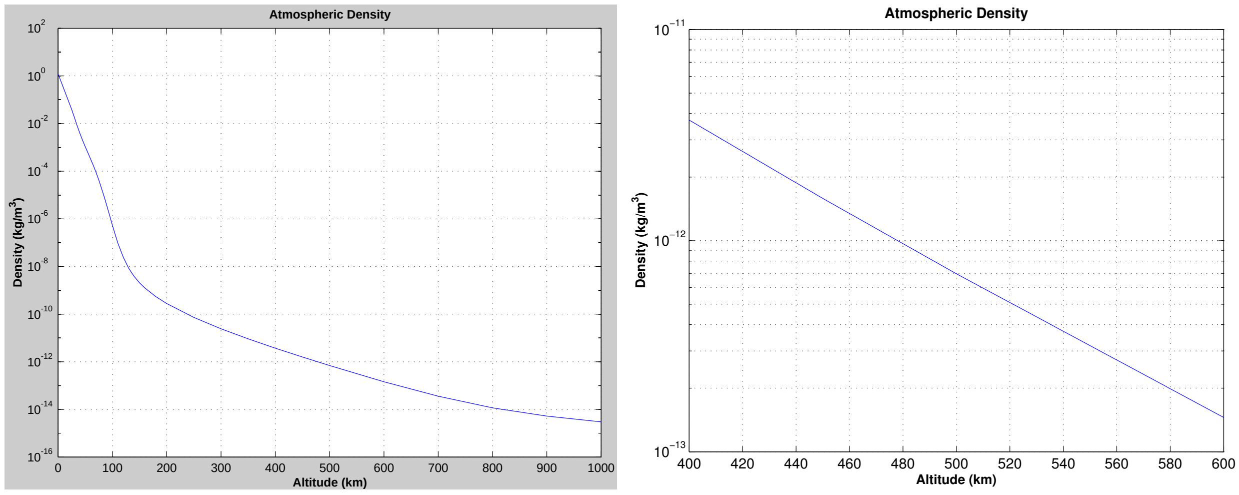
```
if( nargin == 0 )
  jD = Date2JD;
end
```

### 3.2.4 Built-in plotting

Many of the functions in the toolbox will plot the results if there are no output arguments. In many cases, you do not need any input arguments to get useful plots due to the built in defaults, but you can also generate plots with your own inputs. Calling for example `AtmDens2` by itself will generate a plot of the atmospheric density as shown in Figure 3.1. If no inputs are given it automatically computes the density for a range of altitudes. If you pass in a vector such as a smaller range of altitudes, you get the same plot with your input, such as

```
AtmDens2(linspace(400,600))
```

**Figure 3.1:** Atmospheric density from `AtmDens2`



### 3.2.5 Error checking

Many functions perform error checking. However, functions that are designed to be called repeatedly, for example the right-hand-side of a set of differential equations tend not have error checking since the impact on performance would be significant. In that case, if you pass it invalid inputs you will get a MATLAB error message.

### 3.2.6 Variable inputs

Some functions can take different kinds of inputs. An example is `Date2JD`. You can pass it either an array

```
[ year month day hour minute seconds ]
```

or the data structure

```
d.month
d.day
d.year
d.hour
d.minute
d.second
```

The options are listed in the header.

## 3.3 Example Functions

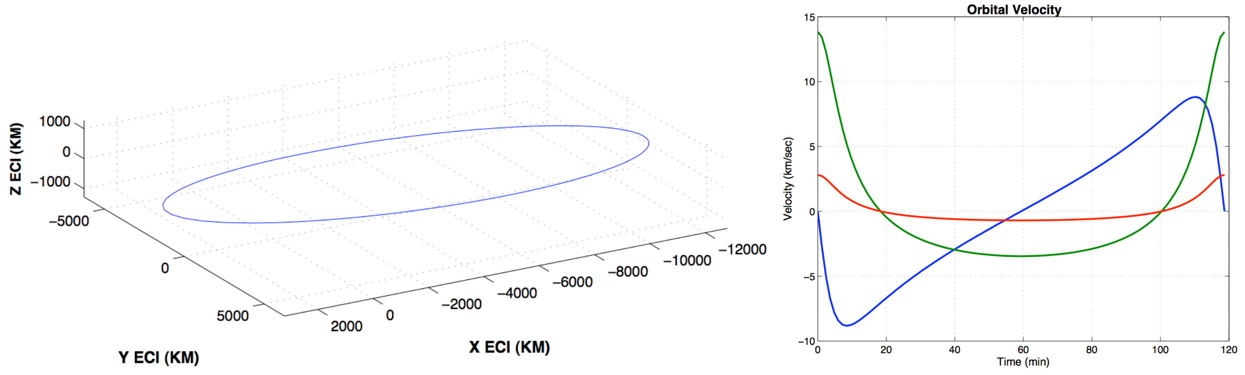
The following sections gives examples for selected functions from the major folders included with the CubeSat toolbox.

### 3.3.1 Basic Orbit

`RVFromKepler` uses Kepler's equation to propagate the position and velocity vectors. The output of the demo is shown in Figure 3.2.

```
>> el = [8000,0.2,0,0,0.6,0]; RVFromKepler( el  )
```

**Figure 3.2:** Elliptical orbit from `RVFromKepler`



```
>> [r,v] = El2RV( el )

r =
        3200
           0
           0

v =
           0
  13.83596536017765
   2.80468902945837
```

### 3.3.2 Coord

Coord has coordinate transformation functions. Many quaternion functions are included. For example, to transform the vector [0;0;1] from ECI to LVLH for a spacecraft in a low earth orbit type

```
>> q = QLVLH( [7000;0;0],[0;7.5;0])
>> QForm( q, [0;0;1] )

q =
                     0.5
                     0.5
                     0.5
                    -0.5
ans =
      0
     -1
      0
```

### 3.3.3 Dynamics

This function can return either a state-space system or the state derivative vector. `RBModel` has analytical expressions for the state space system of a rigid body spacecraft.

```
>> [a, b, c, d] = RBModel( diag([ 2 2 1]),[0;7.291e-5;0]);
>> eig(a)
```

The resulting plant has a double integrator for each axis.

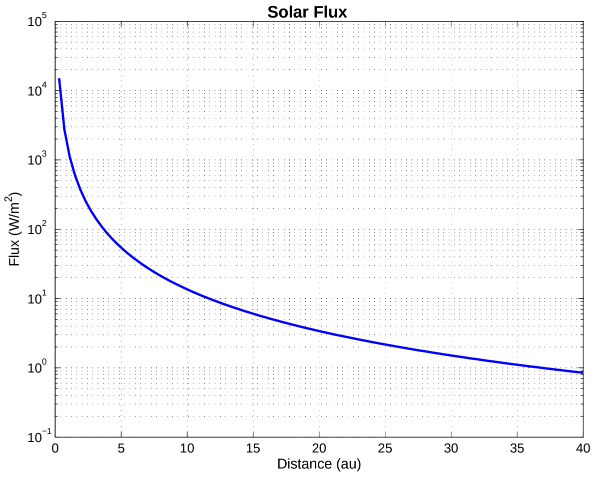
```
ans =
        0
        0
        0
```

### 3.3.4 Environs

Models are available to compute solar flux, planetary radiation, magnetic fields and atmospheric properties. `SolarFlx` computes flux as a function of astronomical units from the sun. The function's output is shown in Example 3.1.

**Example 3.1** Solar flux from the sun

`SolarFlx`



### 3.3.5 Ephem

The ephemeris directory has functions that compute the location and the orientation of the Earth and planets. This includes determining eclipses.

For example, to locate the inertial Sun vector for a spacecraft orbiting the Earth using an almanac model,

```
>> [u, r] = SunV1( JD2000, [7000;0;0] )
```

```
u =
      0.18006
     -0.90249
     -0.39127
r =
   1.4729e+08
```

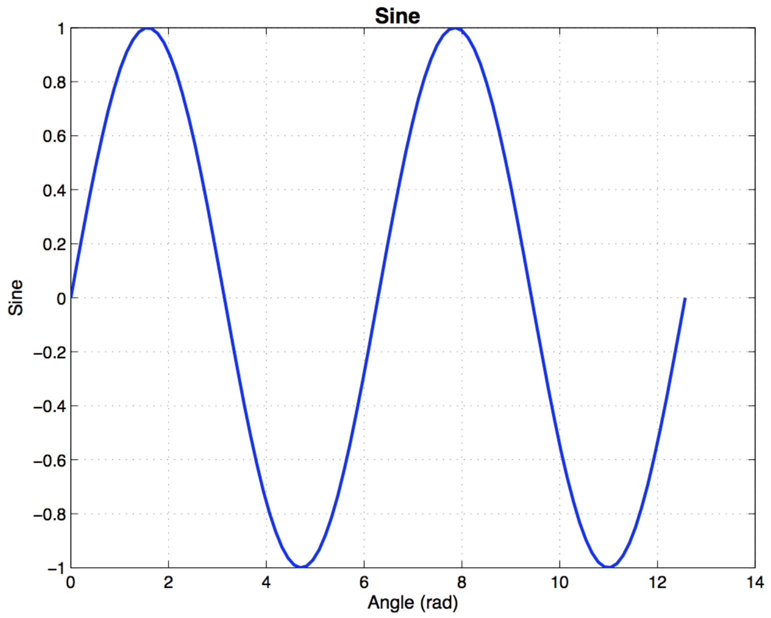
which returns a unit vector to the sun from the spacecraft and the distance from the origin to the sun. The CubeSat toolbox contains only the lower fidelity versions of these functions from the Professional Edition. This includes `SunV1`, `MoonV1`, `Planets`, `Eclipse`, and `ECIToEF`, which is a computationally fast almanac version of the transformation from ECI to the Earth-fixed frame.

### 3.3.6 Graphics

`Plot2D` is used to plot any two dimensional data. It simplifies your scripts by making most popular plotting options available through a single function. `Plot2D` will print out a scalar answer if the inputs are scalar. See Figure 3.3.

```
>> angle = linspace(0,4*pi);
>> Plot2D(angle,sin(angle),'Angle␣(rad)','Sine','Sine')
```

**Figure 3.3:** A sine wave using `Plot2D`



There are many other plot support functions such as `AxesCart`, `Plot3D` and `Map`.

### 3.3.7 OrbitMechanics

The CubeSat toolbox includes functions for computing gravity from a spherical harmonic model. This is handled by `LoadGravityModel` or `LoadGEM` and `AGravity`. Note that the spacecraft position must first be transformed to the planet-fixed frame. For example, to compute the acceleration from the first four terms in the gravity model,

```
>> gem      = LoadGEM( 1 );
>> r        = [5489.150;802.222;3140.916];
>> accel = AGravityC( r, 4, 4, gem )
```

### 3.3.8 Time

The Time directory has functions that convert between various time conventions. The most widely used function is to convert calendar date to Julian Date. Date

```
>> jD  = Date2JD([2019 4 5 0 0 0])
jD =
     2.458578500000000e+06
>> JD2Date(jD)
ans =
     2019           4           5           0           0           0
```

CHAPTER 4

# CUBESAT

This chapter discusses how to use the CubeSat module. This module provides special system modules and mission planning tools suitable for nanosatellite design.

The organization of the module can be seen by typing

```
>> help CubeSat
```

which returns a list of folders in the module. This includes Simulation, MissionPlanning, and Visualization, along with corresponding Demos folders.

The CubeSat integrated simulation includes a simplified surface model for calculating disturbances. The toolbox includes the following features:

- Integrated simulation model including
  - Rigid body dynamics
  - Reaction wheel gyrostat dynamics
  - Point mass
  - Scale height and Jacchia atmospheric models
  - Magnetic dipole, drag, and optical force models
  - Gravity gradient torques
  - Solar cell power model including battery charging dynamics
- Three-axis attitude control using a PID
- Momentum unloading calculations
- Model attitude damping such as with magnetic hysteresis rods
- 2D and 3D visualization including
  - Model visualization with surface normals
  - 2D and 3D orbit plotting
  - 3D attitude visualization
- Ephemeris
  - Convert ECI to Earth-fixed using almanac models
  - Sun vector

  - Moon vector
- Advanced orbit dynamics
  - Spherical harmonic gravity model
  - Relative orbit dynamics between two close satellites
- Observation time windows
- Subsystem models
  - Link bit error probabilities
  - Isothermal spacecraft model
  - Cold gas propulsion

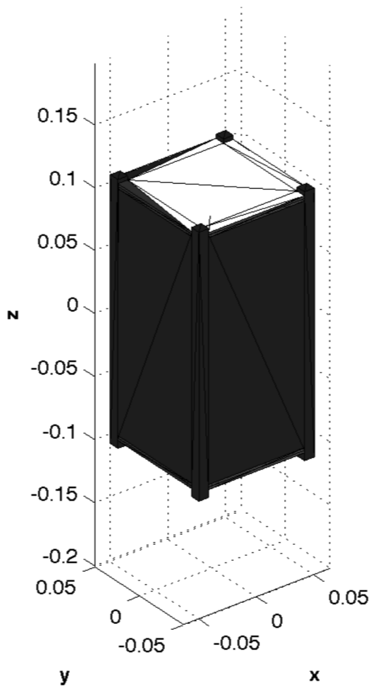
## 4.1 CubeSat Modeling

The CubeSat model is generated by `CubeSatModel`, in the Utilities folder. The default demo creates a 2U satellite. The model is essentially a set of vertices and faces defining the exterior of the satellite. The function header is below and the resulting model is in Figure 4.1 on the facing page. This model has 152 faces.

```
>> help CubeSatModel
  Generate vertices and faces for a CubeSat model.
  If there are no outputs it will generate a plot with surface normals, or
  you can draw the cubesat model using patch:

    patch('vertices',v,'faces',f,'facecolor',[0.5 0.5 0.5]);

  type can be '3U' or [3 2 1] i.e. a different dimension for x, y and z.

  Type CubeSatModel for a demo of a 3U CubeSat.

  This function will populate dRHS for use in RHSCubeSat. The surface
  data for the cube faces will be 6 surfaces that are the dimensions of
  the core spacecraft. Additional surfaces are added for the deployable
  solar panels. Solar panels are grouped into wings that attached to the
  edges of the CubeSat.

  The function computes the inertia matrix, center of mass and total
  mass. The mass properties of the interior components are computed from
  total mass and center of mass.

  If you set frameOnly to true (or 1), v and f will not contain the
  walls. However, dRHS will contain all the wall properties.
 --------------------------------------------------------------------------
    Form:
    d            = CubeSatModel( 'struct' )
    [v, f]       = CubeSatModel( type, t )
    [v, f, dRHS] = CubeSatModel( type, d, frameOnly )
    Demo:
    CubeSatModel
 --------------------------------------------------------------------------

    ------
    Inputs
    ------
    type         (1,:) 'nU' where n may be any number, or [x y z]
    d             (.)  Data structure for the CubeSat
            .thicknessWall          (1,1) Wall thickness (mm)
            .thicknessRail          (1,1) Rail thickness (mm)
            .densityWall            (1,1) Density of the wall material (kg/m3)
            .massComponents         (1,1) Interior component mass (kg)
            .cMComponents           (1,1) Interior components center of mass
```

```
              .sigma                    (3,6) [absorbed; specular; diffuse]
              .cD                       (1,6) Drag coefficient
              .solarPanel.dim           (3,1) [side attached to cubesat, side perpendicular,
                  thickness]
              .solarPanel.nPanels       (1,1) Number of panels per wing
              .solarPanel.rPanel        (3,w) Location of inner edge of panel
              .solarPanel.sPanel        (3,w) Direction of wing spine
              .solarPanel.cellNormal    (3,w) Wing cell normal
              .solarPanel.sigmaCell     (3,1) [absorbed; specular; diffuse] coefficients
              .solarPanel.sigmaBack     (3,1) [absorbed; specular; diffuse]
              .solarPanel.mass          (1,1) Panel mass
         - OR -
      t                                 (1,1) Wall thickness (mm)
      frameOnly    (1,1) If true just draw the frame, optional

      -------
      Outputs
      -------
      v            (:,3) Vertices
      f            (:,3) Faces
      dRHS         (1,1) Data structure for the function RHSCubeSat

   --------------------------------------------------------------------------
      Reference: CubeSat Design Specification (CDS) Revision 9
   --------------------------------------------------------------------------
```

**Figure 4.1:** Model of 2U CubeSat



For the purposes of disturbance analysis, the CubeSat module uses a simplified model of areas and normals. See `CubeSatFaces`. The `CubeSatModel` function will output a data structure with the surface model in addition to the vertices and faces, which are strictly for visualization. This function does have the capability to model deployable solar wings. The solar areas and normals for power generation are specified separately from the satellite surfaces, as they may be only a portion of any given surface. See `SolarCellPower` for the power model.

The `CubeSatRHS` function documents the simulation data model. The function returns a data structure by default for initializing simulations. The surface data from the `CubeSatModel` function is in the `surfData` and `power` fields. The default data assumes no reaction wheels, as can be seen below since the `kWheels` field is empty. The `atm` data structure contains the atmosphere model data for use with `AtmJ70`. If this structure is empty, the simpler and faster scale height model in `AtmDens2` will be used instead.

```
>> d = RHSCubeSat
d =
  struct with fields:

             jD0: 2.4552e+06
            mass: 1
         inertia: 0.0016667
          dipole: [3?1 double]
           power: [1?1 struct]
        surfData: [1?1 struct]
       aeroModel: @CubeSatAero
    opticalModel: @CubeSatRadiationPressure
             atm: [1x1 struct]
         kWheels: []
      inertiaRWA: []
            tRWA: []
```

Note in the above structure that the aerodynamics and optical force models are function handles. These functions are designed to accept the surface model data structure within `RHSCubeSat`.

The key functions for modeling CubeSats are summarized in Table 4.1.

**Table 4.1:** CubeSat Modeling Functions

| | |
|---|---|
| AddMass | Combine component masses and calculate inertia and center-of-mass. |
| InertiaCubeSat | Compute the inertia for standard CubeSat types. |
| CubeSatFaces | Compute surface areas and normals for the faces of a CubeSat. |
| CubeSatModel | Generate vertices and faces for a CubeSat model. |
| TubeSatModel | Generate a TubeSat model. |

## 4.2 Simulation

Example simulations are in the Demos/RelativeOrbit and Demos/Simulation folders. The first has a formation flying demo, `FFSimDemo`. The second has a variety of simulations including an attitude control simulation demo, `CubeSatSimulation`. `CubeSatRWASimulation` demonstrates a set of three orthgonomal reaction wheels. `CubeSatGGStabilized` shows how to set up the mass properties for a gravity-gradient boom.

Attitude control loops can be designed using the `PIDMIMO` function and implemented using `PID3Axis`. These are included from the standard Spacecraft Control Toolbox.

The orbit simulations, `TwoSpacecraftSimpleOrbitSimulation` and `TwoSpacecraftOrbitSimulation`, simulate the same orbits, but the simple version uses just the central force model and the second adds a variety of disturbances. Both use MATLAB's `ode113` function for integration, so that integration occurs on a single line, without a `for` loop. `ode113` is a variable step propagator that may take very long steps for orbit sims. The integration line looks like

```
% Numerically integrate the orbit
%--------------------------------
[t,x] = ode113( @FOrbitMultiSpacecraft, [0 tEnd], x, opt, d );
```

The default simulation length is 12 hours, and the simple sim results in 438 timesteps while the one with disturbances computes 733 steps; Figure 4.2 on the facing page compares the time output of the two examples, where we can see that the simple simulation had mostly a constant step size. Figure 4.3 on the next page shows the typical orbit results.

**Figure 4.2:** Orbit simulation timestep results, simple on the left with with disturbances on the right.

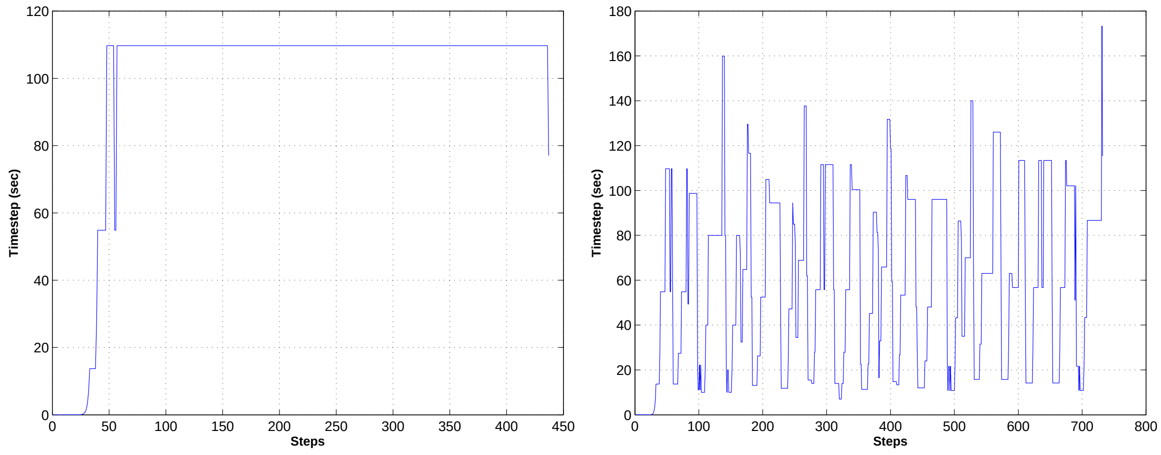
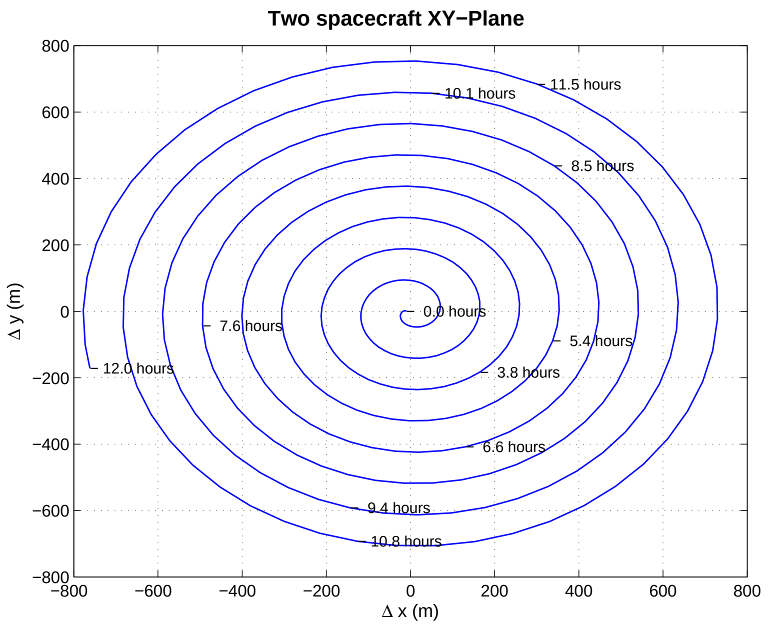


**Figure 4.3:** Orbit evolution for an initial separation of 10 meters

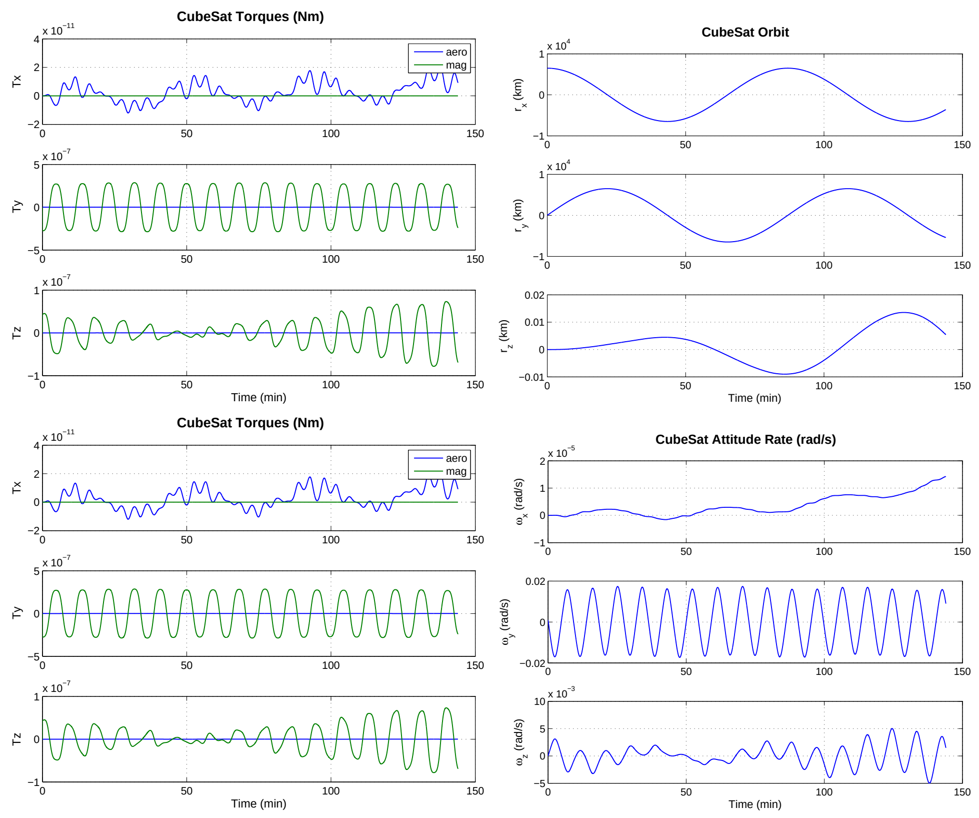


`CubeSatSimulation` simulates the attitude dynamics of the CubeSat in addition to a point-mass orbit and the power subsystem. This simulation includes forces and torques from drag, radiation pressure, and an magnetic torque, such as from a torquer control system. Since this simulation can include control, it steps through time discretely in a `for` loop. `RK4` is used for integration. In this case, the integration lines look like

```
for k = 1:nSim

    % Control system placeholder - apply constant dipole
    %---------------------------------------------------
    d.dipole      = [0.01;0;0]; % Amp-turns m^2

    % A time step with 4th order Runge-Kutta
    %---------------------------------------
    x             = RK4( @RHSCubeSat, x, dT, t, d );

    % Update plotting and time
    %-------------------------
    xPlot(:,k+1)  = x;
    t             = t + dT;

end
```

where the control, `d.dipole`, would be computed before the integration at every step for a fixed timestep (1 second). See Figure 4.4 for sample results. The simulation takes about one minute of computation time per low-Earth orbit.

**Figure 4.4:** `CubeSatSimulation` example results



The key functions used in simulations are summarized in Table 4.2 on the next page. The space environment calculations from `CubeSatEnvironment` are then passed to the force models in `CubeSatAero` and `CubeSatRadiationPressure`.

**Table 4.2:** CubeSat Simulation Functions

| | |
|---|---|
| RHSCubeSat | Dynamics model including power and optional reaction wheels |
| CubeSatEnvironment | Environment calculations for the CubeSat dynamical model. |
| CubeSatAero | Aerodynamic model for a CubeSat. |
| CubeSatRadiationPressure | Radiation pressure model for a CubeSat around the Earth. |
| CubeSatAttitude | Attitude model with either ECI or LVLH reference |
| SolarCellPower | Compute the power generated for a CubeSat. |

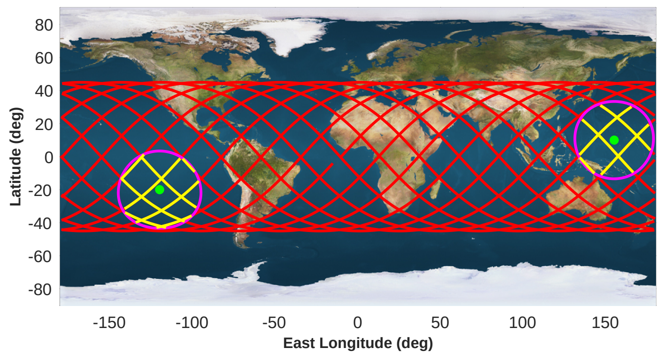
`RHSCubeSat` also models battery charging if the `batteryCapacity` field of the `power` structure has been appropriately set. Power beyond the calculated consumption will be used to charge the battery until the capacity is reached. The battery charge is always be the last element of the spacecraft state (after the states of any optional reaction wheels).

## 4.3 Mission Planning

The MissionPlanning folder provides several tools for planning a CubeSat mission. These include generating attitude profiles and determining observation windows.

`ObservationTimeWindows` has a built-in demo which also demonstrates `ObservationTimeWindowsPlot`, as shown in Figure 4.5. There are two ground targets, one in South America and one in France (large green dots).

**Figure 4.5:** Observation time windows

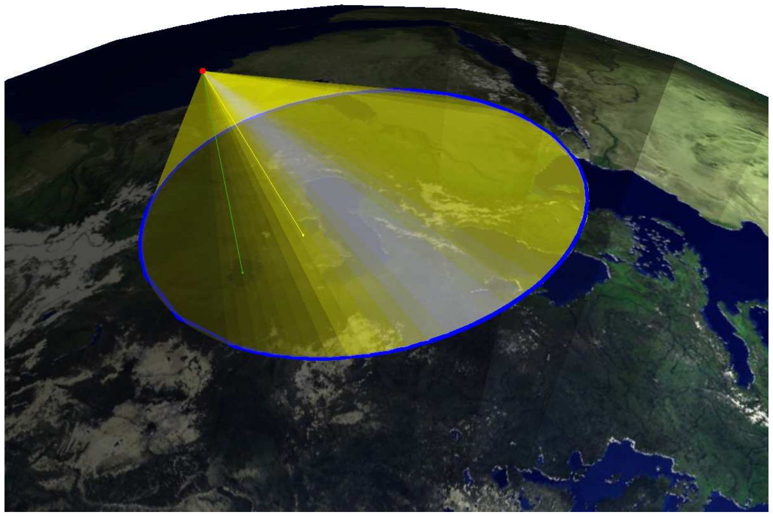


The satellite is placed in a low Earth orbit, given a field of view of 180 degrees, and the windows are generated over a 7 hour horizon. The figure shows the field of view in magenta and the satellite trajectory segments when the target is in the field of view are highlighted in yellow. This function can operate on a single Keplerian element set or a stored trajectory profile.

`RapidSwath` also has a built-in demo. The demos uses an altitude of 2000 km and a field of view half-angle of 31 degrees. The function allows you to specify a pitch angle between the sensor boresight axis and the nadir axis, in this case 15 degrees. When called with no outputs, the function generates a 3D plot. In Figure 4.6 on the next page we have used the camera controls to zoom in on the sensor cone. The nadir axis is drawn in green and the boresight axis in yellow.

The `AttitudeProfileDemo` shows how to assemble several profile segments together and get the resulting observation windows. The segments can be any of a number of modes, such as latitude/longitude pointing, nadir or sun

**Figure 4.6:** `RapidSwath` built-in demo results



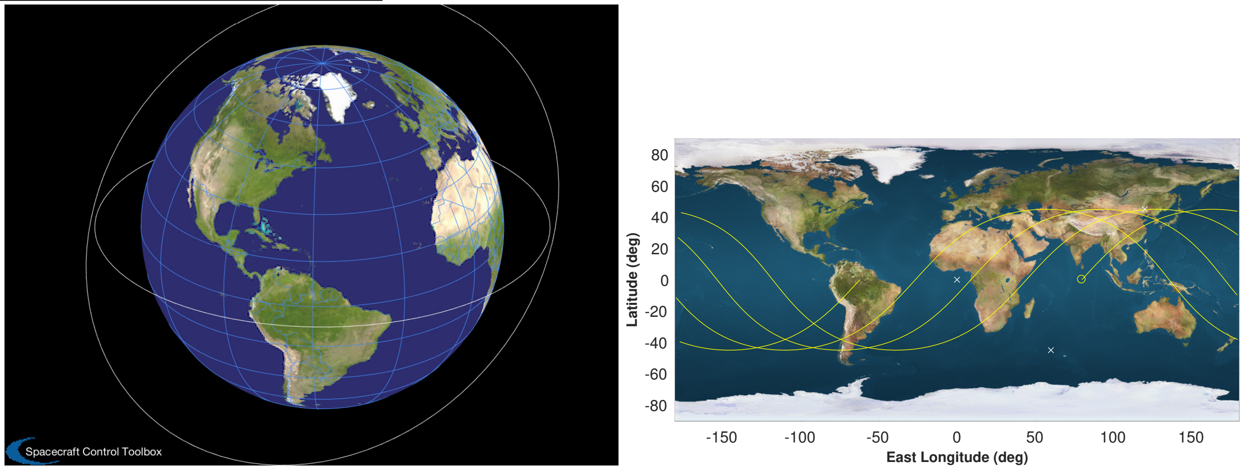
pointing, etc.

## 4.4 Visualization

The CubeSat Toolbox provides some useful tools to visualize orbits, field of view, lines of sight, and spacecraft orientations.

Use `PlotOrbit` to view a spacecraft trajectory in 3D with an Earth map. The `GroundTrack` function plots the trajectory in 2D and has the option of marking ground station locations.

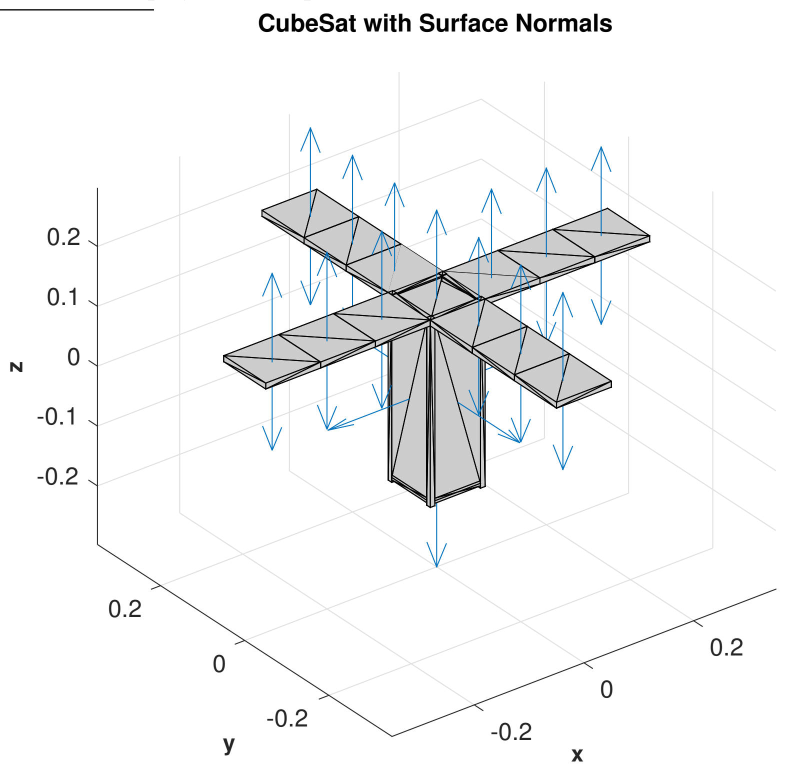
**Figure 4.7:** Orbit visualization with `PlotOrbit` and `GroundTrack`



The spacecraft model from `CubeSatModel` can be viewed with surface normals using `DrawCubeSat`. The vertex and face information is not retained with the dynamical data, so `DrawCubeSatSolarAreas` can be used to verify the solar cell areas directly from the RHS data structure.

A representative model of the spacecraft may also be viewed in its orbit, along with a sensor cone and lines of sight to

**Figure 4.8:** CubeSat model with deployable solar panels viewed with `DrawCubeSat`



all of the visible GPS satellites. Use `DrawSpacecraftInOrbit.m` to generate this view. An example is shown in Figure 4.9. The image on the left shows the spacecraft orbit, its sensor cone projected on the Earth, the surrounding GPS satellites, and lines of sight to the visible GPS satellites. The image on the right is a zoomed-in view, where the spacecraft CAD model may be clearly seen.

**Figure 4.9:** Spacecraft visualization with sight lines using `DrawSpacecraftInOrbit`

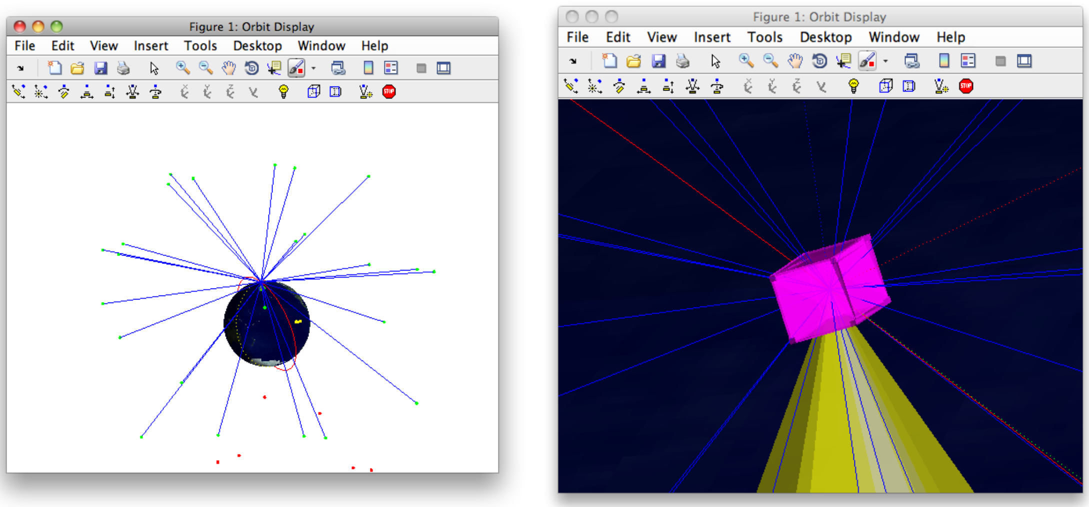
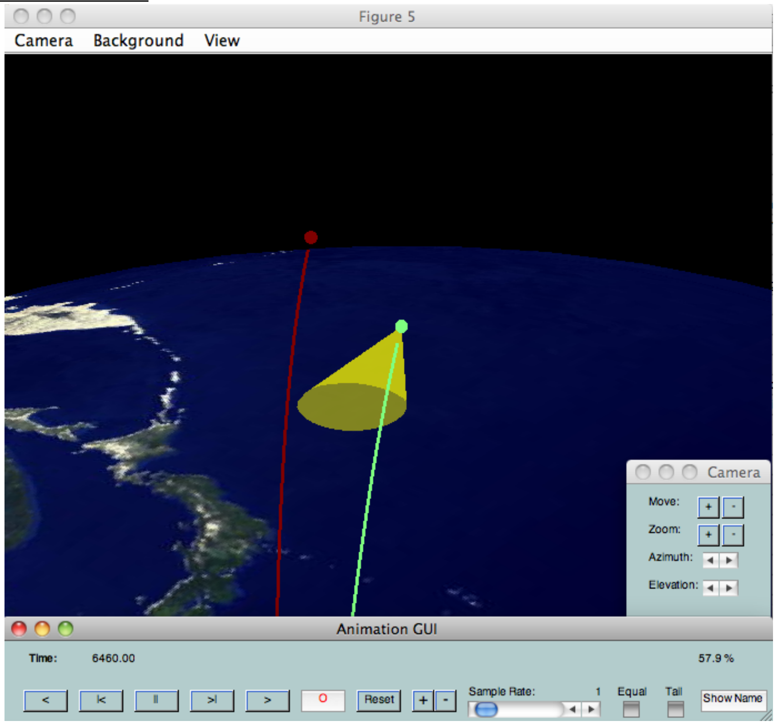


Figure 1: Orbit Display
File Edit View Insert Tools Desktop Window Help
Figure 1: Orbit Display
File Edit View Insert Tools Desktop Window Help

Run the `OrbitAndSensorConeAnimation.m` mission planning demo to see how to generate simulated orbits, compute sensor cone geometry, and package the data for playback using `PackageOrbitDataForPlayback` and `PlaybackOrbitSim`. The playback function loads two orbits into the `AnimationGUI`, which provides VCR like controls for playing the simulation forward and backward at different speeds. Set the background color to black and point the camera at a spacecraft, then use the camera controls to move in/out, zoom in/out, and rotate the camera around the spacecraft within a local coordinate frame. The screenshot in Figure 4.10 shows the 3D animation window, the `AnimationGUI` playback controls, and the camera controls. Press the Record button (with the red circle) to save the frames to the workspace so that they may be exported to an AVI movie.

**Figure 4.10:** Playback demo using `PlaybackOrbitSim`



## 4.5 Subsystems Modeling

The CubeSat Toolbox contains select models for key subsystems. The relevant functions and demos are:

- `BatterySizing` - compute power storage requirements given a spacecraft power model and an orbit
- `LinkOrbitAnalysis` - Compute bit error probability along an orbit
- `IsothermalCubeSatDemo` - modeling the CubeSat as a block at a single temperature, calculate the fluctuations over an orbit including the effect of eclipses.
- `CubeSatPropulsion` - Returns the force, torque and mass flow for a cold gas system.
- `DesignMagneticTorquer` - Design an air coil magnetic torquer for a CubeSat.

CHAPTER 5

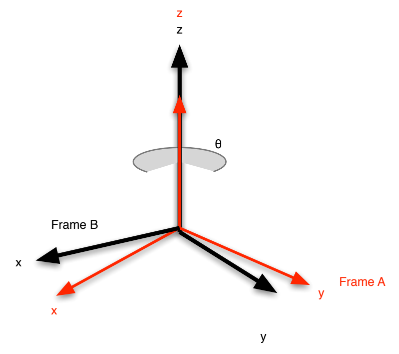
# Coordinate Transformations

This chapter shows you how to use Spacecraft Control Toolbox functions for coordinate transformations. There is a very extensive set of functions in the Common/Coord folder covering quaternions, Euler angles, transformation matrices, right ascension/declination, spherical coordinates, geodetic coordinates, and more. A few are discussed here. For more information on coordinate frames and representations, please see the Coordinate Systems and Kinematics chapters in the accompanying book *Spacecraft Attitude and Orbit Control*.

## 5.1 Transformation Matrices

Transforming a vector $u$ from its representation in frame $A$ to its representation in frame $B$ is easily done with a transformation matrix. Consider two frames with an angle $\theta$ between their $x$ and $y$ axes.

**Figure 5.1:** Frames A and B



```
uA    = [1;0;0];
theta = pi/6;
m = [ cos(theta),sin(theta),0;...
     -sin(theta),cos(theta),0;...
      0,0,1];
uB = m*uA
```

Using SCT functions, this code can be written as a function of Euler angles using a rotation about the $z$ axis:

```
uB = Eul2Mat([0,0,theta])*uA;
```

Use `Mat2Eul` to switch back to an Euler angle representation.

## 5.2 Quaternions

A quaternion is a four parameter set that embodies the concept that any set of rotations can be represented by a single axis of rotation and an angle. PSS uses the shuttle convention so that our unit quaternion (obtained with `QZero`) is [1 0 0 0]. In Figure 5.1 on the preceding page the axis of rotation is `[0 0 1]` (the $z$ axis) and the angle is `theta`. Of course, the axis of rotation could also be `[0 0 -1]` and the angle `-theta`.

Quaternion transformations are implemented by the functions `QForm` and `QTForm`. `QForm` rotates a vector in the direction of the quaternion, and `QTForm` rotates it in the opposite direction. In this case

```
q  = Mat2Q(m);
uB = QForm(q,uA)
uA = QTForm(q,uB)
```

We could also get `q` by typing

```
q = Eul2Q([0;0;theta])
```

Much as you can concatenate coordinate transformation matrices, you can also multiply quaternions. If `qAToB` transforms from A to B and `qBToC` transforms from B to C then

```
qAToC = QMult(qAToB,qBToC);
```

The transpose of a quaternion is just

```
qCToA = QPose(qAToC);
```

You can extract Euler angles by

```
eAToC = Q2Eul(qAToC);
```

or matrices by

```
mAToC = Q2Mat(qAToC);
```

If we convert the three Euler angles to a quaternion

```
qIToB = Eul2Q(e);
```

`qIToB` will transform vectors represented in $I$ to vectors represented in $B$. This quaternion will be the transpose of the quaternion that rotates frame $B$ from its initial orientation to its final orientation or

```
qIToB = QPose(qBInitialToBFinal);
```

Given a vector of small angles `eSmall` that rotate from vectors from frame $A$ to $B$, the transformation from $A$ to $B$ is

```
uB = (eye(3)-SkewSymm(eSmall))*uA;
```

where

```
SkewSymm([1;2;3])
ans =
[0 -3  2;
 3  0 -1;
-2  1  0]
```

Note that `SkewSymm(x)*y` is the same as `Cross(x,y)`.

## 5.3 Coordinate Frames

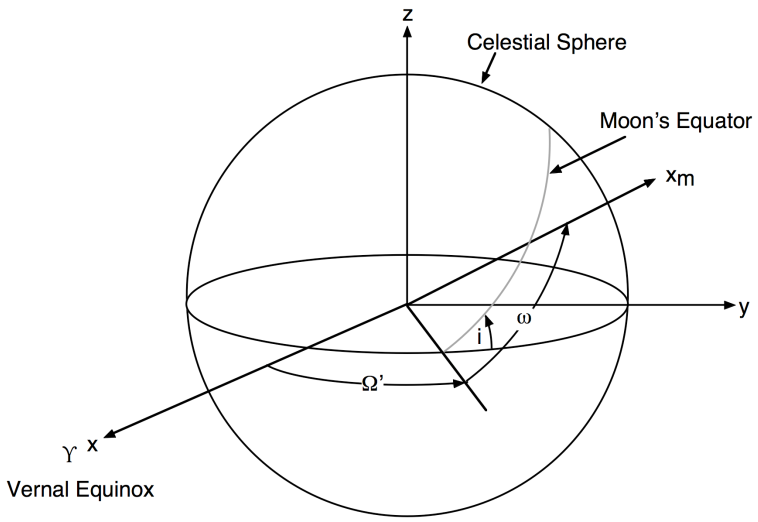
The toolbox has functions for many common coordinate frames and representations, including

- Earth-fixed frame, aerographic (Mars), selenographic (Moon)
- Latitude (geocentric and geodetic), longitude, and altitude
- Earth-centered inertial
- Local vertical, local horizontal
- Nadir and sun-nadir pointing
- Hills frame
- Rotating libration point
- Right ascension and declination
- Azimuth and elevation
- Spherical, cartesian, and cylindrical

Most of these functions can be found in Coord and some are in Ephem due to their dependence on ephemeris data such as the sun vector. A few relevant functions are in OrbitMechanics. For example, the `QLVLH` function in Coord computes a quaternion from the inertial to local-vertical local-horizontal frame from the position and velocity vectors. `QNadirPoint`, also in Coord, aligns a particular body vector with the nadir vector. The `QSunNadir` function in Ephem computes the sun-nadir quaternion from the ECI spacecraft state and sun vector.

The relationship between the Selenographic and the ECI frame is computed by `MoonRot` and shown in Figure 5.2. $\phi'_m$ is the Selenographic Latitude which is the acute angle measured normal to the Moon's equator between the

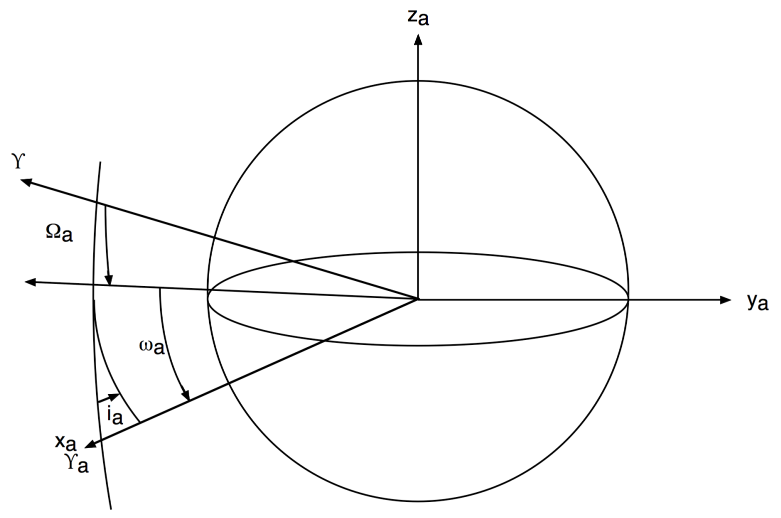
**Figure 5.2:** Selenographic to ECI frame



equator and a line connecting the geometrical center of the coordinate system with a point on the surface of the Moon. The angle range is between 0 and 360 degrees. $\lambda_m$ is the Selenographic Longitude which is the angle measured towards the West in the Moon's equatorial plane, from the lunar prime meridian to the object's meridian. The angle range is between -90 and +90 degrees. North is the section above the lunar equator containing Mare Serenitatis. West is measured towards Mare Crisium.

The areocentric frame computed by `MarsRot` is shown in Figure 5.3 on the following page. $\Omega_a$, $\omega_a$ and $i_a$ are the standard Euler rotations of the Mars vernal equinox, $\Upsilon_a$ with respect to the Earth's vernal equinox, $\Upsilon$.

**Figure 5.3:** Areocentric frame



CHAPTER 6

# COORDINATE FRAMES

This chapter introduces the different coordinate frames functions available in the Formation Flying module. This subset are included in the CubeSat toolbox to support a formation flying simulation function. Only circular reference orbits are supported. Further detail on the derivations is available in the companion textbook.

## 6.1 Overview

The coordinate systems used in this module can first be divided into 2 major groups: *absolute* and *relative*. Absolute coordinates describe the motion of a satellite with respect to its central body, i.e the Earth. Relative coordinates describe the motion of a satellite with respect to a point (i.e. another satellite) on the reference orbit.

A summary of the different types of coordinate systems is provided below:

- Absolute Coordinate Systems
  - Earth Centered Inertial (ECI)
  - Orbital Elements
    - Standard $[a, i, \Omega, \omega, e, M]$
    - Alfriend $[a, \theta, i, q_1, q_2, \Omega]$
    - Mean and Osculating
- Relative Coordinate Systems
  - Differential Elements
  - Cartesian Frames
    - Hill's Frame
    - LVLH Frame
    - Frenet Frame
  - Geometric Parameters

*An important note on terminology:* when discussing formation flying, the satellites with relative motion are termed *relatives*, and the satellite that they move with respect to is termed the *reference*.

## 6.2 Orbital Element Sets

The Formation Flying module makes use of two different orbital element sets. The classical Kepler elements are defined as:

$$[a, i, \Omega, \omega, e, M]$$

where $a$ is the semi-major axis, $i$ is the inclination, $\Omega$ is the right ascension (longitude) of the ascending node, $\omega$ is the argument of perigee, $e$ is the eccentricity, and $M$ is the mean anomaly. This is the standard set of elements used throughout the Spacecraft Control Toolbox.

A second set of elements is particularly useful for formation flying applications. This set is termed the Alfriend element set, after Dr. Terry Alfriend of Texas A&M who first suggested its use. The Alfriend elements are used solely with circular or near-circular orbits. The Alfriend set is:

$$[a, \theta, i, q_1, q_2, \Omega]$$

where $a, i, \Omega$ are defined as before. The remaining elements, $q_1, q_2, \theta$, are defined in order to avoid the problem that arises at zero eccentricity, where the classical argument of perigee and mean anomaly are undefined.

The functions `Alfriend2El` and `El2Alfriend` can be used to convert between the two element sets.

Either of the two above element sets can be defined as mean or osculating. When orbits are governed by a point-mass model for the central body's gravity field, the elements do not osculate. Higher fidelity models have perturbations that cause the elements to change over time, or osculate. The function `Osc2Mean` will convert osculating elements to mean elements. The elements must be defined in the Alfriend system. Similarly, the function `ECI2MeanElements` will compute the mean elements directly from an ECI state.

## 6.3 Relative Coordinate Systems

The three different types of coordinate systems for expressing the relative orbital states of spacecraft are described in the textbook: orbital element differences, cartesian coordinate systems, and geometric parameter sets . The Formation Flying module provides coordinate transformation utilities to switch back and forth between all three systems.

### 6.3.1 Orbital Element Differences

A differential orbital element vector is simply the difference between the orbital element vectors of two satellites. Just as with regular, absolute orbits, this is a convenient way to parameterize the motion of a relative orbit. In the absence of disturbances and gravitational perturbations, 5 of the 6 differential orbital elements remain fixed; only the mean (or true) anomaly changes.

The function `OrbElemDiff` can be used to robustly subtract two element vectors. This function ensures that angle differences around the wrapping points of $\pm\pi$ and $(0, 2\pi)$ are computed properly.

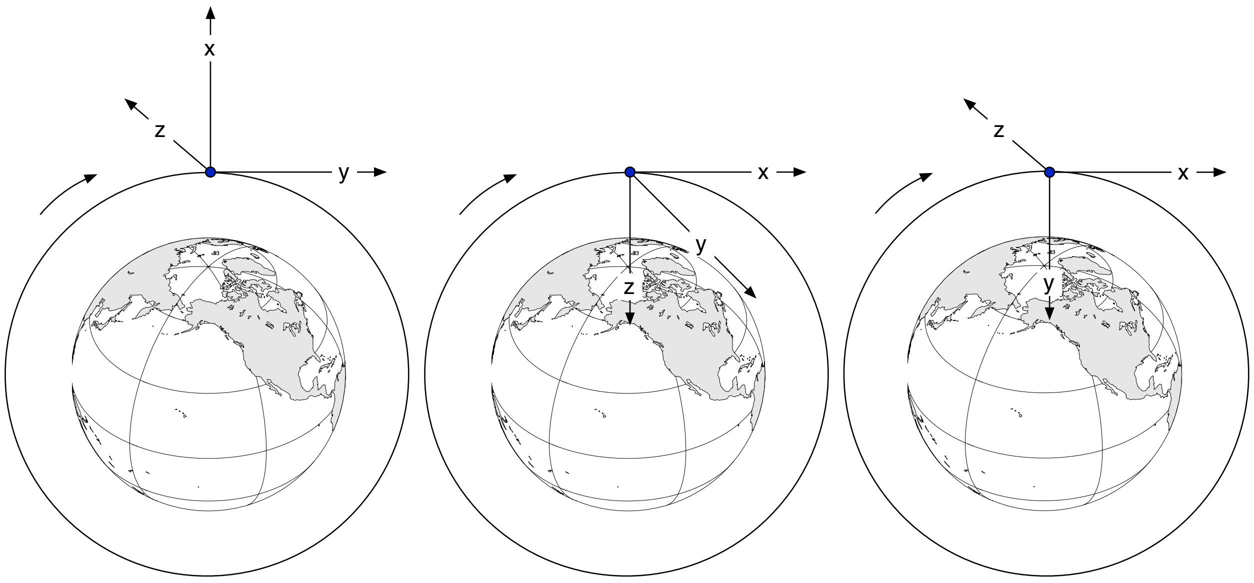
### 6.3.2 Cartesian Coordinate Systems

The three different coordinate systems for relative orbital motion supported by the Formation Flying module are:

- Hills
- LVLH
- Frenet

Each frame is attached to the reference satellite, and rotates once per orbit. Two axes are contained in the orbital plane, and the third axis points normal to the plane. A diagram of each frame is shown in Figure 6.1. For simplicity, the frames are shown with a circular reference orbit.

**Figure 6.1:** Relative Orbit Coordinate Frames



(a) Hills Frame (b) LVLH Frame (c) Frenet Frame

To compute a Hills-frame state from two inertial states, use the function: `ECI2Hills`.

To transform between the Hills and LVLH frames, use the functions: `Hills2LVLH` and `LVLH2Hills`.

To transform from the Hills frame to the Frenet frame, use the function: `Hills2Frenet`.

### 6.3.3 Geometric Parameter Sets

The previous coordinate systems provide an exact way to express the relative orbit state. The Formation Flying module also enables you to express *desired* relative states using geometric parameters. In formation flying, we are generally interested in defining relative orbit trajectories that repeat once each period. This is referred to as "T-Periodic Motion", and is discussed in the textbook. When the trajectory repeats itself periodically, it forms a specific geometric shape in 3 dimensional space. Our sets of geometric parameters can be used to uniquely describe the shape of T-periodic trajectories in circular and eccentric orbits.

For T-periodic motion in circular orbits, remember that in-plane motion takes on the shape of a 2x1 ellipse, with the longer side oriented in the along-track direction and the shorter side along the radial direction. As we know from Hill's equations, the cross-track motion is just a harmonic oscillator, decoupled from the in-plane motion, with a natural frequency equal to the orbit rate.

Circular geometries are defined with the following parameters, shown in Table 6.1 on the following page:

Use the function `Geometry_Structure` to create a data structure of circular geometry parameters:

```
>> g = Geometry_Structure
g =
  struct with fields:
```

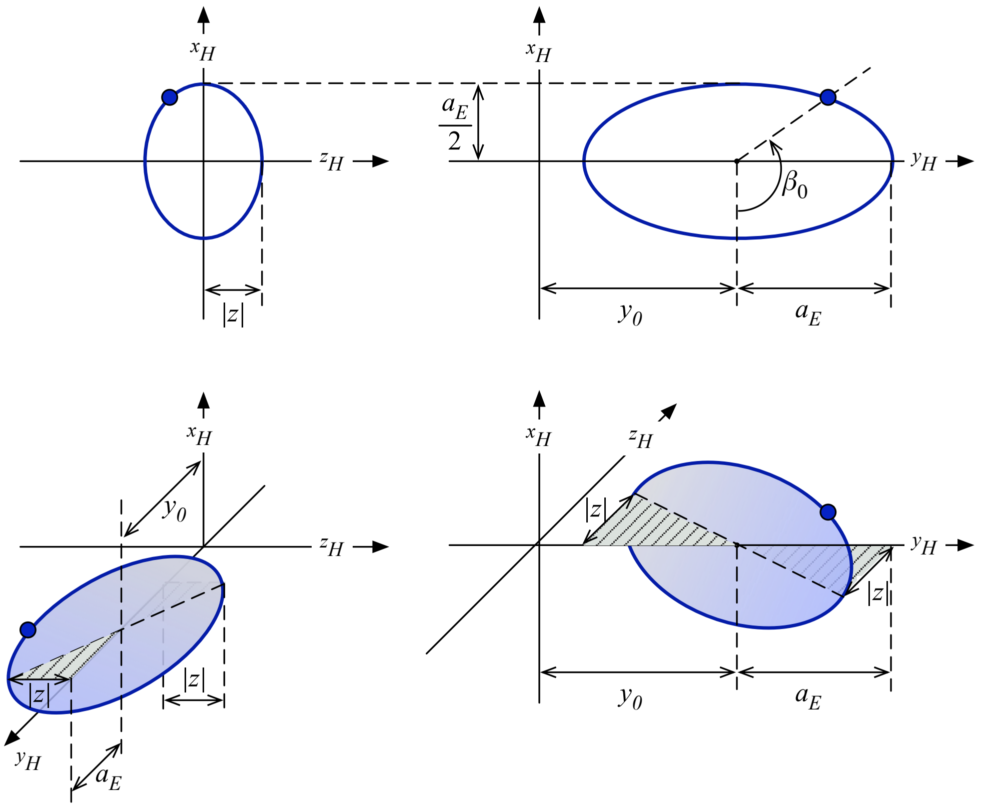
**Table 6.1:** Geometric Parameters for Circular Orbits

| Parameter | | Description |
|---|---|---|
| `y0` | $y_0$ | Along-track offset of the center of the in-plane motion |
| `aE` | $a_E$ | Semi-major axis of relative 2x1 in-plane ellipse |
| `beta` | $\beta$ | Phase angle on ellipse (measured positive clockwise from nadir axis to velocity vector) when the satellite is at the ascending node |
| `zInc` | $z_i$ | Cross-track amplitude due to inclination (Inc) difference |
| `zLan` | $z_\Omega$ | Cross-track amplitude due to longitude of ascending node (Lan) difference |

```
    y0: 0
    aE: 0
  beta: 0
  zInc: 0
  zLan: 0
```

A diagram illustrating these parameters is shown in Figure 6.2.

**Figure 6.2:** Geometric Parameters for Circular Orbits



Separating the cross-track geometry into the contributions from inclination and right ascension differences provides useful insight into the stability of the trajectory. As we know, the J2 perturbation (Earth oblateness) causes secular drift in several orbital elements. If the secular drift is the same for both orbits, then there is no *relative* secular drift introduced by J2. It turns out that J2 has two significant impacts on relative motion: 1) it can cause secular drift in the along-track direction, and 2) it can create a frequency difference between the in-plane and out-of-plane motion. For the stability of formations, we seek to minimize the amount of along-track drift. It can be shown that the along-track drift due to right ascension differences is much smaller than that due to inclination differences. This is the motivation for defining the cross-track geometry parameters according to inclination and right ascension differences.