
$V = \frac{\mu}{r}\sum_{n=0}^{\infty}\sum_{m=0}^{n}\frac{R^n}{r^n}P_{nm}(\sin\phi)(c_{nm}\cos(m\lambda)+s_{nm}\sin(m\lambda))$
ATA
ADVANCED TECHNOLOGY ASSOCIATES, INC.
Aerospace Toolkit
for Labview
NATIONAL INSTRUMENTS
Alliance

ATA Aerospace Toolkit User Manual

Introduction

About the ATA Aerospace Toolkit

Welcome to the ATA Aerospace Toolkit version 2.x for LabVIEW. This development tool provides functionality designed to aid the user in simulation, design, and analysis of vehicle flight using the graphical G language of LabVIEW. With the ATA Aerospace Toolkit the user can rapidly build models for analysis, design and hardware-in-the-loop testing.

The ATA Aerospace Toolkit is divided into 11 modules or "libraries", each containing "virtual-instruments" (VI) or programs that can be easily assembled to create a high-fidelity model. The modules contained within the ATA Aerospace Toolkit include, Attitude Analysis, Coordinate Frame Transformations, Orbit Analysis, Earth Analysis, Mathematical Analysis, Math Utilities, Orbit Propagation, Orbit Adjustment, Aerodynamic Utilities, Mass Properties, and Time Utilities.

The number of possible simulations that can be programmed using the ATA Aerospace Toolkit is endless. The modular design allows users to easily select the appropriate force models and integrators to simulate any trajectory or orbit desired.

The ATA Aerospace Toolkit version 2.x (hereafter referred to as simply ATA Aerospace Toolkit or simply Toolkit) works in conjunction with LabVIEW version 8.6, the LabVIEW Real-Time Module associated with 8.6 and is compatible with the LabVIEW Simulation Module.

A Note on Forward and Backward Compatibility

LabVIEW supports the saving of VIs backward to one previous version. That means users of this Toolkit may save their work as VIs for LabVIEW 8.6. ATA supports this feature.

Typically VIs written for one version of LabVIEW are forward compatible. Each release of the ATA Aerospace Toolkit is designed to work with a specific release of the LabVIEW development environment. Forward compatibility for any Toolkit version is not guaranteed. ATA offers continuing support of the LabVIEW development environment, via its support services. For more information please see www.atacolorado.com

About the User's Manual and Finding Help

This user's manual is intended as a reference to accompany the ATA Aerospace Toolkit. In addition to the information contained herein, other information may be available in the LabVIEW VI documentation, help, or tips. Information may also be

found by going to the Advanced Technology Associates, Inc. website at www.atacolorado.com

This manual is not intended to be an instruction on how to program using LabVIEW and it is assumed that the user will already have familiarity with the programming environment. For more information on LabVIEW or training for LabVIEW please go to www.ni.com.

The manual is organized into four main sections.

- An overview of programming considerations in LabVIEW
- A description of the controls and indicators contained in the ATA Aerospace Toolkit
- A description of the VI's contained in the ATA Aerospace Toolkit
- A description of the examples contained in the ATA Aerospace Toolkit

LabVIEW Programming Considerations

Heritage and Legacy Considerations

The ATA Aerospace Toolkit evolved from decades of intellectual property development in text based programming languages that are based on a procedural paradigm. Many of the algorithms have years of testing in Fortran, C and other proprietary, company-specific text languages. ATA is striving to bring the intellectual property to modern programming environments such as LabVIEW. The first step has been to port all of the text code to C and compile it into a Dynamic Link Library called AEROTOOLKIT.DLL. The functional components have been documented and retested and wrapped into LabVIEW VIs via the Call Library Function Node interface. The user of the Toolkit is able to take advantage of LabVIEW's modern graphical interface and also rely on the heritage of long-standing well-tested algorithm implementations.

Strongly Typed G Code

The ATA Aerospace Toolkit utilizes strongly typed interfaces to minimize errors in the design process. Therefore many of the indicators are based on LabVIEW type definitions (CTL files) which are included with Toolkit. They are also added to the front panel palette with a default installation. Users can take advantage of these controls to efficiently pass data to Toolkit VIs.

The ATA Aerospace Toolkit takes advantage of LabVIEW's physical dimension unit feature. Wherever possible inputs and outputs have a specific unit associated with them. Generally these are in the MKS system, though much of the aerospace industry still uses the English Engineering System. However, the ATA Aerospace Toolkit leverages the power of LabVIEW to allow the user to modify the units at run-time by simply altering the unit string. A dimensionally consistent modification is automatically handled by LabVIEW. For example, the user can change the unit string of an input from

"m" to "ft" in order to enter a value in feet instead of meters. In so doing, LabVIEW will automatically multiply the value by 0.3048 to convert from meters to feet. Additionally, it is not possible to perform dimensionally inconsistent conversion, like if a user attempts to type in "lb" for an input that has a length dimension LabVIEW will simply not allow it.

Strong type definitions are also enforced by using clusters when arrays might otherwise be used in a simpler, but more error prone implementation. For instance, a three-dimensional Cartesian position vector is represented by a cluster with three elements, with specific member names X, Y and Z. This way there is no chance that a developer might accidentally connect a two-element or four-element or even zero-element array. Although array size checking could generate errors at run time, the use of clusters catches such mistakes at development time. Also the use of clusters with explicitly named elements eliminates any possible confusion about the order of elements. For example, in some development libraries a quaternion is a four-element array in the order {i, j, k, s} in others the order is {s, i, j, k}.

It is recognized that there may be some loss of real-time performance due to additional conversion that must take place. It is believed that for many real-time applications this additional overhead will not be significant enough to outweigh the input validation gained during development. However, the user can mitigate this problem by saving their own versions of the ATA VIs, replacing the clusters with arrays, as needed. Use of this technique is encouraged only when the real-time gains are considered to be significant.

Error Handling & Input Validation

The ATA Aerospace Toolkit VIs perform stringent input validation and errors are generated for invalid inputs. Despite input validation, sometimes errors will still occur in the DLL.

All ATA Aerospace Toolkit functions that can have an error return an integer value that corresponds to an error code. The VI wrapper feeds this into an error handler. Upon default installation all ATA Aerospace Toolkit error codes are added to the LabVIEW error code database so that they can be handled by the LabVIEW general error handler. ATA Aerospace Toolkit VIs generate error codes between 5000 and 5200 and also between 6000 and 6200; these are in a range set aside by LabVIEW for user defined error codes. All VIs that can generate errors will call the ATA error handler. There is a single global switch that governs the error handling strategy used by the ATA error handler. For one option it will invoke the general error handler and pop up a dialog box; for the other option the error clusters are passed up and it is the responsibility of the developer to implement error handling at a higher level.

Real Values are Double Precision

In general anything described as a real value is represented in LabVIEW as a double, which is a 64-bit representation defined by IEEE 754. Underlying C code retains and utilizes this double precision representation.

Building Applications

Users can build LabVIEW applications that use the ATA Aerospace Toolkit, but in order to use the application, the computer to which the application is deployed needs the data folder created upon build and the correct LabVIEW runtime engine.

Real-Time

In order to use the ATA Aerospace Toolkit in real-time applications, the real-time DLL must be copied to the real-time target before deployment. The real-time DLL is located in C:\Program Files\ATA Aero Toolkit\Real Time on the computer where the ATA Aero Toolkit is installed. Copy the DLL to C:\ni-rt\system folder on the target real-time system.

Time

Time is a tricky subject in aerospace computing. Not only are there different scales and formats, but different standards for representing time in a computer have emerged.

Time Stamp and UTC Exact Time Structure

The Exact Time Structure has now been removed from the ATA Aerospace Toolkit as of version 2.1. The Aerospace Toolkit now uses a custom typdef control called TimeStampToUTC. This control allows the user to either enter time in UTC (default) or the local time of the host machine.

Unix and Apple Epochs

Another common time concept in aerospace computing is the concept of an epoch. This is the zero point in a time continuum that processors use to represent time. There are two commonly encountered epochs, Unix and Apple. When programming simulations it is often convenient to use a continuum of “seconds since epoch”, rather than convert hours, minutes, seconds, etc. However, this efficiency is quickly lost if frequent conversions must be made from this format to others to track Earth rotation. This trade-off decision is something that must be made by the programmer and could have real-time performance ramifications.

The ATA Aerospace toolkit is implemented over a Win32 DLL, written in C, that works on both Windows machines and the partially Win32-compatible RTOS that comes with LabVIEW Real-Time. The Win32 implementation for the continuous time format (`time_t`) is a 32 bit signed integer representing the number of seconds since January 1, 1970 UTC (also referred to as GMT). This is the Unix Epoch. There are about 136 years in the number of seconds that can be represented by 32 bits, so this gives a comprehensive span from December 13, 1901 to January 18, 2038.

LabVIEW was originally developed for the Apple Macintosh computer, the only microcomputer with a powerful enough GUI at the time LabVIEW premiered. The original (Pre OS X) Apple Macintosh implementation of time_t is a 32 bit *unsigned* integer beginning January 1, 1904 UTC. Because the time type is an unsigned value, one can only proceed forward from the reference; therefore it covers a span of time with the same duration as the Unix Epoch, but approximately two years later.

The difference between the Unix Epoch and the Apple Epoch is 2,082,870,000 seconds.

Prior to LabVIEW 8.2, it could tricky to use LabVIEW support for time, because the LabVIEW VIs often insisted on incorporating local time offsets into every conversion. Beginning with LabVIEW 8.2 there is a UTC flag for conversion back and forth to the LabVIEW Time Stamp type. Note that the flag need not be specified and the default for that flag is False, so it is incumbent upon the user to explicitly wire it true. Given that a LabVIEW time stamp can be generated in UTC, it is reasonable for a user to utilize this interface in applications. If the user chooses to convert the LabVIEW time stamp to seconds since epoch, they must apply the 2,082,870,000 correction to convert from the Apple epoch used by LabVIEW, to the Unix Epoch used by the ATA Aerospace Toolkit. An alternative method is to use the ATA Aerospace Toolkit VI TimeStamptoUTC and then ExactTimetoSSE. If the user wishes to work only with the exact time format they may use AdjustExactTime to increment or decrement the Exact Time Structure.

Aerospace Toolkit Controls and Indicators

The ATA Aerospace Toolkit includes many type definition controls that will be added to the front panel in a default installation. These match the controls and indicators that are used for the VIs in the toolkit. This will facilitate the development of higher level VIs which use the Aerospace Toolkit VIs as components. Many of these controls differ only by the physical units (such as CartPos3D, Cartesian 3D position and CartVel3D, Cartesian 3D velocity). The strict use of physical units adds some burden to the developer, a palette full of controls with the most commonly used units significantly alleviates this burden. Specifically there are many Cartesian 3D vector type definitions that differ only by units.

The Toolkit uses a “type definition” control, as opposed to a LabVIEW “*strict* type definition”, giving user the ability to customize the appearance of the control. The ATA Aerospace Toolkit only restricts the structure and units.

Typedef Quaternion

A quaternion is a structure composed of four real valued elements that is commonly used in aerospace to represent attitude and rotations in three dimensions. This representation avoids problems with singularities that may occur with the more familiar Euler angle representations. Mathematically, quaternions are an extension of complex numbers, that

have four elements i, j, k and s; they were first described by William Hamilton in 1843. An important property of quaternion is that, although they can be multiplied together, multiplication is NOT commutative, i.e., for two quaternions *q1* and *q2*: $q1*q2 \neq q2*q1$.

Quaternions have a defined magnitude (or in strict mathematical terms a modulus) that is a real number. If every element is divided by the modulus the quaternion is said to be normalized. Certain operations using quaternions require a normalized quaternion; the Aerospace toolkit provides a normalization VI called QNormalize (see page 60).

The Aerospace Toolkit Quaternion is a cluster of four doubles.

- i – first imaginary vector component, the square root of -1
- j – second imaginary vector component, also the square root of -1
- k – third imaginary vector component, also the square root of -1
- s – scalar component

Typedef QuaternionDot

It is possible to represent the time rate of change of a quaternion, which is mathematically the same as a quaternion. Since the elements of a quaternion do not have any dimension, the elements of a QuaternionDot, are sec^{-1}.

The Aerospace Toolkit QuaternionDot is a cluster of four doubles.

- i – first vector component of rate of change of a quaternion [sec^{-1}]
- j – second vector component of rate of change of a quaternion [sec^{-1}]
- k – third vector component of rate of change of a quaternion [sec^{-1}]
- s – rate of change of scalar component of a quaternion [sec^{-1}]

Typedef DCM

DCM is for Direction Cosine Matrix. This is a 3X3 matrix of real elements that represents a linear transformation from one 3-dimensional frame to another one. One can multiply a DCM by a vector to transform it to another frame of reference. Since the length of any physical vector used in the toolkit is not dependent upon its frame of reference, mathematics requires that a DCM must be an orthogonal matrix. Mathematically this means the transpose of a DCM is equal to its inverse, i.e. given a DCM $\boldsymbol{A}$, it must be true that $\boldsymbol{A}^T\boldsymbol{A} = \boldsymbol{I}$. Any DCM generated by an Aerospace Toolkit VI is guaranteed to be orthogonal. An externally supplied DCM should be verified with the VI CheckOrthogonality (see page 64). This characteristic is useful because if one has determined a transformation from a frame of reference **B** to another frame **C**; it is possible to determine the reverse transformation (from **C** back to **B**) by performing the significantly less computationally intensive transpose, rather than inverse.

It is a common and encouraged design pattern that given a quaternion representing the attitude of a space vehicle in an inertial frame one can use the QtoDCM VI (see page 60) to generate the Inertial-to-Body DCM, then simply use the LabVIEW Transpose 2D Array VI to also generate the Body-to-Inertial DCM; both are useful in vehicle simulation.

Note that this type definition is an exception to the general rule that Aerospace Toolkit supplied controls are clusters. The supplied control deliberately hides the index displays of the two dimensional matrix to discourage creating a matrix that is greater than 3X3.

DCM is simply a two-dimensional 3X3 array of doubles.

Typedef TimeStampToUTC

The TimeStampToUTC control is the default time interface for the ATA Aerospace Toolkit starting with version 2.1. It allows the user to enter time in either UTC or the local time zone of the computer (as determined by the Windows date and time settings). The default setting for this control is to enter time in UTC. If local time is selected, the input will be assumed to be the time zone of the host Windows computer. If an ATA Aerospace Toolkit function detects that time is being entered in "Local Time", the Time Stamp will then convert time to UTC based on the settings of the computer.

It is possible in Windows to set both the time zone and adjustments for daylight savings time. If the user wishes to work in local time, it is strongly advised that they become familiar with how to set and check these Windows settings.

Typedef UTC Date FracDay

Another representation of Universal Coordinated Time that is well suited for Greenwich Hour Angle and other Earth rotation based calculations. It is a cluster of three doubles, although the first two elements YYYY and MM should be integral values.

This control does NOT use units.

UTC FracDay:

- YYYY – Four digit representation of year in the common era.
- MM – Two digit representation of month Jan=1, Dec=12
- DD(frac) – Day of month plus fraction of day. Noon on the first would be 1.5

Typedef LLH

Geodetic Latitude Longitude and Altitude (above reference ellipsoid). Although underlying algorithms are implemented in radians, the control uses the more user friendly unit of degrees. Conversion of units is handled by.

- Lat – Geodetic Latitude: -90 is S. Pole, +90 is N. Pole ($-90 \leq Lat \leq 90$) [deg]
- Lon – Longitude East from Prime Meridian ($0 \leq Lon \leq 360$)* [deg]
- Alt – Altitude above reference ellipsoid. [m]

*often this constraint is relaxed (e.g. see VI LatLonAltToECI on page 39)

Typedef Cart3D

Cartesian three dimensional vector with no physical units implemented as a structure. This is often used for unit vectors that express only direction. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [ ]
- Y [ ]
- Z [ ]

Typedef CartPos3D

Cartesian three dimensional position vector implemented as a structure. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [m]
- Y [m]
- Z [m]

Typedef CartVel3D

Cartesian three dimensional velocity vector implemented as a structure. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [m/sec]
- Y [m/sec]
- Z [m/sec]

Typedef CartAcc3D

Cartesian three dimensional acceleration vector implemented as a structure.. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [m/sec^2]
- Y [m/sec^2]
- Z [m/sec^2]

Typedef CartAngRate3D

Cartesian three dimensional angular rate vector implemented as a structure. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [rad/sec]
- Y [rad/sec]
- Z [rad/sec]

Typdef SphePos

A representation of position in spherical coordinates. This is a generalized coordinate system.

- Theta - Angle from the +X axis
- Phi – Angle from the XY plane
- Rho – magnitude of the radius

Typdef SpheRate

This is a representation of rate in a spherical coordinate system. The coordinate system is generalized and can apply to any change in rate that can be represented in spherical coordinates.

- ThetaDot – rate of change of theta from with respect to the +X axis
- PhiDot – rate of change or phi with respect to the XY plane
- RhoDot – rate of change of the magnitude of the radius

Typedef Torque

Cartesian three dimensional torque vector implemented as a structure.. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [kg-m^2/sec^2]
- Y [kg-m^2/sec^2]
- Z [kg-m^2/sec^2]

Typdef Force

Representation of a Cartesian 3D linear force. The elements of all Cartesian vectors are designated X, Y and Z, however they can be any orthogonal basis e_1, e_2 and e_3.

- X [N]
- Y [N]
- Z [N]

Typedef Kepler Elements

Kepler Elements are a common way of expressing the orbit of a satellite. The Kepler Elements structure includes the six time-independent elements of a Kepler representation of an orbit.

- a – semi-major axis of the orbit ellipse (used rather than mean motion) [m]
- e – eccentricity ($0 \leq e < 1$) [ ]
- i – inclination: ($0 \leq i \leq \pi$) 0 is an equatorial orbit, $\pi/2$ is a polar orbit [rad]
- o – R.A.A.N. Right Ascension of the Ascending Node ($0 \leq i \leq 2\pi$) [rad]
- w – argument of perigee, 0 means perigee is at R.A.A.N. ($0 \leq w \leq 2\pi$) [rad]
- v – true anomaly ($0 \leq v \leq 2\pi$) [rad]

Typedef Equinoctial

Equinoctial representation of an orbit avoids singularities that can be a problem in classical representations (Orbits with very low inclinations and eccentricities). Many of the elements are dimensionless terms derived from Kepler Elements; it is probably easiest

to understand Equinoctial elements by first understanding Kepler elements and then define the following intermediate value:

wBar = w + o (argument of perigee + right ascension ascending node)

Equinoctial elements:

- a – semi-major axis [m]
- P1 – e*sin(wBar) [ ]
- P2 – e*cos(wBar) [ ]
- Q1 – tan(i/2)sin(o) [ ]
- Q2 – tan(i/2)cos(o) [ ]
- l – mean longitude (w + o + M) [rad] (Argument of perigee + right ascension of the ascending node + mean anomaly)

Typedef ADBARV

ADBARV is an orbital element set; the name is an acronym based on its six components. ADBARV consists of right ascension and declination of the position vector, flight path angle, azimuth from north (angle between the north direction and the projection of the velocity vector onto the plane perpendicular to the geocentric position vector. The angle is measured positive from north clockwise.), radius magnitude, and velocity magnitude.

ADBARV elements:

- Right ascension of the position vector [rad]
- Declination of the position vector [rad]
- Flight path angle [rad] (Defined here as angle between position and velocity vectors)
- Azimuth of velocity vector [rad]
- Radius magnitude [m]
- Velocity magnitude [m/sec]

Typedef Euler Angle

Three angles that define rotations about a sequence of axes (see Typedef Euler Sequence below). Together with an Euler sequence, Euler Angles define a transformation from one orthogonal frame of reference to another. Euler angles are easy to visualize, but can have problems with singularities; therefore quaternion representation is often used in space vehicle simulations.

- phi – first rotation in sequence [rad]
- theta – second rotation in sequence [rad]
- psi – third rotation in sequence [rad]

Typedef Euler Sequence

A sequence of three orthogonal axis (e_1,e_2,e_3) that indicate about which axis to apply the rotation specified in Euler Angles (see above). This is an enumeration that specifies the twelve possible sequences.

Euler Sequence { $1\rightarrow2\rightarrow3$, $1\rightarrow3\rightarrow2$, $2\rightarrow3\rightarrow1$, $2\rightarrow1\rightarrow3$, $3\rightarrow1\rightarrow2$, $3\rightarrow2\rightarrow1$,
$1\rightarrow3\rightarrow1$, $1\rightarrow2\rightarrow1$, $2\rightarrow1\rightarrow2$, $2\rightarrow3\rightarrow2$, $3\rightarrow1\rightarrow3$, $3\rightarrow2\rightarrow3$ }

The first six sequences, those on the first line do not repeat an axis, they are known as type I sequences, there can be singularity issues for interior angles near $\pi/2$. The next six sequences, those on the second line do repeat an axis and are known as type II sequence, they can have singularity issues for interior angles near 0 or π.

Typedef EOP

Earth Orientation Parameters taken from the IERS (International Earth Orientation Services) Bulletin.

- xp – X component of polar motion [arcsecond]
- yp – Y component of polar motion [arcsecond]
- dUT1 – correction from UTC to UT1 (see Time Utilities page 94) [sec]

Typedef Aero Force Coefficients

Aerodynamic Force Coefficients. These are generally empirically determined from wind tunnel tests on a specific vehicle or a computational fluid dynamics model (CFD) and are used to calculate aerodynamic forces based on reference surface area (also determined empirically or by CFD) and dynamic pressure. The forces can then be multiplied by the static margin (difference between center of pressure and center of gravity) to ultimately determine aerodynamic torques. The coefficients themselves are dimensionless.

- CA – Roll Axis Force Coefficient [ ]
- CY – Pitch Axis Force Coefficient [ ]
- CN – Yaw Axis Force Coefficient [ ]

Typedef Aero Torque Coefficients

Aerodynamic Torque Coefficients. These are generally empirically determined from wind tunnel tests on a specific vehicle or a computational fluid dynamics model (CFD) and used to calculate aerodynamic torques. These aerodynamic torques are not related to the static margin, but to a vehicle reference length (also determined empirically or by CFD). The coefficients themselves are dimensionless.

- Cl – Roll Torque Coefficient [ ]
- Cm – Pitch Torque Coefficient [ ]
- Cn – Yaw Torque Coefficient [ ]

Typedef Wind Data

A Data structure for an empirical wind model. The direction and speed of horizontal wind is determined by the first two components, azimuth and speed; the third component can be used to represents an updraft or downdraft component to a wind vector.

- az – Wind direction azimuth from North [rad]
- speed – Horizontal wind speed [m/s]
- Vcomp – Vertical component of wind [m/s]

Typedef Inertia Tensor

Array of Mass Properties. Moments of inertia are on the diagonal. Products of inertia are of the diagonal.

- h = I * Omega
- Tau = I * Alpha

X – Default State Vector

In general a state vector can be any length. For Space Vehicle simulations there is a common design pattern that uses a state vector that has 14 components implemented as an array of doubles. These can be used by the various integrators. For the implementation of state vectors and integrator functions ATA has relaxed its strict policy on physical units; in parentheses are the state vector units used in ATA examples.

- { 0, 1, 2 } Cartesian Position (e.g. [m])
- { 3, 4, 5 } Cartesian Velocity (e.g. [m/sec])
- { 6, 7, 8, 9 } Attitude Quaternion elements i, j, k, s.
- { 10, 11, 12 } Body rates (e.g. [rad/sec])
- { 13 } Mass (e.g. [kg])

Xdot – Default d(State Vector)

The integrators in the Math Analysis Component (page 60) work upon a function implemented as a VI that produces the derivative of a state vector at a specified time. For the implementation of state vectors and integrator functions ATA has relaxed its strict policy on physical units; in parentheses are the state vector units used in ATA examples. This is the definition of the derivative of the state vector that corresponds to the State Vector (default) defined above and so also has 14 components implemented as an array of doubles.

- { 0, 1, 2 } Cartesian Velocity (e.g. [m/sec])
- { 3, 4, 5 } Cartesian Acceleration (e.g. [m/sec^2])
- { 6, 7, 8, 9 } Attitude Quaternion Dot [QuaternionDot] elements i, j, k, s.
- { 10, 11, 12 } Body accelerations (e.g. [rad/sec^2])
- { 13 } Mass Flow (e.g. [kg/sec])

VI Template Force Model State Transition

The integrators in the Math Analysis Component (page 60) integrate a function implemented as a VI called by reference. Such a VI must have the following interface

Inputs:

t – integrator time
state – state vector (see X – default State Vector page 20)

Outputs:

error – computational error in computing derivative of state vector
d(state) – derivative of state vector (see Xdot – default d(State Vector) page 20)

In the 3DOF Ascent Example there exists a non-trivial example of a Force Model (3DOFAscent_Flight_StateTransition.vi). The VI Template ModelStateTransition.vit can be used a starting point for developing new Force Models.

ATA Aerospace Toolkit LabVIEW Virtual Instruments

This section explains in detail each VI contained in the ATA Aerospace Toolkit.

Attitude Analysis Component

The attitude analysis component of the ATA Aerospace Toolkit gives the user the ability to solve a variety of spacecraft attitude analysis and modeling problems. The user can compute "delta attitude quaternions", Euler angles from various local reference frames, compute derivatives of quaternions, compute attitudes from desired pointing directions of spacecraft body axes, integrate attitudes, and generate attitude ephemerides.

With the release of version 2.3 the Aerospace Toolkit now includes slew model profile generation functions. These functions can compute an entire slow profile for a vehicle. These are complicated functions and may take the user a while to master their use. However, when used properly they can compute slew profiles to a high degree of accuracy. Also included is a graphing function to analyze the results of the slew profiles being generated.

VI SlewModelHaversine

This VI is the numerical integration of the haversine slew model in order to create an attitude profile for a space vehicle. The slew model is called haversine, because the angular acceleration magnitude is of the form 1/2*(1.0 - cos(w0*T)). Initial and final angular rates and accelerations are assumed to be 0. This function returns both ephemeris and a single point at a desired time. Angular acceleration limits are enforced.

Inputs:

Path – location of slew profile text file created by the VI
Initial Attitude – Initial attitude represented as a quaternion
Final Attitude – Final attitude represented as a quaternion
Angular Acceleration Limits – Maximum allowable angular accelerations (rad/s^2).
Number of Desired Points – Number of points at which to evaluate the slew model. This will determine how many data points are written to the output file
Time of Evaluation – The slew model time at which the front panel indicators will output data.

Outputs:

Angular Acceleration – Angular acceleration of the slew at the time of evaluation
Angular Rate – Angular rate of the slew at the time of evaluation
Attitude – Attitude of the vehicle at the time of evaluation
Total Slew Time – Total time required for the slew maneuver.

VI SlewModelGeneral

This function is the high fidelity general slew model. It computes an attitude profile for a user specified number of points. Rate, acceleration, and jerk limits are enforced. For slew times longer than a computed value (2.24*sqrt(wLim^3/jLim)), a race, coast, brake profile is computed, in which the vehicle coasts at maximum rate between two intermediate attitudes between the initial and final desired attitudes. In general, initial and final boundary conditions of attitude, rate, and acceleration are met. This function evaluates the interpolating polynomials at a point, as well as generates attitude ephemeris through the duration of the slew.

Inputs:
Path - location and file name to be used for creating the output file.
Initial Quaternion - Quaternion representing the initial attitude.
Final Quaternion - Quaternion representing the final attitude.
Initial Angular Rate - Initial angular rate magnitude in the body frame (rad/sec).
Initial Angular Acceleration - Initial angular acceleration magnitude in the body frame (rad/sec^2).
Final Angular Rate - Final angular rate magnitude in the body frame (rad/sec).
Final Angular Acceleration - Final angular acceleration magnitude in the body frame (rad/sec).
Time of Evaluation - Time of desired profile point (sec).
Number of Points- Number of desired points in attitude ephemeris.
Axes – user defined axes (1,0,0; 0,1,0; 0,0,1) represents the typical X, Y, Z orientation.
- a1 - First unit body axis about which limits are given.
- a2 - Second unit body axis about which limits are given.
- a3 - Third unit body axis about which limits are given.

Angular Rate Limit - Attitude rate limit vector (magnitude about a1, a2, a3)(rad/sec)
Angular Alpha Limit - Attiude acceleration limit vector (magnitude about a1, a2, a3) (rad/sec^2)
Angular Jerk Limit - Attitude jerk limit vector (magnitude about a1, a2, a3) (rad/sec^3)

Outputs:
Slew Time - Total slew time (sec).
Instantaneous Jerk - Instantaneous jerk vector at time tEval (rad/sec^3)
Angular Acceleration - Angular acceleration vector at tEval (rad/sec^2).
Angular Rate - Angular rate vector at tEval (rad/sec)
Instantaneous Slew - Instantaneous slew vector at tEval(Unit axis of rotation times angle of rotation) (rad)
Attitude - Quaternion representing attitude at tEval.

File output:
general.out - File containing the following: time, jerk vector, angular acceleration vector, angular rate vector, slew vector (slew angle times axis of rotation), attitude quaternion.

VI Slew ModelZerotoZero

This function is the zero-to-zero slew model. That is, the initial and final angular rates, accelerations, and jerk are zero. This function returns the pertinent vectors at a user

specified time, as well as generates an attitude ephemeris from the beginning to the end of the attitude slew.

Inputs:

Path - location and file name to be used for creating the output file.
Initial Quaternion - Quaternion representing the initial attitude.
Final Quaternion - Quaternion representing the final attitude.
Axes – user defined axes (1,0,0; 0,1,0; 0,0,1) represents the typical X, Y, Z orientation.
a1 - First unit body axis about which limits are given.
a2 - Second unit body axis about which limits are given.
a3 - Third unit body axis about which limits are given.
Angular Rate Limit - Attitude rate limit vector (magnitude about a1, a2, a3)(rad/sec)
Angular Acceleration Limit - Attiude acceleration limit vector (magnitude about a1, a2, a3) (rad/sec^2)
Angular Jerk Limit - Attitude jerk limit vector (magnitude about a1, a2, a3) (rad/sec^3)
Time of Evaluation - Time of desired profile point (sec).
Number of points - Number of desired points in attitude ephemeris.

Outputs:
Total Slew Time - Total slew time (sec).
Instantaneous Jerk - Instantaneous jerk vector at time tEval (rad/sec^3)
Angular Acceleration - Angular acceleration vector at tEval (rad/sec^2).
Angular Rate - Angular rate vector at tEval (rad/sec)
Instantaneous Slew - Instantaneous slew vector at tEval (Unit axis of rotation time angle of rotation) (rad)
Attitude - Quaternion representing attitude at tEval.

File output:
File containing the following: time, jerk vector, angular acceleration vector, angular rate vector, slew vector (slew angle times axis of rotation), attitude quaternion is created at the location designated by the user.

VI EulerEq

This function is Euler's equations which gives the time evolution of the angular velocity vector in the vehicle body frame in response to a torque vector also expressed in the body frame.

Inputs:

Inertia Tensor- Vehicle inertia tensor (I) in the body frame (kg-m^2)
Torque - Torque vector (Tau) in the body frame (N-m)
Angular Rate - Angular velocity vector (omega) resolved in the body frame (rad/sec)

Outputs:

Angular Acceleration - Angular acceleration vector (alpha) resolved in the body frame (rad/s^2)

VI EhatdthQuaternion

Computes the axis of rotation and slew angle between two attitudes. Euler's theorem states that a rigid body can be brought from an arbitrary initial orientation to an arbitrary final orientation by a single rotation about a judiciously chosen axis fixed in both the initial and final frames by some angle.

Inputs:

qI – Initial attitude quaternion
qF – Final attitude quaternion

Ouputs:

eAxis – Eigen axis of rotation
theta – Rotation angle [rad]

VI EulerFromRIC

Converts a quaternion that represents an attitude in an ECI frame to Euler angles from RIC (radial, in-track, Cross track) coordinates. An RIC frame is defined as follow: the R vector is the unit SV radius, the V vector is a unit vector in the general direction of the orbital velocity vector, and the C vector is a unity vector in the direction of the orbital angular momentum vector. The internal angle ambiguity is resolved both ways and two possible Euler sequences are produced.

Inputs:

r – Position vector in ECI [m]
v – Velocity vector in ECI [m/sec]
q – Quaternion representing attitude in ECI (ECI to body)
seq – Desired Euler sequence (123, 313, 213, etc.). All 12 Euler sequences are supported.

Outputs:

seq1 – First Euler sequence (phi, theta, psi) [rad]
seq2 – Second Euler sequence (phi, theta, psi) [rad]

VI FlipAxis

Ensures that a sequence of quaternions keeps all of the axes of rotation within $\pi/2$ radians of each other. A quaternion in general represents a rotation between two attitudes. There are 4 different way to represent a given rotation θ

- An angle of rotation θ about the axis of rotation,
- $-(2\pi - \theta)$ about the axis of rotation,
- $-\theta$ about the negative axis of rotation, and
- $(2\pi - \theta)$ about the negative axis of rotation.

The first quaternion in the array with a non-zero angle of rotation is used as a reference. If one cannot be found, an error message is returned. This VI is in general used prior to curve fitting a sequence of quaternion components, to prevent discontinuities in the quaternion curve fit.

Inputs:

qInArray – Array of quaternions (not to exceed 1000 elements)

Outputs:

qOutArray – Array of quaternions with eigen axes less than $\pi/2$ from each other.

VI QMinimumRotation

Ensures that two quaternions are the "closest" in attitude to each other. In other words, there are two angles of rotation; θ and $2\pi - \theta$. The angle of rotation is forced to be the smaller rather than the larger.

Inputs:

q1 – Quaternion to compare the rotation angle to
q2 – Quaternion to ensure minimum rotation from q1.

Outputs:

qMin – Quaternion with minimum rotation from q1.

VI QDotDiff

This VI computes the quaternion derivative using a finite differencing method.

Inputs:

q – Quaternion at the desired time of the derivative for forward and backward differencing. It is at t + dt for central differencing.
qDt – Quaternion at some time step from q for differencing. qDt is at t - dt for central differencing.
dt – Time step for computing the derivative (must be positive). [sec]
diff – Type of differencing enumeration { Central, Backward, Forward }

Outputs:

qDot – Derivative of the quaternion at desired time t

VI RateQuaternion

Performs a simple differencing technique to estimate the angular rate vector in the body frame. Forward or backward differencing can be used.

Inputs:

q1 – First unit quaternion to estimate the attitude rate; assumed to be at the time of the desired angular rate vector.
q2 – Second unit quaternion used to estimate the attitude rate; assumed to be dt seconds from q1.
dt – Time step (must be positive) [sec]
diff – Type of differencing enumeration { Forward, Backward }

Outputs:

w – Angular rate vector resolved in the body frame [rad/sec]

VI QDotIncremental

Uses an incremental quaternion to compute the derivative of a quaternion numerically. The incremental quaternion is composed of the axis of rotation between the two quaternions and the angle of rotation about that axis.

Algorithm notes:

$\delta q_i = \sin(\Delta\theta/2)e_1$

$\delta q_j = \sin(\Delta\theta/2)e_2$

$\delta q_k = \sin(\Delta\theta/2)e_3$

$\delta q_s = \cos(\Delta\theta/2)$

$qIncremental_i$ = sin(deltaTheta/2)*e[0]
$qIncremental_j$ = sin(deltaTheta/2)*e[1]
$qIncremental_k$ = sin(deltaTheta/2)*e[2]
$qIncremental_s$ = cos(deltaTheta/2)
e – Axis of rotation.
deltaTheta – Angle of rotation about axis.

Angular rate between the two quaternions is estimated using the incremental quaternion and the time step, which is used to compute the quaternion differential equation.

qDot = ½*[omega]*q

Note: The quaternion represents the inertial to body transformation. Omega is the 4X4 skew symmetric matrix from the angular rate vector.

Inputs:

q – Quaternion at the time the derivative is estimated.
qDt – Quaternion at some time step from q. Forwards for forward differencing, and backwards for backwards differencing.
dt – Step size for numerically taking the derivative. (Always positive) [sec]
diff – Differencing type enumeration for taking numerical derivative{Forward, Backward }

Outputs:

qDot – Quaternion derivative

VI Skew3X3

Sets a 3X3 skew symmetric matrix.

Inputs:

w – Angular rate vector resolved in the body frame [rad/sec]

Outputs:

skew – 3X3 skew symmetric matrix

VI Skew4X4

Computes the derivative of a quaternion using the 4x4 skew symmetric matrix. The quaternion represents a transformation from inertial to body. The output is a 4x4 skew symmetric matrix.

Inputs:

w – Three dimensional angular rate vector [rad/sec]

Outputs:

skew – 4X4 skew symmetric matrix

VI QDotSkew:

Computes the derivative of a quaternion using the 4x4 skew symmetric matrix. The quaternion represents a transformation from inertial to body.
qDot = ½*[omega]*q. Omega is the 4X4 skew symmetric matrix from the angular rate vector.

Inputs:

q – Quaternion to take the derivative of
w – Angular rate vector in body frame [rad/sec]

Outputs:

qDot – Derivative of the input vector expressed in quaternion form.

VI TDotSkew

Computes the derivative of a direction cosine matrix using a 3X3 skew symmetric matrix. The DCM represents the transformation from inertial to body

Inputs:

DCM – Direction cosine matrix
w – Angular rate vector in the body frame [rad/sec]

Outputs:

dcmDot – Derivative of input direction cosine matrix

VI EulerFromNED

This VI computes the Euler angles from a North-East-Down (NED) local coordinate frame to the SV (Space Vehicle) body frame. The desired Euler sequence must be input. In the NED frame, the +X axis points north, the +Y axis points east, and the +Z axis points down along the local vertical. The internal angle ambiguity is resolved both ways and two possible Euler sequences are produced.
The twelve possible Euler sequences are divided into two types.
Type I = { 123, 132, 213, 231, 321, 312 }
Type II = { 121, 131, 212, 232, 313, 323 }

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

r – Position vector of SV in ECI frame [m]

q – Quaternion representing the ECI to body transformation

seq – Euler sequence enumeration

Outputs:

seq1 – First Euler sequence (phi1, theta1, psi1) [rad]

seq2 – Second Euler sequence (phi2, theta2, psi2) [rad]

VI ReferenceGenerator

Computes a direction cosine matrix between any two general reference frames. A target frame (typically ECI) and a point frame (typically body). This VI also checks for singularities. All four of the input vectors are in the same plane. The body alignment and inertial alignment vectors are collinear. The body constraint and inertial constraint vectors are in the same plane in the same general direction. The constraint vectors serve to define the plane for all four vectors.

Inputs:

tAlign –The desired vector in the target reference frame that the desired vector in the point reference frame is aligned with.

tPlanar – The vector in the target frame that along with tAlign vector constitutes the plane that the desired vector in the point frame is to lie in.

pAlign – The vector in the point frame that will align with tAlign.

pPlanar – The vector in the point frame that will lie in the plane determined by tAlign and tPlanar, in the general direction of tPlanar.

Note: No two of these vectors need be orthogonal

Outputs:

dcm – Target to point frame direction cosine matrix. (Usually inertial to body)

VI QIntegrator

Integrates a quaternion using second order Taylor series closed form expression. The quaternion is inertial to body (scalar last). The rate vector is expressed in the body frame and is assumed to be average over the integration step.

Algorithm notes:

qDt = (cos(wMag*dt/2)*I + 1/wMag*sin(wMag*dt/2)*OMEGA)*q0

I – 4X4 Identity matrix

OMEGA – 4X4 skew symmetric matrix

wMag – Magnitude of the angular rate vector [rad/sec]

dt – Time step [sec]

Inputs:

q0 – Initial quaternion (inertial to body)

w – Average angular rate vector in body frame [rad/sec]
dt – Step size (Always positive) [sec]

Outputs:
qDt – Quaternion at end of integration step (inertial to body)

VI QIntegIncremental

This VI propagates a quaternion using an incremental quaternion
q(t)*qIncremental = q(t+dt)
The incremental quaternion is composed of the axis of rotation between the two quaternions and the angle of rotation about that axis.
$qIncremental_i$ = sin(deltaTheta/2)*e[0]
$qIncremental_j$ = sin(deltaTheta/2)*e[1]
$qIncremental_k$ = sin(deltaTheta/2)*e[2]
$qIncremental_s$ = cos(deltaTheta/2)
e - Axis of rotation
deltaTheta - Angle of rotation about axis.
Axis of rotation: e = w/|w|.
Angle of rotation: deltaTheta = |w|*dt

Inputs:
q – Initial quaternion (Inertial to body)
w – Angular rate vector resolved in the body frame [rad/sec]
dt – Time step (Always Positive) [sec]

Outputs:
qDt – Quaternion at dt seconds later (Inertial to body)

VI QIntegExponential

Integrates a quaternion using e raised to a matrix power (matrix exponential)
q(t+dt) = exp(omega*t/2)*q(t). Omega is the 4X4 skew symmetric matrix from the angular rate vector.

Inputs:
q – Initial quaternion (Inertial to body)
w – Angular rate vector resolved in the body frame [rad/sec]
dt – Time step (Always Positive) [sec]

Outputs:
qDt – Quaternion at dt seconds later (Inertial to body)

VI EulerKinematicEquations

This VI is the Euler kinematical equations of motion. All 12 possible Euler sequences are represented. This VI can be used to convert an angular rate vector in the body frame to

Euler angle rates, or it can be used to integrate the Euler angular rates and compute an attitude ephemeris. Initial Euler angles must be known.
Type I sequences: { 123, 132, 213, 231, 321, 312 }
Type II sequences: { 121, 131, 212, 232, 313, 323 }

Inputs:

w – Angular rate vector in the body frame [rad/sec]
seq – Euler sequence enumeration (123, etc.)
type – Euler sequence type enumeration {Type I, Type II }
eulerAngles - Vector containing the initial Euler angles (phi, theta, psi) [rad]

Outputs:

eulerRates – Vector containing the Euler body axis angular rates relative to the inertial frame [rad/sec]

VI EulerFromLVOP

This function converts a quaternion that represents an attitude relative to an ECI frame to Euler angles from LVOP (Local Vertical Orbital Plane) coordinates. LVOP is defined as follows:
+X - In the general direction of the orbital velocity vector
+Y - Anti normal (Negative of unit orbital angular momentum)
+Z - NADIR (Negative of position vector)

Inputs:

r - Position vector in ECI (m)
v - Velocity vector in ECI (m/sec)
q - Quaternion representing attitude relative to ECI (ECI to body)
seq - Desired Euler sequence (3 dimensional vector of integers)

Outputs:

seq1 - First Euler sequence (phi, theta, psi) (rad)
seq2 - Second Euler sequence (phi, theta, psi) (rad)

VI EulerEquations

This function is Euler's equations which give the time evolotion of the angular velocity vector in the vehicle body frame in responce to a torque vector also expresed in the body frame.

Inputs:

Inertia Tensor- Vehicle inertia tensor (I) in the body frame(kg-m^2)
Torque - Torque vector (Tau) in the body frame (N-m)
Angular Rate - Angular velocity vector (omega) resolved in the body frame (rad/sec)

Outputs:

Angular Acceleration - Angular acceleration vector (alpha) resolved in the body frame (rad/s^2)

VI SlewProfileAnalysis

This VI assists in the analysis of slew profiles produced by the SlewModel VIs included in this tool kit. It will read a text file produced by the slew model VIs, open the appropriate graphing subVI and graph data in the designated file.

Inputs:

Type - type of slew profile to be analyzed (general, haversine, or zero-to-zero)
Path - location of the file to be analyzed

Outputs:

None.

Coordinate Frame Transformation Component

The coordinate frame component gives the user the ability to express state vectors in a variety of commonly used coordinate systems. Among those provided are ECI J2000 (also often referred to as EMEJ2000) Mean of Epoch, Mean of Date, True of Date, and others.

Two inertial frames are represented in the ATA Aerospace Toolkit. ECIJ2000 is a high-fidelity reference frame defined below. This frame takes into account full Earth precession and nutation. The second inertial frame is a "simpler" reference frame based on Bate, Mueller, White. While not as accurate as the ECIJ2000 frame, many users may find this computationally efficient frame useful for much of their routine analysis work.

VI ECEFtoJ2000

Implements the transformation from ECEF (Earth Centered Earth Fixed) to ECI (Earth Centered Inertial) J2000 Mean of Epoch. ECEF is the International Terrestrial Reference System (ITRS). The algorithm employs 1976 IAU (International Astronomical Union) precession theory and IAU 1980 nutation theory. For EME ECI J2000, the fundamental plane is the mean earth equator of J2000, and the x axis is aligned with the mean equinox of J2000. The center is the center of mass of the earth. The ITRS has the center as the center of mass of the earth (including oceans and atmospheres). The x axis is the intersection of the mean equator, and the mean prime meridian. The z axis is the mean spin axis of the earth between 1900 until 1905 (Conventional International Origin - CIO).

Inputs:

- rECEF – Position vector in ECEF [m]
- vECEF – Velocity vector in ECEF [m/sec]
- TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
- EOPData - Earth orientation parameters
  - xp – x component of polar motion [arcseconds]
  - yp – y component of polar motion [arcseconds]
- dUT1 – Correction from utc to ut1 [sec]
  - Note: These values can be found in the IERS (International Earth Rotation Service) bulletins.

Outputs:

- rECI – Position in ECI J2000 [m]
- vECI – Velocity in ECI J2000 [m/sec]

VI ECIJ2000toECEF

Implements the transformation from ECI (Earth Centered Inertial) J2000 Mean of Epoch to ECEF (Earth Centered Earth Fixed). ECEF is International Terrestrial Reference System (ITRS). This VI employs 1976 IAU precession theory and IAU 1980 nutation theory.

Inputs:

rECI – Position vector in ECI [m]
vECI – Velocity vector in ECI [m/sec]
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
EOPData – Earth orientation parameters
xp – x component of polar motion (arcseconds)
yp – y component of polar motion (arcseconds)
dUT1 – Correction from UTC to ut1 (sec)
Note: These values can be found in the IERS (International Earth Rotation Service) bulletins.

Outputs:

rECEF – Position in ECEF (m).
vECEF – Velocity in ECEF (m/sec).

VI ECIJ2000toB1950:

Converts from ECI J2000 MOE to Besselian Epoch of 1950 coordinate system.

Inputs:

rJ2000 – Position in ECI J2000 MOE [m]
vJ2000 – Velocity in ECI J2000 MOE [m/sec]

Outputs:

rB1950 – Position in B1950 [m]
vB1950 – Velocity in B1950 [m/sec]

VI B1950toJ2000:

Converts from Besselian Epoch of 1950 coordinate system to ECI J2000 MOE.

Inputs:

rB1950 – Position in B1950 [m]
vJ2000 – Velocity in B1950 [m/sec]

Outputs:

rJ2000 – Position in ECI J2000 MOE [m]
vJ2000 – Velocity in ECI J2000 MOE [m/sec]

VI J2000toMOD:

Converts from ECI J2000 (Mean equator and mean equinox of J2000) to mean-of-date coordinate system. The x axis points to the mean equinox of date, and the fundamental plane is defined by the mean equator of date for the MOD frame.

Inputs:

rJ2000 – Position in ECI J2000 MOE [m]

vJ2000 – Velocity in ECI J2000 MOE [m/sec]

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

rMOD – Position in Mean of Date [m]

vMOD – Velocity in Mean of Date [m/sec]

VI MODtoJ2000

Converts from Mean of date (Mean equator and mean equinox of date) to ECIJ2000 (Mean equator and mean equinox of J2000)

Inputs:

rMOD – Position in Mean of Date [m]

vMOD –Velocity in Mean of Date [m/sec]

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

rJ2000 – Position in ECI J2000 MOE (m)

vJ2000 – Velocity in ECI J2000 MOE (m/sec)

VI J2000toTOD

Converts from ECI J2000 (Mean equator and mean equinox of J2000) to true-of-date coordinate system. The x axis points to the true equinox of date, and the fundamental plane is defined by the true equator of date for the TOD frame.

Inputs:

rJ2000 – Position in ECI J2000 MOE [m]

vJ2000 – Velocity in ECI J2000 MOE [m/sec]

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

rTOD – Position in True of Date [m]

vTOD – Velocity in True of Date [m/sec]

VI TODtoJ2000

Converts from True of date (True equator and true equinox of date) to ECIJ2000 (Mean equator and mean equinox of J2000)

Inputs:

rTOD – Position in True of Date [m]
vTOD – Velocity in True of Date [m/sec]
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

rJ2000 – Position in ECI J2000 MOE [m]
vJ2000 – Velocity in ECI J2000 MOE [m/sec]

VI ECIJ2000toUDTopo

Implements the transformation from ECI J2000 MOE to a user defined topocentric coordinate frame. User defined topocentric coordinate frame is defined to be centered at a specified latitude, longitude, and altitude with the x axis pointed south, the z axis pointed perpendicular to the tangent plane away from the center of the earth, and y axis completes the RHS.

Inputs:

rECI – Position vector in ECI J2000 [m]
vECI – Velocity vector in ECI J2000 [m/sec]
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
EOPData –
xp – x component of polar motion [arcseconds]
yp – y component of polar motion [arcseconds]
dUT1 - Correction from UTC to ut1 [sec]
Note: These values can be found in the IERS (International Earth Rotation Service) bulletins.
lat – User defined latitude of topocentric frame [rad]
lon – User defined longitude of topocentric frame [rad]
alt – User defined altitude of topocentric frame [m]

Outputs:

rTOPO – Position in user defined topocentric frame [m]
vTOPO – Velocity in user defined topocentric frame [m/sec]

VI UDTopoToECIJ2000

Implements the transformation from a user defined topocentric coordinate frame to ECI J2000 MOE. User defined topocentric coordinate frame is defined to be centered at a specified latitude and longitude with the x axis pointed south, the z axis pointed perpendicular to the tangent plane away from the center of the earth, and y axis completes the RHS.

Inputs:

rTOPO – Position vector in topocentric coordinate frame [m]
vTOPO – Velocity vector in topocentric coordinate frame [m/sec]
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
EOPData –
xp – x component of polar motion [arcseconds]
yp – y component of polar motion [arcseconds]
dUT1 – Correction from UTC to ut1 [sec]
Note: These values can be found in the IERS (International Earth Rotation Service) bulletins.
lat – User defined latitude of topocentric frame [rad]
lon – User defined longitude of topocentric frame [rad]
alt – User defined altitude of topocentric frame [m]

Outputs:
rJ2000 – Position in ECI J2000 [m]
vJ2000 – Velocity in ECI J2000 [m/sec]

VI ECIToRIC

Implements the transformation from an inertial frame to radial, in-track, cross-track frame (RIC). The radial basis vector is the unit vehicle radius. The in-track basis vector is in the general direction of the velocity vector perpendicular to the radial vector in the orbital plane. The cross track basis vector is the unit angular momentum vector. This is a rotating local frame, therefore, the angular rotation must be considered in the transformation of the velocity. The derivative of the transformation is estimated numerically by estimating the position and velocity a small time step into the future using a quadratic expansion ($r = r0 + v0*t + 1/2*a*t^2$), ($v = v0 + a*t$)
of the equations of motion, which yields the transformation a small time step into the future, then the derivative of the transformation is estimated using the finite divided difference.

This transformation only addresses orientation, not translation.

Inputs:
rI – Postion vector in inertial frame to transform [m]
vI – Velocity vector in inertial frame to transform [m/sec]
rIV – Position of the vehicle in the inertial frame [m]
vIV – Velocity of the vehicle in the inertial frame [m/s]
aTIV – Total acceleration of the vehicle in the inertial frame. Includes acceleration due to thrust, aerodynamics, and gravity.

Outputs:
rRIC – Position expressed in RIC frame [m]
vRIC – Velocity expressed in RIC frame [m/s]

VI RICToECI

Implements the transformation from a Radial, in-track, cross-track frame (RIC) to inertial frame. For RIC, the radial basis vector is the unit vehicle radius. The in-track basis vector is in the general direction of the velocity vector perpendicular to the radial vector in the orbital plane. The cross track basis vector is the unit angular momentum vector. This is a rotating local frame, therefore, the angular rotation must be considered in the transformation of the velocity. The derivative of the transformation is estimated numerically by estimating the position and velocity a small time step into the future using a quadratic expansion (r = r0 + v0*t + 1/2*a*t^2), (v = v0 + a*t) of the equations of motion, which yields the transformation a small time step into the future, then the derivative of the transformation is estimated using a finite divided difference.

This transformation only addresses orientation, not translation.

Inputs:

rRIC – Position vector in RIC frame to transform [m]
vRIC – Velocity vector in RIC frame to transform [m/sec]
rIV – Position of the vehicle in the inertial frame [m]
vIV – Velocity of the vehicle in the inertial frame [m/s]
aTIV – Total acceleration of the vehicle in the inertial frame. Includes acceleration due to thrust, aerodynamics, and gravity

Outputs:

rI – Position expressed in inertial frame [m]
vI – Velocity expressed in inertial frame [m/s]

VI ECIToNED

Implements the transformation from an inertial frame to North, East, Down frame (NED). The North basis vector is the unit vector pointing due north. The East basis vector is the unit vector pointing due east. The down basis vector is the unit vector pointing down along the local vertical. This is a rotating local frame, therefore, the angular rotation must be considered in the transformation of the velocity. The derivative of the transformation is estimated numerically by estimating the position and velocity a small time step into the future using a quadratic expansion (r = r0 + v0*t + ½*a*t^2), (v = v0 +a*t) of the equations of motion, which yields the transformation a small step into the future, then the derivative of the transformation is estimated using a finite divided difference.
Note: The inertial frame referenced here is the simple inertial frame referenced in Bate, Mueller, and White.

This transformation only addresses orientation, not translation.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
rI – Position vector in inertial frame to transform [m]

vI – Velocity vector in inertial frame to transform (m/sec)
rIV – Position of the vehicle in the inertial frame [m]
vIV – Velocity of the vehicle in the inertial frame [m/s]
aTIV – Total acceleration of the vehicle acceleration due to thrust, Aerodynamics, and gravity.

Outputs:
rNED – Position expressed in NED frame [m]
vNED – Velocity expressed in NED frame [m/s]

VI NEDToECI

Implements the transformation from a North, East, Down frame (NED) to inertial frame. The North basis vector is the unit vector pointing due north. The East basis vector is the unit vector pointing due east. The down basis vector is the unit vector pointing down along the local vertical. This is a rotating local frame, therefore, the angular rotation must be considered in the transformation of the velocity. The derivative of the transformation is estimated numerically by estimating the position and velocity a small time step into the future using a quadratic expansion ($r = r0 + v0*t + 1/2*a*t^2$), ($v = v0 + a*t$) of the equations of motion, which yields the transformation a small time step into the future, then the derivative of the transformation is estimated using a finite divided difference. Note: The inertial frame referenced here is the simple inertial frame referenced in Bate, Mueller, and White.

Inputs:
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
rNED – Position vector in NED frame to transform [m]
vNED – Velocity vector in NED frame to transform [m/sec]
rIV – Position of the vehicle in the inertial frame [m]
vIV – Velocity of the vehicle in the inertial frame [m/s]
aTIV – Total acceleration of the vehicle in the inertial frame. Includes acceleration due to thrust aerodynamics, and gravity.

Outputs:
rI – Position expressed in inertial frame [m]
vI – Velocity expressed in inertial frame [m/s]

VI LatLonAltToECI

VI to convert lat, lon, alt to an ECI position vector. The algorithm is from *Fundamentals of Astrodynamics* by Bate, Mueller and White. For the ECI frame, the +X axis points to the mean Vernal Equinox of date. The +Z axis is the instantaneous spin axis of the earth. +Y completes the RHS.

Inputs:
LLH [Geodetic Cluster]
lat – Geodetic latitude of the position ($0° \leq lat \leq 90°$) [deg]

lon – Longitude of the position (0° ≤ lon ≤ 360°)* [deg]
alt - altitude above reference ellipsoid [m]

date [Date Cluster]
yyyy – Four digit year in common era, integral value
month – 1 based integral value (Jan=1, Dec=12)
dd(frac) – day + fraction (1 based, so noon on the first is 1.5)

allowWrap – Allow input longitude to be outside range 0° ≤ lon ≤ 360°, without generating an error (the VI ModuloAngle is used internally, see page 76)

Outputs:

rECI – position vector in ECI frame [m]

VI ECIToLatLonAlt

Converts ECI position to latitude, longitude and altitude.

Inputs:

r – Position in ECI frame [m]

date [Date Cluster]
yyyy – Four digit year in common era, integral value
month – 1 based integral value (Jan=1, Dec=12)
dd(frac) – day + fraction (1 based, so noon on the first is 1.5)

Outputs:

LLH [Geodetic Cluster]
lat – Geodetic latitude of the position (0° ≤ lat ≤ 90°) [deg]
lon – Longitude of the position (0° ≤ lon ≤ 360°) [deg]
alt – altitude above reference ellipsoid [m]

VI LatLonAltToECEF

Converts geodetic latitude, longitude, and an altitude above the reference ellipsoid to an earth centered earth fixed frame. International Terrestrial Reference Frame (ITRF).
z axis - Mean position of earth spin axis between 1900 and 1905 Conventional International Origin (CIO).
x axis - Intersection of prime meridian and earth equator
y axis - Completes RHS.

Inputs:

LLH [Geodetic Cluster]
lat – Geodetic latitude of the position (0° ≤ lat ≤ 90°) [deg]
lon – Longitude of the position (0° ≤ lon ≤ 360°) [deg]
alt – altitude above reference ellipsoid [m]

Outputs:

rECEF – Position vector in ECEF frame.

VI ECEFtoLatLonAlt

Converts a position vector in ECEF to a latitude, longitude, and altitude.

Inputs:

rECEF – position vector in ECEF (meters)

Outputs:

LLH [Geodetic Cluster]

lat – Geodetic latitude of the position ($0° \leq lat \leq 90°$) [deg]
lon – Longitude of the position ($0° \leq lon \leq 360°$) [deg]
alt – altitude above reference ellipsoid [m]

VI LatLonAltToState

This function takes user inputs of latitude, longitude, and altitude at a time, and computes the position of the vehicle. The user can specify if velocity should be computed as Wearth X rECI, or computed from Vertical and horzontal components, and direction from north for the horizontal component. Attitude is specified relative to the local vertical by a tilt angle, and azimuth from north of the tilt angle.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
lat - Geodetic latitude (-pi to pi) (rad).
lon - Longitude (0 to 2*pi) (rad).
alt - Altitude above reference ellipsoid (m).
cmpVel - Flag to indicate if velocity should be computed as Wearth X rECI.
Y - Compute velocity
N - Velocity based on user input parameters.
vVelMag - Velocity magnitude relative to vehicle sub point in the vertical direction m/s. (Up is positive, and down is negative)
hVelMag - Horizontal velocity magnitude relative to vehicle subpoint (m/s).
hVelAz - Azimuth from north for direction of horizontal velocity component (rad).
tiltAz - Azimuth from north direction to tilt the roll axis of the vehicle (rad).
Note: +Y axis is 90 degrees from tilt Az upon final alignment.
tiltPhi - Angle from local vertical to tilt roll axis of vehicle (rad).

Outputs:

initPosI - Position in inertial space (m).
initVelI - Velocity in inertial space (m/s).
qIB - Quaternion representing the inertial to body attitude

Orbit Analysis Component

The orbit analysis component allows the user to perform various analyses on the orbits of spacecraft. It allows the user to transform orbit state vectors between various representations such as Cartesian (position and velocity) and Kepler (Semi-major axis, eccentricity, inclination, longitude of the ascending node, argument of perigee, and true anomaly). It allows the user to gain insight into complex rendezvous maneuvers by solving Lambert's problem. It also allows the user to compute various orbital quantities such as flight path angle and orbital period and compute the position of the sun and moon at any given time.

VI SphereGravity

Computes gravity based on a simple spherical (two-body) gravity.

Inputs:

r – Position vector in ECI space [m]

Outputs:

accGrav – Acceleration due to gravity [m/sec^2]

VI J2Gravity

Computes acceleration due to gravity using an oblate spheroid earth model (J2) in an ECI frame.

Inputs:

r – ECI position vector [m]

Outputs:

aG – Acceleration due to gravity [m/sec^2]

VI VintiJ6Gravity

Computes gravitational acceleration using Vinti theory out to J6 zonal harmonics in an ECI frame.

Inputs:

r – Position vector in ECI [m]

Outputs:

aG – Acceleration vector in ECI [m/sec^2]

VI FlightPathAngle

Computes the flight path angle. Here, flight path angle is the angle between the local vertical (Perpendicular to the position vector) and the velocity vector.

Inputs:

r – Position vector in geocentric coordinates [m]
v – Velocity vector in geocentric coordinates [m/s]

Outputs:

fpa – Flight path angle [rad] ($0 \leq fpa \leq 2\pi$)

VI ElementsToFlightPathAngle

Computes the flight path angle without using vectors.

Inputs:

a – Semi-major axis [m]
e – Eccentricity [ ] ($0 \leq e \leq 1$)
v – True anomaly [rad]
vMag - Velocity Magnitude [m/sec]

Outputs:

fpa – Flight path angle [rad] ($0 \leq fpa \leq 2\pi$)

VI PeriodToSemiMajorAxis

Takes the orbital period and computes the corresponding semi-major axis.

Inputs:

per – Orbital period [sec]

Outputs:

sma – Semi-major axis [m]

VI SMAToMeanMotion

Takes the semi-major axis and computes the corresponding mean motion

Inputs:

sma – Semi-major axis [m]

Outputs:

n – Mean motion [rad/sec]

VI SMAToPeriod

Takes the semi-major axis and computes the corresponding orbital period

Inputs:

sma – Semi-major axis [m]

Outputs:

per – Orbital period [sec]

VI SunMeanLongitude

Computes the mean longitude of the sun.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

sunLong – Mean longitude of the sun [rad]

VI EccentricToTrueAnomaly

Converts the eccentric anomaly to the true anomaly.

Inputs:

E – Eccentric anomaly [rad] ($0 \leq E \leq 2\pi$)
e – Eccentricity [ ] ($0 \leq e < 1$)

Outputs:

v – True anomaly [rad]

VI TrueToEccentricAnomaly

This VI computes the eccentric anomaly from the true anomaly.

Inputs:

e – Eccentricity of the orbit [ ] ($0 \leq e < 1$)
v – True anomaly of the orbit [rad] ($0.0 \leq v \leq 2\pi$)

Outputs:

E – Eccentric anomaly [rad]

VI MeanToTrueAnomaly

Convert's mean anomaly to true anomaly. Kepler's equation is solved iteratively for the eccentric anomaly. The eccentric anomaly is then used to find the true anomaly.

Inputs:

m – Mean anomaly [rad] ($0.0 \leq m \leq 2\pi$)
e – Orbital eccentricity [ ] ($0 \leq e < 1$)

Outputs:

tru – True anomaly [rad]

VI TrueToMeanAnomaly

This VI converts the mean anomaly to the true anomaly.

Inputs:

e – Eccentricity [ ] ($0 \leq e < 1$)
v – True anomaly [rad] ($0.0 \leq v \leq 2\pi$)

Outputs:

m – Mean anomaly [rad]

VI CartesianToEhnVectors

This VI computes the eccentricity vector, the angular momentum vector, and the node vector of an elliptical orbit. If the orbit is equatorial and eccentric, the node vector is aligned along the x axis. If the orbit is equatorial and circular, the node vector and the eccentricity vector are aligned along the x axis. If the orbit is circular and inclined, the eccentricity vector is aligned with the node vector. This VI also computes the eccentricity of the orbit, and the inclination.

Inputs:

v – Velocity in Cartesian geocentric frame [m/sec]
r – Position in Cartesian geocentric frame [m/sec]

Outputs:

eVec – Eccentricity vector [ ]
hVec – Angular momentum vector [m^2/sec]
nVec – Node vector (Points to ascending node and has unit length) [ ]
e – Eccentricity of the orbit [ ]
i – Inclination of the orbit [ ]

VI CartesianToKepler

Converts a position and velocity vector in a geocentric Cartesian reference frame to a set of Keplerian elements (a, e, i, o, w, v). For an equatorial orbit, o = 0. For an equatorial and circular orbit o = 0, and w = 0. For a circular and inclined orbit, w = 0.
a - semi-major axis (meters)
e - eccentricity

i - inclination (radians)
o – Right ascension of the ascending node (radians)
w - Argument of perigee (radians)
v - True anomaly (radians)

Inputs:
r – Position vector [m]
v – Velocity vector [m/sec]

Outputs:
kepVec – Kepler Elements [Kepler Element Structure]

VI KeplerToCartesian

Takes a set of Kepler elements, and converts them to position and velocity in a geocentric Cartesian reference frame. The following special cases are addressed:
1) Circular inclined orbit (argument of latitude = w + v)
2) Equatorial elliptical orbit (longitude of periapses = o + w)
3) Equatorial circular orbit (true longitude = o + w + v)

Inputs:
kepVec [Kepler Element Structure]
a – Semi-major axis [m]
e – Eccentricity [ ]
i – inclination [rad]
o – longitude of the ascending node [rad]
w – argument of perigee [rad]
v – true anomaly [rad]

Outputs:
r – Position vector [m]
v – Velocity vector [m]

VI Lambert

Implements a solution to Lambert's problem. The solution to Lambert's problem finds the velocities at two points on some orbit given a transfer time between the two positions.

Inputs:
r1 – 3 dimensional vector representing the initial position [m]
r2 – 3 dimensional vector representing the final position [m]
xinc – Inclination of transfer orbit (used for 180 degree transfers) [rad]
dt – Orbital transfer time [sec]
motion – Direction of orbit { Posigrade, Retrograde }

Outputs:
v1 – velocity at position 1 [m/s]
v2 – velocity at position 2 [m/s]

VI AnalyticalSunPosition

Implements an analytic low fidelity expression for the position of the sun in ECI J2000 mean of epoch coordinates. The approximation is valid for years 1950 through 2050.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

rSun – Position of the sun in ECI J2000 coordinates [m]

VI AnalyticalMoonPosition

Implements an analytic low fidelity expression of the moon position in ECI J2000 MOE. The accuracy is within 0.3 degrees in ecliptic longitude, 0.2 degrees in ecliptic latitude, and 1,275 km in distance.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

rMoon – Position of the moon in ECI J2000 MOE [m]

VI CircularSatelliteSpeed

Computes the circular satellite speed.

Inputs:

rad – Orbit radius [m]

Outputs:

speed – speed of the satellite [m/sec]

VI CartesianToADBARV

Converts a Cartesian position and velocity to ADBARV element representation. ADBARV consists of right ascension and declination of the position vector, flight path angle, azimuth from north (angle between the north direction and the projection of the velocity vector onto the plane perpendicular to the geocentric position vector. The angle is measured positive from north clockwise.), radius magnitude, and velocity magnitude.

Inputs:

r – Cartesian position vector [m].
v – Cartesian velocity vector [m/sec]

Outputs:
ADBARVVec - ADBARV elements [ADBARV]
Right ascension of the position vector [rad]
Declination of the position vector [rad]
Flight path angle [rad] (Defined here as angle between position and velocity vectors)
Azimuth of velocity vector [rad]
Radius magnitude [m]
Velocity magnitude [m/sec]

VI ADBARVToCartesian

Implements conversion from ADBARV elements to Cartesian elements. ADBARV consists of right ascension and declination of the position vector, flight path angle, azimuth from north (angle between the north direction and the projection of the velocity vector onto the plane perpendicular to the geocentric position vector. The angle is measured positive from north clockwise.), radius magnitude, and velocity magnitude.

Inputs:
ADBARVVec - ADBARV elements [ADBARV]
Right ascension of the position vector [rad]
Declination of the position vector [rad]
Flight path angle [rad] (Defined here as angle between position and velocity vectors)
Azimuth of velocity vector [rad]
Radius magnitude [m]
Velocity magnitude [m/sec]

Outputs:
r – Cartesian position vector [m]
v – Cartesian velocity vector [m/sec]

VI KeplerTimeOfFlight

This VI computes Kepler's time of flight. This is the time of flight between two points on an elliptical orbit. The eccentricity is assumed to be between 0 and 1. Multiple revolutions are considered.

Inputs:
a – Semi-major axis of the orbit [m]
e – Eccentricity of the orbit [ ] ($0 \leq e \leq 1$)
E0 – Initial position eccentric anomaly [rad]
E – Final position eccentric anomaly [rad]
k – Constant to account for multiple revs (The number of times the orbit passes through perigee)

Outputs:
tof – Time of flight [sec]

VI CartesianToEquinoctial

Implements a conversion from Cartesian elements to non-singular (equinoctial) elements. Non-singular elements are used to represent an orbit where classical elements present singularities (i.e. Orbits with very low inclinations and eccentricities).

Inputs:
r – Cartesian position vector [m]
v – Cartesian velocity vector [m/sec]

Outputs:
equVec – Equinoctial Elements [Equinoctial] (see def. page 17)

VI EquinoctialToCartesian

Implements the conversion from non-singular (equinoctial) elements to Cartesian elements. Non-singular elements are used to represent an orbit where classical elements present singularities (i.e. Orbits with very low inclinations and eccentricities).

Inputs:
equVec – Equinoctial Elements [Equinoctial] (see def. page 17)

Outputs:
r – Cartesian position vector [m]
v – Cartesian velocity vector [m/sec]

VI DetermineLineOfSite

This VI determines if a line of sight exists between two objects in space. It does so by forming the parametric representation of a line between the two objects and finding the minimum parameter that brings the perpendicular to the line closest to the earth. If this is above the earth's surface, then a line of sight exists between the two objects. Note: This function is conservative since it assumes that the earth is spherical with radius equal to the equatorial radius of the earth. In other words, this function will yield more falses that actually occur, since the polar radius is about 20 kilometers smaller than the equatorial radius. Also, if one or both of the position vectors are inside the radius of the earth, this algorithm may yield erroneous results.

Inputs:
r1 – First object position vector [m]
r2 – Second object position vector [m]

Outputs:
losFlag – Line Of Sight exists between two objects [bool]

VI OblateLOS

This VI determines if there is line of site between two points in space around an oblate earth that is characterized by an equatorial axis and a polar axis as defined by WGS84. Both points must be "above" the surface of the oblate earth. The function returns a boolean value indicating if line of site exists. This function is valid for a cartesian coordinate system with its origin at the center of the earth.

Inputs:
r1 - Position of a point in an earth fixed cartesian coordinate system (m).
r2 - Position of a second point in an earth fixed cartesian coordinate system(m).

Outputs:
Line of Site - Boolean indicating if a line of site between the two points exists.

VI MeanMotionToSMA

This function converts mean motion to semi-major axis.

Inputs:
n - Mean motion (rad/sec)

Outputs:
sma - Semi-major axis (m)

VI CartesianToGeodetic

This function is the conversion from the Cartesian representation of an orbit (r, v), to the geodetic representation of an orbit. The geodetic representation is latitude, longitude, altitude, with the corresponding derivatives (latDot, lonDot, altDot). The position and velocity are assumed to be in ECIJ2000.

Inputs:
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
EOPData - Pointer to array of earth orientation parameters (in order)
xp - x component of polar motion (arcseconds)
yp - y component of polar motion (arcseconds)
dUT1 - Correction from utc to ut1 (sec)
rJ2000 - Position vector in ECIJ2000 (m)
vJ2000 - Velocity vector in ECIJ2000 (m/sec)

Outputs:
geoVec - Vector containing geodetic state representation of r and v.
phi - Geodetic latitude (rad).
lambda - Longitude (rad).
h - Altitude (m)
phiDot - Time rate of change of geodetic latitude (rad/sec)
lambdaDot - Time rate of change of latitude (rad/sec)

hDot - Time rate of change of altitude (m/sec)

VI GeodeticToCartesian

This function is the conversion of the geodetic representation of an orbit to the Cartesian representation of the orbit. The geodetic representation is latitude, longitude, altitude, with the corresponding derivatives (latDot, lonDot, altDot). These are assumed to be with respect to ITRS (International Terrestrial Reference System). The position and velocity are converted to ECIJ2000.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

EOPData - Pointer to array of earth orientation parameters (in order)

xp - x component of polar motion (arcseconds)

yp - y component of polar motion (arcseconds)

dUT1 - Correction from utc to ut1 (sec)

phi - Geodetic latitude (rad).

lambda - Longitude (rad).

h - Altitude (m)

phiDot - Time rate of change of geodetic latitude (rad/sec)

lambdaDot - Time rate of change of latitude (rad/sec)

hDot - Time rate of change of altitude (m/sec)

Outputs:

rJ2000 - Position vector in ECIJ2000 (m)

vJ2000 - Velocity vector in ECIJ2000 (m/sec)

VI NodeRegress

This function is the computation of nodal regression due to Earth oblateness. Nodal regression is movement of the line of nodes opposite Earth rotation.

Inputs:

Semi-Major Axis - Semi-major axis (m)

Eccentricity - Orbit eccentricity

Inclination - Inclination (rad)

Outputs:

Omega Dot - Nodal regression rate (rad/sec)

VI OrbitRateVectors

This function computes the orbital angular rate vector and orbital angular acceleration vector.

Inputs:

r - Geocentric position vector in inertial coordinates (m)
v - Geocentric velocity vector in inertial coordinates (m/sec)

Outputs:

Angular Rate – three deminsional rotational rate vector in inertial coordinates (rad/sec)

Angular Acceleration - three deminsional rotational acceleration vector in inertial space (rad/sec^2)

VI OrbitOverPositionJ2000

This function computes a user defined orbit that passes over a user specified latitude, and longitude at a user specified point in time. The user can specify the period (semi-major axis), eccentricity, true anomaly, and inclination of the desired orbit. Since the position is fixed in inertial space by the latitude, longitude, and time, the argument of perigee is determined by the input true anomaly. Note, that for a circular orbit (e = 0), true anomaly will be the argument of perigee plus true anomaly. Therefore, the input true anomaly will not be distinguishable. The RAAN (Right Ascension of the Ascending Node) is determined by the input time, inclination, and longitude. Note, that the input inclination must satisfy the following constraint: lat <= inc <= PI - lat. The position vector is computed by finding the geocentric latitude corresponding to the input geodetic latitude (WGS84). The position magnitude is found using the semi-major axis (from input period), eccentricity and true anomaly. These combined with the input longitude are used to compute a position vector in ECEF (spherical to rectangular conversion) which is converted to EMEJ2000. The orbit normal vector is found by a search using a vector that is perpendicular to the position vector. +Z cross R (position vector) is rotated incrementally about R until the angle between the rotated vector and +Z is within a tolerance of the input inclination. If the orbit is polar, (inc = 90 degrees) the orbit normal vector is found by rotating the +X axis by GAST (Greenwich Apparent Sidereal Time) + longitude - PI/2. The unit position vector (unit radial), unit normal (cross-track, and the cross product of these two (In-track) are used to form a RIC to ECI conversion. The velocity vector is computed in RIC using the flight path angle (computed from SMA, e, tru, and V magnitude). Then the RIC to ECI conversion is used to convert the velocity vector in RIC to ECI.

Inputs:

Latitude - Desired geodetic latitude of the position the orbit is to pass over (rad).
Longitude - Desired longitude of the position the orbit is to pass over (rad).
Period - Desired period of the orbit (seconds).
Eccentricity - Desired eccentricity of the orbit.
Inclination - Desired inclination of the orbit (rad).
True Anomaly - Desired true anomaly of the input latitude and longitude (rad).

TimeStampToUTC - Desired time the orbit passes over the input latitude and longitude. Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

File Inputs:

A default EOP data file is included with the ATA Aerospace Toolkit and is located in c:\Program Files\ATA Aero Toolkit. The file contains EOP data from the IERS Bulletin A, dated 6/12/2008. The user has the option creating a user defined data file by copying data from the Bulletin. In such a case the user can specify the file and location in the VI.

Outputs:

Position J2000 - Position of desired orbit in EMEJ2000 (m).
Velocity J2000 - Velocity of desired orbit in EMEJ2000 (m/sec).

VI AzElRangeToEMEJ2000

This method takes antenna gimbal readings (Azimuth from +X, Elevation from the local horizon, Az rate, El rate), range, and range rate, and determines the position and velocity in EMEJ2000. The range, range rate, antenna gimbal angle readings (Az, El, AzDot, ElDot) are converted to position and velocity in the Topocentric Horizon coordinate system (South, East, Up).
This is in turn converted to position and velocity in EMEJ2000.

Inputs:

TimeStampToUTC – Time of RADAR range, range rate, and antenna gimbal resolver measurements in exact time format. Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
EOP - Earth orientation parameters for input time.
These are:
xp - x component of polar motion (arcseconds)
yp - y component of polar motion (arcseconds)
dUT1 - Correction from utc to ut1 (sec)
Azimuth Elevation and Range - Readings taken at time of sighting.
Azimuth - Azimuth from +X read from antenna gimbal resolver (rad).
Elevation - Elevation from local plane tangent to local vertical read from antenna gimbal resolver (rad).
Range - RADAR measured range to space vehicle relative to topocentric frame (m).
Azimuth Elevation and Range Rates - Readings taken at time of sighting.
Azimuth Dot - Time rate of change of azimuth read from antenna gimbal resolver (rad).
ElevationDot - Time rate of change of elevation from local plane tangent to local vertical read from antenna gimbal resolver (rad).

RangeDot - RADAR measured range rate of space vehicle relative to topocentric frame (m/sec).
Site Location:
Lat - Geodetic latitude of the topocentric frame (Ground station) (rad).
Lon - Longitude of the topocentric frame (Ground station) (rad).
Alt - Altitude of the topocentric frame (Ground station) (m).

Outputs:
Position - Position of SV in EMEJ2000 coordinates (m).
Velocity - Velocity of SV in EMEJ2000 coordinates (m/sec).

VI EMEJ2000ToAzElRange

This function computes the Azimuth (Angle from +X), elevation, range, and time derivatives of the above relative to a user specified topocentric horizontal coordinate frame (South, East, Up), from position and velocity in EMEJ2000.

Inputs:
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
Position - Position of in EMEJ2000 coordinates (m).
Velocity - Velocity of in EMEJ2000 coordinates (m/sec).
EOP - Earth orientation parameters for input time from IERS bulletin.
These are (in order)
xp - x component of polar motion (arcseconds)
yp - y component of polar motion (arcseconds)
dUT1 - Correction from utc to ut1 (sec)
Site Location:
Lat - Geodetic latitude of the topocentric frame (Ground station) (rad).
Lon - Longitude of the topocentric frame (Ground station) (rad).
Alt - Altitude of the topocentric frame (Ground station) (m).

Outputs:
topoVec - Vector containing the following in order:
Azimuth - Azimuth from +X (rad).
Elevation - Elevation from local plane tangent to local vertical (rad).
Range - Range to space vehicle relative to topocentric frame (m).
AzimuthDot - Time rate of change of azimuth from north (rad).
ElevationDot - Time rate of change of elevation from local plane tangent to local vertical (rad).
RangeDot - Range rate of space vehicle relative to topocentric frame (m/sec).

Earth Analysis Component

This component of the ATA Aerospace Toolkit allows the user to perform various tasks related to Earth orientation, such as converting between geodetic coordinates

(geodetic latitude, longitude, and altitude) and Earth Centered Earth Fixed (International Terrestrial Reference System) coordinates. Beginning with version 2.0 the user has the added ability to read, interpolate and extrapolate EOP parameters. The user also gains the ability to compute a local vertical vector, often useful for navigation near a planet's surface. Additionally, the Toolkit allows the user to compute the obliquity of the ecliptic for a given epoch along with various other useful quantities and conversions.

VI GeodeticLatToGeocentricLat

Converts geodetic latitude to geocentric latitude. Geodetic (aka geographic) latitude is the angle between the equatorial plane and a line that is normal to the reference spheroid. Geocentric latitude is the angle between the equatorial plane and a line from the center of the Earth.

Inputs:
gdl – geodetic latitude (rad)

Ouputs:
gcl – geocentric latitude (rad)

VI MeanObliquityAngle

Computes the mean obliquity of the ecliptic of date. The obliquity of the ecliptic is the angle between the mean equator and the mean ecliptic plane (The plane of the earth's orbit around the sun).

Inputs:

utc – Date of obliquity (UTC) [Exact_time] (see def. page 15)

Outputs:

epsilon – The angle of the mean obliquity [rad]

VI ComputePrecessionMatrix

Computes the precession matrix which is used to convert vectors between J2000 MOE (Mean of Epoch) coordinate frame and Earth Centered Earth Fixed coordinate frame. This algorithm employs 1976 IAU precession theory and converts UTC to terrestrial time.

Inputs:

utc – Desired day UTC of transformation [Exact_time] (see def. page 15)

Outputs:

P – Precession matrix (3 X 3)

VI ComputePolarMotionMatrix

This VI computes the matrix that accounts for polar motion in the conversion from ECEF to ECI J2000 MOE. The polar motion earth orientation parameters are quadratically interpolated from IERS bulletin B.

Inputs:

xp – Earth orientation parameter [arc second]
yp – Earth orientation parameter [arc second]

Outputs:

PM – Rotation matrix to compensate for polar motion (3 X 3)

VI ComputeLocalVertical

This function computes the local vertical vector in inertial space. The local vertical vector is the vector perpendicular to the earth surface at some desired latitude. This is the simple ECI frame described in the Coordinate Frame section, not ECIJ2000.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
Position - Vehicle position in inertial space, i.e. simple ECI frame [m]

Outputs:

Local Vertical - Unit vector pointing up along local vertical

VI LocalVerticalECEF

This function computes the local vertical vector in an earth centered earth fixed reference frame. The local vertical vector is the vector perpendicular to the earth surface at some desired point.

Inputs:

LLH [Geodetic Cluster]
lat – Geodetic latitude of the position ($0° \leq lat \leq 90°$) [deg]
lon – Longitude of the position ($0° \leq lon \leq 360°$)* [deg]
alt - altitude above reference ellipsoid [m]

Outputs:

Local Vertical Unit Vector - Unit vector pointing up along local vertical

VI FastNutation

This function is a fast approximation to nutation in obliquity and nutation in longitude. The angles are accurate to within 11 arcseconds. This function also produces the nutation matrix needed to convert from ECEF to ECI J2000.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

delta Psi - Nutation in longitude (angle needed to compute Greenwich Apparent Sidereal Time from GMST) (rad)

delta Epsilon - Nutation of the obliquity (needed to compute the true obliquity) (rad)

Nutation Matrix - matrix used to convert from ECEF to ECI J2000 MOE.

VI ComputeNutationMatrix

This VI computes the nutation matrix used to convert from an ECEF frame to an ECI J2000 MOE frame. IAU 1980 theory is used along with the corrections to the nutation angles, if desired.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

improve – improvements to nutation angles {IMPROVE, DON'T IMPROVE}

Outputs:

deltaPsi – Nutation angle needed to compute Greenwich Apparent Sidereal Time from GMST (rad)

deltaEps - Nutation of the obliquity (needed to compute the true obliquity) (rad)

N – Nutation matrix used to convert from ECEF to ECI J2000 MOE.

VI TrueObliquityAngle

Computes the true obliquity of date. This is the angle between the true ecliptic and the true equator of date.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

epsilon – True obliquity angle (rad)

VI GAcceleration

Puts sensed acceleration in terms of a multiple of the acceleration at mean sea level.

Inputs:

Sensed acceleration

Outputs:

Sensed acceleration in terms of multiple of acceleration due to gravity at sea level.

VI ComputeLocalVertical

This function computes the local vertical vector in inertial space. The local vertical vector is the vector perpendicular to the earth surface at some desired latitude. This is the simple ECI frame.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Position - Vehicle position in inertial space, i.e. simple ECI frame [m]

Outputs:

Local Vertical - Unit vector pointing up along local vertical

VI LocalVerticalECEF

This function computes the local vertical vector in an earth centered earth fixed reference frame. The local vertical vector is the vector perpendicular to the earth surface at some desired point.

Inputs:

LLH [Geodetic Cluster]

lat – Geodetic latitude of the position ($0° \leq lat \leq 90°$) [deg]

lon – Longitude of the position ($0° \leq lon \leq 360°$) [deg]

alt - altitude above reference ellipsoid [m]

Outputs:

Local Vertical Unit Vector - Unit vector pointing up along local vertical

VI FastNutation

This function is a fast approximation to nutation in obliquity and nutation in longitude. The angles are accurate to within 11 arcseconds. This function also produces the nutation matrix needed to convert from ECEF to ECI J2000.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

delta Psi - Nutation in longitude (angle needed to compute Greenwich Apparent Sidereal Time from GMST) (rad)

delta Epsilon - Nutation of the obliquity (needed to compute the true obliquity) (rad)

Nutation Matrix - matrix used to convert from ECEF to ECI J2000 MOE.

VI ReadEOP

This function creates a table in static memory and reads into the table Earth Orientation Parameters from a file containing data from the IERS Bulletin A. Note, a default file EOPData.dat has been included with the ATA Aerospace Toolkit and is located in the LabVIEW vi.lib directory, in addons\aero_toolkit. The data in the default file was published on 6/12/2008. It is up to the user to determine if this data is sufficient.

Alternatively, the user may supply their own EOP data file by specifying the file path in this VI. The expected format of the file is such that a user can cut and paste data from the IER Bulletin A into a text file. The EOP data file must be of specific format and the format is as follows:

- Only one record per line ending in a carriage return
- The data elements of each record are separated by white space (tabs, space character)
- No spaces between records
- Maximum 120 characters per record
- The first line should contain the number of data points (records) to be read
- The second line should contain the ending date as year, month, day to signal the end of the data
- Beginning at the third line and continuing through end of the file are the EOP records as contained in IERS Bulletin A. The line format has six columns, separated by a white space character, Year, month, day, XP, YP dUT1

Inputs:
File Path - location of the user input EOP data file (if not supplied the default file is used).

Outputs:
Lines Read - the number of lines read from the EOP data file.

VI InterpolateEOP

This function is used to linearly interpolate Earth orientation parameters from the IERS bulletin A. The companion function ReadEOP included in the ATA Aerospace Toolkit must be called prior to using Interpolate EOP. ReadEOP will create a static table in memory to be read by Interpolate EOP.

If data is needed that is beyond the time span provided by the user's file (or the default EOP file included with the ATA Aerospace Toolkit), the method of extrapolation set forth in the IERS Bulletin A is used.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

EOP - Three dimentional array containing

XP, YP, UTC-UT1 at the input UTC.

VI ExtrapolateEOP

This function extrapolates the EOP data (x component of polar motion, y component of polar motion, Correction from utc to ut1) beyond the table from the IERS bulletin A using the method set forth in IERS Bulletin A.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

EOPDataEx - Three dimensional vector containing extrapolated EOP data.

Mathematical Analysis Component

The mathematical analysis component provides the user with various tools for conversion between mathematical representations of coordinate frame transformations such as quaternion to direction cosine matrices, Euler angles to direction cosine matrices, and direction cosine matrices to axis of rotation and angle about that axis. The math analysis components provide a suite of Runge Kutta integrators that integrate a coupled set of first order ordinary differential equations (Systems of the form: $\dot{\mathbf{X}} = f(t, \mathbf{X})$ where **X** is the n dimensional state vector of the system). The suite of integrators includes both fixed step and adaptive step size integrators. A VI for decomposing square matrices into eigenvectors and eigenvalues is also provided.

VI QNormalize

Normalizes a quaternion to the 4-dimensional unit sphere.

Inputs:

q – Quaternion to be normalized

Outputs:

qNormal – Normalized quaternion

VI QToDCM

Converts a quaternion to a direction cosine matrix. The DCM is also normalized in the process.

Inputs:

q – Quaternion.

Outputs:

dcm – Direction cosine matrix

VI DCMToQuaternion

This VI converts a direction cosine matrix to a quaternion. There are four representations of the quaternion. The first representation with no numerical problems is returned. All quaternions are in canonical form (The scalar portion is positive).

Inputs:

dcm – 3x3 direction cosine matrix.

Outputs:

q – The quaternion representation of the direction cosine matrix.

VI QProduct

This VI performs quaternion multiplication. Order of the quaternions is important; multiplication is NOT commutative.

Inputs:

b – First quaternion in the multiplication
c – Second quaternion in the multiplication

Outputs:

retQ – Quaternion as the result of bXc

VI XAxisQRotation

Performs a basic x axis frame rotation using a quaternion.

Input:

angle – angle [rad]
inVec – three dimensional vector.

Output:

retV – Three dimensional vector expressed in new rotated frame.

VI XAxisRotation

Computes an x axis rotation matrix for some angular frame rotation about the x axis.

Inputs:
angle – angle of rotation [rad]

Outputs:
retMat – the desired rotation matrix.

VI YAxisQRotation

This VI performs a basic y axis frame rotation using a quaternion.

Input:
angle – angle in [rad]
inVec – three dimensional vector

Output:
retV – Three dimensional vector expressed in rotated frame.

VI YAxisRotation

Computes a y rotation matrix for some angular frame rotation.

Inputs:
angle – angle of rotation [rad]

Outputs:
retMat – the desired rotation matrix.

VI ZAxisQRotation

Performs a basic z axis frame rotation using a quaternion.

Input:
angle – angle in [rad]
inVec – three dimensional vector

Output:
retV – Three dimensional vector expressed in rotated frame.

VI ZAxisRotation

This VI computes a z rotation matrix for some angular frame rotation.

Inputs:
angle – angle of rotation in [rad]

Outputs:

retMat – the desired rotation matrix

VI eToDCM

This VI takes an eigen axis (axis of rotation) and an angle of rotation and computes the corresponding direction cosine matrix

Inputs:

theta – Angle of rotation (rad)
e – Unit vector representing the axis of rotation

Outputs:

dcm – Direction cosine matrix

VI DCMToEigenAxisAngle

This VI takes a DCM and finds the axis of rotation and the angle of rotation about that axis that represents the frame rotation.

Inputs:

dcm – Direction cosine matrix

Outputs:

e – Unit vector representing the axis of rotation
theta – Angle of rotation (rad)

VI ConjugateQuaternion

Conjugates a quaternion. Conjugation of a quaternion involves negating the vector portion of the quaternion.

Inputs:

q – Quaternion to conjugate

Outputs:

qStar – Conjugate of the input quaternion

VI DCMToEuler

This VI converts a DCM to Euler angles for a given Euler sequence.
Note: This VI also checks for singularities. For a type I sequence, singularities occur at an interior angle of $\pi/2$. For a type II sequence, singularities occur at interior angles of π and 0.
Type I sequences - 123, 132, 213, 231, 312, 321
Type II sequences - 131, 121, 212, 232, 313, 323

Inputs:

seq – Desired Euler angle sequence [Euler Sequence] (see def. page 19)
(One of the 12 possible three rotation sequences)
dcm – Direction cosine matrix to convert

Outputs:
euler1 – First set of Euler angles (phi1, theta1, psi1)
euler2 – Second set of Euler angles (phi2, theta2, psi2)

VI EulerToDCM

Converts Euler angles to a DCM. (Note: For a type I sequence, the interior angle cannot be close to $\pi/2$. For a type II sequence, the interior angle cannot be close to 0 or π.) See also VI DCMtoEuler.

Inputs:
seq – Desired Euler angle sequence [Euler Sequence] (see def. page 19)
theta – Three dimensional vector containing the rotation angles

Outputs:
dcmR – Direction cosine matrix from Euler angles

VI eToQuaternion

Converts an eigen axis of rotation and angle about that axis to a quaternion.

Inputs:
eAxis – Eigen Axis of rotation.
theta – Rotation about the eigen axis [rad]

Outputs:
q – Quaternion representing the rotation normalized in canonical form (scalar portion positive).

VI CheckOrthogonality

This VI checks the orthonormality of an NxN matrix. A matrix is orthonormal if the following two conditions hold:

1. The dot product of any two distinct row/columns is zero
2. The dot product of a row/column with itself is 1

Note that a tolerance of 1.0e-7 is applied in matrix element comparisons.

Inputs:
dcm – Direction cosine matrix to check orthoganality of

Outputs:
flag – true if orthoganal [bool]

VI ExtractEigenAxisAngle

This VI extracts the rotation angle and the eigen axis of rotation from a quaternion. The eigen axis of rotation is the axis of rotation in the transformation between two reference frames. (See Euler's theorem).

Inputs:

q – Unit quaternion

Outputs:

eAxis – Vector representing the eigen axis of rotation
theta – Angle of rotation about the axis [rad]

VI EulerToQuaternion

Converts Euler angles to a quaternion. The returned quaternion is in canonical form (scalar part is positive). Note: For a type I sequence (No axes repeated), the interior angle cannot be close to $\pi/2$. For a type II sequence (exterior axes the same) the interior angle cannot be close to 0 or π.

Inputs:

seq – Desired Euler angle sequence [Euler Sequence] (see def. page 19)
theta – 3 angles of rotation about the corresponding axes in seq [Euler Angles]

Outputs:

qR – Quaternion representing the input Euler sequence

VI Jacobi3x3

This VI is Jacobi diagonalization optimized for a 3X3 matrix. This is used for real symmetric matrices. The cyclic method is implemented here, where the rows of the upper triangular of A are searched in strict order, and if the element exceeds a threshold, it is zeroed out. Some defined maximum number of sweeps is allowed. Eigenvalues are in the D matrix, and Eigenvectors are in the V matrix upon convergence.

Inputs:

A – 3X3 square symmetric real matrix to diagonalize

Outputs:

D – 3X3 diagonal matrix containing eigenvalues of A
V – 3X3 matrix containing eigenvectors of A

VI jacobi6x6

This VI is Jocobi diagonalization optimized for a 6X6 matrix. This is used for real symmetric matrices. The cyclic method is implemented here, where the rows of the upper triangular of A are searched in strict order, and if the element exceeds a threshold, it is zeroed out. Some defined maximum number of sweeps is allowed. Eigenvalues are in the D matrix, and Eigenvectors are in the V matrix upon convergence.

Inputs:

A – 6X6 square symmetric real matrix to diagonalize

Outputs:

D – 6X6 diagonal matrix containing eigenvalues of A
V – 6X6 matrix containing eigenvectors of A

VI mathAnalysisLinearInterp

Simple linear interpolation utility.

Inputs:

a – Lower x value of total interval
b – Upper x value of total interval
l – Lower y value of total interval
u – Upper y value of total interval
t – x value to interpolate at

Outputs:

c – VI approximation at t

VI mathAnalysisiInverseLinearInterp

Simple inverse linear interpolation utility.

Inputs:

a – Lower x value of total interval
b – Upper x value of total interval
l – Lower y value of total interval
u – Upper y value of total interval
ft – y value to interpolate at

Outputs:

c – x-value approximation at ft

VI IntegrateEqOfMotion

This VI is the numerical integration scheme for the equations of motion. The equations of motion are de-coupled. Translational and angular acceleration are assumed constant over

the interval. Both the translational and angular acceleration integration account for two terms of the Taylor series.

Inputs:

dt – Integration step size [sec]
r0 – Initial position vector in ECI [m]
v0 – Initial velocity vector in ECI [m/sec]
q0 – Initial quaternion representing ECI to body transformation.
w0 – Initial angular rate of body frame with respect to inertial frame expressed in body coordinates. [rad/sec]
alpha – Angular acceleration of body frame with respect to inertial frame expressed in body coordinates. [rad/sec^2]
aBT – Total translational acceleration in body frame [m/s^2]
aG – Acceleration due to gravity in ECI frame [m]

Outputs:

rF – Position vector in ECI after integration step [m]
vF – Velocity vector in ECI after integration step [m/sec]
qF – quaternion representing ECI to body transformation after integration time step.
wF – Angular rate vector in the body frame after integration step (rad/sec).

VI RungeKutta2

Carries out second order Runge Kutta integration. Integrates a

Inputs:

stateIn – State vector at time t.
Model Refnum – model to integrate [VI Refnum] (see page 20)
dt – Step size for integration (e.g. [sec])

Outputs:

stateOut - Vector containing state at time t + dt

VI RungeKutta4

Carries out fourth order Runge Kutta integration.

Inputs:

stateIn – State vector at time t.
Model Refnum – model to integrate [VI Refnum] (see page 20)
dt – Step size for integration (e.g. [sec])

Outputs:

stateOut – Vector containing state at time t + dt

VI RungeKutta4Adaptive

This VI is 4th order Runge Kutta numerical integration with step size doubling in order to control the error of integration. The idea being that function step size should be small when the function is changing rapidly for accuracy and large when the function is not changing rapidly for speed. An estimate of the Taylor series truncation error is obtained by estimating the state at some step size dt, then estimating the state by taking two steps at ½dt. The difference between the two states is RSSed and required to be below an input tolerance to accept the current step size. The RSS is used in a computation to determine if an increase in step size is warranted.

The integrator has several user input features. The integrator will never take a step smaller than the input minimum step size. It will constantly adjust the step to make sure it does not fall below the minimum step size. In addition, if a minimum step size is input that cannot be achieved with the input truncation error tolerance (user input "tolerance"), the integrator will ultimately perform an integration step with the minimum step size. The maximum truncation error encountered on any given integration step is reported to the user. Also, if the integrator continues without meeting tolerance a warning is returned to the user by returning true for the Boolean "warning". It is left to the user to determine if this is acceptable or if a smaller minimum step is warranted.

Inputs:

Model Refnum – model to integrate [VI Refnum] (see page 20)
a – Initial time ([sec])
b – Final time ([sec])
dtMin – Minimum allowed step size ([sec])
dtMax – Maximum allowed step size ([sec])
tol – Integration error tolerance at each step [ ]
iState – Vector containing the initial state

Outputs:

stateOut – Vector containing state at integration end point b.
minStepSize – Minimum step size used during integration ([sec])
maxStepSize – Maximum step size used during integration ([sec])
maxStepErr – Maximum relative error that occured during integration
nFuncCalls – Number of functions calls during integration.

VI RungeKutta45Adaptive

This VI is Runge-Kutte Fehlberg (4/5) numerical integration for a coupled set of linear ordinary differential equations. It employs adaptive step size control by computing a desired step size based on the ratio of the desired error tolerance to the estimated Taylor series truncation error. As a matter of practicality, the step size is not allowed to increase by more than a factor of 5 from one step to the next, in order to avoid wild oscillations of the step size.

The integrator has several user input features. The integrator will never take a step smaller than the input minimum step size. It will constantly adjust the step to make sure it does not fall below the minimum step size. In addition, if a minimum step size is input

that cannot be achieved with the input truncation error tolerance (user input “tolerance”), the integrator will ultimately perform an integration step with the minimum step size. The maximum truncation error encountered on any given integration step is reported to the user. Also, if the integrator continues without meeting tolerance a warning is returned to the user by returning true for the Boolean "warning". It is left to the user to determine if this is acceptable or if a smaller minimum step is warranted.

Inputs:

Model Refnum – model to integrate [VI Refnum] (see page 20)
a – Initial time ([sec])
b – Final time ([sec])
dtMin – Minimum allowed step size ([sec])
dtMax – Maximum allowed step size ([sec])
tol – Integration error tolerance at each step [ ]
iState – Vector containing the initial state

Outputs:

stateOut – Vector containing state at integration end point b
minStepSize – Minimum step size used during integration ([sec])
maxStepSize – Maximum step size used during integration ([sec])
maxStepErr – Maximum relative error that occured during integration
nFuncCalls – Number of functions calls during integration

VI TransformMassProperties

This VI transforms the mass properties (Center of Gravity vector and inertia tensor) from one frame into another frame (frame A to frame B)

Inputs:

cAB – 3X3 transformation matrix from frame A to frame B
rBA – 3X1 translation vector from the origin of frame B to the origin of frame A expressed in frame B [m]
cgVecA – Center of gravity vector in the A frame [m]
itMatA – Inertia tensor in the A frame [kg-m/sec^2]
M – Mass of component to transform mass properties of from frame A to B [kg]

Outputs:

cgVecB – Center of gravity vector in the B frame [m]
itMatB – Inertia tensor in the B frame [kg-m/sec^2]

VI HInterpolation

This function is 5th order hermitian interpolation. It is use to interpolate between orbit ephemeris points.

Inputs:

t - Time at which to interpolate (seconds)
t0 - Time of initial state (seconds)
t1 - Time of final state (seconds)
state1 - Initial state vector (p, v, a) (m. m/s, m/s^2)
state2 - End state vector (p, v, a) (m. m/s, m/s^2)

Outputs:

iState - Interpolated state vector at t. (p, v, a) (m. m/s, m/s^2)

VI GenECIToNEDDCM

This function computes the dcm to transform from ECI to a local North-East-Down (NED) frame. In the NED frame, the x axis points north. The z axis points down along the local vertical (See computeLocalVertical function). The y axis completes the RHS.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
pos - ECI position vector (meters)

Outputs:

nDCM - Direction cosine matrix (ECI to NED)

VI GenECIToRICDCM

This function computes the DCM to convert a vector in ECI to a vector in RIC (Radial, In-track, Cross track) coordinates. This transformation aligns the position along the Radial vector. In-Track is in the general direction of the velocity vector. Cross track is the unit angular momentum vector.

Inputs:

pos - Position vector in ECI frame
vel - Velocity vector in ECI frame

Outputs:

tDCM - Transfoirmation DCM from ECI to RIC

VI ECEFToNED

This function computes the DCM to transform from ECEF to a local North-East-Down (NED) frame. In the NED frame, the x axis points north. The z axis points down along the local vertical. The y axis completes the RHS.

Inputs:

lat - Geodetic latitude at desired location to compute ECEF to NED transformation (rad).

lon - Longitude at desired location to compute ECEF to NED transformation (rad).

Outputs:

nDCMECEF - Direction cosine matrix (ECEF to NED)

VI PosVelRecToSphe

This function converts both position and velocity vectors in rectangular cooridinates to spherical coordinates (theta, phi, rho, thetaDot, phiDot, rhoDot).

Inputs:

r - Position vector in rectangular coordinates (m)

v - Velocity vector in rectangular coordinates (m)

Outputs:

Position - Position vector in spherical coordinates (theta, phi, rho)

Velocity - Velocity vector in spherical coordinates (thetaDot, phiDot, rhoDot)

VI PosVelSpheToRec

This function is the conversion of full spherical coordinates (theta, phi, rho, thetaDot, phiDot, rhoDot) to rectangular coordinates (position and velocity).

Inputs:

Spherical - Vector containing position and velocity in spherical coordinates in the following order:

- Angle from the +X axis (theta) (rad)
- Angle from the XY plane (phi) (rad)
- Radius magnitude (rho)
- Derivative of theta with respect to time (thetaDot) (rad/sec)
- Derivative of phi with respect to time (phiDot) (rad/sec)
- Derivative of radius magnitude with respect to time

Outputs:

r - Position vector in rectangular coordinates

v - Velocity vector in rectangular coordinates

VI ECItoLVOP

This function computes the DCM to convert a vector in ECI to a vector in LVOP (Local Vertical Orbital Plane) coordinates. LVOP is defined as follows:
+X - In the general direction of the orbital velocity vector
+Y - Anti normal (Negative of unit orbital angular momentum)
+Z - NADIR (Negative of position vector)

Inputs:

Position - Position vector in ECI frame
Velocity - Velocity vector in ECI frame

Outputs:

DCM - Transformation DCM from ECI to LVOP

VI BodyToInertial

This function converts vectors (position, velocity, and acceleration) in a rotating body frame to a fixed inertial frame. Coriolis and centripetal accelerations are accounted for. Note that the conversions will not be exact if the body frame is experiencing higher order rotational derivatives (i.e. angular acceleration).

Inputs:

Position Body - Position vector in the rotating body frame (m).
Velocity Body - Velocity vector in the rotating body frame (m/sec).
Acceleration Body - Acceleration vector in the rotating body frame (m/sec^2).
Position IB - Position of body frame relative to inertial frame expressed in the inertial frame. (m)
Velocity IB - Velocity of body frame relative to inertial frame expressed in the inertial frame. (m/sec)
Acceleration IB - Acceleration of body frame relative to inertial frame expressed in the inertial frame. (m/sec^2)
Angular Rate IB - Angular rate of body frame relative to the inertial frame expressed in the body frame (rad/sec).
Quaternion IB - Quaternion representing inertial to body transformation.

Outputs:

Position Inertial - Position vector in the inertial frame (m).
Velocity Inertial - Velocity vector in the inertial frame (m/sec).
Acceleration Inertial - Acceleration vector in the inertial frame (m/sec^2).

VI InertialToBody

This function converts vectors (position, velocity, and acceleration) in an inertial non-rotating body frame to a body fixed frame. Coriolis and centripetal accelerations are accounted for. Note that the conversions will not be exact if the body frame is experiencing higher order rotational derivatives (i.e. angular acceleration).

Inputs:

Inertial Frame Grouping
Position Inertial - Position vector in the non-rotating inertial frame (m).
Velocity Inertial- Velocity vector in the non-rotating frame (m/sec).
Acceleration Inertial- Acceleration vector in the non-rotating inertial frame (m/sec^2).
Position IB - Position of body frame relative to inertial frame expressed in the inertial frame. (m)
Velocity IB - Velocity of body frame relative to inertial frame expressed in the inertial frame. (m/sec)
Acceleration IB- Acceleration of body frame relative to inertial frame expressed in the inertial frame. (m/sec^2)
Angular Rate IB - Angular rate of body frame relative to the inertial frame expressed in the body frame (rad/sec).
Transformation Quaternion
Quaternion IB - Quaternion representing inertial to body transformation.

Outputs:

Position Body - Position vector in the body fixed frame (m).
Velocity Body - Velocity vector in the body fixed frame (m/sec).
Acceleration Body - Acceleration vector in the body fixed frame (m/sec^2).

VI Jacobian

This function computes the Jacobian of a vector function of vector values.
The Jacobian is a square matrix of partial derivatives of the following form:

```
| dF1/dx1 dF1/dx2 dF1/dx3 ..... dF1/dxn |
| dF2/dx1 ..................... dF2/dxn |
|  .                           |
|  .                           |
|  .                           |
| dFn/dx1 ..................... dFn/dxn |
```

Inputs:

Vector Function - VI reference to state transition function. Although this reference is for a prototype that takes, not just a state vector, but also a time t, or other independent variable, the jacobian perturbation does not include perturbing time. The function does not need be time dependent, in which case the optional control t to this VI is not needed..
Number of Equations - Number of equations and variables to evaluate.
Input Vector - n dimensional vector to evaluate f at.
Differencing - Enumerated type indicating type of derivative to be used in computing the Jacobian matrix.
'f' - Forward differencing.
'b' - Backward differencing.
'c' - Central differencing.
dt - User defined step size for computing the finite divided differences.

Outputs:

J - Jacobian of f evaluated at inVec

VI Gradient

This function computes the gradient of an n dimensional real-valued function. The gradient is a vector of partial derivatives of the function with respect to each of the variables. [dF/dx1, dF/dx2, dF/dx3, ... , dF/dxn]. Derivatives are computed numerically using finite divided differences. The gradient is the direcitional derivative of a function at a point. The magnitude of the gradient gives the rate of change of the function. The unit gradient gives the direction of the change.

Inputs:

Scalar Function - Reference to scalar function of vector values.
vector - n dimensional vector at which to evaluate f.
differencing - type of derivative to be used in computing the gradient.
dt - Step size for computing numerical derivatives.

Outputs:

gradient - n dimensional gradient of function at desired point.

Math Utilities Component

The Math Utilities Component of the LabVIEW Aerospace Toolkit provides the user with various fundamental mathematical capabilities such as matrix multiplication, matrix inversion, matrix transposition, and computing the angle between two vectors. In many cases, the utility is very specialized, such as multiplying a 3X3 matrix by a 3 dimensional vector. This is generally not restrictive in the aerospace analysis, since much of the analysis occurs in three dimensional space. However, in most cases, general counterparts to these specialized functions are provided. Many of these function exist in the LabVIEW base and math analysis package; ATA extends their functionality with unit polymorphism to encourage strict use of physical units. These VIs that are extensions of existing LabVIEW VIs but with unit polymorphism are prepended with the prefix poly.

VI PolyMatrixXVector

Post multiplies a 3X3 matrix by a 3 dimensional vector.

Inputs:

inMat – 3X3 matrix [$1]
inVec – 3 dimensional vector [$2]

Outputs:

outVec – 3 dimensional vector result of multiplication [$1 $2]

VI PolyMatrixXMatrix

Multiply a N X N matrix by another N X N matrix.

Inputs:
inMat1 – First N X N matrix [$1]
inMat2 – Second N X N matrix [$2]

Outputs:
outMat – Product of input matrix [$1 $2]

VI PolyCrossProduct

Perform the cross product between two three dimensional vectors.

Note there are some anomalies with LabVIEW strict physical units and cross product, when one of the units is radian. According to SI a radian is dimensionless, but LabVIEW strict unit checking does not consider radian equivalent to dimensionless. (e.g. v = w X r is common expression for rotational velocity where w is [rad/sec] and r is perhaps [m]; strict unit checking would require v to be [m-rad/sec], but it is actually [m/s]) Therefore it is sometimes necessary to multiply the output of a polymorphic cross product by a constant of 1 rad^{-1} to preserve proper and meaningful physical units.

Inputs:
inVec1 – Three dimensional vector [$1]
inVec2 – Three dimensional vector [$2]

Outputs:
outVec – inVec1 X inVec2 [$1 $2]

VI PolyMagnitudeVector

Compute the magnitude of a vector.

Inputs:
inVec – N dimensional vector [$1]

Outputs:
magnitude – Magnitude of the vector [$1]

VI ECPolyDotProduct

Calculates the dot product of two vectors. Generates an error if they are not of the same length.

Inputs:
A – N dimensional vector [$1]
B – N dimensional vector [$2]

Outputs:

A dot B – dot product [$1 $2]

VI AngleBetweenVectors

Compute the smallest angle between two vectors by taking the arc cosine of the dot product.

Inputs:

inVec1 – 3 dimensional vector [$1]
inVec2 – 3 dimensional vector [$1]

Outputs:

theta – Smallest angle between the two vectors [rad]

VI TrigZero

Finds a solution to the following equation:
A*SIN(X) + B*COS(X) + C = 0

Inputs:

A – First coefficient
B – Second coefficient
C – Third coefficient

Outputs:

X1 – First possible solution
X2 – Second possible solution

VI Sign2

Multiplies the sign of the second argument (+1 or -1) times the first argument (0 is considered negative.)

Inputs:

arg1 – First argument
arg2 – Second argument

Outputs:

Val – first argument multiplied by sign of the second argument

VI ModuloAngle

Determines the equivalent angle between 0 and 2π

Input:

ang – Angle to put between 0 and 2π [rad]

Output:

angOut – Angle between 0 and 2π [rad]

VI PolyUnitizeVector

Unitizes an N dimensional vector.

The input can take any physical units, but the unit vector is dimensionless.

There are two special cases handled by this VI. The first is a zero vector. The zero vector has N dimensions, but a magnitude of zero. The user is provided with a Boolean control to either allow, or not, the input of a zero vector. If the control value is true then the VI will accept a vector of zero magnitude and return a vector of zero magnitude and N dimension. Else, if the control is false the VI will return an error.

The second special case is a null vector, a vector of zero dimension(N dimensional array where N=0). If a null vector is passed as an input the VI will return an error.

Inputs:

inVec – Vector to unitize [$1]
allowNull – (optional default is FALSE) if true return zero magnitude vector if inVec has zero magntiude; otherwise an error is generated [bool]

Outputs:

uVec – unitized vector [ ]
Magnitude – Magnitude of input vector.

VI SphereToRectangle

Conversion from spherical to rectangular coordinates.

Inputs:

ra – Right ascension of the vector (relative to the x axis) [rad] (0 - 2π)
dec – Declination of the vector (relative to the x-y plane) [rad] (-π/2 - π/2)
rMag – Magnitude of the vector

Outputs:

rVec – Vector in rectangular coordinates relative to reference frame

VI SphereToRectangle

This function computes the quadrant in a coordinate system in which the angle lies.

Inputs:

Angle in radians

Outputs:

Quadrant of the angle

3DCrossProduct

This function computes the cross product of two three dimensional vectors.

Inputs:

A - First three dimensional vector
B - Second three dimensional vector

Outputs:

Cross Product

3DDotProduct

This function computes the dot product result of two three dimensional vectors.

Inputs:

A - First three dimensional vector
B - Second three dimensional vector

Outputs:

A dot B

3DMagnitude

This function computes the magnitude of a three dimensional input vector.

Inputs:

Three dimensional vector

Outputs:

Magnitude

3DMatrixXVector

This function computes the resultant vector of two three dimensional vectors.

Inputs:

Direction Cosine Matrix

Outputs:

Three dimensional vector.

3DUnitizeVector

This function unitizes a three dimensional vector.

The input can take any physical units, but the unit vector is dimensionless.

There are two special cases handled by this VI. The first is a zero vector. The zero vector has three dimensions, but a magnitude of zero. The user is provided with a Boolean control to either allow, or not, the input of a zero vector.

The second special case is a null vector. If a null vector is passed as an input the VI will return an error.

Inputs:

Allow Zero – If the control value is true then the VI will accept a vector of zero magnitude and return a vector of zero magnitude. Else, if the control is false the VI will return an error. Default value is false.

3D Vector – Vector to be unitized

Outputs:

Unit Vector – Unitized vector

Magnitude – Magnitude of input vector

AngBetweenVectors

This function computes the smallest angle between two vectors.

Inputs:

A – First three dimensional input vector

B – Second three dimensional input vector

Outputs:

Theta – smallest angle between the two input vectors (rad).

rXF

This VI calculates torque by crossing the position vector with the force vector. It simplifies the common operation of computing torque and is optimized to work with other Aerospace Toolkit functions.

Inputs:

r – position vector (m)
F – Force vector (N)

Outputs:

Torque – Resulting torque (Nm)

omegaXr

This VI computes the instantaneous velocity vector of an object that has a rotational rate vector defined.

Inputs:

Omega - Angular rate (rad/s)
r - Position Vector (m)

Outputs

Instantaneous Velocity Vector (m/s)

Torque

This VI computes the torque of a body of revolution.

Inputs:

itA – Interia tensor of the body (Kg * m^2)
Alpha - Angular accelerations exerted on the body. (rad/sec^2)
Omega - Angular rate in the body frame (rad/sec)

Outputs:

Torque - Torque exected on the body (Nm)
h body - Angular Momentum (Kg*m^2/sec)

TauIAlpha

This VI computes torque by multiplying the inertia tensor (I) by the angular acceleration (alpha). This VI is optimized to work with the Aerospace Toolkit by supporting the use of physical units.

Inputs:

itA - Inertia Tensor (Kg * m^2)
Alpha - Angular acceleration (rad/sec^2)

Outputs

Torque - torque exerted on the body. (Nm)

CompareFloatingPoint

Compares inputs x and y to determine equality.

Inputs:

x - floating point value

y - floating point value

digits - the number of digits of precision to which numbers are compared for equality.

Outputs:

Returns 0 if x ~= y
+1 if x > y
-1 if x < y

Usage:
This VI is specifically designed where LabVIEW's subtraction, absolute value and sign math primitives might previously have been used. However there are known difficulties with determining the equality of numbers represented as floating points when their values are calculated by different computational paths. This allows to user to accommodate those issues and also just allow a more lax definition of equality.

Notes on algorithm:
~= or "approximately equal" is parameterized by an optional input digits which defaults to 6.
First a decimal order of magnitude of the larger input o(max(x,y)) is determined by taking the integral part of log10 of the larger input.
The tolerance for the difference is always a power of 10, specifically
tol = 10^(o - digits + 1)
If the absolute value of the difference of the inputs is less than the tolerance the values are considered approximately equal and 0 is returned.

Orbit Propagation Component

The Orbit Propagation Component gives the user the ability to propagate the orbit of a spacecraft using various force models. The user may wish to propagate an orbit for use in a real-time simulation, in which case, a simple spherical gravity model might be the force model of choice for speed considerations. The user can also select a J2 gravity model, and a J6 gravity model. An exponential atmospheric drag can also be selected. Runge Kutte Fehlberg 4/5 with adaptive step size control is used to integrate the equations of motion.

VI HFPropagator

This function is a high fidelity propagator. This propagator uses EGM96 gravity with spherical harmonics to degree and order 12. This function accounts for solar radiation pressure including partial eclipse of the vehicle by the Earth. The function also accounts for gravitational effects of the Sun and Moon.
Inputs:

Force Model Parameters - Settings to configure the force model used for propagation.
Gravitational Field Spherical Harmonics - Gravitational Field Spherical Harmonics -
This value may be between 0 and 12. An input of 0 represents simple two body gravity.
Solar Gravity - True = solar gravity, False = no solar gravity
Lunar Gravity - True = lunar gravity, False = no lunar gravity
Solar Radiation Pressure - True = solar radiation pressure, False = no solar radiation pressure
Atmospheric Drag - No drag = no atmospheric drag effects, Exponential = an exponential atmosphere drag model.
Inertial Frame Transformation - 1) Hi Fidelity = use hi fidelity EMEJ2000 to ECEF accounting for precession, nutation, polar motion, Earth rotation; 2) Simple Rotation = use a simple rotation about +z inertial axis by GMST (Greenwich Mean Sidereal time)

Aerospace Toolkit Time Control - initial time of the state vector.
Initial Position - Initial position of the vehicle (m)
Initial Velocity – Initial velocity of the vehicle (m/s)
Duration - Time span over which to propagate the orbit (seconds)
dtMin - Minimum allowed step size during integration (seconds)
dtMax - Maximum allowed step size during integration (seconds)
Tolerance - Truncation error tolerance during integration

Vehicle Parameters
Drag coefficient - vehicles aerodynamic drag coefficient
Mass - vehicle mass (kg)
Area - vehicle cross sectional area (m^2)
Solar Pressure Coefficient - Solar radiation pressure coefficient
Solar Exposed Area - Cross sectional area of SV facing the sun (m^2)

Outputs:
Final position - Final position in Cartesian coordinates (m)
Final velocity – Final velocity in Cartesian coordinates (m/s)
Final acceleration – Final acceleration in Cartesian coordinates (m/s^2)
minStepSize - Minimum step size used during integration (seconds)
maxStepSize - Maximum step size used during integration (seconds)
maxStepErr - Maximum relative error allowed during integration
Function Calls - Number of force model function calls made during integration.

VI UpdateState

The Orbit Propagation Component gives the user the ability to propagate the orbit of a spacecraft using various force models. The user may wish to propagate an orbit for use in a real-time simulation, in which case, a simple spherical gravity model might be the force model of choice for speed considerations. The user can also select a J2 gravity model, and a J6 gravity model. An exponential atmospheric drag can also be selected. Runge Kutte Fehlberg 4/5 with adaptive step size control is used to integrate the equations of motion.

This VI is the state update function for the orbit propagator. This VI is the executive that calls the desired force model and performs error handling. This propagator is an on-the-fly propagator, but can be put in a loop to generate ephemeris.

The integrator has several user input features. The integrator will never take a step smaller than the input minimum step size. It will constantly adjust the step to make sure it does not fall below the minimum step size. In addition, if a minimum step size is input that cannot be achieved with the input truncation error tolerance (user input "tolerance"), the integrator will ultimately perform an integration step with the minimum step size. The maximum truncation error encountered on any given integration step is reported to the user. Also, if the integrator continues without meeting tolerance a warning is returned to the user by returning true for the Boolean "warning". It is left to the user to determine if this is acceptable or if a smaller minimum step is warranted.

* Note - If modeling drag, you must first call AeroDragParameters.vi to specify aerodynamic drag parameters.

Inputs:

configVec - Vector to specify the force model selection.
Gravitational force model { SPHERICAL, J2, VINTIJ6 }
Drag Model { NO DRAG, DRAG }
TimeStampToUTC – Initial time for state vector. Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
init pos – Initial position [m]
init vel – Initial velocity [m/sec)]
duration - Time span over which to propagate the orbit [sec]
dtMin - Minimum allowed step size during integration [sec]
dtMax - Maxmum allowed step size during integration [sec]
tol - Truncation error tolerance [ ]

Outputs:

final pos – Final orbit position [m]
final vel – final orbit velocity [m]
final acc – final orbital acceleration [m/s^2]
minStepSize – Minimum step size used during integration [sec]
maxStepSize – Maximum step size used during integration [sec]
maxStepErr – Maximum relative error that occured during integration
nFuncCalls – Number of force model function calls made during integration [int]

VI AeroDragParameters

This function is needed to specify the aerodynamic parameters for force models that model drag in UpdateState.vi. This function must be called prior to the first call to UpdateState.vi.

Inputs:

CD – Drag coefficient [ ]

mass – SV mass [kg]
S – SV Cross sectional area [m^2]

Outputs:
N – Number of times these parameters have been specified.

VI SGPSemi-AnalyticProp

This function is the SGP semi-analytic orbit propagator. It uses the 2-line mean element sets. SGP stands for Simplified General Perturbations. SGP was developed by Hilton & Kuhlman (1966) and is used for near-Earth satellites (period < 225 minutes). This model uses a simplification of the work of Kozai (1959) for its gravitational model and it takes the drag effect on mean motion as linear in time. This assumption dictates a quadratic variation of mean anomaly with time. The drag effect on eccentricity is modeled in such a way that perigee height remains constant. The coordinate system of operation is True Equator and Mean Equinox (TEME) of date.

Inputs:
Initial Epoch - Epoch of initial state (minutes).
Mean Motion - Initial mean motion (rad/min).
Mean Eccentricity - Initial mean eccentricity (n/a).
Mean Inclination - Initial mean inclination (rad).
Mean Anomoly - Initial mean mean anomaly (rad).
Mean Argument of Perigee - Initial mean argument of perigee (rad).
Mean Right Ascension of the Ascending Node - Initial mean right ascension of the ascending node (rad).
First Derivative - Derivative of mean motion with respect to time (rad/min^2).
Second Derivative - Second derivative of mean motion with respect to time (rad/min^3).
Time - Time of requested state (minutes)

Outputs:
Position - Position of at time t in TEME (m).
Velocity – Velocity at time t in TEME (m/sec)

VI MeanMotionProp

This function is a lower fidelity orbit propagator. It simply converts the initial true anomaly to the mean anomaly and then uses the mean motion to propagate the mean anomaly forward in time (m = m0 + n*dt). The mean anomaly is then converted back to true anomaly. The orientation of the position vector is computed using the 313 classic aerospace rotation sequences (Vector rotation). The first rotation is by the right ascension of the ascending node. The second rotation is by the inclination. The third rotation is by the argument of latitude (true anomaly + argument of perigee). Position magnitude is computed according to an elliptical orbit relationship according to the true anomaly. Velocity magnitude is computed using another elliptical orbit relationship. The orientation of the velocity vector is computed by an additional rotation (Vector rotation) by PI/2 - flight path angle to the orientation of the position vector.

Inputs:

rInit - Initial position in a geocentric reference frame (m).
vInit - Initial velocity in a geocentric reference frame (m/sec).
dt - Orbit propagation time (seconds)

Outputs:

r - Orbit position vector in the geocentric reference frame dt seconds later (m).
v - Orbit velocity vector in the geocentric reference frame dt seconds later (m/sec).

Orbit Adjust Component

The Orbit Adjust component gives the user the ability to guide the trajectory of a spacecraft or rocket. It provides a standard guidance package known as Lambert guidance (Refer to the Math Analysis Component for more information on the solution to Lambert's problem). It also provides another guidance scheme for guiding a rocket into an elliptical orbit. Some propulsion functionality is also provided which utilizes the ideal rocket law, which will aid the user in exercises such as engine sizing and burn time computations.

VI BurnTime

Computes engine burn time for a desired delta velocity for given set of engine characteristics using the ideal rocket law.

Inputs:

m0 – Initial mass [kg]
DeltaV – Delta velocity for which burn time is required [m/sec]
ISP – Engine specific impulse [sec]
thrust – Engine thrust [N]

Outputs:

outVec – Vector containing the following in order:
mDot – Mass flow rate [kg/sec]
tBurn – Burn time for required delta V [sec]
mf – Final mass after burn [kg]
deltaMass – Change in mass over the burn [kg]

VI DeltaV

Computes the delta-V imparted for a given engine burn time for given set of engine characteristics using the ideal rocket law. (Note: engine specific impulse is seconds which assumes gravity at the surface of the Earth of 9.81 m/s^2 or 32.185 ft/sec^2)

Inputs:

m0 – Initial mass [kg]
tBurn – Burn time [sec]
ISP – Engine specific impulse [sec]
thrust – Engine thrust [N]

Outputs:

outVec – Vector containing the following in order:
mDot – Mass flow rate [kg/sec]
deltaV – Delta velocity [m/sec]
mf – Final mass [kg]
deltaMass – Change in mass [kg]

VI CrossProductSteering

Implements cross product steering. It produces the unit direction to point the thrust vector. It drives the components of the velocity-to-be-gained (Vg) vector to zero simultaneously by achieving dVg/dt X Vg = 0. Velocity-to-be-gained is the difference between velocity required to meet specific mission objectives and current vehicle velocity. (See *An Introduction to the Mathematical Methods of Astrodynamics* by Battin)

Inputs:

r – Current vehicle position in inertial space [m]
v – Current vehicle velocity in inertial space [m/sec]
aTI – Total vehicle sensed acceleration in inertial space (Aero and thrust) [m/s^2]
vR – Required velocity to meet mission objectives [m/sec]
steerDt – Steering step size [sec]

Outputs:

uAt – 3 dimensional vector representing the desired direction of the thrust vector in inertial space

VI LambertGuidance

This VI is Lambert guidance. It starts by solving Lambert's problem. The solution to Lambert's problem finds the velocities at two points on some orbit given a transfer time between the two positions. This VI is passed the current state of the vehicle (position and velocity), then solves Lambert's problem to get the required velocity to intercept an input desired position in an input total time of flight. Cross product steering is then called to get an optimal direction to point the acceleration due to thrust in light of motion in a conservative force field (i.e. gravity).

Inputs:

r – Vehicle current position in inertial space [m]
v – Vehicle current velocity in inertial space [m/s]
aTI – Total sensed acceleration in inertial space [m/s^2]
target – Targeted position in inertial space [m]
tTOF – Total time of flight [sec]

gStep – Guidance step size (guidance cycle) [sec]

Outputs:

vRequired – Velocity required to achieve target from Lambert [m/sec]
uAtD – Unit vector in inertial space. The direction to point the thrust vector.

VI EllipticalGuidance

This VI is the required velocity to inject into an elliptical orbit with some specified semi-latus rectum and eccentricity.
Note: This will not work for eccentricity equal to 0, but it will work for very small eccentricities.

Inputs:

p – Semi-latus rectum of target orbit [m]
e – Eccentricity of target orbit (0 < e < 1)* [ ]
r – Vehicle ECI position vector [m]
uH – Unit vector perpendicular to desired plane of the orbit. This is the unit angular momentum vector. [ ]

Outputs:

vR – Required velocity to inject into target orbit. [m/s]

VI LanderGuidance

This function is used to compute desired (commanded) vehicle accelerations for a lander vehicle to land on the Earth, the Moon or Mars. This function is designed to be called repeatedly within a simulation. The time-to-go calculation for the lander guidance algorithm consists of two functions, a time-to-go function which computes the time to land by numerical solution of the Euler-Lagrange equations. The function then computes the desired acceleration with a calculus of variations solution to a two-point boundary value problem used to land a vehicle on a planetary surface. It is assumed that h/R << 1 (Altitude is small compared to the planet equatorial radius) , and that the acceleration due to aerodynamics is insignificant with respect to gravity and thrust.

The coordinate frame that the algorithm operates in is defined as follows:
Centered at arbitrary planet position
X: Down range (Due east)
Y: Cross range (Due north)
Z: Up (Directed up along the local vertical)
This is sometimes referred to as DR/CR/UP

The gravitational accelerations used in this function are as follows:
1. Earth: 9.81 m/sec^2
2. Moon:1.6 m/sec^2
3. Mars: 3.72067 m/sec^2

Inputs:

-Target Position - Position target in DR/CR/UP coordinates (m) (Typically (0, 0, 0))
-Vehicle Position - Position of the vehicle in DR/CR/UP (m)
-Vehicle Velocity - Velocity of the vehicle in DR/CR/UP (m)
-Gravity vector - User has the ability to select Earth, Moon, or Mars.
-gamma - Weighting parameter (0 - 200). Values close to 0, the trajectory will be minimum acceleration. Values close to 200, trajectory will be minimum time.

Outputs:

-Command Acceleration - Commanded acceleration of vehicle in DR/CR/UP coordinates (m/s^2)

VI MinJerkGuidance

This function is minimum jerk guidance between two three dimensional points in space. It is the calculus of variations solution that yields the desired position and velocity as a function of time between a starting point, and a target that minimizes jerk along the trajectory.

Inputs:

Initial Position - Initial position of vehicle in guidance frame [m]
Target Position - Target position of vehicle in guidance frame [m]
Time Into Trajectory - Time into the trajectory between boundary points [s]
Time To Target - Time duration to reach target. [s]

Outputs:

dPosition - Desired position vectors in the guidance frame at desired time. [m]
dVelocity - Desired velocity vector in the guidance frame at the desired time. [m/s]

Aerodynamic Utilities Component

The Aerodynamic Utilities Component of the LabVIEW Aerospace Tool Kit provides the user with the ability to simulate various aerodynamic forces and torques on a vehicle traveling through the atmosphere. Two atmosphere models are provided by the Toolkit: US Standard of 1976 and the exponential model.

VI ExpAtmosphere

This VI is a three-parameter exponential atmosphere (Nominal density, base altitude, and scale height).

Inputs:

h – altitude above reference ellipsoid [m]

Outputs:

rho – atmospheric density [kg/m^3]

VI USStandard1976Atmos

This VI is the 1976 US standard atmosphere up to 71 kilometers.

Inputs:

h – Altitude above reference ellipse [m]

Outputs:

T – Temperature [K]
r – Atmospheric density [kg/m^3]
P – Atmospheric pressure [N/m^2]
warning – input altitude above 71 km; user may want to use a model with more precise characterization of the exosphere. [bool]

VI WindToECI

This VI transforms data to describe a wind in the NED (North East Down) frame into a vector in NED, and then transforms the vector into ECI (Earth Centered Inertial). In the NED frame, the +X axis points north, the +Y axis points east, and the +Z axis points down along the local vertical. The data describing the wind is azimuth from north, speed, and vertical component (updraft or downdraft) of wind velocity.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
az – Azimuth clockwise from north [rad]
speed – Wind speed [m/s]
vComp – Vertical component of wind velocity [m/sec]
r – Position in ECI [m]

Outputs:

wVelECI – Wind velocity vector in the ECI frame at desired position [m/s]

VI ComputeRelativeWind

Computes the velocity of the vehicle relative to the wind.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the

manual for more detail.
rECI – Position vector in ECI [m]
vECI – Velocity vector in ECI [m]
windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [bool]
windData – Vector containing external wind data
az – azimuth clockwise from north [rad]
speed – magnitude of horizontal component [m/s]
Vcomp – vertical component: positive up, negative down [m/s]

Outputs:
vRel – Vehicle velocity relative to the wind [m/sec]

VI SpeedOfSound

Computes the speed of sound $C = \sqrt{GAMMA * R * T}$
(see constants section on page 104 for definitions of GAMMA and R)

Inputs:
T – Atmospheric temperature [K]

Outputs:
C – Speed of sound [m/sec]

VI MachNumber

This VI is used to compute the Mach number $M = \frac{\left|\overrightarrow{v\,\mathrm{Re}\,l}\right|}{C}$.
The Mach number is defined as the ratio of vehicle speed relative to the atmosphere, and the speed of sound.

Inputs:
TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
rECI – Vehicle position in ECI [m]
vECI – Vehicle velocity in ECI [m/sec]
T - Atmospheric temperature [K]
windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [bool]
windData – Vector containing external wind data
az – azimuth clockwise from north [rad]
speed – magnitude of horizontal component [m/s]
Vcomp – vertical component: positive up, negative down [m/s]

Outputs:

M - mach number [ ]

VI ComputeDynPressure

This VI computes dynamic pressure $\bar{q} = \frac{1}{2}\rho\|\vec{v}_{rel}\|^2$.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
rECI – Position vector in ECI space [m]
vECI – Velocity vector in ECI space [m]
rho – Atmospheric density [Pa]
windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [Bool]
windData – Vector containing external wind data
az – azimuth clockwise from north [rad]
speed – magnitude of horizontal component [m/s]
Vcomp – vertical component: positive up, negative down [m/s]

Outputs:
qBar – Dynamic pressure [Pa]

VI AccelDueToDragECI

This VI computes acceleration due to drag in the inertial frame.
This is simple acceleration due to drag which is applied opposite of the relative wind velocity vector. $\overrightarrow{aDrag_{ECI}} = -\frac{1}{2}C_D\frac{S}{Mass}\rho\|\vec{v}_{rel}\|\frac{\vec{v}_{rel}}{\|\vec{v}_{rel}\|}$

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
CD – Drag coefficient
rho – Atmospheric density [kg/m^3]
mass – Mass of vehicle [kg]
rECI – Vehicle ECI position vector [m]
vECI – Vehicle ECI velocity vector [m/sec]
S – Vehicle cross sectional reference area [m^2]

Outputs:

aDragECI - Acceleration due to drag in ECI frame [m/sec^2]

VI AngOfAttack

Computes the angle of attack. Angle of attack is the angle between the vehicle velocity vector relative to the wind in the body frame projected into the pitch plane and the body roll axis.
The quadrant of the angle is also needed for the purposes of computing body accelerations and torques.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
rECI – Position vector in ECI space [m]
vECI – Velocity vector in ECI space [m]
cIB - Inertial to body transformation matrix
windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [Bool]
windData – Vector containing external wind data
az – azimuth clockwise from north [rad]
speed – magnitude of horizontal component [m/s]
Vcomp – vertical component: positive up, negative down [m/s]

Outputs:

alpha – Magnitude of angle of attack [rad]
quad – Classical trigonometric quadrant of the angle

VI AngSideSlip

Computes the angle of attack. Angle of attack is the angle between the vehicle velocity vector relative to the wind in the body frame projected into the pitch plane and the body roll axis.
The quadrant of the angle is also needed for the purposes of computing body accelerations and torques.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
rECI – Position vector in ECI space [m]
vECI – Velocity vector in ECI space [m]
cIB - Inertial to body transformation matrix
windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [Bool]
windData – Vector containing external wind data
az – azimuth clockwise from north [rad]
speed – magnitude of horizontal component [m/s]

Vcomp – vertical component: positive up, negative down [m/s]

Outputs:

beta – Magnitude of angle of side slip [rad]
quad – Classical trigonometric quadrant of the angle

VI ComputeAccel

This VI computes the total acceleration in the body frame due to aerodynamic forces.
aDx = -CA*q*S/mass
aDy = -CY*q*S/mass
aDz = -CN*q*S/mass
Where: q – Dynamic pressure [Pa]
S – Reference area
mass – Mass of vehicle

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.
Aero Force Coefficients
CA – Roll axis aero force coefficient
CY – Pitch axis aero force coefficient
CN – Yaw axis aero force coefficient
mass – Vehicle mass [kg]
rECI – Vehicle ECI position vector [m]
vECI – Vehicle ECI velocity vector [m/sec]
rho – Atmospheric density [kg/m^3]
S – Vehicle cross sectional reference area [m^2]
windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [Bool]
windData – Vector containing external wind data
az – azimuth clockwise from north [rad]
speed – magnitude of horizontal component [m/s]
Vcomp – vertical component: positive up, negative down [m/s]

Outputs:

aBdAero – Acceleration due to aero forces in body frame [m/sec^2]

VI ComputeTorque

This VI computes the total torque in the body frame due to aerodynamic forces.

$$tBdAero_X = C_l \bar{q} SD$$

$$tBdAero_Y = C_m \bar{q} SD + C_N \bar{q} S \cdot h_{Arm}$$

$$tBdAero_Z = C_n \bar{q} SD + C_Y \bar{q} S \cdot h_{Arm}$$

Where: $\bar{q}$ - Dynamic pressure [Pa]
S - Reference Area
D - Reference diameter

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Aero Force Coefficients

CA – Roll axis aero force coefficient (NOT USED)

CY – Pitch axis aero force coefficient

CN – Yaw axis aero force coefficient

Aero Torque Coefficients

Cl – Roll axis aero torque coefficient

Cm – Pitch axis aero torque coefficient

Cn – Yaw axis aero torque coefficient

rECI – Vehicle ECI position vector [m]

vECI – Vehicle ECI velocity vector [m/sec]

rho – Atmospheric density [kg/m^3]

S – Vehicle cross sectional reference area [m^2]

D – Vehicle reference length [m]

mArm – Moment arm (Center of Gravity from Theoretical Nose Tip minus Center of Pressure from Theoretical Nose Tip in body frame, both CG and CP are positive quantities) [m]

windFlag – wind model (TRUE external, FALSE - use earth rotation rate) [Bool]

windData – Vector containing external wind data

az – azimuth clockwise from north [rad]

speed – magnitude of horizontal component [m/s]

Vcomp – vertical component: positive up, negative down [m/s]

Outputs:

aeroTorque –Torque due to aero forces in body frame [N-m]

Time Utilities Component

Time is fundamental to many astrodynamic computations and system simulations. It is especially important in high precision applications involving geo-location (placing things on the earth), and high fidelity orbit propagation. Miss-management of time leads to a variety of S/W difficulties that are very hard to trace and correct. It is therefore imperative to have a complete time management library. The LabVIEW Aerospace Toolkit provides the user with the ability to perform various time conversions and computations and manage time effectively.

Time as used in astrodynamics or in simulation can have many scales and forms. Additionally, the advent of computers has brought the additional complication of different standards for representing time. The ATA Aerospace Toolkit uses common computing conventions for representing and calculating time. For more information

about these conventions please see the section in this manual “LabVIEW Programming Considerations”, subsection “Time”.

Time has various scales and forms of these scales. The ATA time library handles the following scales in the listed forms:

Time Scales

- UTC – Universal Time Coordinated. This is civil time that is used all over the world. Time zones are offsets from Greenwich Mean Time (Time at Greenwich). Forms:
    - Year, Month, Day, Hour, Minute, Second
    - Seconds since Unix Epoch (epoch of January 1, midnight, 1970)
    - Julian day (number of days that have elapsed since Monday, January 1, 4713 B.C. (Noon)) Convenient for astronomical computations.
    - Date (YEAR, MONTH, DAY + Fraction) Year is simply four digit common era. Month is 1-based (Jan=1, Dec=12). Day is 1-based and includes fraction component (e.g. noon on the first is 1.5) Convenient representation for Greenwich Hour Angle type calculations.

- UT1 - Designed to closely track earth rotation, which is irregular. Leap seconds are introduced into UTC so that UTC is never more than 0.9 seconds from UT1.
  Forms:
    - TUT1 - Julian centuries since January 1, Noon, 2000 (JD 2451545.0). Note: There are 36525.0 Julian days in a Julian Century.

- TT - This is Terrestrial Time, which is the independent variable in the equations of motion for things that move around the earth. Conceptually, it is a uniform time scale that would be measured by an ideal clock that is on the surface of the geoid.
  Forms:
  Julian centuries since January 1, Noon, 2000

- TDB - This is Barycentric Dynamical Time. It is the independent variable in the equations of motion for things that move around the barycenter (Center of mass) of the solar system.
  Forms:
  Julian centuries since January 1, Noon, 2000

- TAI - This is International Atomic Time. It is the practical realization of a uniform time scale based on atomic clocks (The correct frequency for the particular cesium resonance is now defined by international agreement as 9,192,631,770 Hz so that when divided by this number the output is exactly 1 Hz, or 1 cycle per second.) TAI agrees with TT at a constant offset of 32.184 seconds. TAI is only used in the ATA library to convert from UTC to TT, TDB, and TUT1.

UTC-TAI History
Relationship between TAI and UTC, until 27 December 2005

Limits of validity(at 0h UTC)	TAI – UTC
1972 Jan. 1 - 1972 Jul. 1	10s
1972 Jul. 1 - 1973 Jan. 1	11s
1973 Jan. 1 - 1974 Jan. 1	12s
1974 Jan. 1 - 1975 Jan. 1	13s
1975 Jan. 1 - 1976 Jan. 1	14s
1976 Jan. 1 - 1977 Jan. 1	15s
1977 Jan. 1 - 1978 Jan. 1	16s
1978 Jan. 1 - 1979 Jan. 1	17s
1979 Jan. 1 - 1980 Jan. 1	18s
1980 Jan. 1 - 1981 Jul. 1	19s
1981 Jul. 1 - 1982 Jul. 1	20s
1982 Jul. 1 - 1983 Jul. 1	21s
1983 Jul. 1 - 1985 Jul. 1	22s
1985 Jul. 1 - 1988 Jan. 1	23s
1988 Jan. 1 - 1990 Jan. 1	24s
1990 Jan. 1 - 1991 Jan. 1	25s
1991 Jan. 1 - 1992 Jul. 1	26s
1992 Jul. 1.- 1993 Jul 1	27s
1993 Jul. 1 - 1994 Jul. 1	28s
1994 Jul. 1 - 1996 Jan. 1	29s
1996 Jan. 1 - 1997 Jul. 1	30s
1997 Jul. 1.- 1999 Jan. 1	31s
1999 Jan. 1.- 2006 Jan. 1	32s
2006 Jan. 1.- 2009 Jan. 1	33s
2009 Jan. 1.-	34s

Leap Seconds Implementation

The Aerospace Toolkit supports leap second conversion between UTC, TAI and GPS Time. It DOES NOT support leap second conversion between UTC and SSE.

Properly accounting for leap seconds can be a tricky issue in simulation as time moves forward (or back) across leap second boundaries. The organization IERS is the official body that determines when leap seconds occur and it publishes this information in the IERS Bulletin C. This bulletin is available via the Internet at the IERS website.

To account for leap seconds the ATA Aerospace Toolkit uses a global variable in LabVIEW in the form of a table (multi-dimensional array). The default table covers the time frame from January 1 1972 through December 31, 2009. At the time of this release the last leap second occurred on December 31, 2008 at 11:59:60.

The ATA Aerospace Toolkit time conversion functions support leap seconds beyond December 31 2009 by allowing the user to include data from future IERS Bulletin C issues. The IERS will publish a bulletin periodically to announce if and when leap

seconds will occur. These leap second updates are also available on the IERS website in the form of an XML file. The Aerospace Toolkit function ReadIERS will look for additional XML files and include them in the leap second table.

To be read properly by the ReadIERS function the XML use the XML schema used by the IERS Bulletin. An example of XML file is follows:

```
<BulletinC>
    <data>
        <date>2005-01-14</date>
         <number>29</number>
         <UT lineID="33">
              <startDate>1999-01-01</startDate>
               <startUTC>0</startUTC>
               <UTC_TAI unit="s">-32</UTC_TAI>
          </UT>
    </data>
</BulletinC>
```

Considerations For Real-Time Operation
As the ATA Aerospace Toolkit is intended for real-time applications, consideration has been given to the operation of these functions. File I/O and table lookups are costly processes in terms of CPU usage. Therefore, the functions in this toolkit are designed minimize potential costly operations. The function readIERS must be called at the beginning of an application, prior to time conversion functions. Ideally this is done as part of an initialization routine. Once called, this function should not be called again, in particular inside of a loop.

Included in this tool kit is a VI that gives the user the ability to perform ultra-fast leap second table lookups called LeapSecondTableFastCore.vi Again, the user will have first called the VI ReadIERS. It avoids the use of timestamps and clusters and can be used to adjust UTC in applications requiring the most stringent real-time performance.

VI UTCToJulianDay

Convert a Gregorian date to a Julian date. Note: this VI will not handle negative dates

Inputs:

- Date – [UTC Date] (see def. page 15)
  - yy – year (4 digit)
  - mm – month
  - dd – day including fraction

Outputs:

- jd – Julian day corresponding to input Date

VI GreenwichMeanSiderialTime

Computes the mean Greenwich hour angle, which is the angle between prime meridian and the mean vernal equinox of date. (Note: This VI does not accept negative inputs.)

Inputs:

Date – Date and time for Greenwich Hour Angle[UTC Date] (see def. page 15)
- yy – year (4 digit)
- mm – month
- dd – day including fraction

Outputs:

gha – Mean Greenwich hour angle [rad]

VI GreenwichApparentSiderialTime

Computes the rotation matrix associated with Greenwich Apparent Sidereal Time (GAST) Earth rotation. GAST is the Greenwich hour angle from the true vernal equinox.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

dUT1 –Correction from UTC to UT1 (from IERS bulletin) [sec]

deltaPsi - Nutation angle needed to compute Greenwich Apparent Sidereal Time from GMST (rad). Note: This is computed by the Earth Analysis Component by the VI ComputePolarMotionMatrix

Outputs:

theta – Rotation matrix to account for Greenwich Apparent Sidereal time

VI TimeStampToUTCDate

Converts UTC time in Exact_time structure to UTC Date format, consisting of three real values YEAR, MONTH, and DAY (fractional).

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

Date – Date and time corresponding to UTC input [UTC Date] (see def. page 15)
- yy – year (4 digit)
- mm – month
- dd – day including fraction

VI TimeStampToSSE

Converts a time UTC in exact time format to integer representation of the Unix Epoch or number of seconds since January 1, 1970 midnight. Valid during the Unix Epoch (safely any dates between 1970 and 2037)

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

Unix Epoch - Seconds since Unix Epoch (January 1, 1970 midnight) [int]

VI SSEToTimeStamp

Converts UTC in Unix Epoch form (seconds since midnight January 1, 1970) to UTC in Exact_time format. Valid during Unix Epoch (from 1970 through 2036).

Inputs:

Unix Epoch – Seconds since epoch

Outputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

VI ConvertUTCToTTime

Computes the Julian centuries in terrestrial time base from January 1.5 2000 from an input time UTC.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

T – Number of Julian centuries from 2000 1.5 terrestrial time base.

VI convertUTCToTDB

Computes Julian centuries since 1.5 2000 Barycentric Dynamical Time (TDB) base. This is defined as the independent variable of the equations of motion with respect to the barycenter of the solar system.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time

(converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

TDB – Julian centuries since 1.5 2000 (Barycentric Dynamical Time base)

VI ConvertUTCToTUT1

Converts time UTC to number of Julian centuries since 2000 1.5 UT1 time base.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

dUT1 – Time correction from UTC to UT1

Outputs:

TUT1 – Julian centuries since 2000 1.5 UT1 time base

VI UTCtoMJD

This function takes UTC time in the Exact Time structure and computes Modified Julian Days. Julian Days (JD) is a continuous count of days since Monday, noon on January 1st, 4713 BC. Modified Julian Days is a convenient form that first converts the scale to start at midnight, keeping with the tradition of other time scales and also converts JD to a number more easily stored in computers.

Inputs:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Outputs:

Modified Julian Days - Modified Julian Days

VI UTCToTAIandGPS

This VI converts time from UTC to both TAI and GPS Time. The output TAI format is the Gregorian Calendar (Year, Month, Day, Hours, Minutes, Seconds and Fractional Second). The GPS Time output is in Week Number and Seconds-Into-Week Number. This function supports leap seconds, which are read in from a global table. See ReadIERS for more details regarding leap seconds.

This function does not support local time input, all times are considered to be in UTC.

Inputs:

Time Stamp to UTC - Input UTC time to be converted.

Outputs:

TAI - TAI time represented in Gregorian calendar format
Week Number - Number of weeks that have passed since 23:59:47 UTC on August 21 1999. Week Number repeats every 19.6 years
S-I-W Number - Seconds that have elapsed since the beginning of the current week number.
Fractional Second - Remainder of S-I-W Number

VI ReadIERSXml:

This function must be called in an application once prior to the use of GPStoAtomicandUTC or UTCtoAtomicAndGPS. This function creates a table in LabVIEW in the form of a global variable for use leap second calculations.

In the ATA Aerospace Toolkit, leap seconds are supported back to Jan 1 1972. The default mode of operation (does not require further data from the user) accounts for leap seconds through Dec 1 2009. Any time entered between these dates does not require any additional information from the user for correct accounting of leap seconds.

Leap seconds conversion is supported in future dates by allowing the user to add IERS bulletin C updates via the IERS bulletin C XML file format available on the IERS website. The user simply needs to download all future updates in XML format and place them in a folder on the user's computer.

Inputs:

Path - location of folder containing IERS bulletin C update files in XML format.
Filter - filename convention used to locate IERS update files in the designated path. Default value will work with IERS bulletin filenames.

VI TAIToGPS

This VI converts time from a TAI Scale and Gregorian format to the GPS Time scale and Week Number/S-I-W Number format.

Inputs:

TAI - Atomic time in Gregorian format

Outputs:

Week Number - number of weeks since 23:59:47 UTC on August 21 1999
S-I-W Number - number of seconds that have elapsed since the start of the current week number.

VI GPSToTAIandUTC

This VI converts GPS Time to UTC. There are also optional outputs for conversion to TAI and for the representation of GPS Time in a Gregorian calendar format.
Inputs:

Week Number - Number of weeks since 23:59:47 UTC on August 21 1999.

S-I-W Number - Seconds-Into-Week Number, the number of seconds that have elapsed since the start of the current Week Number. (sec)

Outputs:

UTC - Time in UTC
GPS Gregorian - GPS Time in a Gregorian calendar format
TAI Gregorian - TAI time represented in Gregorian calendar format

Mass Properties Computations Component

The Mass Properties Computations Component of the LabVIEW Aerospace Toolkit provides the user with the ability to transform mass properties into different frames of reference as well as compile mass properties of constituent parts into a common reference frame. This is accomplished through repeated application of the parallel axis theorem.

VI CompileMassProperties

Starting with the mass properties of a primary component repeatedly applies the parallel axis theorem to add in the mass properties of additional components to determine a composite mass, center of gravity and inertial tensor in the body frame. The mass properties of the component elements to be added to the primary component are given in array form. Each component corresponds to an element index in the arrays (e.g. element 0 of the Component DCM array, Body to Component array, and IT Component array are the component mass array all correspond to the mass properties of component 0). The frame of the primary component is the frame used to express the composite mass properties.

Inputs:

- Primary CG – Primary component center of gravity vector (body frame) [m]
- Primary mass – primary component mass [kg]
- Primary IT – primary component inertial tensor (body frame) [kg-m^2/rad]
- Component DCM – Array of DCMs to translate from component frame to body frame [m]
- Body to Component – Array of vectors for each component from origin of component frame to origin of body frame expressed in the body frame [m]
- Component CG – Array of center of gravity vectors of each component expressed in the component frame [m]
- Component mass – Array of masses for each component [kg]
- IT Components – Array of Inertia tensors of each component expressed in component frame [kg-m^2/rad]

Outputs:

- CompositeCG – Composite center of gravity vector in the body frame [m]
- CompositeMass – Total mass of composite [kg]
- CompositeIT – Composite inertia tensor in the body frame [kg-m^2/rad]

VI ComputeMassProperties

This function computes the mass properties from vehicle data input from a file. The user must specify the path of the input file.

The format of the mass component input file is a text file in which each line represents a mass component. The data file must have the following format:

- Only one record per line
- The data elements of each record are separated by a white space character (tab or space).
- No spaces between records (lines)
- Maximum 120 characters per record
- Each record contains four columns (mass (kg), X component of mass position (m), Y component of mass position (m), Z component of mass position (m))

Input:

n - Number of data points
Input mass component file

Outputs:

Total Mass - Total mass (kg)
CG Vector - Center of gravity vector in body frame (m)
Inertia Tensor - Inertia tensor in the body frame (kg*m^2)

Express VIs

The ATA Aerospace Toolkit currently offers one Express VI. The ExpressFindOrbit VI was created to aid users in quickly finding orbits.

ExpressFindOrbit

This function is the Express VI version of OrbitOverPositionJ2000. Express VIs give the user the ability to configure VIs through a graphical user interface. This VI has three basic modes of operation, dialog only, wired inputs and wired inputs and outputs. If a user simply wants to find an orbit, they only need to drop the Express VI on the block diagram. From the dialog box the user may enter their parameters and select “Find”. The orbit solution will be output to the dialog box. A user may also wire any of the inputs to the VI and open the dialog box (by double clicking on the Express VI). The user then has the option of selecting “Find”, or configuring any additional inputs. The user may also use the ExpressFindOrbit VI as a standard VI. Note, Express VIs do have an additional processing overhead. If a knows that this VI will be used strictly as a standard VI, with wired inputs and outputs, then they should use the standard OrbitOverJ2000 VI located on the orbits palette.

This function computes a user defined orbit that passes over a user specified latitude, and longitude at a user specified point in time. The user can specify the period (semi-major axis), eccentricity, true anomaly, and inclination of the desired orbit. Since the position is fixed in inertial space by the latitude, longitude, and time, the argument of perigee is determined by the input true anomaly. Note, that for a circular orbit (e = 0), true anomaly will be the argument of perigee plus true anomaly. Therefore, the input true anomaly will not be distinguishable. The RAAN (Right Ascension of the Ascending Node) is determined by the input time, inclination, and longitude. Note, that the input inclination must satisfy the following constraint: lat <= inc <= PI - lat. The position vector is computed by finding the geocentric latitude corresponding to the input geodetic latitude (WGS84). The position magnitude is found using the semi-major axis (from input

period), eccentricity and true anomaly. These combined with the input longitude are used to compute a position vector in ECEF (spherical to rectangular conversion) which is converted to EMEJ2000. The orbit normal vector is found by a search using a vector that is perpendicular to the position vector. +Z cross R (position vector) is rotated incrementally about R until the angle between the rotated vector and +Z is within a tolerance of the input inclination. If the orbit is polar, (inc = 90 degrees) the orbit normal vector is found by rotating the +X axis by GAST (Greenwich Apparent Sidereal Time) + longitude - PI/2. The unit position vector (unit radial), unit normal (cross-track, and the cross product of these two (In-track) are used to form a RIC to ECI conversion. The velocity vector is computed in RIC using the flight path angle (computed from SMA, e, tru, and V magnitude). Then the RIC to ECI conversion is used to convert the velocity vector in RIC to ECI.

Inputs:

Latitude - Desired geodetic latitude of the position the orbit is to pass over (rad).
Longitude - Desired longitude of the position the orbit is to pass over (rad).
Period - Desired period of the orbit (seconds).
Eccentricity - Desired eccentricity of the orbit.
Inclination - Desired inclination of the orbit (rad).
True Anomaly - Desired true anomaly of the input latitude and longitude (rad).

TimeStampToUTC - Desired time the orbit passes over the input latitude and longitude. Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

File Inputs:

A default EOP data file is included with the ATA Aerospace Toolkit and is located in c:\Program Files\ATA Aero Toolkit. The file contains EOP data from the IERS Bulletin A, dated 6/12/2008. The user has the option creating a user defined data file by copying data from the Bulletin. In such a case the user can specify the file and location in the VI.

Outputs:

Position J2000 - Position of desired orbit in EMEJ2000 (m).
Velocity J2000 - Velocity of desired orbit in EMEJ2000 (m/sec).

Sub VIs

Several Sub VIs are included on the Palette as part of the ATA Aerospace Toolkit. These VIs were used in the development of the Toolkit and are made available to the user. Several VIs may be of use in application development, in particular the ATAErrorlGlobal.vi and the ATAErrorHandler.

VI ATAErrorGlobal

This global variable controls the occurrence of error dialogs. Error dialogs can be suppressed by switching the drop down menu on the global to "no dialog". This gives the user the option of utilizing error dialog pop-up menus during debug, or suppressing them and handling errors at another level.

VI ATAErrorHandler

The custom error handler used with the ATA Aerospace Toolkit. This VI handles errors generated by the DLL in the Toolkit. See Error Handling and Input Validation for more information.

Input:

Numeric – the error code associated with the error. See Error Handling and Input Validation in this manual, and also National Instruments documentation on custom error handling.
Custom Error Message – customized string corresponding to the error.
Type of Dialog – Type of dialog box (if any) the user chooses.

Output:

Error Out: a LabVIEW error cluster that can be handled by the LabVIEW error handler.

VI TimeStamptoDLLArray

Utility for converting the Exact Time UTC structure into an integer array for passing to the ATA Aerospace Toolkit DLL.

Input:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

Output:

DLL Array – Exact Time UTC array passed from the ATA Aerospace Toolkit

VI ExactTimeToTimeStamp

Utility for converting the Exact Time UTC structure into an Aerospace Toolkit Time Stamp Control.

Input:

DLL Array – Exact Time UTC array passed from the ATA Aerospace Toolkit

Output:

TimeStampToUTC – Time input for the function in either UTC or Local Time (converts to UTC). See TimeStampToUTC in the control section of the manual for more detail.

VI greaterThan0

Since the ATA Aerospace Toolkit has implemented physical units in the LabVIEW environment, the “greater than zero” and “less than zero” VIs provided in the LabVIEW comparison palette cannot be used unless the units are stripped from the value. This VI simply does that operation for the user.

Input:

Output:

x>0? – Boolean indicating result of the comparison

VI lessThan0

Since the ATA Aerospace Toolkit has implemented physical units in the LabVIEW environment, the “greater than zero” and “less than zero” VIs provided in the LabVIEW comparison palette cannot be used unless the units are stripped from the value. This VI simply does that operation for the user.

Input:

x – unitized value to compare to zero

Output:

x<0? - Boolean indicating result of the comparison

VIpolySkew3x3

Computes a skew 3x3 matrix from a unitized input vector.

Input:

3vec: input vector, may be unitized

Output:

Skew3x3 – 3x3 skew matrix with the same units as the input vector

VI trackExtrema

Keep running track of extrema of set of data point. Note this is different from a post-processing in that it keeps track AS DATA IS COLLECTED. Caller is responsible for setting initial value (VI must be initialized) of extrema input such that min is 'very high' and max is 'very low'

Input:

datum: data point being tracked.
extrema in:

Max – max datum value from previous call(s) for comparison
Min - minimum datum value from previous call(s) for comparison

Output:

extrema out:

Max – max datum value after comparison on this call
Min – min datum value after comparison on this call

VI LeapSecondTableFastCore

This VI is used by time conversion functions to determine the leap second offset between UTC and TAI.

Inputs

Year - Year of lookup
Month - Month of lookup
Day - Day of lookup

Outputs:

Difference - offset number of seconds (sec)

VI UnixTimeToLVDateTimeRec

LabVIEW often insists on making corrections for day light savings time based on the time zone settings of the host machine. This VI correctly translates the ATA Exact Time Structure (based on ANSI C time structure) into the LabVIEW date time record format.

Constants

The following are the constant values and definitions used during computations for LabVIEW Aerospace Toolkit.

MU – (3.986005e+14) gravitational constant [m^3/sec^2]
RE – (6378137.001) Equatorial radius of the earth [m]
RP – (6356752.316) Polar radius of the earth [meters]
J2 – (1.0826393e-3) J2 zonal harmonic [ ]
J3 – (-2.53215307e-6) J3 zonal harmonic [ ]
J4 – (-1.61098761e-6) J4 zonal harmonic [ ]
J5 – (-2.35785649e-7) J5 zonal harmonic [ ]
J6 – (5.43169846e-7) J6 zonal harmonic [ ]
ECC – (0.081819191) Earth ellipse eccentricity [ ]
FL – (0.003352811) Earth flattening factor [ ]
OMEGA – (7.292115e-5) earth axis rotation rate [rad/sec]
gSl – (9.81) Gravity at sea level [m/sec^2]
DAYSTOSECONDS - (86400) days to seconds conversion
RADTOARCS - (206264.8063) Radians to arc seconds
RADTOARCM - (3437.746771) Radians to arc minutes

DAT – (33.0) UTC to TAI (Leap seconds from IERS bulletin C)
DTT – (32.184) TAI to Terrestrial time
RHO0 – (1.752) Density parameter for exponential atmosphere model [kg/m^3]
H0 – (6.7e+3) Altitude parameter for exponential atmosphere model [m]
ENL – (2.718281828) Euler's Constant
R – (287.0) Gas constant
GAMMA – (1.405) Adiabatic polytropic constant

Example LabVIEW Aerospace Toolkit Computations

Several examples can be found in the LabVIEW installation path in:

```
\National Instruments\LabVIEW7.1\vi.lib\AeroToolkitVIs\EXAMPLES
```

These examples are provided for the user to provide ideas and insight into how the ATA Aerospace Toolkit can be used.

Attitude Analysis component

Example ReferenceGenerator

In this example, we show the user how to use the ATA LabVIEW Aerospace Toolkit to gain a handle on an SV attitude parameterization. For this example we want to point the SV +x body axis towards nadir, and the SV +Y body axis in the general direction of the orbital velocity vector. ReferenceGenerator will compute the DCM that describes this orientation of the SV body frame relative to the inertial frame (Inertial to body transformation). Once the DCM is computed, the ATA LabVIEW Aerospace Toolkit can be used to transform the DCM to any number of other attitude parameterizations (see Math Analysis Component).

tAlign is set to the nadir vector in the inertial frame
tAlign = -10000000.0 0.0 0.0 (m)
tPlanar is set to orbital velocity vector in the inertial frame
tPlanar = 0.0 4058.227750865 4836.407501659 (m/sec)
pAlign is set to the +X body axis
pAlign = 1.0 0.0 0.0
pPlanar is set to the +Y body axis
pPlanar = 0.0 1.0 0.0

The resulting DCM is:
-1.0000000000 0.0000000000 0.0000000000
0.0000000000 0.6427876097 0.7660444431

0.0000000000 0.7660444431 -0.6427876097

Example EulerFromNED

Often times, it is convenient for the purpose of display, to output a set of Euler angles for the purpose of visualizing an SV attitude. In this example, we show the user how to compute Euler angles from A North East down coordinate system given an SV attitude (inertial frame to body frame).

The time in UTC is:

12:00:00.000 PM
7/15/2006

The attitude in the form of a quaternion (Inertial to body frame):

q: 0.707106781 0.0 0.0 0.707106781

The position vector in inertial space was taken to be:

r: 4948354.85143538 2653380.46305995 3220373.7358087297 (m)

The classical aerospace Euler sequence was used:

seq: 313

Resolving the internal ambiguity both ways yields the two possible Euler angle sequences (NED to SV body) in degrees:

Seq1:

Phi: 15.0085

Theta: 65.8421

Psi: -123.229

Seq2:

Phi: -164.992

Theta: -65.8421

Psi: 56.7706

Coordinate Frame Transformation Component

Example ECEFToJ2000

The ability to convert to and express vectors in a variety of standard coordinate systems is essential to aerospace analysis. In this example, we show the user one such conversion. We show the user how to convert a state vector (position and velocity) in Earth Centered Earth Fixed coordinates to Earth Centered Inertial J2000 Mean of Epoch coordinates using the ATA LabVIEW Aerospace Toolkit.

The state in ECEF is as follows:

rECEF: 455933.1300034 -5211339.52250121 3638440.4871104858 (m)
vECEF: 0.0 0.0 0.0 (m/sec)

The time in UTC:

12:00:00.000 AM
8/7/2006

Earth orientation parameters for Aug 7th, 2006 (From IERS bulletin)
XP = 0.108131 (Arcseconds)
YP = 0.264367 (Arcseconds)
UT1 - UTC = 0.1818118 (sec)

The resulting state in EME J2000 is:

rECI: -3339610.79419292 -4024418.03606559 3640775.0590273980 (m)
vECI: 293.4765528138 -243.6994249794 -0.1787799674 (m/sec)

Orbit Analysis Component

Example FlightPathAngle

A very important quantity in orbital analysis and flight dynamics, is the flight path angle. This quantity is very easy to compute with the ATA LabVIEW Aerospace Toolkit.

Using a vehicle state vector in inertial space:

r: 1000000.0 0.0 0.0 (m)
v: 0.0 19964.9818432174 0.0 (m/sec)

The resulting flight path angle in degrees:

Flight Path Angle: 0.0

Example CartesianToKepler

In orbit analysis, it is very important to be able to paramerize an orbit in many different ways. For computations it may be important to express an orbit in a Cartesian representation (position and velocity vectors), but for understanding and visualization, it may be helpful to see the orbit in Keplerarian elements (semi-major axis, eccentricity, inclination, longitude of the ascending node, argument of perigee, and true anomaly). In this example, we show the user how to perform this conversion operation using the ATA LabVIEW Aerospace Toolkit. Many other conversions are posible (See Orbit analysis Component).

The SV state vector in a Cartesian reference frame:

r: -52571.6725787441 177121.1412202642 -1003127.2703775450 (m)
v: 177.8493430170 19272.2903733871 3393.5676239287 (m/sec)

The corresponding Keplerian elements (m and degrees):

a: 1000000.0
e: 0.02
i: 93.0
o: 90.0
w: 100.0
tru: 180.0

Earth Analysis Component

Example TrueObliquityAngle

Various quantities about the earth are of interest and importance in aerospace analysis. One such quantity is the true angle of the obliquity. This is the angle between the true equator of the earth and the true plane of the ecliptic (earth orbital plane) at a specified date. This computation is straight forward with the ATA LabVIEW Aerospace Toolkit.

Time in UTC:

12:00:00.000 AM
8/6/2006

The corresponding true angle of the obliquity:

23.44091882381386 degrees

Mathematical Analysis Component

Example DCMToQ

The ability to convert between various attitude parametrizations is essential in attitude analysis. In this example we show the user how to perform one such operation. Many conversions between attitude parameteraizations are supported by the ATA LabVIEW Aerospace Toolkit (See Math Analysis Component). In this example, we take the resulting DCM from example a1), and convert it to a quaternion.

Input DCM:

-1.0000000000	0.0000000000	0.0000000000
0.0000000000	0.6427876097	0.7660444431
0.0000000000	0.7660444431	-0.6427876097

The corresponding quaternion is:

i=0.0 j=0.9063077870 k=0.4226182617 s=0.0

Example jacobi3x3

In this example, we demonstrate how the ATA LabVIEW Aerospace Toolkit can be used to perform error/dispersion analysis. Let us say that the following covariance matrix represents the statistical distribution of a vehicle position vector that is assumed to be multivariate normal.

0.0250000000	0.0075000000	0.0017500000
0.0075000000	0.0070000000	0.0013500000
0.0017500000	0.0013500000	0.0004300000

In order to draw a random sample from the distribution, the matrix must be diagonalized so that the individual elements are statistically independent. Using jacobi3X3, the resulting diagonal matrix (matrix of eigenvalues) is:

0.0278769350	-0.0000000000	0.0000000000
-0.0000000000	0.0043938651	0.0000000000
0.0000000000	0.0000000000	0.0001591999

The associated matrix of eigenvectors needed to “re-introduce” the covariance into the random samples taken form the above diagonal matrix is:

0.9367684062 -0.3495846898 -0.0159842982
0.3414806903 0.9231313620 -0.1766901996
0.0765237956 0.1600594675 0.9841367159

Math Utilities Component

Example PolyCrossProduct

Taking the cross product between two vectors is fundamental to aerospace analysis. We demonstrate how to do this with the ATA LabVIEW Aerospace Toolkit.

Input Vectors:

Vector 1:

-3152563.0046245046 -5544544.8961477615 0.0000000000

Vector 2:

3623805.4792285943 5248682.2586472211 0.0

Resulting cross product:

0.0 0.0 3545550662848.9805000000

Example angleBetweenVector

Computing the angle between two vectors is also central to aerospace analysis. Here is how it is done with the ATA LabVIEW Aerospace Toolkit.

Input Vectors:

Vector 1:

-3152563.0046245046 -5544544.8961477615 0.0000000000

Vector 2:

3623805.4792285943 5248682.2586472211 0.0

The angle between the two vectors in degrees is:

175

Example SphereToRectangle

It is often easier to visualize a vector in terms of its right ascension (angle from the +x axis), declination (angle from fundamental plane) and magnitude, but more desirable computationally in the form of a 3 dimensional Cartesian vector. It is very straight forward to convert between the two with the ATA LabVIEW Aerospace Toolkit.

Inputs in degrees and meters:

RA: 30.0
DEC: 250.0
MAG: 1000.0

Resulting 3D Cartesian vector:

-296.1981327260 -171.0100716628 -939.6926207859

Orbit Adjust Components

Example BurnTime

The ideal rocket law is very useful for engine sizing, and engine burn duration needed to achieve a desired delta velocity for orbital maneuvers. The ATA LabVIEW Aerospace Toolkit makes this computation very accessible to the user by providing a number of computations associated with the ideal rocket law.

Inputs:

Initial mass 20,000.0 (kg)
Delta velocity for which burn time is required 50.0 (m/sec)
Engine specific impulse 250.0 (sec)
Engine thrust 40,000.0 (newtons)

Outputs:

Mass flow rate 16.3098878695 (kg/sec)
Burn time for required delta V 24.7468810628 (sec)
Final mass after burn 19596.3811447458 (kg)
Change in mass over the burn 403.6188552542 (kg)

Aerodynamic Utilities Components

Example AccelDueToDragECI

While a vehicle is traveling through the atmosphere, the effect of the atmosphere must be accounted for in order to accurately predict the vehicles trajectory. A lower fidelity, yet effective method for doing this is to compute the effect of drag an the vehicle. The ATA LabVIEW Aerospace Toolkit makes the computation available to the user.

Inputs:

Time in UTC:

2:06:02.000 AM
8/10/2006

Position vector in inertial frame:

858492.7907877 -5463821.375371399 3171873.7358087297

Velocity vector in inertial frame:

150.0 100.0 20.0

mass= 20000.0 (kg)
Reference Aerea = 10.0 (m^2)
Coefficient of Drag = 1.5
Atmosheric density = 0.909110881 (From US Standard 76 atmosphere)

Acceleration due to drag in the inertial frame:

21.3445744751 -3.2131560956 -1.7183701202

Example AngOfAttack

Another quantity of great importance in aerodynamic computations is the angle of attack. This functionality is also available to the user in the ATA LabVIEW Aerospace Toolkit.

Inputs:

Time in UTC:

2:06:02.000 AM
8/10/2006

Position vector in inertial frame:

858492.7907877 -5463821.375371399 3171873.7358087297

Velocity vector in inertial frame:

150.0 100.0 20.0

Attitude:

-0.8875945606	0.4564755998	0.0616921614
0.4318041934	0.7779300517	0.4564755998
0.1603777869	0.4318041934	-0.8875945606

Angle of attack in degrees and vehicle body frame quadrant the angle is in:

9.84574303190131 4

Time Utilities Component

Example UTCToJulianDay

Time conversions and management are extremely important in many aerospace applications. The ATA LabVIEW Aerospace Toolkit gives the user access to many time conversions and time management functions (See Time Utilities Component). In this example, we show the user how to convert from a time UTC, to Julian day. Julian Days are very useful in many astrodynamical computations.

This VI needs the time input to be in date format (See Time Utilities Component). It may be easier to first input time in Exact_time format (See Time Utilities Component), then use another ATA LabVIEW Aerospace Toolkit VI (ExactTimeToUTCDate) to convert from Exact Time format to date format.

Inputs:

Time in UTC:

6:00:00.000 AM
8/10/2006

Date format output:

2006.0 8.0 10.25

Input date format:

2006.0 8.0 10.25

Output Julian Day:

2453957.75000000000000

LabVIEW Aerospace Toolkit Projects

3DOF Ascent Trajectory

Introduction
This example is of a three degree-of-freedom simulation of a 3 stage rocket from ignition to orbit. The example is implemented as the evolution of a 14 component state vector, which is evolved using Runge-Kutta 4 integration. The state vector is in an Earth centered inertial frame and contains 3 Cartesian components for position, 3 Cartesian components for velocity, a 4 component quaternion for attitude, a 3 Cartesian components for angular rate, and a component for mass.

The simulation breaks up the integration of the state vector into 11 phases. Since the simulation fidelity is only three degrees of freedom, there is no way to simulate torques that would introduce angular acceleration and change the angular rates. Therefore in between phases angular rates are simply introduced in a kinematical way based on engine specifications and mission requirements. Also instantaneous changes in the mass are introduced to simulate the jettison of used stages. During each phase the state vector evolves due to accelerations due to motor thrusts, gravity and aerodynamic drag. Essentially forces are added together in a free-body diagram for all phases except the very first, the hold down.

The output of the simulation uses the 14 component state vector to calculate latitude, longitude and altitude, Kepler orbital elements, and body rates, which are displayed along with the state vector components on the front panel.

Inputs
Input parameters are all found on the Parameters Tab of the main front panel.

Ignition Time is necessary to determine actual UTC time during the mission simulation, so that it is possible to translate back and forth between inertial and geodetic frames and along with latitude, longitude and altitude an initial position in inertial space can be calculated. Since the mission begins in hold down the initial velocity in inertial space is assumed to be that of the surface of the Earth; this also determines the initial rotational velocity in inertial space. The simulation assumes an initial tilt angle of 0 degrees, i.e. the rocket is pointing straight up; along with a default rotation that defines the pitch rate as heading west. With these, the initial attitude can be fully determined. An initial vehicle mass is specified that includes fuel and payload. The burning of fuel, which has a

significant effect on the mass of a launch vehicle, is simulated by applying stage-dependent mass flow specifications.

The Sequence duration values and the Inclination of the Orbital Plane are the equivalent of mission sequencing and trajectory shaping parameters that would be loaded into the flight computer, if this were a real mission. They define the length of those phases of the mission which could be controlled by a flight computer. The powered stage flight mission phases are designed to execute for as long as there is fuel in the stage. Therefore the duration of those phases are determined by the Burn Time parameter associated with each of the three Engine Parameter clusters.

The body rate transition parameter and the Stage-based pitch rate parameters are kinematical assertions necessary to simulate a vehicle pitching over at different rates in a 3DOF simulation. The default values supplied with the 3DOF example are based on an actual mission. The body Rate Transition parameter is based on the arbitrary assumption that every state will be divided into two sub-phases; all using the same time ratio specified by this parameter. One easy way to increase the flexibility of this simulation (while keeping it a 3DOF) would be to have a different transition ratio for each stage.

Outputs & Execution Control

The supplied example does not create any output file or archive the results. Data is displayed numerically on the Control Tab of the main front panel. These also include two user controls that govern execution control. The user can execute the simulation as fast as possible by setting the Single Step/Continuous toggle switch to Continuous. Alternately in the Single Step mode every time the Start button is pressed the simulation advances one integration step of 10 ms. The user can change modes during a simulation executing part of it continuously and part of it stepwise. The mission time is displayed on the Control tab.

The outputs include ECI (Earth Centered Inertial) position and velocity vectors, plus an attitude quaternion to represent relationship between the body frame and ECI frame. The body rates angular output vector is in the body frame, NOT the ECI frame. Theta and eAxis represent an Eigen Axis of rotation and angle of rotation about the axis; these are another way of representing the attitude quaternion. Kepler element representation is also included on the front panel as well as conventional geodetic latitude, longitude and altitude.

The geodetic output, latitude, longitude and altitude are also displayed graphically on two other tabs on the main control panel, Alt and LatLon. Body rates (again – in the body frame) are also displayed graphically, although for this mission there is generally only a very small pitch rate.

G Code Design

The initial vehicle position is fed into the HoldDown VI that performs specific hold down phase logic and outputs a state vector at the end of hold down. The kinematical and mission sequence parameters are fed into a VI entitled timeBasedFlightSequencer where the values are stored in shift registers for faster execution during the main integration loop. The 3DOFAscentPseudoTorques VI is initialized with the kinematical parameters.

Integration of the equations of motion is done in a nested while loop. The outer loop represents the succession through the phases of the mission. At the beginning of each phase a VI entitled 3DOFAscentPseudoTorques is executed to kinematically change the body rates, since actual torques are not possible in this 3DOF simulation. Note that the kinematical parameters only had to be wired once, in an initialization mode before the loop; the parameters are remembered by the VI in shift registers. This is a design decision that both improves execution speed performance and prevents “wire clutter” in the main integration loop – allowing the wires inside the loop to represent true process variables and not parameters.

The inner loop basically performs Runge-Kutta 4 integration through each mission phase. Then flight sequencer logic checks to see if it is time to exit the inner loop and proceed to the next mission phase. Most of the other VIs in the inner loop related to converting the output state vector of the integration into the representations and formats displayed as output.