

Programmer's Toolbox

This document describes a library of MATLAB functions that can be used to create and support your own orbital mechanics analysis scripts. MATLAB demonstration scripts and functions are provided in the following areas of orbital mechanics:

- Kepler's equation
- Two-body orbital motion (initial value problem)
- Analytic orbit propagation functions
- Two-body state transition matrix
- Differential equations of orbital motion
- Lambert's problem (two-point boundary value problem)
- Mean and osculating orbital elements
- Predicting trajectory events
- Earth atmosphere models
- Atmosphere and gravity model data files

KEPLER'S EQUATION

This section describes several MATLAB functions that can be used to solve Kepler's equation for circular, elliptical, parabolic and hyperbolic orbits.

kepler1.m – Danby's method

This MATLAB function is based on a numerical solution of Kepler's equation devised by Professor J. M. A. Danby at North Carolina State University. Additional information about this algorithm can be found in "The Solution of Kepler's Equation", *Celestial Mechanics*, **31** (1983) 95-107, 317-328 and **40** (1987) 303-312.

The initial guess for Danby's method is

$$E_0 = M + 0.85 \operatorname{sign}(\sin M) e$$

The fundamental equation we want to solve is

$$f(E) = E - e \sin E - M = 0$$

which has the first three derivatives given by

$$f'(E) = 1 - e \cos E$$

$$f''(E) = e \sin E$$

$$f'''(E) = e \cos E$$

Orbital Mechanics with MATLAB

The iteration for an updated eccentric anomaly based on a current value E_n is given by the next four equations:

$$\begin{aligned}\Delta(E_n) &= -\frac{f}{f'} \\ \Delta^*(E_n) &= -\frac{f}{f' + \frac{1}{2}\Delta f''} \\ \Delta_n(E_n) &= -\frac{f}{f' + \frac{1}{2}\Delta f'' + \frac{1}{6}\Delta^2 f'''} \\ E_{n+1} &= E_n + \Delta_n\end{aligned}$$

This algorithm provides quartic convergence of Kepler's equation. This process is repeated until the following convergence test involving the fundamental equation is satisfied:

$$|f(E)| \leq \varepsilon$$

where ε is the convergence tolerance. This tolerance is hardwired in the software to $\varepsilon = 1.0\text{e-}10$. Finally, the true anomaly can be calculated with the following two equations

$$\sin \theta = \sqrt{1 - e^2} \sin E$$

$$\cos \theta = \cos E - e$$

and the four quadrant inverse tangent given by

$$\theta = \tan^{-1}(\sin \theta, \cos \theta)$$

If the orbit is hyperbolic the initial guess is

$$H_0 = \log\left(\frac{2M}{e} + 1.8\right)$$

where H_0 is the hyperbolic anomaly. The fundamental equation and first three derivatives for this case are as follows:

$$f(H) = e \sinh H - H - M$$

$$f'(H) = e \cosh H - 1$$

$$f''(H) = e \sinh H$$

$$f'''(H) = e \cosh H$$

Otherwise, the iteration loop which calculates Δ, Δ^* , and so forth is the same. The true anomaly for hyperbolic orbits is determined with this next set of equations:

$$\sin \theta = \sqrt{e^2 - 1} \sinh H$$

$$\cos \theta = e - \cosh H$$

The true anomaly is then determined from a four quadrant inverse tangent evaluation of these two equations.

The syntax of this MATLAB function is

```
function [eanom, tanom] = kepler1 (manom, ecc)

% solve Kepler's equation for circular,
% elliptic and hyperbolic orbits

% Danby's method

% input

% manom = mean anomaly (radians)
% ecc   = orbital eccentricity (non-dimensional)

% output

% eanom = eccentric anomaly (radians)
% tanom = true anomaly (radians)
```

kepler2.m – Danby’s method with Mikkola’s initial guess

This function solves the elliptic and hyperbolic form of Kepler’s equation using Danby’s method and Mikkola’s initial guess. This method uses a cubic approximation of Kepler’s equation for the initial guess. Additional information about this initial guess can be found in “A Cubic Approximation For Kepler’s Equation”, *Celestial Mechanics*, **40** (1987) 329-334.

The elliptic orbit initial guess for this method is given by the expression

$$E_0 = M + e(3s - 4s^3)$$

where

$$s = s_1 + ds$$

$$s_1 = z - \frac{\alpha}{2}$$

$$ds = -\frac{0.078s_1^5}{1 + e}$$

and

$$z = \text{sign}(\beta) \left(|\beta| + \sqrt{\alpha^2 + \beta^2} \right)^{1/3}$$

$$\alpha = \frac{1-e}{4e + \frac{1}{2}}$$

$$\beta = \frac{\frac{1}{2}M}{4e + \frac{1}{2}}$$

Updates to the eccentric anomaly are calculated using the same set of equations as those in Danby's method. For hyperbolic orbits this method uses the following for the correction to s

$$ds = \frac{0.071s^5}{e(1+0.45s^2)(1+4s^2)}$$

and the following initial guess for the hyperbolic anomaly:

$$H_0 = 3 \log \left(s + \sqrt{1+s^2} \right)$$

The syntax of this MATLAB function is

```
function [eanom, tanom] = kepler2 (manom, ecc)

% solve Kepler's equation for circular,
% elliptic and hyperbolic orbits

% Danby's method with Mikkola's initial guess

% input

% manom = mean anomaly (radians)
% ecc   = orbital eccentricity (non-dimensional)

% output

% eanom = eccentric anomaly (radians)
% tanom = true anomaly (radians)
```

kepler3.m – Gooding's two iteration method

This MATLAB function solves the elliptic form of Kepler's equation using Gooding's two iteration method. This algorithm performs two, and only two iterations when solving Kepler's equation. Additional information about this technique can be found in "Procedures For Solving Kepler's Equation", *Celestial Mechanics*, **38** (1986) 307-334.

Orbital Mechanics with MATLAB

The syntax of this MATLAB function is

```
function [eanom, tanom] = kepler3 (manom, ecc)
% solve Kepler's equation for elliptic orbits
% Gooding's two iteration method
% input
% manom = mean anomaly (radians)
% ecc   = orbital eccentricity (non-dimensional)
% output
% eanom = eccentric anomaly (radians)
% tanom = true anomaly (radians)
```

kepler4.m – parabolic and near-parabolic orbits

This MATLAB function solves Kepler's equation for heliocentric parabolic and near-parabolic orbits. It is based on the numerical method described in Chapter 4 of *Astronomy on the Personal Computer* by Oliver Montenbruck and Thomas Pfleger. This algorithm uses a modified form of Barker's equation and Stumpff functions to solve this problem.

The form of Kepler's equation solved by this function is

$$E(t) - e \sin E(t) = \sqrt{\frac{\mu}{a^3}} (t - t_0)$$

where

E = eccentric anomaly

e = orbital eccentricity

μ = gravitational constant of the Sun

a = semimajor axis

t = time

t_0 = time of perihelion passage

The relationship between semimajor axis a and perihelion radius q is as follows:

$$a = \frac{q}{1 - e}$$

By introducing the variable

$$U = \sqrt{\frac{3ec_3(E)}{1 - e}} E$$

Kepler's equation is now given by

$$U + \frac{1}{3}U^3 = \sqrt{6ec_3(E)} \sqrt{\frac{\mu}{2q^3}} (t - t_0)$$

where $c_3(E) = (E - e \sin E)/E^3$. The `kepler4` function iteratively solves for U .

The heliocentric distance is determined from the expression

$$r = q \left(1 + \left[\frac{2c_2}{6c_3} \right] U^2 \right)$$

The true anomaly is determined from the x and y components of the heliocentric position vector as follows:

$$\theta = \tan^{-1}(y, x)$$

where

$$x = q \left(1 - \left[\frac{2c_2}{6ec_3} \right] U^2 \right)$$

$$y = 2q \sqrt{\frac{1+e}{2e} \left[\frac{1}{6c_3} \right]} c_1 U$$

The true anomaly can also be determined from

$$\tan\left(\frac{\theta}{2}\right) = \left(\sqrt{\frac{1+e}{3ec_3} \frac{c_2}{c_1}} \right) U$$

The c functions used in these equations are called Stumpff functions. They are named after the German astronomer Karl Stumpff and defined by the series

$$c_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{(2k+n)!} \quad k = 0, 1, 2, \dots$$

For x real and $x \neq 0$, the first few terms are given by the following expressions:

$$\begin{aligned} c_0(x^2) &= \cos x & c_0(-x^2) &= \cosh x \\ c_1(x^2) &= \frac{\sin x}{x} & c_1(-x^2) &= \frac{\sinh x}{x} \\ c_2(x^2) &= \frac{1 - \cos x}{x} & c_2(-x^2) &= \frac{\cosh x - 1}{x} \end{aligned}$$

Orbital Mechanics with MATLAB

The Stumpff functions also satisfy the recursion relationship defined by

$$xc_{k+2}(x) = \frac{1}{k!} - c_k(x) \quad k = 0, 1, 2, \dots$$

For $x = 0$,

$$c_n(x) = \frac{1}{n!}$$

It is most efficient to compute c_2 and c_3 by series and then compute c_0 and c_1 by recursion according to the following:

$$\begin{aligned} c_0(x) &= 1 - xc_2 \\ c_1(x) &= 1 - xc_3 \end{aligned}$$

The syntax of this MATLAB function is

```
function [r, tanom] = kepler4(t, q, ecc)

% Kepler's equation for heliocentric
% parabolic and near-parabolic orbits

% input

% t = time relative to perihelion passage (days)
% q = perihelion radius (AU)
% ecc = orbital eccentricity (non-dimensional)

% output

% r = heliocentric distance (AU)
% tanom = true anomaly (radians)
```

TWO BODY ORBITAL MOTION

The following three MATLAB functions can be used to propagate two body or “unperturbed” satellite orbits. For programming flexibility, the input and output arguments for each routine are identical. This propagation is also called the orbital initial value problem (IVP).

twobody1.m – Goodyear’s method

This function is a MATLAB implementation of Goodyear’s two body propagation algorithm.

Goodyear’s method uses a change of variables that allows one to use a single set of equations to predict elliptic, hyperbolic and parabolic motion. This change of variables is defined by

$$\dot{\psi} = 1/r$$

where r is the distance of the spacecraft or celestial body.

The relationship between this generalized anomaly ψ and the classical eccentric anomaly E and hyperbolic anomaly F is given by

$$\psi = \frac{E - E_0}{\sqrt{\mu/a}} = \frac{F - F_0}{\sqrt{-\mu/a}}$$

The equations used to calculate the final position vector \mathbf{r} and the final velocity vector $\dot{\mathbf{r}}$ from the initial position and velocity vectors \mathbf{r}_0 and $\dot{\mathbf{r}}_0$ are as follows:

$$\begin{aligned} r_0 &= \sqrt{\mathbf{r}_0 \bullet \mathbf{r}_0} \\ \sigma_0 &= \mathbf{r}_0 \bullet \dot{\mathbf{r}}_0 \\ \alpha &= \mathbf{r}_0 \bullet \dot{\mathbf{r}}_0 - 2\frac{\mu}{r_0} \end{aligned}$$

The generalized eccentric anomaly ψ and the four transcendental functions given by

$$\begin{aligned} s_0 &= 1 + \frac{\alpha\psi^2}{2!} + \frac{\alpha^2\psi^4}{4!} + \frac{\alpha^3\psi^6}{6!} + \dots \\ s_1 &= \psi + \frac{\alpha\psi^3}{3!} + \frac{\alpha^2\psi^5}{5!} + \frac{\alpha^3\psi^7}{7!} + \dots \\ s_2 &= \frac{\psi^2}{2!} + \frac{\alpha\psi^4}{4!} + \frac{\alpha^2\psi^6}{6!} + \frac{\alpha^3\psi^8}{8!} + \dots \\ s_3 &= \frac{\psi^3}{3!} + \frac{\alpha\psi^5}{5!} + \frac{\alpha^2\psi^7}{7!} + \frac{\alpha^3\psi^9}{9!} + \dots \end{aligned}$$

can be obtained by solving the next equation

$$t = t_0 + r_0 s_1 + \sigma_0 s_2 + \mu s_3$$

for the value of ψ which satisfies all these equations.

The position magnitude at the final time is given by

$$r = r_0 s_0 + \sigma_0 s_2 + \mu s_3$$

The f and g functions and their derivatives are calculated with the four equations given by

Orbital Mechanics with MATLAB

$$f = 1 - \mu \frac{S_2}{r_0}$$

$$\dot{f} = -\mu \frac{S_1}{rr_0}$$

$$g = (t - t_0) - \mu S_3$$

$$\dot{g} = 1 - \mu \frac{S_2}{r}$$

Finally, the position and velocity vectors at the final time t are computed from the f and g functions with the next two equations:

$$\mathbf{r}(t) = f \mathbf{r}_0 + g \dot{\mathbf{r}}_0$$

$$\dot{\mathbf{r}}(t) = \dot{f} \mathbf{r}_0 + \dot{g} \dot{\mathbf{r}}_0$$

The two-body gravity acceleration vector at the final time is given by

$$\ddot{\mathbf{r}}(t) = -\mu \frac{\mathbf{r}}{|\mathbf{r}|^3}$$

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
tbcoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script which calls this function.

The syntax of this MATLAB function is

```
function [rf, vf] = twobody1 (mu, tau, ri, vi)  
  
% solve the two body initial value problem  
  
% Goodyear's method  
  
% input  
  
% mu = gravitational constant (km**3/sec**2)  
% tau = propagation time interval (seconds)  
% ri = initial eci position vector (kilometers)  
% vi = initial eci velocity vector (km/sec)  
% output  
  
% rf = final eci position vector (kilometers)  
% vf = final eci velocity vector (km/sec)
```

twobody2.m – Sheppard’s method

This function is a MATLAB implementation of Stanley Sheppard’s two body propagation algorithm. The derivation of this algorithm is described in “Universal Keplerian State Transition Matrix”, *Celestial Mechanics*, **35** (1985) 129-144. This algorithm uses a combination of universal variables and continued fractions to propagate two-body orbits.

The syntax of this MATLAB function is

```
function [rf, vf] = twobody2 (mu, tau, ri, vi)

% solution of the two body initial value problem

% Sheppard's method

% input

% mu = gravitational constant (km**3/sec**2)
% tau = propagation time interval (seconds)
% ri = initial eci position vector (kilometers)
% vi = initial eci velocity vector (km/sec)

% output

% rf = final eci position vector (kilometers)
% vf = final eci velocity vector (km/sec)
```

twobody3.m – Stumpff/Danby method

This function is a MATLAB implementation of the Stumpff/Danby two body propagation algorithm. It is based on a universal variable technique and Stumpff functions. This function is valid for elliptic and hyperbolic orbits. The derivation of this algorithm can be found in Professor Danby’s classic text, *Fundamentals of Celestial Mechanics*.

The algorithm begins by computing an initial guess for s according to

$$x = \alpha s^2$$

where

\mathbf{r}_0 = position vector at initial time t_0

\mathbf{v}_0 = velocity vector at initial time t_0

$$r_0 = |\mathbf{r}_0| = \sqrt{r_{0x}^2 + r_{0y}^2 + r_{0z}^2}$$

$$v_0 = |\mathbf{v}_0| = \sqrt{v_{0x}^2 + v_{0y}^2 + v_{0z}^2}$$

$$\alpha = \frac{2\mu}{r_0} - v_0^2$$

Orbital Mechanics with MATLAB

This algorithm uses a universal variable form of Kepler's equation defined by

$$f(s) = r_0 s c_1(\alpha s^2) + r_0 \dot{r}_0 s^2 c_2(\alpha s^2) + \mu s^3 c_3(\alpha s^2) - (t - t_0) = 0$$

The first three derivatives of this form of Kepler's equation are given by

$$f'(s) = r_0 c_0(\alpha s^2) + r_0 \dot{r}_0 s c_1(\alpha s^2) + \mu s^2 c_2(\alpha s^2)$$

$$f''(s) = (-r_0 \alpha + \mu) s c_1(\alpha s^2) + r_0 \dot{r}_0 c_0(\alpha s^2)$$

$$f'''(s) = (-r_0 \alpha + \mu) c_0(\alpha s^2) - r_0 \dot{r}_0 \alpha s c_1(\alpha s^2)$$

The c functions used in these equations are called Stumpff functions. They are named after the German astronomer Karl Stumpff and are defined by the series

$$c_n(x) = \sum_{k=0}^{\infty} \frac{(-1)^k x^k}{(2k+n)!} \quad k = 0, 1, 2, \dots$$

for x real and $x \neq 0$

$$c_0(x^2) = \cos x \quad c_0(-x^2) = \cosh x$$

$$c_1(x^2) = \frac{\sin x}{x} \quad c_1(-x^2) = \frac{\sinh x}{x}$$

$$c_2(x^2) = \frac{1 - \cos x}{x} \quad c_2(-x^2) = \frac{\cosh x - 1}{x}$$

The Stumpff functions also satisfy the recursion relationship defined by

$$x c_{k+2}(x) = \frac{1}{k!} - c_k(x) \quad k = 0, 1, 2, \dots$$

For $x = 0$,

$$c_n(x) = \frac{1}{n!}$$

It is most efficient to compute c_2 and c_3 by series, and then compute c_0 and c_1 by recursion according to the following:

$$c_0(x) = 1 - x c_2$$

$$c_1(x) = 1 - x c_3$$

Orbital Mechanics with MATLAB

The equations for the f and g functions and derivatives are as follows:

$$f = 1 - \left(\frac{\mu}{r_0} \right) s^2 c_2(\alpha s^2)$$

$$g = t - t_0 - \mu s^3 c_3(\alpha s^2)$$

$$\dot{f} = - \left(\frac{\mu}{r r_0} \right) s c_1(\alpha s^2)$$

$$\dot{g} = 1 - \left(\frac{\mu}{r} \right) s^2 c_2(\alpha s^2)$$

Finally, the position and velocity vectors at the final time t are given by

$$\mathbf{r}(t) = f \mathbf{r}_0 + g \dot{\mathbf{r}}_0$$

$$\dot{\mathbf{r}}(t) = \dot{f} \mathbf{r}_0 + \dot{g} \dot{\mathbf{r}}_0$$

The syntax of this MATLAB function is

```
function [rf, vf] = twobody3 (mu, tau, ri, vi)

% solve the two body initial value problem
% Danby/Stumpff method
% input
% mu = gravitational constant (km**3/sec**2)
% tau = propagation time interval (seconds)
% ri = initial eci position vector (kilometers)
% vi = initial eci velocity vector (km/sec)

% output
% rf = final eci position vector (kilometers)
% vf = final eci velocity vector (km/sec)
```

THE TWO-BODY STATE TRANSITION MATRIX

These two MATLAB functions can be used to calculate the two-body state transition matrix of satellites in elliptic and hyperbolic orbits. For programming flexibility, the input and output arguments for each routine are identical.

Orbital Mechanics with MATLAB

The state transition matrix consists of the 36 partial derivatives of the final state vector defined by the six components $x, y, z, \dot{x}, \dot{y}$ and \dot{z} with respect to the six components of the initial state vector defined by $x_0, y_0, z_0, \dot{x}_0, \dot{y}_0$ and \dot{z}_0 .

$$\Phi(t, t_0) = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix}$$

where

$$\Phi_{11} = \begin{bmatrix} \frac{\partial \mathbf{r}}{\partial \mathbf{r}_0} \end{bmatrix} = \begin{bmatrix} \partial x / \partial x_0 & \partial x / \partial y_0 & \partial x / \partial z_0 \\ \partial y / \partial x_0 & \partial y / \partial y_0 & \partial y / \partial z_0 \\ \partial z / \partial x_0 & \partial z / \partial y_0 & \partial z / \partial z_0 \end{bmatrix}$$

$$\Phi_{12} = \begin{bmatrix} \frac{\partial \mathbf{r}}{\partial \mathbf{v}_0} \end{bmatrix} = \begin{bmatrix} \partial x / \partial \dot{x}_0 & \partial x / \partial \dot{y}_0 & \partial x / \partial \dot{z}_0 \\ \partial y / \partial \dot{x}_0 & \partial y / \partial \dot{y}_0 & \partial y / \partial \dot{z}_0 \\ \partial z / \partial \dot{x}_0 & \partial z / \partial \dot{y}_0 & \partial z / \partial \dot{z}_0 \end{bmatrix}$$

$$\Phi_{21} = \begin{bmatrix} \frac{\partial \mathbf{v}}{\partial \mathbf{r}_0} \end{bmatrix} = \begin{bmatrix} \partial \dot{x} / \partial x_0 & \partial \dot{x} / \partial y_0 & \partial \dot{x} / \partial z_0 \\ \partial \dot{y} / \partial x_0 & \partial \dot{y} / \partial y_0 & \partial \dot{y} / \partial z_0 \\ \partial \dot{z} / \partial x_0 & \partial \dot{z} / \partial y_0 & \partial \dot{z} / \partial z_0 \end{bmatrix}$$

$$\Phi_{22} = \begin{bmatrix} \frac{\partial \mathbf{v}}{\partial \mathbf{v}_0} \end{bmatrix} = \begin{bmatrix} \partial \dot{x} / \partial \dot{x}_0 & \partial \dot{x} / \partial \dot{y}_0 & \partial \dot{x} / \partial \dot{z}_0 \\ \partial \dot{y} / \partial \dot{x}_0 & \partial \dot{y} / \partial \dot{y}_0 & \partial \dot{y} / \partial \dot{z}_0 \\ \partial \dot{z} / \partial \dot{x}_0 & \partial \dot{z} / \partial \dot{y}_0 & \partial \dot{z} / \partial \dot{z}_0 \end{bmatrix}$$

stm1.m – Goodyear’s method

This MATLAB function calculates the two-body state transition matrix using Goodyear’s method.

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
stmcoef = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script which calls this function.

The syntax for this MATLAB function is

```
function [rf, vf, stm] = stm1 (mu, tau, ri, vi)
```

```
% two body state transition matrix
```

Orbital Mechanics with MATLAB

```
% Goodyear's method

% input

% mu = gravitational constant (km**3/sec**2)
% tau = propagation time interval (seconds)
% ri = initial eci position vector (kilometers)
% vi = initial eci velocity vector (km/sec)

% output

% rf = final eci position vector (kilometers)
% vf = final eci velocity vector (km/sec)
% stm = state transition matrix
```

stm2.m – Sheppard’s method

This MATLAB function calculates the two-body state transition matrix using Stanley Sheppard’s method. The derivation of this algorithm is described in “Universal Keplerian State Transition Matrix”, *Celestial Mechanics*, **35** (1985) 129-144.

The syntax for this MATLAB function is

```
function [rf, vf, stm] = stm2 (mu, tau, ri, vi)

% two body state transition matrix

% Sheppard's method

% input

% mu = gravitational constant (km**3/sec**2)
% tau = propagation time interval (seconds)
% ri = initial eci position vector (kilometers)
% vi = initial eci velocity vector (km/sec)

% output

% rf = final eci position vector (kilometers)
% vf = final eci velocity vector (km/sec)
% stm = state transition matrix
```

DIFFERENTIAL EQUATIONS OF ORBITAL MOTION

This section describes a series of MATLAB functions which compute the geocentric and heliocentric differential equations of orbital motion. For geocentric motion, both first-order and second-order routines are provided. Several functions also include the effects of orbital perturbations due to atmospheric drag, solar radiation pressure, and point-mass, third-bodies. MATLAB functions are also included that evaluate the modified equinoctial and relative flight path form of the equations of motion

j2eqm.m – first order equations of orbital motion with j2

This function calculates the first order ECI equations of motion due to a J_2 gravity model. The state vector is passed to this routine is a column vector \mathbf{y} with the components given by

$$\begin{aligned} y_1 &= r_x & y_2 &= r_y & y_3 &= r_z \\ y_4 &= v_x & y_5 &= v_y & y_6 &= v_z \end{aligned}$$

where r_x, r_y and r_z are the rectangular components of the position vector, and v_x, v_y and v_z are the rectangular components of the velocity vector.

The system of six first-order differential equations is defined by

$$\begin{aligned} \dot{y}_1 &= v_x = y_4 \\ \dot{y}_2 &= v_y = y_5 \\ \dot{y}_3 &= v_z = y_6 \\ \dot{y}_4 &= -\mu \frac{r_x}{r^3} \left\{ 1 + \frac{3 J_2 r_{eq}^2}{2 r^2} \left(1 - \frac{5 r_z^2}{r^2} \right) \right\} \\ \dot{y}_5 &= -\mu \frac{r_y}{r^3} \left\{ 1 + \frac{3 J_2 r_{eq}^2}{2 r^2} \left(1 - \frac{5 r_z^2}{r^2} \right) \right\} \\ \dot{y}_6 &= -\mu \frac{r_z}{r^3} \left\{ 1 + \frac{3 J_2 r_{eq}^2}{2 r^2} \left(3 - \frac{5 r_z^2}{r^2} \right) \right\} \end{aligned}$$

where $r = \sqrt{r_x^2 + r_y^2 + r_z^2} = \sqrt{y_1^2 + y_2^2 + y_3^2}$. In these equations, μ and r_{eq} are the gravitational constant and equatorial radius of the Earth, respectively.

The syntax of this MATLAB function is

```
function ydot = j2eqm (t, y)

% first order equations of orbital motion

% j2 gravitational acceleration

% input

% t = simulation time (seconds)
% y = state vector

% output
```

`% ydot = integration vector`

j4eqm.m – first order equations of orbital motion with j2, j3 and j4

This function calculates the first order form of the Earth-centered-inertial (ECI) equations of motion due to a gravity model that includes the zonal gravity effects of J_2, J_3 and J_4 .

The acceleration vector contributions due to J_3 are given by

$$a_x = -\frac{5J_3\mu r_{eq}^3 r_x}{2r^7} \left(3r_z - \frac{7r_z^3}{r^2} \right)$$

$$a_y = -\frac{5J_3\mu r_{eq}^3 r_y}{2r^7} \left(3r_z - \frac{7r_z^3}{r^2} \right)$$

$$a_z = -\frac{5J_3\mu r_{eq}^3}{2r^7} \left(6r_z^2 - \frac{7r_z^4}{r^2} - \frac{3}{5}r^2 \right)$$

The acceleration vector contributions due to J_4 are given by

$$a_x = \frac{15J_4\mu r_{eq}^4 r_x}{8r^7} \left(1 - \frac{14r_z^2}{r^2} + \frac{21r_z^4}{r^4} \right)$$

$$a_y = \frac{15J_4\mu r_{eq}^4 r_y}{8r^7} \left(1 - \frac{14r_z^2}{r^2} + \frac{21r_z^4}{r^4} \right)$$

$$a_z = \frac{15J_4\mu r_{eq}^4 r_z}{8r^7} \left(5 - \frac{70r_z^2}{3r^2} + \frac{21r_z^4}{r^4} \right)$$

The acceleration vector contributions due to J_2 are described in the previous function.

The syntax of this MATLAB function is

```
function ydot = j4eqm(t, y)

% first order equations of orbital motion

% includes j2, j3, j4

% input

% t = simulation time (seconds)
% y = state vector
```

```
% output
% ydot = integration vector
```

j4eqmn.m – second order equations of orbital motion with j2, j3 and j4

This function calculates the second order form of the ECI equations of motion due to a gravity model that includes the J_2, J_3 and J_4 zonal gravity effects.

The syntax of this MATLAB function is

```
function yddot = j4eqmn(t, r, v)
% second order equations of orbital motion
% includes j2, j3, j4
% input
% t = simulation time (seconds)
% r = position vector
% v = velocity vector
% output
% yddot = integration vector
```

j6eqm.m – first order equations of orbital motion with j2 through j6

This function calculates the first order ECI equations of motion due to a gravity model that includes the J_2, J_3, J_4, J_5 and J_6 zonal gravity coefficients.

The acceleration vector contributions due to J_5 are given by

$$a_x = \frac{3J_5\mu r_{eq}^4 r_x r_z}{8r^9} \left(35 - \frac{210r_z^2}{r^2} + \frac{231r_z^4}{r^4} \right)$$

$$a_y = \frac{3J_5\mu r_{eq}^5 r_y r_z}{8r^9} \left(35 - \frac{210r_z^2}{r^2} + \frac{231r_z^4}{r^4} \right)$$

$$a_z = \frac{3J_5\mu r_{eq}^5 r_z^2}{8r^7} \left(105 - \frac{315r_z^2}{r^2} + \frac{231r_z^4}{r^4} \right) + \frac{15J_5\mu r_{eq}^5}{8r^7}$$

The acceleration vector contributions due to J_6 are given by

Orbital Mechanics with MATLAB

$$a_x = -\frac{J_6 \mu r_{eq}^6 r_x}{16r^9} \left(35 - \frac{945r_z^2}{r^2} + \frac{3465r_z^4}{r^4} - \frac{3003r_z^6}{r^6} \right)$$
$$a_y = -\frac{J_6 \mu r_{eq}^6 r_y}{16r^9} \left(35 - \frac{945r_z^2}{r^2} + \frac{3465r_z^4}{r^4} - \frac{3003r_z^6}{r^6} \right)$$
$$a_z = -\frac{J_6 \mu r_{eq}^6 r_z}{16r^9} \left(245 - \frac{2205r_z^2}{r^2} + \frac{4851r_z^4}{r^4} - \frac{3003r_z^6}{r^6} \right)$$

The syntax of this MATLAB function is

```
function ydot = j6eqm(t, y)  
  
% first order equations of orbital motion  
  
% includes j2, j3, j4, j5, j6  
  
% input  
  
% t = simulation time (seconds)  
% y = state vector  
  
% output  
  
% ydot = integration vector
```

jzeqm1.m – first order “zonal” equations of orbital motion

This function calculates the first order form of the ECI equations of motion due to a gravity model that includes only zonal gravity coefficients.

The syntax of this MATLAB function is

```
function ydot = jzeqm1 (t, y)  
  
% first order "zonal" equations of motion  
  
% input  
  
% t = simulation time (seconds)  
% y = state vector  
  
% output  
  
% ydot = integration vector
```

jzeqm2.m – second order “zonal” equations of orbital motion

This function calculates the second order form of the ECI equations of motion due to a gravity model that includes only zonal gravity harmonic coefficients. The second-order, nonlinear

Orbital Mechanics with MATLAB

differential equations of orbital motion subject to both the spherical and zonal components of the Earth's gravitation attraction are given by

$$\ddot{\mathbf{r}} = -\mu \frac{\mathbf{r}}{|\mathbf{r}|^3} + \sum_{n=2}^N J_n \mathbf{U}_n$$

where \mathbf{r} is the inertial position vector of the satellite, J_n are the zonal coefficients of the Earth gravity and the "perturbed" or non-spherical contributions are

$$U_{2_x} = \frac{3}{2} \mu \frac{r_{eq}^2}{r^5} \left(5 \frac{z^2}{r^2} - 1 \right) x$$

$$U_{2_y} = \frac{3}{2} \mu \frac{r_{eq}^2}{r^5} \left(5 \frac{z^2}{r^2} - 1 \right) y$$

$$U_{2_z} = \frac{3}{2} \mu \frac{r_{eq}^2}{r^5} \left(5 \frac{z^2}{r^2} - 1 \right) z$$

$$U_{3_x} = \frac{1}{2} \mu \frac{r_{eq}^3}{r^6} \left(35 \frac{z^3}{r^3} - 15 \frac{z}{r} \right) x$$

$$U_{3_y} = \frac{1}{2} \mu \frac{r_{eq}^3}{r^6} \left(35 \frac{z^3}{r^3} - 15 \frac{z}{r} \right) y$$

$$U_{3_z} = \frac{1}{2} \mu \frac{r_{eq}^3}{r^6} \left(35 \frac{z^3}{r^3} - 30 \frac{z}{r} + 3 \frac{z}{r} \right) z$$

In these equations, μ is the gravitational constant of the Earth, r_{eq} is the equatorial radius of the Earth, r is the geocentric radius of the satellite and the inertial rectangular components of the position vector are x , y and z . The recurrence relationship for the U 's is as follows:

$$U_{n_x} = \left(\frac{2n+1}{n} \right) \frac{r_{eq}}{r} \frac{z}{r} U_{n-1_x} - \left(\frac{n+1}{n} \right) \frac{r_{eq}^2}{r^2} U_{n-2_x}$$

$$U_{n_y} = \left(\frac{2n+1}{n} \right) \frac{r_{eq}}{r} \frac{z}{r} U_{n-1_y} - \left(\frac{n+1}{n} \right) \frac{r_{eq}^2}{r^2} U_{n-2_y}$$

$$U_{n_z} = \left(\frac{2n+1}{n} \right) \frac{r_{eq}}{r} \frac{z}{r} U_{n-1_z} - \left(\frac{n+1}{n} \right) \frac{r_{eq}^2}{r^2} U_{n-2_z}$$

The syntax of this MATLAB function is

```
function yddot = jzeqm2(t, r, v)
% second order "zonal" equations of motion
```

```

% input

% t = simulation time (seconds)
% r = position vector
% v = velocity vector

% output

% yddot = integration vector
    
```

gravity.m – n by m gravity model – true-of-date coordinates

This MATLAB function evaluates an n by m gravity model data file and evaluates the Earth-centered-inertial (ECI) cartesian acceleration vector. This algorithm assumes that the position vector input to this routine is true-of-date

The syntax of the function that reads the gravity model data file is as follows:

```

function [ccoef, scoef] = readegm(fname)

% read gravity model data file

% input

% fname = name of gravity data file

% output

% ccoef, scoef = gravity model coefficients
    
```

In terms of the Earth's geopotential Φ , the *true-of-date*, inertial rectangular cartesian components of the satellite's acceleration vector are as follows:

$$\ddot{x} = \left(\frac{1}{r} \frac{\partial \Phi}{\partial r} - \frac{z}{r^2 \sqrt{x^2 + y^2}} \frac{\partial \Phi}{\partial \phi} \right) x - \left(\frac{1}{x^2 + y^2} \frac{\partial \Phi}{\partial \lambda} \right) y$$

$$\ddot{y} = \left(\frac{1}{r} \frac{\partial \Phi}{\partial r} - \frac{z}{r^2 \sqrt{x^2 + y^2}} \frac{\partial \Phi}{\partial \phi} \right) y + \left(\frac{1}{x^2 + y^2} \frac{\partial \Phi}{\partial \lambda} \right) x$$

$$\ddot{z} = \left(\frac{1}{r} \frac{\partial \Phi}{\partial r} \right) z + \left(\frac{\sqrt{x^2 + y^2}}{r^2} \frac{\partial \Phi}{\partial \phi} \right)$$

The three partial derivatives of the geopotential with respect to r, ϕ, λ are given by

$$\frac{\partial \Phi}{\partial r} = -\frac{1}{r} \left(\frac{\mu}{r} \right) \sum_{n=2}^N \left(\frac{R}{r} \right)^n (n+1) \sum_{m=0}^n (C_n^m \cos m\lambda + S_n^m \sin m\lambda) P_n^m(\sin \phi)$$

$$\frac{\partial \Phi}{\partial \phi} = \left(\frac{\mu}{r} \right) \sum_{n=2}^N \left(\frac{R}{r} \right)^n \sum_{m=0}^n (C_n^m \cos m\lambda + S_n^m \sin m\lambda) [P_n^{m+1}(\sin \phi) - m \tan \phi P_n^m(\sin \phi)]$$

$$\frac{\partial \Phi}{\partial \lambda} = \left(\frac{\mu}{r} \right) \sum_{n=2}^N \left(\frac{R}{r} \right)^n \sum_{m=0}^n m (S_n^m \cos m\lambda - C_n^m \sin m\lambda) P_n^m(\sin \phi)$$

where

R = radius of the Earth

r = geocentric distance of the satellite

S_n^m, C_n^m = harmonic coefficients

ϕ = geocentric latitude of the satellite = $\sin^{-1} \left(\frac{z}{r} \right)$

λ = longitude of the satellite = $\alpha - \alpha_g$

α = right ascension of the satellite = $\tan^{-1} \left(\frac{y}{x} \right)$

α_g = right ascension of Greenwich

Right ascension is measure positive east of the vernal equinox, longitude is measured positive east of Greenwich, and latitude is positive above the Earth's equator and negative below.

For $m = 0$ the coefficients are called *zonal* terms, when $m = n$ the coefficients are *sectorial* terms, and for $n > m \neq 0$ the coefficients are called *tesseral* terms.

The Legendre polynomials with argument $\sin \phi$ are computed using recursion relationships given by:

$$P_n^0(\sin \phi) = \frac{1}{n} [(2n-1) \sin \phi P_{n-1}^0(\sin \phi) - (n-1) P_{n-2}^0(\sin \phi)]$$

$$P_n^n(\sin \phi) = (2n-1) \cos \phi P_{n-1}^{n-1}(\sin \phi), \quad m \neq 0, m < n$$

$$P_n^m(\sin \phi) = P_{n-2}^m(\sin \phi) + (2n-1) \cos \phi P_{n-1}^{m-1}(\sin \phi), \quad m \neq 0, m = n$$

where the first few associated Legendre functions are given by

$$P_0^0(\sin \phi) = 1, \quad P_1^0(\sin \phi) = \sin \phi, \quad P_1^1(\sin \phi) = \cos \phi$$

and $P_i^j = 0$ for $j > i$.

The trigonometric arguments are determined from expansions given by

Orbital Mechanics with MATLAB

$$\sin m\lambda = 2 \cos \lambda \sin(m-1)\lambda - \sin(m-2)\lambda$$

$$\cos m\lambda = 2 \cos \lambda \cos(m-1)\lambda - \cos(m-2)\lambda$$

$$m \tan \phi = (m-1) \tan \phi + \tan \phi$$

This next function evaluates the ECI gravitation acceleration vector of a user-defined n degree by m order gravity model. The degree and order values are passed to this routine via a `global` statement in the MATLAB variables `lgrav` and `mgrav`, respectively.

The syntax of this MATLAB function is

```
function agrav = gravity (t, y)

% first order equations of orbital motion

% N degree and M order
% gravitational acceleration

% input

% t = simulation time (seconds)
% y = state vector

% output

% agrav = ECI gravitational acceleration
%         vector (km/sec/sec)
```

gravity_2000.m – n by m gravity model – EME2000 coordinates

This MATLAB function evaluates an n by m gravity model data file and evaluates the Earth-centered-inertial (ECI) cartesian acceleration vector. This algorithm assumes that the spacecraft position vector input to this routine is referenced to the Earth mean equator and equinox of J2000 (EME2000) coordinate frame.

The true-of-date spacecraft position vector (subscript *TOD*) required by this function is computed according to

$$\mathbf{r}_{TOD} = [PN] \mathbf{r}_{EME2000}$$

where $[PN]$ is the combined precession-nutation matrix.

The true-of-date gravity vector is converted to the EME2000 system for use in the equations of motion using the transpose of the combined precession-nutation matrix as follows

$$\mathbf{a}_{EME2000} = [PN]^T \mathbf{a}_{TOD}$$

The syntax of this MATLAB function is

Orbital Mechanics with MATLAB

```
function agrav = gravity_2000 (t, y)

% N degree and M order gravitational acceleration
% (EME2000 input and output)

% input

% t = simulation time (seconds)
% y = state vector

% output

% agrav = ECI gravitational acceleration
%         vector (km/sec/sec)
```

ceqm1.m – first-order equations of orbital motion with non-spherical gravity, aerodynamic drag, solar radiation pressure, and sun and moon perturbations

This MATLAB function evaluates the first-order form of the orbital equations of motion of an Earth satellite. This algorithm includes the non-spherical gravity of the Earth, aerodynamic drag, solar radiation pressure, and sun and moon perturbations. The aerodynamic drag calculations use a U. S. Standard 1976 atmosphere model.

The syntax of this MATLAB function is

```
function ydot = ceqm1 (t, y)

% first order form of Cowell's
% equations of orbital motion

% includes perturbations due to:

%   non-spherical earth gravity
%   aerodynamic drag (us 76 model)
%   solar radiation pressure
%   sun and moon

% input

% t = current simulation time
% y = current eci state vector

% output

% ydot = eci acceleration vector
```

ceqm2.m – first-order heliocentric equations of orbital motion

This MATLAB function evaluates the first-order form of the heliocentric equations of motion. It includes the point-mass gravity perturbations due to Venus, Earth, Mars, Jupiter and Saturn. The fundamental coordinate system used in this algorithm is the true-of-date ecliptic and equinox.

Orbital Mechanics with MATLAB

The syntax of this MATLAB function is

```
function ydot = ceqm2 (time, y)

% first order form of the heliocentric
% equations of motion

% includes point mass gravity
% perturbations of Venus, Earth,
% Mars, Jupiter and Saturn

% input

% time = current simulation time
% y     = current state vector

% output

% ydot = equations of motion
```

The second-order heliocentric equations of motion of a satellite or celestial body subject to the *point-mass* gravitational attraction of the Sun and planets are given by

$$\ddot{\mathbf{r}} = \frac{d^2 \mathbf{r}}{dt^2}(\mathbf{r}, t) = -\mu_s \frac{\mathbf{r}_{s-b}}{|\mathbf{r}_{s-b}|^3} - \sum_{i=1}^9 \mu_{p_i} \left(\frac{\mathbf{r}_{(p-b)_i}}{|\mathbf{r}_{(p-b)_i}|^3} + \frac{\mathbf{r}_{p_i}}{|\mathbf{r}_{p_i}|^3} \right)$$

where

μ_s = gravitational constant of the Sun

μ_{p_i} = gravitational constant of planet i

\mathbf{r}_p = position vector from the Sun to planet

\mathbf{r}_{s-b} = position vector from the Sun to the body

\mathbf{r}_{p-b} = position vector from the planet to the body

These position vectors are related according to

$$\mathbf{r}_{s-b} = \mathbf{r}_p + \mathbf{r}_{p-b}$$

ceqm2_jpl.m – first-order heliocentric equations of orbital motion – JPL ephemeris

This MATLAB function evaluates the first-order form of the heliocentric equations of motion. It includes the point-mass gravity perturbations due to Venus, Earth, Mars, Jupiter, Saturn and Uranus. The ephemeris of each planet is computed using a JPL binary ephemeris, and the fundamental coordinate system is the Earth mean equator and equinox of J2000 (EME2000).

The syntax of this MATLAB function is

Orbital Mechanics with MATLAB

```

function ydot = ceqm2_jpl (t, y)

% first order form of the heliocentric equations of motion
% in the EME2000 coordinate system

% includes planetary point-mass perturbations due to Mercury,
% Venus, Earth, Mars, Jupiter, Saturn and Uranus

% JPL ephemeris and Battin's f(q) formulation

% input

% time = elapsed time since jdate0 (seconds)
% y     = current state vector (kilometers and kilometer/second)

% output

% ydot = equations of motion

```

The general vector equation for secondary body perturbations such as the Moon or planets is given by

$$\delta \mathbf{q} = - \sum_{j=1}^n \mu_j \left[\frac{\mathbf{d}_j}{d_j^3} + \frac{\mathbf{s}_j}{s_j^3} \right]$$

In this equation, \mathbf{s}_j is the vector from the primary body to the secondary body j , μ_j is the gravitational constant of the secondary body and $\mathbf{d}_j = \mathbf{r} - \mathbf{s}_j$, where \mathbf{r} is the position vector of the spacecraft relative to the primary body.

To avoid numerical problems, use is made of Battin's $F(q)$ function given by

$$F(q_k) = q_k \left[\frac{3 + 3q_k + q_k^2}{1 + (\sqrt{1 + q_k})^3} \right]$$

where

$$q_k = \frac{\mathbf{r}^T (\mathbf{r} - 2\mathbf{s}_k)}{\mathbf{s}_k^T \mathbf{s}_k}$$

The $\delta \mathbf{q}$ term can now be expressed as

$$\delta \mathbf{q} = - \sum_{k=1}^n \frac{\mu_k}{d_k^3} \left[\mathbf{r} + F(q_k) \mathbf{s}_k \right]$$

meeeqm.m – modified equinoctial equations of motion

This MATLAB function evaluates the modified equinoctial equations of motion.

The syntax for this MATLAB function is

```
function ydot = meeeqm(t, y)  
  
% modified equinoctial equations of motion  
  
% input  
  
% y(1) = semilatus rectum of orbit (kilometers)  
% y(2) = f equinoctial element  
% y(3) = g equinoctial element  
% y(4) = h equinoctial element  
% y(5) = k equinoctial element  
% y(6) = true longitude (radians)  
  
% output  
  
% ydot = first time derivatives of  
%       modified equinoctial elements
```

The perturbation vector used in these calculations is computed in a MATLAB function called `ceqm4.m`. The syntax of this function is as follows:

```
function ywrk = ceqm4 (t, r, v)  
  
% eci perturbation vector  
  
% support function for cowell4.m  
  
% includes perturbations due to:  
  
%   non-spherical earth gravity  
%   aerodynamic drag (us 76 model)  
%   solar radiation pressure  
%   sun and moon  
  
% input  
  
% t = current simulation time  
% r = current eci position vector  
% v = current eci velocity vector  
  
% output  
  
% ywrk = eci perturbation vector
```

The system of first-order modified equinoctial equations of orbital motion are given by

$$\dot{p} = \frac{dp}{dt} = \frac{2p}{w} \sqrt{\frac{p}{\mu}} \Delta_t$$

$$\dot{f} = \frac{df}{dt} = \sqrt{\frac{p}{\mu}} \left[\Delta_r \sin L + [(w+1) \cos L + f] \frac{\Delta_t}{w} - (h \sin L - k \cos L) \frac{g \Delta_n}{w} \right]$$

$$\dot{g} = \frac{dg}{dt} = \sqrt{\frac{p}{\mu}} \left[-\Delta_r \cos L + [(w+1) \sin L + g] \frac{\Delta_t}{w} + (h \sin L - k \cos L) \frac{g \Delta_n}{w} \right]$$

$$\dot{h} = \frac{dh}{dt} = \sqrt{\frac{p}{\mu}} \frac{s^2 \Delta_n}{2w} \cos L$$

$$\dot{k} = \frac{dk}{dt} = \sqrt{\frac{p}{\mu}} \frac{s^2 \Delta_n}{2w} \sin L$$

$$\dot{L} = \frac{dL}{dt} = \sqrt{\mu p} \left(\frac{w}{p} \right)^2 + \frac{1}{w} \sqrt{\frac{p}{\mu}} (h \sin L - k \cos L) \Delta_n$$

where $\Delta_r, \Delta_t, \Delta_n$ are *non-two-body* perturbations in the radial, tangential and normal directions, respectively. The radial direction is along the geocentric radius vector of the spacecraft measured positive in a direction away from the geocenter, the tangential direction is perpendicular to this radius vector measured positive in the direction of orbital motion, and the normal direction is positive along the angular momentum vector of the spacecraft's orbit.

In vector form the equations of motion can be expressed as follows:

$$\dot{\mathbf{y}} = \frac{d\mathbf{y}}{dt} = \mathbf{A}(\mathbf{y}) \mathbf{P} + \mathbf{b}$$

where

$$\mathbf{A} = \begin{pmatrix} 0 & \frac{2p}{q} \sqrt{\frac{p}{\mu}} & 0 \\ \sqrt{\frac{p}{\mu}} \sin L & \sqrt{\frac{p}{\mu}} \frac{1}{q} \{(q+1) \cos L + f\} & -\sqrt{\frac{p}{\mu}} \frac{g}{q} \{h \sin L - k \cos L\} \\ -\sqrt{\frac{p}{\mu}} \cos L & \sqrt{\frac{p}{\mu}} \{(q+1) \sin L + g\} & \sqrt{\frac{p}{\mu}} \frac{f}{q} \{h \sin L - k \cos L\} \\ 0 & 0 & \sqrt{\frac{p}{\mu}} \frac{s^2 \cos L}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}} \frac{s^2 \sin L}{2q} \\ 0 & 0 & \sqrt{\frac{p}{\mu}} \{h \sin L - k \cos L\} \end{pmatrix}$$

and

$$\mathbf{b}^T = \left\{ 0 \ 0 \ 0 \ 0 \ 0 \ \sqrt{\mu p} \left(\frac{q}{p} \right)^2 \right\}$$

The total *non-two-body* acceleration vector is given by

$$\Delta = \Delta_r \hat{\mathbf{i}}_r + \Delta_t \hat{\mathbf{i}}_t + \Delta_n \hat{\mathbf{i}}_n$$

where $\hat{\mathbf{i}}_r$, $\hat{\mathbf{i}}_t$ and $\hat{\mathbf{i}}_n$ are unit vectors in the radial, tangential and normal directions computed from the ECI position \mathbf{r} and velocity vectors \mathbf{v} according to

$$\hat{\mathbf{i}}_r = \frac{\mathbf{r}}{\|\mathbf{r}\|}$$

$$\hat{\mathbf{i}}_n = \frac{\mathbf{r} \times \mathbf{v}}{\|\mathbf{r} \times \mathbf{v}\|}$$

$$\hat{\mathbf{i}}_t = \hat{\mathbf{i}}_n \times \hat{\mathbf{i}}_r = \frac{(\mathbf{r} \times \mathbf{v}) \times \mathbf{r}}{\|\mathbf{r} \times \mathbf{v}\| \|\mathbf{r}\|}$$

For *unperturbed* two-body motion, $\Delta = 0$ and the first five equations of motion are simply $\dot{p} = \dot{f} = \dot{g} = \dot{h} = \dot{k} = 0$. Therefore, for two-body motion these modified equinoctial orbital elements are constant.

Non-spherical Earth Gravity

The non-spherical gravitational acceleration vector is

$$\delta \mathbf{g} = \delta g_N \hat{\mathbf{i}}_N - \delta g_r \hat{\mathbf{i}}_r$$

where

$$\hat{\mathbf{i}}_N = \frac{\hat{\mathbf{e}}_N - (\hat{\mathbf{e}}_N^T \hat{\mathbf{i}}_r) \hat{\mathbf{i}}_r}{\|\hat{\mathbf{e}}_N - (\hat{\mathbf{e}}_N^T \hat{\mathbf{i}}_r) \hat{\mathbf{i}}_r\|}$$

and

$$\hat{\mathbf{e}}_N = (0 \ 0 \ 1)^T$$

In these equations the north direction component is indicated by subscript N and the radial direction component is subscript r .

The orbital perturbations due to the *zonal* gravity effects of J_2, J_3, J_4 are as follows:

$$\delta g_N = -\frac{\mu \cos \phi}{r^2} \sum_{k=2}^4 \left(\frac{R_e}{r} \right)^k P_k' J_k$$

$$\delta g_r = -\frac{\mu}{r^2} \sum_{k=2}^4 (k+1) \left(\frac{R_e}{r} \right)^k P_k J_k$$

where

- μ = gravitational constant
- r = geocentric distance of the spacecraft
- R_e = equatorial radius of the Earth
- ϕ = geocentric latitude
- J_k = zonal gravity coefficient
- P_k = k^{th} order Legendre polynomial

For a zonal only Earth gravity model, the east component is identically zero.

Therefore, the zonal gravity perturbation contribution is

$$\Delta_g = \mathbf{Q}^T \delta \mathbf{g}$$

where $\mathbf{Q} = [\hat{\mathbf{i}}_r \ \hat{\mathbf{i}}_t \ \hat{\mathbf{i}}_n]$.

Secondary Body Perturbations

The general vector equation for secondary body perturbations such as the Moon or planets is given by

$$\delta \mathbf{q} = - \sum_{j=1}^n \mu_j \left[\frac{\mathbf{d}_j}{d_j^3} + \frac{\mathbf{s}_j}{s_j^3} \right]$$

In this equation, \mathbf{s}_j is the vector from the primary body to the secondary body j , μ_j is the gravitational constant of the secondary body and $\mathbf{d}_j = \mathbf{r} - \mathbf{s}_j$, where \mathbf{r} is the position vector of the spacecraft relative to the primary body.

To avoid numerical problems, use is made of Battin's $F(q)$ function given by

$$F(q_k) = q_k \left[\frac{3 + 3q_k + q_k^2}{1 + (\sqrt{1 + q_k})^3} \right]$$

where

$$q_k = \frac{\mathbf{r}^T (\mathbf{r} - 2\mathbf{s}_k)}{\mathbf{s}_k^T \mathbf{s}_k}$$

The $\delta \mathbf{q}$ term can now be expressed as

$$\delta \mathbf{q} = - \sum_{k=1}^n \frac{\mu_k}{d_k^3} \left[\mathbf{r} + F(q_k) \mathbf{s}_k \right]$$

Finally, the perturbation due to secondary bodies in the modified equinoctial coordinate frame is given by

$$\Delta_q = \mathbf{Q}^T \delta \mathbf{q}$$

where $\mathbf{Q} = \left[\hat{\mathbf{i}}_r \quad \hat{\mathbf{i}}_t \quad \hat{\mathbf{i}}_n \right]$.

Propulsive Thrust

The acceleration due to propulsive thrust can be expressed as

$$\Delta_T = \frac{T(t)}{m(t)} \hat{\mathbf{u}}(t)$$

where T is the thrust, m is the spacecraft mass and $\hat{\mathbf{u}} = [u_r \quad u_t \quad u_n]$ is the unit pointing thrust vector expressed in the spacecraft-centered radial-tangential-normal coordinate system. The

components of the unit thrust vector can also be defined in terms of the in-plane pitch angle θ and the out-of-plane yaw angle ψ as follows:

$$\begin{aligned} u_r &= -\sin \theta \\ u_t &= \cos \theta \cos \psi \\ u_n &= -\cos \theta \sin \psi \end{aligned}$$

Finally, the pitch and yaw angles can be determined from the components of the unit thrust vector according to

$$\begin{aligned} \theta &= \sin^{-1}(-u_r) \\ \psi &= \tan^{-1}(-u_n, u_t) \end{aligned}$$

The pitch angle is positive above the “local horizontal” and the yaw angle is positive in the direction of the angular momentum vector.

drag1.m – aerodynamic drag ECI acceleration vector – U. S. Standard 1976 atmosphere

This MATLAB function computes the ECI acceleration vector of a satellite due to aerodynamic drag. The atmospheric density in this algorithm is determined using a U. S. Standard 1976 atmosphere model. The syntax of this function is as follows:

The syntax of this MATLAB function is

```
function adrag = drag(sv)

% acceleration due to atmospheric drag

% us standard 1976 atmosphere model

% input

% sv = ECI state vector

% output

% adrag = ECI drag acceleration vector
%           (kilometers/second/second)
```

The acceleration experienced by the satellite due to atmospheric drag is computed using the following vector expression:

$$\mathbf{a}_d(\mathbf{r}, \mathbf{v}, t) = -\frac{1}{2} \rho(\mathbf{r}, t) |\mathbf{v}_r| \mathbf{v}_r \frac{C_d A}{m}$$

where

Orbital Mechanics with MATLAB

- μ = gravitational constant of the Earth
- \mathbf{v}_r = satellite velocity vector relative to the atmosphere
- ρ = atmospheric density
- C_d = drag coefficient of the satellite
- A = reference area of the satellite
- m = mass of the satellite

This algorithm uses constant values for the mass, drag coefficient and the reference area of the satellite. Please note that the reference area is measured perpendicular to the relative velocity vector.

The aerodynamic drag algorithm assumes that the atmosphere rotates at the same angular speed as the Earth. With this assumption the relative velocity vector is given by

$$\mathbf{v}_r = \mathbf{v} - \mathbf{w} \times \mathbf{r}$$

where $\vec{\omega}$ is the inertial rotation vector of the Earth. The angular velocity vector of the Earth is

$$\mathbf{w} = \omega_e \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

where $\omega_e = 7.292115486\text{E-}5$ radians per second (WGS 84 value).

The cross product expansion of the previous equation gives the three components of the relative velocity vector as follows:

$$\mathbf{v}_r = \begin{bmatrix} v_x + \omega_e r_y \\ v_y - \omega_e r_x \\ v_z \end{bmatrix}$$

drag2.m – aerodynamic drag ECI acceleration vector – Jacchia 1970 atmosphere

This MATLAB function computes the ECI acceleration vector of a satellite due to aerodynamic drag. The atmospheric density used in this algorithm is determined using a 1970 Jacchia atmosphere model.

The syntax of this MATLAB function is

```
function drag2(ut, sv)  
  
% aerodynamic drag acceleration vector  
  
% jacchia 1970 atmosphere model
```

Orbital Mechanics with MATLAB

```
% input

% ut = universal time (seconds)
% sv = current state vector

% output (via global)

% adrag = eci drag acceleration vector
```

A technical discussion about this algorithm can be found under the description of the `drag1.m` function given above.

srp.m – solar radiation pressure ECI acceleration vector

This MATLAB function computes the ECI acceleration vector of a satellite due to solar radiation pressure.

The syntax of this MATLAB function is

```
function asrp = srp (t, sv)

% ECI acceleration vector due
% to solar radiation pressure

% input

% t = simulation time
% sv = current state vector

% output

% asrp = eci acceleration vector
```

We can define a *solar radiation constant* for any satellite as a function of its size, mass and surface reflective properties according to the equation:

$$C_{srp} = \gamma P_s a^2 \frac{A}{m}$$

where

γ = reflectivity constant

P_s = solar radiation constant

a = astronomical unit

A = surface area normal to the incident radiation

m = mass of the satellite

The reflectivity constant is a dimensionless number between 0 and 2. For a perfectly absorbent body $\gamma = 1$, for a perfectly reflective body $\gamma = 2$, and for a translucent body $\gamma < 1$. For example, the reflectivity constant for an aluminum surface is approximately 1.96.

The value of the solar radiation pressure on a perfectly absorbing satellite surface at a distance of one Astronomical Unit from the Sun is

$$P_s = 4.4 \times 10^{-3} \frac{kg}{km - sec^2} = \frac{1358 \text{ watts}}{c \text{ m}^2}$$

where c is the speed of light.

The acceleration vector of the satellite due to solar radiation pressure is given by:

$$\mathbf{a}_{srp} = C_{srp} \frac{\mathbf{r}_{sat-to-Sun}}{r_{sat-to-Sun}^3}$$

where

$$\mathbf{r}_{sat-to-Sun} = \mathbf{r}_{sat} - \mathbf{r}_{Earth-to-Sun}$$

\mathbf{r}_{sat} = geocentric, inertial position vector of the satellite

$\mathbf{r}_{Earth-to-Sun}$ = geocentric, inertial position vector of the Sun

During the integration process, the software must determine if the satellite is in Earth shadow or sunlight. Obviously, there can be no solar radiation perturbation during Earth eclipse of the satellite orbit. The software makes use of a *shadow parameter* to determine eclipse conditions.

This parameter is defined by the following expression:

$$\varphi = -\frac{|\mathbf{r}_{sc} \times \mathbf{r}_{es}|}{|\mathbf{r}_{es}|} \text{sign}(\mathbf{r}_{sc} \bullet \mathbf{r}_{es})$$

where \mathbf{r}_{sc} = is the geocentric, inertial position vector of the satellite and \mathbf{r}_{es} = is the geocentric, inertial position vector of the Sun relative to the satellite.

The *critical* values of the shadow parameter for the penumbra (subscript p) and umbra part (subscript u) of the shadow are given by:

$$\varphi_p = |\mathbf{r}_{sc}| \sin \psi_p$$

$$\varphi_u = |\mathbf{r}_{sc}| \sin \psi_u$$

The penumbra and umbra shadow angles are found from:

$$\psi_p = \eta + \theta_p$$

$$\psi_u = \eta - \theta_u$$

These are the angles between the geocentric anti-Sun vector and the vector to a satellite at the time of shadow entrance or exit.

If we represent the shadow as a cylinder, the shadow angle is given by:

$$\eta = \sin^{-1} \left(\frac{r_e}{r_{sc}} \right)$$

The corresponding penumbra and umbra *cone* angles are as follows:

$$\theta_p = \sin^{-1} \left(\frac{r_s + r_e}{r_{es}} \right)$$

$$\theta_u = \sin^{-1} \left(\frac{r_s - r_e}{r_{es}} \right)$$

where

r_e = radius of the Earth

r_s = radius of the Sun

r_{es} = distance from the Earth to the Sun

If the condition $\varphi_u < \varphi \leq \varphi_p$ is true, the satellite is in the penumbra part of the Earth's shadow, and if the inequality $0 \leq \varphi \leq \varphi_u$ is true, the satellite is in the umbra part of the shadow. If the absolute value of the shadow parameter is larger than the penumbra value, the satellite is in full sunlight. The shadow calculations also assume that the Earth's atmosphere increases the radius of the Earth by two percent.

fpeqms.m – flight path equations of motion

This MATLAB function computes the first-order form of the equations of motion in the flight path coordinate system. These equations include the effects of aerodynamic lift and drag, and gravity due to a spherical Earth.

The syntax of this MATLAB function is

```
function ydot = fpeqms(t, y)
% flight path equations of motion
% includes lift, drag, spherical Earth
% gravity and Earth rotation
% input
```

Orbital Mechanics with MATLAB

```

% state variables

% y(1) = altitude (kilometers)
% y(2) = longitude (radians)
% y(3) = geocentric declination (radians)
% y(4) = relative velocity (kilometers/second)
% y(5) = relative flight path angle (radians)
% y(6) = relative azimuth (radians)

% output

% ydot = equations of motion

% global

% req  = Earth equatorial radius (kilometers)
% mu   = Earth gravitational constant (km**3/sec**2)
% omega = Earth inertial rotation rate (radians/second)
% mass = vehicle mass (kilograms)
% sref = aerodynamic reference area (cubic kilometers)

```

The equations of motion in this system are as follows:

Geocentric radius

$$\dot{r} = \frac{dr}{dt} = v \sin \gamma$$

Longitude

$$\dot{\lambda} = \frac{d\lambda}{dt} = v \frac{\cos \gamma \sin \psi}{r \cos \delta}$$

Geocentric declination

$$\dot{\delta} = \frac{d\delta}{dt} = v \frac{\cos \gamma \cos \psi}{r}$$

Speed

$$\dot{V} = \frac{dV}{dt} = \frac{(T \cos \alpha - D)}{m} - g \sin \gamma + \omega_e^2 r \cos \delta (\sin \gamma \cos \delta - \sin \delta \cos \gamma \cos \psi)$$

Flight path angle

$$\begin{aligned} \dot{\gamma} = \frac{d\gamma}{dt} = & \frac{V}{r} \cos \gamma + \left(\frac{T \sin \alpha + L}{mV} \right) \cos \beta - \frac{g \cos \gamma}{V} \\ & + 2\omega_e \sin \psi \cos \delta + \omega_e^2 \frac{r}{V} \cos \delta (\cos \psi \sin \gamma \sin \delta + \cos \gamma \cos \delta) \end{aligned}$$

Flight azimuth

$$\dot{\psi} = \frac{d\psi}{dt} = \frac{V}{r} \tan \delta \sin \psi \cos \gamma + \left(\frac{T \sin \alpha + L}{mV \cos \gamma} \right) \cos \beta$$

$$+ 2\omega_e (\sin \delta - \cos \psi \cos \delta \tan \gamma) + \frac{r}{V \cos \gamma} \omega_e^2 \sin \psi \cos \delta \sin \delta$$

where

- r = geocentric radius
- V = speed
- γ = flight path angle
- δ = geocentric declination
- λ = longitude (+ east)
- ψ = flight azimuth (+ clockwise from north)
- β = bank angle (+ for a right turn)
- α = angle of attack
- ω_e = Earth inertial rotation rate
- g = Earth acceleration of gravity = μ/r^2
- μ = Earth gravitational constant
- L = aerodynamic lift force = $\frac{1}{2} \rho V^2 C_L S$
- D = aerodynamic drag force = $\frac{1}{2} \rho V^2 C_D S$
- T = propulsive thrust
- m = spacecraft mass
- C_L = lift coefficient (non-dimensional)
- C_D = drag coefficient (non-dimensional)
- S = aerodynamic reference area
- ρ = atmospheric density

This software suite includes a MATLAB script named `demo_fpeqm` which demonstrates how to interact with this function. This example flies an STS maximum cross range re-entry trajectory using angle-of-attack and bank information extracted from a trajectory optimization program.

The following is the program output created by this script along with several graphic displays of flight parameters.

Orbital Mechanics with MATLAB

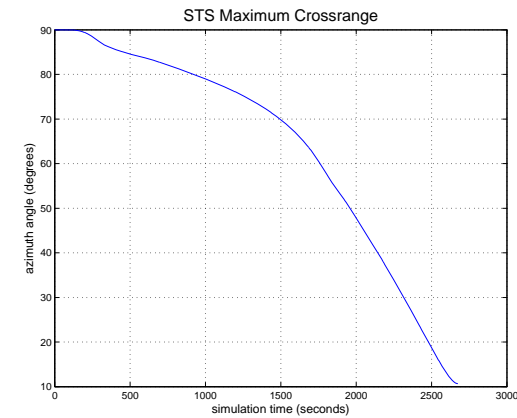
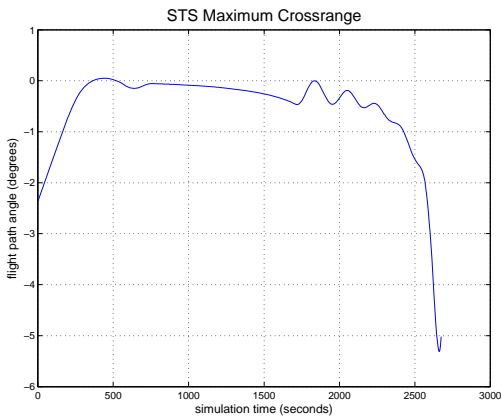
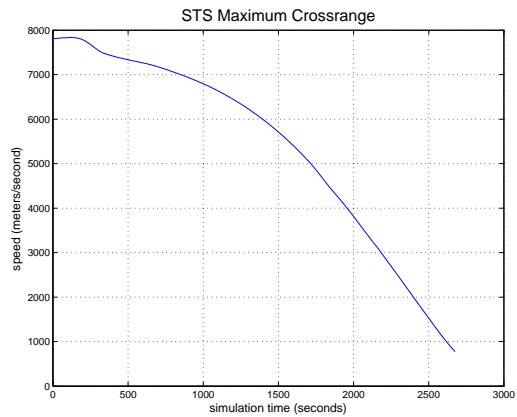
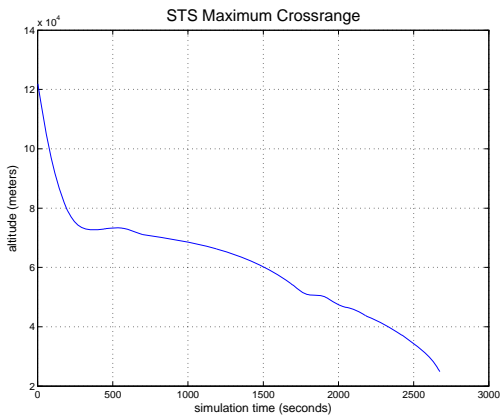
program demo_fpeqm

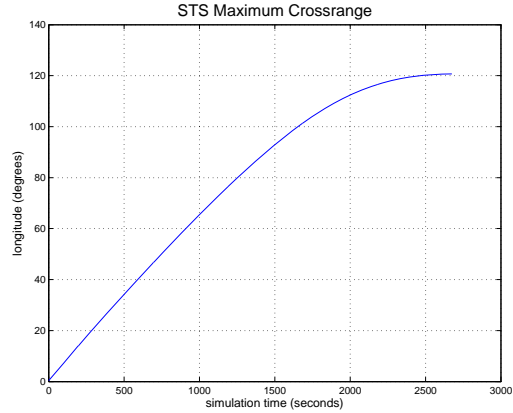
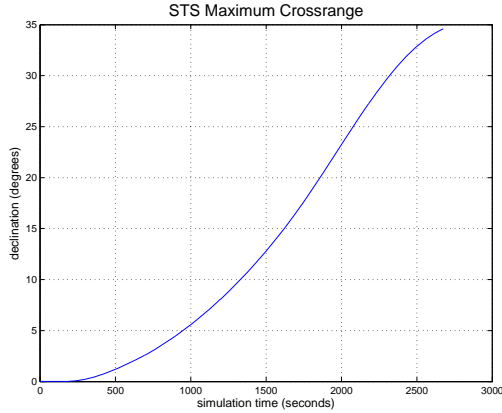
initial flight path coordinates

altitude	121920.0000	meters
velocity	7802.8800	meters/second
declination	0.0000	degrees
longitude	0.0000	degrees
azimuth	90.0000	degrees
flight path angle	-2.3978	degrees

final flight path coordinates

altitude	24518.4877	meters
velocity	770.5555	meters/second
declination	34.5880	degrees
longitude	120.7278	degrees
azimuth	10.6927	degrees
flight path angle	-5.0200	degrees





ANALYTIC ORBIT PROPAGATION ALGORITHMS

This section describes three MATLAB algorithms that can be used to analytically propagate J_2 perturbed satellite orbits.

aorbit.m – method of Markley and Jeletic

This MATLAB function is based on the algorithm described in “Fast Orbit Propagator for Graphical Display”, F. Landis Markley and James F. Jeletic, *AIAA Journal of Guidance and Control*, March-April 1991. This algorithm works with osculating orbital elements for both input and output. The actual propagation is performed internally with mean orbital elements.

This algorithm begins by performing the following initialization at the initial time t_0 :

$$g_1(t_0) = \frac{1 + e \cos \theta_0}{(1 - e^2)^3}$$

$$g_2(t_0) = 1 - 3 \sin^2 i \sin^2(\omega_0 + \theta_0)$$

$$g(t_0) = g_1(t_0) g_2(t_0)$$

The mean semimajor axis used during the actual orbit propagation is defined by

$$\hat{a} = \frac{1}{2} \left(a + \sqrt{a^2 - 4J_2 r_{eq}^2 g(t_0)} \right)$$

The mean motion, mean semi-parameter and perturbation rates used during the orbit propagation are given by

$$\tilde{n} = \sqrt{\frac{\mu}{\hat{a}^3}}$$

$$\hat{p} = \hat{a}(1 - e^2)$$

$$\dot{\omega} = \frac{3}{2} J_2 \tilde{n} \left(\frac{r_{eq}}{\hat{p}} \right)^2 \left(2 - \frac{5}{2} \sin^2 i \right)$$

$$\dot{\Omega} = -\frac{3}{2} J_2 \tilde{n} \left(\frac{r_{eq}}{\hat{p}} \right)^2 \cos i$$

The orbit is propagated to any time t with this next set of equations

$$\omega = \omega_0 + \dot{\omega}(t - t_0)$$

$$\Omega = \Omega_0 + \dot{\Omega}(t - t_0)$$

$$M = M_0 + \tilde{n}(t - t_0)$$

where ω_0, Ω_0 and M_0 are the values of argument of perigee, right ascension of the ascending node and mean anomaly at the initial time t_0 . The algorithm then solves Kepler's equation for the current true anomaly and computes the following "g" constants for this true anomaly:

$$g_1(t) = \frac{1 + e \cos \theta}{(1 - e^2)^3}$$

$$g_2(t) = 1 - 3 \sin^2 i \sin^2(\omega + \theta)$$

$$g(t) = g_1(t) g_2(t)$$

Finally, the osculating semimajor axis at the current true anomaly is

$$a = \hat{a} + \frac{J_2 r_{eq} g}{\hat{a}}$$

where r_{eq} is the equatorial radius of the Earth and J_2 is the second zonal gravity coefficient.

The first time this function is called the `iniz` flag should be set to 1.

The syntax of this MATLAB function is

```
function [oev2, r, v] = aorbit (iniz, t, oev1)
% analytic orbit propagation
```

Orbital Mechanics with MATLAB

```
% method of Markley and Jeletic

% input

% iniz = initialization flag (1 = initialize, 0 = bypass)
% t     = elapsed simulation time (seconds)

% initial osculating orbital elements

% oev1(1) = semimajor axis (kilometers)
% oev1(2) = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oev1(3) = orbital inclination (radians)
%           (0 <= oev1(3) <= pi)
% oev1(4) = argument of perigee (radians)
%           (0 <= oev1(4) <= 2 pi)
% oev1(5) = right ascension of ascending node (radians)
%           (0 <= oev1(5) <= 2 pi)
% oev1(6) = mean anomaly (radians)
%           (0 <= oev1(6) <= 2 pi)
% oev1(7) = true anomaly (radians)
%           (0 <= oev1(7) <= 2 pi)

% output

% final osculating orbital elements and state vector at time = t

% oev2(1) = semimajor axis (kilometers)
% oev2(2) = orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oev2(3) = orbital inclination (radians)
%           (0 <= oev2(3) <= pi)
% oev2(4) = updated argument of perigee (radians)
%           (0 <= oev2(4) <= 2 pi)
% oev2(5) = updated right ascension of ascending node (radians)
%           (0 <= oev2(5) <= 2 pi)
% oev2(6) = updated mean anomaly (radians)
%           (0 <= oev2(6) <= 2 pi)
% oev2(7) = updated true anomaly (radians)
%           (0 <= oev2(6) <= 2 pi)
% r       = eci position vector (kilometers)
% v       = eci velocity vector (kilometers/second)
```

kozai1.m – Kozai’s method; ECI version

According to Yoshihide Kozai’s method, “The Motion of a Close Earth Satellite”, *The Astronomical Journal*, **64**, No. 1274, pp. 367-377, the time evolution of the mean orbital elements due to first-order secular perturbations of the gravity harmonic J_2 is as follows:

Orbital Mechanics with MATLAB

$$M(t) = M_0 + \tilde{n}(t - t_0)$$

$$\Omega(t) = \Omega_0 + \dot{\Omega}(t - t_0)$$

$$\omega(t) = \omega_0 + \dot{\omega}(t - t_0)$$

where M_0 is the mean anomaly, Ω_0 is the right ascension of the ascending node (RAAN) and ω_0 is the argument of perigee, all at the initial time t_0 . In the first expression \tilde{n} is called the *perturbed mean motion* and is equal to the time rate of change of mean anomaly.

The perturbed mean motion can be calculated from:

$$\tilde{n} = \frac{dM}{dt} = n \left\{ 1 + \frac{3}{2} J_2 \left(\frac{r_{eq}}{p} \right)^2 \sqrt{1 - e^2} \left(1 - \frac{3}{2} \sin^2 i \right) \right\}$$

where

J_2 = Earth oblateness gravity term

r_{eq} = equatorial radius of the Earth

a = semimajor axis

e = orbital eccentricity

i = orbital inclination

n = unperturbed or Keplerian mean motion = $\sqrt{\mu/a^3}$

$p = a(1 - e^2)$ = semiparameter

The time rate of change of RAAN is determined from

$$\dot{\Omega} = \frac{d\Omega}{dt} = -\frac{3}{2} J_2 \tilde{n} \left(\frac{r_{eq}}{p} \right)^2 \cos i$$

The secular perturbation of the argument of perigee is given by:

$$\dot{\omega} = \frac{d\omega}{dt} = \frac{3}{2} J_2 \tilde{n} \left(\frac{r_{eq}}{p} \right)^2 \left(2 - \frac{5}{2} \sin^2 i \right)$$

The first time this function is called the `iniz` flag should be set to 1. During subsequent calls to this function, this flag should be set to 0.

The inputs, outputs and calling syntax for this function are as follows:

```
function [oev2, r, v] = kozail (iniz, t, oev1)
```

Orbital Mechanics with MATLAB

```
% analytic orbit propagation

% Kozai's method - ECI version

% input

% iniz = initialization flag (1 = initialize, 0 = bypass)
% t     = elapsed simulation time (seconds)

% initial orbital elements

% oev1(1) = semimajor axis (kilometers)
% oev1(2) = orbital eccentricity (non-dimensional)
%          (0 <= eccentricity < 1)
% oev1(3) = orbital inclination (radians)
%          (0 <= oev1(3) <= pi)
% oev1(4) = argument of perigee (radians)
%          (0 <= oev1(4) <= 2 pi)
% oev1(5) = right ascension of ascending node (radians)
%          (0 <= oev1(5) <= 2 pi)
% oev1(6) = mean anomaly (radians)
%          (0 <= oev1(6) <= 2 pi)

% output

% final orbital elements and state vector at time = t

% oev2(1) = semimajor axis (kilometers)
% oev2(2) = orbital eccentricity (non-dimensional)
%          (0 <= eccentricity < 1)
% oev2(3) = orbital inclination (radians)
%          (0 <= oev2(3) <= pi)
% oev2(4) = updated argument of perigee (radians)
%          (0 <= oev2(4) <= 2 pi)
% oev2(5) = updated right ascension of ascending node (radians)
%          (0 <= oev2(5) <= 2 pi)
% oev2(6) = updated mean anomaly (radians)
%          (0 <= oev2(6) <= 2 pi)
% oev2(7) = updated true anomaly (radians)
%          (0 <= oev2(6) <= 2 pi)
% r       = eci position vector (kilometers)
% v       = eci velocity vector (kilometers/second)
```

kozai2.m – Kozai’s method; ECF version

This algorithm is identical to `kozai1.m` except that the RAAN is replaced by the east longitude of the ascending node. The state vector calculated by this function is relative to the Earth-centered fixed (ECF) coordinate system. The first time this function is called the `iniz` flag should be set to 1. During subsequent calls to this function, this flag should be set to 0.

The syntax of this MATLAB function is

```
function [oev2, r, v] = kozai2 (iniz, t, oev1)
```

Orbital Mechanics with MATLAB

```
% analytic orbit propagation

% Kozai's method - ECF version

% input

% iniz = initialization flag (1 = initialize, 0 = bypass)
% t     = elapsed simulation time (seconds)

% initial orbital elements

% oev1(1) = semimajor axis (kilometers)
% oev1(2) = orbital eccentricity (non-dimensional)
%          (0 <= eccentricity < 1)
% oev1(3) = orbital inclination (radians)
%          (0 <= oev1(3) <= pi)
% oev1(4) = argument of perigee (radians)
%          (0 <= oev1(4) <= 2 pi)
% oev1(5) = east longitude of ascending node (radians)
%          (0 <= oev1(5) <= 2 pi)
% oev1(6) = mean anomaly (radians)
%          (0 <= oev1(6) <= 2 pi)

% output

% final orbital elements and state vector at time = t

% oev2(1) = semimajor axis (kilometers)
% oev2(2) = orbital eccentricity (non-dimensional)
%          (0 <= eccentricity < 1)
% oev2(3) = orbital inclination (radians)
%          (0 <= oev2(3) <= pi)
% oev2(4) = updated argument of perigee (radians)
%          (0 <= oev2(4) <= 2 pi)
% oev2(5) = updated east longitude of ascending node (radians)
%          (0 <= oev2(5) <= 2 pi)
% oev2(6) = updated mean anomaly (radians)
%          (0 <= oev2(6) <= 2 pi)
% oev2(7) = updated true anomaly (radians)
%          (0 <= oev2(6) <= 2 pi)
% r       = ecf position vector (kilometers)
% v       = ecf velocity vector (kilometers/second)
```

NORAD Two Line Element Sets and the SGP4 algorithm

The NORAD orbit propagators are popular, accurate and easy to use. They are based on the work of Dirk Brouwer, Lane and Cranford, and others. The Two Line Element (TLE) sets required by these propagators are widely distributed on the Internet and maintained by NORAD. Each algorithm is documented in the classic SpaceTrack Report No. 3, "Models for Propagation of NORAD Element Sets" by Felix R. Hoots and Ronald L. Roehrich. This document also contains Fortran source code and test cases for each propagator.

The following is a typical TLE for the NOAA 14 spacecraft:

Orbital Mechanics with MATLAB

NOAA 14

```
1 23455U 94089A 97320.90946019 .00000140 00000-0 10191-3 0 2621
2 23455 99.0090 272.6745 0008546 223.1686 136.8816 14.11711747148495
```

The *mean* orbital elements contained in this data are ECI coordinates with respect to the true equator of date and the mean equinox of date. They do not include the effect of nutation.

The following is a brief description of the data contained in each line of a TLE. Each item must appear in its column field in exactly the format specified.

Line 1

The first line is a twenty four character satellite name. Software that reads a database of TLEs will look for this name in order to find the correct data.

Line 2

Column	Description
01	line number of element data
03-07	satellite number
08	classification (u=unclassified)
10-11	international designator (last two digits of launch year)
12-14	international designator (launch number of the year)
15-17	international designator (piece of the launch)
19-20	epoch year (last two digits of year)
21-32	epoch (day of the year and fractional portion of the day)
34-43	first time derivative of the mean motion
45-52	second time derivative of mean motion (decimal point assumed)
54-61	bstar drag term (decimal point assumed)
63	ephemeris type
65-68	element number
69	checksum (modulo 10) (letters, blanks, periods, plus signs = 0; minus signs = 1)

Line 3

Column	Description
01	line number of element data
03-07	satellite number
09-16	orbital inclination (degrees)
18-25	right ascension of the ascending node (degrees)
27-33	orbital eccentricity (decimal point assumed)
35-42	argument of perigee (degrees)
44-51	mean anomaly (degrees)
53-63	mean motion (orbits per day)
64-68	revolution number at epoch (orbits)
69	checksum (modulo 10)

All other columns are blank or fixed.

A good Internet source for the latest TLEs is <http://celestrak.com>. You can also find a Postscript and PDF version of SpaceTrack Report No. 3 at that location.

Warning: Do not use TLEs with other orbit propagators. They are only compatible with the SGP4, SDP4 and other algorithms developed by NORAD.

Important Note

Since a TLE represents the epoch calendar date with only two digits, there is a potential Y2K problem. To avoid this problem the SGP algorithms in this software suite assume that a TLE data field less than 50 is a calendar date after 2000. The code that “corrects” this problem can be found in the `readtle.m` function and looks like the following:

```
if (iyr < 50)
    iyear = 2000 + iyr;
else
    iyear = 1900 + iyr;
end
```

In this source code `iyr` is the two digit calendar date read by `readtle.m` and `iyear` is the integer calendar year actually used by the software.

readtle.m, sgp4.m and demo_sgp4.m – NORAD SGP4 method

These three MATLAB functions can be used to read a Two Line Element set (TLE) database disk file in ASCII format, select a satellite in the database, and propagate the orbit for a user-specified time interval using a MATLAB version of the NORAD SGP4 algorithm. This algorithm was ported to MATLAB using the Fortran version of SGP4 found in SpaceTrack Report #3.

The format and arguments of the MATLAB function (`readtle`) which reads the TLE database file and finds the user selected satellite is as follows:

```
function fid = readtle(satstr, tlename)

% read NORAD two line element file
% and extract orbital information

% input

% satstr = name of satellite
% tlename = name of TLE data file

% output

% fid = file id
```

Orbital Mechanics with MATLAB

The following is the syntax of the MATLAB function that propagates the satellite orbit. Please note that all information required by this function (other than `tsince`) is passed via `global` statements.

```
function [r, v] = sgp4 (tsince)

% SGP4 orbit propagation

% input

%   tsince = time since initialization (minutes)

% output

%   r = position vector (kilometers)
%   v = velocity vector (km/sec)
```

The following is part of a typical TLE data file.

```
1994007A
1 22978U 94007A   94034.91970139   .00000150   00000-0   00000+0 0   16
2 22978   30.5073 159.4204 0007002 226.1375 134.0418 15.37491022   03
STS 60
1 22977U 94006A   94035.04166667   .00000202   00000-0   58718-5 0   37
2 22977   56.9857 213.7143 0008536 262.8823 261.4244 15.72145451   89
1994005B
1 22976U 94005B   94028.08392211  -.00003826  12096-4   00000+0 0   29
2 22976   51.3409 170.1185 0019041  92.4280 267.4761 16.28728285   05
```

Notice that the first line of each TLE contains the satellite name. For example, the name associated with the second TLE is `STS 60`. This corresponds to the `satstr` input to the `readtle` function. Two example TLE data files (`tle229.tle` and `visual.tle`) are included with this software suite.

This software suite contains a script called `demo_sgp4.m` that demonstrates how to interact with these functions. The following is a typical user interaction with this script.

```
program demo_sgp4

< Earth orbital motion - SGP4 method >

please input the name of the TLE database file
(please include the file name extension)
? tle091.tle

please input the name of the satellite
? mir

please wait, searching TLE data file ...
```

Orbital Mechanics with MATLAB

the epoch of this TLE is

01-Apr-1999
03:43:43

initial calendar date

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? **5,1,1999**

initial universal time

please input the universal time
(0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
? **0,0,0**

please input the simulation period (days)
? **30**

would you like to create and display graphics (y = yes, n = no)
? **y**

please select the item to plot

- <1> semimajor axis
- <2> eccentricity
- <3> orbital inclination
- <4> argument of perigee
- <5> right ascension of the ascending node
- <6> true anomaly
- <7> geodetic perigee altitude
- <8> geodetic apogee altitude
- <9> geodetic altitude

? **3**

please input the graphics step size (minutes)
? **30**

The following is the screen display for this example.
program demo_sgp4

< Earth orbital motion - SGP4 method >

initial calendar date 01-May-1999

initial universal time 00:00:00

initial classical orbital elements

sma (km)	eccentricity	inclination (deg)	argper (deg)
6.7165549789e+003	1.9111596384e-003	5.1651141449e+001	8.6173666118e+001

Orbital Mechanics with MATLAB

```
    raan (deg)    true anomaly (deg)    arglat (deg)    period (min)
2.2441923631e+002  2.1324525627e+002  2.9941892239e+002  9.1301846214e+001
```

```
final calendar date    31-May-1999
```

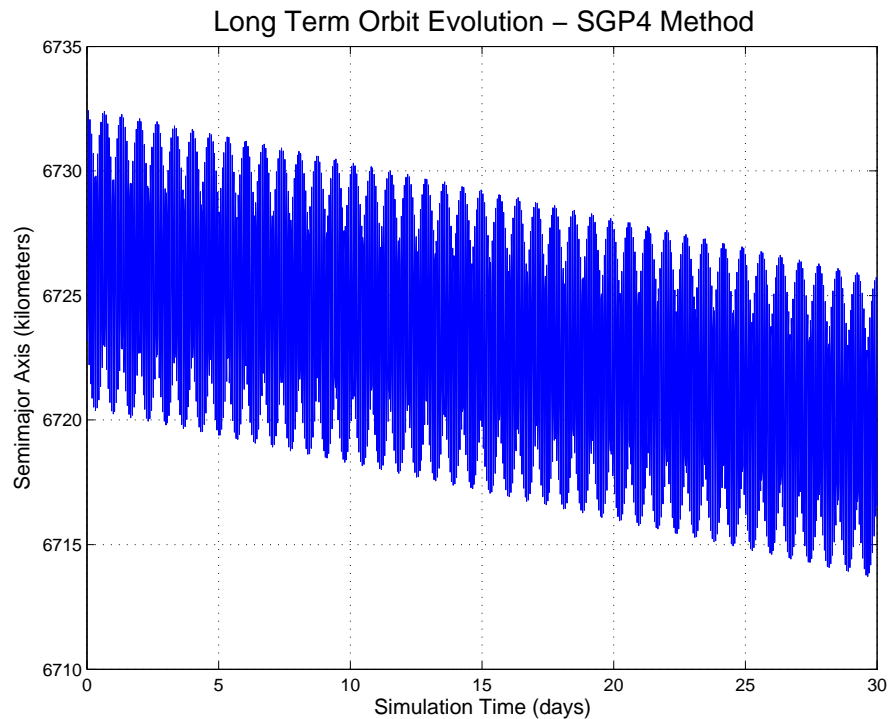
```
final universal time    00:00:00
```

```
final classical orbital elements
```

```
    sma (km)    eccentricity    inclination (deg)    argper (deg)
6.7165549789e+003  1.9111596384e-003  5.1651141449e+001  8.6173666118e+001
```

```
    raan (deg)    true anomaly (deg)    arglat (deg)    period (min)
2.2441923631e+002  2.1324525627e+002  2.9941892239e+002  9.1301846214e+001
```

The following is a typical graphic output created with the `demo_sgp4` MATLAB script.



The Programmer's Toolbox also contains two MATLAB scripts that can be used to convert classical orbital elements or a state vector to a two line element set. These algorithms use the SGP4 propagator and should not be used to process orbits with periods greater than 225 minutes.

The first script is called `sv2tle.m`. It can be used to convert an ECI position and velocity vector or a set of classical orbital elements provided by the user to an equivalent TLE.

From the state vector provided by the user, the program can determine the companion classical *osculating* orbital elements (subscript *osc*). The initial guess for the NORAD or SGP4 compatible *mean* orbital elements (subscript *sgp*) are set to these values as follows:

Orbital Mechanics with MATLAB

$$n_{sgp} = n_{osc} = \text{mean motion}$$

$$e_{sgp} = e_{osc} = \text{eccentricity}$$

$$i_{sgp} = i_{osc} = \text{inclination}$$

$$\omega_{sgp} = \omega_{osc} = \text{argument of perigee}$$

$$\Omega_{sgp} = \Omega_{osc} = \text{right ascension of ascending node}$$

$$M_{sgp} = M_{osc} = \text{mean anomaly}$$

This script solves the following vector system of nonlinear equations:

$$\mathbf{r}_i - \mathbf{r}_{sgp} = 0$$

$$\mathbf{v}_i - \mathbf{v}_{sgp} = 0$$

Essentially, this numerical method is trying to minimize the difference between the state vector input by the user and the state vector computed by the SGP4 algorithm using current values for the SGP4 orbital elements. Since the two state vectors are not dramatically different, the algorithm converges in a reasonable CPU time.

Within the main script, the line of code that solves the system of nonlinear equations using the built-in `fsolve` MATLAB algorithm is

```
xf = fsolve('tlefunc', x);
```

where `x` is the initial guess array and `xf` is the solution array.

The following is the MATLAB source code for the function which evaluates the system of nonlinear equations.

```
function y = tlefunc(x)  
  
% TLE objective function  
  
% required by sv2tle.m  
  
% input  
  
% x = array of current dependent variables  
  
% output  
  
% y = function value array evaluated at x  
  
% Orbital Mechanics with Matlab
```

Orbital Mechanics with MATLAB

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

global iflag ri vi

global xmo xnodeo omegao eo xincl

global xno xndt2o xndd6o bstar

y = zeros(6, 1);

% "unload" current orbital elements

xincl = x(1);
omegao = mod(x(2), 2.0 * pi);
xnodeo = mod(x(3), 2.0 * pi);
eo = x(4);
xmo = mod(x(5), 2.0 * pi);
xno = x(6);

% zero "unknown" elements

bstar = 0;

xndt2o = 0;

xndd6o = 0;

% call sgp4 algorithm and compute state vector

iflag = 1;

tsince = 0;

[rtmp, vtmp] = sgp4(tsince);

% define system of nonlinear equations

y(1) = ri(1) - rtmp(1);
y(2) = ri(2) - rtmp(2);
y(3) = ri(3) - rtmp(3);

y(4) = vi(1) - vtmp(1);
y(5) = vi(2) - vtmp(2);
y(6) = vi(3) - vtmp(3);
```

Since we do not know the value of the first time derivative of the mean motion, second time derivative of mean motion and the *bstar* drag term, these SGP4 orbital elements are set to zero for all computations. This information could be computed by processing many state vectors and performing some type of *differential correction* or orbit determination process.

The following is a typical user interaction with this MATLAB script. The user can elect to input either an ECI state vector (position and velocity vector components) or the osculating classical orbital elements.

Orbital Mechanics with MATLAB

convert state vector to TLE

please input the name of this satellite
? **mysat**

TLE calendar date and universal time

please input the calendar date
(1 <= month <= 12, 1 <= day <= 31, year = all digits!)
? **10,21,1998**

please input the universal time
(0 <= hours <= 24, 0 <= minutes <= 60, 0 <= seconds <= 60)
? **10,20,38**

user input menu

<1> user input of state vector

<2> user input of orbital elements

? **2**

please input the semimajor axis (kilometers)
(semimajor axis > 0)
? **8000**

please input the orbital eccentricity (non-dimensional)
(0 <= eccentricity < 1)
? **.015**

please input the orbital inclination (degrees)
(0 <= inclination <= 180)
? **28.5**

please input the argument of perigee (degrees)
(0 <= argument of perigee <= 360)
? **100**

please input the right ascension of the ascending node (degrees)
(0 <= raan <= 360)
? **200**

please input the true anomaly (degrees)
(0 <= true anomaly <= 360)
? **45**

The following is the osculating state vector and orbital element data for this example and the TLE created by the program.

osculating state vector

rx (km)	ry (km)	rz (km)	rmag (km)
+7.45643912752328e+003	-1.53143414665499e+003	+2.16602932328762e+003	+7.91425663201181e+003

Orbital Mechanics with MATLAB

```
vx (kps)          vy (kps)          vz (kps)          vmag (kps)
+2.15927484581766e+000 +6.21127434865756e+000 -2.76808218520815e+000 +7.13475128355144e+000
```

osculating orbital elements

```
sma (km)          eccentricity      inclination (deg)  argper (deg)
+8.00000000000000e+003 +1.50000000000000e-002 +2.85000000000000e+001 +1.00000000000000e+002

raan (deg)        true anomaly (deg)  arglat (deg)      period (min)
+2.00000000000000e+002 +4.50000000000000e+001 +1.45000000000000e+002 +1.18684684295007e+002
```

Two Line Element set

MYSAT

```
1 XXXXXU XXXXXXXX 98294.43099537 .00000000 00000-0 00000-0
2 XXXXX 28.4958 200.0244 0139902 98.3657 45.4159 12.14276755
```

The second MATLAB script is named `demo_rv2tle` and is based on the numerical method described on Scott Campbell's Satellite Orbit Determination web site which is located at www.coastalbend.edu/acdem/math/sats/. The MATLAB version was created by porting the C source code provided by Scott. This demonstration script is “hard-wired” with position and velocity vectors of a satellite in a typical low Earth orbit (LEO).

The following is the output created by this script. The epoch for this example is June 2, 2006, at 21:11:30 UTC.

osculating state vector

```
rx (km)          ry (km)          rz (km)          rmag (km)
-5.33976186573000e+003 +5.72143584226500e+003 +9.21276953805000e+002 +7.88014188358758e+003

vx (kps)        vy (kps)        vz (kps)        vmag (kps)
-4.88969089550000e+000 -3.83304653050000e+000 +3.18013811100000e+000 +6.97958459820843e+000
```

osculating orbital elements

```
sma (km)          eccentricity      inclination (deg)  argper (deg)
+7.59945293926128e+003 +1.34343969368849e-001 +2.73468214107603e+001 +2.61496877001562e+002

raan (deg)        true anomaly (deg)  arglat (deg)      period (min)
+1.19866833983555e+002 +1.13247099828464e+002 +1.47439768300260e+001 +1.09883687500392e+002
```

components of the TLE

```
eccentricity      0.135214
mean motion       13.118567 orbits per day
mean anomaly      98.898122 degrees
inclination       27.334750 degrees
argper            261.155709 degrees
raan              119.851953 degrees
```

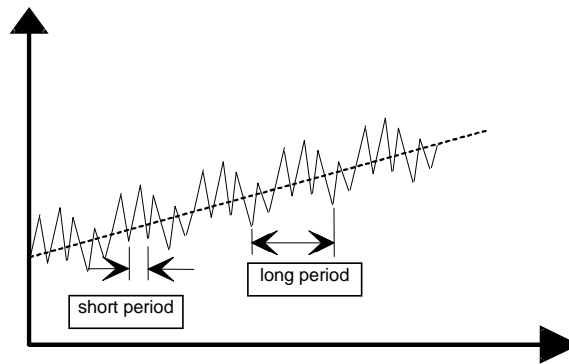
Two Line Elements

```
1 XXXXXU XXXXXXXX 06153.88298611 .00000000 00000-0 00000-0
2 XXXXX 27.3348 119.8520 1352144 261.1557 98.8981 13.11856673
```

As with the previous example, the bstar drag term is not estimated by this algorithm.

MEAN AND OSCULATING ORBITAL ELEMENTS

This section describes two MATLAB functions that can be used to convert osculating classical orbital elements to and from mean orbital elements. These two routines include the effect of J_2 during the coordinate conversion process. The behavior of a typical orbital element consists of short period, medium period, long period and secular components. The following diagram illustrates these effects.



Short, medium and long period perturbations are called “periodics” and each refers to the frequency of the respective perturbation. The short periodics have frequencies that are multiples of the satellite’s orbital period, and long periodics have frequencies which are proportional to the period of the satellite's argument of perigee. The frequency of medium periodics is proportional to the Earth's rotation period relative to a satellite's line of nodes.

Stated another way, the short period variations of the orbital elements are functions of the “fast” varying mean anomaly through trigonometric relations. The long period variations are functions of the “slowly” varying argument of perigee through trigonometric relations.

The classical osculating orbital elements of an artificial satellite are functions of the mean orbital elements, denoted by the subscript m , and the short-period contributions, denoted by the subscript sp , as follows:

Orbital Mechanics with MATLAB

$$a = a_m + a_{sp}(a_m, e_m, i_m, \omega_m, M_m)$$

$$e = e_m + e_{sp}(a_m, e_m, i_m, \omega_m, M_m)$$

$$i = i_m + i_{sp}(a_m, e_m, i_m, \omega_m, M_m)$$

$$\Omega = \Omega_m + \Omega_{sp}(a_m, e_m, i_m, \omega_m, M_m)$$

$$\omega = \omega_m + \omega_{sp}(a_m, e_m, i_m, \omega_m, M_m)$$

$$M = M_m + M_{sp}(a_m, e_m, i_m, \omega_m, M_m)$$

where a is the semimajor axis, e is the orbital eccentricity, i is the inclination, Ω is the right ascension of the ascending node, ω is the argument of perigee, and M is the mean anomaly. The osculating or “kissing” orbital elements define the *two-body* orbit that the satellite would follow if the perturbations were “turned off” at any time during the motion.

From these equations it is apparent that a numerical method must be used to solve for the mean elements in terms of the osculating elements. A typical approach is to set the initial guesses for the mean elements equal to the osculating values, and numerically iterate until the difference between two successive iterations is “small”.

A typical expression for the short-period perturbation of semimajor axis is given by

$$a_{sp}(a_m, e_m, i_m, \omega_m, \theta_m) = J_2 \left(\frac{R^2}{a_m} \right) \left\{ \left(\frac{a_m}{r_m} \right)^3 \left[\left(1 - \frac{3}{2} \sin^2 i_m \right) + \frac{3}{2} \sin^2 i_m \cos 2(\omega_m + \theta_m) \right] - \left(1 - \frac{3}{2} \sin^2 i_m \right) (1 - e_m^2)^{-3/2} \right\}$$

where J_2 is the zonal harmonic coefficient due to oblateness, R is the equatorial radius of the planet, $r_m = P_m / (1 + e_m \cos \theta_m)$ is the mean geocentric radius, and θ_m is the (mean) true anomaly of the satellite.

The calculation of the true anomaly required in the short-period equations involves the solution of Kepler's equation in the “mean” form given by:

$$M_m = E_m - e_m \sin E_m$$

The “mean” true anomaly is then determined from the following three equations:

$$\sin \theta_m = \frac{\sqrt{1 - e_m^2} \sin E_m}{1 - e_m \cos E_m}$$

Orbital Mechanics with MATLAB

$$\cos \theta_m = \frac{\cos E_m - e_m}{1 - e_m \cos E_m}$$

$$\theta_m = \tan^{-1}(\sin \theta_m, \cos \theta_m)$$

In these equations, M is the mean anomaly and E is the eccentric anomaly.

osc2mean.m – convert osculating to mean elements

This function converts a set of six classical osculating orbital elements to their equivalent mean classical orbital elements.

The syntax of this MATLAB function is

```
function oemean = osc2mean (oeosc)

% convert osculating classical orbital elements
% to mean classical orbital elements

% input

% oeosc(1) = osculating semimajor axis (kilometers)
% oeosc(2) = osculating orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oeosc(3) = osculating orbital inclination (radians)
%           (0 <= inclination <= pi)
% oeosc(4) = osculating argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
% oeosc(5) = osculating right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
% oeosc(6) = osculating true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)

% output

% oemean(1) = mean semimajor axis (kilometers)
% oemean(2) = mean orbital eccentricity (non-dimensional)
%           (0 <= eccentricity < 1)
% oemean(3) = mean orbital inclination (radians)
%           (0 <= inclination <= pi)
% oemean(4) = mean argument of perigee (radians)
%           (0 <= argument of perigee <= 2 pi)
% oemean(5) = mean right ascension of ascending node (radians)
%           (0 <= raan <= 2 pi)
% oemean(6) = mean true anomaly (radians)
%           (0 <= true anomaly <= 2 pi)
```

mean2osc.m – convert mean to osculating elements

This function converts a set of six classical mean orbital elements to a set of equivalent osculating classical orbital elements.

The syntax of this MATLAB function is

```
function oeosc = mean2osc (oemean)

% convert mean classical orbital elements to
% osculating classical orbital elements

% input

% oemean(1) = mean semimajor axis (kilometers)
% oemean(2) = mean orbital eccentricity (non-dimensional)
%             (0 <= eccentricity < 1)
% oemean(3) = mean orbital inclination (radians)
%             (0 <= inclination <= pi)
% oemean(4) = mean argument of perigee (radians)
%             (0 <= argument of perigee <= 2 pi)
% oemean(5) = mean right ascension of ascending node (radians)
%             (0 <= raan <= 2 pi)
% oemean(6) = mean true anomaly (radians)
%             (0 <= true anomaly <= 2 pi)

% output

% oeosc(1) = osculating semimajor axis (kilometers)
% oeosc(2) = osculating orbital eccentricity (non-dimensional)
%             (0 <= eccentricity < 1)
% oeosc(3) = osculating orbital inclination (radians)
%             (0 <= inclination <= pi)
% oeosc(4) = osculating argument of perigee (radians)
%             (0 <= argument of perigee <= 2 pi)
% oeosc(5) = osculating right ascension of ascending node (radians)
%             (0 <= raan <= 2 pi)
% oeosc(6) = osculating true anomaly (radians)
%             (0 <= true anomaly <= 2 pi)
```

LAMBERT'S PROBLEM

This section describes a MATLAB function that can be used to solve Lambert's Problem which is also known as the orbital two-point boundary-value problem (TPBVP). Lambert's problem is concerned with the determination of an orbit that passes between two positions within a specified time-of-flight.

The algorithm used in this MATLAB function is based on the method described in "A Procedure for the Solution of Lambert's Orbital Boundary-Value Problem" by R. H. Gooding, *Celestial Mechanics and Dynamical Astronomy* **48**: 145-165, 1990. This iterative solution is valid for elliptic, parabolic and hyperbolic transfer orbits which may be either posigrade or retrograde, and involve one or more revolutions about the central body.

glambert.m – Gooding's solution of Lambert's problem

The syntax of this MATLAB function is

Orbital Mechanics with MATLAB

```
function [vi, vf] = glambert(cbm, sv1, sv2, tof, nrev)

% Gooding's solution of Lambert's problem

% input

% cbm = central body gravitational constant
% sv1 = initial 6-element state vector (position + velocity)
% sv2 = final 6-element state vector (position + velocity)
% tof = time of flight (+ posigrade, - retrograde)
% nrev = number of full revolutions
%       (positive for long period orbit,
%       negative for short period orbit)

% output

% vi = initial velocity vector of the transfer orbit
% vf = final velocity vector of the transfer orbit
```

PREDICTING TRAJECTORY EVENT TIMES

This section describes MATLAB functions that can be used to determine the two-body time-of-flight between important orbital events.

tevent.m – trajectory event times function

This MATLAB function determines the times until the next perigee passage and ascending node crossing. Two-body motion is assumed during the solution process.

The eccentric anomaly at any true anomaly θ can be found from the following three equations:

$$\begin{aligned}\sin E &= \sin \theta \sqrt{1 - e^2} \\ \cos E &= e + \cos \theta \\ E &= \tan^{-1}(\sin E, \cos E)\end{aligned}$$

where the inverse tangent calculation is four quadrant. Since the true, mean and eccentric anomalies at perigee are 0, the mean anomaly increment between perigee and the current location of a satellite is given by

$$\Delta M = -E + e \sin E$$

The time until the next perigee passage is determined from

$$t_{pp} = \frac{\Delta M \tau_k}{2\pi}$$

Orbital Mechanics with MATLAB

where τ_k is the Keplerian orbital period defined by $\tau_k = 2\pi\sqrt{a^3/\mu}$. If t_{pp} is negative, one orbital period should be added to the answer.

The true and eccentric anomalies of the ascending node are determined from

$$\begin{aligned}\theta_{an} &= 2\pi - \omega \\ \sin E_{an} &= \sin \theta_{an} \sqrt{1 - e^2} \\ \cos E_{an} &= e + \cos \theta_{an} \\ E_{an} &= \tan^{-1}(\sin E_{an}, \cos E_{an})\end{aligned}$$

The mean anomaly difference between a current location defined by θ and E is given by this next expression:

$$\Delta M = E_{an} - E - e(\sin E_{an} - \sin E)$$

The time until the next ascending node crossing is determined from

$$t_{an} = \frac{\Delta M \tau_k}{2\pi}$$

If this calculation results in a negative time, simply add one orbital period.

The syntax of this MATLAB function is

```
function [tttp, ttanc] = tevent (sma, ecc, argper, tanom)
% trajectory event times
% input
% sma    = semimajor axis
% ecc    = orbital eccentricity
% argper = argument of perigee (radians)
% tanom  = true anomaly (radians)
% output
% tttp   = time until perigee passage (seconds)
% ttanc  = time until ascending node crossing (seconds)
```

tofl.m – time of flight between two true anomalies

This function determines the flight time between any two true anomalies on an elliptic orbit. The orbital motion is assumed to be two-body or unperturbed.

The time of flight between perigee and another true anomaly on an elliptic orbit is given by:

$$t(\theta) = \frac{\tau}{2\pi} \left[2 \tan^{-1} \left\{ \sqrt{\frac{1-e}{1+e}} \tan \frac{\theta}{2} \right\} - \frac{e\sqrt{1-e^2} \sin \theta}{1+e \cos \theta} \right]$$

where

τ = orbital period
 e = orbital eccentricity
 θ = true anomaly

Therefore, the flight time between any two true anomalies is given by

$$\Delta t = t(\theta_1) - t(\theta_2)$$

The syntax of this function is

```
function dtof = tof1(sma, ecc, tanom1, tanom2)
% time of flight between two true anomalies
% input
% sma    = semimajor axis (kilometers)
% ecc    = eccentricity (non-dimensional)
% tanom1 = initial true anomaly (radians)
% tanom2 = final true anomaly (radians)
% output
% dtof = time-of-flight between tanom1 and tanom2 (seconds)
```

tof2.m – time of flight between two altitudes

This function determines the flight time between any two altitudes on an elliptic orbit. The orbital motion is assumed to be two-body and the central body is assumed spherical.

The time of flight between perigee and any geocentric radius r is given by

$$t = \sqrt{\frac{a}{\mu}} \left\{ a \cos^{-1} \left(\frac{a-r}{ae} \right) - \left[2ar - a^2(1-e^2) - r^2 \right]^{1/2} \right\}$$

where

a = semimajor axis
 r = geocentric radius
 e = orbital eccentricity

Orbital Mechanics with MATLAB

Therefore, the flight time between any two altitudes h_1 and h_2 is given by

$$\Delta t = t(r_1) - t(r_2)$$

where $r_1 = h_1 + r_e$ and $r_2 = h_2 + r_e$ with r_e equal to the radius of the Earth.

The syntax of this MATLAB function is

```
function dtof = tof2(sma, ecc, alt1, alt2)

% time of flight between two altitudes

% input

% sma = semimajor axis (kilometers)
% ecc = eccentricity (non-dimensional)
% alt1 = initial altitude (kilometers)
% alt2 = final altitude (kilometers)

% output

% dtof = time-of-flight between alt1 and alt2 (seconds)
```

ATMOSPHERE MODELS

This section describes three MATLAB functions that implement the U. S. Standard 1976 *static* atmosphere model and the Jacchia 1970 *dynamic* atmosphere model.

atmos76.m – 1976 U.S. Standard Atmosphere – tabular data

This MATLAB function calculates atmospheric density using a tabular form of the 1976 U. S. Standard Atmosphere. The tabular data used by this function is contained in a simple one column ASCII data file named `atmos76.dat`. The data in this ASCII file consists of the density in the units of kilograms per cubic kilometer at 500 meter increments and is valid for geodetic altitudes between 0 and 1000 kilometers.

The syntax of the function that reads the `atmos76.dat` ASCII data file is as follows:

```
function [fid, ad76] = read76

% read U.S. Standard 76 atmosphere data file

% output

% fid = file id
% ad76 = atmospheric density data array
```

Orbital Mechanics with MATLAB

This function must be called before actually calculating the density. The `ad76` data array is passed to the function that evaluates the density using a `global` statement in both functions.

The syntax of the MATLAB function that linearly interpolates this data file and calculates the atmospheric density is as follows:

```
function rho = atmos76 (h)  
  
% U.S. Standard 1976 atmosphere model  
  
% linear interpolation - 0 to 1000 kilometers  
  
% input  
  
% h = altitude (kilometers)  
  
% output  
  
% rho = atmospheric density (kg/km^3)
```

us76.m – 1976 U.S. Standard Atmosphere – analytic algorithm

This MATLAB function calculates atmospheric density using an analytic implementation of the 1976 U. S. Standard Atmosphere. It is based on the classic “U.S. Standard Atmosphere, 1976” document published by NOAA (NOAA-S/T 76-1562).

This function requires initialization the first time it is called. The following statement in the main MATLAB script will accomplish this:

```
iatmos = 1;
```

This variable should also be placed in a `global` statement at the beginning of the main script which calls this function. After the first call, the function will set this value to 0.

The syntax of this MATLAB function is

```
function [atmtmp, atmdns, atmprs, atmsos] = us76 (alt)  
  
% U.S. 1976 standard atmosphere  
  
% input  
  
% alt = altitude (kilometers)  
  
% output  
  
% atmtmp = temperature (degrees Kelvin)  
% atmdns = density (kilograms/cubic meter)  
% atmprs = pressure (newton/square meter)  
% atmsos = speed of sound (meters/second)
```

jatmos70.m – 1970 Jacchia atmosphere model

This MATLAB function computes the properties of the Earth’s atmosphere using a 1970 Jacchia dynamic atmosphere model. This model is described in “New Static Models of the Thermosphere and Exosphere with Empirical Temperature Profiles”, by Luigi G. Jacchia, Smithsonian Institution Astrophysical Observatory Special Report Number 313, and is valid for altitudes between 90 and 2500 kilometers. This algorithm also includes the improved integration or quadrature method described in “An Improvement in the Numerical Integration Procedure Used in the NASA Marshall Engineering Thermosphere Model”, by M. P. Hickey, NASA CR 179389, August 1988.

The following is the syntax of the function that calculates the atmospheric properties.

```
function outdata = jatmos70(indata)

% Jacchia 1970 atmosphere main driver

% input

% indata(1) = geodetic altitude (kilometers)
% indata(2) = geodetic latitude (radians)
% indata(3) = geographic longitude (radians)
% indata(4) = calendar year (all digits)
% indata(5) = calendar month
% indata(6) = calendar day
% indata(7) = utc hours
% indata(8) = utc minutes
% indata(9) = geomagnetic index type
%           (1 = indata(12) is Kp, 2 = indata(12) is Ap)
% indata(10) = solar radio noise flux (jansky)
% indata(11) = 162-day average F10 (jansky)
% indata(12) = geomagnetic activity index

% output

% outdata(1) = exospheric temperature (deg K)
% outdata(2) = temperature at altitude (deg K)
% outdata(3) = N2 number density (per meter-cubed)
% outdata(4) = O2 number density (per meter-cubed)
% outdata(5) = O number density (per meter-cubed)
% outdata(6) = A number density (per meter-cubed)
% outdata(7) = He number density (per meter-cubed)
% outdata(8) = H number density (per meter-cubed)
% outdata(9) = average molecular weight
% outdata(10) = total density (kilogram/meter-cubed)
% outdata(11) = log10(total density)
% outdata(12) = total pressure (pascals)
```

Prior to calling this function, your main script must call a function called `j70iniz`. This function performs initialization required by `jatmos70` and its support functions. The `j70iniz` function should only be called once.

The solar activity data, `indata` array elements 10, 11 and 12, are extracted from solar activity bulletins called *Future Solar Activity Estimates for Use in Prediction of Space Environmental Effects on Spacecraft*. These bulletins contain tables of 5, 50 and 95 percentile solar activity data. Solar activity bulletins and data files are issued monthly by NASA and are available on the Internet at <http://sail.msfc.nasa.gov/nse/solar.html>. This site also contains a PDF version of NASA TM-4759, “Statistical Technique for Intermediate and Long-Range Estimation of 13-Month Smoothed Solar Radio Flux and Geomagnetic Index”.

The following is a typical plot of actual and predicted solar activity included in a solar activity bulletin:

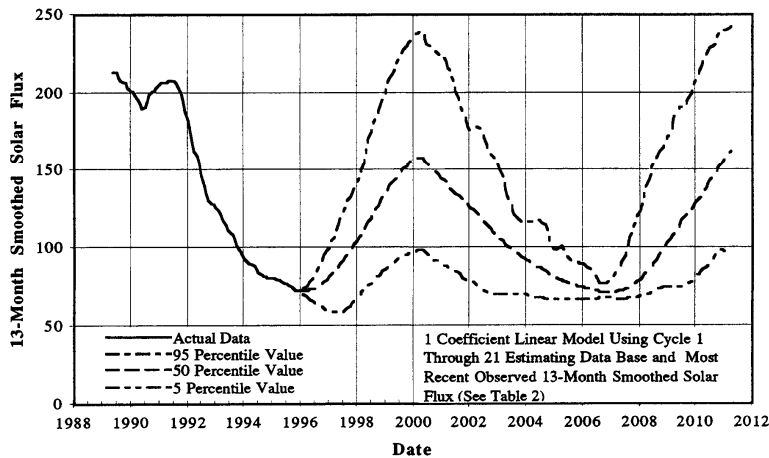


Figure 2. Estimate of 13-Month Smoothed Solar Flux for Balance of Cycle 22, 23, and Beginning of Cycle 24

This collection of software contains a data file called `solar.dat` that contains typical data from one of these bulletins. The data in each of the seven columns of this data file is as follows:

- column 1 = calendar date
- column 2 = solar flux (F10.7) 95% value
- column 3 = solar flux (F10.7) 50% value
- column 4 = solar flux (F10.7) 5% value
- column 5 = geomagnetic index (Ap) 95% value
- column 6 = geomagnetic index (Ap) 50% value
- column 7 = geomagnetic index (Ap) 5% value

The following are the first few lines of information in a typical solar activity data file.

2000.1673	180.5	176.7	171.6	16.5	15.5	14.9
2000.2507	185.6	176.2	166.0	16.6	15.6	14.5
2000.3340	191.3	175.8	162.2	16.8	15.6	14.3

Orbital Mechanics with MATLAB

2000.4173	193.6	174.6	160.0	17.0	15.6	14.0
2000.5007	193.1	172.4	158.6	17.6	15.8	13.5
2000.5840	193.9	170.4	156.9	17.9	15.8	13.0
2000.6673	195.4	168.9	152.4	18.2	15.6	11.9
2000.7507	195.6	168.2	150.2	18.6	15.4	11.1

This software suite includes a MATLAB script named `demo_jatmos70` that demonstrates how to interact with this function. The following is the output created by this script.

```
demo_jatmos70

calendar date      03-Jun-1979

universal time     00:00:00.000

geodetic altitude      303.0417 kilometers
geodetic latitude      -21.0000 degrees
east longitude         36.0000 degrees
solar radio noise flux 60.0700 jansky
162-day average F10    109.5600 jansky
geomagnetic activity index 9.3000

exospheric temperature 698.9464 degrees K
temperature           690.8524 degrees K
total density          6.47341420e-012 kg/m**3
total pressure         2.24837188e-006 pascals
average molecular weight 1.65377812e+001
```

UTILITY FUNCTIONS

This section describes several useful MATLAB functions.

tperiod.m – nodal and anomalistic period function

This MATLAB function computes the nodal and anomalistic periods of a satellite subject to the gravity perturbation due to J_2 .

The syntax of this MATLAB script is:

```
function [tnodal, tanomal] = tperiod (sma, ecc, inc, argper)
```

Orbital Mechanics with MATLAB

```
% orbital periods

% input

% sma    = semimajor axis
% ecc    = orbital eccentricity
% inc    = orbital inclination (radians)
% argper = argument of perigee (radians)

% output

% tnodal = nodal period (seconds)
% tanomal = anomalistic period (seconds)
```

intrsect.m – intersection of two orbits function

This MATLAB function determines the number of intersections and true anomalies at intersection between two elliptic Earth orbits.

```
function [nsol, tanom1, tanom2] = intrsect

% intersection of two orbits

% input (global)

% oev1 = classical orbital elements of first orbit
% oev2 = classical orbital elements of second orbit

% output

% nsol  = number of solutions
% tanom1 = intersection true anomaly array on first orbit (radians)
% tanom2 = intersection true anomaly array on second orbit (radians)
```

readegm.m – read gravity model data file function

This MATLAB function will open and read a gravity model data file. In a typical data file, column one is the model degree index, column two is the order index, and columns three and four are the corresponding gravity coefficients (zonals and tesserals, respectively). Please see the next section for example gravity coefficient data files.

The syntax of this function is

```
function [ccoef, scoef] = readegm(fname)

% read gravity model data file

% input

% fname = name of gravity data file

% output
```

```
% ccoef, scoef = gravity model coefficients
```

DATA FILES

This MATLAB software suite contains several gravity model coefficients and atmospheric data files. The following is a brief description of these ASCII data files and their contents.

egm96.dat – Earth Gravity Model 1996 data file

This ASCII data file is an 18 by 18 version of the Earth Gravity Model 1996 (EGM96) gravity model. The following is a portion of this data file. In this file column one is the model degree index, column two is the order index, and columns three and four are the corresponding *un-normalized* gravity coefficients (zonals and tesserals, respectively).

2	0	-1.08262668355E-003	0.00000000000E+000
3	0	2.53265648533E-006	0.00000000000E+000
4	0	1.61962159137E-006	0.00000000000E+000
5	0	2.27296082869E-007	0.00000000000E+000
6	0	-5.40681239107E-007	0.00000000000E+000
7	0	3.52359908418E-007	0.00000000000E+000
8	0	2.04799466985E-007	0.00000000000E+000
9	0	1.20616967365E-007	0.00000000000E+000
10	0	2.41145438626E-007	0.00000000000E+000
11	0	-2.44402148325E-007	0.00000000000E+000
12	0	1.88626318279E-007	0.00000000000E+000
13	0	2.19788001661E-007	0.00000000000E+000
14	0	-1.30744533118E-007	0.00000000000E+000
15	0	8.23528409456E-009	0.00000000000E+000
16	0	-1.81139265112E-008	0.00000000000E+000
17	0	1.16904733834E-007	0.00000000000E+000
18	0	3.09424678746E-008	0.00000000000E+000

Gravity model coefficients are often published in *normalized* form. The relationship between normalized $\bar{C}_{l,m}, \bar{S}_{l,m}$ and un-normalized gravity coefficients $C_{l,m}, S_{l,m}$ is given by the following expression:

$$\begin{Bmatrix} \bar{C}_{l,m} \\ \bar{S}_{l,m} \end{Bmatrix} = \left[\frac{1}{(2 - \delta_{m0})(2l + 1)(l - m)!} \right]^{1/2} \begin{Bmatrix} C_{l,m} \\ S_{l,m} \end{Bmatrix}$$

where δ_{m0} is equal to 1 if m is zero and equal to zero if m is greater than zero.

This software suite contains a MATLAB script named `unnormalize.m` that demonstrates how to process a data file of normalized gravity coefficients and create the corresponding un-normalized gravity coefficients. This script processes a simple 4 by 4 ASCII data file named `lp150q.dat`.

The following is the output computed by this script.

Orbital Mechanics with MATLAB

normalized gravity coefficients

2	0	-9.090109494810e-005	+0.000000000000e+000
3	0	-3.203071679590e-006	+0.000000000000e+000
4	0	+3.214095450280e-006	+0.000000000000e+000
2	1	-1.862736081840e-009	-1.424538946100e-009
3	1	+2.634183586220e-005	+5.463078608820e-006
4	1	-6.000619397400e-006	+1.659556447270e-006
2	2	+3.463762742080e-005	+1.440635035400e-008
3	2	+1.418533167860e-005	+4.889139117950e-006
4	2	-7.093701015440e-006	-6.785627355580e-006
3	3	+1.228626450440e-005	-1.782462707200e-006
4	3	-1.358804665940e-006	-1.343325717370e-005
4	4	-6.029391501930e-006	+3.935256944400e-006

un-normalized gravity coefficients

2	0	-2.032610275331e-004	+0.000000000000e+000
3	0	-8.474531095709e-006	+0.000000000000e+000
4	0	+9.642286350840e-006	+0.000000000000e+000
2	1	-2.404781941115e-009	-1.839071871423e-009
3	1	+2.845243462382e-005	+5.900799313130e-006
4	1	-5.692687400271e-006	+1.574393483697e-006
2	2	+2.235849235882e-005	+9.299259166779e-009
3	2	+4.845213176981e-006	+1.669958927652e-006
4	2	-1.586199768258e-006	-1.517312403706e-006
3	3	+1.713237731043e-006	-2.485525493170e-007
4	3	-8.120411056143e-008	-8.027906645276e-007
4	4	-1.273941470320e-007	+8.314747875024e-008

jgm3.dat – Joint Gravity Model 3 data file

This ASCII data file is an 18 by 18 version of the Joint Gravity Model 3 (JGM3) gravity model. The following is a portion of this data file. In this file column one is the model degree index, column two is the order index, and columns three and four are the corresponding *un-normalized* gravity coefficients (zonals and tesserals, respectively).

2	0	-0.10826267E-02	0.00000000E+00
3	0	0.25324353E-05	0.00000000E+00
4	0	0.16193312E-05	0.00000000E+00
5	0	0.22771610E-06	0.00000000E+00
6	0	-0.53964849E-06	0.00000000E+00
7	0	0.35136844E-06	0.00000000E+00
8	0	0.20251872E-06	0.00000000E+00
9	0	0.11936871E-06	0.00000000E+00
10	0	0.24805686E-06	0.00000000E+00
11	0	-0.24056521E-06	0.00000000E+00
12	0	0.18191170E-06	0.00000000E+00
13	0	0.20756773E-06	0.00000000E+00
14	0	-0.11741739E-06	0.00000000E+00
15	0	0.17627270E-07	0.00000000E+00
16	0	-0.31194308E-07	0.00000000E+00
17	0	0.10713059E-06	0.00000000E+00
18	0	0.44216724E-07	0.00000000E+00

Orbital Mechanics with MATLAB

2	1	-0.24140001E-09	0.15431000E-08
3	1	0.21927988E-05	0.26801189E-06
4	1	-0.50872530E-06	-0.44945994E-06
5	1	-0.53716510E-07	-0.80663464E-07
6	1	-0.59877977E-07	0.21164664E-07
7	1	0.20514873E-06	0.69369894E-07
8	1	0.16034587E-07	0.40199782E-07
9	1	0.92419272E-07	0.14236570E-07
10	1	0.51755787E-07	-0.81289149E-07
11	1	0.95084276E-08	-0.16465464E-07
12	1	-0.30680009E-07	-0.23784484E-07
13	1	-0.28851306E-07	0.21721093E-07
14	1	-0.99977097E-08	0.14437496E-07
15	1	0.61088619E-08	0.41541858E-08
16	1	0.13562787E-07	0.16604395E-07
17	1	-0.12621442E-07	-0.14278224E-07
18	1	0.19583334E-08	-0.18176561E-07

wgs84.dat – World Geodetic System 1984 data file

This ASCII data file is an 18 by 18 version of the World Geodetic System 1984 (WGS84) gravity model. The following is a portion of this data file. In this file column one is the degree index, column two is the model order index, and columns three and four are the corresponding *un-normalized* gravity coefficients (zonals and tesserals, respectively).

2	0	-0.10826300D-02	0.00000000D+00
3	0	0.25321531D-05	0.00000000D+00
4	0	0.16109876D-05	0.00000000D+00
5	0	0.23578565D-06	0.00000000D+00
6	0	-0.54316985D-06	0.00000000D+00
7	0	0.33237640D-06	0.00000000D+00
8	0	0.17721040D-06	0.00000000D+00
9	0	0.14459876D-06	0.00000000D+00
10	0	0.23339780D-06	0.00000000D+00
11	0	-0.27870829D-06	0.00000000D+00
12	0	0.17036617D-06	0.00000000D+00
13	0	0.25024428D-06	0.00000000D+00
14	0	-0.13764093D-06	0.00000000D+00
15	0	-0.30920023D-07	0.00000000D+00
16	0	0.55350560D-07	0.00000000D+00

jzonal.dat – zonal gravity coefficients data file

This simple data file contains 18 un-normalized zonal gravity coefficients coded as follows:

```
jzonal(1) = 1;  
jzonal(2) = 1.08262668355e-3;  
jzonal(3) = -2.53265648533e-6;  
jzonal(4) = -1.61962159137e-6;  
jzonal(5) = -2.27296082869e-7;  
jzonal(6) = 5.40681239107e-7;  
jzonal(7) = -3.52359908418e-7;  
jzonal(8) = -2.04799466985e-7;  
jzonal(9) = -1.20616967365e-7;
```

Orbital Mechanics with MATLAB

```
jzonal(10) = -2.41145438626e-7;  
jzonal(11) =  2.44402148325e-7;  
jzonal(12) = -1.88626318279e-7;  
jzonal(13) = -2.19788001661e-7;  
jzonal(14) =  1.30744533118e-7;  
jzonal(15) = -8.23528409456e-9;  
jzonal(16) =  1.81139265112e-8;  
jzonal(17) = -1.16904733834e-7;  
jzonal(18) = -3.09424678746e-8;
```

atmos76.dat – 1976 U. S. Standard atmosphere data file

This ASCII data file contains density values based the 1976 U. S. Standard atmosphere. The unit of this data is kilograms/cubic kilometer, data is valid from 0 to 1000 kilometers, and the altitude interval is 500 meters. The following are the first few lines of data in this file.

```
.1225000E+10  
.1167273E+10  
.1111660E+10  
.1058104E+10  
.1006554E+10  
.9569545E+09  
.9092544E+09  
.8634021E+09  
.8193470E+09  
.7770388E+09  
.7364289E+09  
.6974689E+09  
.6601116E+09  
.6243101E+09  
.5900187E+09
```