

Nodal Period of a Satellite (Numerical Integration Solution)

This *Numerit* application (`period2`) calculates the nodal period of an Earth satellite using a gravity model that includes J_2 . This program demonstrates how to perform one-dimensional root-finding while numerically integrating a satellite's equations of motion. To illustrate this useful numerical technique the computer program will be explained in detail.

The major steps of this numerical procedure are as follows:

- (1) define the initial value of the objective function
- (2) propagate the satellite's orbit from t to $t + \Delta t$
- (3) examine the objective function at time t and $t + \Delta t$ for a bracketed root
- (4) if bracketed, solve for the root of the objective function
- (5) continue the search and solve procedure until the final time is reached

The first part of the program defines several astrodynamic constants and initializes the Runge-Kutta-Fehlberg algorithm. This source code is as follows:

```
` period2.num      October 9, 1999
` nodal period of an Earth satellite
` numerically integrated solution
` Orbital Mechanics with Numerit
.....
common mu, req, j2, rkcoef

clear

draft

outprec 15

` astrodynamic and utility constants

j2 = 0.00108263      ` second zonal gravity coefficient (nd)
mu = 398600.5        ` Earth gravitational constant (km^3/sec^2)
req = 6378.14        ` Earth equatorial radius (km)

pi2 = 2 * pi
dtr = pi / 180      ` degrees to radians
rtd = 180 / pi      ` radians to degrees

` initialize rkf78 integration

rkcoef = 1

neq = 6
```

Orbital Mechanics with Numerit

The tolerance for the root-finder and integration are set in the next few lines of code.

```
` define root-finding tolerance
rtol = 1.0e-4
` set rkf78 truncation error tolerance
tetol = 1.0e-8
```

For this example the initial true anomaly is set to the ascending node and the initial right ascension of the ascending node (RAAN) is set to zero as follows:

```
` set true anomaly to ascending node
oev[6] = -oev[4]
` set raan to zero
oev[5] = 0
```

The *Numerit* program then calculates the Keplerian period and the initial ECI state vector with the next few lines of code.

```
` Keplerian period
tkepler = pi2 * sqrt(oev[1] * oev[1] * oev[1] / mu)
` determine initial eci state vector
orb2eci(mu, oev, r, v)
```

This initial state vector is loaded into a *working* array with

```
` load state vector working array
for i = 1 to 3
  yold[i] = r[i]
  yold[i + 3] = v[i]
```

The step size guess and propagation step size are set in the next lines:

```
` step size guess (seconds)
h = 60
` propagation step size
dtstep = 60
```

The propagation step size should not be set too large or the algorithm may step over valid roots of the objective function.

Orbital Mechanics with Numerit

Before the integration and root-finding begin, the final time is set to zero as follows:

```
` initialize "final" time
tf = 0
```

The part of the algorithm which propagates the orbit and calculates the value of the objective function after each step is illustrated by the following source code:

```
` initial geocentric declination
fxnew = 0
loop
` initialize root flag
rflg = 0
` save current value of objective function
fxold = fxnew
` set initial time to final time
ti = tf
` increment final time
tf = ti + dtstep
` save initial time as left side of bracket
tisaved = ti
` integrate equations of motion
orbeqm -> j2eqm
rkf78(neq, ti, tf, h, tetol, yold, ynew)
` compute current value of declination
rmag = sqrt(ynew[1] * ynew[1] + ynew[2] * ynew[2] + ynew[3] *
ynew[3])
fxnew = ynew[3] / rmag
yold = ynew
```

After integrating the equations of motion, the algorithm computes the new or current value of the objective function. The software determines if a single root has been bracketed during this integration step with the following code:

```
` check to see if the user-defined objective
```

Orbital Mechanics with Numerit

```
` function has been bracketed during this step
```

```
if (fxnew * fxold < 0 and ynew[6] > 0) rflg = 1
```

Notice for this example that we are looking for the condition $f(t)f(t + \Delta t) < 0$ and also the condition that the z component of the velocity is positive, v_z . The second condition or *mission constraint* ensures that the root is also an ascending node crossing. The objective function for this example is the sine of the geocentric declination of the satellite. This quantity is calculated from $\sin \delta = r_z/r$ where r_z is the z component of the position vector and r is the scalar magnitude of the position vector.

If the root is bracketed and all mission constraints are satisfied, the algorithm solves for the actual root with this next part of the code:

```
` if bracketed, find root of objective function

if (rflg = 1)
  ` load "working" time and state vector array
  ` as values on right side of bracket

  tiwrk = tf

  yi = ynew

  ` find root of objective function

  objfunc -> pfunc2

  brent(tisaved, tiwrk, rtol, xroot, froot)

  break
```

Otherwise, the algorithm increments both the initial and final times and repeats the orbit integration and root bracketing check. Notice how we use the "redirection" feature of **Numerit Pro** to tell the software the name of the objective function.

The objective function called by the root-finding routine is as follows:

```
function pfunc2(x, fx)

` declination objective function

` input

` x = current simulation time

` output

` fx = geocentric declination at x (radians)

` Orbital Mechanics with Numerit
.....
` number of differential equations
```

Orbital Mechanics with Numerit

```
neq = 6
` step size guess (seconds)
h = 10
` integrate from tiwrk to requested time = x
tetol = 1.0e-8
orbeqm -> j2eqm
rkf78(neq, %tiwrk, x, h, tetol, %yi, %yfinal)
` compute current value of geocentric declination
rmag = sqrt(%yfinal[1] * %yfinal[1] + %yfinal[2] * %yfinal[2] \
           + %yfinal[3] * %yfinal[3])
fx = %yfinal[3] / rmag
```

Notice that both the "working time" `%tiwrk` and the state vector at this time `%yi` are passed to this routine via global. This time and state vector correspond to the values on the right hand side of the integration interval. Notice also that the final state vector `%yfinal` calculated by this function is passed to the calling routine via global. If the final state vector is the converged root, it is available to the calling routine. The calling argument `x` is the integration time requested by the `brent` routine. The equations of motion for this example are coded in the `j2eqm` function. However, the equations of motion can include aerodynamic drag, third body perturbations and so forth. Any equations of motion can be used provided they are compatible with the `rkf78` function. We can use the "redirection" feature of *Numerit Pro* to tell the software the name of the equations of motion function.

The following is a typical draft output created with this software. It illustrates the results for the "hardwired" default program input. The value for geocentric declination is an indication of how well the algorithm solved for a root.

```
program period2
< nodal period - integrated solution >
perigee altitude          1501.86 kilometers
apogee altitude          1741.86 kilometers
semimajor axis           8000 kilometers
eccentricity             0.015
inclination              28.5 degrees
argument of perigee      270 degrees
Keplerian period         118.6847 minutes
nodal period             118.3866 minutes
```