

## Predicting Orbital Events: Part 2

This *Numerit* program (demoevt2) demonstrates how to calculate *root-finding* types of orbital events. These are events which involve finding the roots of a *scalar objective function* which is typically a function of time. The roots of an objective function are the points where the value of the user-defined function is zero. Since many orbital events can be expressed as a function of time, a variety of important and unique events can be calculated using this technique. Typical root-finding events include rise and set of an Earth satellite, shadow conditions of satellites and "constrained" close approach between two satellites or a satellite and ground site.

This demonstration program calculates the calendar date, universal time and slant range for closest approach conditions between an Earth satellite and a ground site subject to a user-defined distance constraint.

The basic procedure for predicting these types of events involves the following steps and *Numerit* functions:

- (1) locate an extremum using the functions `oevent2` and `minima`
- (2) bracket a "forward" and "backward" root using the function `broot`
- (3) find each root using the function `brent`
- (4) calculate the event circumstances with a user-coded function
- (5) display the event circumstances with a second user-coded function

The main *Numerit* function which solves this type of problem has the following syntax and arguments:

```
function oevent2 (ti, tf, dt, dtssl)
` predict root-finding type orbital events
` input
`  ti    = initial simulation time
`  tf    = final simulation time
`  dt    = step size used for bounding minima
`  dtssl = small step size used to determine whether
`          the function is increasing or decreasing
```

The search function `oevent2` requires both a function which evaluates the user-defined scalar objective value and a function which displays the circumstances of the orbital events. The main program "reroutes" the names of both the objective and event functions prior to calling the `oevent2` function which searches for the events. Rerouting is a powerful feature unique to *Numerit Pro*. The user must write the source code for both of these support functions.

## Orbital Mechanics with Numerit

For this example the code which does the rerouting is as follows:

```
objfunc -> objfunc1
events -> events1
```

### Choosing the Search Parameters

The proper selection of the search parameters  $dt$  and  $dt_{sm1}$  depends on the type of orbital event you are trying to predict. For example, the value of  $dt$  depends on how often the event occurs. For events which happen once per orbital period (aphelion, perihelion, etc.), a good value for  $dt$  is one fourth of the satellite's orbital period. When calculating events involving two satellites, one should use one fourth of the smaller of the two orbital periods. Be careful not to make  $dt$  too large or the algorithm may "step over" one or more solutions.

The value of  $dt_{sm1}$  must be selected such that it produces a "big" enough change in the objective function to tell the algorithm in which direction to move. This is similar to choosing the value of  $x$  when numerically estimating the derivative of a function of the form  $y = f(x)$ . For most events a value of  $dt_{sm1} = 10 / 86400$  days should be adequate. An examination of a plot of the objective function over a period of time can also provide insight into the proper selection of the  $dt_{sm1}$  parameter.

During the search process the `oevent2` function calls a function named `minima` which calculates the *extremum* of a one-dimensional user-defined objective function. The syntax of this optimization function is as follows:

```
function minima (a, b, tolm, xmin, fmin)
` one-dimensional minimization function
` Brent's method
` input
` a    = initial x search value
` b    = final x search value
` tolm = convergence criterion
` output
` xmin = minimum x value
` fmin = minimum function value
```

This function in turn requires an *objective function* coded as `name(x, fx)` where  $x$  is the current function argument,  $fx$  is the objective function evaluated at  $x$  and `name` is the actual name of the user-coded function. The function argument is usually the time, either as a Julian date or perhaps time since a simulation began. The function value  $fx$  might be

## Orbital Mechanics with Numerit

geocentric declination, the separation angle between satellites, the scalar distance between two satellites and so forth.

The source code of the objective function for this example is as follows:

```
function objfunc1(x, fx)
` objective function
` slant range between an Earth
` satellite and ground site
` input
` x = function argument
` output
` fx = function value at x
` Orbital Mechanics with Numerit
.....
` observer local sidereal time
obslst = modulo(%gsti + %omega * x + %obslong)
` compute ground site position vector
sobslst = sin(obslst)
cobslst = cos(obslst)
rsite[1] = %gx * %cobslat * cobslst
rsite[2] = %gx * %cobslat * sobslst
rsite[3] = %gy * %sobslat
` compute satellite position vector
kozai1(0, x, %oev1, oev2, rsat, vsat)
` eci vector from observer to satellite
for i = 1 to 3
    obs2sat[i] = rsat[i] - rsite[i]
` observer-to-satellite slant range
srange = sqrt(vdot(obs2sat, obs2sat))
%drsaved = srange
fx = srange - %camin
```

Variables preceded with % are global items computed either in the main program or in other support or utility functions. Notice how the the distance constraint is enforced using %camin.

## *Orbital Mechanics with Numerit*

The syntax of the function which calculates and displays the actual roots of the user-defined objective function is as follows:

```
function events1 (ti, tf, topt)
` compute circumstances of orbital events
` input
`  ti   = initial simulation time
`  tf   = final simulation time
`  topt = extrema time
` Orbital Mechanics with Numerit
.....
` define root-bracketing and root-finding control parameters
factor = 0.25           ` geometric acceleration factor
dxmax  = 300 / 86400   ` rectification interval
rtol   = 0.00000001   ` convergence tolerance
` compute and display event start conditions
tlin = topt
t2in = tlin - (10 / 86400)
broot(tlin, t2in, factor, dxmax, t1out, t2out)
brent(t1out, t2out, rtol, troot, froot)
` set to initial time if before ti
if (troot < ti)
    troot = ti
    objfunc(ti, froot)
jdate = %jdatei + troot
r2sprint(1, jdate)
` compute and display conditions at optimum
objfunc(tlin, froot)
jdate = %jdatei + tlin
r2sprint(2, jdate)
` compute and display event end conditions
t2in = tlin + (10 / 86400)
broot(tlin, t2in, factor, dxmax, t1out, t2out)
brent(t1out, t2out, rtol, troot, froot)
` set to final time if after tf
```

## Orbital Mechanics with Numerit

```
if (troot > tf)
    troot = tf
    objfunc(tf, froot)

%trr = troot

jdate = %jdatei + troot

r2sprint(3, jdate)
```

The `%trr = troot` line of this function "saves" the forward root which is made available to the `oevent2` function with a common `trr` statement at the beginning of the main program. This technique forces the event-finding routine to continue the search after the forward root is calculated which speeds up the process.

The `events1` function is a "driver" routine which uses the time of the extremum `ttopt` to first bracket the backward and forward roots and then actually find the value of each root. After each root is calculated this routine calls a function which displays the event conditions.

For this example the *Numerit* source code listing for the display function is as follows:

```
function r2sprint(iflag, jdate)
    ` print ground site-to-satellite
    ` closest approach conditions

    ` Orbital Mechanics with Numerit
    .....
    rtd = 180 / pi

    gdate(jdate, month, day, year)

    thours = 24 * (day - trunc(day))

    println

    if (iflag = 1)
        println "time and conditions at constraint entry"
        jdsaved = jdate

    if (iflag = 2)
        println "closest approach conditions"

    if (iflag = 3)
        println "time and conditions at constraint exit"

    println

    print "calendar date          "

    prtdate(month, trunc(day), year)
```

## Orbital Mechanics with Numerit

```
print "universal time          "
prtttime(thours)
outprec 10
println "Julian date          ", jdate
` display slant range
println
println "topocentric slant range    ", %drsaved, " kilometers"
if (iflag = 3)
  deltat = 24 * (jdate - jdsaved)
  println
  print "event duration          "
  prtttime(deltat)
println
```

The following is a brief description of the function which brackets each root.

### **function broot**

This function can be used to bracket a single root of a single nonlinear equation of the form  $y=f(x)$ . It uses a combination of *geometric acceleration* and *rectification* to bracket single roots. The basic idea is to find the endpoints of an interval  $x_1$  and  $x_2$  such that  $f(x_1)f(x_2)<0$ . This condition guarantees that there is at least one root in the interval because the objective function changes *sign*.

The user must supply this function with an initial guess for  $x_1$  and  $x_2$ . Typically  $x_1$  is the time of an objective function minimum or maximum and  $x_2$  is a time 10/86400 days after (forward root) or 10/86400 days before (backward root) the value of  $x_1$ .

The syntax of this *Numerit* root bracketing function is as follows:

```
function broot (x1in, x2in, factor, dxmax, x1out, x2out)
` bracket a single root of a nonlinear equation
` input
` x1in   = initial guess for first bracketing x value
` x2in   = initial guess for second bracketing x value
` factor = acceleration factor (non-dimensional)
` dxmax  = rectification interval
` output
```

## Orbital Mechanics with Numerit

```
` x1out = final value for first bracketing x value
` x2out = final value for second bracketing x value
```

The acceleration factor used in this demonstration program is 0.25 and the rectification interval is 300/86400 days. The acceleration factor increases the size of each step geometrically and the rectification interval monitors the length of the current bracketing interval. When necessary the rectification logic reinitializes the search and prevents the interval from becoming too large and skipping one or both ends of the bracketing interval completely. The size of each step increases geometrically until the length of the current step reaches the value of the rectification interval. At that point the step size is reset and the process begins again. Eventually, the process will bracket the root and the function will return the endpoints of this bracketing interval.

The following is a brief description of the function which solves for each root.

### **function brent**

This function uses Brent's method to find a single root of a single nonlinear equation of the form  $y=f(x)$ . Analytic derivatives are not required for this algorithm. However, the user should ensure that a root is bracketed before calling this routine. Whenever a root is bracketed the following condition is satisfied;  $f(x_1)f(x_2) < 0$ . This inequality implies that the *sign* of the function has changed between  $x_1$  to  $x_2$ .

Additional information about this numerical method can be found in *Algorithms for Minimization Without Derivatives*, R. P. Brent, Prentice-Hall, 1972. As the title of this book indicates, this algorithm does not require derivatives of the objective function. This feature is important because the analytic first derivative of many objective functions is difficult to derive and code.

The syntax and arguments of this useful function are

```
function brent (x1, x2, rtol, xroot, froot)
` solve for a single real root of a nonlinear equation
` Brent's method
` input
` x1 = lower bound of search interval
` x2 = upper bound of search interval
` rtol = algorithm convergence criterion
` output
` xroot = real root of  $f(x) = 0$ 
` froot = function value at  $f(x) = 0$ 
```

## Orbital Mechanics with Numerit

The root-finding performance of this algorithm is determined by `rtol`. A value of `1.0e-8` should be adequate for most problems. Smaller values will predict roots more accurately at the expense of longer program execution times.

Along with the initial orbital elements, the program will prompt you for an initial calendar date, universal time and search interval in days. It will also ask you for the geographic coordinates of the observer. Please note that north latitudes are positive and south latitudes are negative. Also note that east longitudes are positive and west longitudes are negative. Furthermore, observer sites above sea level have positive altitudes and sites below sea level are negative.

The following is a typical draft output created with this software. The distance constraint for this example was 2000 kilometers.

```
demoevt2

< 'constrained' closest approach between a satellite and ground site >

time and conditions at constraint entry

calendar date          January 1, 1998
universal time         9 h 54 m 59.568 s
Julian date            2450814.913

topocentric slant range 1999.999173 kilometers

closest approach conditions

calendar date          January 1, 1998
universal time         9 h 56 m 52.6426 s
Julian date            2450814.914

topocentric slant range 1884.715944 kilometers

time and conditions at constraint exit

calendar date          January 1, 1998
universal time         9 h 58 m 45.7028 s
Julian date            2450814.916

topocentric slant range 1999.999173 kilometers

event duration         00 h 3 m 46.1348 s

time and conditions at constraint entry

calendar date          January 1, 1998
universal time         11 h 59 m 44.6953 s
Julian date            2450815

topocentric slant range 2000.001436 kilometers
```

## *Orbital Mechanics with Numerit*

### closest approach conditions

calendar date                    January 1, 1998  
universal time                    12 h 3 m 0.995719 s  
Julian date                        2450815.002

topocentric slant range        1630.360609 kilometers

### time and conditions at constraint exit

calendar date                    January 1, 1998  
universal time                    12 h 6 m 17.3117 s  
Julian date                        2450815.004

topocentric slant range        2000.001436 kilometers

event duration                    00 h 6 m 32.6164 s

### time and conditions at constraint entry

calendar date                    January 1, 1998  
universal time                    14 h 6 m 21.1403 s  
Julian date                        2450815.088

topocentric slant range        2000.001384 kilometers

### closest approach conditions

calendar date                    January 1, 1998  
universal time                    14 h 9 m 34.5016 s  
Julian date                        2450815.09

topocentric slant range        1641.729279 kilometers

### time and conditions at constraint exit

calendar date                    January 1, 1998  
universal time                    14 h 12 m 47.8447 s  
Julian date                        2450815.092

topocentric slant range        2000.001384 kilometers

event duration                    00 h 6 m 26.7044 s