

Closest Approach Between a Satellite and Ground Site (Numerical Integration Solution)

This *Numerit* application (`rts2sat2`) calculates close approach conditions between a satellite and a ground site on an oblate Earth. This program demonstrates how to perform one-dimensional minimization while numerically integrating a satellite's equations of motion. To illustrate this useful numerical technique the computer program will be explained in detail.

The first part of this program defines several "commons" and astrodynamic constants and also sets the initialization flag for the Runge-Kutta-Fehlberg 7(8) integration method.

```
` rts2sat2.num      October 11, 1999

` closest approach conditions between
` a ground site and Earth satellite

` numerical integration method

` Orbital Mechanics with Numerit
.....
common rkcoef, mu, req, j2

clear

draft

` astrodynamic and utility constants

mu = 398600.5
req = 6378.14
j2 = 0.00108263
flat = 1 / 298.257
omega = 7.29211564186e-05

` initialize rkf78 algorithm

rkcoef = 1
```

The next part of the main program requests the initial calendar date and time, the *osculating* orbital elements of the satellite, the geographic coordinates of the ground site and the total simulation duration.

```
` begin simulation

println "program rts2sat2"

println

println "closest approach between a ground site and Earth satellite"

println
```

Orbital Mechanics with Numerit

```
println "< numerical integration method >"
` request initial calendar date
cdate = "1,1,1998"
getdate(cdate, month, day, year)
` request initial universal time
utstr = "0,0,0"
gettime(utstr, uthr, utmin, utsec)
` request initial orbital elements
ioev[6] = 0
ioev = 1,1,1,1,1,1
oev1[1] = 8000
oev1[2] = 0
oev1[3] = 40
oev1[4] = 55
oev1[5] = 55
oev1[6] = 10
getoe(ioev, oev1)
` request ground site coordinates
obslatstr = "40,0,0"
obslongstr = "-105,0,0"
obsalt = 1000
getobs(obslatstr, obslongstr, obslat, obslong, obsalt)
` request the simulation duration
loop
  ndays = 1
  input "simulation duration in days" ndays
  if (ndays > 0) break
```

The next section calculates the orbital period of the satellite normalized with respect to $2p$, the geodetic constants of the ground site, the initial Julian date and the satellite's position and velocity vectors.

```
` initial julian date
julian(month, day, year, jdate)
jdatei = jdate + uthr / 24 + utmin / 1440 + utsec / 86400
```

Orbital Mechanics with Numerit

```
` calculate orbital period
period = sqrt(oev1[1] * oev1[1] * oev1[1] / mu)
` compute geodetic constants
sobslat = sin(obslat)
cobslat = cos(obslat)
b = sqrt(1 - (2 * flat - flat * flat) * sobslat * sobslat)
gx = req / b + obsalt
gy = req * (1 - flat) * (1 - flat) / b + obsalt
` initial greenwich sidereal time (radians)
gast2(jdatei, gsti)
` determine initial eci state vector
orb2eci(mu, oev1, r, v)
` load state vector working array
for i = 1 to 3
  ysaved[i] = r[i]
  ysaved[i + 3] = v[i]
```

The next section of the code defines the initial and final times of the search interval, in seconds, and the "search" and "function" step sizes. The search step size tells the algorithm how far to move when looking for minima and the function step size tells the program how far to move when looking for an increase (or decrease) in the objective function values. Since most orbital events occur no more than once per orbit, a search step size of one fourth of an orbital period is adequate. A function step size of 10 seconds will ensure a "big" enough change in the objective function value, and the algorithm can successfully determine in what direction to move. If the objective function changes slowly, a longer function step size will be necessary.

```
` initialize search parameters
ti = 0
tf = 86400 * ndays
tisaved = ti
dt = period / 4
dtsml = 10
```

The final action of the main program is to call the `oevent1` function which actually finds the minima within the user-defined search interval. The source code which does this is as follows:

Orbital Mechanics with Numerit

```
` find and display orbital events  
  
objfunc -> r2sfunc2  
  
events -> events2  
  
oevent1(ti, tf, dt, dtsml)
```

Notice how we have used the "redirection" feature of *Numerit Pro* to tell the `oevent1` function which objective and event functions to use.

The source code for the objective function which calculates the current topocentric slant range between the satellite and ground site is as follows:

```
function r2sfunc2(x, y)  
  
` numerical integration closest  
` approach objective function  
  
` input  
  
` x = simulation time argument  
  
` output  
  
` y = rts-to-satellite slant range (kilometers)  
  
` 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  
  
ttmp = %tisaved  
  
for i = 1 to 6  
    ytmp[i] = %ysaved[i]  
  
` number of differential equations  
  
neq = 6  
  
` step size guess (seconds)  
  
h = 10
```

Orbital Mechanics with Numerit

```
` integrate from tisaved to requested time = x
tetol = 1.0e-8
orbeqm -> j2eqm
rkf78(neq, ttmp, x, h, tetol, ytmp, %ysaved)
` compute satellite position vector
for i = 1 to 3
    rsat[i] = %ysaved[i]
` 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
y = srange
%tisaved = x
```

Notice that both the "saved time" `%tisaved` and the state vector at this time `%ysaved` are passed to this routine via global. This time and state vector correspond to the current values being processed by the `minima` routine. Notice also that the final state vector and time calculated by this function are "re-saved" after the integration is complete. This function also saves the current slant range value in `drsaved` in case this value is in fact the "answer". The calling argument `x` is the integration time determined 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.

The following is the *Numerit* code for the support function which calculates the Julian date and close approach conditions.

```
function events2 (ti, tf, topt)
` compute and display minimization type orbital event
` input
` ti = initial time
` tf = final time
` topt = extrema time
` Orbital Mechanics with Numerit
```

Orbital Mechanics with Numerit

```
.....  
` compute and display conditions at closest approach  
  
r2sfunc2(topt, froot)  
  
jdate = %jdatei + topt / 86400  
  
r2sprint(2, jdate)
```

Finally, the following is the *Numerit* code for the support function which actually displays the calendar date, universal time, Julian date and topocentric slant range for each close approach found by this technique.

```
function r2sprint(iflag, jdate)  
` print closest approach conditions  
` Orbital Mechanics with Numerit  
.....  
gdate(jdate, month, day, year)  
thours = 24 * (day - trunc(day))  
  
println; println  
  
if (iflag = 1)  
    println "time and conditions at constraint entry"  
    jdsaved = jdate  
  
if (iflag = 2)  
    println "time and conditions at closest approach"  
  
if (iflag = 3)  
    println "time and conditions at constraint exit"  
  
println  
  
print "calendar date          "  
prtdate(month, trunc(day), year)  
print "universal time          "  
prtttime(thours)  
  
outprec 10  
println "Julian date              ", jdate  
` display slant range  
println  
println "slant range                ", %drsaved, " kilometers"
```

Orbital Mechanics with Numerit

The `iflag` input argument determines which event is being processed and `jdate` corresponds to the Julian date of this event. For this example there is only one event.

The following is a typical draft output created with this software. It illustrates the first few close approaches for the "hardwired" default program input.

```
program rts2sat2

closest approach between a ground site and Earth satellite

< numerical integration method >

time and conditions at closest approach

calendar date           January 1, 1998
universal time          1 h 59 m 43.4388 s
Julian date             2450814.583

slant range             5936.094322 kilometers

time and conditions at closest approach

calendar date           January 1, 1998
universal time          4 h 3 m 48.1364 s
Julian date             2450814.669

slant range             3841.68199 kilometers

time and conditions at closest approach

calendar date           January 1, 1998
universal time          6 h 8 m 22.9443 s
Julian date             2450814.756

slant range             2254.186285 kilometers

time and conditions at closest approach

calendar date           January 1, 1998
universal time          8 h 14 m 2.10404 s
Julian date             2450814.843

slant range             1653.092865 kilometers

time and conditions at closest approach

calendar date           January 1, 1998
universal time          10 h 20 m 29.0976 s
Julian date             2450814.931

slant range             1623.784802 kilometers
```