Writing a SPICE (FORTRAN) Based Program

January 2020
Undefined variables are displayed in red
Results are displayed in blue
First, let's go over the important steps in the process of writing a SPICE-based Fortran program and putting it to work:

- Understand the geometry problem.
- Identify the set of SPICE kernels that contain the data needed to perform the computation.
- Select the SPICE APIs needed to compute the quantities of interest.
- Write and compile the program.
- Get actual kernel files and verify that they contain the data needed to support the computation for the time(s) of interest.
- Run the program.

To illustrate these steps, let's write a program that computes the apparent intersection of the boresight ray of a given CASSINI science instrument with the surface of a given Saturnian satellite. The program will compute:

- Planetocentric and planetodetic (geodetic) latitudes and longitudes of the intercept point.
- Range from spacecraft to intercept point.
- Illumination angles (phase, solar incidence, and emission) at the intercept point.
We want the boresight intercept on the surface, range from s/c to intercept, and illumination angles at the intercept point.

When? TIME (UTC, TDB or TT)

On what object? SATNM

In what frame? FIXREF

For which instrument? INSTNM

For what spacecraft? SCNM

Using what model? SETUPF
Needed Data

Time transformation kernels

Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft, Saturn barycenter and satellites.
Data required to compute vectors, rotations and other parameters shown in the picture are stored in the SPICE kernels listed below.

Note: these kernels have been selected to support this presentation; they should not be assumed to be appropriate for user applications.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Kernel Type</th>
<th>File name</th>
</tr>
</thead>
<tbody>
<tr>
<td>time conversions</td>
<td>generic LSK</td>
<td>naif0009.tls</td>
</tr>
<tr>
<td>satellite orientation</td>
<td>CASSINI SCLK</td>
<td>cas00084.tsc</td>
</tr>
<tr>
<td>satellite shape</td>
<td>CASSINI PCK</td>
<td>cpck05Mar2004.tpc</td>
</tr>
<tr>
<td>satellite position</td>
<td>planet/sat PCK</td>
<td>020514_SE_SAT105.bsp</td>
</tr>
<tr>
<td>planet barycenter position</td>
<td>planet SPK</td>
<td>981005_PLTEPH-DE405S.bsp</td>
</tr>
<tr>
<td>spacecraft position</td>
<td>spacecraft SPK</td>
<td>030201AP_SK_SM546_T45.bsp</td>
</tr>
<tr>
<td>spacecraft orientation</td>
<td>spacecraft CK</td>
<td>04135_04171pc_psiv2.bc</td>
</tr>
<tr>
<td>instrument alignment</td>
<td>CASSINI FK</td>
<td>cas_v37.tf</td>
</tr>
<tr>
<td>instrument boresight</td>
<td>Instrument IK</td>
<td>cas_iss_v09.ti</td>
</tr>
</tbody>
</table>
The easiest and most flexible way to make required kernels available to the program is via FURNISH. For this example we make a setup file (also called a “metakernel” or “furnsh kernel”) containing a list of kernels to be loaded:

Note: these kernels have been selected to support this presentation they should not be assumed to be appropriate for user applications.

\begindata

KERNELS_TO_LOAD = ( 'naif0009.tls', 'cas00084.tsc', 'cpck05Mar2004.tpc', '020514_SE_SAT105.bsp', '981005_PLTEPH-DE405S.bsp', '030201AP_SK_SM546_T45.bsp', '04135_04171pc_psiv2.bc', 'cas_v37.tf', 'cas_iss_v09.ti')

\begintext

and we make the program prompt for the name of this setup file:

CALL PROMPT ( 'Enter setup file name > ', SETUPF )
CALL FURNISH ( SETUPF )
• Prompt for setup file (“metakernel”) name load kernels specified via setup file. (Done on previous chart.)

• Prompt for user inputs required to completely specify problem. Obtain further inputs required by geometry routines via SPICELIB calls.

• Compute the intersection of the boresight direction ray with the surface of the satellite, presented as a triaxial ellipsoid.

    If there is an intersection,

    • Convert Cartesian coordinates of the intercept point to planetocentric latitudinal and planetodetic coordinates
    • Compute spacecraft-to-intercept point range
    • Find the illumination angles (phase, solar incidence, and emission) at the intercept point

• Display the results.

We discuss the geometric portion of the problem next.
Compute the intercept point (**POINT**) of the boresight vector (**INSITE**) specified in the instrument frame (**IFRAME**) of the instrument mounted on the spacecraft (**SCNM**) with the surface of the satellite (**SATNM**) at the TDB time of interest (**ET**) in the satellite’s body-fixed frame (**FIXREF**). This call also returns the light-time corrected epoch at the intercept point (**TRGEPC**), the spacecraft-to-intercept point vector (**SRFVEC**), and a flag indicating whether the intercept was found (**FOUND**). We use "converged Newtonian" light time plus stellar aberration corrections to produce the most accurate surface intercept solution possible. We model the surface of the satellite as an ellipsoid.

```
CALL SINCPT ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, IFRAME, 
            INSITE, POINT, TRGEPC, SRFVEC, FOUND )
```

The range we want is obtained from the outputs of **SINCPT**. These outputs are defined only if a surface intercept is found. If **FOUND** is true, the spacecraft-to-surface intercept range is the norm of the output argument **SRFVEC**. Units are km. We use the SPICELIB function **VNORM** to obtain the norm:

```
VNORM ( SRFVEC )
```

We'll write out the range data along with the other program results.
Compute the planetocentric latitude \((PCLAT)\) and longitude \((PCLON)\), as well as the planetodetic latitude \((PDLAT)\) and longitude \((PDLON)\) of the intersection point.

\[
\text{IF ( FOUND ) THEN}
\]
\[
\text{CALL RECLAT ( POINT, R, PCLON, PCLAT )}
\]
\[
\text{C Let RE, RP, and F be the satellite's longer equatorial C radius, polar radius, and flattening factor.}
\]
\[
\begin{align*}
\text{RE} & = \text{RADII}(1) \\
\text{RP} & = \text{RADII}(3) \\
F & = (\text{RE} - \text{RP}) / \text{RE}
\end{align*}
\]
\[
\text{CALL RECGEO ( POINT, RE, F, PDLON, PDLAT, ALT )}
\]

The illumination angles we want are the outputs of \texttt{ILUMIN}. Units are radians.

\[
\text{CALL ILUMIN ( 'Ellipsoid', SATNM, ET, FIXREF,}
\]
\[
\text{. \quad 'CN+S', SCNM, POINT, TRGEPC, SRFVEC,}
\]
\[
\text{. \quad PHASE, SOLAR, EMISSN )}
\]
CALL SINCPT ('Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, IFRAME, INSITE, POINT, TRGEPC, SRFVEC, FOUND)

C     Compute the boresight ray intersection with the surface of the target body.

C     If an intercept is found, compute planetocentric and planetodetic latitude and longitude of the point.

IF( FOUND ) THEN

    CALL RECLAT ( POINT, R, PCLON, PCLAT )

C     Let RE, RP, and F be the satellite's longer equatorial radius, polar radius, and flattening factor.

RE  =  RADII(1)
RP  =  RADII(3)
F   =  ( RE - RP ) / RE

CALL RECGEO ( POINT, RE, F, PDLON, PDLAT, ALT )

C     Compute illumination angles at the surface point.

CALL ILUMIN ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, POINT, TRGEPC, SRFVEC, PHASE, SOLAR, EMISSN )

ELSE

    ...

ENDIF
The code above used quite a few inputs that we don't have yet:

- TDB epoch of interest (ET)
- satellite and s/c names (SATNM, SCNM)
- satellite body-fixed frame name (FIXREF)
- satellite ellipsoid radii (RADII)
- instrument fixed frame name (IFRAME)
- instrument boresight vector in the instrument frame (INSITE)

Some of these values are user inputs others can be obtained via SPICELIB calls once the required kernels have been loaded.

Let's prompt for the satellite name (SATNM), satellite frame name (FIXREF), spacecraft name (SCNM), instrument name (INSTNM) and time of interest (TIME):

```fortran
CALL PROMPT ( 'Enter satellite name > ', SATNM )
CALL PROMPT ( 'Enter satellite frame > ', FIXREF )
CALL PROMPT ( 'Enter spacecraft name > ', SCNM )
CALL PROMPT ( 'Enter instrument name > ', INSTNM )
CALL PROMPT ( 'Enter time > ', TIME )
```
Then we can get the rest of the inputs from SPICELIB calls:

To get the TDB epoch (\( ET \)) from the user-supplied time string (which may refer to the UTC, TDB or TT time systems):

\[
\text{CALL STR2ET (TIME, ET )}
\]

To get the satellite’s ellipsoid radii (\( \text{RADII} \)):

\[
\text{CALL BODVRD (SATNM, 'RADII', 3, I, \text{RADII} )}
\]

To get the instrument boresight direction (\( \text{INSITE} \)) and the name of the instrument frame (\( \text{IFRAME} \)) in which it is defined:

\[
\text{CALL BODN2C (INSTNM, INSTID, FOUND )}
\]

\[
\text{IF ( .NOT. FOUND ) THEN}
\]
\[
\text{CALL SETMSG ( 'Instrument name # could not be ' //}
\]
\[
\text{ 'translated to an ID code.' )}
\]
\[
\text{CALL ERRCH ( '#', INSTNM )}
\]
\[
\text{CALL SIGERR ( 'NAMENOTFOUND' )}
\]
\[
\text{END IF}
\]

\[
\text{CALL GETFOV ( INSTID, ROOM, SHAPE, IFRAME,}
\]
\[
\text{INSITE, N, BUNDRY )}
\]
Getting Inputs: Summary

Prompt for the user-supplied inputs for our program.

CALL PROMPT ( 'Enter setup file name > ', SETUPF )
CALL FURNSH ( SETUPF )
CALL PROMPT ( 'Enter satellite name > ', SATNM )
CALL PROMPT ( 'Enter satellite frame > ', FIXREF )
CALL PROMPT ( 'Enter spacecraft name > ', SCNM )
CALL PROMPT ( 'Enter instrument name > ', INSTNM )
CALL PROMPT ( 'Enter time > ', TIME )

Get the epoch corresponding to the input time:

CALL STR2ET ( TIME, ET )

Get the radii of the satellite.

CALL BODVRD ( SATNM, 'RADII', 3, I, RADII )

Get the instrument boresight and frame name.

CALL BODN2C ( INSTNM, INSTID, FOUND )

IF ( .NOT. FOUND ) THEN
  CALL SETMSG ( 'Instrument name # could not be ' //
  'translated to an ID code.' )
  CALL ERRCH ( '#', INSTNM )
  CALL SIGERR ( 'NAMENOTFOUND' )
END IF

CALL GETFOV ( INSTID, ROOM, SHAPE, IFRAME, INSITE, N, BUNDRY )
Display Results

Navigation and Ancillary Information Facility

C Display results. Convert angles from radians to degrees
C for output.
WRITE ( *, '(1X,A,F12.6)')
   'Intercept planetocentric longitude (deg): ', DPR()*PCLON
WRITE ( *, '(1X,A,F12.6)')
   'Intercept planetocentric latitude  (deg): ', DPR()*PCLAT
WRITE ( *, '(1X,A,F12.6)')
   'Intercept planetodetic longitude  (deg): ', DPR()*PDLON
WRITE ( *, '(1X,A,F12.6)')
   'Intercept planetodetic latitude   (deg): ', DPR()*PDLAT
WRITE ( *, '(1X,A,F12.6)')
   'Range from spacecraft to intercept point (km): ',
   VNORM(SRFVEC)
WRITE ( *, '(1X,A,F12.6)')
   'Intercept phase angle (deg): ', DPR()*PHASE
WRITE ( *, '(1X,A,F12.6)')
   'Intercept solar incidence angle  (deg): ', DPR()*SOLAR
WRITE ( *, '(1X,A,F12.6)')
   'Intercept emission angle        (deg): ',
   DPR()*EMISSN
ELSE
   WRITE (*,*) 'No intercept point found at '//' TIME
END IF
To finish up the program we need to declare the variables we've used.

- We'll highlight techniques used by NAIF programmers
- Add remaining Fortran code required to make a syntactically valid program
Complete Source Code - 1

Writing a FORTRAN-based program

```fortran
PROGRAM PROG26
IMPLICIT NONE
DOUBLE PRECISION    EMISSN
DOUBLE PRECISION    ET
DOUBLE PRECISION    F
DOUBLE PRECISION    INSITE(3)
DOUBLE PRECISION    SRFVEC(3)
DOUBLE PRECISION    PCLAT
DOUBLE PRECISION    PCLON
DOUBLE PRECISION    PDLAT
DOUBLE PRECISION    PDLAT
DOUBLE PRECISION    PDLO
DOUBLE PRECISION    PHASE
DOUBLE PRECISION    POINT (3)
DOUBLE PRECISION    R
DOUBLE PRECISION    RADII (3)
DOUBLE PRECISION    RE
DOUBLE PRECISION    RP
DOUBLE PRECISION    SOLAR
DOUBLE PRECISION    TRGEPC
INTEGER             FILESZ
PARAMETER         ( FILESZ =    255 )
INTEGER             WORDSZ
PARAMETER         ( WORDSZ =     40 )
INTEGER             ROOM
PARAMETER         ( ROOM   =     10 )
CHARACTER*(WORDSZ)  IFRAME
CHARACTER*(WORDSZ)  INSTNM
CHARACTER*(WORDSZ)  SATNM
CHARACTER*(WORDSZ)  FIXREF
CHARACTER*(WORDSZ)  SCNM
CHARACTER*(FILESZ)  SETUPF
CHARACTER*(WORDSZ)  SHAPE
CHARACTER*(WORDSZ)  TIME
DOUBLE PRECISION    ALT
DOUBLE PRECISION    BUNDRY (3, ROOM) LOGICAL  FOUND
```
Prompt for the user-supplied inputs for our program.
CALL PROMPT ( 'Enter setup file name > ', SETUPF )
CALL FURNSH ( SETUPF )
CALL PROMPT ( 'Enter satellite name > ', SATNM )
CALL PROMPT ( 'Enter satellite frame > ', FIXREF )
CALL PROMPT ( 'Enter spacecraft name > ', SCNM )
CALL PROMPT ( 'Enter instrument name > ', INSTNM )
CALL PROMPT ( 'Enter time            > ', TIME )

Get the epoch corresponding to the input time:
CALL STR2ET ( TIME, ET )

Get the radii of the satellite.
CALL BODVRD ( SATNM, 'RADII', 3, I, RADII )

Get the instrument boresight and frame name.
CALL BODN2C ( INSTNM, INSTID, FOUND )
IF ( .NOT. FOUND ) THEN
    CALL SETMSG ( 'Instrument name # could not be ' //
                     'translated to an ID code.' )
    CALL ERRCH ( '#', INSTNM )
    CALL SIGERR ( 'NAMENOTFOUND' )
END IF
CALL GETFOV ( INSTID, ROOM, SHAPE, IFRAME, 
                     INSITE, N, BUNDRY )
C Compute the boresight ray intersection with the surface of the target body.
CALL SINCPT ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, IFRAME, 
.              INSITE, POINT, TRGEPC, SRFVEC, FOUND )
C If an intercept is found, compute planetocentric and planetodetic latitude and longitude of the point.
IF( FOUND ) THEN
C Let RE, RP, and F be the satellite's longer equatorial radius, polar radius, and flattening factor.
RE  =  RADII(1)
RP  =  RADII(3)
F   =  ( RE - RP ) / RE
CALL RECGEO ( POINT, RE, F, PDLON, PDLAT, ALT )
C Compute illumination angles at the surface point.
CALL ILUMIN ( 'Ellipsoid', SATNM, ET, FIXREF, 'CN+S', SCNM, 
.              POINT, TRGEPC, SRFVEC, PHASE, SOLAR, EMISSN )
C Display results. Convert angles from radians to degrees for output.
WRITE ( *, * )
WRITE ( *, '(1X,A,F12.6)' )
   'Intercept planetocentric longitude (deg): ', DPR()*PCLON
WRITE ( *, '(1X,A,F12.6)' )
  'Intercept planetocentric latitude (deg): ', DPR()*PCLAT
WRITE ( *, '(1X,A,F12.6)' )
  'Intercept planetodetic longitude (deg): ', DPR()*PDLOM
WRITE ( *, '(1X,A,F12.6)' )
  'Intercept planetodetic latitude (deg): ', DPR()*PDLAT
WRITE ( *, '(1X,A,F12.6)' )
  'Range from spacecraft to intercept point (km): ', VNORM(SRFVEC)
WRITE ( *, '(1X,A,F12.6)' )
  'Intercept phase angle (deg): ', DPR()*PHASE
WRITE ( *, '(1X,A,F12.6)' )
  'Intercept solar incidence angle (deg): ', DPR()*SOLAR
WRITE ( *, '(1X,A,F12.6)' )
  'Intercept emission angle (deg): ', DPR()*EMISSN

ELSE
  WRITE (*,*) 'No intercept point found at ' // TIME
END IF
END
• First be sure that both the SPICE Toolkit and a Fortran compiler are properly installed.
  – A "hello world" Fortran program must be able to compile, link, and run successfully in your environment.
  – Any of the mkproduc.* scripts in the toolkit/src/* paths of the SPICE Toolkit installation should execute properly.

• Ways to compile and link the program:
  – If you're familiar with the "make" utility, create a makefile. Use compiler and linker options from the mkproduc.* script found in the toolkit/src/cookbook path of your SPICE Toolkit installation.
  – Or, modify the cookbook mkproduc.* build script.
    » Your program name must be *.pgm, for example demo.pgm, to be recognized by the script.
    » Change the library references in the script to use absolute pathnames.
    » Change the path for the executable to the current working directory.
    » On some platforms, you must modify the script to refer to your program by name.
Or, compile the program on the command line. The program must be linked against the SPICELIB object library spicelib.a (spicelib.lib under MS Windows systems). On a PC running Linux and g77, if

» The g77 compiler is in your path
  • As indicated by the response to the command "which g77"

» the Toolkit is installed in the path (for the purpose of this example) /myhome/toolkit

» You've named the program demo.f

then you can compile and link your program using the command

```bash
$ g77 -o demo demo.f \
  /myhome/toolkit/lib/spicelib.a
```
Prompt> mkprodct.csh

Using the g77 compiler.

Setting default Fortran compile options:
-c -C

Setting default C compile options:
-c

Setting default link options:

Compiling and linking: demo.pgm
Compiling and linking: demo.pgm

Prompt>
It looks like we have everything taken care of:

- We have all necessary kernels
- We made a setup file (metakernel) pointing to them
- We wrote the program
- We compiled and linked it

Let's run it.
Running the Program - 2

Prompt> demo
Enter setup file name > setup.ker
Enter satellite name > PHOEBE
Enter satellite frame > IAU_PHOEBE
Enter spacecraft name > CASSINI
Enter instrument name > CASSINI_ISS_NAC
Enter time > 2004 jun 11 19:32:00

Intercept planetocentric longitude (deg): 39.843719
Intercept planetocentric latitude (deg): 4.195878
Intercept planetodetic longitude (deg): 39.843719
Intercept planetodetic latitude (deg): 5.048011
Range from spacecraft to intercept point (km): 2089.169724
Intercept phase angle (deg): 28.139479
Intercept solar incidence angle (deg): 18.247220
Intercept emission angle (deg): 17.858309
• Latitude definitions:
  - Planetocentric latitude of a point P: angle between segment from origin to point and x-y plane (red arc in diagram).
  - Planetodetic latitude of a point P: angle between x-y plane and extension of ellipsoid normal vector N that connects x-y plane and P (blue arc in diagram).