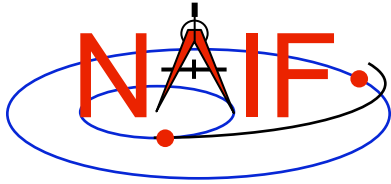


---

Navigation and Ancillary Information Facility

# Writing an Mice (MATLAB) Based Program

September 2009

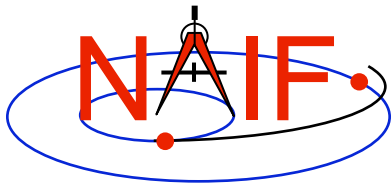


# Viewing This Tutorial

---

Navigation and Ancillary Information Facility

**Undefined variables are displayed in red; results are displayed in blue.**



# Introduction

---

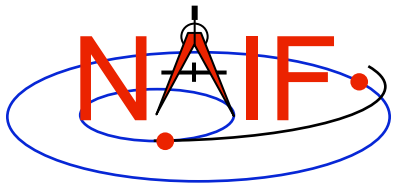
## Navigation and Ancillary Information Facility

**First, let's go over the important steps in the process of writing a Mice-based 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.**
- **Formulate an algorithm to compute the quantities of interest using SPICE.**
- **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.**



# Observation geometry

Navigation and Ancillary Information Facility

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)

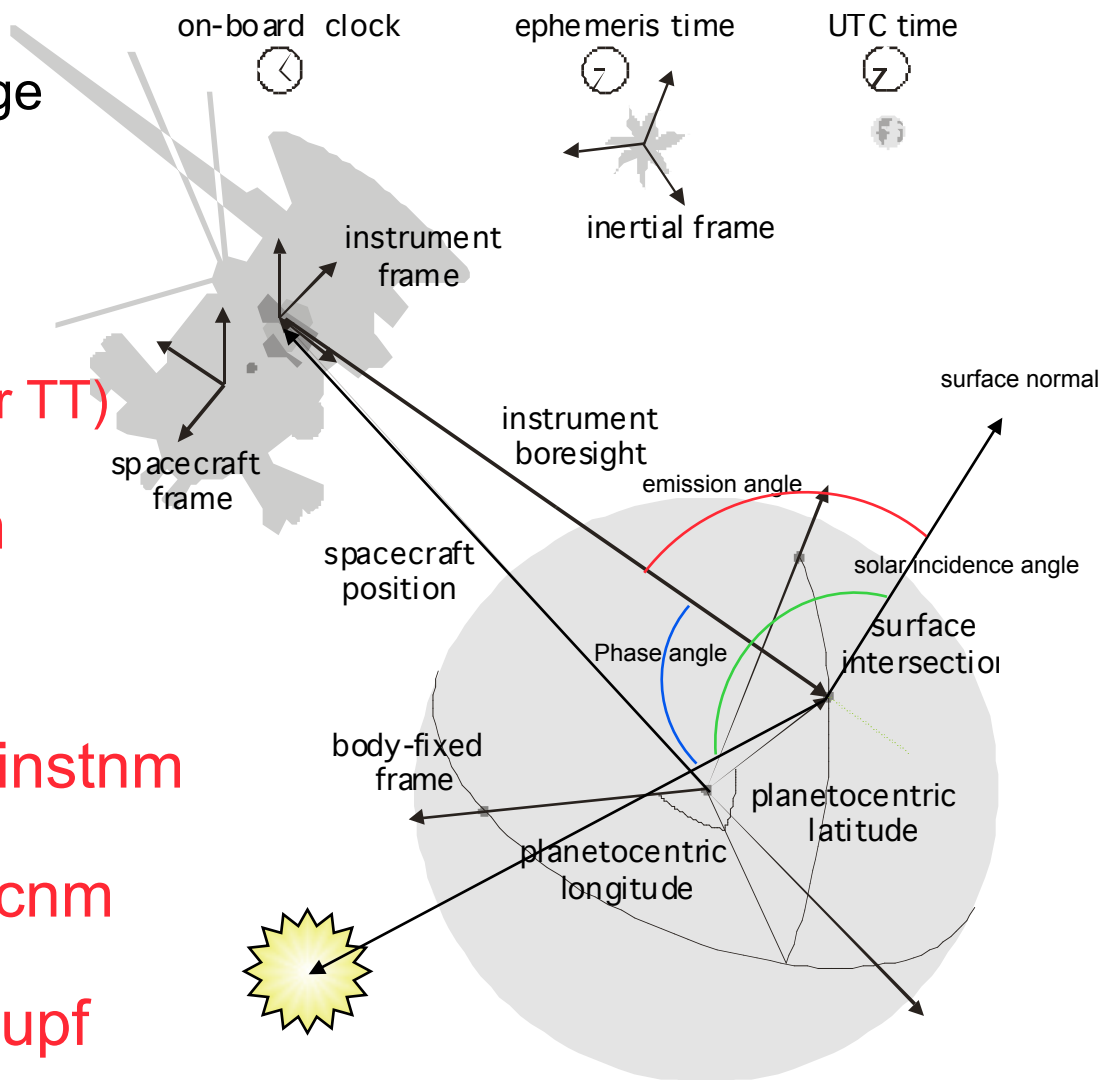
On what object? **satnm**

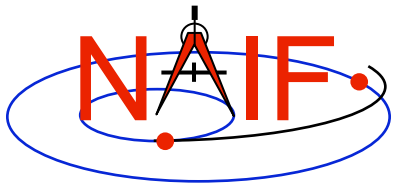
In what frame? **fixref**

For which instrument? **instnm**

For what spacecraft? **scnm**

Using what model? **setupf**





# Needed Data

Navigation and Ancillary Information Facility

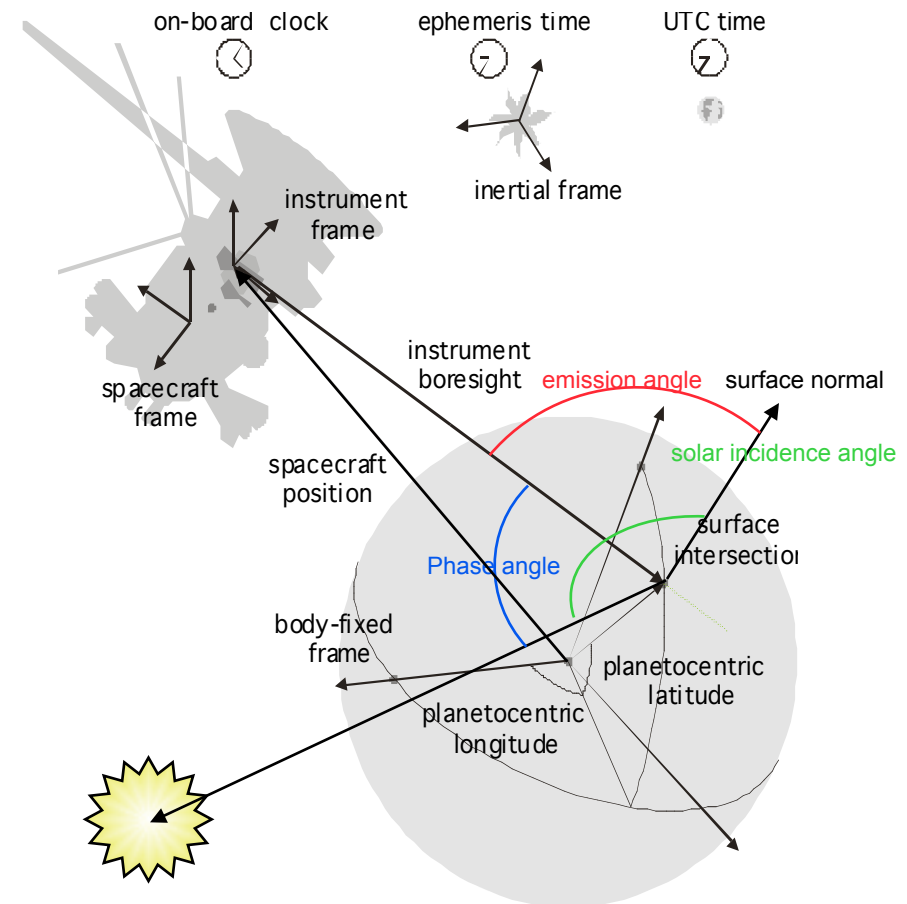
Time transformation kernels

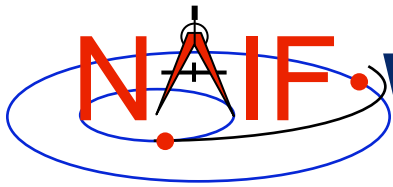
Orientation models

Instrument descriptions

Shapes of satellites, planets

Ephemerides for spacecraft,  
Saturn barycenter and satellites.





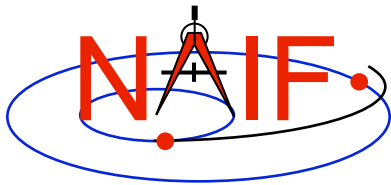
# Which Kinds of Kernels are Needed?

## Navigation and Ancillary Information Facility

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.

Parameter	Kernel Type	File name
-----	-----	-----
time conversions	generic LSK	naif0008.tls
	CASSINI SCLK	cas00084.tsc
satellite orientation	CASSINI PCK	cpck05Mar2004.tpc
satellite shape	CASSINI PCK	cpck05Mar2004.tpc
satellite position	planet/sat	
	ephemeris SPK	020514_SE_SAT105.bsp
planet barycenter position	planet SPK	981005_PLTEPH-DE405S.bsp
spacecraft position	spacecraft SPK	030201AP_SK_SM546_T45.bsp
spacecraft orientation	spacecraft CK	04135_04171pc_psiv2.bc
instrument alignment	CASSINI FK	cas_v37.tf
instrument boresight	Instrument IK	cas_iss_v09.ti



# Load kernels

Navigation and Ancillary Information Facility

The easiest and most flexible way to make these kernels available to the program is via `cspice_furnsh`. 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.

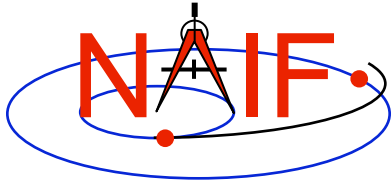
```
\begindata
```

```
KERNELS_TO_LOAD = ('naif0008.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:

```
setupf = input('Enter setup file name > ', 's');  
cspice_furnsh( setupf )
```



# Programming Solution

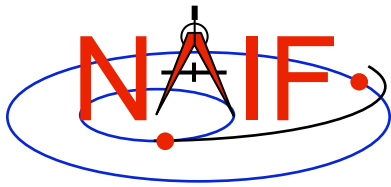
---

## Navigation and Ancillary Information Facility

- 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 Mice 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 intersection 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 first.





# Compute surface intercept

Navigation and Ancillary Information Facility

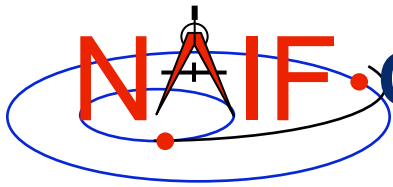
Compute the intercept point (**point**) of the boresight vector (**insite**) specified in the instrument frame (**iframe**) of the instrument mounted on the spacecraft (**scnm**) with the surface of the satellite (**satnm**) at the TDB time of interest (**et**) in the satellite's body-fixed frame (**fixref**). 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.

```
[point, trgepc, srfvec, found] = cspice_sincpt( ...  
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, insite );
```

The range we want is obtained from the outputs of `cspice_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 MATLAB function `norm` to obtain the norm:

```
norm( srfvec )
```

We'll write out the range data along with the other program results.



# Compute Lat/Lon and Illumination Angles

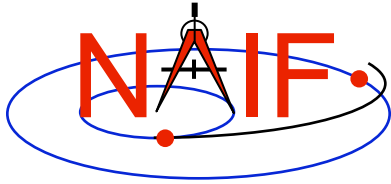
Navigation and Ancillary Information Facility

Compute the planetocentric latitude (**pclat**) and longitude (**pclon**), as well as the planetodetic latitude (**pdlat**) and longitude (**pdlon**) of the intersection point.

```
if ( found )  
    [r, pclon, pclat] = cspice_reclat( point );  
  
    % 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;  
  
    [pdlat, pdlon, alt] = cspice_recgeo( point, re, f );
```

The illumination angles we want are the outputs of `cspice_ilumin`. Units are radians.

```
[trgepc, srfvec, phase, solar, emissn] = cspice_ilumin( ...  
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, point );
```

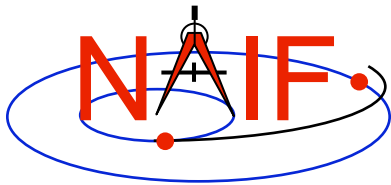


# Geometry Calculations: Summary

---

## Navigation and Ancillary Information Facility

```
% Compute the boresight ray intersection with the surface of the
% target body.
[point, trgepc, srfvec, found] = cspice_sincpt( ...
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, insite );
% If an intercept is found, compute planetocentric and planetodetic
% latitude and longitude of the point.
if ( found )
    [r, pclon, pclat] = cspice_reclat( point );
    % 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;
    [pdlon, pdlat, alt] = cspice_recgeo( point, re, f );
    % Compute illumination angles at the surface point.
    [trgepc, srfvec, phase, solar, emissn] = cspice_ilumin( ...
        'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, point );
    ...
else
    ...
```



# Get inputs - 1

Navigation and Ancillary Information Facility

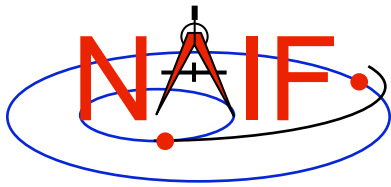
The code above used quite a few inputs that we don't have yet:

- 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 CSPICE 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`):

```
satnm = input( 'Enter satellite name > ', 's');  
fixref = input( 'Enter satellite frame > ', 's');  
scnm = input( 'Enter spacecraft name > ', 's');  
instnm = input( 'Enter instrument name > ', 's');  
time = input( 'Enter time > ', 's');
```



## Get Inputs - 2

---

### Navigation and Ancillary Information Facility

Then we can get the rest of the inputs from Mice calls:

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

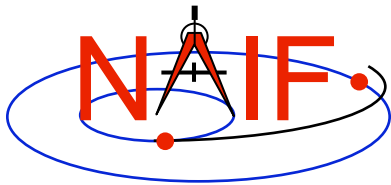
```
et = cspice_str2et( time );
```

To get the satellite's ellipsoid radii (**radii**):

```
radii = cspice_bodvrd( satnm, 'RADII', 3 );
```

To get the instrument boresight direction (**insite**) and the name of the instrument frame (**iframe**) in which it is defined:

```
[instid, found] = cspice_bodn2c( instnm );  
if ( ~found )  
    txt = sprintf( 'Unable to determine ID for instrument: %d', ...  
                   instnm );  
    error(txt)  
end  
  
[shape, iframe, insite, bundry] = cspice_getfov( instid, ROOM );
```



# Getting inputs: summary

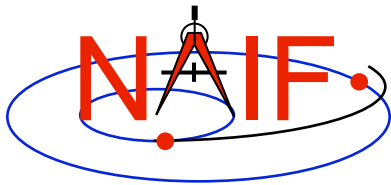
## Navigation and Ancillary Information Facility

```
% Prompt for the user-supplied inputs for our program.
setupf = input( 'Enter setup file name > ', 's');
cspice_furnsh( setupf )
satnm   = input( 'Enter satellite name > ', 's');
fixref  = input( 'Enter satellite frame > ', 's');
scnm    = input( 'Enter spacecraft name > ', 's');
instnm  = input( 'Enter instrument name > ', 's');
time    = input( 'Enter time                > ', 's');

% Get the epoch corresponding to the input time:
et = cspice_str2et( time );

% Get the radii of the satellite.
radii = cspice_bodvrd( satnm, 'RADII', 3 );

% Get the instrument boresight and frame name.
[instid, found] = cspice_bodn2c( instnm );
if ( ~found )
    txt = sprintf( 'Unable to determine ID for instrument: %d', ...
                   instnm );
    error(txt)
end
[shape, iframe, insite, bundry] = cspice_getfov( instid, ROOM );
```



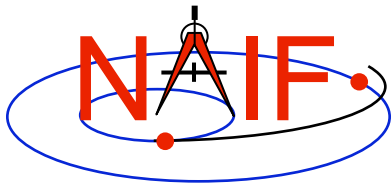
# Display results

## Navigation and Ancillary Information Facility

```

...
% Display results.  Convert angles from radians to degrees
% for output.
fprintf( 'Intercept planetocentric longitude      (deg):  %11.6f\n', ...
        R2D*pclon )
fprintf( 'Intercept planetocentric latitude      (deg):  %11.6f\n', ...
        R2D*pclat )
fprintf( 'Intercept planetodetic longitude      (deg):  %11.6f\n', ...
        R2D*pdlon )
fprintf( 'Intercept planetodetic latitude      (deg):  %11.6f\n', ...
        R2D*pdlat )
fprintf( 'Range from spacecraft to intercept point (km): %11.6f\n', ...
        norm(srfvec) )
fprintf( 'Intercept phase angle                (deg):  %11.6f\n', ...
        R2D*phase )
fprintf( 'Intercept solar incidence angle      (deg):  %11.6f\n', ...
        R2D*solar )
fprintf( 'Intercept emission angle            (deg):  %11.6f\n', ...
        R2D*emissn )
else
    disp( ['No intercept point found at ' time ] )
end

```



# Complete the program

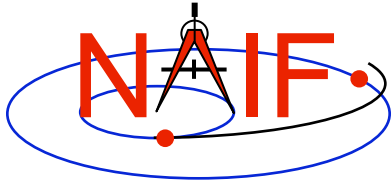
Navigation and Ancillary Information Facility

To finish up the program we need to declare the variables we've used.

- We'll highlight techniques used by NAIF programmers
- Add remaining MATLAB code required to make a syntactically valid program

```
ABCORR = 'CN+S';  
ROOM   = 10;  
R2D    = cspice_dpr;  
  
% Prompt for the user-supplied inputs for our program.  
setupf = input( 'Enter setup file name > ', 's');  
cspice_furnsh( setupf )  
  
satnm  = input( 'Enter satellite name > ', 's');  
fixref = input( 'Enter satellite frame > ', 's');  
scnm   = input( 'Enter spacecraft name > ', 's');  
instnm = input( 'Enter instrument name > ', 's');  
time   = input( 'Enter time > ', 's');
```





# Complete source code - 1

## Navigation and Ancillary Information Facility

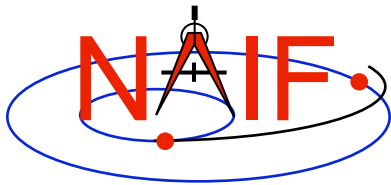
```
% Get the epoch corresponding to the input time:
et = cspice_str2et( time );

% Get the radii of the satellite.
radii = cspice_bodvrd( satnm, 'RADII', 3 );

% Get the instrument boresight and frame name.
[instid, found] = cspice_bodn2c( instnm );

if ( ~found )
    txt = sprintf( 'Unable to determine ID for instrument: %d', ...
                  instnm );
    error(txt)
end

[shape, iframe, insite, bundry] = cspice_getfov( instid, ROOM );
```



## Complete source code - 2

### Navigation and Ancillary Information Facility

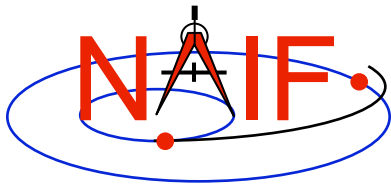
```
% Compute the boresight ray intersection with the surface of the
% target body.
[point, trgepc, srfvec, found] = cspice_sincpt( ...
    'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, iframe, insite );

% If an intercept is found, compute planetocentric and planetodetic
% latitude and longitude of the point.
if ( found )
    [r, pclon, pclat] = cspice_reclat( point );

    % 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;

    [pdlon, pdlat, alt] = cspice_recgeo( point, re, f );

    % Compute illumination angles at the surface point.
    [trgepc, srfvec, phase, solar, emissn] = cspice_ilumin( ...
        'Ellipsoid', satnm, et, fixref, 'CN+S', scnm, point );
```



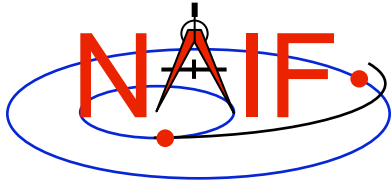
# Complete source code - 3

## Navigation and Ancillary Information Facility

```
% Display results.  Convert angles from radians to degrees
% for output.
fprintf( 'Intercept planetocentric longitude      (deg):  %11.6f\n', ...
        R2D*pclon )
fprintf( 'Intercept planetocentric latitude      (deg):  %11.6f\n', ...
        R2D*pclat )
fprintf( 'Intercept planetodetic longitude      (deg):  %11.6f\n', ...
        R2D*pdlon )
fprintf( 'Intercept planetodetic latitude      (deg):  %11.6f\n', ...
        R2D*pdlat )
fprintf( 'Range from spacecraft to intercept point (km): %11.6f\n', ...
        norm(srfvec) )
fprintf( 'Intercept phase angle                (deg):  %11.6f\n', ...
        R2D*phase )
fprintf( 'Intercept solar incidence angle      (deg):  %11.6f\n', ...
        R2D*solar )
fprintf( 'Intercept emission angle            (deg):  %11.6f\n', ...
        R2D*emissn )

else
    disp( ['No intercept point found at ' time ]
end

% Unload the kernels and clear the kernel pool
cspice_kclear
```



# Running the program

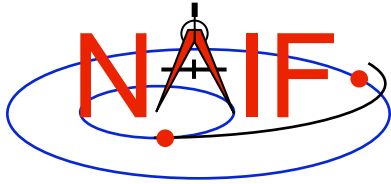
---

Navigation and Ancillary Information Facility

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

Let's run it.



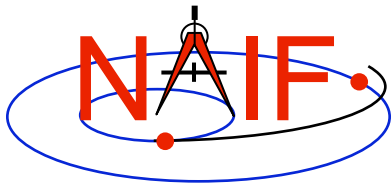
# Running the program

Navigation and Ancillary Information Facility

```
Terminal Window

>>  prog_geometry
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
```



# Backup

Navigation and Ancillary Information Facility

- **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).

