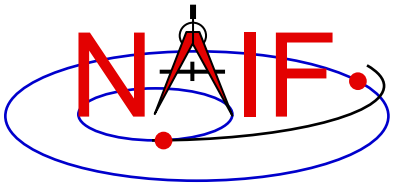
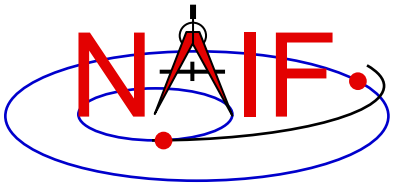


Navigation and Ancillary Information Facility

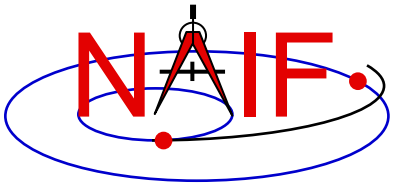
“Mice”
The MATLAB© Interface to CSPICE

January 2020



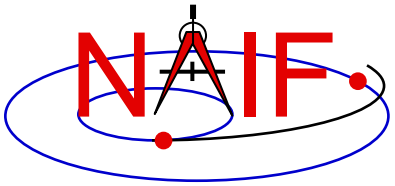
Topics

- Mice Benefits
- How does it work?
- Distribution
- Mice Operation
- Vectorization
- Simple Mice Examples



Mice Benefits

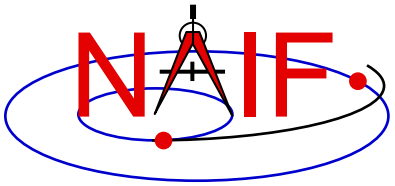
- Mice operates as an extension to the MATLAB environment.
- All Mice calls are functions regardless of the call format of the underlying CSPICE routine, returning MATLAB native data types.
- Mice has some capability not available in CSPICE such as vectorization.
- CSPICE error messages return to MATLAB in the form usable by the *try...catch* construct.



How Does It Work? (1)

- The MATLAB environment includes an intrinsic capability to use external routines.
  - Mice functions as a MATLAB executable, MEX, consisting of the Mice MEX shared object library and a set of .m wrapper files.
    - » The Mice library contains the MATLAB callable C interface routines that wrap a subset of CSPICE wrapper calls.
    - » The wrapper files, named cspice_*.m and mice_*.m, provide the MATLAB calls to the interface functions.
      - » A function prefixed with ‘cspice_’ retains essentially the same argument list as the CSPICE counterpart.
      - » An interface prefixed with ‘mice_’ returns a structure, with the fields of the structure corresponding to the output arguments of the CSPICE counterpart.
    - » The wrappers include a header section describing the function call, displayable by the MATLAB *help* command.

continued on next page

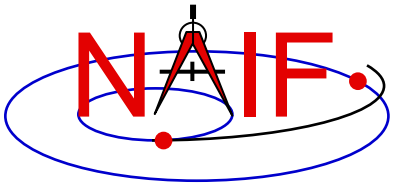


How Does It Work? (2)

When a user invokes a call to a Mice function:

1. MATLAB calls…
   2. the function's wrapper, which calls…
      3. the Mice MEX shared object library, which performs its function then returns the result…
         4. to the wrapper, which…
            5. returns the result to the user

… transparent from the user's perspective.



Mice Distribution

- **NAIF distributes Mice as a complete, standalone package.**
- **The package includes:**
  - **the CSPICE source files**
  - **the Mice interface source code**
  - **platform specific build scripts for Mice and CSPICE**
  - **MATLAB versions of the SPICE cookbook programs, *states*, *tictoc*, *subpt*, and *simple***
  - **an HTML-based help system for both Mice and CSPICE, with the Mice help cross-linked to CSPICE**
  - **the Mice MEX shared library and the M wrapper files. The system is ready for use after installation of the library and wrapper files.**
- **You do not need a C compiler to use Mice.**

Mice Operation (1)

- **A possible irritant exists in loading kernels using the `cspice_furnsh` function.**
  - **Kernels load into your MATLAB session, not into your MATLAB scripts. This means:**
    - » **loaded binary kernels remain accessible ("active") throughout your MATLAB session**
    - » **data from loaded text kernels remain in the kernel pool (in the memory space used by CSPICE) throughout your MATLAB session**
  - **Consequence: some kernel data may be available to one of your scripts even though not intended to be so.**
    - » **You could get incorrect results!**
    - » **If you run only one script during your MATLAB session, there's no problem.**

continued on next page

Mice Operation (2)

- **Mitigation: two approaches**
  - **Load all needed SPICE kernels for your MATLAB session at the beginning of the session, paying careful attention to the files loaded and the loading order (loading order affects precedence)**
    - » **Convince yourself that this approach will provide ALL of the scripts you will run during this MATLAB session with the appropriate SPICE data**
  - **At or near the end of every MATLAB script:**
    - » **include a call to `cspice_unload` for each kernel loaded using `cspice_furnsh`**
    - » **or include a call to `cspice_kclear` to remove ALL kernel data from the kernel pool loaded using `cspice_furnsh`**

Mice Vectorization (1)

- **Most Mice functions include use of vectorized arguments, a capability not available in C or Fortran toolkits.**
- **Example: use Mice to retrieve state vectors and light-time values for 1000 ephemeris times.**
  - **Create an array of 1000 ephemeris times with a step size of 10 hours, starting from July 1, 2005:**

```
start = cspice_str2et('July 1 2005');
et    = (0:999)*36000 + start;
```

  - **Retrieve the state vectors and corresponding light times from Mars to earth at each `et` in the J2000 frame with LT+S aberration correction:**

```
[state, ltime] = cspice_spkezr( 'Earth', et, 'J2000', 'LT+S', 'MARS');
```

or

```
starg = mice_spkezr( 'Earth', et, 'J2000', 'LT+S', 'MARS');
```

continued on next page

Mice Vectorization (2)

– **Access the *ith* state 6-vector (6x1 array) corresponding to the *ith* ephemeris time with the expression**

```
state_i = state(:,i)
```

or

```
state_i = starg(i).state
```

- **Convert the ephemeris time vector `et` from the previous example to UTC calendar strings with three decimal places of precision in the seconds field.**

```
format = 'C';
prec   = 3;
utcstr = cspice_et2utc( et, format, prec );
```

– **The call returns `utcstr`, an array of 1000 strings (dimensioned 1000x24), where each *ith* string is the calendar date corresponding to `et(i)`.**

continued on next page

Mice Vectorization (3)

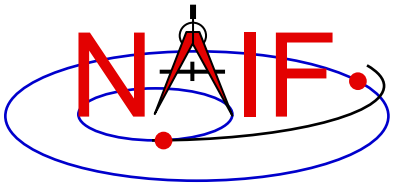
- Access the *ith* string of utcstr corresponding to the *ith* ephemeris time with the expression

```
utcstr_i = utcstr(i,:)
```

- **Convert the position components (the first three components in a state vector) of the N state vectors returned in `state` by the `cspice_spkezr` function to latitudinal coordinates.**

```
[radius, latitude, longitude] = cspice_reclat( state(1:3,:) );
```

- **The call returns three double precision 1x1000 arrays (vectorized scalars):** `radius, latitude, longitude.`



Simple Mice Example (1)

- **As an example of using Mice, calculate and plot the trajectory of the Cassini spacecraft, in the J2000 inertial frame, from June 20 2004 to December 1 2005. This example uses the `cspice_spkpos` function to retrieve position data.**

```
% Construct a meta kernel, "standard.tm", which will be used to load the needed
% generic kernels:  "naif0011.tls," "de421.bsp," and "pck00010.tpc."

% Load the generic kernels using the meta kernel, and a Cassini spk.

cspice_furnsh( { 'standard.tm', '/kernels/cassini/spk/030201AP_SK_SM546_T45.bsp'} )

% Define the number of divisions of the time interval.
STEP       = 1000;
et         = cspice_str2et( {'Jun 20, 2004', 'Dec 1, 2005'} );
times      = (0:STEP-1) * ( et(2) - et(1) )/STEP + et(1);

[pos,ltime]= cspice_spkpos( 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER' );

% Plot the resulting trajectory.
x = pos(1,:);
y = pos(2,:);
z = pos(3,:);

plot3(x,y,z)

cspice_kclear
```

continued on next page

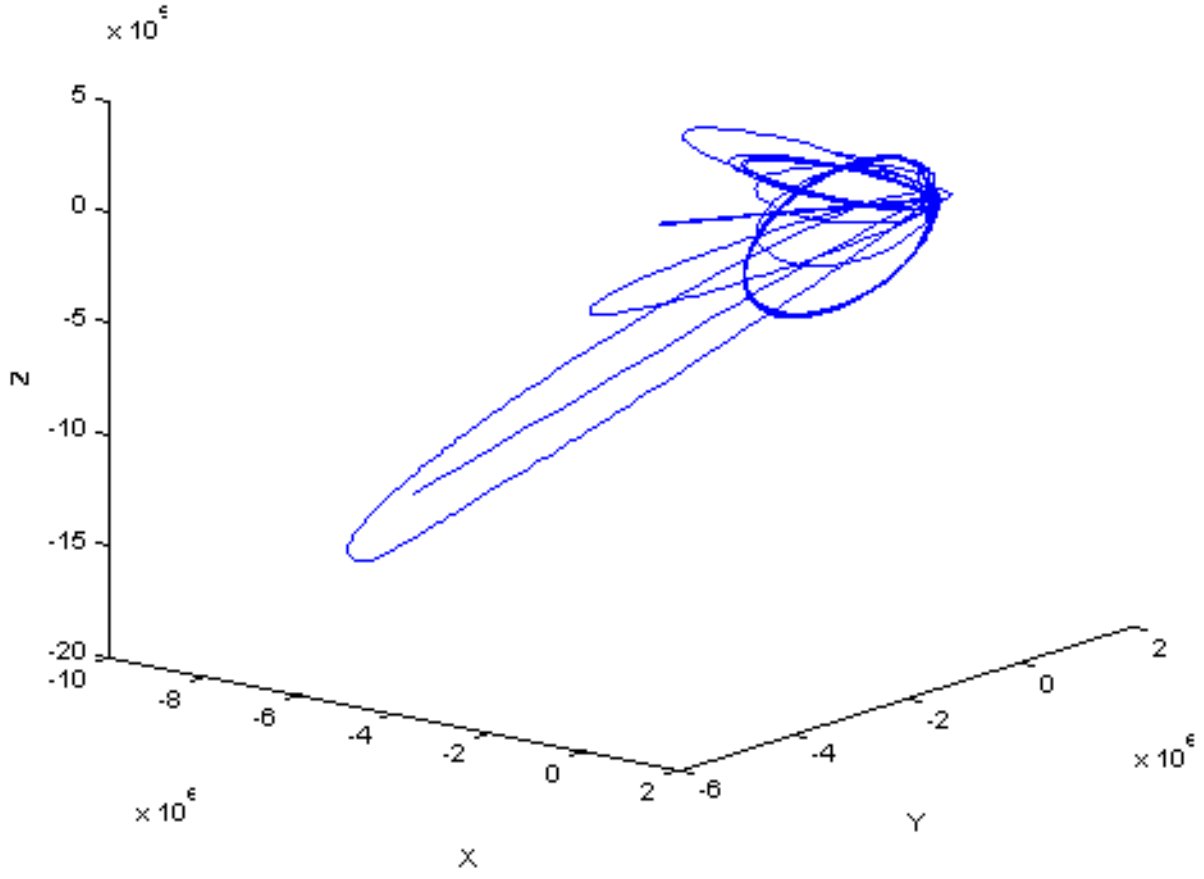
Simple Mice Example (2)

- **Repeat the example of the previous page, except use the `mice_spkezr` function to retrieve full state vectors.**

```
% Define the number of divisions of the time interval.
STEP = 1000;

% Construct a meta kernel, "standard.tm", which will be used to load the needed
% generic kernels:  "naif0009.tls," "de421.bsp," and "pck00009.tpc."

% Load the generic kernels using the meta kernel, and a Cassini spk.

cspice_furnsh( { 'standard.tm', '/kernels/cassini/spk/030201AP_SK_SM546_T45.bsp'} )

et    = cspice_str2et( {'Jun 20, 2004', 'Dec 1, 2005'} );
times = (0:STEP-1) * ( et(2) - et(1) )/STEP + et(1);

ptarg = mice_spkpos( 'Cassini', times, 'J2000', 'NONE', 'SATURN BARYCENTER' );
pos   = [ptarg.pos];

% Plot the resulting trajectory.
x = pos(1,:);
y = pos(2,:);
z = pos(3,:);

plot3(x,y,z)

cspice_kclear
```

Mice Example Graphic Output

Navigation and Ancillary Information Facility



Trajectory of the Cassini spacecraft, in the J2000 frame, from June 20 2004 to Dec 1 2005