A MATLAB Script for Terrestrial and Celestial Coordinate Conversion

This document describes two MATLAB functions and a companion demonstration script that can be used to rotate a vector to and from the terrestrial and celestial systems. The first function transforms a vector in the International Terrestrial Reference System (ITRS; a rotating earth-fixed system) to the Geocentric Celestial Reference System (GCRS; a local space-fixed system) by applying rotations for polar motion, earth rotation, nutation, precession, and the dynamical-to-gcrs frame tie. The second MATLAB function performs the GCRS-to-ITRS transformation.

The transformation used to relate the ITRS to the GCRS at a date *t* of the observation is given by

$$[GCRS] = Q(t)R(t)W(t)[ITRS]$$

where $Q(t), R(t),$ and $W(t)$ are the transformations matrices due to the effects of the motion of the celestial pole in the celestial reference system, the rotation of the Earth around the axis associated with the pole, and the polar motion, respectively.

Additional information about these methods can be found in Chapter 5 of IERS Technical Note No. 36, "Transformation between the International Terrestrial Reference System and the Geocentric Celestial Reference System" which is part of the IERS Conventions (2010). Another excellent resource is "The IAU Resolutions on Astronomical Reference Systems, Times Scales, and Earth Rotation Models", United States Naval Observatory Circular No. 179, October 20, 2005.

These functions were ported to MATLAB using the Fortran source code provided with NOVAS 3.1 and documented in "User's Guide to NOVAS Version F3.1", U.S. Naval Observatory, March 2011. Please note that this MATLAB script computes the accumulated precession in right ascension using the analytic equation given in Capitaine, et al., Astronomy and Astrophysics, **412**, 567-586, (2003) equation (42). This implementation is provided by support functions `cioloc.m` and `eqxra.m`.

Script example

The companion MATLAB script that illustrates how to interact with these two functions is named `demo_tercel.m`. The following is the script output for the example described in USNO/AA Technical Note 2010-04, "Testing Coordinate Frame Transformations". The script is "hard-wired" with numerical data for this example and uses the equinox-based method with full accuracy.

```
demo_tercel

rotates a vector from the terrestrial to the celestial system
-------------------------------------------------------------

equinox-based method, full accuracy

UT1 calendar date         05-Apr-2007

UT1 time                  11:59:59.928

UT1 Julian date           2454195.99999917
```

```
position vector, geocentric equatorial rectangular
coordinates, referred to ITRS axes (terrestrial system)

 +1.00000000000000e+000  +0.00000000000000e+000  +0.00000000000000e+000


position vector, geocentric equatorial rectangular
coordinates, referred to GCRS axes (celestial system)

 +9.73104317701089e-001  +2.30363826224119e-001  -7.03163482963017e-004

terrestrial-to-celestial transformation matrix

 +9.73104317701089e-001  +2.30363826224119e-001  -7.03163482963017e-004
 -2.30363800441031e-001  +9.73104570636353e-001  +1.18545358540577e-004
 +7.11560161552747e-004  +4.66264120217281e-005  +9.99999745754025e-001

position vector, geocentric equatorial rectangular
coordinates, referred to ITRS axes (terrestrial system)

 +9.99999999999943e-001  +3.05884982816352e-008  +3.38673387753519e-007
```

The following is the script output for this example using the CIO-based method with full accuracy.

```
demo_tercel

rotates a vector from the terrestrial to the celestial system
-------------------------------------------------------------

cio-based method, full accuracy

UT1 calendar date         05-Apr-2007

UT1 time                  11:59:59.928

UT1 Julian date           2454195.99999917

position vector, geocentric equatorial rectangular
coordinates, referred to ITRS axes (terrestrial system)

 +1.00000000000000e+000  +0.00000000000000e+000  +0.00000000000000e+000

position vector, geocentric equatorial rectangular
coordinates, referred to GCRS axes (celestial system)

 +9.73104317701090e-001  +2.30363826224111e-001  -7.03163482961698e-004

terrestrial-to-celestial transformation matrix

 +9.73104317701090e-001  +2.30363826224111e-001  -7.03163482961698e-004
 -2.30363800441025e-001  +9.73104570636355e-001  +1.18545358546144e-004
 +7.11560161552747e-004  +4.66264120217281e-005  +9.99999745754025e-001

position vector, geocentric equatorial rectangular
coordinates, referred to ITRS axes (terrestrial system)

 +9.99999999999942e-001  +3.05884872071636e-008  +3.38673389072703e-007
```

Each row of the transformation matrix was created by calling the ITRS-to-GCRS function three times with the following set of input vectors; `v1 = [1 0 0];` `v2 = [0 1 0];` `v3 = [0 0 1].`

The main script includes the option to set the method and accuracy of the numerical calculations. The following is a summary of these different modes.

```
% set mode = 0 for cio-based method, full accuracy
% set mode = 1 for cio-based method, reduced accuracy
% set mode = 2 for equinox-based method, full accuracy
% set mode = 3 for equinox-based method, reduced accuracy
```

The desired calculation mode can be set by editing the following line of MATLAB source code;

```
setmod(2);
```

MATLAB functions

The following is the syntax for the MATLAB function that performs the ITRS-to-GCRS coordinate conversion.

```
function vec2 = tercel (tjdh, tjdl, xp, yp, vec1)

% this function rotates a vector from the terrestrial to the
% celestial system. specifically, it transforms a vector in the
% itrs (a rotating earth-fixed system) to the gcrs (a local
% space-fixed system) by applying rotations for polar motion,
% earth rotation, nutation, precession, and the dynamical-to-gcrs
% frame tie.

%  tjdh   = ut1 julian date, high-order part (in)

%  tjdl   = ut1 julian date, low-order part (in)
%           the julian date may be split at any point, but
%           for highest precision, set tjdh to be the integral
%           part of the julian date, and set tjdl to be the
%           fractional part

%  xp     = conventionally-defined x coordinate of celestial
%           intermediate pole with respect to itrs pole, in arcseconds (in)

%  yp     = conventionally-defined y coordinate of celestial
%           intermediate pole with respect to itrs pole, in arcseconds (in)

%  vec1   = position vector, geocentric equatorial rectangular
%           coordinates, referred to itrs axes (terrestrial system) (in)

%  vec2   = position vector, geocentric equatorial rectangular
%           coordinates, referred to gcrs axes (celestial system) (out)

% note 1:  set xp = yp = 0.0d0 to eliminate polar motion rotation.

% note 2:  see also function setdt to set the value of delta-t
%          (delta-t = tt - ut1) to be used here.

% note 3:  both tjdh and tjdl should be non-negative for normal use
%          (tjdl=0.0d0 is ok). a negative value of tjdh is used to invoke a
```

```
%          special option where the output vector is produced with respect
%          to the equator and equinox of date, and the date for which the
%          transformation applies is taken from tjdl only. this option

%          works only in 'equinox' mode.

% note 4: input parameters xp, yp were xpole, ypole in novas f3.0.
%         the names were changed for consistancy throughout novas and with
%         iers conventions.
```

The following is the syntax for the MATLAB function that performs the GCRS-to-ITRS coordinate conversion.

```
function vec2 = celter (tjdh, tjdl, xp, yp, vec1)

% this function rotates a vector from the celestial to the
% terrestrial system.  specifically, it transforms a vector in the
% gcrs (a local space-fixed system) to the itrs (a rotating
% earth-fixed system) by applying rotations for the gcrs-to-
% dynamical frame tie, precession, nutation, earth rotation,
% and polar motion.

%      tjdh   = ut1 julian date, high-order part (in)

%      tjdl   = ut1 julian date, low-order part (in)
%               the julian date may be split at any point, but
%               for highest precision, set tjdh to be the integral
%               part of the julian date, and set tjdl to be the
%               fractional part

%      xp     = conventionally-defined x coordinate of celestial
%               intermediate pole with respect to itrs pole,
%               in arcseconds (in)

%      yp     = conventionally-defined y coordinate of celestial
%               intermediate pole with respect to itrs pole,
%               in arcseconds (in)

%      vec1   = position vector, geocentric equatorial rectangular
%               coordinates, referred to gcrs axes (celestial
%               system) (in)

%      vec2   = position vector, geocentric equatorial rectangular
%               coordinates, referred to itrs axes (terrestrial
%               system) (out)

% note 1:  set xp = yp = 0.0d0 to eliminate polar motion rotation.

% note 2:  see also subroutine setdt to set the value of delta-t
%          (delta-t = tt - ut1) to be used here.

% note 3:  both tjdh and tjdl should be non-negative for normal use
%          (tjdl=0.0d0 is ok). a negative value of tjdh is used to invoke a
%          special option where the input vector is assumed to be with
%          respect to the equator and equinox of date, and the date for which
%          the transformation applies is taken from tjdl only. this option
%          works only in 'equinox' mode.
```

Algorithm resources

NOVAS (Naval Observatory Vector Astrometry Subroutines), U. S. Naval Observatory, http://www.usno.navy.mil/USNO/astronomical-applications/software-products/novas.

Explanatory Supplement to the Astronomical Almanac, Edited by P. K. Seidelmann, University Science Books, 1992.

Gérard Petit and Brian Luzum (eds.), "IERS Conventions (2010)", IERS Technical Note 36.

J. H. Lieske, "Precession Matrix Based on IAU (1976) System of Astronomical Constants", *Astronomy and Astrophysics*, 73, 282-284 (1979)

"Numerical Expressions for Precession Formulae and Mean Elements for the Moon and Planets", *Astronomy and Astrophysics*, **282**: 663-683

D. McCarthy and B. Luzum, "An Abridged Model of the Precession-Nutation of the Celestial Pole", *Celestial Mechanics and Dynamical Astronomy*, **85**: 37-49, 2003

N. Capitaine, P. T. Wallace, and J. Chapront, "Expressions for IAU 2000 Precession Quantities", *Astronomy and Astrophysics*, 412, 567-586 (2003)

"SOFA Tools for Earth Attitude", Software version 4, Document version 1.2, IAU, September 5, 2010.

"SOFA Time Scales and Calendar Tools", Software version 1, Document version 1.1, IAU, December 24, 2010.

V. Coppola, J. Seago and D. Vallado, "The IAU 2000A and IAU 2006 Precession-Nutation Theories and Their Implementation", AAS 09-159, American Astronautical Society.

L. Barker, A. Dorsey and N. Stamatakos, "Validation of the IAY2000A/IAU2006 Frame Transformation Implementation", AAS 10-054, American Astronautical Society.

D. Vallado, J. Seago and P. Seidelmann, "Implementation Issues Surrounding the New IAU Reference Systems for Astrodynamics", AAS 06-134, American Astronautical Society.

G. Kaplan, "Nutation Series Evaluation in NOVAS 3.0", United States Naval Observatory Circular No. 181, December 15, 2009.

G. Kaplan, "The IAU Resolutions on Astronomical Reference Systems, Times Scales, and Earth Rotation Models", United States Naval Observatory Circular No. 179, October 20, 2005.

A. Monet, G. Kaplan and W. Harris, "Testing Coordinate Frame Transformations NOVAS vs SOFA", USNO/AA Technical Note 2010-04, July 14, 2010.

N. Capitaine and P. T. Wallace, "Precession-nutation Procedures Consistent with IAU 2006 Resolutions", *Astronomy and Astrophysics*, 459, 981-985 (2006)

Appendix A

Precession and Nutation

This appendix summarizes the numerical methods used to compute precession and nutation in this MATLAB script. The algorithms used to compute precession and nutation were ported to MATLAB using the Fortran version of NOVAS 3.1.

Precession

Precession is the slow drift of the Earth's rotational axis due mainly to the gravitational attraction of the Sun and Moon. The precession matrix transforms coordinates referred to the mean Earth equator and equinox of J2000 to coordinates measured with respect to the mean Earth equator and equinox of date.

The precession algorithm used in this MATLAB script is based on the method described in "Expressions for IAU 2000 Precession Quantities", by N. Capitaine, P. T. Wallace, and J. Chapront, *Astronomy and Astrophysics*, 412, 567-586 (2003).

The precession matrix is determined from the following transformation

$$\mathbf{P} = R_3(\chi_A) \bullet R_1(-\omega_A) \bullet R_3(-\psi_A) \bullet R_1(\varepsilon_0)$$

The precession angles are given by

$$\psi_A = 5038''.481507t - 1''.0790069t^2 - 0''.0011404t^3 + 0''.000132851t^4 - 0''.0000000951t^5$$

$$\omega_A = \varepsilon_0 - 0''.025754t + 0''.0512623t^2 - 0''.00772503t^3 - 0''.000000467t^4 + 0''.0000003337t^5$$

$$\chi_A = 10''.556403t - 2''.3814292t^2 - 0''.00121197t^3 + 0''.000170663t^4 - 0''.0000000560t^5$$

where $\varepsilon_0 = 84381''.406$ and the unit of these angular arguments is arc seconds. The fundamental time argument is given by the expression

$$t = (JD_1 - JD_2)/36525$$

In this equation JD_1 is the Julian Date of the first epoch and JD_2 is the Julian Date at 12 hours on January 1, 2000, both measured on the Terrestrial Time (TT) scale.

The syntax of this MATLAB function that performs these calculations is

```
function pos2 = preces (tjd1, pos1, tjd2)

% this function precesses equatorial rectangular coordinates from
% one epoch to another.  the coordinates are referred to the mean
% dynamical equator and equinox of the two respective epochs.  see
% explanatory supplement to the astronomical almanac, pp. 103-104,
% and capitaine et al. (2003), astronomy and astrophysics 412,
% 567-586.
```

```
% input

%  tjd1 = tdb julian date of first epoch

%  pos1 = position vector, geocentric equatorial rectangular
%         coordinates, referred to mean dynamical equator and
%         equinox of first epoch

%  tjd2 = tdb julian date of second epoch

% output

%  pos2 = position vector, geocentric equatorial rectangular
%         coordinates, referred to mean dynamical equator and
%         equinox of second epoch

% note: either tjd1 or tjd2 must be 2451545.0 (j2000.0) tdb
```

Nutation

The nutation function implemented in this MATLAB script is based on the methods described in "Modeling of nutation and precession: New nutation series for nonrigid Earth and insights into the Earth's interior", P. M. Mathews, T. A. Herring and B. A. Buffett, *Journal of Geophysical Research*, Vol. 107, No. B4, 2002.

The high precision nutation calculations in this script are based on the IAU 2000A nutation algorithm and the low precision form uses the IAU 2000K algorithm.

The nutation in longitude is determined from a series of the form

$$\Delta\psi = \sum_{i=1}^{N}\left(A_i + A_i' t\right)\sin\theta + A_i''\cos\theta$$

Likewise, the nutation in obliquity is determined from

$$\Delta\varepsilon = \sum_{i=1}^{N}\left(B_i + B_i' t\right)\cos\theta + B_i''\sin\theta$$

where

$$\theta = \sum_{1}^{14} N_i F_i$$

In this last summation, N_i are integer multipliers and F_i are fundamental arguments. For the IAU 2000A version, $N = 1365$ in the summation of which 678 of the terms are luni-solar contributions and 687 are planetary contributions to nutation. For the IAU 2000K version, $N = 488$ and 323 terms are luni-solar contributions and 165 are planetary contributions.

The first five angular elements are luni-solar Delaunay arguments given by the following expressions;

$$F_1 = l = \text{ mean anomaly of the Moon}$$
$$= 485868''.249036 + 1717915923''.2178t$$

$$F_2 = l' = \text{ mean anomaly of the Sun}$$
$$= 1287104''.79305 + 129596581''.0481t$$

$$F_3 = F = L - \Omega \ \left(L \text{ is the mean longitude of the Moon}\right)$$
$$= 335779''.526232 + 1739527262''.8478t$$

$$F_4 = D = \text{ mean elongation of the Moon from the Sun}$$
$$= 1072260''.70369 + 1602961601''.2090t$$

$$F_5 = \Omega = \text{ mean longitude of the ascending node of the lunar orbit}$$
$$= 450160''.398036 - 6962890''.5431t$$

The remaining arguments are planetary mean longitudes given by the following expressions;

$$F_6 = L_{Me} = 4.402608842 + 2608.7903141574t$$

$$F_7 = L_{Ve} = 3.176146697 + 1021.3285546211t$$

$$F_8 = L_E = 1.753470314 + 628.3075849991t$$

$$F_9 = L_{Ma} = 6.203480913 + 334.0612426700t$$

$$F_{10} = L_J = 0.599546497 + 52.9690962641t$$

$$F_{11} = L_{Sa} = 0.874016757 + 21.3299104960t$$

$$F_{12} = L_U = 5.481293872 + 7.4781598567t$$

$$F_{13} = L_{Ne} = 5.311886287 + 3.8133035638t$$

$$F_{14} = p_A = 0.02438175t + 0.00000538691t^2$$

In these equations, the time argument t is the number of Julian centuries since J2000 and is given by $t = \left(JD - 2451545.0\right)/36525$ where JD is the Terrestrial Time (TT) Julian Date. Additional information about these arguments can be found in "Numerical Expressions for Precession Formulae and Mean Elements for the Moon and Planets", *Astronomy and Astrophysics*, **282**: 663-683.

The nutation matrix is given by

$$\mathbf{N} = \begin{bmatrix} \cos\Delta\psi & -\sin\Delta\psi\cos\varepsilon_0 & -\sin\Delta\psi\sin\varepsilon_0 \\ \sin\Delta\psi\cos\varepsilon & \cos\Delta\psi\cos\varepsilon\cos\varepsilon_0 + \sin\varepsilon\sin\varepsilon_0 & \cos\Delta\psi\cos\varepsilon\cos\varepsilon_0 - \sin\varepsilon\cos\varepsilon_0 \\ \sin\Delta\psi\sin\varepsilon & \cos\Delta\psi\sin\varepsilon\cos\varepsilon_0 - \cos\varepsilon\sin\varepsilon_0 & \cos\Delta\psi\sin\varepsilon\sin\varepsilon_0 + \cos\varepsilon\cos\varepsilon_0 \end{bmatrix}$$

In this matrix ε_0 is the mean obliquity of the ecliptic and $\varepsilon = \varepsilon_0 + \Delta\varepsilon$ is the true obliquity. The mean obliquity of the ecliptic is calculated from

$$\varepsilon_0 = 23^0 26' 21''.448 - 46''.8150T - 0''.00059T^2 + 0''.001813T^3$$

where $T = (JD - 2451545.0)/36525$ and JD is the TDB Julian Date.

The nutation matrix can also be expressed as a combination of individual rotations according to

$$\mathbf{N} = \mathbf{R}_1(-\varepsilon)\mathbf{R}_3(-\Delta\psi)\mathbf{R}_1(+\varepsilon_0)$$

where

$$R_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \qquad R_3(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

This function requires initialization the first time it is called. This can be accomplished by placing the following statement in the main script along with a `global inutate` statement.

```
inutate = 1;
```

The following is the MATLAB source code for the function that calls the correct nutation function. This function determines which algorithm to call based on the value of `mode` which is set in the main script.

```
function [dpsi, deps] = nod (t)

% this function returns the values for nutation in longitude and
% nutation in obliquity for a given tdb julian date.

%  t     = tdb time in julian centuries since j2000.0 (in)

%  dpsi  = nutation in longitude in arcseconds (out)

%  deps  = nutation in obliquity in arcseconds (out)

% ported from NOVAS 3.1

%%%%%%%%%%%%%%%%%%%%%%%

seccon = 180.0d0 * 3600.0d0 / pi;

% t0 = tdb julian date of epoch j2000.0 (tt)

t0 = 2451545.0d0;

% get method/accuracy mode

mode = getmod;
```

```
t1 = t * 36525.0d0;

% evaluate nutation series

% resulting nutation in longitude and obliquity in arc seconds

if (mod (mode, 2) == 0)

    % high accuracy mode -- iers 2000a

    [dp, de] = nut2000a (t0, t1);

else

    % low accuracy mode -- iau 2000k

    [dp, de] = nut2000k (t0, t1);

end

dpsi = dp * seccon;

deps = de * seccon;
```

The following is the calling syntax for the high precision version of the nutation function which is named `nut2000a.m`.

```
function [dpsi, deps] = nut2000a (date1, date2)

% nutation based on iau 2000a theory

% input

%  date1, date2 = tt julian date
%  (julian date = date1 + date2)

% output

%  dpsi = nutation in longitude in radians

%  deps = nutation in obliquity in radians
```

The low precision version is named `nut2000k.m` with the following syntax. The input and output for this function are the same as those for the `nut2000a` function described above.

```
function [dpsi, deps] = nut2000k (date1, date2)
```

Appendix B

Time Scales

This appendix is a brief explanation of the time scales used in this MATLAB script.

Coordinated Universal Time, UTC

Coordinated Universal Time (UTC) is the time scale available from broadcast time signals. It is a compromise between the highly stable atomic time and the irregular earth rotation. UTC is the international basis of civil and scientific time.

Terrestrial Time, TT

Terrestrial Time is the time scale that would be kept by an ideal clock on the geoid - approximately, sea level on the surface of the Earth. Since its unit of time is the SI (atomic) second, TT is independent of the variable rotation of the Earth. TT is meant to be a smooth and continuous "coordinate" time scale independent of Earth rotation. In practice TT is derived from International Atomic Time (TAI), a time scale kept by real clocks on the Earth's surface, by the relation **TT = TAI + 32^s.184.** It is the time scale now used for the precise calculation of future astronomical events observable from Earth.

TT = TAI + 32.184 seconds

TT = UTC + (number of leap seconds) + 32.184 seconds

Barycentric Dynamical Time, TDB

Barycentric Dynamical Time is the time scale that would be kept by an ideal clock, free of gravitational fields, co-moving with the solar system barycenter. It is always within 2 milliseconds of TT, the difference caused by relativistic effects. TDB is the time scale now used for investigations of the dynamics of solar system bodies.

TDB = TT + periodic corrections

where typical periodic corrections (USNO Circular 179) are

$$
\begin{aligned}
TDB = TT &+ 0.001657\sin\left(628.3076T + 6.2401\right) \\
&+ 0.000022\sin\left(575.3385T + 4.2970\right) \\
&+ 0.000014\sin\left(1256.6152T + 6.1969\right) \\
&+ 0.000005\sin\left(606.9777T + 4.0212\right) \\
&+ 0.000005\sin\left(52.9691T + 0.4444\right) \\
&+ 0.000002\sin\left(21.3299T + 5.5431\right) \\
&+ 0.000010T\sin\left(628.3076T + 4.2490\right) + \cdots
\end{aligned}
$$

In this equation, the coefficients are in seconds, the angular arguments are in radians, and T is the number of Julian centuries of TT from J2000; T = (Julian Date(TT) – 2451545.0) / 36525.

The following is the MATLAB source code for the routine ported from the NOVAS Fortran subroutine. Notice that the NOVAS name was simply `times` and the ported version is named `novas_times` to avoid confusion with the built-in MATLAB function.

```
function [ttjd, secdif] = novas_times (tdbjd)

% this function computes the terrestrial time (tt) julian date
% corresponding to a barycentric dynamical time (tdb) julian date.
% the expression used in this version is a truncated form of a
% longer and more precise series given by fairhead & bretagnon
% (1990) a&a 229, 240. the result is good to about 10 microseconds.

% input

%  tdbjd = tdb julian date

% output

%  ttjd = tt julian date

%  secdif = difference tdbjd - ttjd, in seconds

% ported from NOVAS 3.0

%%%%%%%%%%%%%%%%%%%%%%

% t0 = tdb julian date of epoch j2000.0 (tt)

t0 = 2451545.0d0;

t = (tdbjd - t0) / 36525.0d0;

% expression given in usno circular 179, eq. 2.6

secdif = 0.001657d0 * sin(628.3076d0 * t + 6.2401d0) ...
    + 0.000022d0 * sin(575.3385d0 * t + 4.2970d0) ...
    + 0.000014d0 * sin(1256.6152d0 * t + 6.1969d0) ...
    + 0.000005d0 * sin(606.9777d0 * t + 4.0212d0) ...
    + 0.000005d0 * sin(52.9691d0 * t + 0.4444d0) ...
    + 0.000002d0 * sin(21.3299d0 * t + 5.5431d0) ...
    + 0.000010d0 * t * sin(628.3076d0 * t + 4.2490d0);

ttjd = tdbjd - secdif / 86400.0d0;
```