Contents

1 Introduction

RSSUBS is a FORTRAN subroutine library that contains some handy routines to be used for many purposes.

The RSSUBS library has been ported to various systems. Presently it is available at DEOS/LR on the IBM Risk 6000 and the SGI servers. At NOAA it is available for Linux.

The library can be linked to any of your FORTRAN programs by specifying the library in you FORTRAN link step, *e.g.*

```
f77 example.f -o example $ALTIM/lib/rssubs.a
```

The RSSUBS library is courtesy of Remko Scharroo, Delft University of Technology, Delft Institute for Earth-Oriented Space Research. (E-mail: `remko@deos.tudelft.nl`)

2 Subroutine Synopsis

2.1 ALTBIAS – Altimeter range biases and drifts

```
    SUBROUTINE ALTBIAS (PROD, UTC, COR0, COR1, COR2)
    implicit none
    INTEGER  PROD, UTC
    REAL*8   COR0, COR1, COR2

This function returns altimeter range bias and drifts for various altimeter
products. These range deltas can be either constant (range bias)
or time variant (drift). Because of the different problems associated
to each satellite product, the actual source of the biases and drifts
are different.

Input to the function are the satellite product identifier (PROD),
and the epoch in UTC seconds of 1985 (UTC). The function returns
three deltas: a constant bias (COR0) and two time-variant terms
(COR1 and COR2), all in metres. To correct the range, COR0 should
be SUBTRACTED FROM the ranges provided on the data and COR1 and COR2
should be ADDED (due to conventions).

Thus:    RANGE (corrected) = RANGE - COR0 + COR1 + COR2
Or:      SSH   (corrected) = SSH   + COR0 - COR1 - COR2

Arguments (see below for further description):
 PROD      (input) : Satellite ID or Product identifier
 UTC       (input) : Time in UTC seconds since 1.0 Jan 1985
                     or TOPEX/POSEIDON Cycle number
 COR0     (output) : Range bias, constant part
 COR1     (output) : Range bias, variable part
 COR2     (output) : USO correction

Satellite ID or Product identifier (PROD)
-----------------------------------------
3/300     : Geosat GDR
4/400     : ERS-1 OPR Version 5 and higher
5/500     : TOPEX (T/P Merged GDR, new format)
6/600     : POSEIDON (T/P Merged GDR, new format)
7/700     : ERS-2 OPR Version 5 and higher
8         : GFO GDR
9         : Jason-1 (I)GDR
10        : Envisat (I)GDR
  401     : ERS-1   OPR Version 3
  410/710 : ERS-1/2 QLOPR
  411/711 : ERS-1/2 URA
  412/712 : ERS-1/2 IGDR/RGDR
  420/720 : ERS-1/2 WAP
  501/601 : TOPEX/POSEIDON Merged GDR, old format

Meaning of COR0, COR1, and COR2
-------------------------------
          COR0            COR1                  COR2
Geosat    range bias    Internal calibration    USO drift
ERS-1     range bias    SPTR bias               USO drift
TOPEX     range bias    Internal calibration    USO drift correction
POSEIDON  range bias         0                      0
ERS-2     range bias    SPTR bias               USO drift
GFO            0             0                      0
Jason-1        0             0                      0
Envisat   range bias         0                  USO drift
```

2.5 ANMTOT – Convert Mean anomaly to True anomaly.

```
    FUNCTION ANMTOT (M, E)
    REAL*8 M, E, ANMTOT

Computes the true anomaly out of the mean anomaly (and eccentricity)
with an accuracy of (eccentricity**7)

Arguments:
 M       (input): Mean anomaly (radians)
 E       (input): Eccentricity (unity)
 ANMTOT (output): True anomaly (radians)
```

2.6 ANTTOM – Convert True anomaly to Mean anomaly.

```
    FUNCTION ANTTOM (THETA, E)
    REAL*8 THETA, E, ANTTOM

Computes the mean anomaly out of the true anomaly (and eccentricity)
using straightforward algorithms. No approximation. No iteration.

Arguments:
 THETA   (input): True anomaly (radians)
 E       (input): Eccentricity (unity)
 ANTTOM (output): Mean anomaly (radians)
```

2.7 BUBBLE – Make a bubble sort of an integer array

```
    SUBROUTINE BUBBLE (INDEX, VALUE, NR)
    INTEGER*4 NR, INDEX(NR), VALUE(*)

This routine bubble-sorts an integer INDEX according to the corresponding
integer value in array VALUE. The array VALUE may be larger than NR,
while array INDEX should contain NR values.

Arguments:
 INDEX (input) : One-dimensional array of index numbers
       (output) : Row of index numbers sorted on corresponding value
 VALUE (input) : One-dimensional array of integer values
 NR    (input) : Number of indices/values

Example:
Assume you have 6 values 100, 10, 11, 21, 17, and 90, to be sorted in
the right order, then your input can be like this:

  INDEX    1    2    3    4    5    6
  VALUE  100   10   11   21   17   90
     NR    6

After running BUBBLE, the output will look like this:

  INDEX    2    3    5    4    6    1
  VALUE  100   10   11   21   17   90
     NR    6

Note that the array VALUE hasn't changed, but that INDEX is sorted by
the order in which VALUE should be sorted.
```

2.8 CARPOL – Convert Cartesian to Polar

```
    SUBROUTINE CARPOL (VECTOR, LAT, LON, R)
    REAL*8 VECTOR(3), LAT, LON, R

Convert Cartesian VECTOR() = (X,Y,Z) to polar coordinates (LAT,LON,R)

Arguments:
VECTOR  (input) : Vector of X, Y, and Z coordinates.
LAT     (output) : Latitude in radians
                   = angle from XY-plane towards +Z-axis
LON     (output) : Longitude in radians
                   = angle in XY-plane measured from +X-axis towards +Y-axis
R       (output) : Radius
```

2.9 CHECKENV – Get environment variable or leave default

```
    SUBROUTINE CHECKENV (ENV, STRING, L)
    CHARACTER*(*) ENV, STRING
    INTEGER*4     L

This routine returns the contents of environment variable ENV in the
variable STRING.
If the environment variable ENV is not set, STRING is unchanged.
Upon return the L will be the length of the string STRING, without
trailing spaces.

Arguments:
  ENV     (input) : Name of the environment variable.
  STRING  (input) : Default value for STRING.
         (output) : Contents of the environment variable or default.
  L       (output) : Length of STRING
```

2.10 CHRDAT – Convert seconds past 1 Jan 1985 to character

```
    SUBROUTINE CHRDAT (SEC85, DATE)
    INTEGER       SEC85
    CHARACTER*(*) DATE

Arguments:
SEC85  (input) : Number of integral seconds past 1.0 Jan 1985
DATE  (output) : Character containing date, hours, minutes, seconds in the
                 form '910513 14:22:29'. Should be declared at least
                 CHARACTER*15 in the calling (sub)program.
```

2.11 CHRLOC – Write latitude and longitude in character format

```
    SUBROUTINE CHRLOC (LAT, LON, POSITION)
    INTEGER*4     LAT, LON
    CHARACTER*(*) POSITION

Arguments:
LAT       (input) : Latitude in microdegrees
LON       (input) : Longitude in microdegrees
POSITION (output) : Position in format '72.0N 179.9E'
```

2.12 COVAR – Determine undulation covariance of two points.

```
    FUNCTION COVAR (PSI)
    REAL*8 PSI, COVAR

Compute the covariance between the geoid height values in two points
as a function of their spherical distance in spherical approximation.
WARNING: this function can only be used after
         'CALL COVINI (FILENM, ITYPE, MINDEG, MAXDEG)'

Arguments:
```

```
 PSI    (input): Spherical distance between the two points (rad).
 COVAR (output): Geoid undulation covariance of the two points (m**2).
```

2.13 COVAR2 – Determine gravity covariance of two points.

```
    FUNCTION COVAR2 (Q1, Q2, PSI)
    INTEGER*4 Q1, Q2
    REAL*8 PSI, COVAR2

Compute the covariance between the gravity values in two points,
or some of its derived quantities, as a function of the spherical
distance between the points in spherical approximation.
WARNING: this function can only be used after
         'CALL COVINI (FILENM, ITYPE, MINDEG, MAXDEG)'

Arguments:
 Q1, Q2  (input): Integers denoting the quantities that are considered:
                  1 = Gravity potential (units: m**2/s**2)
                  2 = Gravity anomaly   (units: mgal)
                  3 = Geoid undulation  (units: m)
 PSI     (input): Spherical distance between the two points (rad).
 COVAR2 (output): Covariance of the two points.
```

2.14 COVINI – Initiate covariance function.

```
    SUBROUTINE COVINI (FILENM, ITYPE, MINDEG, MAXDEG)
    CHARACTER*(*) FILENM
    INTEGER*4 ITYPE, MINDEG, MAXDEG

This subroutine initializes the geoid undulation degree variances
(ln) taking the differences between two geoid models complete to
degree 'ndeg' and a model for higher degrees.
This subroutine must be executed before using the function 'covar'.

Arguments:
 FILENM (input): Name of the file containing the degree variances for low
                 degrees.
 ITYPE  (input): Type of model used:
                 0 = all higher orders set to zero.
                 1 = Rapp (1979).
                 2 = Kaula.
                 3 = Kaula (to fit OSU89b order 36-360).
                 4 = Kaula (to fit OSU89b-OSU86f order 36-360).
 MINDEG (input): Minimum degree to be included in covariance function.
 MAXDEG (input): Maximum degree to be included in covariance function.
```

2.15 DATEARG – Processes standard datation arguments

```
    FUNCTION DATEARG (ARG, T0, T1, DT)
    LOGICAL*4 DATEARG
    CHARACTER*(*) ARG
    REAL*8    T0, T1, DT

This function processes standard datation arguments of either of the
```

```
following forms:

  mjd=t0[,t1[,dt]] : Modified Julian Dates
  sec=t0[,t1[,dt]] : Seconds since 1.0 Jan 1985
  ymd=t0[,t1[,dt]] : [YY]YYMMDD.DDD or [YY]YYMMDDHHMMSS.SSS
  doy=t0[,t1[,dt]] : YYDDD.DDD
    t=t0[,t1[,dt]] : MJD.DDD or [YY]YYMMDD.DDD or [YY]YYMMDDHHMMSS.SSS

Normally ARG is read from the argument line of a command using the GETARG
routine. When ARG is parsed through to DATEARG it checks whether ARG is
of one of the above forms. If not, DATEARG gets the value .FALSE.
If ARG is of one of the standard datation forms, the values of T0 and
T1 (when given) are read and interpreted as stated above and converted
to Seconds since 1.0 Jan 1985. When provided, DT is read too, but not
converted. DATEARG will get the value .TRUE.

When T1 or DT are not provided in ARG, the arguments of the function call
will keep their values unchanged.

Arguments:
  DATEARG (output): .TRUE. if ARG is in a standard datation format
  ARG      (input): datation argument (see above)
  T0, T1  (output): datation arguments converted to SEC85
                    (Seconds since 1.0 Jan 1985)
  DT      (output): Third datation argument (not converted)
```

2.16 DHELLIPS – Compute height difference between ellipsoids

```
    FUNCTION DHELLIPS (CONV, LAT)
    REAL*8   DHELLIPS, LAT
    INTEGER  CONV
```

Compute height difference between WGS84, GEOSAT and TOPEX ellipsoids. The input is LAT in degrees, the returned function value is DHELLIPS in metres (the result is always non-negative). CONV specifies the conversion (see below).

The ellipsoids are defined by their equatorial radius and inverse flattening as shown in the table below. Also indicated are which products use which ellipsoids.

Ellipsoid	Eq. radius (m)	Inv. flattening (-)	Products
WGS84	6378137.0	298.257223563	ESA ERS and Envisat
GEOSAT	6378137.0	298.257	GEOSAT T2 GDRs, DEOS ERS orbits
TOPEX	6378136.3	298.257	AVISO T/P and Jason, GEOSAT J3 GDRs, RADS

The conversion is linearised in sin(lat)**2, changes in latitude are ignored.

Input Arguments:

```
 CONV     : Conversion indicator
            CONV=1 : Height of WGS84 over TOPEX ellipsoid
            CONV=2 : Height of WGS84 over GEOSAT ellipsoid
            CONV=3 : Height of GEOSAT over TOPEX ellipsoid
            Other  : Zero height difference is returned
 LAT      : Latitude (degrees)
```

Function value (output):

```
 DHELLIPS : Height difference between ellipsoids (m)
```

Example 1: Convert DEOS orbital altitudes (provided relative to the GEOSAT ellipsoid) to the WGS84 reference ellipsoid.

```
    CALL GETORB (TIME, DLAT, DLON, HORBIT, PATH, VERBOSE)
    HORBIT = HORBIT - DHELLIPS (2, DLAT)
```

Example 2: Convert heights in RA2 data products (WGS84 reference) to TOPEX ellipsoid reference.

```
    IDH = NINT(DHELLIPS(1,LAT/1D6)*1D3)
    ALT_COG_ELLIP = ALT_COG_ELLIP + IDH
    GEOID_HT = GEOID_HT + IDH
    M_SEA_SURF_HT = M_SEA_SURF_HT + IDH
```

2.17 DQSORT – Make a quick sort of an REAL*8 array

```
    SUBROUTINE DQSORT (INDEX, VALUE, NR)
    INTEGER*4 NR, INDEX(NR)
    REAL*8 VALUE(*)
```

```
This routine quick-sorts an INTEGER*4 array INDEX according to the
corresponding value in array VALUE of type REAL*8.
The array INDEX contains NR pointers to the values in array VALUE, which
does not have to be equally large.

Arguments:
 INDEX (input) : One-dimensional array of index numbers (INTEGER*4)
      (output) : Row of index numbers sorted on corresponding value
 VALUE (input) : One-dimensional array of values (REAL*8)
 NR    (input) : Number of indices/values

Example 1:
Assume you have 6 values 100, 10, 11, 21, 17, and 90, stored in array
VALUE. These values have to be sorted in ascending order. Your input
will be like this:

  INDEX    1    2    3    4    5    6
  VALUE  100   10   11   21   17   90
     NR    6

After running DQSORT, the output will look like this:

  INDEX    2    3    5    4    6    1
  VALUE  100   10   11   21   17   90
     NR    6

Note that the array VALUE has not been changed, but that INDEX is sorted by
the order in which VALUE should be sorted. You may print the values in
the ascending order as follows:

    DO I=1,NR
       WRITE (*,*) VALUE(INDEX(I))
    ENDDO

Example 2:
It is also possible that the indices are not in logical order. As long as
they point to the respective locations of the values in array VALUE, the
sorting will be done accordingly. Assume you have the following input:
(values ** are irrelevant)

  INDEX    1    2    4    5    6    8
  VALUE  100   10   **   21   17   90   **   1
     NR    6

Then after running DQSORT, the result will be:

  INDEX    8    2    5    4    6    1
  VALUE  100   10   **   21   17   90   **   1
     NR    6
```



```
Printing the values in ascending order after this goes the same as in
Example 1.
```

2.18 DYNTOPO – Compute velocity from dynamic topography

```
    SUBROUTINE DYNTOPO (Z, VX, VY, NX, NY, MX, X0, X1, Y0, Y1)
    INTEGER*4 NX, NY, MX
    REAL*4 Z(MX,*), VX(MX,*), VY(MX,*), X0, X1, Y0, Y1

Determine velocity field components VX (due West) and VY (due North)
based on the dynamic topography height Z. The velocity grids and
the dynamic topography grid have the same resolution and are for the
same area.

Arguments:
 Z       (input): grid of dynamic height (unit: metres)
 VX, VY (output): grids of X and Y component of the velocity field
                  (unit: metres per second)
 NX, NY  (input): dimension of grids
 MX      (input): first dimension of arrays Z, VX, and VY
 X0, X1  (input): longitude boundaries (deg)
 Y0, Y1  (input): latitude boundaries (deg)
```

2.19 SETEARTH – Specify parameters of the reference ellipsoid

```
    SUBROUTINE SETEARTH (A_E, F_INV)
    REAL*8 A_E, F_INV

Specify the equatorial radius (A_E) and inverse flattening (F_INV)
of the reference ellipsoid to be used by GEOCEN, GEODET, GEOXYZ
and XYZGEO.

When this routine is not called the default values will be used:
A_E = 6378137.0 meters
F_INV = 298.257

Input arguments:
 A_E : Mean equatorial radius in meters
 F_INV : Inverse flattening
```

2.20 GETEARTH – Inquire parameters of the reference ellipsoid

```
    SUBROUTINE GETEARTH (A_E, F_INV)
    REAL*8 A_E, F_INV

Get the equatorial radius (A_E) and inverse flattening (F_INV)
of the reference ellipsoid to be used by GEOCEN, GEODET, GEOXYZ
and XYZGEO.

Output arguments:
 A_E : Mean equatorial radius in meters
 F_INV : Inverse flattening
```

2.21 ELEVEC – Convert Keplerian elements to position-velocity vector

```
    SUBROUTINE ELEVEC (ELEM, VECT, GM)
    REAL*8 ELEM(6), VECT(*), GM

This routine converts the 6 Keplerian elements (a, e, i, O, w, th) to
a position vector (X, Y, Z) or a position-velocity vector
(X, Y, Z, Xdot, Ydot, Zdot). When the given gravitational parameter GM is
less or equal to zero, then only the position vector is determined.

Arguments:
 ELEM  (input): Array of 6 Keplerian elements:
                ELEM(1): Semi-major axis (meters),
                ELEM(2): Eccentricity (unity),
                ELEM(3): Inclination (radians),
                ELEM(4): Right ascension of the ascending node (radians),
                ELEM(5): Argument of the pericenter (radians),
                ELEM(6): True anomaly (radians).
 VECT (output): Array of position (and velocity):
                VECT(1)..VECT(3): Position vector (X,Y,Z) (meters),
                VECT(4)..VECT(6): Velocity vector (Xdot,Ydot,Zdot)
                                  (meter/sec).
 GM    (input): Gravitational parameter of the central body
                (meter**3/sec**2). If GM<=0 then no velocity vector is
                computed.
```

2.22 EPHARG – Determine ephemeris arguments of sun and moon.

```
    FUNCTION EPHARG (K, TIME)
    INTEGER*4 K
    REAL*8  TIME, EPHARG

Compute mean longitude of Moon and Sun, their perigee and node,
or their rates.
Based on Newcomb' and Brown' Theory.

Angles describing long-term motion of the Sun and the Moon
 a = a0 + a1*T + a2*T**2 + a3*T**3  with
 T = centuries past JD 2415020 (31.5 Dec 1899) = 15019.5 MJD

   a0            a1          a2         a3
270.43659   +481267.89057  +0.00198  +0.000002  s = mean long. of the Moon
279.69660    +36000.76892  +0.00030   0.000000  h = mean long. of the Sun
334.32956     +4069.03403  -0.01032  -0.000010  p = mean long. of lunar perig
259.18328     -1934.14201  +0.00208  +0.000002  N = mean long. of lunar node
281.22083        +1.71902  +0.00045  +0.000003  ps= mean long. of solar perig

Ref: Stefano Casotto - Nominal ocean tide models for TOPEX precise orbit
     determination, University of Texas at Austin, Dec 1989.

Arguments:
 K       (input): Argument identifier:
                  K=1: Greenwich siderial time (theta-g)
                  K=2: Mean longitude of the Moon (s)
                  K=3: Mean longitude of the Sun (h)
                  K=4: Mean longitude of the lunar perigee (p)
                  K=5: Mean longitude of the lunar node (N)
                  K=6: Mean longitude of the solar perigee (ps)
                  K<0: Rate of the argument identified by -K
 TIME    (input): MJD of the epoch at which argument or argument rate must
                  be determined.
 EPHARG (output): Ephemeris argument identified by K (in degrees) or
                  the rate of the argument identified by -K (in deg/day)
```

2.23 EPHPOS – Determine ephemeris of Sun and Moon.

```
    SUBROUTINE EPHPOS (PLANET, TIME, XYZ)
    INTEGER*4 PLANET
    REAL*8  TIME, XYZ(3)

Compute the inertial position of the Moon and Sun.
Based on Newcomb' and Brown' Theory.

The XYZ coordinates are inferred from the angles describing the long-term
motion of the Sun and the Moon
 a = a0 + a1*T + a2*T**2 + a3*T**3  with
 T = centuries past JD 2415020 (31.5 Dec 1899) = 15019.5 MJD

   a0            a1          a2         a3
270.43659   +481267.89057  +0.00198  +0.000002  s = mean long. of the Moon
279.69660    +36000.76892  +0.00030   0.000000  h = mean long. of the Sun
334.32956     +4069.03403  -0.01032  -0.000010  p = mean long. of lunar perig
259.18328     -1934.14201  +0.00208  +0.000002  N = mean long. of lunar node
281.22083        +1.71902  +0.00045  +0.000003  ps= mean long. of solar perig

Ref: Stefano Casotto - Nominal ocean tide models for TOPEX precise orbit
     determination, University of Texas at Austin, Dec 1989.

Arguments:
 PLANET (input): Planet identifier: 1=Moon, 2=Sun.
 TIME   (input): MJD of the epoch at which argument or argument rate must
                 be determined.
 XYZ    (output): X, Y, and Z component of the ephemeris of the Moon or Sun
                 in meters.
```

2.24 F1F2 – Convert sigma0 to wind and vv using Gourrion 2-parameter model

```
    FUNCTION F1F2 (I, VAL, SWH)
    REAL*8    F1F2, VAL, SWH
    INTEGER*4 I

This function computes wind speed (U10, referenced to 10 m above
sea level) from altimeter backscatter coefficient (SIGMA0) and
significant wave height (SWH). The reverse is also possible:
converting U10 and SWH to SIGMA0.
The parameter I selects whether the forward (I=1) or backward
(I=2) conversion is requested.

The unit for SIGMA0 is dB, for SWH is m and for U10 is m/s.

The respective models F1(SIGMA0,SWH) and F2(U10,SWH) are described in
Gourrion et al. (2002).

Input arguments:
 I   : Select direction of conversion
       1 = Convert SIGMA0 and SWH to U10 (F1)
       2 = Convert U10 and SWH to SIGMA0 (F2)
 VAL : Either SIGMA0 (I=1) or U10 (I=2)
 SWH : Significant wave height

Returned value:
 F1F2: Either U10 (I=1) or SIGMA0 (I=2)

Reference:

Gourrion, J., D. Vandemark, S. Bailey, B. Chapron, G. P. Gommenginger,
P. G. Challenor, and M. A. Srokosz,
A two-parameter wind speed algorithm for Ku-band altimeters,
J. Atmos. Ocean. Technol., 19(12), 2030-2048, 2002.
```

2.25 FASTIO – Fast low-level UNIX I/O routines

```
FASTIO is a set of functions that makes fast, low-level Unix I/O routines
available to a Fortran program, accounting for some of the differences in the
C/Fortran interfaces of different machines.

Specifically, some linkers expect a C-routine which is called from a
Fortran program to have a name ending in an underscore.  So, for example,
if the Fortran program calls "openf()", the C-function's name must
be "openf_()".  Other vendor's (like HP and RS6000) look for a C-routine with
the actual name as called from Fortran (without the underscore).

Therefore, one can define UNDERSCOREAFTER during the compilation to get
the underscores after the function names. Obviously, one can compile
with and without the -DUNDERSCOREAFTER option and link both objects
```

```
if you are not sure. Other options are: -DUNDERSCOREBEFORE and -DCAPITALS.

Secondly, when character strings are passed from Fortran to C, the
string's length is implicitly passed, unbeknownst to the Fortran caller.
Some C/Fortran interfaces put the string length as the next argument after
the string itself; others place the string length at the end of the
argument list.  Again, to support both implementations, when a string is
passed to the following routines it is the last argument in the Fortran
call, so that the string and its length are the last two arguments in the
corresponding C-function.

For the definition of the open flag in the call "openf()", it is required
to include the file "fcntl.h". On most operating systems, this file
is in "/usr/include", but we have encountered systems where it is in
"/usr/include/sys". To support both, you should compile this source-code
with the additional search path specified: -I/usr/include/sys.

The syntax of the Fortran calls are provided below.

-------
$Log: fastio.c,v $
Revision 1.12  2006/10/10 17:36:54  rads
- Introduced new OPENF function

Revision 2.3  2005/05/16 00:21:27  rads
- Added include string.h

Revision 2.2  2004/08/28 15:29:01  remko
Reintroduced unistd.h (required by seekf on Mac OS X); spiffed up manual

24-Jul-2003 - Added IOFLAG function and removed O_TRUNC from read/write flag.
19-Jun-2003 - Included understanding of '-' to indicate stdin/stdout.
20-Dec-2001 - Improved manual, added O_TRUNC to the read/write flag.
12-Oct-2001 - Added line reading and writing. Add read/write flag to IOCONST.
 7-Jan-1996 - Underscores through UNDERSCOREAFTER. Warning also without
              carriage return.
17-May-1994 - Nice manual
11-Nov-1993 - Addition of seekf and warning by Remko Scharroo, DUT/SSR&T
 2-Dec-1992 - fastio.c: John L. Lillibridge, NOAA/NOS/OES Geosciences Lab
-------
```

2.25.1 OPENF – Open file for FASTIO

```
    FUNCTION OPENF (FNAME, MODE)
    CHARACTER*(*) FNAME, MODE
```

```
OPENF opens file FNAME for reading and or writing, depending on the
string MODE, which is the same as the "mode" argument of the C function
fopen().

FNAME can be up to 1024 characters long. Specify '-' for reading from
standard input or writing to standard output.
```

```
MODE is a 1- or 2-byte string indicating whether the file needs to
be opened for reading, writing, or both, and whether trunctation needs
to happen. The different options are:
'r'  : Open file for reading; start at the beginning of the file.
'r+' : Open file for reading and writing; start at the beginning
       of the file.
'w'  : Truncate file to zero length or create it for writing only.
'w+' : Open file for reading and writing. Truncate it to zero length
       if it exists, create it otherwise.
'a'  : Open for writing. Create the file if it does not exist. Start
       at the end of the file. Subsequent writes are appended at the end.
'a+' : Open for reading and writing. Create the file if it does not
       exist. Start at the end of the file. Subsequent write are
       appended at the end of the file.

OPENF returns the file descriptor for use in subsequent calls to
readf, writef, or closef. If OPENF is negative, an error occurred while
opening the file. Use PERRORF to report the error.

Usage for reading an EXISTING file.

    INTEGER FD, OPENF, STRING

    FD = OPENF ('input_file', 'r')
    CALL READF (FD, 4, STRING)
    CALL CLOSEF (FD)

Usage for writing a file which may or may not exist.

    FD = OPENF ('output_file', 'w')
    CALL WRITEF (FD, 4, STRING)
    CALL CLOSEF (FD)

Finally, to read and write to an existing file, use:

    FD = OPENF ('io_file', 'r+')

Note: Standard input and standard output can be opened through OPENF
by specifying '-' as the file name FNAME and using MODE 'r' and 'w'
respectively

Input argument:
 FNAME : File name of the input or output file. Maximum 1024 chars.
         Use '-' for standard input or standard output.
 MODE  : Open mode for reading or writing, see above.

Returned value:
 OPENF : Contains the file descriptor on return.
         If the returned value is negative an error occurred while
         opening the file.
```

2.25.2 CLOSEF – Close file from FASTIO access

```
    FUNCTION CLOSEF (FD)
```

```
    INTEGER CLOSEF (FD)
```

Closes the file with descriptor FD from FASTIO access. CLOSEF returns 0 when properly closed. Otherwise, use PERRORF to report the error.

Usage:

```
     IOS = CLOSEF (FD)
```

or:

```
     CALL CLOSEF (FD)
```

In the last case the return code is ignored.

Input argument:
FD : File descriptor returned by OPENF.
Returned value:
CLOSEF : Error code or 0 on proper closing.

2.25.3 READF – FASTIO read routine

```
     FUNCTION READF (FD, NBYTE, BUFFER)
     INTEGER FD, NBYTE, READF
     BYTE    BUFFER (NBYTE)
```

Reads a string of bytes or a line (until carriage return) from the file associated with descriptor FD (returned by the OPENF call).
When reading from standard input, use FD=0.

When reading a line up to AND INCLUDING the carriage return, use NBTYE=0. In this case the string that is read may contain no more than 1024 bytes and will end with a line-feed (CHAR(10)). Upon return, the value of READF will be the number of bytes of the string, including the line-feed.

When reading a string of a specified length (binary or otherwise), NBYTE specifies the length of the string to be read. READF will reflect the number of bytes actually read.

The array BUFFER will hold the data, but can (of course) also be associated with any other string, scalar, or n-dimensional array.

The function returns the number of bytes actually read in READF. If READF < 0, a read error occurred which can be reported by calling PERRORF. READF = 0 indicates an end-of-file mark.

Example of binary reading:

```
     INTEGER FD, RBYTE, READF, BUFFER(20)
     RBYTE = READF (FD, 80, BUFFER)
     IF (RBYTE .EQ. 80) THEN
        WRITE (6,*) 'All bytes returned'
        WRITE (6,*) BUFFER
     ELSE IF (RBYTE .EQ. 0) THEN
        WRITE (6,*) 'End-of-file'
     ELSE IF (RBYTE .LT. 0) THEN
        CALL PERRORF ('readf error')
```

```
     ELSE
        WRITE (6,*) 'Premature EOF after ',RBYTE,' bytes'
     ENDIF
```

Example of line reading:

```
     INTEGER FD, RBYTE, READF
     CHARACTER*80 LINE
     RBYTE = READF (FD, 0, LINE)
     IF (RBYTE .EQ. 0) THEN
        WRITE (6,*) 'End-of-file'
     ELSE IF (RBYTE .LT. 0) THEN
        CALL PERRORF ('readf error')
     ELSE
        WRITE (6,*) 'Read ',RBYTE,' bytes'
        WRITE (6,*) 'String = ',LINE(:RBYTE-1)   ! Strip off trailing CR
     ENDIF
```

```
Input arguments:
 FD     : File descriptor returned by OPENF.
 NBYTE  : Number of bytes to be read. Use NBYTE=0 to read to next carriage ret

Output argument:
 BUFFER : Buffer containing the bytes that have been read.

Returned value:
 READF  : Number of bytes read, or (if negative) error code.
```

2.25.4 WRITEF – FASTIO write routine

```
    FUNCTION WRITEF (FD, NBYTE, BUFFER)
    INTEGER FD, NBYTE, WRITEF
    BYTE    BUFFER(NBYTE)

Writes a string of bytes or a line (followed by carriage return) into
the file associated by descriptor FD (which is returned by the OPENF
call). When writing to standard output use FD=1.

When a line is written to the file, simply specify NBYTE=0. The string
should then include a trailing line-feed (CHAR(10)) and may not be
longer that 1024 bytes.
The function value of WRITEF will be the number of bytes of the string
including the line-feed.

When writing a binary string, NBYTE specifies the number of bytes
contained of the binary string. WRITEF will reflect the actual
numbers of bytes written.

The array BUFFER contains the data that has to be written, but can
(of course) also be associated with any other string, scalar, or
n-dimensional array.

The function returns the number of bytes actually written in WRITEF. If
WRITEF < 0, a write error occurred which can be reported by calling
```

```
PERRORF.

Input arguments:
 FD     : File descriptor returned by OPENF
 NBYTE  : Number of bytes to be written. Use 0 for a string ending in
          carriage return.
 BUFFER : Buffer containing the bytes that have to be written

Returned value:
 WRITEF : Number of bytes written, or (if negative) error code.
```

2.25.5 SEEKF – FASTIO file positioning routine

```
    FUNCTION SEEKF (FD, OFFSET, WHENCE)
    INTEGER FD, OFFSET, WHENCE, SEEKF

Spools file to the position indicated by the parameter OFFSET (in
bytes). The parameter WHENCE indicates whether OFFSET is counted
- from the beginning of the file (WHENCE = 0),
- from the current position      (WHENCE = 1),
- from the end of the file       (WHENCE = 2)
Upon return SEEKF reports the number of bytes skipped, or contains
an error code (to be reported by PERRORF).

Input arguments:
 FD     : File descriptor returned by OPENF
 OFFSET : Offset in bytes from the point indicated by WHENCE
 WHENCE : Indicates whether the OFFSET is counted from the
          start of the file (0), current position (1), or the
          end of the file (2).

Returned value:
 SEEKF  : On proper return: new position in file, otherwise error code
```

2.25.6 PERRORF – Report last FASTIO error

```
    SUBROUTINE PERRORF (COMMENT)
    CHARACTER*(*) COMMENT

Reports the last low-level I/O error returned by the system. The form of
the reported string which is sent to standard-error is:
"COMMENT: error message", where COMMENT is held in the parameter in the
call to PERRORF.

Input argument:
 COMMENT : Additional comment to the error message
```

2.26 FDATE – Get date and time as character string

```
    SUBROUTINE FDATE (STRING)
    CHARACTER*(*) STRING

Return the current date and time.
To receive the whole string, the STRING should be declared
at least CHARACTER*24 in the calling (sub)program.

Argument:
 STRING : receives date and time, truncated or extended with
          blanks as necessary.
```

2.27 FIN – Routine to end or abort program gracefully

```
    SUBROUTINE FIN (STRING)
    CHARACTER*(*) STRING

This routine terminates the execution of a program gracefully.
If STRING is empty, the program ends normally with the message
"Normal end of program xxx" printed to standard output.
Otherwise it first echos STRING and then "Execution of program
xxx terminated".
After that the STOP statement is excuted in both cases.

Argument:
 STRING (input): Message echoed at program termination. If empty, the
                 program ends normally.
```

2.28 FREEUNIT – Scan for free unit number for opening file

```
    FUNCTION FREEUNIT()
    INTEGER*4 FREEUNIT

This function returns a unit number that is not currently connected to
any file. A value of 0 is returned if no free unit is found.

Example:
    INTEGER*4 UNIT, FREEUNIT
    UNIT=FREEUNIT()
    OPEN (UNIT=UNIT, STATUS='OLD')
    ...
```

2.29 GAUSS1D – One-dimensional Gaussian filter

```
    SUBROUTINE GAUSS1D (N, X, SIGX, HORX, Z, SIGZ, ZF)
    INTEGER*4 N
    REAL*8    X(N), SIGX, HORX, Z(N), SIGZ(N), ZF(N)

This subroutine filters a set of N data points aligned along a (semi-)
one-dimensional coordinate. The independent variable is X, the data
values are N. All data points should be ordered in ascending or
descending order of X.

The data are weighted by their respective variances SIGZ,
and the spacial scale SIGX.
- Data with negative SIGZ are not used, but the filtered value is return.
- When SIGZ(1)=0, all data points are asigned an equal variance: SIGZ(i)=1

The horizon HORX determines the window to
which the filtering is applied (usually HORX = 2.5 * SIGX).
The (low-pass) filtered data are returned as ZF.

The Guassian filtering is formulated as follows:

ZF(i) = sum [ w(i,j) * Z(i) ] / sum w(i,j)
         j                       j

where w(i,j) is the weight contribution of point j to point i, given by

w(i,j) = exp [ -(X(i)-X(j))**2 / SIGX**2 ] / SIGZ(j)

ZF(i) is set to 1d30 when the value can not be determined.

Arguments:
N    (input) : Number of data points
X    (input) : Independent variable
SIGX (input) : Distance weighting sigma
HORX (input) : Horizon (best value: 2.5*SIGX)
Z    (input) : Data values
SIGZ (input) : Data variances. When SIGZ(i)<0, point i is not used in
         the filtering, but ZF(i) will be defined.
               When SIGZ(1)=0, SIGZ(i)=1 will be used for all i.
ZF   (output) : Low-pass filtered data
```

2.30 GEOCEN – Convert geodetic coordinates to geocentric coordinates

```
SUBROUTINE GEOCEN (LAT, HEIGHT, LATC, R)
REAL*8 LAT, HEIGHT, LATC, R
```

```
This subroutine converts the geodetic coordinates (latitude and height)
on the GRS80 reference ellipsoid to geocentric coordinates (geocentric
latitude and geocentric distance).

Arguments:
LAT    (input) : Geodetic latitude (rad).
HEIGHT (input) : Height above the reference ellipsoid (m).
LATC  (output) : Geocentric latitude (rad).
R     (output) : Distance to geocenter (m).
```

2.31 GEODET – Convert geocentric coordinates to geodetic coordinates

```
SUBROUTINE GEODET (LATC, HEIGHT, LAT, R)
REAL*8 LATC, HEIGHT, LAT, R
```

```
This subroutine converts the geocentric latitude (LATC) and
height (HEIGHT) above the reference ellipsoid GRS80 to the
geodetic latitude (LAT) and distance to the geocenter (R).
The method is iterative and stops after 100 iterations or
when LAT is converged to 1.D-11 radian and R up to 1.D-4 meter.

Arguments:
LATC   (input) : Geocentric latitude (rad).
HEIGHT (input) : Height above the reference ellipsoid (m).
LAT   (output) : Geodetic latitude (rad).
R     (output) : Distance to geocenter (m).
```

2.32 GEOUTM – Convert geodetic latitude and longitude to UTM coordinates

```
SUBROUTINE GEOUTM (CENMED, LAT, LON, X, Y)
REAL*8 CENMED, LAT, LON, X, Y
```

```
This routine converts the geodetic coordinates of a point on the
Earth to the Universal Transvers Mercator coordinates (UTM) on the
GRS80 spheroid.

Arguments:
 CENMED (input) : Central meridian of the projection (radians)
 LAT    (input) : Geodetic latitude of the point (radians)
 LON    (input) : East longitude (radians).
 X     (output) : UTM X-coordinate (meters).
 Y     (output) : UTM Y-coordiante (meters).
```

2.33 GEOXYZ – Convert geodetic latitude, longitude and height to ECF coordinates.

```
    SUBROUTINE GEOXYZ (LAT, LON, HEIGHT, XYZ, R)
    REAL*8 LAT, LON, HEIGHT, XYZ(3), R
```

```
This subroutine converts geodetic latitude, longitude and height above the
GRS80 reference ellipsoid to Earth Centered Fixed coordinates X, Y and Z.

Arguments:
LAT     (input) : Geodetic latitude (rad).
LON     (input) : Geodetic longitude (rad).
HEIGHT  (input) : Height above the reference ellipsoid (m).
XYZ(3) (output) : Earth Centered Fixed coordinates X, Y, and Z (m).
R      (output) : Distance to geocenter (m).
```

2.34 GETORB – Get orbital position of satellite from ODR files

```
    FUNCTION GETORB (TIME, LAT, LON, ORBIT, PATH, VERBOSE)
    INTEGER*4 GETORB
    REAL*8 TIME, LAT, LON, ORBIT
    CHARACTER PATH*(*)
    LOGICAL VERBOSE
```

```
This function reads the Orbital Data Records (ODR) in a directory or a
file indicated by the character PATH and interpolates the position of
the sub-satellite point (LAT and LON) and the orbital altitude (ORBIT)
above the GRS80 reference ellipsoid ( ae = 6378137.0 meter; f = 1/298.257 )
at the requested epoch TIME (in UTC seconds since 1985).

If PATH starts with + the rest of the character string is assumed to be
the pathname of a regular file. Otherwise, it is assumed to be a
directory name.

(1) PATH = directory

If this function is called for the first time, it will scan the directory for
existing ODR files (with the regular numbering ODR.iii), and store
the begin and end epoch of the precise part of each file. This scanning
is only repeated when a change of directory is made. The scanning is
accelerated if a file 'arclist' is available in the directory with the
same information.

This routine will forthwith load the ODR file that contains the orbital
positions around the requested epoch. A new file is only loaded when a
next call of GETORB is made for an epoch TIME outside the limits of the
loaded file, or when the directory path is changed.
Therefore, GETORB works fastest when subsequent calls are made for epochs
which are time sorted (ascending or descending), or at least sorted by
arc number.

This routine also supports orbital arcs including a prediction. A
```

```
position will be interpolated up to the end of the last available ODR
in the given directory.

(2) PATH = +file

A single ODR file is loaded at the first call with a new string PATH.
The routine will not scan for alternative ODR files. Interpolation is
restricted to this single ODR file indicated by the string following
the plus sign.

In both cases an error return code (GETORB > 0) is given it the requested
epoch is outside the limits of the loaded file or the available files in
the given directory. A warning return code (GETORB < 0) is given
when the file is interpolated outside the recommended precise part of
the arc.

Finally the position and altitude is interpolated. This interpolation is
done in pseudo-Cartesian coordinates: latitude, longitude, and altitude
are treated as polar coordinates and are changed to pseudo-Cartesian,
and are converted back after interpolation of these coordinates.

Arguments:
 TIME    (input): Epoch at which the position of the satellite is requested.
                  This epoch has to be given in UTC seconds since
                  1.0 January 1985.
 LAT    (output): Geodetic latitude of the sub-satellite point at the
                  requested epoch (in degrees).
 LON    (output): Longitude of the sub-satellite point at the requested
                  epoch (in degrees, interval -180 to 180 degrees).
 ORBIT  (output): Orbital altitude above the GRS80 ellipsoid (in meters),
                  interpolated at the requested epoch.
 PATH    (input): (1) Name of the directory in which the ODRs are stored,
                      e.g. '/home/ers1/ODR/DGM-E04'
                  (2) Name of the ODR file, preceeded by a +
 VERBOSE (input): Gives some processing information if .TRUE.
 GETORB (output):     0 = no error/warning
                  Fatal error codes:
                      1 = error opening ODR file,
                      2 = no ODR for requested epoch, 3 = too many records,
                  Warning codes (non fatal):
                     -1 = time within ODR, but outside precise part.
```

2.35 GLOBPRES - Determine global mean pressure over oceans

```
    SUBROUTINE GLOBPRES (TYPE, UTC, VALUE)
    INTEGER*4 TYPE
    REAL*8    UTC, VALUE

Determine the global mean pressure over oceans by linear
interpolation in a table of 6-hourly global mean pressure
values based on ECMWF and NCEP grids of sea surface pressure.

The value returned can be the one based on NCEP grids
(available for the GEOSAT and GFO time frames) or on
ECMWF grids (available for the T/P time frame). Also
smoothing can be applied. The smoother is a windowed sinc
function with a (2 days)**(-1) cut-off frequency.

Input arguments:
 TYPE  : Type of value requested
     =1: Value from ECMWF pressure fields
     =2: Idem, smoothed
     =3: Value from NCEP pressure fields
     =4: Idem, smoothed
 UTC   : Time in UTC seconds since 1.0 Jan 1985

Output argument:
 VALUE : Global mean pressure over oceans (mbar)
```

2.36 GRDATE – Get current date

```
    SUBROUTINE GRDATE (DATE)
    CHARACTER*(*) DATE

This routine returns the current date in format '20-Jan-1998 18:22:13'
(this is a total of 20 characters, the rest of DATE will be filled with
spaces).

Output argument:
 DATE : The current date in the format 20-Jan-1998 18:22:13
```

2.37 HELMERT1 – Determine Helmert transformation coefficients

```
    FUNCTION HELMERT1 (N, XYZFROM, XYZTO, SIGMA, COEFF, WRMS)
    INTEGER HELMERT1, N
    REAL*8 XYZFROM(3,N), XYZTO(3,N), SIGMA(3,N), COEFF(7), WRMS(2)
```

```
This routine determines the Helmert transformation coefficients by
matching the Cartesian coordinates of N points stored in the array
XYZFROM with those stored in XYZTO. The solution of the seven
transformation coefficients are obtained by least squares fitting. The
different points may be assigned different weights by means of the
standard deviations stored in the array SIGMA. Points can be ignored
by giving a negative SIGMA. When SIGMA(1,1)=0, all points will get equal
weights.

The seven Helmert transformation coefficients are stored in the array
COEFF. They are:

COEFF(1..3) : Translation vector (X,Y,Z) in meters
COEFF(4)    : Scale difference
COEFF(5..7) : Rotations around the X, Y and Z axes in radians

Arguments:
 N       (input) : Number of points to transform
 XYZFROM (input) : XYZ-Coordinates of the points to transform
 XYZTO   (input) : Reference XYZ-Coordinates
 SIGMA   (input) : Standard deviation in XYZ of each of the points
 COEFF  (output) : Helmert transformation coefficients (see above)
 WRMS   (output) : Weighted RMS residual. Two values:
                   WRMS(1) = a apriori, WRMS(2) = a posteriori
 HELMERT1 (output) : Returned error value:
                   0 = No errors, 1 = Insufficient number of points
                   2 = Too few weighted points,
                   3 = Error in normal matrix factorisation
                   4 = Error in normal matrix solve

Note that the position coordinates in XYZFROM and XYZTO and the
standard deviations SIGMA all have to be in the same units as
COEFF(1..3).
```

2.38 HELMERT2 – Perform Helmert transformation of a number of points

```
    SUBROUTINE HELMERT2 (COEFF, N, XYZFROM, XYZTO)
    INTEGER N
    REAL*8 COEFF(7), XYZFROM(3,N), XYZTO(3,N)
```

This routine performs the Helmert transformation of N points of which the Cartesian coordinates are given in the array XYZFROM. The result ends up in the array XYZTO. XYZFROM and XYZTO may share the same memory space.

The seven Helmert transformation coefficients are stored in the array COEFF. They are:

```
COEFF(1..3) : Translation vector (X,Y,Z) in meters
COEFF(4)    : Scale difference
COEFF(5..7) : Rotations around the X, Y and Z axes in radians
```

Note that the position coordinates in XYZFROM and XYZTO have to be in the same units as COEFF(1..3).

```
Arguments:
 COEFF   (input) : Helmert transformation coefficients (see above)
 N       (input) : Number of points to transform
 XYZFROM (input) : XYZ-Coordinates of the points to transform
 XYZTO  (output) : Transformed XYZ-Coordinates
```

2.39 I2SWAP – Swap one or more 2-byte integers

```
    SUBROUTINE I2SWAP (N, I)
    INTEGER*4 N
    INTEGER*2 I(N)
```

This routine swaps a 2-byte integer variable or the elements of a 2-byte integer array, i.e. it converts their representation between little and big endian (in either direction).

```
Arguments:
 N  (input) : Number of elements to swap
 I  (input) : Integer or integer array to swap
   (output) : Swapped integer or integer array
```

```
Examples:
    CALL I2SWAP(1,I)    ! Swap integer I
    CALL I2SWAP(4,I)    ! Swap integer elements I(1) through I(4)
    CALL I2SWAP(3,I(7)) ! Swap integer elements I(7) through I(9)
```

2.40 I4SWAP – Swap one or more 4-byte integers

```
    SUBROUTINE I4SWAP (N, I)
    INTEGER*4 N
    INTEGER*4 I(N)

This routine swaps a 4-byte integer variable or the elements of a
4-byte integer array, i.e. it converts their representation between
little and big endian (in either direction).

Arguments:
 N  (input) : Number of elements to swap
 I  (input) : Integer or integer array to swap
   (output) : Swapped integer or integer array

Examples:
    CALL I4SWAP(1,I)    ! Swap integer I
    CALL I4SWAP(4,I)    ! Swap integer elements I(1) through I(4)
    CALL I4SWAP(3,I(7)) ! Swap integer elements I(7) through I(9)
```

2.41 INTAB2 – Interpolate a table using 2nd order polynomial (parabola)

```
    SUBROUTINE INTAB2 (NDIM, NVEC, TABLE, T1, TN, T, VEC)
    INTEGER*4 NDIM, NVEC
    REAL*8  TABLE(NDIM,NVEC), T1, TN, T, VEC(NDIM)

This subroutine interpolates a NDIM dimensional vector VEC at a given
time T from a table of NVEC (>=3) vectors, equally spaced in time, starting
at T1 and ending at TN. 'Time' can be seen in this respect as any
independent variable of which the vector is a function. A 2nd order
polynomial interpolation is used to interpolate the vector.

Arguments:
 NDIM  (input) : Dimension (= number of elements) of the vector to be
                 interpolated. Thus NDIM=1 denotes the interpolation of a
                 scalar.
 NVEC  (input) : Number of table entries. The first table entry is for
                 time T1, the last for time TN. NVEC must be greater or
                 equal to 3.
 TABLE (input) : Array containing NVEC vectors of dimension NDIM.
 T1    (input) : Time corresponding to the first vector stored in TABLE
                 ( TABLE(1,1)...TABLE(NDIM,1) )
 TN    (input) : Time corresponding to the last vector stored in TABLE
                 ( TABLE(1,NVEC)...TABLE(NDIM,NVEC) )
 T     (input) : Time at which a vector must be interpolated.
 VEC  (output) : Interpolated NDIM-dimensional vector at time T. The dimension
                 defined in the calling (sub)program must be at least NDIM.
```

2.42 INTAB8 – Interpolate a table using 8th order Legendre interpolation

```
    SUBROUTINE INTAB8 (NDIM, NVEC, TABLE, T1, TN, T, VEC)
    INTEGER*4 NDIM, NVEC
    REAL*8  TABLE(NDIM,NVEC), T1, TN, T, VEC(NDIM)
```

```
This subroutine interpolates a NDIM dimensional vector VEC at a given
time T from a table of NVEC (>=9) vectors, equally spaced in time, starting
at T1 and ending at TN. 'Time' can be seen in this respect as any
independent variable of which the vector is a function. A 8th order
Legendre interpolation is used to interpolate the vector.

Arguments:
 NDIM  (input) : Dimension (= number of elements) of the vector to be
                 interpolated. Thus NDIM=1 denotes the interpolation of a
                 scalar.
 NVEC  (input) : Number of table entries. The first table entry is for
                 time T1, the last for time TN. NVEC must be greater or
                 equal to 9.
 TABLE (input) : Array containing NVEC vectors of dimension NDIM.
 T1    (input) : Time corresponding to the first vector stored in TABLE
                 ( TABLE(1,1)...TABLE(NDIM,1) )
 TN    (input) : Time corresponding to the last vector stored in TABLE
                 ( TABLE(1,NVEC)...TABLE(NDIM,NVEC) )
 T     (input) : Time at which a vector must be interpolated.
 VEC  (output) : Interpolated NDIM-dimensional vector at time T. The dimension
                 defined in the calling (sub)program must be at least NDIM.
```

2.43 INTER8 – Interpolate vector using 8th order Legendre interpolation

```
    SUBROUTINE INTER8 (N, T, V, VT)
    INTEGER*4 N
    REAL*8  T, V(N,9), VT(N)
```

```
This subroutine interpolates a N dimensional vector V at time T given
9 equi-distant N dimensional vectors V. These vectors refer to:
 T = 0  ->  V(1,1) ... V(N,1)
 T = 1  ->  V(1,2) ... V(N,2)
  ...
 T = 8  ->  V(1,9) ... V(N,9)

Arguments:
N   (input) : Number of dimensions.
T   (input) : Time at which VT must be interpolated. E.g. T=3.5 refers
              to interpolation between V(.,4) and V(.,5).
V   (input) : 9 consecutive vectors of dimension N.
VT (output) : Interpolated vector at time T.
```

2.44 IQSORT – Make a quick sort of an INTEGER*4 array

```
    SUBROUTINE IQSORT (INDEX, VALUE, NR)
    INTEGER*4 NR, INDEX(NR)
    INTEGER*4 VALUE(*)
```

This routine quick-sorts an INTEGER*4 array INDEX according to the corresponding value in array VALUE of type INTEGER*4.
The array INDEX contains NR pointers to the values in array VALUE, which does not have to be equally large.

Arguments:
 INDEX (input) : One-dimensional array of index numbers (INTEGER*4)
      (output) : Row of index numbers sorted on corresponding value
 VALUE (input) : One-dimensional array of values (INTEGER*4)
 NR   (input) : Number of indices/values

Example 1:
Assume you have 6 values 100, 10, 11, 21, 17, and 90, stored in array VALUE. These values have to be sorted in ascending order. Your input will be like this:

```
  INDEX    1    2    3    4    5    6
  VALUE  100   10   11   21   17   90
     NR    6
```

After running IQSORT, the output will look like this:

```
  INDEX    2    3    5    4    6    1
  VALUE  100   10   11   21   17   90
     NR    6
```

Note that the array VALUE has not been changed, but that INDEX is sorted by the order in which VALUE should be sorted. You may print the values in the ascending order as follows:

```
    DO I=1,NR
       WRITE (*,*) VALUE(INDEX(I))
    ENDDO
```

Example 2:
It is also possible that the indices are not in logical order. As long as they point to the respective locations of the values in array VALUE, the sorting will be done accordingly. Assume you have the following input:
(values ** are irrelevant)

```
  INDEX    1    2    4    5    6    8
  VALUE  100   10   **   21   17   90   **   1
     NR    6
```

Then after running IQSORT, the result will be:

```
  INDEX    8    2    5    4    6    1
  VALUE  100   10   **   21   17   90   **   1
     NR    6
```


Printing the values in ascending order after this goes the same as in Example 1.

2.45 ISNAN – Checks if value is Not-A-Number

```
    FUNCTION ISNAN (X)
    implicit none
    LOGICAL ISNAN
    REAL*8  X
```

The function ISNAN returns TRUE is X is NAN (Not-A-Number)

2.46 J2000 – Transform ephemerides between J2000 and TOD

```
    SUBROUTINE J2000 (ICH, MJD, DPSI, DEPS, NDIM, A, B)
    REAL*8 MJD, DPSI, DEPS, A(*), B(*)
    INTEGER*4 NDIM, ICH

This routine transforms ephemerides from J2000 to True-Of-Date
(TOD) reference frame or vice versa.

A and B are state vectors (position and/or velocity) in the inertial
reference frame.
A and B may share the same array in the calling routine.

NDIM indicates whether the vectors contain 3 or 6 coordinates.
ICH determines the transformation direction (to or from J2000).
The epoch of the TOD system is given by MJD (Terrestial Time).

Note 1: J2000 is the system of reference date 1.5 Jan 2000.
Note 2: Following recommendations by IERS [McCarthy, 1996] the
        expressions are to be evaluated as a function of Terrestial
        Time and not Barycentric Dynamical Time.
Note 3: Routine is based on IAU 1980 nutation theory [Seidelmann, 1982;
        Wahr, 1981]. The coefficient values from IERS 1996
        [McCarthy, 1996] are used to compute the nutation angles in
        longitude and obliquity.
        These angles can be corrected for the observed celestial pole
        offsets (DPSI and DEPS) reported in the IERS Bulletins.

Arguments:
 ICH  (input): Direction of the transformation:
                1 = J2000 -> TOD
               -1 = TOD -> J2000
 MJD  (input): Reference epoch of the TOD system (Mean Julian Date)
 DPSI (input): Correction on nutation in longitude (milliarcseconds)
 DEPS (input): Correction on nutation in obliquity (milliarcseconds)
 NDIM (input): Dimension of the state vector:
                3 = position vector only
                6 = full state vector (position and velocity)
 A    (input): Input state vector
 B   (output): Transformed state vector

References:

D. D. McCarthy, "IERS Conventions (1996)", IERS Technical Note 21,
Central Bureau of IERS, Observatoire de Paris, 1996.

O. Montenbruck and E. Gill, "Satellite Orbits: Models, Methods,
Applications", Springer Verlag, 2000.

P. K. Seidelmann, "1980 IAU Nutation: The final report of the IAU
Working Group on Nutation", Celest. Mech., 27, 79-106, 1982.

J. M. Wahr, "The forced nutations of an elliptical, rotating, elastic,
and oceanless Earth", Geophys. J. Roy. Astron. Soc., 64, 705-727, 1981.
```

2.47 J2000P – Compute precession matrix


```
    SUBROUTINE J2000P (XMJD,PREC)
    REAL*8 XMJD, PREC(3,3)

Computation of precession matrix from J2000 to epoch XMJD
```

2.51 LAND – Check whether point is over land

```
    FUNCTION LAND (LAT, LON)
    LOGICAL LAND
    REAL*4  LAT, LON
```

```
This function queries the direct-access binary data file "ut_land12.bin"
to determine if a given latitude/longitude location is on land or over the
ocean.
It returns a .TRUE. value if the point is on land or a .FALSE. value if the
point is over the ocean.  The landmask file is a 1/12th degree grid of the
world, with single bits set to 1 for Land, 0 for Ocean.

The file "ut_land12.bin" must be located in directory $ALTIM/data.

Arguments:
 LAT   (input): Latitude of the point (deg)
 LON   (input): Longitude of the point (deg) (non restricted)
 LAND (output): .TRUE. if point is over land, .FALSE. otherwise.
```

2.52 LISTARGS – List all input arguments

```
    SUBROUTINE LISTARGS (UNIT, LENGTH)
    INTEGER UNIT, LENGTH
```

```
This subroutine lists all input arguments from the command line,
including the name of the command. Output will be written
to unit number UNIT. Lines will be cut before the maximum length
of the output lines, specified by LENGTH is encountered.
Specify LENGTH=0 for infinitely long lines.

Arguments:
 UNIT   (input): Unit number for output
 LENGTH (input): Maximum length of output lines (0 for infinite lines)
```

2.53 LOADCOORD – Read station coordinates from file

```
    SUBROUTINE LOADCOORD (FILENM, MJDREF, N, STA, POS, SIG)
    CHARACTER*(*) FILENM
    INTEGER MJDREF, N, STA(*)
    REAL*8  POS(6,*),SIG(6,*)

This routine will load station coordinates from a variety of file
formats, such as
- XYZ format used by DUT/DEOS
- LST format used by UT/CSR
- SSC format used by ITRF
- SINEX format

Apart from the coordinates, the routine also attempts to read the
velocities and the standard deviations of the station positions and the
velocties.
The X, Y, Z coordinates are stored in POS(1..3,I) and the
X, Y, Z velocties are stored in POS(4..6,I), while STA(I) is the
station number. The respective standard deviations (when available)
are stored in SIG(1..6,ISTA).

The XYZ format may contain the line 'NUVEL'. In this case the NUVEL1A
velocities are calculated.

Arguments:
 FILENM  (input) : Name of the file in which coordinates are stored
                   Use "-" for standard input.
 MJDREF (output) : Reference epoch of the coordinates in MJD
 N      (output) : Number of stations in the file
 STA    (output) : Array containing the station numbers
 POS    (putput) : Array containing the XYZ coordinates (in meters)
                   and the XYZ velocities (in meters/year)
 SIG    (output) : Standard deviations of the XYZ coordinates
                   (in meters) and of the XYZ velocities (in
                   meters/year), when available (otherwise 0)
```

2.54 LOWERCASE – Change string to lower case

```
    SUBROUTINE LOWERCASE (STRING)
    CHARACTER*(*) STRING

This routine changes all upper case characters in a string to lower
case characters.

Argument:
 STRING (input/output): character string
```

2.55 LTLEND – Determined byte order of integers

```
    FUNCTION LTLEND()
    LOGICAL  LTLEND

This function returns .TRUE. if your system uses the LITTLE ENDIAN
notation of INTEGERS. For BIG ENDIAN notation, it returns .FALSE.

The terms LITTLE ENDIAN and BIG ENDIAN refer to the order of the
bytes in an INTEGER word, both for two-byte and four-byte integers.
Within a BYTE, the BITS are always ordered most significant to
least significant. Within a WORD the BYTES may be ordered:
- Most significant to least significant: BIG endian
  (IBM RS6000, Sun, SGI, Convex, MacIntosh).
- Least significant to most significant: LITTLE endian
  (PC, VAX, DEC).

Arguments:
 (none)
Returned value:
 LTLEND : .TRUE. for LITTLE endian notation of integers
          .FALSE. for BIG endian notation of integers
```

2.56 MALLOCF/DALLOCF – Variable memory allocation

2.56.1 MALLOCF – Allocate memory block

```
    FUNCTION MALLOCF (SIZE, POINTER)
    INTEGER MALLOCF, SIZE, POINTER
```

This function allocates a memory block of SIZE bytes. At return, POINTER is the pointer to the memory block. If the allocation is successful, MALLOCF will be a zero.

The function should be called as:

```
    IER = MALLOCF (SIZE, POINTER)
```

Then the memory block can be used in your program. Obviously, you can not use POINTER as an array. You should use it in this fashion, e.g.,

```
    CALL FILL (SIZE/4, %val(POINTER))
```

with the routine FILL defined as:

```
    SUBROUTINE FILL(N,X)
    INTEGER N,I
    REAL X(*)
    DO I=1,N
       X(I)=1e0
    ENDDO
    END
```

To deallocate the memory (i.e., to give the memory free), use the function DALLOCF

Arguments:
 SIZE   (input): Size of the memory block in bytes
 POINTER (output): Pointer to the memory block
 MALLOCF (output): = 0 on proper return, = 1 on error.

2.56.2 DALLOCF – Deallocate memory block

```
    SUBROUTINE DALLOCF (POINTER)
    INTEGER POINTER
```

This subroutine deallocates a memory block. This means that the memory allocated by MALLOCF will be given free for other use. POINTER is the pointer to the memory block.

The routine should be called as:

```
     CALL DALLOCF (POINTER)
```

Then the memory block can be used again by others.

Argument:
 POINTER (input): Pointer to the memory block

2.57 MATSY1 - Fill an integer array of pointers for packed matrices

```
    SUBROUTINE MATSY1 (N, H)
    INTEGER N, H(N)
```

This subroutine fills an integer with pointers that can be used to find the index of elements of an UPPER packed symmetric matrix as used in LAPACK and the (old) NUMLIB.

Example: A(.) is an upper packed symmetric matrix. The element (I,J) has index H(I)+J for I>J or H(J)+I for I<J

Remark: H(I)=I*(I-1)/2

2.58 MEMLOC/MEMGET/MEMPUT – Direct manipulation of memory location

```
    FUNCTION MEMLOC (VAR)
    INTEGER*4 MEMLOC, VAR

    SUBROUTINE MEMGET (PNT, NR, VAR)
    INTEGER*4 PNT, NR, VAR

    SUBROUTINE MEMPUT (PNT, NR, VAR)
    INTEGER*4 PNT, NR, VAR
```

The function MEMLOC returns the location of a variable in memory. The returned value in MEMLOC is basically a pointer to variable VAR. VAR can be of any type.

The routine MEMGET gets NR bytes out of the memory location pointed to by PNT and places them into variable VAR. VAR can be of any type.

The routine MEMPUT copies NR bytes from the variable VAR and place them into the memory location pointed to by PNT. VAR can be of any type.

These routine were created to avoid the use of the %VAL construct in subroutine calls.

2.59 MCW – Convert SIGMA0 to wind speed using Modified Chelton-Wentz model

```
FUNCTION MCW (SIGMA0, U10)
REAL*8    SIGMA0, U10
INTEGER*4 MCW
```

```
This function computes wind speed (U10, referenced to 10 m above
sea level) from altimeter backscatter coefficient (SIGMA0) based
on the Modified Chelton-Wentz algorithm [Witter and Chelton, 1991].

The MCW model function provides wind speed estimates for SIGMA0 values
ranging from 19.6 dB to 7.0 dB at intervals of 0.2 dB, corresponding
to 10 m wind speeds between 0 and 20.2 m/s. This subroutine
interpolates in that table, found in Witter and Chelton [1991].

In case SIGMA0 exceeds the upper limit of the table (19.6 dB), the
wind speed is assumed to be zero. At the same time MCW is set to +1.
Wind speeds for SIGMA0 less than the lower limit of the table (7.0 dB)
are determined by linear extrapolation of the first two table values.
In that case MCW is set to -1.
When SIGMA0 is VOID (> 1d20) the value returned for U10 is VOID (1d30)
and the function value MCW is set to 2.
When SIGMA0 is NaN, U10 will be NaN and MCW is set to 2.

Arguments:
 SIGMA0 (input): Backscatter coefficient in dB
 U10   (output): Wind speed (in m/s) referenced to 10 m above sea level
 MCW      (out): Function value indicating success of table
                 interpolation:
                   0 = success, SIGMA0 value within bounds
                  -1 = SIGMA0 value below minimum; linear extrapolation
                  +1 = SIGMA0 value above maximum; wind speed set to 0
                   2 = SIGMA0 > 1d20

Reference:

Witter D. L. and D. B. Chelton, A Geosat altimeter wind speed
algorithm and a method for wind speed algorithm development,
J. Geophys. Res., 96(C5), pp 8853-8860, 1991.
```

2.60 MDATE – Convert MJD to (YY)YYMMDD or v.v.

```
    FUNCTION MDATE (I, J)
    implicit none
    INTEGER  MDATE, I, J

This function converts Modified Julian Dates (MJD) to Year-Month-Day
in the format YYMMDD or YYYYMMDD, or vice versa.

Dates in the 21st century are given in 6-digits, similar to dates in
the 20th century. So 13 October 2001 is written as 011013.
Use I=1 to convert MJD to YYMMDD and I=2 to convert YYMMDD to MJD.
The algorithm is valid for the years 1955 till 2054 only.

Optionally, one can enter the YYYYMMDD including the century indication,
or have it returned by MDATE. Thus 13 October 2001 is written as 20011013.
Use I=2 to convert YYYYMMDD to MJD and I=3 to convert MJD to YYYYMMDD.
With these options, the algorithm is valid for 1 March 1900 until
28 February 2100.

When YYYYMMDD or YYMMDD is input to the function, a month number of
0 or 13 will also work. Day numbers 0 and 32 are also excepted.
For example: 000000 will be recognized as 30 November 1999.

Arguments:
 I      (input): I=1, convert        MJD -> YYMMDD
                 I=2, convert (YY)YYMMDD -> MJD
                 I=3, convert        MJD -> YYYYMMDD
 J      (input): Either MJD, YYMMDD, or YYYYMMDD (depending on I).
 MDATE (output): Either YYMMDD, MJD, or YYYYMMDD (depending on I).
```

2.61 YMD2MJD – Convert YY, MM and DD to MJD

```
    SUBROUTINE YMD2MJD (YY, MM, DD, MJD)
    implicit none
    INTEGER YY, MM, DD, MJD
    integer t1901,cal(0:13,0:1),leap,j
    real*8  year
    parameter (t1901=15384,year=365.25d0)
    save cal
    data cal /-31,0,31,59,90,120,151,181,212,243,273,304,334,365,
   |          -31,0,31,60,91,121,152,182,213,244,274,305,335,366/
    j=yy
    if (j.lt.55) then
       j=j+100
    else if (j.gt.1900) then
       j=j-1900
    endif
    leap=0
    if (mod(j,4).eq.0) leap=1
    mjd=t1901+dint((j-1)*year)+cal(mm,leap)+dd
    end
```

2.62 MJD2YMD – Convert MJD to YY, MM and DD


```
    SUBROUTINE MJD2YMD (MJD, YY, MM, DD)
    implicit none
    INTEGER MJD, YY, MM, DD
    integer t1901,cal(0:13,0:1),leap,t
    real*8  year
```

2.63 MJDATE – Convert MJD to YYMMDD or v.v.

```
    SUBROUTINE MJDATE(IND, MJD, YYMMDD, YY, MM, DD)
    INTEGER*4 IND, MJD, YYMMDD, YY, MM, DD

This function converts Modified Julian Dates (MJD) to Year-Month-Day
in the format YYMMDD, or vice versa.
The algorithm is good for the years 1900 till 1999 only.

Arguments:
 IND      (input): IND=1, convert MJD --> YYMMDD and YY, MM, DD
                   IND=2, convert YYMMDD --> MJD and YY, MM, DD
                   IND=3, convert YY, MM, DD --> MJD
                   IND=4, convert YYMMDD --> YY, MM, DD
 YYMMDD  (in/out): Date in the form 851231.
 YY, MM, DD (i/o): Date as separate integers, e.g. 85, 12, 31.
 MJD     (in/out): Modified Julian Day, e.g. 46430
```

2.64 NFF – Check status of NetCDF return code

```
    SUBROUTINE NFF(IOS)
    INTEGER*4 IOS

This routine checks the return value of any NetCDF call and verifies
if an error had resulted. If so, the program will terminate.

Example:
    call nff(nf_open(filenm,nf_nowrite,ncid))
```

2.65 NOISE – Create Gaussian noise

```
      FUNCTION NOISE ()
      REAL*8 NOISE

This function returns a value from a Gaussian distribution with
expectation = 0 and variance = 1
To give the "seed value" to the randomizer, use the standard Fortran
routine SRAND

Example:
      REAL*8 R, NOISE
      INTEGER SEED

      CALL SRAND(SEED)
      R = NOISE()

Argument:
NOISE (output): Random value from Gaussian distribution
```

2.66 NUVEL1A – Compute station velocities according to NUVEL1A

```
    SUBROUTINE NUVEL1A (PLATE, XYZ, UVW)
    INTEGER*4 PLATE
    REAL*8    XYZ(3), UVW(3)
```

```
This routine determines station velocities according to the NUVEL1A
NNR plate motion model.

Arguments:
  PLATE (input) : Number of the reference plate
  XYZ   (input) : XYZ coordinates of station (m)
  UVW  (output) : XYZ velocities according to NUVEL1A (m/s)
```

2.67 ODRINFO – Open ODR file and return some information

```
    FUNCTION ODRINFO (UNIT, FILENM, SATEL, REP, ARC, VER, NREC,
   |   TIME0, TIME1, TSTEP, BEGIN, END, REV)
    INTEGER*4 ODRINFO, UNIT, REP, ARC, VER, NREC,
   | TIME0, TIME1, TSTEP, BEGIN, END
    REAL*8    REV
    CHARACTER FILENM*(*), SATEL*8
```

```
This routine opens an ODR (Orbital Data Record) with filename FILENM,
on unit UNIT and returns some information on the contents of the file
Upon return the ODR file will NOT be closed, unless an error occurs.

This routine can also be called as a function, in that case, ODRINFO
will hold the value 0 when the old ODR format (specifier: @ODR) is
read, and 1 when the new ODR format is encountered (specifier: xODR).

Arguments:
  UNIT   (output): File unit number.
  FILENM  (input): Name of ODR file.
  SATEL  (output): Satellite name.
  REP    (output): Length of the repeat cycle in 10^-3 days.
  ARC    (output): Arc number.
  VER    (output): Orbit version number.
  NREC   (output): Number of records (0 if file not found).
  TIME0  (output): Time in UTC seconds of first record.
  TIME1  (output): Time in UTC seconds of last record.
  TSTEP  (output): Time-step in seconds.
  BEGIN  (output): Begin of precise part of arc (in UTC seconds).
  END    (output): End of precise part of arc (in UTC seconds).
  REV    (output): Length of a revolution (in seconds).
  ODRINFO(output): 0 = old format (@ODR), 1 = new format (xODR),
                   -1 = can not find file, -2 = format error,
                   -3 = unknown satellite
```

2.68 POLCAR – Convert Polar to Cartesian

```
    SUBROUTINE POLCAR (LAT, LON, R, VECTOR)
    REAL*8 LAT, LON, R, VECTOR(3)

Convert polar coordinates (LAT,LON,R) to Cartesian VECTOR() = (X,Y,Z)

Arguments:
LAT     (input) : Latitude in radians
                  = angle from XY-plane towards +Z-axis
LON     (input) : Longitude in radians
                  = angle in XY-plane measured from +X-axis towards +Y-axis
R       (input) : Radius
VECTOR (output) : Vector of X, Y, and Z coordinates.
```

2.69 QUAINT – Quadratic interpolation

```
    FUNCTION QUAINT (T, H, TT)
    REAL*8 QUAINT, T(0:2), H(0:2), TT

Produce interpolated value QUAINT on epoch TT from tables H() and T(),
using parabolic fit through 3 points (need not be equidistant).

Arguments:
 T      (input): Time tags for values in table H
 H      (input): Values on time tags T
 TT     (input): time-tag on which interpolated value must be determined
 QUAINT (output): Interpolated value.
```

2.70 REGRES – Regression analysis

```
    SUBROUTINE REGRES (N, X, Y, A, B, R)
    INTEGER*4 N
    REAL*8  X(N), Y(N), A, B, R

This subroutine computes the best fitting straight line trough a number
of points with coordinates (X,Y). Upon return, A and B will be the
coefficients of the line

    Y = A + B * X

that is the best fit through the data points. R is the correlation
between the fit and the data.

Arguments:
 N  (input) : Number of data points
 X  (input) : Array of X coordinates of the data points
 Y  (input) : Array of Y coordinates of the data points
 A (output) : Coefficient of linear fit (Y = A + B * X)
 B (output) : Coefficient of linear fit (Y = A + B * X)
 R (output) : Correlation
```

2.71 ROTATE – Rotate coordinate system

```
    SUBROUTINE ROTATE (AXIS, ANGLE, A, B)
    INTEGER*4 AXIS
    REAL*8 ANGLE, A(3), B(3)
```

```
This routine computes the object of a coordinate vector while
rotating the coordinate system over an angle 'ANGLE' around the
coordinate axis 'AXIS'. Vector A in the original system becomes
vector B in the new system. In the call to this subroutine arrays
A and B may be the same array.

Arguments:
 AXIS  (input): Number of the coordinate axis (1,2,3)
 ANGLE (input): Rotation angle (right-handed) in radians
 A     (input): Vector in the original system
 B    (output): The same vector, but now in the rotated coordinate
                system
```

2.72 SCAPRD – Scalar product of two 3D vectors

```
    FUNCTION SCAPRD (X, Y)
    REAL*8 SCAPRD, X(3), Y(3)
```

```
This function returns the scalar (or inner) product of two 3-dimensional
vectors X and Y

Arguments:
 X, Y    (input): Two 3-dimensional vectors
 SCAPRD (output): Scalar (or inner) product of X and Y (X.Y)
```

2.73 SFDIST – Spherical distance between two points

```
    FUNCTION SFDIST (LAT0, LON0, LAT1, LON1)
    REAL*8 LAT0, LON0, LAT1, LON1, SFDIST
```

```
Compute spherical distance between two points of given geocentric latitude
and longitude. This function is optimized for computing very small spherical
distances.

Arguments:
 LAT0, LON0  (input): Geocentic latitude and longitude of one point (rad).
 LAT1, LON1 (output): Geocentic latitude and longitude of other point (rad).
 SFDIST     (output): Spherical distance in radians.
```

2.74 SEC85 – Convert MJD or YYMMDD or YYDDD to SEC85

```
    FUNCTION SEC85 (I, DATE)
    REAL*8 SEC85, DATE
    INTEGER*4 I

This function converts Modified Julian Dates (MJD) or Year-Month-Day
(YYMMDD) or Year-Day (YYDDD) or Year-Month-Day-Hours-Minutes-Seconds
(YYMMDDHHMMSS) to seconds from 1.0 Jan 1985 (SEC85). Fractions of days or
seconds can also be included.

Except for a 2-digit year indication (YY) it is now also possible to use
a 4-digit indication (YYYY). The routine automatically recognises the
form: whether YY or YYYY is specified.

The first parameter (I) defines the input format: I=1 indicates MJD,
I=2 means YYMMDD or YYYYMMDD, I=3 specifies YYDDD or YYYYDDD, and
I=4 implies YYMMDDHHMMSS or YYYYMMDDHHMMSS, I=5 is a combination of I=2
and I=4 (i.e, it takes YYMMDD, YYYYMMDD, YYMMDDHHMMSS and YYYYMMDDHHMMSS,
as input).

One can also choose NOT to specify the input format and let the function
'guess' which format it is. Obviously, there are limitations. it recognises
        YYMMDD :  000101 - 040101  ( 2000/01/01 - 2004/01/01 )
           MJD :   40587 - 53005   ( 1970/01/01 - 2004/01/01 )
         YYDDD :   53005 - 99365   ( 1953/01/05 - 2000/01/01 )
        YYMMDD :  500101 - 991232  ( 1950/01/01 - 2000/01/01 )
       YYYYDDD :  1950d3 - 2050d3  ( 1950/01/01 - 2050/01/01 )
      YYYYMMDD :  1950d4 - 2050d4  ( 1950/01/01 - 2050/01/01 )
  YYMMDDHHMMSS :     1d8 - 1d12    ( 1950/01/01 - 2050/01/01 )
YYYYMMDDHHMMSS : 1950d10 - 2050d10 ( 1950/01/01 - 2050/01/01 )

Arguments:
 I      (input): I=1, convert MJD to SEC85
                 I=2, convert YYMMDD or YYYYMMDD to SEC85
                 I=3, convert YYDDD or YYYYDDD to SEC85
                 I=4, convert YYMMDDHHMMSS or YYYYMMDDHHMMSS to SEC85
                 I=5, convert [YY]YYMMDD[HHMMSS] to SEC85
                 I=0, convert any of the above to SEC85
 DATE   (input): Input value.
 SEC85 (output): Output value, seconds from 1.0 Jan 1985.
```

2.75 SLAND – Check whether point is over deep ocean

```
FUNCTION SLAND (LAT, LON)
LOGICAL SLAND
REAL*4  LAT, LON
```

This function queries the direct access binary file "sland3.bin" to determine if a given latitude/longitude location is over land or "shallow" seas (< 2251 m), or is over "deep" seas (> 2251 m).
The function returns a logical value of .TRUE. for land/shallow sea and a value of .FALSE. for deep sea points.

The data file "sland3.bin" is derived from Russ Agreen's sequential binary version. One-degree squares in the region 79 S -> 81 N; 0 -> 360 E are represented by bits within the mask file. Hence there are 360*160=57600 bits, which are contained within 7200 bytes, or 1800 integer*4 words. A bit "set" to "1" indicates a land/shallow sea 1-degree square, while a bit "cleared" to "0" indicates a deep sea square. The cycling of the indices has the bit number cycling fastest from 0->31, the longitude cycles next fastest from 0->360, and the latitude cycles slowest from -79->81. Since 360 bits do not divide evenly by the 32-bit I*4 word boundary, the array is 1-D vs. 2-D (lat/long). The element or word number within the mask array is determined from a combination of latitude and longitude: ilat=int(lat+79); ilon=int(lon); ibit=ilat*360+ilon; iwrd=ibit/32+1; ibit=mod(ibit,32).
The zero bit of the first word of the array corresponds to 79S,0E and bit 31 of the 1800th word corresponds to 81N,360E. Note that the grid cells are centered on 1/2 degree lat/long points... it is the EDGES that fall on the whole degree boundaries.

Any locations south of 79S or north of 81N are considered land/shallow sea and a .TRUE. value is returned.

The file "sland3.bin" must be located in directory $ALTIM/data.

Arguments:

LAT	(input):	Latitude of the point (deg)
LON	(input):	Longitude of the point (deg)
SLAND	(output):	Flag indicating whether point is over deep ocean (.TRUE. if depth > 2251 m)

2.76 STATBAR – Print status bar

```
    SUBROUTINE STATBAR (FNC, N, TEXT)
    INTEGER*4     FNC, N
    CHARACTER*(*) TEXT
```

```
This routine prints a status bar during some processing loop, keeping
track of the progress of the processing. After each 2% of the processing
a # is drawn, until 100% of the processing is performed.
Upon initialisation, the maximum count to be reached should be specified.
After that, just the continuing count should be given and the routine
will update the status bar.
You may specify a header in the form

       2%  .    .    .    .   50%   .    .    .    .  100%
to be printed, plus a short text (max 25 characters).

Arguments:
FNC  (input): Function:
               0 = Initialise and print header
              -1 = Initialise but do not print header
              >0 = Update status bar
N    (input): Upon initialisation: maximum count
              During processing  : count
TEXT (input): Text to be printed in front of status bar
```

2.77 STATINFO – Return information for given laser station

```
    FUNCTION STATINFO (TIME, STATION, SITENAME, OCCODE, CODE,
   |  NOISE, WAVELENGTH, PLATE, ECCENTRICITY)
    REAL*8 TIME, ECCENTRICITY(3)
    CHARACTER SITENAME*32, OCCODE*6, CODE*4
    INTEGER*4 STATION, NOISE, WAVELENGTH, PLATE, STATINFO
```

For given STATION and TIME, this routine returns the correct WAVELENGTH, SITENAME, occupation CODE, and system NOISE. When TIME is >= 1d20 then TIME is not used as a selection criterion. The most recent information for STATION is returned.

The routine initialises upon first the first call. It requires the access to file ${ALTIM}/data/tables/system.data.

The routine uses a fast query mechanism to get the station information with only a little expense of memory.

Arguments:

TIME	(input)	: Time in MJD
STATION	(input)	: Station ID
SITENAME	(output)	: Name of the laser station site
OCCODE	(output)	: 6-letter occupation code
CODE	(output)	: 4-letter IGS code
NOISE	(output)	: Laser ranging noise level in centimetres
WAVELENGTH	(output)	: Wavelength of laser station in nanometres
PLATE	(output)	: Tectonic plate ID
ECCENTRICITY	(out)	: North, East, and Up component of eccentricity vector in metres.

Exit code:

STATINFO	(output)	: 0=no error, 1=no entry found

2.78 STATIS – Compute statistics of a series

```
    SUBROUTINE STATIS (N, X, XMEAN, XRMS, XSIGMA)
    INTEGER*4 N
    REAL*8  X(N), XMEAN, XRMS, XSIGMA
```

This routine computes the average (XMEAN), root-mean-squate (XRMS) and standard deviation (XSIGMA) of a series of N values (X).

Arguments:

N	(input)	: Number of values in the series.
X	(input)	: Series of values.
XMEAN	(output)	: Average of the series.
XRMS	(output)	: RMS of the series.
XSIGMA	(output)	: Standard deviation of the series.

2.79 STRF1970 – Fortran mimic of the C strftime routine

```
    FUNCTION STRF1970 (BUF, FMT, SEC)
    CHARACTER*(*) BUF, FMT
    INTEGER*4     SEC, STRF1970

This function will make it easy on you to convert your system time
(reckonned in seconds from 1970) into almost any sensible
string representation (see 'Format specifications' below).

At the function call the integer variable SEC will contain the number
of UTC seconds since 1.0 January 1970. This time stamp is then transfered
into a character string BUF based on the format specification FMT.

The function returns an integer value that determines the lengths
of the string after substitution according to the format, desregarding
trailing spaces.

If during substitution the capacity of the string buffer BUF is exceeded,
the function returns a value STRF1970=0 and the buffer will be empty
(all spaces).

Limitation: FMT and BUF may be no longer than 256 characters.

See also: STRF1985 and STRF2000

Arguments:

    BUF    (output) : Output string. The lengths of this string is
                      returned as the value of the function STRF1970
    FMT     (input) : Format specification of the output string
                      (see below)
    SEC     (input) : Clock value in seconds from 1970, 1985, or 2000
    STRF1970  (out) : Functions all return the length of BUF

Format specifications:

    A full list of the format specifications can be found in the manual
    of strftime ('man strftime'). This manual will give the exact list
    appropriate to your system. Nevertheless, we provide a list of the
    special sequences that can be used. But you can also use normal
    characters.

    %%  same as %
    %a  locale's abbreviated weekday name
    %A  locale's full weekday name
    %b  locale's abbreviated month name
    %B  locale's full month name
    %c  locale's appropriate date and time representation
    %C  locale's century number (the year divided by 100 and
        truncated to an integer) as a decimal number [00-99]
    %d  day of month ( 01 - 31 )
    %D  date as %m/%d/%y
    %e  day of month (1-31; single digits are preceded by a blank)
```

```
    %h  locale's abbreviated month name.
    %H  hour ( 00 - 23 )
    %I  hour ( 01 - 12 )
    %j  day number of year ( 001 - 366 )
    %KC locale's appropriate date and time representation
    %m  month number ( 01 - 12 )
    %M  minute ( 00 - 59 )
    %n  same as new-line
    %p  locale's equivalent of either AM or PM
    %r  time as %I:%M:%S [AM|PM]
    %R  time as %H:%M
    %S  seconds ( 00 - 61 ), allows for leap seconds
    %t  same as a tab
    %T  time as %H:%M:%S
    %U  week number of year ( 00 - 53 ), Sunday is the first day of week 1
    %w  weekday number ( 0 - 6 ), Sunday = 0
    %W  week number of year ( 00 - 53 ), Monday is the first day of week 1
    %x  locale's appropriate date representation
    %X  locale's appropriate time representation
    %y  year within century ( 00 - 99 )
    %Y  year as ccyy ( e.g. 1986)
    %Z  time zone name or no characters if no time zone exists

    Example: FMT="Today (%A %d/$b) I feel 100%%"
```

2.80 STRF1985 – Convert altimeter time to a character string

```
    FUNCTION STRF1985 (BUF, FMT, SEC)
    CHARACTER*(*) BUF, FMT
    INTEGER*4 SEC, STRF1985
```

```
This function will make it easy on you to convert altimeter time
(reckonned in seconds from 1985) into almost any sensible
string representation (see 'Format specifications' below).

At the function call the integer variable SEC will contain the number
of UTC seconds since 1.0 January 1985 . This time stamp is then transfered
into a character string BUF based on the format specification FMT.

The function returns an integer value that determines the lengths
of the string after substitution according to the format, desregarding
trailing spaces.

If during substitution the capacity of the string buffer BUF is exceeded,
the function returns a value STRF1985=0 and the buffer will be empty
(all spaces).

Limitation: FMT and BUF may be no longer than 256 characters.

See also: STRF1970 and STRF2000

Arguments:

BUF (output) : Output string. The lengths of this string is
```

```
returned as the value of the function
FMT (input) : Format specification of the output string
(see below)
SEC (input) : Time in UTC seconds from 1.0 Jan 1985
STRF1985 (out) : Function returns the length of BUF

Format specifications:

A full list of the format specifications can be found in the manual
of strftime ('man strftime'). This manual will give the exact list
appropriate to your system. Nevertheless, we provide a list of the
special sequences that can be used. But you can also use normal
characters.

%% same as %
%a locale's abbreviated weekday name
%A locale's full weekday name
%b locale's abbreviated month name
%B locale's full month name
%c locale's appropriate date and time representation
%C locale's century number (the year divided by 100 and
truncated to an integer) as a decimal number [00-99]
%d day of month ( 01 - 31 )
%D date as %m/%d/%y
%e day of month (1-31; single digits are preceded by a blank)
%h locale's abbreviated month name.
%H hour ( 00 - 23 )
%I hour ( 01 - 12 )
%j day number of year ( 001 - 366 )
%KC locale's appropriate date and time representation
%m month number ( 01 - 12 )
%M minute ( 00 - 59 )
%n same as new-line
%p locale's equivalent of either AM or PM
%r time as %I:%M:%S [AM|PM]
%R time as %H:%M
%S seconds ( 00 - 61 ), allows for leap seconds
%t same as a tab
%T time as %H:%M:%S
%U week number of year ( 00 - 53 ), Sunday is the first day of week 1
%w weekday number ( 0 - 6 ), Sunday = 0
%W week number of year ( 00 - 53 ), Monday is the first day of week 1
%x locale's appropriate date representation
%X locale's appropriate time representation
%y year within century ( 00 - 99 )
%Y year as ccyy ( e.g. 1986)
%Z time zone name or no characters if no time zone exists

Example: FMT="Today (%A %d/$b) I feel 100%%"
```

2.81 STRF2000 – Convert 3rd-milennium time to a character string

```
FUNCTION STRF2000 (BUF, FMT, SEC)
CHARACTER*(*) BUF, FMT
```

```
    INTEGER*4 SEC, STRF2000

This function will make it easy on you to convert altimeter time
(reckonned in seconds from 2000) into almost any sensible
string representation (see 'Format specifications' below).

At the function call the integer variable SEC will contain the number
of UTC seconds since 1.0 January 2000 . This time stamp is then transfered
into a character string BUF based on the format specification FMT.

The function returns an integer value that determines the lengths
of the string after substitution according to the format, desregarding
trailing spaces.

If during substitution the capacity of the string buffer BUF is exceeded,
the function returns a value STRF2000=0 and the buffer will be empty
(all spaces).

Limitation: FMT and BUF may be no longer than 256 characters.

See also: STRF1970 and STRF1985

Arguments:

BUF (output) : Output string. The lengths of this string is
returned as the value of the function
FMT (input) : Format specification of the output string
(see below)
SEC (input) : Time in UTC seconds from 1.0 Jan 2000
STRF2000 (out) : Function returns the length of BUF

Format specifications:

A full list of the format specifications can be found in the manual
of strftime ('man strftime'). This manual will give the exact list
appropriate to your system. Nevertheless, we provide a list of the
special sequences that can be used. But you can also use normal
characters.

%% same as %
%a locale's abbreviated weekday name
%A locale's full weekday name
%b locale's abbreviated month name
%B locale's full month name
%c locale's appropriate date and time representation
%C locale's century number (the year divided by 100 and
truncated to an integer) as a decimal number [00-99]
%d day of month ( 01 - 31 )
%D date as %m/%d/%y
%e day of month (1-31; single digits are preceded by a blank)
%h locale's abbreviated month name.
%H hour ( 00 - 23 )
%I hour ( 01 - 12 )
%j day number of year ( 001 - 366 )
```

```
%KC locale‘s appropriate date and time representation
%m month number ( 01 - 12 )
%M minute ( 00 - 59 )
%n same as new-line
%p locale‘s equivalent of either AM or PM
%r time as %I:%M:%S [AM|PM]
%R time as %H:%M
%S seconds ( 00 - 61 ), allows for leap seconds
%t same as a tab
%T time as %H:%M:%S
%U week number of year ( 00 - 53 ), Sunday is the first day of week 1
%w weekday number ( 0 - 6 ), Sunday = 0
%W week number of year ( 00 - 53 ), Monday is the first day of week 1
%x locale‘s appropriate date representation
%X locale‘s appropriate time representation
%y year within century ( 00 - 99 )
%Y year as ccyy ( e.g. 1986)
%Z time zone name or no characters if no time zone exists

Example: FMT="Today (%A %d/$b) I feel 100%%"
```

2.82 UPPERCASE – Change string to upper case

```
    SUBROUTINE UPPERCASE (STRING)
    CHARACTER*(*) STRING

This routine changes all lower case characters in a string to upper
case characters.

Argument:
 STRING (input/output): character string
```

2.83 VECELE – Convert state vector to Keplerian elements

```
    SUBROUTINE VECELE (VECT, ELEM, GM)
    REAL*8 VECT(6), ELEM(6), GM

This routine converts a state-vector (X, Y, Z, Xdot, Ydot, Zdot) to
the 6 Keplerian elements (a, e, i, O, w, th).
The gravitational parameter GM has to be given as well.

Arguments:
 VECT  (input): Array of position (and velocity):
                VECT(1)..VECT(3): Position vector (X,Y,Z) (meters),
                VECT(4)..VECT(6): Velocity vector (Xdot,Ydot,Zdot)
                                  (meter/sec).
 ELEM (output): Array of 6 Keplerian elements:
                ELEM(1): Semi-major axis (meters),
                ELEM(2): Eccentricity (unity),
                ELEM(3): Inclination (radians),
                ELEM(4): Right ascension of the ascending node (radians),
                ELEM(5): Argument of the pericenter (radians),
                ELEM(6): True anomaly (radians).
 GM    (input): Gravitational parameter of the central body
                (meter**3/sec**2). If GM<=0 then no velocity vector is
                computed.
```

2.84 VECNRM – Normalize a 3D vector

```
    SUBROUTINE VECNRM (X, V)
    REAL*8 X(3), V(3)

This routine computes the normalized vector V of a
3-dimensional vector X. ( V = X / || X || )

Arguments:
 X    (input): 3-dimensional vector
 V   (output): Normalized vector
```

2.85 VECPRD – Vectorial product of two 3D vectors

```
    SUBROUTINE VECPRD (X, Y, V)
    REAL*8 X(3), Y(3), V(3)

This routine computes the vectorial (or outer) product V of two
3-dimensional vectors X and Y (V = X * Y)

Arguments:
 X, Y (input): Two 3-dimensional vectors
 V   (output): Vectorial (or outer) product of X and Y
```

2.86 VNORM – Norm of a vector

```
    FUNCTION VNORM (V)
    REAL*8 VNORM, V(3)

This function returns the norm (or length) of a 3-dimensional vector V.

Arguments:
 V      (input): A 3-dimensional vector
 VNORM (output): The norm (or length) of V
```

2.87 XYZGEO – Convert ECF coordinates to geodetic longitude and latitude.

```
    SUBROUTINE XYZGEO (XYZ, R, LAT, LON, HEIGHT)
    REAL*8 XYZ(3), R, LAT, LON, HEIGHT

This subroutine converts the Earth Centered Fixed coordinates X, Y, and Z
to the geodetic latitude (LAT) and longitude (LON) and the height (HEIGHT)
above the reference ellipsoid GRS80.
An iterative method is used, where a parameter DZ (to be added to the Z
coordinate) has to converge up to 1.D-4 m. This results in an accuracy in
LAT of better than 1.D-11 radian and HEIGHT of 1.D-4 metres.
Generally this routine converges in 4-5 iterations, after which LAT
and HEIGHT are computed and control is given back to the calling
(sub)routine.
In any event, the iteration is stopped after 10 iterations, after which
a warning message is printed and results are returned.

Reference:
O. Montenbruck and E. Gill, "Satellite Orbits: Models, Methods,
Applications", Springer Verlag, p. 188, 2000.

Arguments:
XYZ(3)  (input) : Earth Centered Fixed coordinates X, Y, and Z (m).
R      (output) : Distance to geocenter (m).
LAT    (output) : Geodetic latitude (rad).
LON    (output) : Geodetic longitude (rad).
HEIGHT (output) : Height above the reference ellipsoid (m).
```

2.88 YMDHMS – Convert seconds past 1 Jan 1985 to YYMMDDHHMMSS.SS

```
    FUNCTION YMDHMS (SEC85)
    REAL*8 YMDHMS, SEC85

Arguments:
SEC     (input) : Number of seconds past 1.0 Jan 1985
YMDHMS (output) : Real number formatted as YYMMDDHHMMSS.SS
```