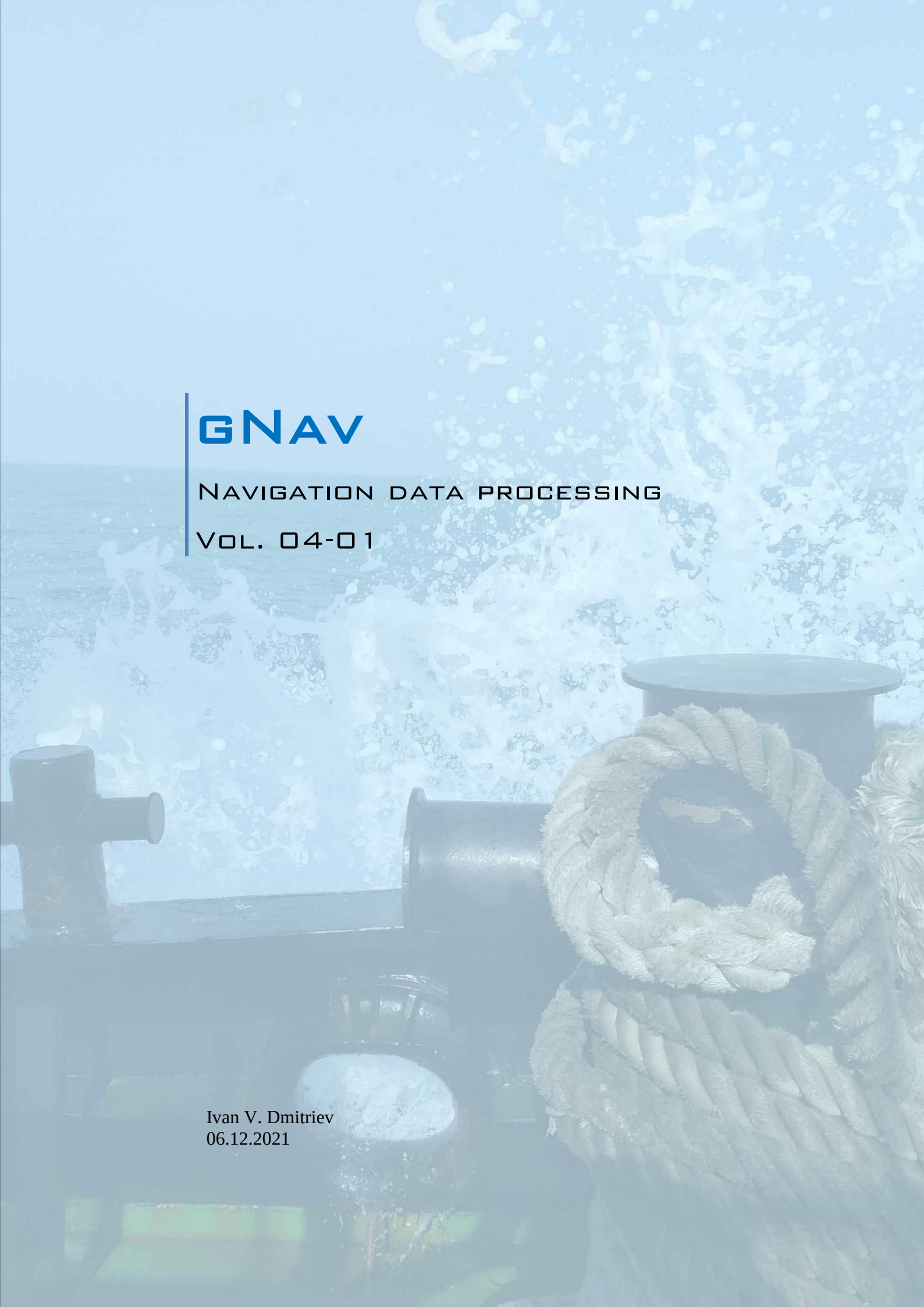

GNAV

Navigation data processing

Vol. 04-01

Ivan V. Dmitriev
06.12.2021

Contents

Tables list

Figures list

1. gNav general description

MatLab functions set for manipulations with Navigation data. There are: time/angle transformations, coordinates datum's and transformations, Nod's and Layback calculation.

The set's functions are shown in ***Table 1.1***. The functions are depended from structure's fields' names; there are several structures with predefined format (***Table 1.5***, ***Table 1.6***).

Table 1.1 gNav functions

Function name	Function description
Time/angle transformations	
gNavAng2Ang	Convert angle-to-angle
gNavTime2Time	Convert time-to-time
gNavDayCheck	Check than date was changed in 00:00:00; correct bad date.
gNavDV2DV	Convert distance and speed
Coordinates datum's and transformations	
gNavCoord2Coord	Convert coordinates between SensorNavigation datum to ProjectNavigation datum
gNavEllipName2EllipParam	Get Ellipsoid Parameters for Ellipsoid Name
gNavGeoc2Geoc1032	Geocentric to Geocentric, Helmert 7-parameter transformations
gNavGeoc2Geoc1032inv	Geocentric to Geocentric inverse, Helmert 7-parameter transformations
gNavGeoc2Geog	Geocentric to Geographic transformation
gNavGeog2Geoc	Geographic to Geocentric transformation
gNavGeog2ProjMercator	Geographic to Mercator transformation
gNavProjMercator2Geog	Mercator to Geographic transformation
gNavGeog2ProjUtm	Geographic to Transverse Mercator transformation
gNavProjUtm2Geog	Transverse Mercator to Geographic transformation
gNavGeog2ProjLambAz	Geographic to Lambert Azimuthal Equal Area transformation
gNavProjLambAz2Geog	Lambert Azimuthal Equal Area to Geographic transformation
Coordinates/time simple manipulations	
gNavArc2Len	Arc to meters
gNavReducingHeadingToProjUtm	Angle/scale between Geographic and Transverse Mercator
gNavReducingHeading	Angles/scale for Geographic-to-Rectangular
gNavCoordDerepeat	Remove and interpolate repeating time/position
gNavTrackMadeGood2D	2D-track smoothing and instantaneous direction calculation
gNavTrackDirection2D	The 2D-track-line direction calculation
gNavLLogRead	Read Survey log for survey lines with tab-delimiter.
Nod's calculation	
gNavTait	3D Tait-Bryan matrixes calculation
gNavTiatLever	LeverArm's coordinates calculates using scalar Head, Pitch, Roll
gNavTiatNLever	LeverArm's coordinates calculates using dimensions of Head, Pitch, Roll
gNavTiatLeverN	LeverArm's coordinates calculation using dimensions LeverX,LeverY,LeverZ
gNavTiatTest	Calculate "error ellipsoid" for USBL misalignment angles, using 2 ship's location with target's coordinates by USBL.
Layback calculation	
gNavLayback	Towing body coordinates calculation
gNavLaybackCurrent	Towing body coordinates calculation using sea current

1.1 Angles

The gNav/ge0mlib angles values are Degree. The function **gNavAng2Ang** can convert different angle dimensions to Degree (***Figure 1.1***).

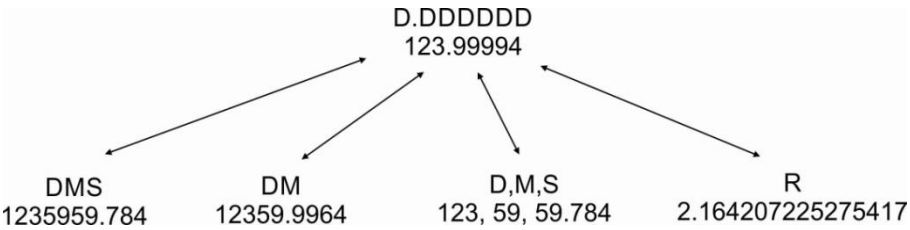

Figure 1.1 Angle formats

1.2 Time

The gNav/ge0mlib time axis values are vector with two rows: Day-from-Jan-1-0000 and Second-per-Day (DxSd). The function **gNavTime2Time** can convert different time formats to DxSd (***Figure 1.2***).

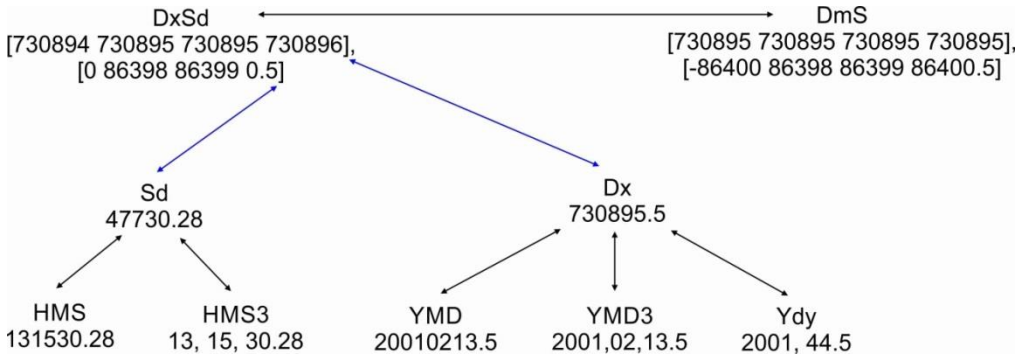

Figure 1.2 Time/date formats

The two types of time "sources" are realized; there are

-- Computer clock, as a "local" source with 0.001 second discrecity;

-- GPS clock (it is mean Date/Time received with navigation string from GPS), as a "global" source; it can be linked with GPS PPS data flow (1 second discrecity).

On the one hand, the computer clock time is used for interpolation inside GPS's 1-second intervals; on the other hand the GPS clock applies long-time-drift correction to computer's clock data. The one GPS data flow with GPS PPS can be used for several computer clocks' time reference (using single GPS-flow in several computers).

The link from any serial port messages to computer's clock is used in gLog function set (data logging software).

The fields' names for Computer Clock and GPS Clock are shown in ***Table 1.2***.

Table 1.2 Computer Clock and GPS Clock fields' names

Field name	Field Description
CompDay	Computer date number, corresponds to Jan-1-000; the year 0000.
CompTime	Computer's second per day. Vector; Row-content.
CompClockID	Computer's clock ID.
GpsDay	Gps date number, corresponds to Jan-1-000.
GpsTime	GPS's second per day.
TimeLocShift	Defined for all computers' difference between Local time and UTM in seconds.
CompTimeShift	Calculated mean difference between Computer time and Gps time in seconds; calculates for used data.
CompTimeDelta	Calculated difference between Computer time and Gps time, calculates for each point of used data decremented to CompTimeShift.

The function gNavDayCheck is used to check right conformity between Day and Time fields (for Day changes moment).

1.3 "Navigations" fields

The predefined coordinates and other "navigations" fields for some point are shown in *Table 1.3*.

Table 1.3 "Navigation" fields' names

Field name	Field Description
GpsLat	Latitude/Easting (WGS84 usually); DD.DDDDDD.
GpsLon	Longitude/Nording (WGS84 usually); DDD.DDDDDD.
GpsAltSea	Height by Gps, GpsAltSea data
GpsHgtGeoid	Height by Gps, GpsHgtGeoid data
GPSHeading	True heading by differential_GPS/Gyrocompass/MRU.
GpsE	Rectangular projection Easting.
GpsN	Rectangular projection Northing.
GpsH	Height created when Ellipsoid-to-Ellipsoid coordinates transformation take place; field used for coordinates transformation's stability
GpsKP	Point's ID, it can be measurement number, kilometer point's number or any other.
GpsZ	Points Z-coordinate for vertical geodetic datum (pipeline or towing equipment position).
Altitude	Point's Altitude (usually by altimeter for towing equipment)
Depth	Point's Depth (usually by depth sensor for towing equipment)
Heading	Heading in Rectangular projection.
WaterDepth	Water Depth for current points coordinates (can be "actual" or referenced to some system MLS, LAT, Baltic, etc).
XXXXRaw	Raw data (no smoothing, de-spiking, etc), where XXXX is any field name.

1.4 DTEN-fields

GpsKP, GpsDay, GpsTime, GpsE, GpsN, GpsH are named the DTEN-fields, it is define measurement time and coordinates in planar projection. The fields are used for "universalized"

time/coordinates fields for different structures, for example sgy-structure, xtf-structure, jsf-structure (the functions were used gSgyDTEN, gSgyDTENinv, gXtfDTEN, gJsfDTEN).

1.5 Track polyline structure

Track is on-plane polyline structure, which used as "interface structure" for track-plots, line-planning and other "planar objects" drawing and exports to AutoCad. The Track-polyline structure fields are shown in ***Table 1.4***. Usually Track-polyline is not containing GpsDay and GpsTime fields, but it can be include for compatibility with DTEN-fields functions.

Table 1.4 Track polyline structure fields' names

Field name	Field Description
PLName	Track-polyline name.
Type	Track-polyline types: Trackplot, LinePlan, PipeLineTrack, etc.
KeyLineDraw	String key for polyline drawing in MatLab figure (for example: '-r','xb').
GpsE	Polyline's points rectangular projection Easting.
GpsN	Polyline's points rectangular projection Northing.
GpsH	Optional field. Polyline's points height created when Ellipsoid-to-Ellipsoid coordinates transformation take place; field used for coordinates transformation's stability.
GpsZ	Optional field. Polyline's points Z-axis coordinate for vertical geodetic datum (pipeline or towing equipment position).
GpsKP	Optional field, polyline KP. This field has different meaning for polyline types: PipeLineTrack – pipeline KP; Trackplot – measurement/ping/shot number.
GpsDay	Optional field. Used as part of DTEN-fields.
GpsTime	Optional field. Used as part of DTEN-fields.
PipeD	Optional field. Pipeline diameter.
WaterDepth	Optional field. Water Depth for current points coordinates (can be "actual" or referenced to some system MLS, LAT, Baltic, etc).

1.6 Coordinates datum's and transformations

The two ellipsoids characteristics are used by ge0mlib functions (field Nav.EllipParam):

a – major ellipsoid axis;

et – first eccentricity of ellipsoid.

Other characteristics: b – minor ellipsoid axis; A – polar flattening; f – polar flattening's denominator; et2 – second eccentricity of ellipsoid.

$$A=\frac{1}{f}=\frac{a-b}{a};et=\frac{\sqrt{a^2-b^2}}{a}=\sqrt{2A-A^2};et2=\frac{\sqrt{a^2-b^2}}{b};$$

$$n=\frac{a-b}{a+b};m=\frac{a^2-b^2}{a^2+b^2};c=\frac{a^2}{b};$$

$$\frac{b}{a}=1-A=\sqrt{1-et^2}=\frac{1}{\sqrt{1+et2^2}}=\frac{1-n}{1+n}=\frac{a}{c}=\frac{et}{et2}$$

The three navigation datum are take place:

-- **Sensor's (input) navigation datum (NavS).** It is mean datum for raw GPS data; the datums are different for satellite systems (Navstar, Glonass, Bedow);

-- **Project's navigation datum (NavP).** Usually it is Cartesian coordinates for some ellipsoid. It is used for Layback, Nods and other calculations. The axis must create orthonormal basis (perpendicular and equal scale), the ellipsoid must be approximate to geoid.

-- **Output navigation datum (NavOutGeog, NavOutProj).** This datum used for output data-set creation and for mapping in accordance with client's requirements.

The general function for coordinate's transformation is **gNavCoord2Coord**. The coordinate re-calculation scheme is showed in ***Figure 1.3***. There are follow steps: transformed_rectangular-to-rectangular, rectangular-to-geographic, geographic-to-geocentric, geocentric_ellipsoide_1-to-geocentric_ellipsoide_2, geocentric-to-geographic, geographic-to-rectangular, rectangular-to-transformed_rectangular. Each step is defined using function and parameters. The sensor's datum fields are showed in ***Table 1.5***; project's datum fields – in ***Table 1.6***.

If not exist filed NavS.EllipTransParam, than means "no ellipsoid transformation" take place (steps geographic-to-geocentric, geocentric_ellipsoide_1-to-geocentric_ellipsoide_2, geocentric-to-geographic can be ignored).

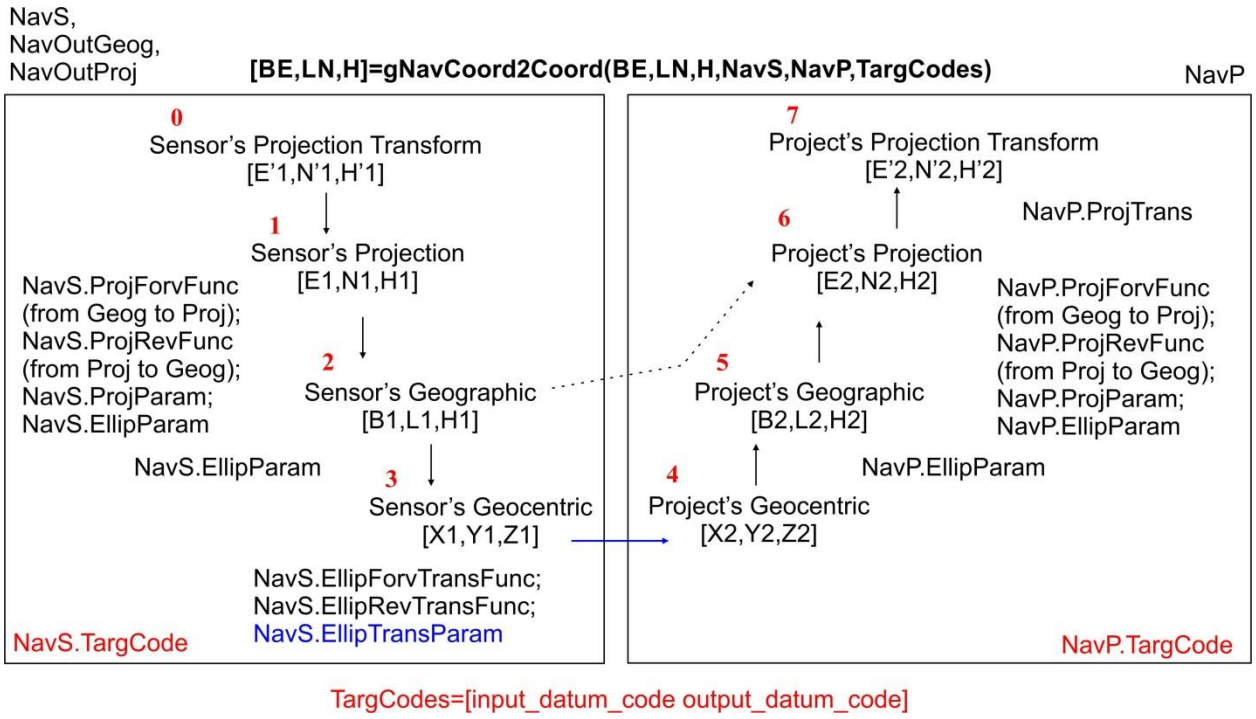

Figure 1.3 Coordinates re-calculation from ellipsoid_1 to ellipsoid_2 (red numbers are values for Nav.TargCode field)

Table 1.5 Sensor's navigation datum structure fields' names

Field name	Field Description
NavS.EllipParam	[a,et] – Defined semi-major axis (length of the semi-major axis of the

	ellipsoid, the radius of the equator) and eccentricity.
NavS.ProjParam	Defined projection parameters; for UTM is [Latitude of natural origin (B0); Longitude of natural origin (L0); Scale factor at natural origin; False easting; False northing].
NavS.ProjForvFunc	Defined projection function name was used for geographic-to-rectangular transformation.
NavS.ProjRevFunc	Defined projection function name was used for rectangular-to-geographic transformation.
NavS.EllipTransParam	The transformation to another Ellipse parameters (if necessary). If not exist, than means "no ellipsoid transformation".
NavS.EllipForvTransFunc	The transformation function's name from Sensor's Ellipse to Project's.
NavS.EllipRevTransFunc	The transformation function's name from Project's Ellipse to Sensor's.
NavS.TargCode	Sensor's datum code (see ***Figure 1.3***).

Table 1.6 Project's navigation datum structure fields' names

Field name	Field Description
NavP.EllipParam	[a,et] – Defined semi-major axis (length of the semi-major axis of the ellipsoid, the radius of the equator) and eccentricity.
NavP.ProjParam	Defined projection parameters; for UTM is [Latitude of natural origin (B0); Longitude of natural origin (L0); Scale factor at natural origin; False easting; False northing].
NavP.ProjForvFunc	Defined projection function name was used for geographic-to-rectangular transformation.
NavP.ProjRevFunc	Defined projection function name was used for rectangular-to-geographic transformation.
NavP.ProjTrans	Projection additional transformation (shift, rotation, rescaling)
NavP.TargCode	Project's datum code (see ***Figure 1.3***).

There are no geocentric-to-geocentric transformation parameters for NavP datum. It is mean, those sensors' coordinates from different ellipsoids convert to single project's ellipsoid (only sensor's datum includes conversion parameters and function to project's ellipsoid).

NavS.TargCode and NavP.TargCode linked to original coordinate's datums. The NavS.TargCode define data type from sensor (usually is 2 – geographic coordinates); NavP.TargCode define data type using in project (usually is 6 – rectangular coordinates).

The transformed rectangular projection is shifted, rotated and rescaled rectangular projection, the **gNavGeoc2GeogProjTr** function is used. The geographic-to-geocentric and geocentric-to-geographic always used functions **gNavGeoc2Geog** and **gNavGeog2Geoc** defined for 2-axise ellipsoid. The functions used same parameters Nav.ProjParam, Nav.EllipTransParam for forward and reverse transformations. The geocentric-to-geocentric functions are: **gNavGeoc2Geoc1032**, **gNavGeoc2Geoc1032inv**. The rectangular-to-geographic and geographic-to-rectangular functions are: **gNavGeog2ProjLambAz**, **gNavGeog2ProjUtm**, **gNavProjLambAz2Geog**, **gNavProjUtm2Geog**.

The fields values examples shown in ***Table 1.7*** and ***Table 1.8***.

Table 1.7 Coordinates re-calculation from geographic to UTM without ellipsoid changes

Sensor's Field name	Sensor's Field Example
NavS.TimeLocShift	10
NavS.TargCode	2
Project's Field name	Project's Field Example
NavP.EllipParam	[6378137 0.081819190842]
NavP.ProjParam	[0 142 0.9996 500000 0]
NavP.ProjForvFunc	'gNavGeog2ProjUtm'
NavP.ProjRevFunc	'gNavProjUtm2Geog'
NavP.TargCode	6

Table 1.8 Coordinates re-calculation from geographic to UTM with ellipsoid changes

Sensor's Field name	Sensor's Field Example
NavS.EllipParam	[6378137 0.0818191908425]
NavS.EllipTransParam	[-43.8 108.8 119.5 1.4 -0.76 0.73 0.54e-6]
NavS.EllipForvTransFunc	'gNavGeoc2Geoc1032'
NavS.EllipRevTransFunc	'gNavGeoc2Geoc1032inv'
NavS.TargCode	2
Project's Field name	Project's Field Example
NavP.EllipParam	[6378245 0.081813]
NavP.ProjParam	[0 51 1 500000 0]
NavP.ProjForvFunc	'gNavGeog2ProjUtm'
NavP.ProjRevFunc	'gNavProjUtm2Geog'
NavP.ProjRevFunc	[10000 30000 25 1 1.1]
NavP.TargCode	6

Any geometrical calculations (nod's, layback) made in rectangular coordinates; TrueHeading or MagneticHeading must be reduced to "rectangular" Heading.

1.7 Layback Calculation

The layback calculated using modified ***"Dragging" algorithm***.

The follow is written a "basic Dragging algorithm" description by Geometrics (***Figure 1.4***):

"We assume that we know position of the ship (A) and the magnetometer (N) at time t0 and we know the cable length. At time t 1 we know the position of the ship (B) but do not know the position of the Fish (M.) To find it we draw a straight line NB between old magnetometer position N and new ship position B. Then we count the cable length from B towards N. A new point M is the estimated position of the magnetometer sensor at time t1. It can happen that distance BM is greater then BN (if ship does a sharp turn). In this case magnetometer position not changed (literally it would sink). The method explained above assumes that the magnetometer sensor position at time t0 is known but it is unknown at the start of the line. Therefore to start calculation process we need to find somehow initial magnetometer position. Different techniques might be employed but one of the easiest is to use an initial part of the recorded path to find direction of motion. We can approximate these positions with a straight line using least-squares

method and count cable length back along this line. This gives a reasonable estimation for the initial magnetometer position. After a short time the influence of the initial position becomes negligible."

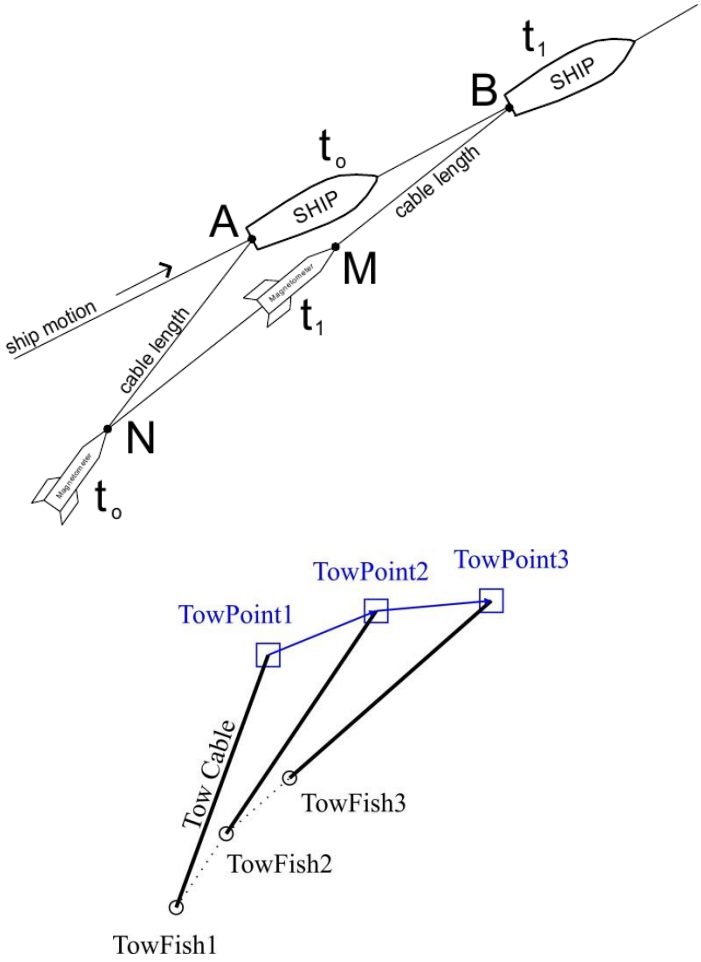

Figure 1.4 Layback calculations – "Dragging" algorithm

There are follow modifications to the "basic Dragging algorithm" for gNavLayback function:

-- The direction from second to first point and cable length are used to calculate initial towfish position.

-- The not "cable length" is used, but cable length projection in horizontal plane – Layback. The Layback is calculated by SHOM Zero Surface method (***Figure 1.5***) with catenary factor. The catenary factor is depend from cable length (and another cable's and towing body characteristics) and tow-point speed relative water. The example of tow-cable shapes for a number of speed and length are shown in ***Figure 1.6***. The towfish depth is estimated using depth sensor (pressure sensor).

-- The simple estimation of water-current influence is presented: each point of towfish track (was calculated using Dragging-algorithm-with-depth) can be rotated in the angle A, from survey line direction. The angle A is estimated using fish's compass data or vessel's heading (mean for current survey line).

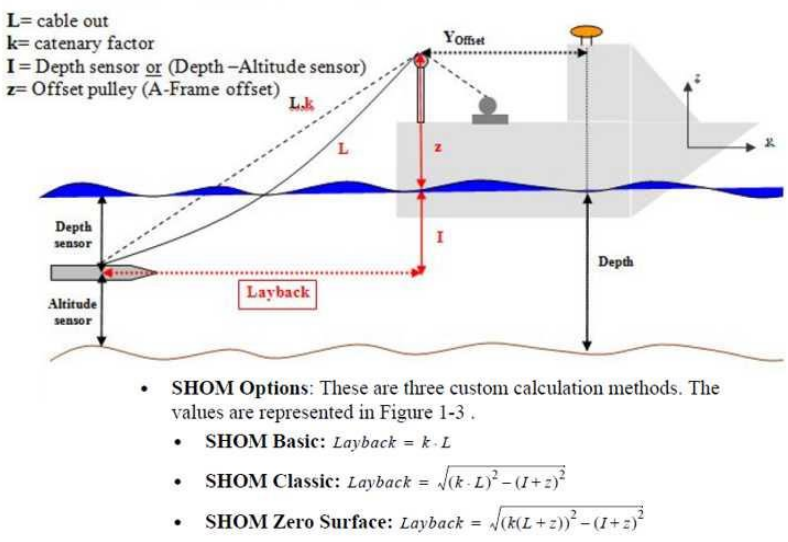

Figure 1.5 SHOM Layback calculation [HYPACK® Driver Interface...]

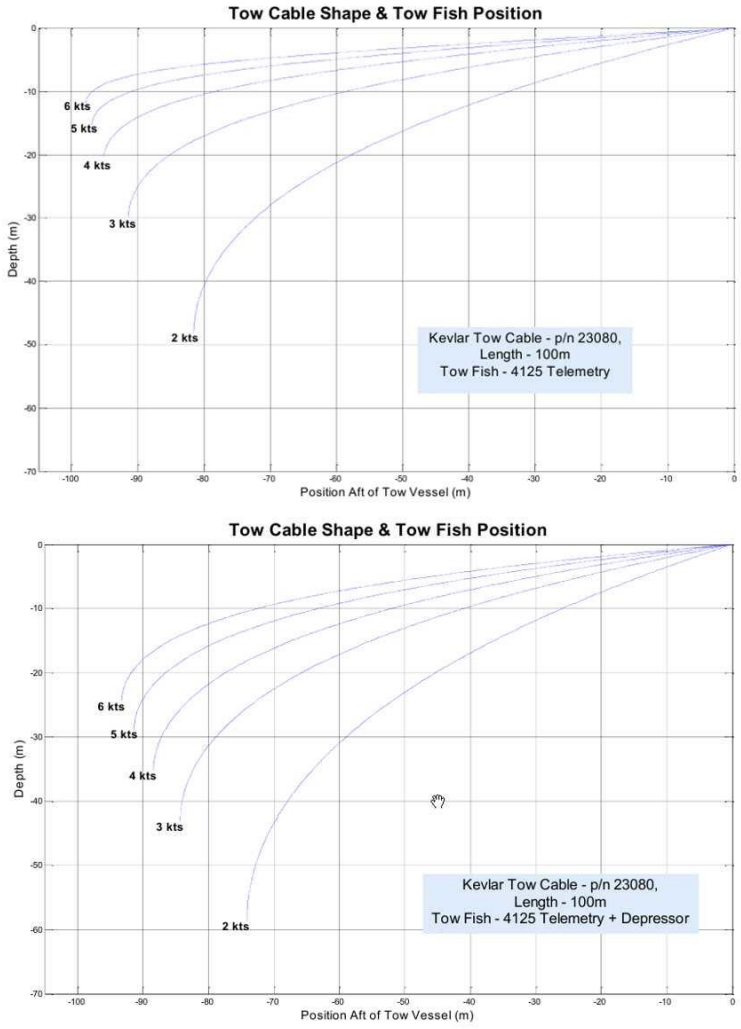

Figure 1.6 ET3200 SSS fish Depth-Towing_speed-Cable_length dependance [4125 Side Scan Sonar System....]

2. Time/angle conversions functions

2.1 Convert angle-to-angle

function varargout=gNavAng2Ang(key,varargin)

Convert angle format varargin{1} to angle format varargout{1} accordance key.

Parameters:

varargin – converted angle;

varargout – conversion result;

key – formats key: DMS2D, D2DMS, DMS32D, D2DMS3, DM2D, D2DM, D2R, R2D.

DMS is DDDMMSS.SSS; DMS3 is D,M,S; DM is DDDMM.MMM; D is DD.DDDDD; R is RR.RRRR (Degree/Minute/Second/Radian); see ***Figure 1.1***.

Example.

```
Ang2=gNavAng2Ang('DMS2D',1001010.11);
```

2.2 Convert time-to-time

function varargout=gNavTime2Time(key,varargin)

Convert date/time format varargin{1} to date/time format varargout{1} accordance key.

Parameters:

varargin – converted date/time;

varargout – date/time conversion result;

key – formats key: HMS2Sd, Sd2HMS, HMS32Sd, Sd2HMS3, YMD2Dx, DMY2Dx, Dx2YMD, YMD32Dx, Dx2YMD3, YDy2Dx, Dx2YDy, DxSd2DmS, DmS2DxSd, DxSdDm2DmS, DxHMS32DmT, DmT2DxHMS3; see ***Figure 1.2***.

varargin/varargout: HMS is HHMMSS.SSS; HMS3 is HH,MM,SS.SSSSSS; YMD is YYYYMMDD; YMD3 is YYYY,MM,DD.DDDD; Sd is second per day; Dx is serial date number of 1 corresponds to Jan-1-0000; Dy is Julian day; Dm is "point of origin" for Dx; S is second; Y is year.

Example.

```
>> Z=gNavTime2Time('YMD2Dx',20010210);[y,m,d]=gNavTime2Time('Dx2YMD3',Z);
```

2.3 Check Day

function [Dx,Sd]=gNavDayCheck(Dx,Sd)

Check than date was changed in 00:00:00; correct bad date.

Parameters:

Dx – raw or scalar, serial date number of 1 corresponds to Jan-1-0000 or another day number (Julian, "point of origin", etc).

Sd – raw, input second per day;

[Dx,Sd] – raws, output serial date number and second per day.

Example.

```
>> Sd=gNavTime2Time('HMS32Sd',Head.HourOfDay,Head.MinuteOfHour,Head.SecondOfMinute);
>> [Head.DayOfYear,Sd]=gNavDayCheck(Head.DayOfYear,Sd);
>> [Head.HourOfDay,Head.MinuteOfHour,Head.SecondOfMinute]=gNavTime2Time('Sd2HMS3',Sd);
```

2.4 Convert distance and speed

function varargout=gNavDV2DV(key,varargin)

Convert distance/speed format varargin{1} to distance/speed format varargout{1} accordance key.

Parameters:

varargin – converted distance/speed;

varargout – conversion distance/speed;

key – formats key: inch2m, m2inch, mile2m, m2mile, knot2ms, ms2knot, kmh2ms, ms2kmh (mile is 1852m).

Example.

```
>> D=gNavDV2DV('mile2m',10.1);
```

3. Coordinates datum's and transformations functions

3.1 Main coordinates transformation

function [BE,LN,H]=gNavCoord2Coord(BE,LN,H,NavS,NavP,TargCodes)

Convert coordinates between SensorNavigation datum to ProjectNavigation datum (TargCodes can be used; see ***Figure 1.3***).

Parameters:

BE – row, Latitude on Easting for conversion;

LN – row, Longitude or Northing for conversion;

H – row, Height for conversion;

NavS – navigation datum for Sensor, fields: EllipParam, ProjParam, ProjForvFunc, ProjRevFunc, EllipTransParam, EllipForvTransFunc, EllipRevTransFunc, TargCode.
if ~isfield(NavS.EllipTransParam), then transformation Sensor's_Ellipsoid-to-Project's ellipsoid not calculate (fields EllipTransParam, EllipForvTransFunc, EllipRevTransFunc not used).

NavP – navigation datum for Project, fields: EllipParam, ProjParam, ProjForvFunc, ProjRevFunc, TargCode.

TargCodes=[input_datum_code output_datum_code_2] – forced datum codes. There are: 1) sensor rectangular; 2) sensor geographic; 3) sensor geocentric; 4) project geocentric; 5) project geographic; 6) project cartesian.
If isempty(TargCodes), than create TargCodes=[NavS.TargCode NavP.TargCode].

[BE,LN,H] – output data Lat/E, Lon/N, H; rows.

Example 1 – Single ellipsoid; calculate from 2 (sensor geographic) to 6 (project rectangular):

```
>> NavS=struct('TargCode',2);
>> NavP=struct('EllipParam',[6378137 0.081819190842], 'ProjParam',[0 142 0.9996 500000 0],
'ProjForvFunc','gNavGeog2ProjUtm', 'ProjRevFunc','gNavProjUtm2Geog', 'TargCode',6);
>> [E,N,H]=gNavCoord2Coord(B,L,0,NavS,NavP,[]);
```

Example 2 – Two Ellipsoids; calculate from 2 (sensor geographic) to 6 (project rectangular):

```
>> NavS=struct('EllipParam',[6378137 0.0818191908425], 'EllipTransParam',[-43.8 108.8 119.5 1.4 -
0.76 0.73 0.54e-6], 'EllipForvTransFunc','gNavGeoc2Geoc1032',
'EllipRevTransFunc','gNavGeoc2Geoc1032inv', 'TargCode',2);
>> NavP=struct('EllipParam',[6378245 0.081813], 'ProjParam',[0 51 1 500000 0],
'ProjForvFunc','gNavGeog2ProjUtm', 'ProjRevFunc','gNavProjUtm2Geog', 'TargCode',6);
>> [E,N,H]=gNavCoord2Coord(B,L,zeros(size(B)),NavS,NavP,[]);
```

Example 3 – Single ellipsoid; calculate from 2 (sensor geographic) to 1 (sensor rectangular):

```
>> NavS=struct('EllipParam',[6378137 0.081819190842], 'ProjParam',[0 142 0.9996 500000 0],
'ProjForvFunc','gNavGeog2ProjUtm', 'ProjRevFunc','gNavProjUtm2Geog', 'TargCode',2);
>> NavP=struct('TargCode',6);
>> [E,N,H]=gNavCoord2Coord(B,L,0,NavS,NavP,[2 1]);
```

3.2 Ellipsoid Name to Ellipsoid Parameters

function EllipParam=gNavEllipName2EllipParam(EllipName)

Get Ellipsoid Parameters [semi_major_axis first_eccentricity] for Ellipsoid Name.

Parameters:

EllipName – ellipsoid names: sphere, everest, bessel, airy, clarke66, clarke80, international, krasovsky46, wgs60, iau65, wgs66, iau68, wgs72, grs80, wgs84, pz90;

EllipParam – ellipsoid parameters [Semi_major_axis_a First_eccentricity_et];

Example:

```
>> EllipParam=gNavEllipName2EllipParam('wgs84');
```

3.3 Geocentric to geocentric, Helmert 7-parameter transformations

function [X2,Y2,Z2]=gNavGeoc2Geoc1032(X1,Y1,Z1,conv_dat)

Convert coordinates from first geocentric reference system to second (coordinates conversion direction strict inverse). Used Coordinate Frame Rotation (geocentric domain)/EPSG Dataset coordinate operation method code 1032, where wxyz has changed sign relative to Helmert 7-parameter transformations (Position Vector transformation /EPSG Dataset coordinate operation method code 1033) [Geomatics Guidance Note Number 7...].

Parameters:

X1,Y1,Z1 – input/first geocentric coordinates rows, meters;

conv_dat – conversion parameters [dx dy dz wx wy wz m] or conversion key;

X2,Y2,Z2 – output/second geocentric coordinates rows, meters;

Example:

```
>> GpsEllipseParam=[6378137 0.081819190842]; pr=[23.57 -140.95 -79.8 0.000 -0.35 -0.79 -0.22e-6];
>> [X,Y,Z]=gNavGeog2Geoc(51:.1:52,142:.1:143,100,GpsEllipseParam);
>> [X2,Y2,Z2]=gNavGeoc2Geoc1032inv(X,Y,Z,pr);[X1,Y1,Z1]=gNavGeoc2Geoc1032(X2,Y2,Z2,pr);
>> dXYZ_inv=[X;Y;Z]-[X1;Y1;Z1];
>> [X2,Y2,Z2]=gNavGeoc2Geoc1032(X,Y,Z,pr);[X1,Y1,Z1]=gNavGeoc2Geoc1032(X2,Y2,Z2,pr);
>> dXYZ=[X;Y;Z]-[X1;Y1;Z1];
```

3.4 Geocentric to geocentric inverse, Helmert 7-parameter transformations

function [X2,Y2,Z2]=gNavGeoc2Geoc1032inv(X1,Y1,Z1,conv_dat)

Convert coordinates from first geocentric reference system to second (coordinates conversion direction strict inverse). Used Coordinate Frame Rotation (geocentric domain)/EPSG Dataset coordinate operation method code 1032, where wxyz has changed sign relative to Helmert 7-parameter transformations (Position Vector transformation /EPSG Dataset coordinate operation method code 1033).

Parameters:

X1,Y1,Z1 – input/first geocentric coordinates rows, meters;

conv_dat – conversion parameters [dx dy dz wx wy wz m] or conversion key;

X2,Y2,Z2 – output/second geocentric coordinates rows, meters;

Example:

```
>> GpsEllipseParam=[6378137 0.081819190842]; pr=[23.57 -140.95 -79.8 0.000 -0.35 -0.79 -0.22e-6];
>> [X,Y,Z]=gNavGeog2Geoc(51:.1:52,142:.1:143,100,GpsEllipseParam);
>> [X2,Y2,Z2]=gNavGeoc2Geoc1032inv(X,Y,Z,pr);[X1,Y1,Z1]=gNavGeoc2Geoc1032(X2,Y2,Z2,pr);
>> dXYZ_inv=[X;Y;Z]-[X1;Y1;Z1];
>> [X2,Y2,Z2]=gNavGeoc2Geoc1032(X,Y,Z,pr);[X1,Y1,Z1]=gNavGeoc2Geoc1032(X2,Y2,Z2,pr);
>> dXYZ=[X;Y;Z]-[X1;Y1;Z1];
```

3.5 Geocentric to Geographic

function [B,L,H]=gNavGeoc2Geog(X,Y,Z,EllipParam)

Convert geocentric coordinates to geographic coordinates. Transformation based on the ГОСТ Р 51794-2001.

Parameters:

B,L,H – rows, geographic Latitude, Longitude (degrees) and Height of geoid above ellipsoid (meters);

X,Y,Z – rows, geocentric coordinates (meters);

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et];

Example.

```
>> EllipParam=[6378137 0.081819190842];
>> [X,Y,Z]=gNavGeog2Geoc(51:.1:52,142:.1:143,100,EllipParam);
>> [B,L,H]=gNavGeoc2Geog(X,Y,Z,EllipParam);
```

3.6 Geographic to Geocentric

function XYZ=gNavGeog2Geoc(B,L,H,EllipParam)

Convert geographic coordinates to geocentric coordinates. Transformation based on the ГОСТ Р 51794-2001.

Parameters:

B,L,H – rows, geographic Latitude, Longitude (degrees) and Height of geoid (mean sea level) above ellipsoid (meters);

X,Y,Z – rows, geocentric coordinates (meters);

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et] or ellipsoid name;

Transformation based on the ГОСТ Р 51794-2001

Example.

```
>> EllipParam=[6378137 0.081819190842];
>> [X,Y,Z]=gNavGeog2Geoc(51:.1:52,142:.1:143,100,EllipParam);
>> [B,L,H]=gNavGeoc2Geog(X,Y,Z,EllipParam);
```

3.7 Geographic to Mercator

function [E,N]=gNavGeog2ProjMercator(B,L,EllipParam,ProjParam)

Convert Geographic coordinates to Mercator projection. EPSG dataset coordinate operation method codes 9804,9805,1044,1026. Formulas source: "Geomatics Guidance Note 7, part 2 Coordinate Conversions&Transformations including Formulas", Report 373-7-2, October 2017.

EPSG recognizes three variants of the Mercator projection, the methods called A (1SP; EPSG 9804), B (EPSG 9805), C (2SP; EPSG 1044), Spherical (EPSG 1026) and Pseudo-Mercator (EPSG 1024; "Web Mercator"):

-- EPSG 9804. The projection is defined with the equator as the single standard parallel, with scale factor on the equator also defined. False grid coordinates are applied at the natural origin of the projection, the intersection of the equator and the longitude of origin.

-- EPSG 9805. Defined through the latitude of two parallels equidistant either side of the equator upon which the grid scale is true. False grid coordinates are applied at the natural origin of the projection, the intersection of the equator and the longitude of origin.

-- EPSG 1044. Defined through the latitude of two parallels equidistant either side of the equator upon which the grid scale is true, as in variant (B). However in variant C false grid coordinates are applied at a point other than the natural origin of the projection, called the false origin.

-- EPSG 1026. Mercator (Spherical). If latitude 90deg, N is infinite. The above formula for N will fail near to the pole, and should not be used poleward of 88deg.

-- EPSG 1024. Pseudo-Mercator ("Web Mercator"). This method is utilised by some popular web mapping and visualisation applications. Strictly speaking the inclusion of 'Mercator' in the method name is misleading: it is not a Mercator projection, it is a different map projection and uses its own distinct formula: it is a separate method. Unlike either the spherical or ellipsoidal Mercator projection methods, this method is not conformal: scale factor varies as a function of azimuth, which creates angular distortion. Despite angular distortion there is no convergence in the meridian, so the graticule has a similar appearance to the graticule of a Mercator projection, but the graticules of the two projections do not overlay.

Parameters:

B,L – rows, Latitude, Longitude in degrees;

EllipParam – ellipsoid parameters [Semi_major_axis_a Eccentricity_et];

>>> ProjParam- EPSG 9804 parameters [Longitude_of_natural_origin(L0) Scale_at_natural_origin(K0) False_easting(FE) False_northing(FN) 1]; Latitude_of_natural_origin(B0)==0 by default.

>>> ProjParam- EPSG 9805 parameters [Latitude_of_standard_parallel(B1) Longitude_of_natural_origin(L0) False_easting(FE) False_northing(FN) 2];

>>> ProjParam- EPSG 1044 parameters [Latitude_of_standard_parallel(B1) Latitude_of_false_origin(Bf) Longitude_of_natural_origin(Lf) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 3];

>>> ProjParam- EPSG 1026 parameters [Longitude_of_natural_origin(L0) False_easting(FE) False_northing(FN) 4].

>>> ProjParam- EPSG 1024 not realized.

E,N – rows, Easting, Northing in meters;

Examples:

```
>> EllipParam=[6377397.155 0.081696831];ProjParam=[110 0.997 3900000 900000
1];[GpsE,GpsN]=gNavGeog2ProjMercator(-3,120,EllipParam,ProjParam);
>> EllipParam=[6378245.0 0.08181333];ProjParam=[42 51 0 0
2];[GpsE,GpsN]=gNavGeog2ProjMercator(53,53,EllipParam,ProjParam);
>> EllipParam=[6378245.0 0.08181333];ProjParam=[42 42 51 0 0
3];[GpsE,GpsN]=gNavGeog2ProjMercator(53,53,EllipParam,ProjParam);
>> EllipParam=[6371007.0 0];ProjParam=[0 0 0
4];[GpsE,GpsN]=gNavGeog2ProjMercator(24.381786944444446,-
100.3333333333333,EllipParam,ProjParam);
```

3.8 Mercator to Geographic

function [B,L]=gNavProjMercator2Geog(E,N,EllipParam,ProjParam)

Convert Mercator projection coordinates to Geographic coordinates. EPSG dataset coordinate operation method codes 9804,9805,1044,1026. Formulas source: "Geomatics Guidance Note 7, part 2 Coordinate Conversions&Transformations including Formulas", Report 373-7-2, October 2017.

EPSG recognises three variants of the Mercator projection, the methods called A (1SP; EPSG 9804), B (EPSG 9805), C (2SP; EPSG 1044), Spherical (EPSG 1026) and Pseudo-Mercator (EPSG 1024; "Web Mercator"):

-- EPSG 9804. The projection is defined with the equator as the single standard parallel, with scale factor on the equator also defined. False grid coordinates are applied at the natural origin of the projection, the intersection of the equator and the longitude of origin.

-- EPSG 9805. Defined through the latitude of two parallels equidistant either side of the equator upon which the grid scale is true. False grid coordinates are applied at the natural origin of the projection, the intersection of the equator and the longitude of origin.

-- EPSG 1044. Defined through the latitude of two parallels equidistant either side of the equator upon which the grid scale is true, as in variant (B). However in variant C false grid coordinates are applied at a point other than the natural origin of the projection, called the false origin.

-- EPSG 1026. Mercator (Spherical). If latitude 90deg, N is infinite. The above formula for N will fail near to the pole, and should not be used poleward of 88deg.

-- EPSG 1024. Pseudo-Mercator ("Web Mercator"). This method is utilised by some popular web mapping and visualisation applications. Strictly speaking the inclusion of 'Mercator' in the method name is misleading: it is not a Mercator projection, it is a different map projection and uses its own distinct formula: it is a separate method. Unlike either the spherical or ellipsoidal Mercator projection methods, this method is not conformal: scale factor varies as a function of azimuth, which creates angular distortion. Despite angular distortion there is no convergence in the meridian, so the graticule

has a similar appearance to the graticule of a Mercator projection, but the graticules of the two projections do not overlay.

Parameters:

E,N – rows, Easting, Northing in meters;

EllipParam – ellipsoid parameters [Semi_major_axis_a Eccentricity_et];

>>> ProjParam- EPSG 9804 parameters [Longitude_of_natural_origin(L0) Scale_at_natural_origin(K0) False_easting(FE) False_northing(FN) 1]; Latitude_of_natural_origin(B0)==0 by default.

>>> ProjParam- EPSG 9805 parameters [Latitude_of_standard_parallel(B1) Longitude_of_natural_origin(L0) False_easting(FE) False_northing(FN) 2];

>>> ProjParam- EPSG 1044 parameters [Latitude_of_standard_parallel(B1) Latitude_of_false_origin(Bf) Longitude_of_natural_origin(Lf) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 3];

>>> ProjParam- EPSG 1026 parameters [Longitude_of_natural_origin(L0) False_easting(FE) False_northing(FN) 4].

>>> ProjParam- EPSG 1024 not realized.

B,L – rows, Latitude, Longitude in degrees;

Examples:

```
>> EllipParam=[6377397.155 0.081696831];ProjParam=[110 0.997 3900000 900000 1];[GpsE,GpsN]=gNavProjMercator2Geog(5009726.58,569150.82,EllipParam,ProjParam);
>> EllipParam=[6378245.0 0.08181333];ProjParam=[42 51 0 0 2];[GpsE,GpsN]=gNavProjMercator2Geog(165704.29,5171848.07,EllipParam,ProjParam);
>> EllipParam=[6378245.0 0.08181333];ProjParam=[42 42 51 0 0 3];[GpsE,GpsN]=gNavProjMercator2Geog(165704.29,1351950.22,EllipParam,ProjParam);
>> EllipParam=[6371007.0 0];ProjParam=[0 0 0 4];[GpsE,GpsN]=gNavProjMercator2Geog(-11156569.90,2796869.94,EllipParam,ProjParam);
```

3.9 Geographic to Transverse Mercator

function [E,N]=gNavGeog2ProjUtm(B,L,EllipParam,ProjParam)

Convert Geographic coordinates to Transverse Mercator. EPSG dataset coordinate operation method code 9807: formulas are based on those of Kruger and published in Finland as Recommendations for Public Administration (JHS) 154 (referred as ‘JHS formulas’).

Parameters:

B,L – rows, Latitude, Longitude in degrees;

E,N – rows, Easting, Northing in meters;

ProjParam – TM parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin False_easting False_northing];

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et].

Example.

```
>> EllipParam=[6378137 0.081819190842];ProjParam=[0 142 0.9996 500000 0];
```

```
>> [GpsE,GpsN]=gNavGeog2ProjUtm(51:.1:52,142:.1:143,GpsEllipParam,GpsProjParam);
>> [GpsLat,GpsLon]=gNavGeog2ProjUtm(GpsE,GpsN,GpsEllipParam,GpsProjParam);
```

3.10 Transverse Mercator to Geographic

function [B,L]=gNavProjUtm2Geog(E,N,EllipParam,ProjParam)

Convert Transverse Mercator to Geographic coordinates. EPSG dataset coordinate operation method code 9807: formulas are based on those of Kruger and published in Finland as Recommendations for Public Administration (JHS) 154 (referred as 'JHS formulas').

Parameters:

E,N – rows, Easting, Northing in meters;

B,L – rows, Latitude, Longitude in degrees;

ProjParam – TM parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin False_easting False_northing];

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et].

Example.

```
>> GpsEllipParam=[6378137 0.081819190842];GpsProjParam=[0 142 0.9996 500000 0];
>> [GpsE,GpsN]=gNavGeog2ProjUtm(51:.1:52,142:.1:143,GpsEllipParam,GpsProjParam);
>> [GpsLat,GpsLon]=gNavProjUtm2Geog(GpsE,GpsN,GpsEllipParam,GpsProjParam);
```

3.11 Geographic to Lambert Azimuthal Equal Area

function [E,N]=gNavGeog2ProjLambAz(B,L,EllipParam,ProjParam)

Convert Geographic coordinates to Lambert Azimuthal Equal Area projection. EPSG dataset coordinate operation method code 9820.

Parameters:

B,L – raws, Latitude, Longitude in degrees;

ProjParam – Lambert Azimuthal Equal Area parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) False_easting False_northing];

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et];

E,N – rows, Easting, Northing in meters;

[E N][B L_west][B0 L0_west FE FN] >> [-E N][B -L_east][B0 -L0_east -FE FN].

Example.

```
>> EllipParam=[6378137 0.081819190842];ProjParam=[52 10 4321000 3210000];
>> [GpsE,GpsN]=gNavGeog2ProjLambAz(49:.1:51,4:.1:6,EllipParam,ProjParam);
>> [GpsLat,GpsLon]=gNavProjLambAz2Geog(GpsE,GpsN,EllipParam,ProjParam);
```

3.12 Lambert Azimuthal Equal Area to Geographic

function [B,L]=gNavProjLambAz2Geog(E,N,EllipParam,ProjParam)

Convert Lambert Azimuthal Equal Area projection coordinates to Geographic. EPSG dataset coordinate operation method code 9820.

Parameters:

E,N – rows, Easting, Northing in meters;

ProjParam – Lambert Azimuthal Equal Area parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) False_easting False_northing];

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et];

B,L – rows, Latitude, Longitude in degrees;

[E N][B L_west][B0 L0_west FE FN] >> [-E N][B -L_east][B0 -L0_east -FE FN]

Example.

```
>> EllipParam=[6378137 0.081819190842];ProjParam=[52 10 4321000  3210000];
>> [GpsE,GpsN]=gNavGeog2ProjLambAz(49:.1:51,4:.1:6,EllipParam,ProjParam);
>> [GpsLat,GpsLon]=gNavProjLambAz2Geog(GpsE,GpsN,EllipParam,ProjParam);
```

3.13 Geographic to Lambert Conic Conformal

function [E,N]=gNavGeog2ProjLambertConicConf(B,L,EllipParam,ProjParam)

Convert Geographic coordinates to Lambert Conic Conformal projection. EPSG dataset coordinate operation method codes 9801,9802,9826,9803,1051.

Formulas source: "Geomatics Guidance Note 7, part 2 Coordinate Conversions&Transformations including Formulas", Report 373-7-2, October 2017.

EPSG 9801; EPSG 9802. EPSG recognizes two variants of the Lambert Conic Conformal, the methods called 1SP (EPSG 9810) and 2SP (EPSG 9802).

EPSG 9826. 1SP West Orientated. In older mapping of Denmark and Greenland the Lambert Conic Conformal (1SP) is used with axes positive north and >>west<<.

EPSG 9803. 2SP Belgium. In 1972, in order to retain approximately the same grid coordinates after a change of geodetic datum, a modified form of the two standard parallel case was introduced in Belgium. In 2000 this modification was replaced through use of the regular Lambert Conic Conformal (2SP) method with appropriately modified parameter values.

EPSG 1051. 2SP Michigan. In 1964, the US state of Michigan redefined its State Plane CS27 zones, changing them from being Transverse Mercator zones orientated north-south to being Lambert Conic Conformal zones orientated east-west to better reflect the geography of the state.

EPSG 9817. Lambert Conic Near-Conformal. The Lambert Conformal Conic with one standard parallel formulas, as published by the Army Map Service, are still in use in several countries. However in some countries the expansion formulas were truncated to the third order and the map projection is not fully conformal.

Parameters:

B,L – rows, Latitude, Longitude in degrees;

>>> ProjParam – 1SP parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin(K0) False_easting(FE) False_northing(FN) 1];

>>> ProjParam – 2SP parameters [Latitude_of_false_origin(Bf) Longitude_of_false_origin(Lf) Latitude_of_1st_standard_parallel(B1) Latitude_of_2nd_standard_parallel(B2) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 2];

>>> ProjParam – 1SP West Orientated parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin(K0) False_easting(FE) False_northing(FN) 3];

>>> ProjParam – 2SP Belgium parameters [Latitude_of_false_origin(Bf) Longitude_of_false_origin(Lf) Latitude_of_1st_standard_parallel(B1) Latitude_of_2nd_standard_parallel(B2) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 4];

>>> ProjParam – 2SP Michigan parameters [Latitude_of_false_origin(Bf) Longitude_of_false_origin(Lf) Latitude_of_1st_standard_parallel(B1) Latitude_of_2nd_standard_parallel(B2) Easting_at_false_origin(EF) Northing_at_false_origin(NF) Ellipsoid_scaling_factor(K) 5];

EllipParam – ellipsoid parameters [Semi_major_axis_a Eccentricity_et];

E,N – rows, Easting, Northing in meters;

Examples:

```
>> EllipParam=[6378206.400 0.08227185];ProjParam=[18 -77 1 250000 150000 1];[GpsE,GpsN]=gNavGeog2ProjLambertConicConf(17.9321666666667,-76.9436833333333,EllipParam,ProjParam);
>> EllipParam=[6378206.400 0.08227185];ProjParam=[27.8333333333333 -99 28.3833333333333 30.2833333333333 2000000*0.304800609601219 0 2];[GpsE,GpsN]=gNavGeog2ProjLambertConicConf(28.5,-96,EllipParam,ProjParam);
>> EllipParam=[6378388 0.08199189];ProjParam=[90 4.35693972222222 49.833333333333333 51.166666666666667 150000.01 5400088.44 4];[GpsE,GpsN]=gNavGeog2ProjLambertConicConf(50.6795725,5.80737027777778,EllipParam,ProjParam);
>> EllipParam=[6378206.400 0.08227185];ProjParam=[43.3166666666667 -84.3333333333333 44.1833333333333 45.7 2000000*0.304800609601219 0 1.0000382 5];[GpsE,GpsN]=gNavGeog2ProjLambertConicConf(43.75,-83.1666666666667,EllipParam,ProjParam);
```

3.14 Lambert Conic Conformal to Geographic

function [B,L]=gNavProjLambertConicConf2Geog(E,N,EllipParam,ProjParam)

Convert Lambert Conic Conformal projection to Geographic coordinates. EPSG dataset coordinate operation method codes 9801,9802,9826,9803,1051.

Formulas source: "Geomatics Guidance Note 7, part 2 Coordinate Conversions&Transformations including Formulas", Report 373-7-2, October 2017.

EPSG 9801; EPSG 9802. EPSG recognises two variants of the Lambert Conic Conformal, the methods called 1SP (EPSG 9810) and 2SP (EPSG 9802).

EPSG 9826. 1SP West Orientated. In older mapping of Denmark and Greenland the Lambert Conic Conformal (1SP) is used with axes positive north and >>west<<.

EPSG 9803. 2SP Belgium. In 1972, in order to retain approximately the same grid coordinates after a change of geodetic datum, a modified form of the two standard parallel case was introduced in Belgium. In 2000 this modification was replaced through use of the regular Lambert Conic Conformal (2SP) method with appropriately modified parameter values.

EPSG 1051. 2SP Michigan. In 1964, the US state of Michigan redefined its State Plane CS27 zones, changing them from being Transverse Mercator zones orientated north-south to being Lambert Conic Conformal zones orientated east-west to better reflect the geography of the state.

EPSG 9817. Lambert Conic Near-Conformal. The Lambert Conformal Conic with one standard parallel formulas, as published by the Army Map Service, are still in use in several countries. However in some countries the expansion formulas were truncated to the third order and the map projection is not fully conformal.

Parameters:

E,N – rows, Easting, Northing in meters;

>>> ProjParam – 1SP parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin(K0) False_easting(FE) False_northing(FN) 1];

>>> ProjParam – 2SP parameters [Latitude_of_false_origin(Bf) Longitude_of_false_origin(Lf) Latitude_of_1st_standard_parallel(B1) Latitude_of_2nd_standard_parallel(B2) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 2];

>>> ProjParam – 1SP West Orientated parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin(K0) False_easting(FE) False_northing(FN) 3];

>>> ProjParam – 2SP Belgium parameters [Latitude_of_false_origin(Bf) Longitude_of_false_origin(Lf) Latitude_of_1st_standard_parallel(B1) Latitude_of_2nd_standard_parallel(B2) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 4];

>>> ProjParam – 2SP Michigan parameters [Latitude_of_false_origin(Bf) Longitude_of_false_origin(Lf) Latitude_of_1st_standard_parallel(B1) Latitude_of_2nd_standard_parallel(B2) Easting_at_false_origin(EF) Northing_at_false_origin(NF) Ellipsoid_scaling_factor(K) 5];

EllipParam – ellipsoid parameters [Semi_major_axis_a Eccentricity_et];

B,L – rows, Latitude, Longitude in degrees;

Examples:

```
>> EllipParam=[6378206.400 0.08227185];ProjParam=[18 -77 1 250000 150000
1];[GpsE,GpsN]=gNavProjLambertConicConf2Geog(255966.58,142493.51,EllipParam,ProjParam);
>> EllipParam=[6378206.400 0.08227185];ProjParam=[27.8333333333333 -99 28.3833333333333
30.2833333333333 2000000*0.304800609601219 0
```

2];[GpsE,GpsN]=gNavProjLambertConicConf2Geog(903277.79915962,77650.9425731394,EllipParam,
ProjParam);
>> EllipParam=[6378388 0.08199189];ProjParam=[90 4.35693972222222 49.833333333333333
51.166666666666667 150000.01 5400088.44
4];[GpsE,GpsN]=gNavProjLambertConicConf2Geog(251763.20,153034.13,EllipParam,ProjParam);
>> EllipParam=[6378206.400 0.08227185];ProjParam=[43.3166666666667 -84.3333333333333
44.1833333333333 45.7 2000000*0.304800609601219 0 1.0000382
5];[GpsE,GpsN]=gNavProjLambertConicConf2Geog(703582.144930931,48832.2520113746,EllipParam,
ProjParam);

3.15 Geographic to Polar Sterographic

function [E,N]=gNavGeog2ProjPolarStereo(B,L,EllipParam,ProjParam)

Convert Geographic coordinates to Polar Stereographic projection. EPSG dataset coordinate operation method codes 9810,9829,9830.

Formulas source: "Geomatics Guidance Note 7, part 2 Coordinate Conversions&Transformations including Formulas", Report 373-7-2, October 2017.

For the polar sterographic projection, three variants are recognised, differentiated by their defining parameters. In the basic variant (variant A) the latitude of origin is either the north or the south pole, at which is defined a scale factor at the natural origin, the meridian along which the northing axis increments and along which intersecting parallels increment towards the north pole (the longitude of origin), and false grid coordinates. In variant B instead of the scale factor at the pole being defined, the (non-polar) latitude at which the scale is unity – the standard parallel – is defined. In variant C the latitude of a standard parallel along which the scale is unity is defined; the intersection of this parallel with the longitude of origin is the false origin, at which grid coordinate values are defined.

EPSG 9810. In the basic variant 'A' the latitude of origin is either the north or the south pole.

EPSG 9829. In variant 'B' instead of the scale factor at the pole being defined, the (non-polar) latitude at which the scale is unity – the standard parallel – is defined.

EPSG 9830. In variant 'C' the latitude of a standard parallel along which the scale is unity is defined; the intersection of this parallel with the longitude of origin is the false origin, at which grid coordinate values are defined.

Parameters:

B,L – rows, Latitude, Longitude in degrees;

>>> ProjParam – 'A' parameters [Longitude_of_natural_origin(L0) Scale_at_natural_origin(K0) False_easting(FE) False_northing(FN) key]; key set to 10 for south pole and 11 for north pole;

>>> ProjParam – 'B' parameters [Latitude_of_standard_parallel(Bf) Longitude_of_natural_origin(L0) False_easting(FE) False_northing(FN) 2];

>>> ProjParam – 'C' parameters [Latitude_of_standard_parallel(Bf) Longitude_of_natural_origin(L0) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 3];

EllipParam – ellipsoid parameters [Semi_major_axis_a Eccentricity_et];

E,N – rows, Easting, Northing in meters;

For the south pole case, latitude B is negative; longitude L measured clockwise in the projection plane.

For the north pole case, longitude L measured anticlockwise in the projection plane.

Examples:

```
>> EllipParam=[6378137 0.081819190842];ProjParam=[0 0.994 2000000 2000000
11];[GpsE,GpsN]=gNavGeog2ProjPolarStereo(73,44,EllipParam,ProjParam);
>> EllipParam=[6378137 0.081819191];ProjParam=[-71 70 6000000 6000000
2];[GpsE,GpsN]=gNavGeog2ProjPolarStereo(-75,120,EllipParam,ProjParam);
>> EllipParam=[6378388 0.081991890];ProjParam=[-67 140 300000 200000
3];[GpsE,GpsN]=gNavGeog2ProjPolarStereo(-66.6052277777778,140.0714,EllipParam,ProjParam);
```

3.16 Polar Stereographic to Geographic

function [B,L]=gNavProjPolarStereo2Geog(E,N,EllipParam,ProjParam)

Convert Polar Stereographic projection to Geographic coordinates. EPSG dataset coordinate operation methods codes 9810,9829,9830.

Formulas source: "Geomatics Guidance Note 7, part 2 Coordinate Conversions&Transformations including Formulas", Report 373-7-2, October 2017.

For the polar sterographic projection, three variants are recognized, differentiated by their defining parameters. In the basic variant (variant A) the latitude of origin is either the north or the south pole, at which is defined a scale factor at the natural origin, the meridian along which the northing axis increments and along which intersecting parallels increment towards the north pole (the longitude of origin), and false grid coordinates. In variant B instead of the scale factor at the pole being defined, the (non-polar) latitude at which the scale is unity – the standard parallel – is defined. In variant C the latitude of a standard parallel along which the scale is unity is defined; the intersection of this parallel with the longitude of origin is the false origin, at which grid coordinate values are defined.

EPSG 9810. In the basic variant 'A' the latitude of origin is either the north or the south pole.

EPSG 9829. In variant 'B' instead of the scale factor at the pole being defined, the (non-polar) latitude at which the scale is unity – the standard parallel – is defined.

EPSG 9830. In variant 'C' the latitude of a standard parallel along which the scale is unity is defined; the intersection of this parallel with the longitude of origin is the false origin, at which grid coordinate values are defined.

Parameters:

E,N – rows, Easting, Northing in meters;

>>> ProjParam – 'A' parameters [Longitude_of_natural_origin(L0) Scale_at_natural_origin(K0) False_easting(FE) False_northing(FN) key]; key set to 10 for south pole and 11 for north pole;

>>> ProjParam – 'B' parameters [Latitude_of_standard_parallel(Bf) Longitude_of_natural_origin(L0) False_easting(FE) False_northing(FN) 2];

>>> ProjParam – 'C' parameters [Latitude_of_standard_parallel(Bf) Longitude_of_natural_origin(L0) Easting_at_false_origin(EF) Northing_at_false_origin(NF) 3];

EllipParam – ellipsoid parameters [Semi_major_axis_a Eccentricity_et];

B,L – rows, Latitude, Longitude in degrees;

For the south pole case, latitude B is negative; longitude L measured clockwise in the projection plane. For the north pole case, longitude L measured anticlockwise in the projection plane.

Examples:

```
>> EllipParam=[6378137 0.081819190842];ProjParam=[0 0.994 2000000 2000000 11];[B,L]=gNavProjPolarStereo2Geog(3320416.75,632668.43,EllipParam,ProjParam);
>> EllipParam=[6378137 0.081819191];ProjParam=[-71 70 6000000 6000000 2];[B,L]=gNavProjPolarStereo2Geog(7255380.79,7053389.56,EllipParam,ProjParam);
>> EllipParam=[6378388 0.081991890];ProjParam=[-67 140 300000 200000 3];[B,L]=gNavProjPolarStereo2Geog(303169.52,244055.72,EllipParam,ProjParam);
```

4. Coordinates/time simple manipulations

4.1 Arc to meters

function dXY=gNavArc2Len(B,L,EllipParam)

Length in meters along Latitude and Longitude arc in degrees. The Simpsons equation used for dXY calculation. The (B(1)+B(2))/2 used for dY calculation.

Parameters:

B – Latitude, 2 values in degrees;

L – Longitude, 2 values in degrees;

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et];

dXY=[dX dY] – length along Latitude and Longitude.

Example:

```
>> dXY=gNavArc2Len([29.99 30.01],[141.99 142.01],[6378137 0.081819190842]);
```

4.2 Angle/scale between Geographic and Transverse Mercator

function [dm,dAng]=gNavReducingHeadingToProjUtm(B,L,et,L0,Sc,key)

Calculate Scale and Angle between Geographic Line and Transverse Mercator projection Line (EPSG 9807 Projection).

Parameters:

B – Latitude;

L – Longitude;

ProjParam – TM parameters [not_used Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin not_used not_used];

EllipParam – ellipsoid parameters [not_used Eccentricity_et];

key – dAng calculation method number (1, 2 or 3);

dm – Gauss Scale; L(proj)=L(geographic_arc)*m;

dAng – Gauss Angle; Heading(UTM/TM)=Heading(geographic_arc)+dAng, the clockwise rotation is +

Example:

```
>> [m,dAng]=gNavReducingHeadingToProjUtm([60 61],[142 143], EllipParam,ProjParam,2);
```

4.3 Angle/scale between Geographic and Rectangular

function [m,ang]=gNavReducingHeading(B,L,EllipParam,ProjForvFunc,ProjParam)

Calculate scale and angles for Geographic-to-Rectangular coordinates transformation.

Parameters:

B – Latitude, 4 values in degrees;

L – Longitude, 4 values in degrees;

There are four points:

|4 (along Longitude)

|

1------2 (along Latitude)

|

|3

EllipParam – ellipsoid parameters [Semi major axis_a Eccentricity_et];

ProjForvFunc – function name for Geographic-to-Rectangular coordinates transformation; 'gNGeog2Utm' -- for UTM.

ProjParam – coordinates transformation's function; UTM parameters [Latitude_of_natural_origin(B0) Longitude_of_natural_origin(L0) Scale_factor_at_natural_origin False_easting False_northing];

m – scale [mX;mY] >> LX(proj)=LX(geographic_arc)*mX;

ang – angles [angX;angY] >> HeadingX(proj)=HeadingX(geographic_arc)+angX, the clockwise rotation is +

Example.

```
>> [ang,m]=gNavCartesianTrans([60 60 59.99 60.01],[141.99 142.01 142 142],[6378137 0.081819190842],'gNavGeog2Utm',[0 138 0.9996 500000 0]);
```

4.4 Interpolate coordinate/time was repeated

function varargout=gNavCoordDerepeat(varargin)

Remove and interpolate repeating position [E,N] or time/position [Day,Second,E,N].

Parameters:

varargin=[E,N,meth] or [Dm,S,E,N,meth] >>

E,N – rows, Easting, Northing in meters;

Dm,S- serial date number of 1 corresponds to Jan-1-0000 and second per day;

meth – interpolation method ('linear','pchip', etc);

varargout=[Ei,Ni] or [Dmi,Si,Ei,Ni] >>

E,N – rows, Easting, Northing in meters were interpolated;

Dm,S – serial date number of 1 corresponds to Jan-1-0000 and second per day were interpolated;

Example.

```
>> [Er,Nr]=gNavCoordDerepeat([1 2 2 3 3 4 5 6],[1 1 1 2 3 4 5 5],'linear');
>> [Dr,Sr,Er,Nr]=gNavCoordDerepeat([95 95 95 95 95 95 95 95],[1 2 2 4 5 6 7 8],[1 2 2 3 3 4 5 6],[1 1 1 2 3 4 5 5],'linear');
```

4.5 2D-track smoothing and instantaneous direction calculation

function END=gNavTrackMadeGood2D(EN,zwin,rk,figDraw)

The point’s 2D-track smoothing and direction calculation.

Parameters:

EN – rows includes point position;

zwin – the smoothing window half without one point zwin=(window-1)/2; if zwin=5, than window=11;

rk – robust coefficient row (usually is rk=[3 2.5]);

END – rows includes [E_new N_new Direction]; smoothing position and direction;

figDraw – figure number for drawing (not drawing if empty).

The coordinate system:

^ N or y

|

----> E or x

Direction - from E left is +

Example (***Figure 4.1***)

```
>> [SgyHead,Head,Data]=gSgyRead('e:\temp\16_10.500a.sgy',[],[]);
>> [E,N]=gNavCoordDerepeat (Head.SourceY,Head.SourceX,'linear');
>> END=gNavTrackMadeGood2D([E;N],10,[3 2.5],1);
```

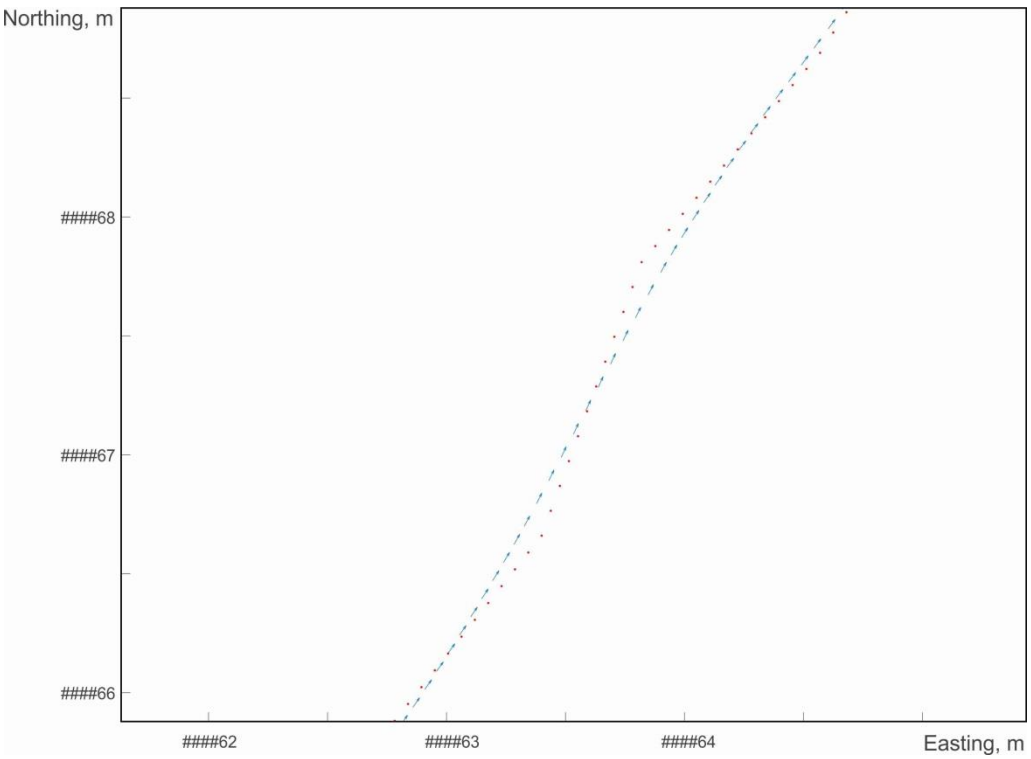

Figure 4.1 Point's 2D-track smoothing and direction calculation (function gNavTrackMadeGood2D)

Point position – red; smoothing point position – blue arrows.

4.6 The 2D-track-line direction calculation

function END=gNavTrackDirection2D(EN,rk,figDraw)

The 2D-track "direction" calculate (track approximated by line).

Parameters:

EN – rows includes point position;

rk – robust coeff row (multipurpose is rk=[3 2.5]);

END – rows includes direction information for two points [E1 N1 Direction;E2 N2 Direction]';

figDraw – figure number for drawing (not drawing if empty).

The coordinate system:

^ N or y

|

|---> E or x

Direction - from E left is +

The algorithm used in gGTrackMadeGood2D.

Example (*Figure 4.2*):

```
>> E=[1 2 3 4 5];N=[1 3 3 5 6];END=gNavTrackDirection2D([E;N],[3 2.5],1);
```

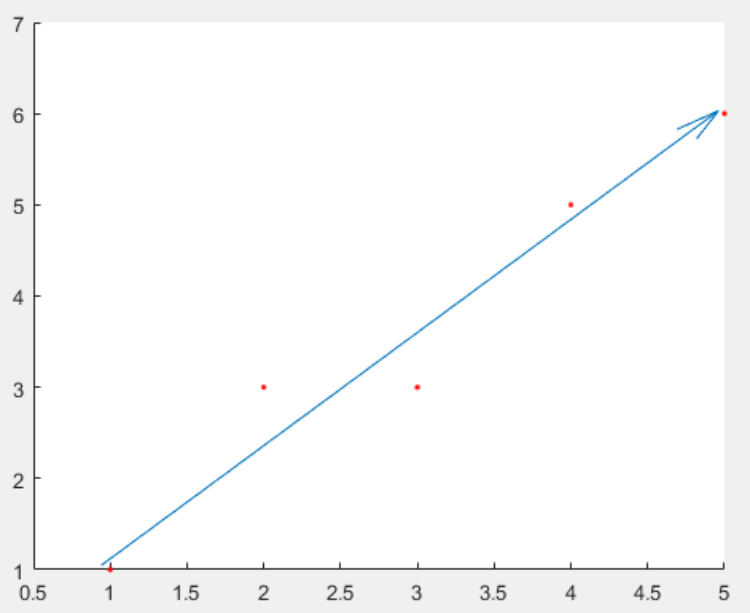

Figure 4.2 2D-track direction calculation (function gNavTrackDirection2D)

2D-track position – red; calculated points position – blue arrow.

4.7 Survey log read

function LLog=gNavLLogRead(fName)

Read LineLog for survey lines with tab-delimiter.

Parameters:

fName – reading file name;

LLog – structure with log records.

File row example: 2019/09/01 0085_CR_1_3(05) 11:47 12:32 29 1

File Format: Date LineName TimeStart TimeEnd Bearing LineNumber

Example

```
>> LLog=gNavLLogRead('c:\temp\log.txt');
```

5. Head, Pitch, Roll angles

The follow functions deal with Head, Pitch, Roll angles. The 3D Tait-Bryan matrixes are used.

The “forward” rotation is calculates using basic rotation matrixes by a formula:

$$R = R_z(\alpha)\,R_y(\beta)\,R_x(\gamma) = \overset{\text{yaw}}{\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}} \overset{\text{pitch}}{\begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix}} \overset{\text{roll}}{\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix}}$$

$$R = \begin{bmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{bmatrix}$$

https://en.wikipedia.org/wiki/Rotation_matrix

The “forward” rotation used Z1Y2X3 formula; the “reverse” rotation used X1Y2Z3 formula.

Proper Euler angles	Tait–Bryan angles
$X_1Z_2X_3 = \begin{bmatrix} c_2 & -c_3s_2 & s_2s_3 \\ c_1s_2 & c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 \\ s_1s_2 & c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$X_1Z_2Y_3 = \begin{bmatrix} c_2c_3 & -s_2 & c_2s_3 \\ s_1s_3 + c_1c_3s_2 & c_1c_2 & c_1s_2s_3 - c_3s_1 \\ c_3s_1s_2 - c_1s_3 & c_2s_1 & c_1c_3 + s_1s_2s_3 \end{bmatrix}$
$X_1Y_2X_3 = \begin{bmatrix} c_2 & s_2s_3 & c_3s_2 \\ s_1s_2 & c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 \\ -c_1s_2 & c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$X_1Y_2Z_3 = \begin{bmatrix} c_2c_3 & -c_2s_3 & s_2 \\ c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 & -c_2s_1 \\ s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 & c_1c_2 \end{bmatrix}$
$Y_1X_2Y_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & s_1s_2 & c_1s_3 + c_2c_3s_1 \\ s_2s_3 & c_2 & -c_3s_2 \\ -c_3s_1 - c_1c_2s_3 & c_1s_2 & c_1c_2c_3 - s_1s_3 \end{bmatrix}$	$Y_1X_2Z_3 = \begin{bmatrix} c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 & c_2s_1 \\ c_2s_3 & c_2c_3 & -s_2 \\ c_1s_2s_3 - c_3s_1 & c_1c_3s_2 + s_1s_3 & c_1c_2 \end{bmatrix}$
$Y_1Z_2Y_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_1s_2 & c_3s_1 + c_1c_2s_3 \\ c_3s_2 & c_2 & s_2s_3 \\ -c_1s_3 - c_2c_3s_1 & s_1s_2 & c_1c_3 - c_2s_1s_3 \end{bmatrix}$	$Y_1Z_2X_3 = \begin{bmatrix} c_1c_2 & s_1s_3 - c_1c_3s_2 & c_3s_1 + c_1s_2s_3 \\ s_2 & c_2c_3 & -c_2s_3 \\ -c_2s_1 & c_1s_3 + c_3s_1s_2 & c_1c_3 - s_1s_2s_3 \end{bmatrix}$
$Z_1Y_2Z_3 = \begin{bmatrix} c_1c_2c_3 - s_1s_3 & -c_3s_1 - c_1c_2s_3 & c_1s_2 \\ c_1s_3 + c_2c_3s_1 & c_1c_3 - c_2s_1s_3 & s_1s_2 \\ -c_3s_2 & s_2s_3 & c_2 \end{bmatrix}$	$Z_1Y_2X_3 = \begin{bmatrix} c_1c_2 & c_1s_2s_3 - c_3s_1 & s_1s_3 + c_1c_3s_2 \\ c_2s_1 & c_1c_3 + s_1s_2s_3 & c_3s_1s_2 - c_1s_3 \\ -s_2 & c_2s_3 & c_2c_3 \end{bmatrix}$
$Z_1X_2Z_3 = \begin{bmatrix} c_1c_3 - c_2s_1s_3 & -c_1s_3 - c_2c_3s_1 & s_1s_2 \\ c_3s_1 + c_1c_2s_3 & c_1c_2c_3 - s_1s_3 & -c_1s_2 \\ s_2s_3 & c_3s_2 & c_2 \end{bmatrix}$	$Z_1X_2Y_3 = \begin{bmatrix} c_1c_3 - s_1s_2s_3 & -c_2s_1 & c_1s_3 + c_3s_1s_2 \\ c_3s_1 + c_1s_2s_3 & c_1c_2 & s_1s_3 - c_1c_3s_2 \\ -c_2s_3 & s_2 & c_2c_3 \end{bmatrix}$

https://en.wikipedia.org/wiki/Euler_angles#Intrinsic_rotations

5.1 3D Tiat matrixes

function [HeadM,PitchM,RollM]=gNavTiat(Head,Pitch,Roll)

The basic rotation matrixes calculation.

Parameters:

Head – rotation angle for x0y in degree;

Pitch – rotation angle for z0x in degree;

Roll – rotation angle for y0z in degree;

HeadM – Z-basic rotation matrix for rotation in x0y (Heading);

PitchM – Y-basic rotation matrix for rotation in z0x (Pitch);

RollM – X-basic rotation matrix for rotation in y0z (Roll).

The rotated vector is defined in primary coordinate system.

The primary coordinate system:

^ x(forward/roll)

|

o---> y(right/pitch)

z(up/head)

All right/clockwise rotation sign is +.

Example.

```
>> [HeadM,PitchM,RollM]=gNavTiat(30,0,0);a=HeadM*[1 0 0]'; %right rotation to 30 degree for x0y
>> [Xm,Ym,Zm]=gNavTiat(8,5,30);a=Xm*Ym*Zm*[1.2 5 3]'; %right rotation for H - P - R angles.
```

5.2 Lever arm rotation

function XYZ_Lev=gNavTiatLever(Head,Pitch,Roll,Lever,RotDirect)

A number LeverArm's coordinates calculation using scalar Head,Pitch,Roll (3D Tait–Bryan matrixes rotation).

Parameters:

Head – row, rotation angle for x0y in degree [H_t1... H_tn];

Pitch – row, rotation angle for z0x in degree [P_t1... P_tn];

Roll – row, rotation angle for y0z in degree [R_t1... R_tn];

Lever – 3 rows with lever arms coordinates [Lx_1 Ly_1 Lz_1; Lx_2 Ly_2 Lz_2; ...; Lx_m Ly_m Lz_m]';

RotDirect – 'frw' for forward rotation; 'rev' for reverse rotation;

XYZ_Lev – "moved" lever arms coordinates: column is [Lx;Ly;Lz]; rows is t1...tn; 3rd-size is Lever_1...Lever_m

if Roll or Pitch or Head is scalar, then vector with necessary length will be created.

Coordinate system is:

^ x(forward/roll)

|

o---> y(right/pitch)

z(up/head)

All right/clockwise rotation sign is + (if vector is rotated relatively axis);

All left/unclockwise rotation sign is + (if axis are rotated relatively vector);

Example:

```
>> tr=[120;60;-80];tmp001=gNavTiatLever(-2,-2,-2,tr,'frw');
```

function varargout=gNavTiatLeverN(HPR,Lx,Ly,Lz,RotDirect)

LeverArm's coordinates calculation using dimensions Head,Pitch,Roll (3D Tait–Bryan matrixes rotation). About 90 times faster for [Head,Pitch,Roll] dimensions, than the loop using gNavTiatLever function.

Parameters:

HPR – 3 rows with [Head;Pitch;Roll] coordinates [Head1...Headm; Pitch1...Pitchm; Roll1...Rollm];

Lx – dimension of LeverX coordinates;

Ly – dimension of LeverY coordinates;

Lz – dimension of LeverZ coordinates;

RotDirect – 'frw' for forward rotation; 'rev' for reverse rotation;

varargout=[X1,Y1,Z1...Xn,Yn,Zn] - "moved" lever arm coordinates, with size same [Lx,Ly,Lz] for different [Head,Pitch,Roll]

if Lx or Ly or Lz is scalar, then vector with necessary size will be created.

Coordinate system is:

^ x(forward/roll)

|

o---> y(right/pitch)

z(up/head)

All right/clockwise rotation sign is + (if vector is rotated relatively axis);

All left/unclockwise rotation sign is + (if axis are rotated relatively vector);

Example

```
>> [X,Y,Z]=gNavTiatLeverN([-10;-11;-12],[20;21;22],[15;16;17],[10;15;20],'frw');
[x,y,z]=gNavTiatLeverN([10;11;12],X,Y,Z,'rev');
>> [X1,Y1,Z1,X2,Y2,Z2]=gNavTiatNLever([10 15 20;10 15 20]',[10;11;12],[20;21;22],[15;16;17],'frw');
>> [Lx,Ly,Lz]=meshgrid(-3:.02:3,-3:.02:3,-3:.02:3);
[X1,Y1,Z1]=gNavTiatNLever([10;15;20],Lx,Ly,Lz,'frw');
```

function varargout=gNavTiatNLever(Head,Pitch,Roll,Lever,RotDirect)

LeverArm's coordinates calculation using dimensions Head,Pitch,Roll (3D Tait–Bryan matrixes rotation). About 90 times faster for [Head,Pitch,Roll] dimensions, than the loop using gNavTiatLever function.

Parameters:

Head – dimension of heading; rotation angle for x0y in degree [H_t1... H_tn];

Pitch – dimension of pitch; rotation angle for z0x in degree [P_t1... P_tn];

Roll – dimension of roll; rotation angle for y0z in degree [R_t1... R_tn];

Lever – 3 rows with lever arms coordinates [Lx1...Lxm; Ly1...Lym; Lz1...Lzm];

RotDirect – 'frw' for forward rotation; 'rev' for reverse rotation;

varargout=[X1,Y1,Z1...Xn,Yn,Zn] – "moved" lever arm coordinates, with size same [Head,Pitch,Roll] for different Levers

if Roll or Pitch or Head is scalar, then vector with necessary size will be created.

Coordinate system is:

^ x(forward/roll)

|

o---> y(right/pitch)

z(up/head)

All right/clockwise rotation sign is + (if vector is rotated relatively axis);

All left/unclockwise rotation sign is + (if axis are rotated relatively vector);

Example:

```
>> [X,Y,Z]=gNavTiatNLever([10;11;12],[20;21;22],[15;16;17],[10;15;20],'frw');
[x,y,z]=gNavTiatNLever([-10;-11;-12],[-20;-21;-22],[-15;-16;-17],[X;Y;Z],'rev');
>> [X1,Y1,Z1,X2,Y2,Z2]=gNavTiatNLever([10;11;12],[20;21;22],[15;16;17],[10 15 20;10 15 20]','frw');
>> [dH,dP,dR]=meshgrid(-3:.02:3,-3:.02:3,-3:.02:3);
[X1,Y1,Z1]=gNavTiatNLever(dH,dP,dR,[10;15;20],'frw');
```

script gNavTiatTest

Calculate "error ellipsoid" for USBL misalignment angles, using 2 ship's location with target's coordinates by USBL.

6. Layback calculation

6.1 Towing body coordinates calculation

function ENout=gNavLayback(ENLZA,angL)

Towing fish position calculation using tow-point position, cable length, height from fish to tow-point.

Parameters:

ENLZA – rows includes [E;N;L;Z;A]; EN- tow point position, L- cable length, Z- height from fish to tow-point;A- TowBody heading angle (if not presented, than towbody-to-towpoint ray not rotated);

angL – survey line direction (constant);

ENout – towing fish position;

Current algorithm is similar to "Dragging" algorithm described in [Data Acquisition Software...].

The coordinate system:

^ N

|

x---> E

z(down)

Model's conditions:

1) tow point is joint;

2) if "cable is sag", then fish position is not changed;

3) for fist point - towfish is installed back in the fist-to-second points direction;

4) Z is fish depth and tow-point-height sum;

5) if A-heading angle is presented, than rotate TowPoint-to-TowBoby vector in direction opposite A.

Example 1.

```
>> EN=gNavLayback([GpsE;GpsN;CableCounter;TPHeigth+DepthSensor,VesselHeading],25);
```

Example 2 (***Figure 6.1***)

```
>> [SgyHead,Head,Data]=gSgyRead('e:\temp\16_10.500a.sgy',[],[]);
>> [E,N]=gNavCoordDerepeat(Head.SourceY,Head.SourceX,'linear');
>> XY=gNavLayback([E;N;E.*0+1000;E.*0+1],[]);axis equal;
```

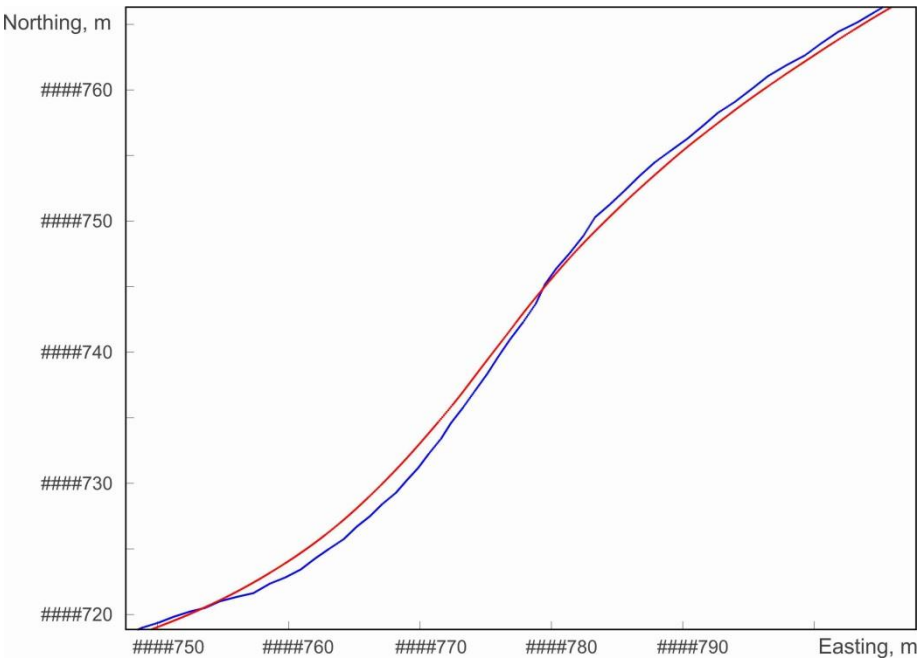

Figure 6.1 Towing fish position calculation using Layback (function gNavLayback)

Tow point position – blue; towing fish position – red.

6.2 Towing body coordinates calculation using sea current

function EN=gNavLaybackCurrent(ENLZt,ENFA,figDraw)

In progress

Citation

1) Geomatics Guidance Note Number 7, part 2: Coordinate Conversions and Transformations including Formulas //Revised - April 2015, page 133
2) Data Acqusition Software / 25479-01 Rev.K / Geometrics Inc
3) G−882 Cesium Marine Magnetometer. Operation Manual, 2005// 25919-OM REV. D
4) Common HYPACK® Drivers Interfacing Notes Updated January / 2017 // Towfish.dll, p.1-71 to 1-76
5) How to Layback // https://confluence.qps.nl/qinsy/ru/kak-primenyat-raschetnuyu-pozitsiyu-buksiruemogo-ustrojstva-35587283.html
6) How to Smooth Vessel or ROV Track // https://confluence.qps.nl/qinsy/en/how-to-smooth-vessel-or-rov-track-35587317.html