How to use the „Geodetic Transformations Toolbox“?

For a start, this toolbox does nothing which is “new” – all the functionality is well known and used in hundreds of commercial and non-commercial products all over the world. There even are similar tools on Matlab File Exchange with different key aspects and usability, of course. Now, why another geodetic transformation and projection toolbox then?

Basically, these functions are meant for usage in the exercises of “Geodesy and Geoinformatics” studies at Technische Universität München. So it was the goal to combine a set of functions which fits exactly our educational needs with full control on source code, updates etc.
Additionally, I was not very satisfied with some code I found on the net; e.g. I did not find any Matlab-coded transformation to UTM projection which could handle non-standard zone width or includes polar stereographic projection.
And last reason, programming is fun and instructional: if you can code it, you have understood it ☺

Now have a few examples and explanations on the usage of the implemented functions. Whenever a function is mentioned the first time, it is highlighted in yellow. For further function details in the examples, please refer to the function help text.

A great acknowledgement goes to Andrea Pinna from the University of Cagliari. He improved my coding in the Helmert parameter estimation functions for speed and memory efficiency, so that it can work on millions of points now (which I never tried). Thanks a lot!

1. ***A first look on different transformation types***

Before starting with applications, let’s have a short look on different standard transformation types included in the toolbox and what has to be obeyed. The commonly used transformation types in geodesy are similarity transformation, affine transformation and projective transformation.

Many tasks in geodesy are done using similarity transformations, i.e. coordinate relations in both systems may differ in position, orientation and scale, but segment ratio and angles are preserved. One may use similarity transformations in 1D (using 1 translation in z and 1 scale factor), in 2D (using 2 translations in x,y, 1 orientation rotation around z-axis and 1 scale factor identical for both axes) and in 3D (using 3 translations, 3 rotations and 1 scale factor):

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}_B = \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}_B + s \cdot R(\alpha,\beta,\gamma) \cdot \begin{pmatrix} x \\ y \\ z \end{pmatrix}_A$$

This toolbox offers the function-files `d1trafo.m`, `d2trafo.m` and `d3trafo.m`.

For proper handling, we first need to define how the parameter set needs to look like. In geodesy we have coordinates given in a source system A (e.g. WGS84) and want to transform them to a destination system B (e.g. state system).
In the transformation formulas for any dimension used in this toolbox,

- the systems are right-handed
- the translation vector describes the origin of the system A in terms of the system B.
- the rotation is defined to convert the source system axes A in the direction of the destination system axes B as a passive rotation (i.e. it describes the orientation of system B in system A).
  There are also opposite definitions of the rotation angle which describe the orientation of system A in system B. If your parameter set is that way, you need to change the rotation element signs.

For 3D transformations the rotation matrix R is a combination of three elemental rotations around the three axes. As there are many possibilities to combine these elemental rotations in different order, we need to define which rotation order is to be used in the Geodetic Transformations Toolbox[1]. We use the Tait-Bryan angle order XYZ for passive intrinsic rotations[2]:

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

The necessary parameters can be calculated by use of a least-squares adjustment if sufficient identical points are given to solve the corresponding equation system (2 points in1D or 2D, 3 points in 3D). Please make sure, that the ID points are sufficiently distributed, i.e. they must not lie close to a line in 2D and close to the same plane in 3D as you would get a configuration defect then.
The toolbox offers the functions `helmert1d.m`, `helmert2d.m` and `helmert3d.m`

Transformation formulas are highly non-linear and therefore need to be linearized when parameters are estimated. This is of no harm for 1D or 2D and also in 3D if rotation angles are small[3], but bears problems if they are not. For big rotation angles in 3D, adjustment may fail if initial approximation values are not good enough, so you will need an idea of the most probably result in advance. If at least 4 identical points are given, an affine approach is used to determine the (possibly big) rotation angles automatically. This works well – with one exception: if rotation around y-axis is close to $\pi/2$ or $3\pi/2$, rotations around x- and z-axes cannot be solved unambiguously[4]. In that case a warning is thrown. To overcome that

[1] A good overview on rotation definitions is given by Wikipedia: http://en.wikipedia.org/wiki/Euler_angles. See especially the "Rotation matrix" section.
[2] If you need to use d3trafo.m with rotation parameters according to another angle order, you may simply do so by adopting the function source code ("Create rotation matrix" section). Please be aware, that other functions in this toolbox which also handle with transformation parameters also rely on the defined rotation order.
[3] In many cases, especially when transforming GNSS data to local systems, this is an acceptable assumption.
[4] This does not mean that the solution will be wrong – it just will bring up one of different possible solutions which all are nearly of the same accuracy.

limitation, if at least 4 ID points are given, an affine transformation may be used instead of a similarity transformation.

Mathematically speaking, similarity transformations are just special cases of affine transformations. An affine transformation is a linear 12-parameter approach where each transformed coordinate is dependent on all 3 coordinates in the initial system

$$\begin{aligned} x' &= x_0 + a_1 x + a_2 y + a_3 z \\ y' &= y_0 + a_4 x + a_5 y + a_6 z \\ z' &= z_0 + a_7 x + a_8 y + a_9 z \end{aligned}$$

with x_0, y_0, z_0 being the translation and a_1 to a_9 being additional affine parameters.
The major advantage is the linearity of the equation system which makes it easy to calculate the parameters without iteration or the need of approximation values. On the other side, nine affine parameters a_1 to a_9 need to be used to describe only six geometric parameters: 3 rotations and 3 scale factors (one per axis). Therefore the values of the affine parameters do not have a geometric meaning directly. Of course, it is possible to derive the geometric parameters from the affine ones (except y-axis-rotation is $\pi/2$ or $3\pi/2$, as mentioned above), but if they are not needed e.g. for geometric interpretation of the results, this is usually not done. Additionally, it is not possible, e.g., to use only one scale factor or keep one scale factor fixed as different geometric parameters are combined with others in the affine approach.
This toolbox offers `d2affinetrafo.m` and `d3affinetrafo.m` to do the transformation, and `helmertaffine2d.m` and `helmertaffine3d.m` to calculate the parameters.

Another transformation type used in geodesy is projective transformation. As these formulas are used for mapping a 2D/3D scene on 2D, it is related to tasks in photogrammetry. The basic formulas for image coordinates derived from 3D points are

$$\begin{aligned} x' &= \frac{a_1 x + a_2 y + a_3 z + a_4}{a_9 x + a_{10} y + a_{11} z + 1} \\ y' &= \frac{a_5 x + a_6 y + a_7 z + a_8}{a_9 x + a_{10} y + a_{11} z + 1} \end{aligned}$$

which e.g. describe how a 3D object will be seen in camera view. It does not longer preserve segment ratio and angles. As the affine transformation is a linear approach to a geometric problem, the parameters of the 3D projective formulas can be converted to inner and outer camera orientation parameters if necessary[5]. Really, this is the main task of its usage, at it allows camera calibration together with image acquisition as long as at least 6 reference points are known distributed in 3D space (called DLT – Direct Linear Transformation). The reverse transformation is of not much interest as it cannot be unambiguous[6]. In 2D case, projective transformation mainly is used for image (photography) rectification with respect to a given plane.
This toolbox offers `d2projectivetrafo.m` and `d3projectivetrafo.m` to do the

[5] Outer orientation is the location of the projection center and its corresponding axes' rotations with respect to the source coordinate system (6 parameters). Inner orientation is the principal point and camera constant (3 parameters). Additionally image shear and scaling parameters can be calculated.
[6] Of course, it will become very interesting if more than one viewpoint are combined, because then 3D coordinates can be obtained using 2D images – the principle of stereo image photography.

transformation and `helmertprojective2d.m` and `helmertprojective3d.m` to calculate the parameters.

2. ***Transform GNSS (GPS) coordinates to a projection system***

Let's assume, we have some coordinate information from a GNSS[7] system with respect to the WGS84 ellipsoid and want to get projected coordinates in a given national system, e.g. for mapping.

P1: L = 11°20' B = 48° h = 600 m

If the coordinates are geographic (ellipsoidal), we need to transform them to Cartesian coordinates:

```
P2=ell2cart(P1,'wgs84')
```

P2 = 4192737.07607126 840328.46328838 4717322.21701069

Don't forget to specify the ellipsoid to use ('wgs84'), as it is not the default one. Now we need to know which ellipsoid is the reference of our mapping system, as we have to transform the WGS84-coordinates to that ellipsoid. This is done using a 7-parameter similarity transformation (3 translations, 3 rotations, 1 scale factor) which is also known as "Helmert-Transformation". Sometimes standard parameter sets can be used (e.g. there are parameter sets for a whole state, but with limited accuracy), but if not, they need to be calculated first. To do so, we need identical points in both systems, i.e. points from which coordinates are known with respect to both ellipsoids. The closer these points lie to P1, the better. We may use official cadastral coordinates or personal measurement data (e.g. GNSS measurements acquired when standing on a known point). So let's assume, we know the following point coordinates of cadastral points around P1 (spanning an area of about 40 x 80 km^2)

ID	WGS84 X	WGS84 Y	WGS84 Z	GK east	GK north	GK height	Undulation
ID1	4178260.42	828039.59	4732196.07	4441346.294	5340573.080	539.124	46.150
ID2	4175739.94	853475.29	4729951.18	4466749.642	5336972.649	565.442	45.668
ID3	4190726.62	839398.78	4719382.28	4449866.289	5321182.832	644.308	46.131
ID4	4213686.65	820422.71	4702783.31	4426535.047	5296320.270	957.808	46.643
ID5	4239168.32	828294.41	4678299.48	4428934.586	5260198.724	731.910	48.857

In this example, as P1 is in Bavaria, we are working with German GK coordinates which is type of transverse Mercator (TM) projection based on the Bessel-Ellipsoid ('besseldhdn').

To be able to compute the transformation parameters, we need to know the Cartesian coordinates with respect to the Bessel ellipsoid, so we need to "unmap" the GK coordinates.

[7] GNSS (Global Navigation Satellite System) is more general than GPS, which only names the US system. Modern receivers can additionally use GLONASS (Russia) and, in future, will use GALILEO (Europe), COMPASS (China), and others.

a) Unmapping without use of undulation

NOTE: The height in most national systems is not an ellipsoidal height which is purely mathematic, but connected to physics by use of gravimetric information.[8] Therefore it is not really accurate to use the GK height as ellipsoidal height, because both vary by the undulation (in Bavaria about +47m); but as we want to calculate a transformation with respect to our identical points, a pure shift in height is of no interest as long as it is constant in all points. If the calculation area is relatively small, we may assume it is (but, of course, we may lose little accuracy). This is the case for working areas which do not extend more than a few km^2.

With our point field being broader, let's see what happens without undulation first:

```
ID_ELL=tm2ell(ID_GK,'gk')
```

```
ID_ELL =      11.2108503729801      48.2013560143094          539.124
              11.5529036293617      48.1708116428913          565.442
              11.3277515252722      48.0276937195613          644.308
              11.0191756457828      47.8018381331175          957.808
              11.0570713190464      47.4772324075114          731.910
```

We could have omitted the 'gk' projection definition as it is the default value.

Now, again, transform geographic coordinates to Cartesian ones:

```
ID_CART=ell2cart(ID_ELL,'besseldhdn')
```

```
ID_CART =     4177627.88519728      828014.781833995      4731750.31639371
              4175107.44627091      853450.591600544      4729505.59844331
              4190094.2012223       839374.173512799      4718936.71090978
              4213054.13253265      820398.152807951      4702337.82116581
              4238534.84436367      828270.013511677      4677852.7653253
```

We already can see that the coordinates of the identical points vary by several 100m between the two systems on WGS84 and Bessel ellipsoid due to different ellipsoid shape and gravity center.

Now compute the transformation parameters in the direction WGS84 -> Bessel. First we will use the 7-parameter similarity transformation, Bursa-Wolf type:

```
[tp,rc,ac,tr]=helmert3d(ID_WGS84,ID_CART)
```

[8] „The same height" often is defined by the fact that water would not flow from one point to the other. Using ellipsoidal heights, this is not fulfilled as mass attraction is not considered. If you do so, you get heights with respect not to an ellipsoid, but to a so-called geoid which is a reference surface of the same geopotential. Regrettably, the geoid is a bumpy surface which cannot be described mathematically. Because of that, there are interpolation models which comprise the difference between ellipsoid and geoid zero-level, the so-called undulation. In terms of mathematical notation, we calculate: $h_{ell} = h_{proj}$ + undulation.
Undulation values worldwide are -110m to +80m, of course in closer areas they are widely constant and might by omitted. But e.g. in mountains it may be that even neighboring points vary in undulation by several cm due to the irregular distributed masses.

```
tp =    -701.568073251141
        -102.281892479284
        -359.905945863748
        7.37636968498118e-006
        -1.75862691992326e-005
        -1.04660412621071e-005
        0.999998745963201

rc =    0     0     0

ac =    26.0286028109207
        83.767630473504
        32.06845055104
        1.04987177218062e-005
        4.92078882323577e-006
        8.98529048792421e-006
        3.85864936172485e-006

tr =    -0.280973413959146      -0.124859500792809      -0.324613041244447
        0.0626319842413068      0.0597453825175762      -0.0120819211006165
        0.194919862318784       0.055114968214184       0.146854887716472
        0.218333503231406       -0.0372454045573249     0.47009860817343
        -0.194911937229335      0.0472445546183735      -0.280258538201451
```

The tp are the transformation parameters (3 translations of the ellipsoid origin, 3 rotations and 1 scale factor), the values in ac give their standard deviations.[9] The rc value is the rotation center of the transformation which is the center of the source ellipsoid WGS84. Of more interest are the tr values which show the coordinate residuals in the identical points after transformation. It gives the best hint on the quality of identical points and the parameter set. Here we can see 2D position quality of up to 22 cm and height quality of up to 50 cm, which is expectable within our working area, but of course not very good.

There is another type of calculation for the datum transformation, which is called the Molodensky-Badekas type. It also is a 7-parameter similarity transformation in principle, but rotation center is the centroid of the identical points in the source datum, so 3 additional parameters are needed to specify it. We can calculate its parameters also:

```
[tp,rc,ac,tr]=helmert3d(ID_WGS84,ID_CART,'10p')
```

```
tp =    -632.688082638383
        -24.613346606959
        -445.821552417798
        7.37636970144985e-006
        -1.7586269220385e-005
        -1.04660412703721e-005
        0.999998745963186

rc =    4199516.39          833926.156          4712522.464
```

[9] Don't mind them being relatively big – this is a result of the geocentric calculation. The distance of the rotation and coordinate origin (earth radius) is much bigger than the distance between the identical points, so by leverage effect it results in a very big statistical uncertainty. Additionally, using the Bursa-Wolf transformation type translation and rotation values are highly constrained.

```
ac =    0.128351334226784
        0.128351334226784
        0.128351334226784
        1.04987177251999e-005
        4.9207888258544e-006
        8.98529049022552e-006
        3.85864936250565e-006

tr =    -0.280973414424807      -0.124859501142055      -0.324613039381802
        0.0626319833099842      0.0597453826339915      -0.0120819201692939
        0.194919862318784       0.0551149684470147      0.146854889579117
        0.218333504162729       -0.0372454046737403     0.470098609104753
        -0.194911936298013      0.0472445547347888      -0.280258536338806
```

As can be seen, the values of rotation and scale are identical to some 10^{-15} while, of course, translation is significantly different. In 10-parameter-type also standard deviations of translations have quite interpretable meaning. The results of transformation and identical point residuals, though, are identical. So it does not make a difference which type is used[10].

At this point we'll break the mapping process without undulation and do the same while using it. Further results without are shown later.

<u>Unmapping with undulation</u>

Now let's see what happens when we add undulation information[11]. Again we compute the geographic ellipsoidal coordinates from the projected ones:

```
ID_ELL=tm2ell(ID_GK,'gk',UND)
```

```
ID_ELL =    11.2108503729801    48.2013560143094     585.274
            11.5529036293617    48.1708116428913     611.110
            11.3277515252722    48.0276937195613     690.439
            11.0191756457828    47.8018381331175    1004.451
            11.0570713190464    47.4772324075114     780.767
```

It gives the same Latitude, Longitude results and undulation value is just added to projected height. Now transform them to Cartesian coordinates,

```
ID_CART=ell2cart(ID_ELL,'besseldhdn')
```

```
ID_CART =   4177658.05791163    828020.762130513    4731784.7208391
            4175137.28577171    853456.691213534    4729539.62733011
            4190124.45132214    839380.233317501    4718971.0078395
            4213084.88486561    820404.141137641    4702372.37551972
            4238567.25299313    828276.346618975    4677888.7733652
```

[10] Strictly speaking, the 10-parameter transformation is a one-way transformation only. As the rotation center coordinates are defined and applied to the source datum only, they must not be used to transform points back from destination to source. For practical purposes though, a datum transformation is always connected with very small rotations only and the vector from destination centroid to source centroid is small compared to ellipsoid center distance, so the back-transformation may be used without significant divergence.

[11] Undulation values may be gotten by federal cadastral offices, usually they are not for free.

and compute transformation parameters

```
[tp,rc,ac,tr]=helmert3d(ID_WGS84,ID_CART)
```

```
tp =        -539.273135594615
            -73.2804996002362
            -510.593550254333
            8.69722747224154e-007
            1.69069540584007e-005
            -1.08350902896085e-005
             1.00000618671287

rc =    0    0    0

ac =    8.52208615017402
        27.4259681804932
        10.4989159387799
        3.43737834858921e-006
        1.61104121645571e-006
        2.94181397157601e-006
        1.26334079821778e-006

tr =    0.0416802992112935       -0.0571302799507976      0.0464044557884336
        0.00277828890830278      0.0438527016667649       -0.0784803116694093
        0.0644033406861126       0.0298502605874091       0.00179089792072773
        -0.160344012081623       -0.109273210749961       0.0374712059274316
        0.0514820823445916       0.0927005278645083       -0.00718624796718359
```

We can see that the transformation parameters tp have changed significantly. Looking at the residuals, we now recognize that they are much smaller than without using undulation values. Except point ID4, we are talking about cm-level now which is quite common over 10s of km as cadastral accuracy is about 2-3 cm and has a key aspect on adjacency preserving.

So whenever you use a transformation parameter set which you did not compute by yourself, be sure to know about its achievable accuracy.

Now we know about the transformation process from WGS84 to Bessel ellipsoid, so we can transform our point P2

```
P3=d3trafo(P2,tp)
```

```
P3 =      4192134.88018414      840309.91249954      4716910.9634969
```

Before we continue with mapping steps, let’s think about a further possible correction: When transforming new points, we could interpolate the residuals of the supporting points to preserve adjacency to neighboring points. Otherwise, for example, the transformation of a point very close to one of the supporting points used for determining the transformation

parameter set would get an additional discrepancy of the supporting point's residual.[12]
So we can use

```
P4=P3+rescorr(P3,ID_CART,tr)
```

```
P4 =      4192134.92157003      840309.928570152      4716910.96438803
```

which calculates the residual corrections and adds it to P3. There are different correction modes, which in principle are all based on a weighting of the supporting point residuals by their distance to the transformed point, we used the standard mode here. Note that residual corrections may be not suitable if there are clusters of supporting points or the correction needs to be an extrapolation rather than an interpolation (supporting points do not surround transformed points).
Now we convert to geographic coordinates again

```
P5=cart2ell(P4,'besseldhdn')
```

```
P5 =      11.3346755657941      48.000900584251      600.057858406566
```

It can be seen that differences in longitude and latitude are several seconds between the two reference ellipsoids.

Last step is the projection to GK again; here we also need to know about the undulation value at our measurement position (+46.146m):

```
P6=ell2tm(P5,'gk',[],46.146)
```

```
P6 =     4450356.94262121      5318199.56186201      553.911858406566
```

Note that the input of the central meridian for the TM projection is left empty so that the function detects it automatically. We're done!

Now, what if we had no undulation information at all? Then we would need to work with the transformation parameters we computed in case a), and the following results would be:

```
P3 =     4192104.41389327      840303.80636562       4716876.4614353
P4 =     4192104.54214521      840303.839144725      4716876.55856218
P5 =       11.334675567171     48.0009006213383      553.757195455022
P6 =     4450356.94275954      5318199.56598437      553.757195455022
```

As the identical points lie around our measurement, we see only a minor difference in 2D (few mm), but of course a bigger one in height (16 cm) which is due to undulation variances in the measurement area. If we hadn't used residual corrections, the height difference would have been even worse (~ 28 cm).

[12] Whether to apply residual corrections or not depends on what we are intended to do: If we have a highly consistent source system and want to preserve relations between the points after transformation to a somewhat distorted target system, we will not correct for residuals (e.g. construction points of a building project are transformed into cadastral system). If we want to transform our source points in a way that the resulting points match the neighboring points, we have to obey the residuals (e.g. property line points are mapped to the cadastral system – it is important that they match surrounding land parcels).

Apart from TM projection, there are also functions for the Lambert conical projection using two standard parallels (e.g. used in France, Belgium, New Zealand): `lambertcc2ell.m` and `ell2lambertcc.m`. Its use is basically identical to the TM mapping functions.

3. ***Other possibilities to do the transformation***

In some cases the (individual) 3D datum transformation cannot be used, because e.g. there are not enough identical points, height information is worse or the mapping projection is not known. In those cases, there are other possibilities to calculate transformations from GNSS data to a given local mapping system. Let's have a look on some of them. In this chapter no numerical examples will be given, but it only uses functions which are already mentioned before[13].
Some of the listed transformation types below separate position and height transformation. In that case identical points for position and height may be different (no need for identical 3D points). As a major advantage, bad height information does not affect position.

1. 2-step transformation

Mapping projection and ellipsoids need to be known and a parameter set for a course 3D pre-transformation must be given (e.g. some state-wide parameter set given by the cadastral office). To use the transformation, perform the following steps:

1. Coarse 3D pre-transformation using `d3trafo` (WGS84 -> local ellipsoid)
   Instead of a pre-transformation, one can also use a simple gravity point translation if one or more ID points are given.
2. `cart2ell`
3. `ell2tm` (gives approximate mapped coordinates)
4. `d2trafo`, probably using rescorr (position)
5. `d1trafo`, probably using rescorr (height)

2. 1-step transformation

No mapping projection or ellipsoids need to be known, but the allowed areal extent of this transformation is lower than for the 2-step method (some km^2). To use the transformation, perform the following steps:

1. `cart2ell`
2. `ell2tm` (gives approximate mapped coordinates)
3. `d2trafo`, probably using rescorr (position)
4. `d1trafo`, probably using rescorr (height)

[13] All similarity transformation functions, d1trafo, d2trafo and d3trafo, are handled the same way. They just differ in the number of transformation parameters necessary for 1D, 2D and 3D transformation.

3. Standard parameter transformation

When there are no identical points at all, one may use standard transformation parameter sets which are available for nearly every datum transformation. Obviously, there are parameter sets from (any) local datum to global datums like WGS84 and vice versa, but in many cases there are also transformation sets from local to local datums, e.g. at neighboring countries. Usually the parameters to use can be found in the internet at the cadastral offices' sites.
Not all of them are 7-parameter sets, though. In many cases there is only a shift in X,Y,Z to be applied – when using it with `d3trafo`, just set the rotations to be 0 and the scaling to be 1. You may store standard parameter sets in `Transformations.mat`. Please be aware, that when using standard transformation parameters, accuracy usually is very much lower than with the individual use of ID points (may be 1 – 25 meters or even worse). If you're lucky, you'll find approximate accuracy information where you find the parameter set.

4. Molodensky transformation

The Molodensky transformation also may be applied when there are no identical points. It makes use of just a translation from one ellipsoid to another, while rotation and scale factor are not regarded. As additional parameters the difference of the ellipsoids' semi-major axes and flattening can be used making it a simple 5 parameter transformation. In the toolbox you may use the `molodenskytrafo.m` function. Parameter sets may be stored in `Transformations.mat`.

With all these transformation types, of course, quality becomes lower with rising extent of the area to be transformed in.

4. ***Transformation to and from UTM coordinates***

In recent time, many state offices begin to transform their individual, historically grown coordinate systems to UTM mapping systems, mainly for the need of cross-border projects and international harmonization. Reference ellipsoid often is GRS80 / WGS84, which are nearly identical, and a specified reference frame defines the coordinate system[14].

Therefore it often is necessary to transform global coordinates (GNSS measurements) to UTM mapping. GNSS measurements are linked to the ITRS system, so we do not need to

[14] Just to define a reference ellipsoid and its origin and axes directions is not yet sufficient for coordinate calculations. Additionally it is necessary to have the coordinate *frame* which is the real, physical realization of the defined *system*, i.e. physical points on earth surface which can be assigned with unique coordinate values. For example, WGS84 coordinate system is a theoretically defined structure, while its realization is linked to ITRF which consists of several hundreds of measurement points all over the world which are computed with reference to the astronomical inertial system ICRS. Due to continental drifts, earthquakes etc. all those points move up to 10 cm per year (Eurasia for example is slightly rotating), so these coordinates define the *frame*. As variable coordinates are not acceptable for most geodetic purposes in cadastre or engineering, their systems apply to a special epoch of the ITRF and keep those coordinates fixed (e.g. European ETRS89 reference system and its frame ETRF89 are the fixed ITRS/ITRF system at epoch 1-1-1989. Of course, in the meantime Europe and ETRF89 have moved about 60 cm with respect to actual ITRF, but inside ETRF point relations remain constant).

perform any datum transformation when we want to use UTM/WGS84, we directly can start with the mapping[15]. This geodetic transformation toolbox offers two functions which both may be used for UTM mapping and unmapping.

Let’s consider the following WGS84 coordinates, given as [Long Lat]-matrix with western longitudes and southern latitudes showing negative sign.

City	Longitude	Latitude
New York	74° 00’ 00” W	40° 43’ 00” N
Stavanger	5° 44’ 00” E	58° 58’ 00” N
Adelaide	138° 35’ 00” E	34° 55’ 00” S
Buenos Aires	58° 25’ 00” W	34° 37’ 00” S
South Pole of Inaccessibility[16]	65° 47’ 00” W	85° 50’ 00” S

The first method to map them to UTM is the `ell2tm`-function we already have used in the previous example:

```
UTM=ell2tm(WGS84,'utm')
```

```
UTM =        18584461.2900142        4507785.97624875
             31657160.1434484        6539553.42854269
             54279229.0733876       -3866467.69241704
             21370122.1161769       -3831446.44949075
             20477430.1980497       -9533314.66085581
```

It is a standard TM mapping with 6°-width zones from pole to pole, as many other tools use it. The first two digits in easting indicate the zone, negative values in northing indicate southern hemisphere as it is not unambiguous to add the false northing of 10 000 000 which normally is used for the south. The results are not a standardized output, but as they just give a matrix with double numbers, they are easy to use in successive functions when you are working in a defined area.

Apart from that, this toolbox offers a special mapping function for standard UTM output:

```
UTM=ell2utm(WGS84,'wgs84')
```

```
UTM =    '18T 584461.290 4507785.976'
         '32V 312188.767 6540929.870'
         '54H 279229.073 6133532.307'
         '21H 370122.116 6168553.550'
         'A 1577930.603 2189833.100'
```

[15] Of course, if we wanted to map in any other reference system, we theoretically would have to. Again look at European ETRS89 – since its definition in 1989 the Eurasian plate has moved about 60cm compared to the ITRS, so GNSS measurements would be always mapped with that displacement. For practical purposes, a single absolute GNSS measurement only has an accuracy of a few meters, so that effect is irrelevant (at the moment). For higher accuracy up to cm- and mm-level (as needed for geodesy), we always have to work in Differential-GNSS-mode. That means, we do not measure absolute positions but 3D vectors related to some reference station. As this reference station already has coordinates given in ETRS89, the difference of ETRS – ITRS is cancelled out automatically.

[16] The point at the South Pole which has the biggest distance to any shippable ocean. It is hard to find a specific example point at the poles ;-)

The main difference in using the results programmatically is that they come as array of strings (or array of strings and doubles if we have heights). This is because every standard UTM coordinate contains of at least one letter which indicates the latitude zone.

Again the coordinates (easting) start with the longitude zone identifier. Note that Stavanger is located in zone 32 as 32V has irregular width to cover also Norway's west. `ell2utm` does recognize such irregularities while `ell2tm` does not due to its fixed mapping rules. Therefore the easting result of Stavanger is different in both functions. The next (the letter) is the latitude zone. This also indicates the polar regions (A, B and Y, Z), which by definition are not mapped with transverse mercator projection, but with universal polar stereographic projection instead. Naturally this change of projection method is not obeyed with `ell2tm`, but it is in `ell2utm`. The coordinates of the South Pole point therefore are completely different. Additionally, using the letter scheme it is clear whether the point is on northern or southern hemisphere so a false northing can be applied to prevent the northing from becoming negative. Apart from the mentioned characteristics the results are identical.

Performing the unmapping, we need to use

```
WGS=tm2ell(UTM,'utm')
```

or

```
WGS =utm2ell(UTM2,'wgs84')
```

depending on the input type. Both produce the right results if fed with the right data. If you need to transform UTM coordinates to ellipsoidal, please be aware about standard and non-standard UTM notation and behavior in irregular zones if present.

5. ***NTv2 projections***

NTv2 (National Transformation Version 2) is an approach to transform large amounts of coordinate data from one coordinate system to another when both systems do not differ very much (i.e. both systems may merely be transformed using a small translation and probably small rotation and scale values). It was initially used to transform non-geocentric ellipsoidal data to their geocentric equivalents[17].

A NTv2 transformation set consists of a regular grid in [Long, Lat] with given shift values [dLong, dLat] at all of the grid points. For each point to transform, the appropriate shift values are interpolated bi-linear using the four grid points defining the specific mesh. This allows replacing the datum transformation we learned in chapter 2 and, depending on the grid mesh size, also considers inconsistencies in the coordinate frames.

A single NTv2 file may contain several subgrids, e.g. for urban areas. Of course always the grid with the highest precision (i.e. smallest grid mesh size) is to be used.

[17] NTv2 was developed by Canada's Geodetic Survey Division as a national high precision transformation from NAD27 to NAD83 datum. Later it was adopted from Australia and other countries.

Let's look at an example of a NTv2 transformation in Germany using the BeTA2007 transformation set[18]. We use the `readntv2`-function to import the data into Matlab:

```
[long, lat, gridlat, gridlong, aclat, aclong, header, info] =
readntv2([your path]\BETA2007.gsa');
```

The main information about each subgrid can be found in the header-variable:

```
header =    SUB_NAME: 'DHDN90'
            PARENT: 'NONE'
            CREATED: '06-11-09'
            UPDATED: '06-11-09'
            S_LAT: 169200
            N_LAT: 199080
            E_LONG: 56400
            W_LONG: 19800
            LAT_INC: 360
            LONG_INC: 600
            GS_COUNT: 5208
```

Please note that, as standard, readntv2 returns western longitudes and shift values with negative sign while the NTv2 definition and all NTv2 files use positive sign[19]. This could be overridden by an input parameter. Latitude and Longitude limits are given in seconds.

Again we will use the five points ID1-ID5 from chapter 2 to be transformed. As NTv2 transformation is defined on ellipsoidal coordinates, we unmap them as shown in chapter 2 to get

```
ID_ELL =    11.2108503729801    48.2013560143094
            11.5529036293617    48.1708116428913
            11.3277515252722    48.0276937195613
            11.0191756457828    47.8018381331175
            11.0570713190464    47.4772324075114
```

NTv2 is defined as 2D transformation only so we need to skip height information.
Then we can transform them to ETRS89 using the NTv2 shift transformation:

```
ID_ntv2 = ntv2trafo(ID_ELL,gridlat,gridlong,header)
```

[18] The parameter set BETA2007.gsa (ASCII-version) can be downloaded at http://crs.bkg.bund.de/crseu/crs/descrtrans/BeTA/de_dhdn2etrs_beta.php.
It is a Germany-wide transformation set with nominal sub-meter accuracy for DHDN (Bessel-ellipsoid) to ETRS89 (GRS80-ellipsoid). NTv2 files mostly are provided as binary (*.gsb) and ASCII (*.gsa) files. Readntv2 only is capable using ASCII files.

[19] I decided to do so to have all my functions consistent in sign convention: negative is western longitude and southern latitude. However, as NTv2 was primarily used in Canada, they didn't want to deal with the negative sign for the whole country, so the official standard is positive sign for western longitude. Normally this is not relevant for the user as ntv2trafo also changes sign convention, but when just readntv2 is used to read in the grid data, one needs to be aware of the actual definition.

```
ID_ntv2 =   11.2095229728126     48.2004303179663
            11.5515255989068     48.1698932312929
            11.3264095908553     48.0267900150388
            11.0178833771655     47.8009572076629
            11.0557777788745     47.4763912062344
```

Now we are on GRS80 ellipsoid and can map the coordinates e.g. to UTM. Let's compare the NTv2 results with the known coordinates of the ID points in ETRS89 (which are identical to the WGS84 coordinates). We can transform the Cartesian coordinates to ellipsoidal and then calculate the differences (in [seconds]):

```
diff =      0.00192425860845447     -6.50218055398e-005
            0.00167127576418125     -0.000860465081586881
            0.000388304984966226    0.000189302090802812
            0.00135538250702893     -0.001017075626919
            -0.00157860898113427    0.00219183598630934
```

Expressed in meters this would be

```
diff =      0.057727758253634       -0.001950654166194
            0.0501382729254374      -0.0258139524476064
            0.0116491495489868      0.00567906272408436
            0.040661475210868       -0.0305122688075699
            -0.047358269434028      0.0657550795892803
```

giving accuracy of a few cm. This is in the order of the residuals when deducing a transformation parameter set of the identical points (only using 2D neglecting height which has the major residual part usually).

6. ***ITRS / ETRS transformations***

The International Terrestrial reference System (ITRS), its realization frames (ITRF) and the European equivalent ETRS/ETRF are a very special chapter when it comes to transformations. ITRS is a global, geocentric reference system[20] co-rotating with the earth, determined with respect to quasi-fixed stellar objects and therefore allows specifying station (point) coordinates and their velocity components in an absolute sense. The relation between the International Celestial Reference Frame (ICRF) and the ITRF are sporadically determined by calculating the Earth Orientation Parameters (EOP) using different major geodetic measurement principles.

There are sets of 14 parameters which describe the transformations between different realizations (7 parameters of a simple Cartesian 3D transformation and 7 change rates of these parameters per year). This allows comparing and combining observations of different epochs (e.g. different years), although continental plates have moved in the meantime[21].

[20] See http://itrf.ensg.ign.fr/itrs_itrf.php
[21] The change rate parameters are a linear change of course. There are formulas to obtain higher orders, but that is very much in detail and rarely used.

To avoid subsequent changes of coordinates in reference stations due to crustal movements, the ETRS89 was defined by just using 23 ITRF points on the Eurasian plate within the epoch 1989.0. As these points are considered to be stable with reference to each other, the whole ETRS89 is moving in the ITRS according to the Eurasian plate. For each ITRF frame there is also a corresponding ETRF frame and sets of parameters (3 translations, 3 rotations change rates) to transform between frames.

ETRS89 is the new reference system of many European cadastral offices (see chapter 4).

For most users, ITRS/ETRS transformations are irrelevant in practical use. So when will one need it then?
Consider a set of GNSS measurements are given at time t_c [year] which is given with respect to WGS84 coordinate system. WGS84 is linked to the actual ITRF, so it differs from ETRF coordinates by the plate movements[22]. Let's use as a station the EPN central bureau close to the Royal Observatory of Belgium with observed Cartesian WGS84 coordinates [4027894.006 307045.600 4919474.910] in actual ITRF2008, measured in epoch 2011.90 (early December 2011).

First step will be to transform it to ITRF2000, which is the corresponding frame to ETRF destination:

```
itrstrafo([4027894.006 307045.600 4919474.910],'ITRF2008',2011.90,
'ITRF2000',2011.90)
```

```
ans =          4027894.01452193   307045.600193748   4919474.88935544
               0.00042            0.00012            -0.00141
```

Internally, this consists of two transformations, ITRF2008 → ITRF2005 → ITRF2000. Note that, as no point velocities were given, [0 0 0] was assumed and by applying change rates the point now has velocity information with respect to zero velocity in ITRF2008.

Next step is the transformation from ITRF to ETRF at the desired frame level:

```
itrstrafo(ans,'ITRF2000',2011.90,'ETRF2000',2011.90)
```

```
ans =          4027894.36314443   307045.252782388   4919474.62499541
               0.01329            -0.01727           -0.01085
```

Last step is to change the reference epoch by using the obtained velocity information:

```
itrstrafo(ans,'ETRF2000',2011.90,'ETRF2000',2005)
```

```
ans =          4027894.27145832   307045.371968243   4919474.69989131
               0.01329            -0.01727           -0.01085
```

[22] To obtain high precision, one needs one or more reference stations within the measurement area. Normally these reference stations are already given in ETRS coordinates, so the GNSS baselines also point to ETRS coordinates. For this example we assume that the reference is given in WGS84, which might e.g. be the case in a precise surveying campaign in geophysical context. Then the GNSS result also is in WGS84 which is linked to ITRS.

Now we could compare these GNSS measurements with others defined in ETRF2000 and measured in 2005 as they refer to the same frame and the same epoch.

The three steps above could also be combined in the single function call

```
itrstrafo([4027894.006 307045.600 4919474.910],'ITRF2008',2011.90,
'ETRF2000',2005)
```

with the same result[23].

7. ***A closer look on the ellipsoid and projection definitions***

Ellipsoids and projections are stored in the files `Ellipsoids.mat` and `Projections.mat`. They are meant to be extended, so please feel free to add any definition which suits your needs.

Ellipsoid definition is very easy; it contains structs with the fields

a	semimajor axis
b	semiminor axis
f	flattening

This is actually redundant as any of them can be calculated out of the other two.
Note that this is just the general ellipsoid definition, there is no information about its center (compared to earth's mass center or elsewhere) or any relation towards any coordinates.

In the projections file, different projection definitions can be stored. At the moment with only two projection types coded in function files one may use 'tm' or 'lambertcc2' type.

Projections are stored as structs where every parameter set has the fields

type	type of projection. The functions read out the type to check whether the right projection type had been chosen.
ellips	The ellipsoid which shall be used for projection. Of course this needs to be the ellipsoid the Lat, Long are given in. If another ellipsoid is to be used, a datum transformation has to be performed first (see chapter 2).

Then there are additional parameter fields depending on the projection type:

TM projection fields:

m0	The scaling factor of the central meridian, e.g. 0.9996 with UTM
rule_L0	The calculations rule to specify the central meridian L0 of the projection zone when only the zone identifier ID is given. This is needed when projected coordinates have to be transformed to ellipsoidal coordinates (`tm2ell`). The ID is taken from the easting of the projected coordinate using ID_pro. For UTM the rule is 'ID*6-183' as every zone is 6° wide and central meridian of

[23] An online tool to perform ITRS/ETRS80 transformations can be found at the EUREF homepage: http://www.epncb.oma.be/_dataproducts/coord_trans/index.php

	the first zone is -177°. The field has to be entered as a string to be evaluated with Matlab and possibly to be combined with other terms before evaluating. ‘ID’ is used as a variable.
ID_ell	The rule to calculate the zone ID when ellipsoidal coordinates and the central meridian are given (`ell2tm`). E.g. for UTM the rule is ‘L0/6+30.5’. The field has to be entered as a string to be evaluated with Matlab and possibly to be combined with other terms before evaluating. ‘L0’ is used as a variable which is either an user input or detected automatically via standard_L0.
standard_L0	The rule to calculate the standard central meridian for each point to be transformed. standard_L0 is used to detect the proper L0 automatically when there is no user input in `ell2tm`. E.g. for UTM the rule is 'floor(L/6)*6+3'. This rule has currently a lack of definition exactly at 180° E as this meridian is already covered by 180° W. The field has to be entered as a string to be evaluated with Matlab and possibly to be combined with other terms before evaluating. ‘L’ may be used as a variable. It should be good practice to detect the central meridian automatically. Common exceptions only are measurement areas in two zones, when projected coordinates shall refer to the same zone.
ID_pro	The rule to calculate the zone ID when projected coordinates are given (`tm2ell`). Usually this derives from the easting of the coordinates. For UTM it is e.g. ‘floor(E/1e6)’: the effective coordinate always has an integer part of 6 places, everything added before is part of the ID. The field has to be entered as a string to be evaluated with Matlab and possibly to be combined with other terms before evaluating. ‘E’ may be used as a variable.
rule_easting	The rule to add the false easting in [m]. This is a combination of false easting and zone ID addendum, so in UTM it is ‘+5e5+ID*1e6’. 500 000 is the “real” false easting and then the zone ID is added additionally. This is used both in `tm2ell` and `ell2tm`. The field has to be entered as a string to be evaluated with Matlab and possibly to be combined with other terms before evaluating (the initial ‘+’ therefore is mandatory).
rule_northing	The rule to calculate the false northing. In UTM there only is a false northing for points on southern hemisphere. In `tm2ell` and `ell2tm` these conditions cannot be indicated as results may be ambiguous then. So in the UTM implementation here no false northing is used: [<empty matrix>] or ‘<empty string>’ See chapter 4 for details.

Lambert conical projection with two standard parallels fields:

lat	The first and second standard parallels [1x2] in degrees Those parallels remain without scaling when projected.. In Austria for example there is an official projection parameter set given by the BEV which uses [46 49].
ORell	The origin (mapping center) of the projection in [Long Lat] [degrees]. The projection of ORell will be [0 0]. E.g. at the 'bev' set it is [13°20' 47°30'].
ORproj	The false easting and northing for the projected origin coordinate to prevent negative coordinates [false_easting false_northing] [meters]. The 'bev' values are [400000 400000].

8. ***The Transformations.mat definitions***

Transformation parameters can be stored in `Transformations.mat`.
They are just 1 x n vectors with vector name = transformation name.

All transformations have a different number of transformation parameters, so each transformation function checks the input parameter set for proper size. See the helptext in `Transformations.m` for details.

9. ***Working with file inputs and outputs***

For many of the files in this toolbox you may specify an ASCII-filename as string instead of the input matrix. In this case, the specified file is opened and the ASCII input is treated as if it was the input data.
This functionality is very basic – there is no "intelligence" to interpret the file data, it is just treated "as is". So it may just contain pure coordinate information, no point IDs, comments etc.

Also, many files have a "FileOut"-parameter in the input list. With this, you may specify a file to write the first output variable to.

It is planned to expand the file functionality in future, but it's not a major topic right now.

10. ***Future developments***

This toolbox will be expanded with additional functions and definitions in future. Depending on our lecture (and sometimes Matlab Central user suggestions) needs we will for sure add further projection types and tools.

For comments, bug reports and suggestions please contact

Dr. Peter Wasmeier
Chair of Geodesy, Technische Universität München
p.wasmeier@bv.tum.de
www.geo.bv.tum.de

April 20th, 2015