HOWTO g01: Synthesis at grids and at scattered points

You will learn how to perform a basic spherical harmonic synthesis at grids and scattered points. The scattered points will be loaded from a text file, a binary "mat" file and also will be taken from MATLAB variables.

Throughout the cookbook, we keep the same names of the input parameters to the GrafLab function. They are all explained in detail in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Synthesis at a grid

Define the GrafLab inputs parameters. Throughout the cookbook, we synthesize the disturbing potential as an example (see the "quantity" variable).

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 360;
ellipsoid         = 1;
GGM_path          = '../data/input/EGM96.mat';
crd               = 0;
point_type        = 0;  % Computation at a grid
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-g01-grd';
quantity_or_error = 0;
quantity          = 5;
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 0;
status_bar        = 1;
```

Do the synthesis.

```
tic
out_grd = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
```

```
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
time_grd = toc;
```

Now you may take a look at the files generated by GrafLab.

- The report file summarizes details of the synthesis (GGM model path, minimum and maximum harmonic degrees, computation time, etc.):

```
fprintf("""%s_Report.txt"".\n", out_path);
```

- Output numerical data in the text format (the structure of the file is always explained at the end of the report file):

```
fprintf("""%s.txt""\n", out_path);
```

- Output numerical data in the MATLAB binary format (the same structure as that of the text file):

```
fprintf("""%s.mat""\n", out_path);
```

Synthesis at scattered points from MATLAB variables

Let's take the points from the grid-wise synthesis and consider them as scattered points now. We have already did a synthesis at these points using the grid mode. Now, we repeat the same computation, but with the point-wise mode. Both results should therefore be the same.

At first, we have to define three arrays of spherical coordinates of the evaluation points. With the grid-wise computation, we had to define only the grid boundaries. With scattered points, we need to get coordinates of each individual grid point. So let's create the grid and transform it to arrays so that we can pretend the points are scattered. Note that you can pass any points to "lat_sctr", "lon_sctr" and "h_sctr", be them regularly or irregularly distributed. We use regularly sampled points to demonstrate the difference between the computational speed at grids and at scattered points.

```
[lon_sctr, lat_sctr] = meshgrid(lon_grd_min:lon_grd_step:lon_grd_max, ...
                                lat_grd_min:lat_grd_step:lat_grd_max);
lat_sctr = lat_sctr(:);
lon_sctr = lon_sctr(:);
h_sctr   = zeros(length(lat_sctr), 1);


% Update the GrafLab input parameters
point_type = 2;  % Evaluation points from MATLAB variables
out_path   = '../data/output/howto-g01-sctr-var';
```

Do the synthesis.

```
tic
out_sctr = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    lat_sctr, ...
    lon_sctr, ...
    h_sctr, ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
```

```
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
time_sctr = toc;
```

Did you notice how slow the synthesis was at scattered points, despite using the very same points as in the grid-wise synthesis? This is because the synthesis in grids takes advantage of an efficient FFT-based algorithm. This technique cannot be used, however, with scattered points. With increasing number of evaluation points and/or maximum degree of the synthesis, the point-wise computations are therefore slower.

```
fprintf("The grid-wise synthesis took %0.3f sec.\n", time_grd);
fprintf("The point-wise synthesis took %0.3f sec.\n", time_sctr);
```

Now let's compute the RMS of the differences between the synthesis at a grid and at scattered points. The value shoud be very small, say, "10^-12" or less.

```
fprintf("The RMS is: %0.16e\n", rms(out_grd(:, end) - out_sctr(:, end)));
```

```
fprintf("Now you may want to explore the ""%s*"" files.\n\n", out_path);
```

Synthesis at scattered points from a text file

A few variables need to be modified to perform the synthesis at scattered points. We will load the evaluation points from the "sctr_points_path" file (5 points).

```
point_type       = 1;  % Evaluation points from a text file
sctr_points_path = '../data/input/sctr-points.txt';
out_path         = '../data/output/howto-g01-sctr-load-txt';
```

Do the synthesis.

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    [], ...
    [], ...
    [], ...
    [], ...
```

```
    [], ...
    [], ...
    [], ...
    sctr_points_path, ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

```
fprintf("Now you may want to explore the ""%s*"" files.\n", out_path);
```

Synthesis at scattered points from a binary mat file

You can import scattered points also from a "mat" file. Let's save now the points from the text file "sctr_points_path" into a binary file and use the new binary file to load the points.

```
sctr_points = load(sctr_points_path);
sctr_points_path = '../data/output/sctr-points.mat';
save(sctr_points_path, 'sctr_points', '-v7.3');
```

Update the GrafLab input parameters.

```
out_path = '../data/output/howto-g01-sctr-load-mat';
```

Do the synthesis.

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    [], ...
    [], ...
    [], ...
```

```
    [], ...
    [], ...
    [], ...
    [], ...
    sctr_points_path, ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

```
fprintf("Now you may want to explore the ""%s*"" files.\n", out_path);
```

HOWTO g02: Gravitational field models

You will learn about the formats of gravitational field models that are supported by GrafLab.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Structure of a global geopotential model (GGM) file

The structure of the GGM coefficients table must follow either "Table 1" or "Table 2". Any other ordering scheme may not be processed correctly by GrafLab, so is not recommended.

Table 1: GGM coefficients table up to degree 2. The order of the columns is harmonic degree, harmonic order, Cnm and Snm

```
0   0    1.00000E+00    0.00000E+00
1   0    0.00000E+00    0.00000E+00
1   1    0.00000E+00    0.00000E+00
2   0   -0.48417E-03    0.00000E+00
2   1   -0.20662E-09    0.13844E-08
2   2    0.24394E-05   -0.14003E-05
```

Table 2: GGM coefficients table up to degree 2. The order of the columns is harmonic degree, harmonic order, Cnm and Snm

```
0   0    1.00000E+00    0.00000E+00
1   0    0.00000E+00    0.00000E+00
2   0   -0.48417E-03    0.00000E+00
1   1    0.00000E+00    0.00000E+00
2   1   -0.20662E-09    0.13844E-08
2   2    0.24394E-05   -0.14003E-05
```

GGM as a MATLAB's binary file

The "mat" file must store one variable only (there is an exception to be explained in "HOWTO g11"). The variable must be a matrix with the structure as shown in "Table 1" or "Table 2" above. An example of a valid MATLAB binary file format of a GGM can be found in "../data/input/EGM96.mat".

Let's define the GrafLab input parameters.

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 'nmaxGGM';
ellipsoid         = 1;
GGM_mat           = '../data/input/EGM96.mat';  % This is the MATLAB binary
                                                % file with GGM coefficients
crd               = 0;
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      =   1.0;
```

```
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-g02-table-mat';
quantity_or_error = 0;
quantity          = 5;
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 0;
status_bar        = 1;
```

Do the synthesis

```
out_mat = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_mat, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

GGM in a text format

This example shows how to import a text format of GGM that obeys the structure of "Table 1" or "Table 2". To this end, we save the "GGM_mat" file to a text file. Next, we update some GrafLab input parameters.

```
GGM = load(GGM_mat);
GGM = GGM.EGM96;
GGM_txt  = '../data/output/EGM96.txt';  % This will be the text version of
                                        % "GGM_mat"
save(GGM_txt, 'GGM', '-ascii', '-double');
out_path = sprintf('../data/output/howto-g02-table-txt');
```

Do the synthesis

```
out_txt = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_txt, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

Now check the synthesis from the MATLAB's binary file and from the text file.

```
fprintf("The RMS of the difference is %0.16e\n", rms(out_mat(:, end) - ...
                                                    out_txt(:, end)));
```

GGM in the GFC format

The "gfc" format is defined by ICGEM (http://icgem.gfz-potsdam.de/ICGEM-Format-2011.pdf).

GrafLab should be able to process most of the *static* models found on the ICGEM website. It *cannot* process temporal models. The temporal models can easily be identified, as they use terms such as "gfct", "trnd", "acos", "asin", etc. If you attempt import a temporal model, you should get an error message.

Whenever you import a "gfc" file, GrafLab reads its header and takes the "GM" and "R" constants from that file. The "GM" and "R" values that we pass to GrafLab are *not* used in case of "gfc" files, so we can use any positive real number, it simply does not matter. GrafLab will print a warning if your "GM" and/or "R" values are different from those found in the "gfc" file. You may verify that GrafLab took correctly the "GM" and "R" values from the "gfc" file by inspecting the report file exported by GrafLab.

In this example, we import some more or less random static model downloaded from the ICGEM website.

Now we need to define the GrafLab input parameters. We intentionally set "GM" and "R" to a wrong value "1.0" to demonstrate that GrafLab will ignore our choice and will use the correct values from the "gfc" file.

```
GM       = 1.0;
R        = 1.0;
GGM_gfc  = '../data/input/GO_CONS_GCF_2_TIM_R6.gfc';  % This is the "gfc" file
out_path = '../data/output/howto-g02-gfc';
```

Do the synthesis

```
out_txt = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_gfc, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
```

```
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

You may now want to inspect the report file to see that GrafLab indeed used the correct "GM" and "R" values from the "gfc" file.

```
fprintf("The name of the report file is ""%s_Report.txt"".\n", out_path);
```

HOWTO g03: Minimum and maximum harmonic degree of the synthesis

You will learn how to modify the minimum and the maximum harmonic degrees of the synthesis.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Synthesis from degree 0 up to degree 10

Define the GrafLab input parameters.

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;  % The minimum degree of the synthesis
nmax              = 10; % The maximum degree of the synthesis
ellipsoid         = 1;
GGM_path          = '../data/input/EGM96.mat';
crd               = 0;
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      =   1.0;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = sprintf('../data/output/howto-g03-nmin%d-nmax%d', ...
                            nmin, nmax);
quantity_or_error = 0;
quantity          = 5;
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 0;
status_bar        = 1;
```

Do the synthesis

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
```

```
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

Synthesis up to the maximum degree of a GGM

The maximum harmonic degree in "GGM_path" is 360. To synthesize up to this degree, you can manually set "nmax" to 360 or, even better, to 'nmaxGGM'. In the latter case, GrafLab scans the file for its maximum harmonic degree and automatically uses the maximum value it founds. This is useful when dealing with multiple GGM files with varying maximum harmonic degree, e.g., monthly gravity field solutions.

Update the GrafLab input parameters.

```
nmax     = 'nmaxGGM';  % Uses automatically the maximum degree of GGM
out_path = sprintf('../data/output/howto-g03-nmin%d-nmaxGGM', nmin);
```

Do the synthesis

```
out_grd = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
```

```
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

Synthesis up to maximum degree smaller than 2

GrafLab refuses to do a synthesis up to maximum degree that is smaller than 2. This is due to some unwise decisions made in the early months of the GrafLab development. Hopefully, this will be fixed one day. Until then, if you need "nmax" smaller than 2, there is a workaround:

- prepare a GGM file with coefficients up to degree at least 2,
- set all coefficients beyond your "nmax" (which is either 0 or 1) to zero, and
- do the synthesis at least up to "nmax = 2".

If is **not** recommended to do this kind of a trick

- for any quantity that involves the normal gravity field (the coefficient "C20_ell" of the normal field would be incorrectly subtracted), and
- for the gravitational and distubring tensor in the local north-oriented reference frame.

Synthesis with "nmin" larger than zero

In addition to modifying "nmax", you may also change the "nmin" value. "nmin" represents the minimum degree of the harmonic synthesis. For some gravity field quantities, GrafLab does not, however, allow non-zero "nmin" value. The quantities includes: 9, 10, 15, 20, 23 (see the code numbers for "quantity" from ../docs/graflab.md). If you attempt to evaluate these quantities with "nmin > 0", you will get an error. If you set "nmin" to a value larger than "nmax", you will get an error, too.

We simply increase "nmin" and modify the name of the output files.

```
nmin     = 100;  % Increase the minimum degree of the synthesis
out_path = sprintf('../data/output/howto-g03-nmin%d-nmaxGGM', nmin);
```

Do the synthesis.

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

HOWTO g04: Methods to compute the Legendre functions

You will learn about the pros and cons of the three supported methods to compute Legendre functions.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Brief summary

The standard forward column method (SFCM, "fnALFs = 1") can be used up to degree 1800.

The modified forward column method (MFCM, "fnALFs = 2") can be used up to degree 2700.

The extended range arithmetic approach (ERA, "fnALFs = 3") can be used up to almost an arbitrary degree.

Let's do some grid-wise and point-wise syntheses with all three methods to see which one should be preferred in case more than one method is stable within your range of harmonic degrees.

Benchmarks for grid-wise computations

Define all GrafLab input parameters except the method to compute the Legendre functions.

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 360;
ellipsoid         = 1;
GGM_path          = '../data/input/EGM96.mat';
crd               = 0;
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   0.1;
lat_grd_max       =  90.0;
lon_grd_min       =   0.1;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-g04-grd';
quantity_or_error = 0;
quantity          = 5;
export_data_txt   = 0;
export_report     = 0;
export_data_mat   = 0;
display_data      = 0;
status_bar        = 1;
```

Do the tests by looping over the methods to compute Legendre functions. Note that the same number of points is use for all grid computations.

```
time_grd = zeros(3, 1);
for fnALFs = [1, 2, 3]
```

```
    fprintf("fnALFs method: %d\n", fnALFs);

    tic
    out_grd = GrafLab('OK', ...
        GM, ...
        R, ...
        nmin, ...
        nmax, ...
        ellipsoid, ...
        GGM_path, ...
        crd, ...
        point_type, ...
        lat_grd_min, ...
        lat_grd_step, ...
        lat_grd_max, ...
        lon_grd_min, ...
        lon_grd_step, ...
        lon_grd_max, ...
        h_grd, ...
        [], ...
        [], ...
        [], ...
        [], ...
        out_path, ...
        quantity_or_error, ...
        quantity, ...
        fnALFs, ...
        [], ...
        export_data_txt, ...
        export_report, ...
        export_data_mat, ...
        display_data, ...
        [], ...
        [], ...
        [], ...
        [], ...
        status_bar);
    time_grd(fnALFs) = toc;

end
```

Benchmarks for point-wise computations

Now let's define the scattered points

```
[lon_sctr, lat_sctr] = meshgrid(0.0:5.0:360.0, ...
                                -90.0:5.0:90.0);
lat_sctr = lat_sctr(:);
lon_sctr = lon_sctr(:);
h_sctr   = zeros(length(lat_sctr), 1);
```

Update some GrafLab input parameters.

```
point_type = 2;  % Synthesis at scattered points defined by MATLAB variables
out_path   = '../data/output/howto-g04-sctr';
```

Do the tests by looping over the methods to compute Legendre functions. Again, the same number of scattered points is use for all point-wise computations.

```
time_sctr = zeros(3, 1);
for fnALFs = [1, 2, 3]

    fprintf("fnALFs method: %d\n", fnALFs);


    % Do the synthesis
    tic
    out_grd = GrafLab('OK', ...
        GM, ...
        R, ...
        nmin, ...
        nmax, ...
        ellipsoid, ...
        GGM_path, ...
        crd, ...
        point_type, ...
        [], ...
        [], ...
        [], ...
        [], ...
        [], ...
        [], ...
        [], ...
        [], ...
        lat_sctr, ...
        lon_sctr, ...
        h_sctr, ...
        out_path, ...
        quantity_or_error, ...
        quantity, ...
        fnALFs, ...
        [], ...
        export_data_txt, ...
        export_report, ...
        export_data_mat, ...
        display_data, ...
        [], ...
        [], ...
        [], ...
        [], ...
        status_bar);
    time_sctr(fnALFs) = toc;

end
```

Results of the benchmarks

Grid-wise computation times in sec:

```
fprintf("SFCM: %0.1f\n", time_grd(1));
fprintf("MFCM: %0.1f\n", time_grd(2));
fprintf(" ERA: %0.1f\n", time_grd(3));
```

Point-wise computation times in sec:

```
fprintf("SFCM: %0.1f\n", time_sctr(1));
fprintf("MFCM: %0.1f\n", time_sctr(2));
fprintf(" ERA: %0.1f\n", time_sctr(3));
```

Conclusions

For grid computations up to degree 1800, use SFCM. Beyond that degree, use ERA. Avoid using MFCM.

For point-wise synthesis up to degree 1800, use SFCM or MFCM. For maximum degrees from 1801 to 2700, use MFCM. Beyond degree 2700, use ERA.

HOWTO g05: Plotting in GrafLab and the output variable

You will learn how to plot the results of the synthesis with GrafLab and about the GrafLab output variable.

Only synthesis at a grid can be plotted with GrafLab.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Plotting with "Mapping Toolbox"

Define the GrafLab input parameters.

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 360;
ellipsoid         = 1;
GGM_path          = '../data/input/EGM96.mat';
crd               = 0;
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-g05-mapping-toolbox';
quantity_or_error = 0;
quantity          = 5;
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 1;  % Use Mapping Toolbox to plot the synthesis
graphic_format    = 6;
colormap          = 1;
number_of_colors  = 60;
dpi               = 300;
status_bar        = 1;
```

Do the synthesis

```
tic
out_mt = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
```

```
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
time_mt = toc;
```

You may now open the following files to see the maps:

```
fprintf("""%s*.png"".\n", out_path);
```

The time needed to perform the synthesis and plot the results is:

```
fprintf("%0.1f sec.\n", time_mt);
```

Conclution: the plots are nice, but the plotting may be very slow for large grids.

Plotting with the "imagesc" function

Update some of the GrafLab input paramaters.

```
display_data = 2;  % Use the "imagesc" function to plot the synthesis
out_path     = '../data/output/howto-g05-imagesc';
```

Do the synthesis

```
tic;
out_imgsc = GrafLab('OK', ...
    GM, ...
```

```
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
time_imagsc = toc;
```

You may now open the following files to see the maps:

```
fprintf("""%s*.png"".\n", out_path);
```

The time needed to perform the synthesis and plot the results is (compare the time with the Mapping Toolbox):

```
fprintf("%0.1f sec.\n", time_imagsc);
```

Conclusion: very simple plots, but also very fast plotting, even with large grids.

Output variable from GrafLab

You may redirect the numerical outputs from GrafLab to a MATLAB variable. The output variable stores the numerical data used by GrafLab to prepare the "txt" and/or "mat" file(s). Note that the "out_path" must be specified, even if you do not export any data or plot.

Now, you may want to explore the GrafLab numerical outputs stored in the "out_mt" and "out_imgsc" variables.

HOWTO g06: Synthesis of planetary topographies

You will learn how to synthesize a planetary topography. In fact, the same approach can be applied to any surface spherical harmonic synthesis of the form

$$f(\varphi,\lambda)=\sum_{n=0}^{n_{\max}}\sum_{m=0}^{n}\left(\bar{C}_{nm}\cos(m\lambda)+\bar{S}_{nm}\sin(m\lambda)\right)\bar{P}_{nm}(\sin\varphi),$$

where $\bar{C}_{nm}$ and $\bar{S}_{nm}$ are 4π-fully-normalized (real) surface spherical harmonic coefficients of the function f, n and m are spherical harmonic degree and order, respectively, $\bar{P}_{nm}(\sin\varphi)$ are the 4π-fully-normalized (real) associated Legendre functions of the first-kind, and, finally, φ and λ are the spherical latitude and longitude, respectively. This means GrafLab can synthesize a wide range of (real) functions given on a sphere.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Synthesis of the Earth's topography in spherical coordinates

We need to perform the basic *surface* spherical harmonic synthesis shown above. This can be achieved with the *surface* synthesis of the gravitational potential, see the equation for "V" in https://github.com/blazej-bucha/graflab/blob/master/docs/Definition_of_functionals_of_the_geopotential_used_in_GrafLab_software.pdf.

The trick is that we have to set "GM = 1.0", "R = 1.0" and the radius of the evaluation points to "r = 1.0". Obviously, we have to do the synthesis on the unit sphere, so "crd = 1". Finally, we set "quantity" to "11" (see ../docs/graflab.md).

Define the GrafLab inputs.

```
GM                = 1.0;  % Important
R                 = 1.0;  % Important
nmin              = 0;
nmax              = 360;
ellipsoid         = 1;
GGM_path          = '../data/input/DTM2006.mat';
crd               = 1;  % Important
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0; % Note that the synthesis is here done at a grid,
                           % so "h_grd" needs to be set to a height above the
                           % sphere with the radius "R", hence "0.0" (see
                           % <../docs/graflab.md ../docs/graflab.md>).  In this
                           % way, the radius of the evaluation points "r" will
                           % be "1.0".  If you do the synthesis at scattered
                           % points, you should set "h_sctr" to "1.0".
out_path          = '../data/output/howto-g06-topography-sph-coord';
```

```
quantity_or_error = 0;
quantity          = 11;  % Gravitational potential; in this case, however,
                         % we synthesize the Earth's topography
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 2;
graphic_format    = 6;
colormap          = 1;
number_of_colors  = 60;
dpi               = 300;
status_bar        = 1;
```

Do the synthesis

```
out = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
```

You may now take a look at the output files.

```
fprintf("The ""%s*_Gravitational_potential.png"" file shows the " + ...
        "synthesized topography.\n", out_path);
```

Note that GrafLab thinks it computed the gravitational potential. It has no idea (how could it?) that we actually computed the Earth's topography. This is why the plot, the report file, the file name, etc. still report the gravitational potential.

Synthesis of the Earth's topography in ellipsoidal coordinates

Now what if you want to synthesize, say, the Earth's topography, but your evaluation points are given in ellipsoidal coordinates rather than in spherical coordinates? In this case, you **cannot** simply set "crd = 0" and use zero ellipsoidal heights. This is because this causes the evaluation points to reside on the ellipsoid, hence the points have (in general) a radius that is not equal to "1.0" (unit sphere). In such cases, GrafLab performs **solid** spherical harmonic synthesis which is not desired here.

We have to fool GrafLab somehow. More specifically, we have to transform the ellipsoidal latitudes of the computation points to their spherical counterpart assuming zero ellipsoidal heights. The spherical coordinates can then be entered to GrafLab similarly as in the previous example.

Let's define grid boundaries in **ellipsoidal** coordinates. Here, we assume the coordinates refer to GRS80. Note that since there is no difference between ellipsoidal and spherical longitudes for biaxial ellipsoids, we use the longitudes from the previous example.

```
% Vector of ellipsoidal latitudes
lat_ell = -90.0:1.0:90.0;

% The first eccentricity of GRS80
eEl = sqrt(0.006694380022903416);

% Now let's transform the ellipsoidal latitudes "lat_ell" to spherical
% latitudes "lat_sph".  The formula holds for points lying on the reference
% ellipsoid only.
lat_sph = atan(tan(lat_ell * pi / 180.0) * sqrt(1.0 - eEl^2)) * 180.0 / pi;
```

Note that the "lat_sph" vector does not have an equal spacing. The grid latitudes must be therefore entered in a special way to GrafLab: set the minimum grid latitude "lat_grd_min" to the "lat_sph" vector and then set both the latitudinal grid step "lat_grd_step" and the maximum grid latitude "lat_grd_max" to "'empty'" (see the GrafLab input parameters in ../docs/graflab.md).

```
crd              = 1;  % Important
lat_grd_min      = lat_sph;
lat_grd_step     = 'empty';
lat_grd_max      = 'empty';
h_grd            =   0.0; % We are still on the unit sphere (see above)
out_path         = '../data/output/howto-g06-topography-ell-cord';
```

Do the synthesis

```
out = GrafLab('OK', ...
```

```
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
```

You may now take a look at the output files.

```
fprintf("The ""%s*_Gravitational_potential.png"" file shows the " + ...
        "synthesized topography.\n", out_path);
```

HOWTO g07: Stop the rotation to get gravitational quantities

You will learn how to stop the Earth's rotation in order to compute, for instance, the *gravitational* vector instead of the *gravity* vector (no centrifugal force).

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Synthesis of the gravitational vector (no centrifugal force)

At first, let's do the synthesis of the gravitational vector in *spherical* coordinates ("crd = 1").

To stop the Earth's rotation, you have to set the angular velocity of the ellipsoid to zero (see the "ellipsoid" variable in ../docs/graflab.md). This is because, GGMs do not have their own value of the angular velocity, so it is usually taken from the definition parameters of the reference ellipsoid. In fact, all elements of the "ellipsoid" array can safely be set to zero as long as you work with spherical coordinates of evaluation points.

Define the GrafLab input parameters

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 360;
ellipsoid         = [0.0 0.0 0.0 0.0 0.0];  % Make all parameters of the
                                            % reference ellipsoid zero.
GGM_path          = '../data/input/EGM96.mat';
crd               = 1;  % Spherical coordinates
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-g07-gravitational-vector-sph-crd';
quantity_or_error = 0;
quantity          = 16;  % Gravity vector; in this case, however,
                         % gravitational vector
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 2;
graphic_format    = 6;
colormap          = 1;
number_of_colors  = 60;
dpi               = 300;
status_bar        = 1;
```

Do the synthesis

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
```

Now, let's do the synthesis of the gravitational vector in *ellipsoidal* coordinates ("crd = 0").

Again, you have to set the angular velocity of the ellipsoid to zero. This time, however, two elements of the "ellipsoid" array are needed, the semimajor axis and the numerical eccentricity. This is because the ellipsoidal coordinates of the evaluation points have to be transformed into spherical coordinates. The latter coordinates are required for spherical harmonic synthesis. To do the coordinates transformation, you need set the semimajor axis and the numerical eccentricity to values of the ellipsoid you want to use (e.g., GRS80, WGS84 and so on).

Define the GrafLab input parameters.

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 360;
ellipsoid         = [0.0 6378137.0 sqrt(0.006694380022903416) 0.0 0.0];  % Note
                    % the semimajor axis of the numerical eccentricity of the
                    % reference ellipsoid.  Here, we are using GRS80.
```

```
GGM_path          = '../data/input/EGM96.mat';
crd               = 0;  % Ellipsoidal coordinates
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto07-gravitational-vector-ell-crd';
quantity_or_error = 0;
quantity          = 16;  % Gravity vector; in this case, however,
                         % gravitational vector
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 2;
graphic_format    = 6;
colormap          = 1;
number_of_colors  = 60;
dpi               = 300;
status_bar        = 1;
```

Do the synthesis

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
```

```
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
```

The same comment as from the very end of HOWTO NO. 6 applies here, too.

HOWTO g08: Grids with varying spacings in latitudes and longitudes

You will learn how to do the efficient grid-wise synthesis, but this time with varying grid step in latitude and/or longitude.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Varying grid spacing in latitude

Let's assume we want to synthesize the disturbing potential at a grid residing on the GRS80 reference ellipsoid. The tricky part is that we need a constant sampling in *spherical* latitude. In terms of the *ellipsoidal* latitude, the spacing therefore varies. Fortunately, GrafLab makes it possible to define grids with varying spacing in latitude and longitude.

Suppose that our grid resides on GRS80, but is defined by a vector of spherical latitudes "lat_sph" with a constant spacing. Next, we have a vector of spherical longitudes "lon" and the constant height of the grid above the reference ellipsoid "h". The spherical latitudes "lat_sph" can be transformed into ellipsoidal ones "lat_ell", and these ellipsoidal latitudes (with varying spacing) can then be used to define a grid in GrafLab.

```
% Vector of spherical latitudes
lat_sph = -90.0:1.0:90.0;

% Vector of longitudes
lon = 0.0:1.0:360.0;

% Constant ellipsoidal height
h = 0.0;

% The first eccentricity of GRS80
eEl = sqrt(0.006694380022903416);

% Now let's transform the spherical latitudes "lat_sph" to ellipsoidal
% latitudes.  The formula holds for points lying on the reference ellipsoid
% only.
lat_ell = atan(tan(lat_sph * pi / 180.0) ./ sqrt(1.0 - eEl^2)) * 180.0 / pi;
```

The spherical and the ellipsoidal longitudes are equal, so no transformation is required for grid longitudes. Now define the GrafLab input parameters.

```
GM               = 3986004.415E+8;
R                = 6378136.3;
nmin             = 0;
nmax             = 360;
ellipsoid        = 1;
GGM_path         = '../data/input/EGM96.mat';
crd              = 0;  % Ellipsoidal coordinates
point_type       = 0;
lat_grd_min      = lat_ell; % Our column vector of ellipsoidal latitudes
```

```
lat_grd_step      = 'empty'; % Required if "lat_grd_min" is an array
lat_grd_max       = 'empty'; % Required if "lat_grd_min" is an array
lon_grd_min       = lon;     % Our column vector of ellipsoidal longitudes
lon_grd_step      = 'empty'; % Required if "lon_grd_min" is an array
lon_grd_max       = 'empty'; % Required if "lon_grd_min" is an array
h_grd             = h;
out_path          = '../data/output/howto-g08-varying-grd-latitude';
quantity_or_error = 0;
quantity          = 5;  % Disturbing potential
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 2;
graphic_format    = 6;
colormap          = 1;
number_of_colors  = 60;
dpi               = 300;
status_bar        = 1;
```

Do the synthesis

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
```

```
    number_of_colors, ...
    dpi, ...
    status_bar);
```

In the very same way, you can also work with varying longitudinal step. The spacing may also vary if the coordinates of the evaluation points that enter GrafLab are spherical ("crd = 0"). It doesn't matter.

HOWTO g09: Exploit the symmetry of Legendre functions

You will learn about the conditions that must be satisfied to take advantage of the symmetry property of Legendre functions.

Legendre functions are symmetric with respect to the equator as follows,

$$\overline{P}_{nm}(\sin\varphi) = (-1)^{(n+m)}\,\overline{P}_{nm}(\sin(-\varphi)).$$

Therefore, if the grid contains both the positive and the negative latitudes, φ and $-\varphi$, respectively, the Legendre function for one of the two needs to be computed only. The other is obtained efficiently by the symmetry property.

In GrafLab, *all* of the following conditions must be satisfied to employ the symmetry of Legendre functions.

- A functional of the geopotential is selected (the symmetry property is not implemented for commission errors).
- The grid-wise computation mode is selected.
- The extended-range arithmetic approach is selected to compute Legendre functions (the whole improvement is tailored to high harmonic degrees, for which the standard and modified forward column approaches do not provide accurate results anyway).
- All positive latitudes have their negative counterpart or vice versa (up to a given threshold to suppress numerical inaccuracies, currently "100.0 * eps" degrees). The zero latitude, i.e., the equator, may be included in the grid (again, within the "100.0 * eps" deg numerical threshold).

If all these conditions are satisfied, the symmetry property is employed *automatically*, meaning that no additional action from the user is required to enable the more efficient variant.

Examples of some symmetric grids (shown are only the latitudes):

- Equator included

```
lat = [-90 -60 -30 0 30 60 90]
```

- Equator included

```
lat = [-80 -60 -40 -20 0 20 40 60 80]
```

- Equator excluded

```
lat = [-35 -25 -15 -5 5 15 25 35]
```

```
lat = [-90 -85 -80 80 85 90]
```

- Varying spacing

```
lat = [-90 -80 -75 -70 -69 -68 -67 -66 -65 65 66 67 68 69 70 75 80 90]
```

Examples of some grids that are not considered as symmetric (shown are only latitudes):

- The negative latitude of -90 deg does not have its positive counterpart

```
lat = [-90 -60 -30 0 30 60]
```

- The difference "abs(abs(-4.99) - 5.0)" is larger than the threshold of "100.0 * eps" degrees

```
lat = [-35 -25 -15 -4.99 5 15 25 35]
```

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Numerical example

Let's define a spherical grid that is symmetric with respect to the equator.

```
% Latitudes
lat = -90.0:0.1:90.0;

% Longitudes.  The grid step is large in this example, as longitudes do not
% affect the grid symmetry in any way
lon = 0.0:5.0:360.0;


% Grid height
h = 0;
```

Now define the GrafLab input parameters.

```
GM                = 3986004.415E+8;
R                 = 6378136.3;
nmin              = 0;
nmax              = 360;
ellipsoid         = 1;
GGM_path          = '../data/input/EGM96.mat';
crd               = 1;
point_type        = 0;
lat_grd_min       = lat;
lat_grd_step      = 'empty';
lat_grd_max       = 'empty';
lon_grd_min       = lon;
lon_grd_step      = 'empty';
lon_grd_max       = 'empty';
h_grd             = h;
out_path          = '../data/output/howto-g09-symm';
quantity_or_error = 0;
quantity          = 5;
fnALFs            = 3;  % Important
export_data_txt   = 0;
export_report     = 1;
export_data_mat   = 0;
display_data      = 0;
status_bar        = 1;
```

Do the synthesis.

```
tic
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
time_symm = toc;
```

Now let's modify a single latitude, such that the last symmetry property discussed above is not satisfied.

```
lat(1)      = lat(1) + 1000.0 * eps;
lat_grd_min = lat;
```

Do the synthesis.

```
tic
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
```

```
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
time_nosymm = toc;
```

Now let's compare the computation times.

```
fprintf("Symmetric grid:     %0.1f sec\n", time_symm);
fprintf("Non-symmetric grid: %0.1f sec\n", time_nosymm);
```

The speed-up factor grows with increasing harmonic degree and/or increasing number of latitudes. If possible, you should always try to exploit the symmetry property.

```
% For instance, if, for some reason, you have to slice your grid into
% latitudinal bands and call GrafLab multiple times, slice your grid like this
lat1 = [-90.0: 0.01:-80.0 80.0:0.01:90.0];
lat2 = [-79.99:0.01:-70.0 70.0:0.01:79.99];

% Not like this
lat3 = [-90.0:0.01:-70.0];
lat4 = [ 70.0:0.01: 90.0];
```

HOWTO g10: Commission error

You will learn how to compute commission errors from a full covariance matrix of spherical harmonic coefficients and how to avoid a common conceptual mistake when computing commission errors.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Computations of the commission error

The input ASCII file with the error variance-covariance matrix must have the structure as shown in Table 3. A binary MAT-file can also be imported. However, in this case, the empty arrays in Table 3 must be filled with zeroes or corresponding covariances.

Table 3: Structure of the error variance-covariance matrix: spherical harmonic coefficients sorted primarily according to orders; the column CS determines whether the variance and covariances in the particular line are related to the coefficient "Cnm" (if "CS"=0) or to the coefficient "Snm" (if "CS"=1)

```
-------------------------------------------------------------------------------------------------------
CS  n   m   variances and covariances of the spherical harmonic coefficients
-------------------------------------------------------------------------------------------------------
0   2   0   -4.31E-25
0   3   0   -2.11E-26-2.48E-25
0   2   1   -3.79E-28-1.15E-27-3.84E-25
1   2   1   -3.44E-28-4.67E-28-1.17E-27-4.16E-25
0   3   1   -1.99E-27-7.61E-29-2.98E-26-3.18E-28-2.48E-25
1   3   1   -1.44E-28-8.80E-29-3.42E-28-2.54E-26-3.16E-27-2.70E-25
0   2   2   -8.17E-27-1.72E-27-2.94E-28-3.67E-28-9.06E-29-1.08E-27 4.02E-25
1   2   2   -1.14E-27-2.94E-28-5.61E-29-3.86E-28-1.23E-27-1.50E-27-8.37E-28-4.25E-25
0   3   2   -9.38E-27-6.35E-27-1.08E-27-1.81E-27-7.12E-28-3.53E-28-3.30E-26-9.75E-29-3.07E-25
1   3   2   -1.27E-28-3.45E-27-1.59E-27-7.97E-28-1.75E-28-1.15E-28-5.51E-28-2.30E-26-2.78E-27-3.09E-2
0   3   3   -7.74E-28-1.36E-28-9.93E-27-5.50E-28-9.55E-28-3.25E-27-1.06E-27-8.60E-28-2.85E-29-1.58E-2
1   3   3   -1.14E-27-2.19E-28-4.51E-28-1.26E-26-1.46E-28-4.90E-27-1.25E-28-1.76E-28-1.18E-28-5.22E-2
-------------------------------------------------------------------------------------------------------
```

Define the GrafLab input parameters.

```
GM               = 3986004.415E+8;
R                = 6378136.3;
nmin             = 2;  % Important, must be "2" or higher
nmax             = 30;
ellipsoid        = 1;
GGM_path         = '../data/input/GRIM5C1_covmat.mat';
crd              = 0;
point_type       = 0;
lat_grd_min      = -90.0;
lat_grd_step     =   1.0;
lat_grd_max      =  90.0;
lon_grd_min      =   0.0;
lon_grd_step     =   1.0;
lon_grd_max      = 360.0;
h_grd            =   0.0;
out_path         = '../data/output/howto-g10-full';
```

```
quantity_or_error = 1;  % Important
quantity          = 5;
fnALFs            = 1;
export_data_txt   = 1;
export_report     = 1;
export_data_mat   = 1;
display_data      = 2;
graphic_format    = 6;
colormap          = 1;
number_of_colors  = 60;
dpi               = 300;
status_bar        = 1;
```

Do the synthesis

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
```

You may now want to inspect the output files. Importantly, this particular covariance matrix "GGM_path" is not calibrated (scaled). As a result, the values of the commission errors are off by several orders of magnitudes. Nevertheless, the relative spatial relations are correct.

```
fprintf("The output files are ""%s*"".\n", out_path);
```

Common mistake

It is a common mistake to attempt to compute commission errors from a diagonal covariance matrix instead of the full covariance matrix. This is usually done by taking the standard deviations of spherical harmonic coefficients from "gfc" files. The "gfc" file, however, never contains the covariances, which, as will be shown in this example, are crucial to get meaningful commission errors.

Let's take the covariance matrix "GGM_path", load it, set all non-diagonal elements to zero and finally save it to a new file.

```
cm = load(GGM_path);
cm = cm.covmat;
cm(:, 4:end) = eye(size(cm(:, 4:end))) .* cm(:, 4:end);
save('../data/output/GRIM5C1_covmat_diag.mat', 'cm', '-v7.3');
```

Now let's repeat the same computation but with the new covariance matrix, this time having only the diagonal elements. Update the GrafLab input parameters.

```
GGM_path = '../data/output/GRIM5C1_covmat_diag.mat';
out_path = '../data/output/howto-g10-diag';
```

Do the synthesis

```
GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
```

```
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    graphic_format, ...
    colormap, ...
    number_of_colors, ...
    dpi, ...
    status_bar);
```

Now compare the commission errors obtained with the full covariance matrix and with the diagonal elements only. It can be seen that the two maps have almost nothing in common. Moreover, it is not difficult to show that the diagonal covariance matrix leads to commission errors that are perfectly symmetric with respect to the equator, *regardless of the matrix elements*. Obviously, this is not at all realistic.

```
fprintf("The output files are ""%s*"".\n", out_path);
```

HOWTO g11: A trick with the minimum and the maximum degree of the synthesis

You will learn a trick to work with a GGM, the size of which exceeds your RAM.

The main idea is to slice your GGM into several spectral bands (1st band: "0" ... "nmax1", 2nd band: "nmax1 + 1" ... "nmax2", 3rd band: "nmax2 + 1" ... "nmax3", etc.), each of which is small enough to be stored in RAM at once. Then, you can perform the synthesis for each sliced GGM and, finally, sum the syntheses (outside GrafLab) to the get final result. Certainly, this is substantially slower than using a single GGM, so this procedure should be avoided whenever possible. Nonetheless, with high-degree expansions, GGMs may require tens of GBs of RAM, so this may be the only option if you do not have a sufficient amount of RAM.

To employ this functionality, the input GGM file must be provided as the binary "mat" file and must contain three variables:

- a matrix of an arbitrary name (following the MATLAB's rules) with two columns specifying the spherical harmonic coefficients, [Cnm Snm]; the columns with harmonic degrees and harmonic orders are omitted; the rows of the table *must* follow the structure from "Table 1" from "HOWTO g02" and no other ordering scheme is supported (no check on this is performed by GrafLab!),
- an integer named as "nmin" specifying the minimum degree of the coefficients to be imported,
- an integer names as "nmax" specifying the maximum degree of the coefficients to be imported.

All the GrafLab input parameters are explained in ../docs/graflab.md.

```
clear; clc; init_checker();
```

Numerical example

A GGM file with an incomplete set of spherical harmonic coefficients can be obtained from the enclosed gravity field model "../data/input/EGM96.mat" using the following commands:

```
load('../data/input/EGM96.mat'); % Load EGM96, a gravity model up to degree
                                 % 360.  Importantly, the coefficients are
                                 % stored in agreement with "Table 1".  No
                                 % reordering of the coefficients is therefore
                                 % necessary.
EGM96(EGM96(:, 1) < 100, :) = []; % Delete coefficients below degree 100
EGM96(:, 1:2) = []; % Delete information on degrees and orders
nmin = 100; % Specify the minimum degree of the coefficients
nmax = 360; % Specify the maximum degree of the coefficients

% Finally, save the harmonic band "100" ... "360" of the EGM96 model and
% variables "nmin" and "nmax"
GGM_path = '../data/output/EGM96_nmin100_nmax360.mat';
save(GGM_path, 'EGM96', 'nmin', 'nmax', '-v7.3');
```

Now define the rest of the GrafLab input parameters.

```
GM                = 3986004.415E+8;
```

```
R                 = 6378136.3;
ellipsoid         = 1;
crd               = 1;
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      =   1.0;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-g11-sliced';
quantity_or_error = 0;
quantity          = 5;
fnALFs            = 1;
export_data_txt   = 0;
export_report     = 1;
export_data_mat   = 1;
display_data      = 0;
status_bar        = 1;
```

Do the synthesis.

```
out_sliced = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
```

```
    [], ...
    [], ...
    [], ...
    status_bar);
```

Now let's do the same synthesis but this time with the original full "../data/input/EGM96.mat" model and compare the results with the previous synthesis.

```
GGM_path          = '../data/input/EGM96.mat';
out_path          = '../data/output/howto-g11-full';
```

Do the synthesis.

```
out_full = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

Now compute the RMS between the two syntheses (should be below "10^-14" or so).

```
fprintf("RMS of the differences is ""%0.16e"".\n", rms(out_sliced(:, end) - ...
```

```
                                                    out_full(:, end)))
```

HOWTO i01: Synthesis at grids residing on the Earth's topography

You will learn how to perform an efficient solid spherical harmonic synthesis at grids residing on irregular surfaces, e.g. planetary topographies.

All the isGrafLab input parameters are explained in ../docs/graflab.md and ../docs/isgraflab.md.

```
clear; clc; init_checker();
```

Synthesis at a grid residing on the Earth's topography in spherical coordinates

The outline of the experiment is as follows. At first, we synthesize heights of the Earth's topography at a grid in spherical coordinates. The heights are used to define the spherical radius of the grid points. Then, we use the point-wise computation in GrafLab to get the *reference* disturbing potential at the grid residing on the previously computed Earth' topography. Finally, we use isGrafLab to compute the disturbing potential at the very same points on the Earth's surface. The synthesis in isGrafLab is approximate but significantly faster than the point-wise evaluation in GrafLab. The approximation errors can be well-controlled and even safely negligible errors for most applications.

Let's start by synthesizing the Earth's topography from "DTM2006" up to degree "360". At first, we need to define some input parameters (see "HOWTO g06").

```
GM                = 1.0;
R                 = 1.0;
nmin              = 0;
nmax              = 360;
ellipsoid         = 1;
GGM_path          = '../data/input/DTM2006.mat';
crd               = 1;
point_type        = 0;
lat_grd_min       = -90.0;
lat_grd_step      =   1.0;
lat_grd_max       =  90.0;
lon_grd_min       =   0.0;
lon_grd_step      = lat_grd_step;
lon_grd_max       = 360.0;
h_grd             =   0.0;
out_path          = '../data/output/howto-i01-topography';
quantity_or_error = 0;
quantity          = 11;
fnALFs            = 1;
export_data_txt   = 0;
export_report     = 0;
export_data_mat   = 0;
display_data      = 0;
status_bar        = 1;
```

Do the synthesize of the Earth's topography.

```
out_t = GrafLab('OK', ...
```

```
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    lat_grd_min, ...
    lat_grd_step, ...
    lat_grd_max, ...
    lon_grd_min, ...
    lon_grd_step, ...
    lon_grd_max, ...
    h_grd, ...
    [], ...
    [], ...
    [], ...
    [], ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
    [], ...
    status_bar);
```

Now, we have *heights above the geoid* in "out_t", because this is the quantity the "DTM2006.0" model offers. To synthesize the reference disturbing potential at the Earth's topography, we will use GrafLab and the point-wise mode. The grid mode cannot be used, as the radius of the grid points varies irregularly. With the point-wise mode in GrafLab, we need, however, the full spherical radius of each evaluation point. Therefore, for simplicity, we approximate the geoid by a sphere and add its radius "R" to the synthesized heights.

```
% Define the radius of the reference sphere of "EGM96"
R = 6378136.3;

% Get the full spherical radius of the grid points
out_t = R + out_t(:, end);
```

Save the synthesized topography to a format later required by isGrafLab. The data could alternatively be saved in to a text file.

```
is_path = '../data/output/howto-i01-dem.mat';
save(is_path, 'out_t', '-v7.3');
```

Now let's compute the reference disturbing potential on the Earth topography. At first, we define the GrafLab parameters.

```
GM                = 3986004.415E+8;
GGM_path          = '../data/input/EGM96.mat';
point_type        = 2;
quantity          = 5;
out_path          = '../data/output/howto-i01-ref';

% Next, we create arrays with the coordinates of the evaluation points.
[lon_sctr, lat_sctr] = meshgrid(lon_grd_min:lon_grd_step:lon_grd_max, ...
                                lat_grd_min:lat_grd_step:lat_grd_max);
lat_sctr = lat_sctr(:);  % Spherical latitudes of the grid points
lon_sctr = lon_sctr(:);  % Spherical longitudes of the grid points
h_sctr   = out_t;        % Spherical radii of the grid points
```

Synthesis of the reference disturbing potential in GrafLab using the point-wise mode (see "HOWTO g01").

```
tic
out_sctr = GrafLab('OK', ...
    GM, ...
    R, ...
    nmin, ...
    nmax, ...
    ellipsoid, ...
    GGM_path, ...
    crd, ...
    point_type, ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    [], ...
    lat_sctr, ...
    lon_sctr, ...
    h_sctr, ...
    out_path, ...
    quantity_or_error, ...
    quantity, ...
    fnALFs, ...
    [], ...
    export_data_txt, ...
    export_report, ...
    export_data_mat, ...
    display_data, ...
    [], ...
    [], ...
    [], ...
```

```
    [], ...
    status_bar);
time_sctr = toc;
```

Now, we have the reference disturbing at a grid residing on the Earth's topography. At the very same grid, we can now compute the disturbing potential using isGrafLab. Let's define some isGrafLab-specific inputs parameters.

```
h_grd         = mean(out_t - R);  % The height *above the reference sphere with
                                  % the radius "R"*, from which the Taylor
                                  % continuation is performed.  Here, we
                                  % upward/downward continue from a mean sphere
                                  % passing through our topography.
is_mat_or_vec = 1;
out_path      = '../data/output/howto-i01-sph';
status_bar    = 0;
```

Do the synthesis in isGrafLab. We loop over various orders of the Taylor series. The higher is the order, the more accurate results are expected. For each order, we print RMS of the differences with respect to the reference values from GrafLab.

```
K = [0 1 2 5 10 15 20 25];
n = 1;
time_i = zeros(length(K), 1);
for ts = K

    tic
    out_i = isGrafLab('OK', ...
        GM, ...
        R, ...
        nmin, ...
        nmax, ...
        ellipsoid, ...
        GGM_path, ...
        crd, ...
        ts, ...
        lat_grd_min, ...
        lat_grd_step, ...
        lat_grd_max, ...
        lon_grd_min, ...
        lon_grd_step, ...
        lon_grd_max, ...
        h_grd, ...
        is_path, ...
        is_mat_or_vec, ...
        out_path, ...
        quantity, ...
        fnALFs, ...
        [], ...
        export_data_txt, ...
        export_report, ...
```

```
        export_data_mat, ...
        display_data, ...
        [], ...
        [], ...
        [], ...
        [], ...
        status_bar);
    time_i(n) = toc;


    fprintf("RMS error for Taylor order %d: " + ...
            "%0.16e\n", ts, rms(out_i(:, end) - out_sctr(:, end)));
    n = n + 1;

end
```

Note that for a few last Taylor orders, the RMS did not decrease. This is because we have reached *numerically* convergence of the Taylor series. In practice, the proper value of the Taylor order needs to be tuned. Generally, it depends mostly on the gravity field itself, the maximum degree "nmax" and the shape of the irregular surface.

Now let's look at the computation times.

```
fprintf("The synthesis in GrafLab using the point-wise mode took " + ...
        "%0.1f sec.\n", time_sctr);
for i = 1:length(K)
    fprintf("The synthesis in isGrafLab with Taylor order %d " + ...
            "took %0.1f.\n", K(i), time_i(i));
end
```

The synthesis at grids residing on planetary topographies is significantly faster with isGrafLab than it is with the point-wise mode of GrafLab. Moreover, this can be achieved without compromising the accuracy.

Synthesis in isGrafLab in ellipsoidal coordinates

If we repeat the same experiment but in *ellipsoidal* coordinates, we find that the accuracy is worse than it is in spherical coordinates. This is because with ellipsoidal coordinates, the upward/downward continuation should be done along the ellipsoidal normal but in reality is done along the spherical radius. This introduces approximation errors that cannot be narrowed by the order of the Taylor series. The accuracy is therefore worse. Further details are provided in the paper on isGrafLab (the reference is provided at the "README.md").