gcmfaces

A Generic Treatment Of Gridded Earth Variables In Matlab

Gaël Forget
Department of Earth, Atmospheric and Planetary Sciences
Massachusetts Institute of Technology

January 2, 2017

Abstract

`gcmfaces` (Forget et al., 2015) is a `Matlab` toolbox that handles gridded earth variables in generic fashion so that compact analysis codes become applicable to a wide variety of grids (e.g., Fig. 1). `MITprof` (Forget et al., 2015) is a companion toolbox for handling unevenly distributed in-situ observations. This note provides an installation guide for both toolboxes (section 1), a documentation of basic `gcmfaces` features (sections 2), and user guidance regarding higher-level `gcmfaces` functionalities such as mapping and transport computations (sections 3 and 4).

Contents

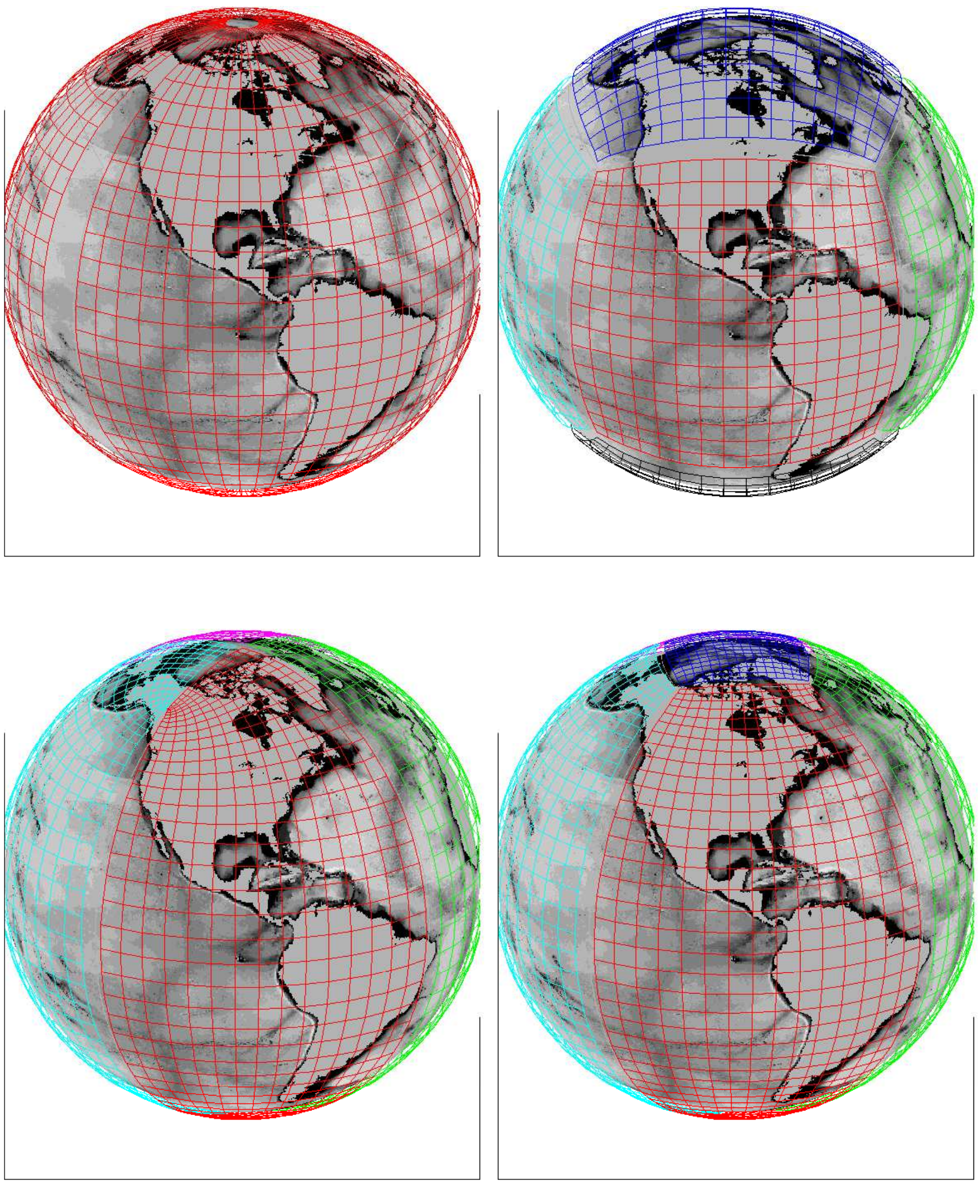


Figure 1: Four approaches to gridding the Earth which are all commonly used in numerical models. Top left: lat-lon grid; mapping the Earth to a single rectangular array ('face'). Top right: cube-sphere grid; mapping the earth to the six faces of a cube. Bottom right: lat-lon-cap, 'LLC', grid (five faces). Bottom left: quadripolar grid (four faces). In this depiction, faces are color-coded, only grid line subsets are shown, and gaps are introduced between faces to highlight the defining characteristics of each grid.

1 Install And Get Started

1.1 Software Installation

The recommended approach consists in downloading the latest `gcmfaces` and `MITprof` software version from github via https://github.com/gaelforget. The code can be downloaded either via a web-browser by using the github interface or via the command line by typing:

```
git clone https://github.com/gaelforget/gcmfaces
git clone https://github.com/gaelforget/MITprof
```

It can later be updated, e.g., by typing `git pull` at the command line.

Alternatively, if needed, earlier versions of the code can be downloaded directly from c66b_gcmfaces.tar and c66b_MITprof.tar or via the `MITgcm` CVS server where the initial development phase, through 2016, is documented. In the latter case, one logs into the `MITgcm` CVS server as explained @ http://mitgcm.org/public/using_cvs.html and then types:

```
cvs co -P -r checkpoint66b -d gcmfaces MITgcm_contrib/gael/matlab_class
cvs co -P -r checkpoint66b -d MITprof MITgcm_contrib/gael/profilesMatlabProcessing
```

1.2 Data Downloads

To get started (sections 1.3 and 2) one downloads the LLC90 grid ('nctiles_grid/'; 145M) either from the MIT ftp server or from its Dataverse permanent archive. To illustrate higher-level functions, sections 3 and 4 rely on the ECCO v4 r2 ocean state estimate (Forget et al., 2016) directories as shown in Fig. 2. The relevant files can be downloaded from the Dataverse permanent archive or from the MIT ftp server, e.g., using commands reported in Fig. 3.

Downloading 'nctiles_climatology/' (10G), 'nctiles_grid/' (145M), and the `Matlab` code (`gcmfaces`, `MITprof`, and `m_map`) suffices for the basic purposes of section 3 and 4. The files in 'profiles/' (7G) and 'nctiles_remotesensing/' (27G)allow for model-data comparisons. The 'nctiles_monthly/' directory contains the full 1992-2011 ECCO v4 r2 monthly time series (170G) and can be used to reproduce the Forget et al. (2016) plots as explained in section 4.

1.3 Get Started

Once 'gcmfaces/', 'MITprof/', and 'nctiles_grid/' have been placed in a common directory ('./' in Fig. 2), one may simply open `Matlab` from that directory and type:

```
%add gcmfaces and MITprof directories to Matlab path:
p = genpath('gcmfaces/'); addpath(p);
p = genpath('MITprof/'); addpath(p);

%load all grid variables from nctiles_grid/ into mygrid:
grid_load;

%make mygrid accessible in current workspace:
gcmfaces_global;
```

```
%display list of grid variables:
disp(mygrid);

%display one gcmfaces variable:
disp(mygrid.XC);

```

Figure 2: Directory structure that allows users to execute `Matlab` code snippets provided in this user guide. The basic `gcmfaces` installation only requires the `gcmfaces/`, `MITprof/`, and `nctiles_grid/` directories (see section 1 for details). The `m_map` toolbox that `gcmfaces` relies on for geographic projections is available at https://www.eoas.ubc.ca/~rich/map.html. The `release2_climatology/`, and `release2/` directories serve to demonstrate higher-level functions in sections 3 and 4.

```
./
|__gcmfaces/ (Matlab toolbox)
|__MITprof/ (Matlab toolbox)
|__m_map/ (Matlab toolbox)
|__nctiles_grid/ (netcdf files)
|__release2_climatology/
|    |__nctiles_climatology/
|    |__mat/ (see section 5)
|    |__tex/ (see section 5)
|__release2/
     |__nctiles_monthly/
     |__nctiles_remotesensing/)
     |__profiles/
     |__mat/ (see section 5)
     |__tex/ (see section 5)
```

Figure 3: Commands to download ECCO v4 r2 (Forget et al., 2016) files used in sections 3-4.

```
setenv FTPv4r2 'ftp://mit.ecco-group.org/ecco_for_las/version_4/release2/'
#export FTPv4r2='ftp://mit.ecco-group.org/ecco_for_las/version_4/release2/'
wget --recursive {$FTPv4r2}/nctiles_grid
wget --recursive {$FTPv4r2}/nctiles_climatology
wget --recursive {$FTPv4r2}/nctiles_monthly
wget --recursive {$FTPv4r2}/nctiles_remotesensing
wget --recursive {$FTPv4r2}/profiles
```

2 The Basic gcmfaces Features

The core of `gcmfaces` lies in its handling of connected arrays/faces via a new `Matlab` class/variable type (section 2.1) and its handling of C-Grid specifications via the `mygrid` global variable (section 2.2). Basic features of `gcmfaces` also include functions that 'exchange' data between faces (section 2.3), 'overloaded' operations (section 2.4), and I/O functions (section 2.5). `gcmfaces` functions are normally documented via help sections that are accessible within `Matlab`.

2.1 The gcmfaces Class

Fig. 1 illustrates four types of grids that are commonly used in general circulation models (GCMs). Despite evident design differences, these grids can all be represented as sets of connected arrays ('faces') as illustrated in Fig. 4 for the LLC90 grid. `gcmfaces` simply takes advantage of this fact by defining an additional `Matlab` class, within `@gcmfaces/`, to represent gridded earth variables generically as sets of connected arrays.

Grid specifics, such as the number of faces and grid points, are embedded within `gcmfaces` objects as illustrated in Table 1. Objects of the `gcmfaces` class can thus be manipulated simply through compact and generic expressions such as 'a+b' that are robust to changes in grid design (Fig. 1). The `gcmfaces` class inherits many of its basic operations (see section 2.4 for details) from the 'double' class as illustrated in Table 2 for `@gcmfaces/plus.m` .

Table 1: Earth variable on the LLC90 grid (Fig. 1, bottom right) represented as an object of the `gcmfaces` class (`@gcmfaces/`). The five face arrays (going from f1 to f5) are depicted in Fig. 4 accordingly.

fld =		
	nFaces:	5
	f1:	[90x270 double]
	f2:	[90x270 double]
	f3:	[90x90 double]
	f4:	[270x90 double]
	f5:	[270x90 double]

2.2 C-Grid Variables

In practice the `gcmfaces` framework gets activated by adding, to the least, the `@gcmfaces/` directory to the `Matlab` path and then loading a grid to memory as done in section 1.3. The default, LLC90, grid can be loaded to memory by calling `grid_load.m` without any argument. 'help grid_load.m' and section 2.5 provide additional information regarding, respectively `grid_load.m` arguments and supported file formats. Alternatively, grids can be read from MITgcm input files using `grid_load_native.m` as shown in this webpage (see `README` and `demo_grids.m`).

`grid_load.m` and `grid_load_native.m` store all C-grid variables at once in a global variable named `mygrid` (Tab. 3). `gcmfaces` functions often rely on `mygrid` that they access via a call to `gcmfaces_global.m` which also provides system information via `myenv`. If these global variables

Table 2: The '+' operation as defined for gcmfaces objects by `@gcmfaces/plus.m`. In executing commands such as 'a+b', `Matlab` will use `@gcmfaces/plus.m` if either 'a' or 'b' is of the `gcmfaces` class.

```
function r = plus(p,q)
%overloaded gcmfaces `+' function :
%  simply calls double `+' function for each face data
%  if any of the two arguments is a gcmfaces object

if isa(p,'gcmfaces'); r=p; else; r=q; end;
for iFace=1:r.nFaces;
   iF=num2str(iFace);
   if isa(p,'gcmfaces')&isa(q,'gcmfaces');
       eval(['r.f' iF '=p.f' iF '+q.f' iF ';']);
   elseif isa(p,'gcmfaces')&isa(q,'double');
       eval(['r.f' iF '=p.f' iF '+q;']);
   elseif isa(p,'double')&isa(q,'gcmfaces');
       eval(['r.f' iF '=p+q.f' iF ';']);
   else;
      error('gcmfaces plus: types are incompatible')
   end;
end;
```

get deleted, typically by a 'clear all', then another call to `grid_load.m` is generally needed. `gcmfaces_global.m` will indicate this situation to the user by issuing warnings that 'mygrid has not yet been loaded to memory'.

Table 3: List of grid variables available via the `mygrid` global variable. The naming convention is directly inherited from the `MITgcm` naming convention. For details, see sections 2.11 and 6.2.4 in http://mitgcm.org/public/r2_manual/latest/online_documents/manual.pdf

XC	:	[1x1 gcmfaces]	longitude (tracer)
YC	:	[1x1 gcmfaces]	latitude (tracer)
RC	:	[50x1 double]	depth (tracer)
XG	:	[1x1 gcmfaces]	longitude (vorticity)
YG	:	[1x1 gcmfaces]	latitude (vorticity)
RF	:	[51x1 double]	depth (velocity along 3rd dim)
DXC	:	[1x1 gcmfaces]	grid spacing (tracer, 1st dim)
DYC	:	[1x1 gcmfaces]	grid spacing (tracer, 2nd dim)
DRC	:	[50x1 double]	grid spacing (tracer, 3nd dim)
RAC	:	[1x1 gcmfaces]	grid cell area (tracer)
DXG	:	[1x1 gcmfaces]	grid spacing (vorticity, 1st dim)
DYG	:	[1x1 gcmfaces]	grid spacing (vorticity, 2nd dim)
DRF	:	[50x1 double]	grid spacing (velocity, 3nd dim)
RAZ	:	[1x1 gcmfaces]	grid cell area (vorticity)
AngleCS	:	[1x1 gcmfaces]	grid orientation (tracer, cosine)
AngleSN	:	[1x1 gcmfaces]	grid orientation (tracer, cosine)
Depth	:	[1x1 gcmfaces]	ocean bottom depth (tracer)
hFacC	:	[1x1 gcmfaces]	partial cell factor (tracer)
hFacS	:	[1x1 gcmfaces]	partial cell factor (velocity, 2nd dim)
hFacW	:	[1x1 gcmfaces]	partial cell factor (velocity, 1rst dim)

The C-grid variable names listed in Tab. 3 derive from the `MITgcm`[1]. In brief, XC, YC, and RC denote longitude, latitude, and vertical position of tracer variable locations. DXC, DYC, DRC and RAC are the corresponding grid spacings, in m, and grid cell areas, in m^2. A different set of such variables (XG, YG, RF, DXG, DYG, DRF, RAZ) corresponds to velocity and vorticity variables that are staggered in a C-grid approach[1].

Indexing and vector orientation conventions also derive from the `MITgcm`[1]. The indexing convention is illustrated in Fig. 4. For vector fields, the first component (U) is directed toward the right of the page and the second component (V) toward the top of the page. As compared with tracers, velocity variable locations are shifted by half a grid point to the left of the page (U components) or the bottom of the page (V components) following the C-grid approach[1].

2.3 Exchange Functions

Many computations of interest (e.g., gradients and flow convergences) involve values from contiguous grid points on neighboring faces. In practice rows and columns need to be appended at

[1]For details, see sections 2.11 and 6.2.4 in http://mitgcm.org/public/r2_manual/latest/online_documents/manual.pdf

each face edge that are 'exchanged' between neighboring faces – e.g., rows and columns from faces #2, #3, and #5 at the face #1 edges in Fig. 4. Exchanges are operated by `exch_T_N.m` for tracer-type variables and by `exch_UV_N.m` for velocity-type variables. They are used to compute gradients (`calc_T_grad.m` and flow convergences (`calc_UV_conv.m`) in sections 3 and 4.

2.4 Overloaded Functions

As illustrated for the '+' operation in Table 2, common functions are overloaded as part of the `gcmfaces` class definition within the `@gcmfaces/` directory:

1. Logical operators: `and, eq, ge, gt, isnan, le, lt, ne, not, or`.

2. Numerical operators: `abs, angle, cat, cos, cumsum, diff, exp, imag, log2, max, mean, median, min, minus, mrdivide, mtimes, nanmax, nanmean, nanmedian, nanmin, nanstd, nansum, plus, power, rdivide, real, sin, sqrt, std, sum, tan, times, uminus, uplus.`

3. Indexing operators: `subsasgn, subsref, find, get, set, squeeze, repmat.`

It may be worth highlighting `@gcmfaces/subsasgn.m` (subscripted assignment) and `@gcmfaces/subsref.m` (subscripted reference) since they overload some of the most commonly used `Matlab` functions. For example, if fld is of the 'double' class then 'tmp2=fld(1);' and 'fld(1)=1;' call subsref.m and subsasgn.m, respectively. If fld instead is of the `gcmfaces` class then `@gcmfaces/subsref.m` behaves as follows:

```
fld{n}     returns the n^{th} face data (i.e., an array).
fld(:,:,n) returns the n^{th} vertical level (i.e., a gcmfaces object).
```

and `@gcmfaces/subsasgn.m` behaves similarly but for assignments.

2.5 I/O Functions

Objects of the `gcmfaces` class can readily be saved to file using `Matlab`'s proprietary I/O format ('.mat' files). Reloading them in a later `Matlab` session works seamlessly as long as the `gcmfaces` class has been defined by including `@gcmfaces/` in the `Matlab` path.

Alternatively, `gcmfaces` variables can be written to files in the 'nctiles' format (Forget et al., 2015). Illustrations in this user guide rely upon ECCO v4 fields which are distributed in this format (see section 1.2; Figs. 2-3). The I/O functions provided as part of `gcmfaces` (`write2nctiles.m` and `read_nctiles.m`) reformat data on the fly.

`gcmfaces` can also read `MITgcm` binary output in the 'mds' format[2]. The provided I/O functions (`rdmds2gcmfaces.m` and `read_bin.m`) rely on `convert2gcmfaces.m` to reformat data on the fly. `gcmfaces` thus readily provides a common tool to analyze any of the ECCO solutions as illustrated in this webpage (see `README` and `demo_grids.m`).

[2]For details, see section 7.3 in http://mitgcm.org/public/r2_manual/latest/online_documents/manual.pdf

3 The gcmfaces_demo.m Tutorial

To proceed further, user should have completed the installation procedure in section 1.3 including for nctiles_climatology/ and m_map/. To illustrate gcmfaces capabilities, gcmfaces_demo.m can then be executed by opening Matlab and typing

```
p = genpath('gcmfaces/'); addpath(p);
p = genpath('m_map/'); addpath(p);
gcmfaces_demo;
```

As prompted by gcmfaces_demo.m , users specify the desired amount of explanatory text output. gcmfaces_demo.m then proceeds various the examples while displaying comments in the Matlab command window. The Matlab GUI and debugger can also be used to run the examples line by line to learn more about the inner workings of gcmfaces functions.

The first section in gcmfaces_demo.m illustrates I/O and plotting capabilities (grid_load.m and example_display.m). gcmfaces relies on m_map) for map projections via the m_map_gcmfaces front-end that typically produces Fig. 5. The second section in gcmfaces_demo.m focuses on data processing capabilities such as interpolation (example_interp.m) and smoothing (example_smooth.m). example_interp.m interpolates gcmfaces fields to a lat-lon grid and vice versa. example_smooth.m integrates a diffusion equation which involves tracer gradient and flux convergence computations. The final section in gcmfaces_demo.m computes oceanic transports (example_transports.m).

4 The gcmfaces_diags/ Standard Analysis

The gcmfaces 'standard analysis' consists of an extensive set of physical diagnostics that are routinely computed to monitor and compare MITgcm simulations and ECCO state estimates (e.g., Forget et al., 2015, 2016). The computational loop is operated by diags_driver.m which expects stores results in a dedicated directory (mat/ in Fig. 2). Afterwards, the display phase is normally carried out via diags_display.m or diags_driver_tex.m as explained below.

At this point, users should have completed the installation procedure in section 1.3 including for nctiles_climatology/ and m_map/ and organized directories as shown in Fig. 2. They can then generate and display variance maps (setDiags='B' encoded in diags_set_B.m) from the ECCO v4 monthly mean climatology (12 monthly fields) by opening Matlab and typing:

```
%add paths:
p = genpath('gcmfaces/'); addpath(p);
p = genpath('MITprof/'); addpath(p);
p = genpath('m_map/'); addpath(p);

%set parameters:
dirModel='release2_climatology/';
dirMat=[dirModel 'mat/'];
setDiags='B';

%compute diagnostics:
diags_driver(dirModel,dirMat,'climatology',setDiags);
```

```
%display results:
diags_display(dirMat,setDiags);
```

which takes ≈ 5 minutes. Each generated plot has a caption that indicates the quantity being displayed. Results of `diags_driver.m` can, alternatively, be displayed via `diags_driver_tex.m` to save plots and create a compilable tex file. This process takes ≈10 minutes:

```
dirTex=[dirModel 'tex/']; nameTex='standardAnalysis';
diags_driver_tex(dirMat,{},dirTex,nameTex);
```

Other diagnostic sets can be computed and displayed accordingly by modifying the 'setDiags' specification: oceanic transports ('A'), mean and variance maps ('B'), sections and time series ('C'), and mixed layer depths ('MLD'). Each set of diagnostics (computation and display) is encoded in one routine named as 'diags_set_XX.m' where 'XX' stands for e.g., 'A', 'B', 'C', or 'MLD'. These routines can be found in the `gcmfaces_diags/` subdirectory.

Computing these four diagnostic sets from ECCO v4 r2 climatology takes ≈1/2 hour. Computing them from the 1992-2011 monthly time series (`nctiles_monthly/` in Fig. 2) per

```
dirModel='release2/'; dirMat=[dirModel 'mat/'];
diags_driver(dirModel,dirMat,[1992:2011]);
```

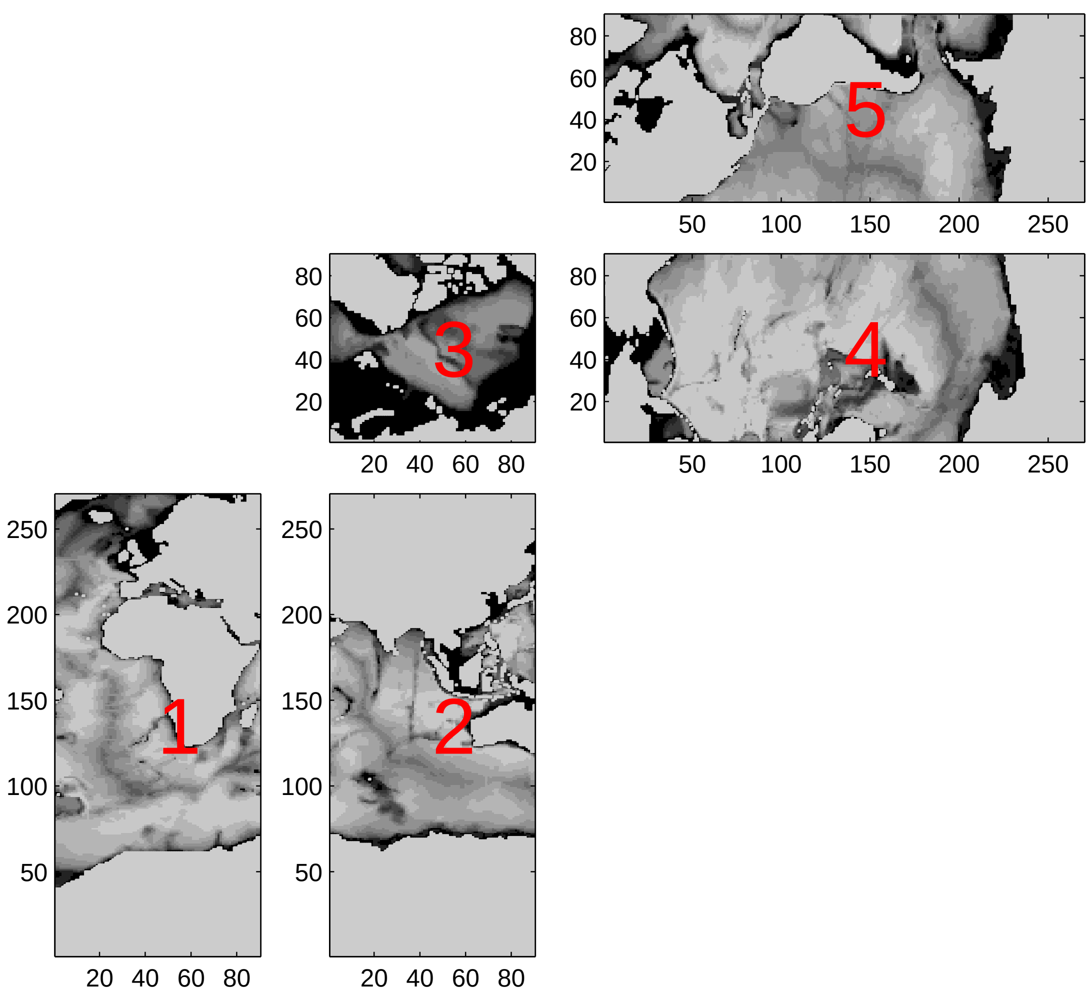
takes ≈ 20 times longer and typically runs overnight. However, to speed up the process, computation can be distributed over multiple processors by splitting [1992:2011] into subsets.

References

Forget, G., J.-M. Campin, P. Heimbach, C. N. Hill, R. M. Ponte, and C. Wunsch, 2015: ECCO version 4: an integrated framework for non-linear inverse modeling and global ocean state estimation. *Geoscientific Model Development*, **8 (10)**, 3071–3104, doi:10.5194/gmd-8-3071-2015, URL http://www.geosci-model-dev.net/8/3071/2015/.

Forget, G., J.-M. Campin, P. Heimbach, C. N. Hill, R. M. Ponte, and C. Wunsch, 2016: ECCO version 4: Second release. URL http://hdl.handle.net/1721.1/102062.

Figure 4: Ocean topography on the LLC90 grid (Fig. 1, bottom right) displayed face by face (going from 1 to 5). This plot generated using `example_display(1)` illustrates how `gcmfaces` organizes data in memory (Tab. 1). Within each face, grid point indices increase from left to right and bottom to top.



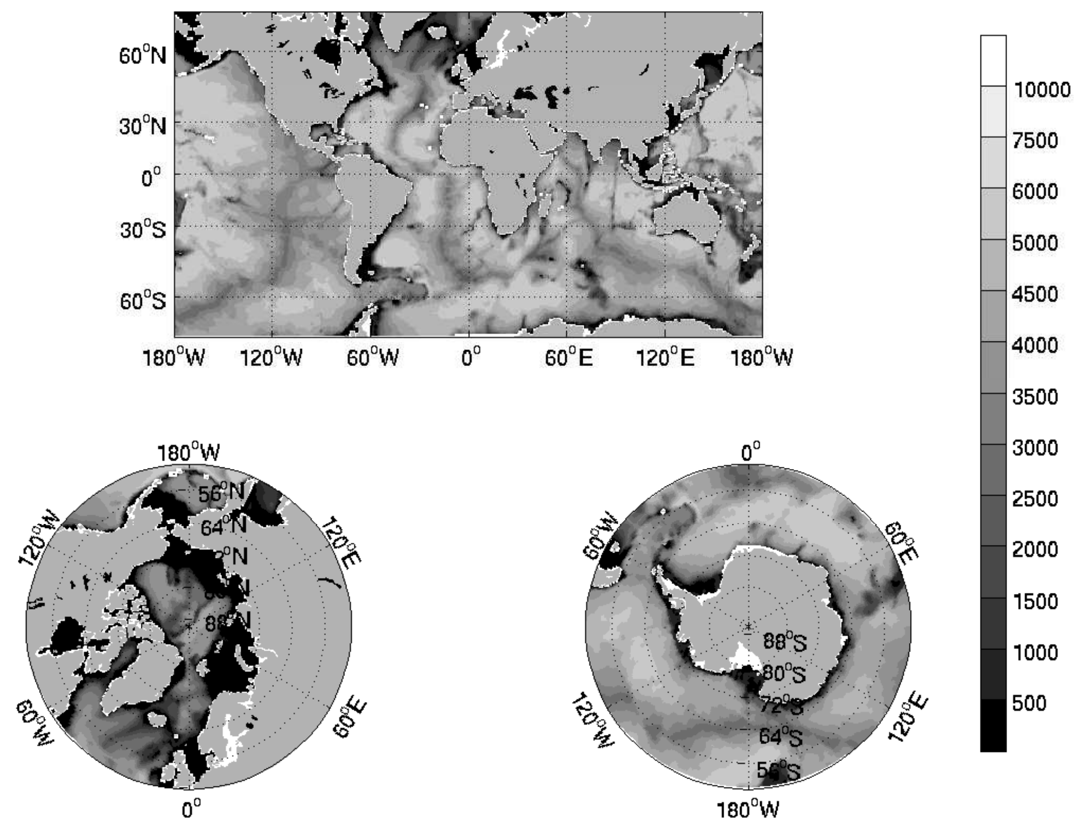


Figure 5: Same as Fig. 4 but plotted in geographical coordinates using `m_map_gcmfaces.m`. This plot is generated by calling `example_display(4)`.