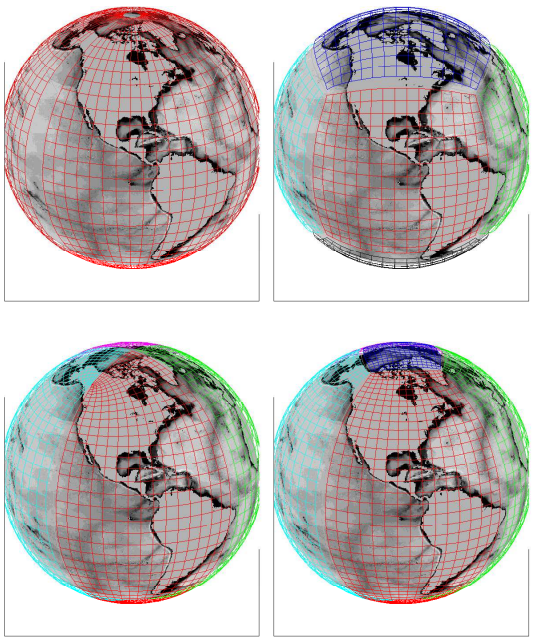
gcmfaces

a matlab framework for the
analysis of gridded earth variables



Gäel Forget *gforget@mit.edu*

Dept. of Earth, Atmospheric and Planetary Sciences
Massachusetts Institute of Technology
Cambridge MA 02139 USA

January 10, 2016

Overview

gcmfaces is a matlab framework designed to handle gridded earth variables; results of MITgcm ocean simulations originally. It allows users to seamlessly deal with a variety of gridding approaches (e.g. see Fig.1) using compact and generic codes. It includes many basic and more evolved functionalities such as plotting, or computing transports, gradients, and budgets. This document provides guidelines to download and update the software, followed by a general presentation of gcmfaces

Download And Update

To download and start using gcmfaces , first donwload the setup_gcmfaces_and_mitprof.csh C-shell script (here); move it to a dedicated directory, and execute

```
source ./setup_gcmfaces_and_mitprof.csh
```

This will download codes from the MITgcm CVS server (see below), and use wget to download sample data sets for testing, then start matlab and test-run the software. This should work just fine under linux and mac os (but probably not under windows) and you should then be ready to use gcmfaces and MITprof [1]. If that approach failed on your system, then you can download a frozen version of the matlab close, and run the tests manually, as depicted in Table 1.

All past and future evolutions of the gcmfaces code can be traced using the cvs version control system. If you downloaded gcmfaces using cvs directly or via setup_gcmfaces_and_mitprof.csh, then you are advised to keep your copy of gcmfaces up to date using the 'cvs update -P -d' command, in order to take advantage of the latest developments. If you are new to cvs , then you may want to be careful and read about this command, e.g. @

```
http://mitgcm.org/public/using_cvs.html
```

An important point is that you want to avoid creating conflicts that may ultimately require extra work. Those could arise if, since your latest update, you modified your copy of a routine in its original directory, and developers

[1] MITprof is a separate package dedicated to the processing and analysis of ocean in situ data (profiles). It uses gcmfaces but is not necessary to gcmfaces , so that you may delete it if you have no use for it.

Table 1: If setup_gcmfaces_and_mitprof.csh cannot be used on your system, then frozen copies of the codes and inputs can be downloaded manually as follows. Note that \ denotes line continuations.

```
wget ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/\
ancillary_data/gcmfaces_MITprof_r1.tar
tar xvf gcmfaces_MITprof_r1.tar

wget ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/\
ancillary_data/GRID_r1.tar
tar xvf GRID_r1.tar

wget ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/\
ancillary_data/OCCAetcONv4GRID_r1.tar
tar xvf OCCAetcONv4GRID_r1.tar

wget --recursive ftp://mit.ecco-group.org/ecco_for_las/version_4/\
release1/nctiles_climatology/ETAN

mkdir gcmfaces/sample_input
mv OCCAetcONv4GRID gcmfaces/sample_input
mv mit.ecco-group.org/ecco_for_las/version_4/release1/\
nctiles_climatology gcmfaces/sample_input/.

matlab -nodesktop
addpath gcmfaces;
gcmfaces_init;
exit;
```

modified it in the cvs repository over the same time span. 'cvs update' usually won't be able to correctly patch two modifications together. You risk ending up with a disfunctional routine that you would have to fix manually. You can avoid this situation by following two simple guidelines: (1) if you need to modify a routine, don't do it in the original directory. Instead copy the routine to another directory of your matlab path, and edit it there. (2) before executing 'cvs update -P -d', always execute 'cvs -n update' to be on the safer side. This command won't do anything to the files but it will print information to screen, telling you what would happen if you actually executed 'cvs update'. This will in particular help you identify and prevent potential conflicts; in case you did not follow guideline #1.

Contents

1 Introduction

MITgcm allows for various ways of discretizing the earth, as do most state of the art general circulation models. A few are illustrated in Fig.1. A key reason for the variety of gridding approaches are numerical limitations implied by grid singularities where grid lines converge. Note that each grid in Fig.1 shows such ' grid poles'. Most advanced grids (e.g. Fig.1, bottom panels) are designed to reduce grid lines convergence, and place 'grid poles' on land rather than in the ocean. Our purpose is not to discuss grid designs in details though – a vast literature exists on this subject. It suffices to say that, at least, the four types of grids shown in Fig.1 are currently used. And, as advanced grids become popular amongst modelers, additional burden can fall on users – a need for new software to analyze the results, and the need for multiple software to compare differently gridded results.

gcmfaces alleviates this burden, by taking advantage of two basic facts. First, despite the apparent heterogeneity and complexity of grids designs, all of the grids we are concerned with can be decomposed into simple rectangular arrays ('faces'). This property is illustrated in Fig.2 for the lat-lon-cap grid of Fig.1 (bottom right). It is a general property simply because those grids are designed for computer use. It is by virtue of this property that MITgcm and gcmfaces can handle various grids in a largely generic way. Second, matlab allows for object oriented programming. As explained in section 2, gcmfaces uses this feature to make grids specifics transparent to the user. The grid specification itself is depicted in section 3. With these elements in place, advanced diagnostics implemented in a generic and compact way are presented in sections 4 and 5.

All routines should include a help section that can be accessed from within matlab. It typipcally states the object of the routine, its input parameters, and its output results. There is no point in duplicating this information here

in details. The routines are organized by topics as depicted in Tab. **to be added**.

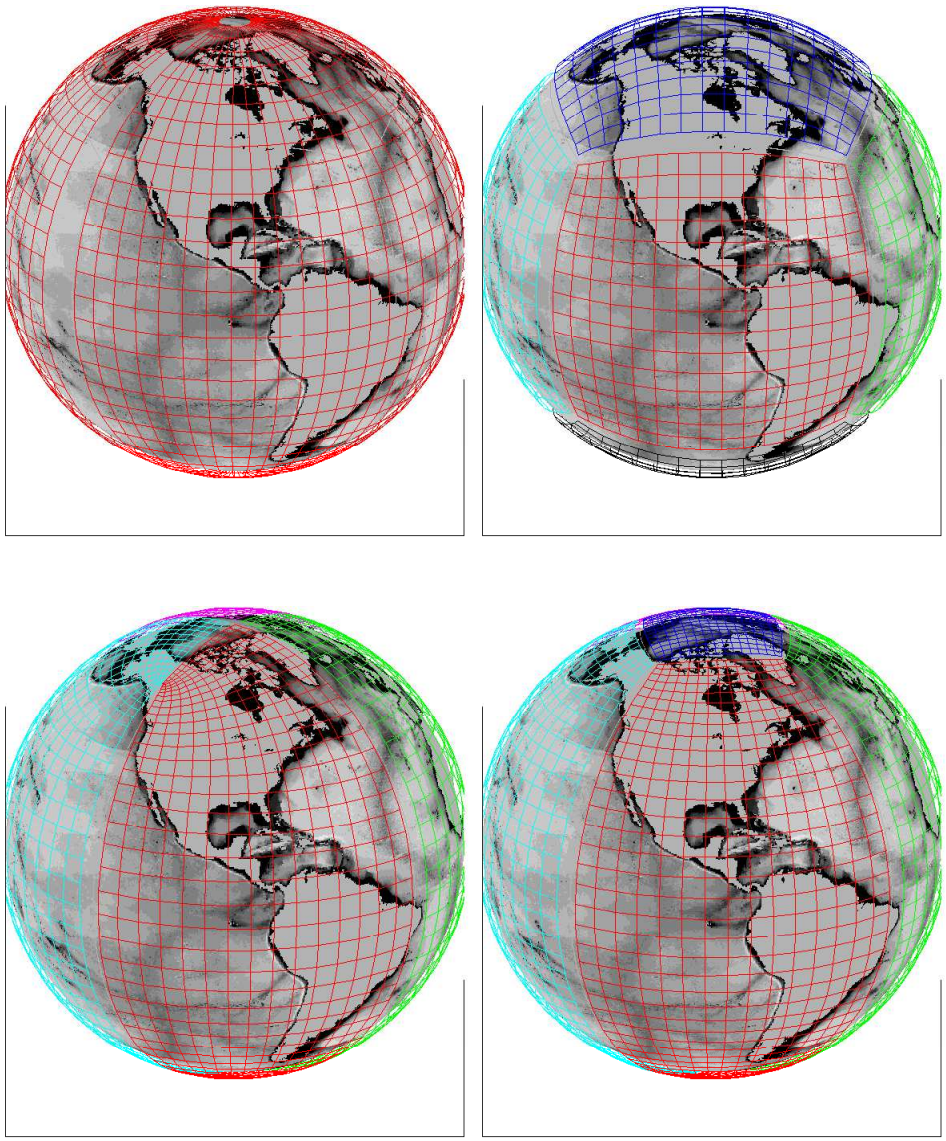
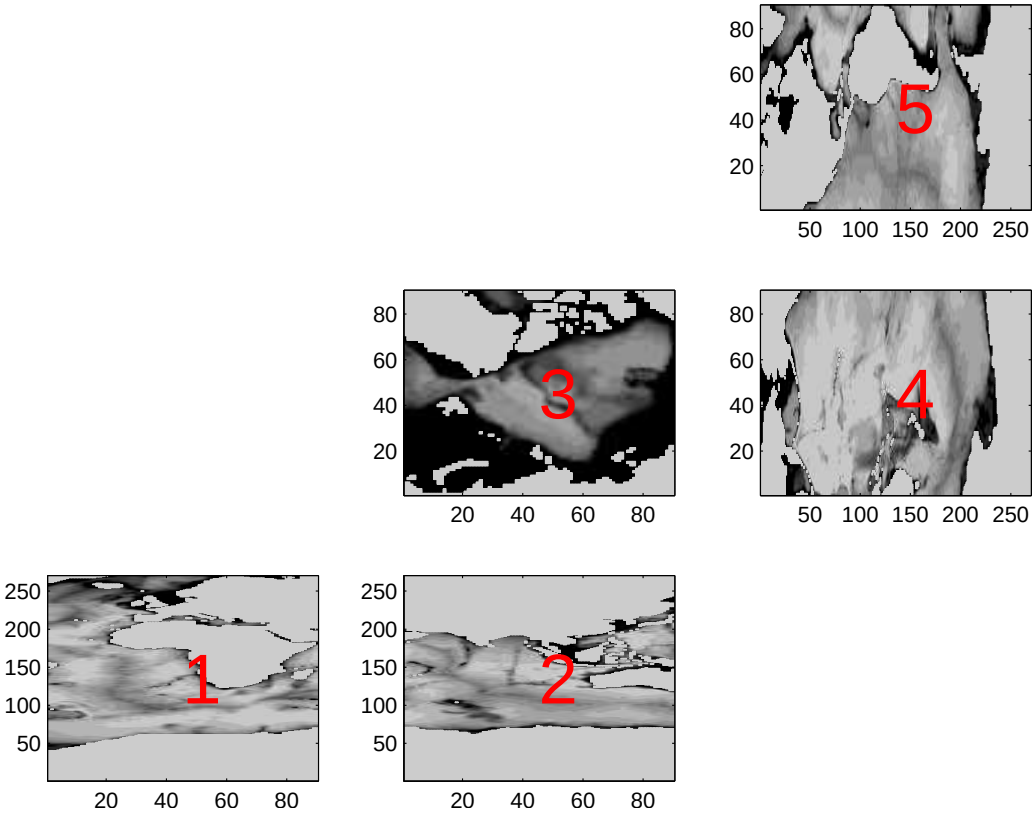


Figure 1: Four different ways of gridding the earth. Top left: lat-lon grid, mapping the earth to a single rectangular array ('face'). Top right: cube-sphere grid, mapping the earth to the six faces of a cube. Bottom right: lat-lon-cap grid (five faces). Bottom left: quadripolar grid (four faces). Faces are color-coded, and the ocean topography underlaid.

Figure 2: Ocean bottom depth, over a lat-lon-cap grid, displayed in gcmfaces format. In each panel, the red number shows the face number, going from one to five. Indexing on each face is discussed in text.



2 The gcmfaces Class

At the core of `gcmfaces` is the gcmfaces class (or data type) defined in the '@gcmfaces' directory. In order to explain its design and specification, it is useful to start by recalling basic notions about matlab classes.

There is a number of pre-defined data types ('classes') within matlab. The most common classes may be 'logical', 'single', 'double' and 'char'. Familiar classes also include 'cell' and 'struct'. Certain operations (e.g. '+' or 'strcat') are associated with certain classes (e.g. double or char). Very rarely is a given operation valid for all classes though (e.g. '+' or 'strcat' are not).

The struct class is relevant to deal with elaborate grid topologies, since it allows us to organize data of various types and sizes (see Tab.2). For our purpose though, a major limitation of struct objects is that numerical operations such as '+' do not apply to struct objects. Adding two such objets requires adding their respective f1, f2, etc. and similarly for any other operation, which would quickly become very cumbersome.

Table 2: gridded earth variable represented as a struct or gcmfaces object, in a case where the grid (Fig.1, bottom right) consists of five faces (f1 to f5).

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

To circumvent this limitation, matlab offers the possibility of user defined

classes, in which the Tab.2 struct is associated with e.g. a valid '+' operation. Once the code for '+' is provided (`@gcmfaces/plus.m` ; see Tab.3) it can be used as 'fld+fld', '1+fld' or 'fld+1'. In executing such commands, as opposed to '1+1', matlab detects that one of the arguments is of the gcmfaces class, so it uses `@gcmfaces/plus.m` rather than the default '+'. This process is often referred to as overloading or overriding an operator. Then the internal logic of `@gcmfaces/plus.m` treats the variety of second arguments (double or gcmfaces) and nFaces values. That basically is what the gcmfaces class boils down to – the Tab.2 struct associated with Tab.3-like operations.

As a result, any higher level `gcmfaces` program (see next sections) can use the compact form '+'. Let us emphasize genericity here. The '1+fld' command will be valid regardless of grid specifics, and won't require any editing when switching grid. A number of such basic operators, as listed below, are readily overloaded as part of `gcmfaces` . They can be found in the @gcmfaces subdirectory that defines the gcmfaces class. The following routines provide a blueprint for users that may want to extend the list of overloaded operators.

- Logical operators: *and, eq, ge, gt, isnan, le, lt, ne, not, or.*
- 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.*
- data operators: *display, find, gcmfaces, get, set, squeeze, subsasgn, subsref, repmat.*

Those functions are generally similar to `@gcmfaces/plus.m` in the sense that they may be called upon just like they would be for arrays, and that they operate face by face. A noteworthy exception is that the overloaded max,

Table 3: @gcmfaces/plus.m : the '+' function for gcmfaces objects.

```
function r = plus(p,q)
%overloaded gcmfaces plus function :
%  simply calls double plus 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;
```

mean, median, min, std, and sum can be applied either globally and face by face. They return global results when passed a sole gcmfaces argument. They return face by face results otherwise. It is also worth emphasizing some of the data operators, namely `@gcmfaces/subsasgn.m` `@gcmfaces/subsref.m` and `@gcmfaces/gcmfaces.m` . While their name may not sound familiar, subsref.m and subsasgn.m are some of the most commonly used matlab functions. Typically, if tmp1 is an array, then 'tmp2=tmp1(1);' and 'tmp1(1)=1;' respectively are compact form calls to subsref and subsasgn. Hence `@gcmfaces/subsref.m` specifies that, for example,

```
fld{n}     will return the n^{th} face data (array).
fld(:,:,n) will return the n^{th} vertical level (gcmfaces).
fld.nFaces will return the nFaces attribute (double).
```

and assignments are consistently done using `@gcmfaces/subsasgn.m` Finally, `@gcmfaces/gcmfaces.m` creates an empty gcmfaces object, according to the specifics of the grid that is in use, which is the next section topic.

3 The Grid Specifications

In practice the `gcmfaces` framework is activated by a grid definition. The lat-lon-cap grid (Fig.1, bottom right) is used for illustration, which is readily available (section 1). Indexing, naming, etc. conventions are simply inherited from `MITgcm` , so we won't present them in extensive details.

`gcmfaces` supports C-grids of various sizes, and for various domain decompostions (see Fig.1). The grid variables (Tab.4) are carried in memory using the `mygrid` global structure. They are first read from file using `gcmfaces_IO/grid_load.m` which requires three arguments : the grid files location (dirGrid), the domain decomposition (nFaces), and the file format (fileFormat). Once the grid fields in Fig.1 are in `mygrid` , this global structure

is available to any routine including a call to gcmfaces_global.m and the user can readily take advantage of the generic operations (previous section) and of higher level gcmfaces functionalities within (next sections).

The grid variables are listed in Tab.4 and output of MITgcm . Extensive details on the C-grid specifications and associated conventions can be found in the MITgcm user manual (section 2.11 and 6.2.4). In brief, XC, YC and RC denote longitude, latitude and vertical position of tracer points (e.g. temperature points). DXC, DYC, DRC and RAC are the corresponding grid spacings (in m) and grid cell areas (in m^2). Another set of such fields (XG, YG, RF, DXG, DYG, DRF, RAZ) is necessary to complete the staggered grid specification. Fig.2 is indicative of the MITgcm indexing and vector conventions. On each face of the lat-lon-cap grid, each tile of the first index goes from left to right, and the second goes from bottom to top. This becomes clear by matching dimensions in Tab.2 with axes in Fig.2.

For a vector field the first component (U) points to the right of the page, whereas the second component (V) points to the top of the page. On the C-Grid, U components are shifted by half a grid point towards the left of the page, while V components are shifted by half a grid point towards the bottom of the page. Computing northward and eastward components (U_n,V_e) of (U,V) proceeds in two steps : (1) average (U,V) to tracer points at the grid cell center and (2) apply a rotation of the coordinate system. To this end, AngleCS and AngleSN give the grid lines orientation relative to meridians and parallels (Tab.4). hFacC, hFacS and hFacW specify the ocean/land mask, and Depth the sea floor depth (Tab.4).

The four grid topologies depicted in Fig.1 are readily supported in gcmfaces The corresponding values for nFaces are 1, 4, 5, and 6. The supported file formats are the MITgcm binary formats[2] and the nctiles defined by gcmfaces

[2]'straight' corresponds to W2_mapIO=-1, 'cube' to W2_mapIO=1, and 'compact' to W2_mapIO=0

Table 4: list of variables contained in the mygrid global structure.

dirGrid	:	'./sample_input/GRIDv4/'	
nFaces	:	5	
fileFormat	:	'compact'	
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)

In practice, introducing a new grid topology requires three items : (1) I/O routines to go map output files to gcmfaces objects, and vice versa; (2) a routine that lays out grid faces to a plane in an array format for plotting purposes (see below); (3) 'exchange' routines that append (to each face) rows and columns from neighboring faces (see below). These only three grid specific elements in gcmfaces allow everything else to be treated generically.

4 A Tutorial : gcmfaces_demo.m

Recall that the setup_gcmfaces_and_mitprof.csh script runs a few basic computational tests (see section 1). Assuming that this process went smoothly: start matlab from the gcmfaces directory and run gcmfaces_demo.m . As prompted: specify the desired amount of explanatory text output. The program (gcmfaces_demo.m) will then carry and depict a series of computations, with text comments printed to the matlab command window. Using the matlab GUI, user can proceed with the computations step by step.

The first call to gcmfaces_global.m adds subdirectories to the matlab path and environment variables in myenv . The other routines called upon by gcmfaces_demo.m in turn call grid_load.m that loads the lat-lon-cap grid into the mygrid global structure (section 3). This call sequence, as explained before, is the basis for any higher level gcmfaces computation. Call to gcmfaces_global.m will be found at the start of any gcmfaces routine that requires grid information. However repeated calls of grid_load.m are not necessary, unless repeatedly clearing global variables.

The first portion of gcmfaces_demo.m focuses on plotting capabilities. A pre-requisite to most plotting operations are format conversions. Indeed common matlab plotting routines expect simple array inputs. Let us start with the quick and dirty method that does not deal with geographical coordinates (qwckplot.m). To this end convert2array.m assembles the

faces data into one array, and imagesc is then used for plotting (Fig.3). The faces numbers as indicated in Fig.3 were rotated and placed consistent with the way `convert2array.m` operates for this grid (compare with Fig.2). For basic geographical coordinates displays, one can use `convert2pcol.m` then pcolor, leading to Fig.4. Finally, `gcmfaces` relies on `m_map` for various geographical projections. The `m_map_gcmfaces` front-end to `m_map` deals with data format conversions, and typically produces Fig.5.

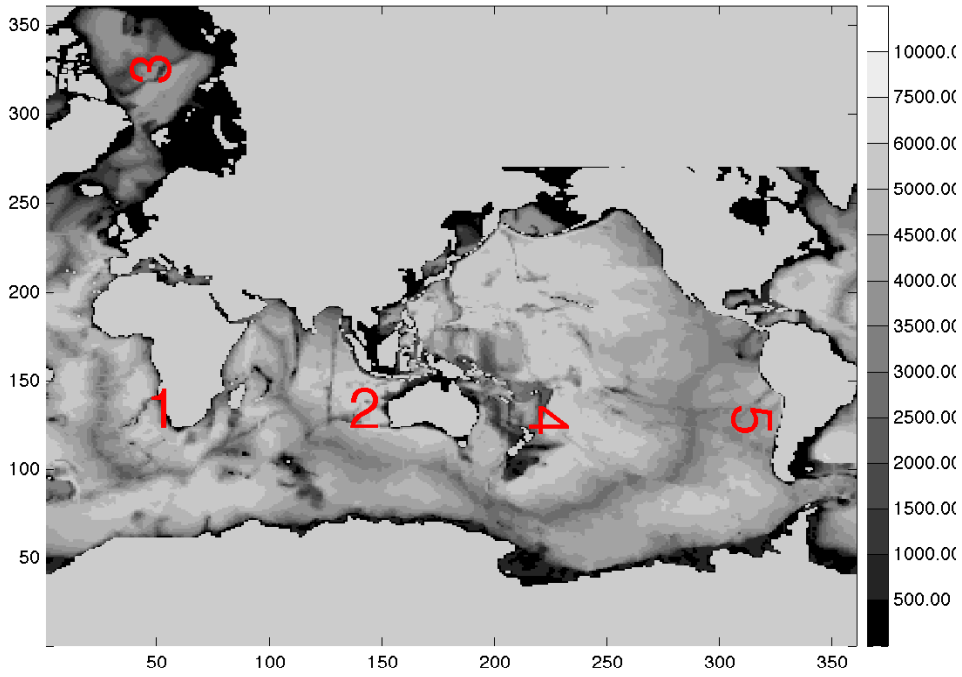


Figure 3: same field as in Fig.2, but once the 5 faces have been assembled into one array, using convert2array.m

The second portion of `gcmfaces_demo.m` focuses on data processing features such as bin averaging, interpolating, smoothing and format conver-

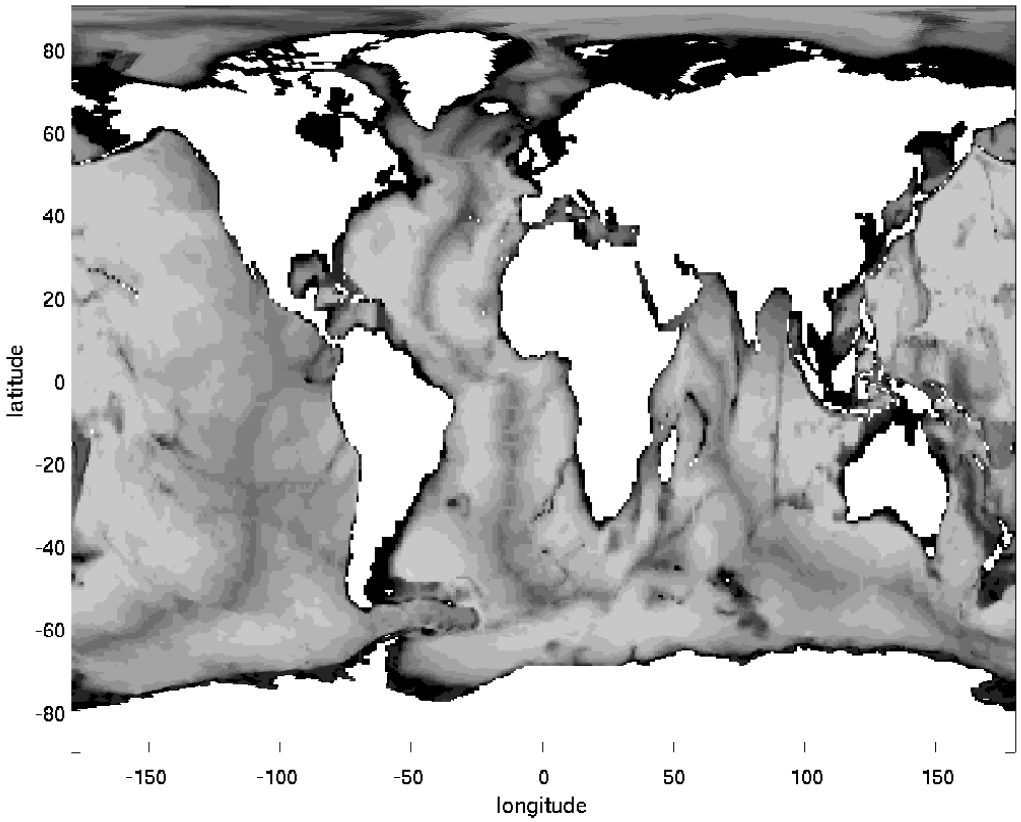


Figure 4: as Fig.2 but in geographical coordinates, using convert2pcol

sions. We want to draw the reader's attention to `example_smooth.m` that has additional value as a tutorial. Indeed it consists in time stepping a diffusion equation, which involves computations of gradients `(calc_T_grad.m)` and divergences `(calcUV_div.m)` . These general interest computations furthermore involve the 'exchange' routines that are instrumental to global computations within the `gcmfaces` and `MITgcm` framework.

Assume, for example, that you want to compute spatial derivatives of a gridded tracer field $\mathcal{T}$ in some ocean domain $\mathcal{D}$ delimited by a closed contour $\mathcal{C}$. It should be clear that you need to know $\mathcal{T}$ not only inside of

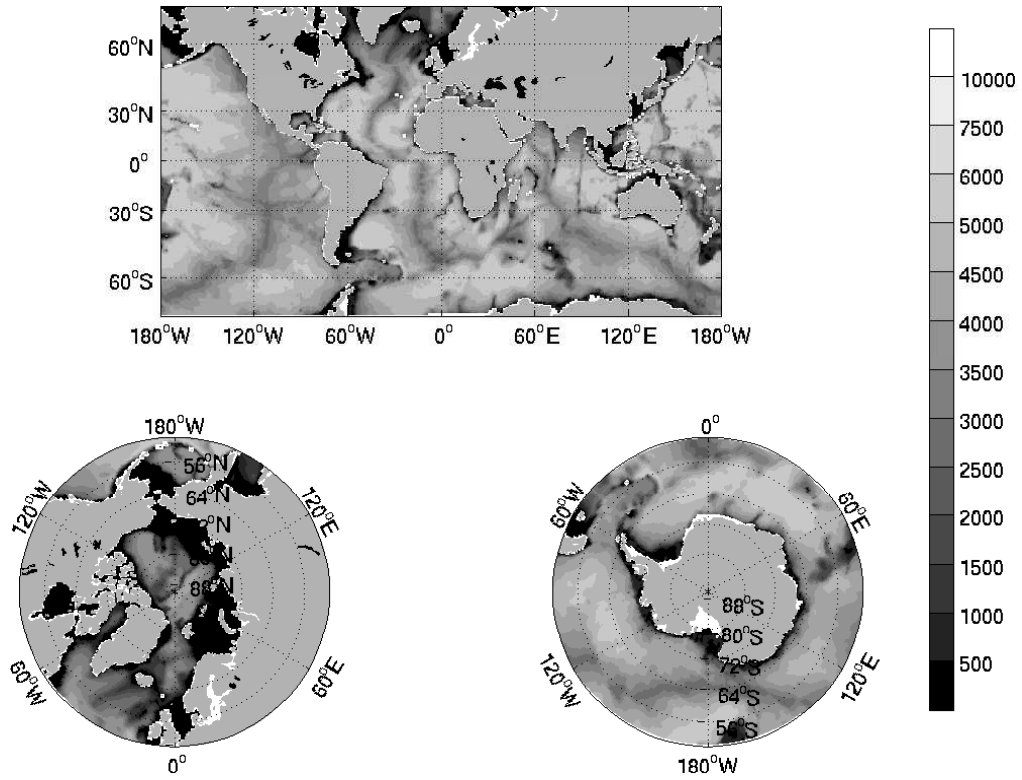


Figure 5: as Fig.2 but in geographical coordinates, using m_map_gcmfaces

$\mathcal{C}$ but also for the neighboring grid points that lie right outside of $\mathcal{C}$. This thought experiment applies to the specific case where $\mathcal{D}$ is one of the grid faces in Fig.2. Thus, to compute gradients on face 1, for example, one needs neighboring values of $\mathcal{T}$ from faces 2, 3, and 5. Exchanging tracer data between neighboring faces is the purpose of `exch_T_N.m` as illustrated in `calc_T_grad.m` . For velocity data, `exch_UV_N.m` is to be used instead as illustrated in `calcUV_div.m` .

The third portion of `gcmfaces_demo.m` charts an analysis of the global ocean circulation (`example_transports.m`). It requires additional model `MITgcm` output (input to `gcmfaces`) that the user can download per

```
mkdir release1
wget --recursive ftp://mit.ecco-group.org/ecco_for_las/\
version_4/release1/nctiles_climatology
mv mit.ecco-group.org/ecco_for_las/version_4/release1/\
nctiles_climatology release1/.
rm -rf mit.ecco-group.org
```

These files will further be used in the gcmfaces standard analysis (section 5), and are expected to be organized according to Fig.7. Each file group (i.e. subdirectory of nctiles_climatology/) can be loaded using read_nctiles.m

To compute transports through oceanic transects, example_transports.m first augments mygrid with mygrid.LINES_MASKS . Indeed, in general, oceanic transects cannot generally be expected to follow grid lines in a simple fashion. Therefore gcmfaces_lines_transp.m determines paths over model grid lines that closely follow specified transects respectively. An example is shown in Fig.6 for a transect defined as the great circle arc between 45E-85N and 135W-85N. Zonal averages and transports are handled in similar fashion via gcmfaces_lines_zonal.m and mygrid.LATS_MASKS .

With these elements in place, example_transports.m illustrates the following computations: barotropic streamfunction (calc_barostream.m) , overturning streamfunction (calc_overturn.m) , transects transports (calc_transports.m) , zonal mean tracer fields (calc_zonmean_T.m) , and meridional heat and fresh water transports (calc_MeridionalTransport.m) The results are displayed by example_transports_disp.m

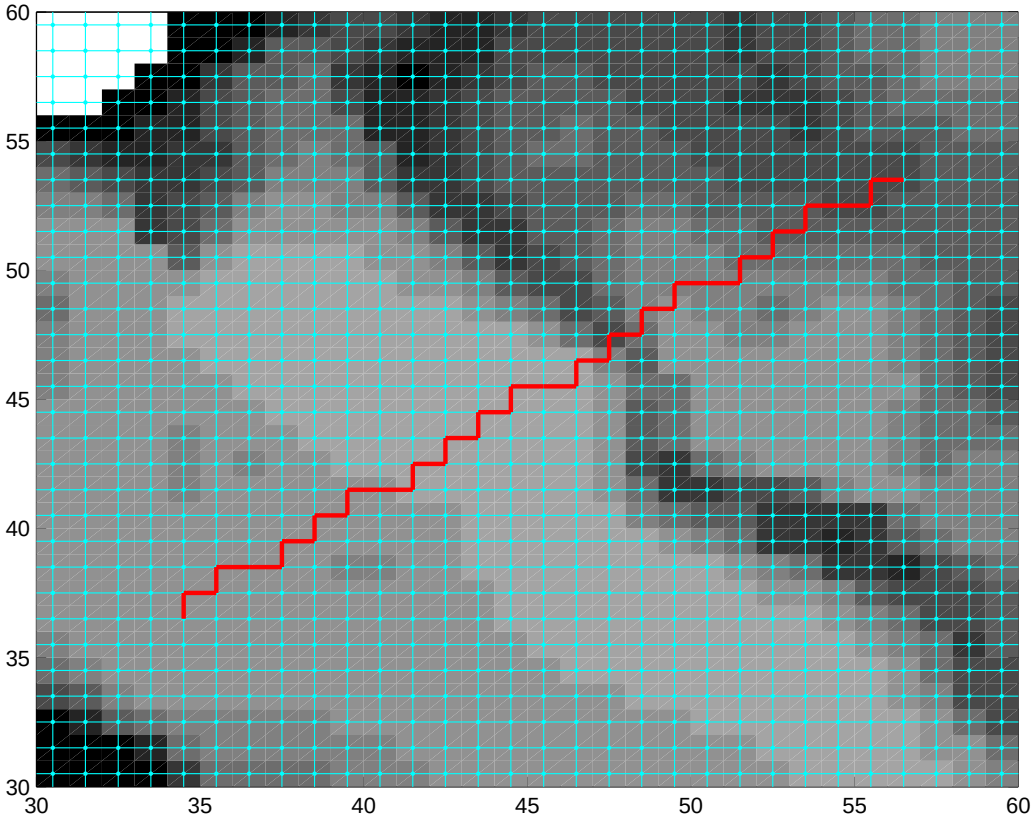


Figure 6: Example of a grid line path (in red) that approximates a transect between 45E-85N and 135W-85N. Location : central part of face 3 from Fig.2. Shading : ocean bottom depth. Blue lines : grid cell edges.

5 The gcmfaces Standard Analysis

The standard analysis (codes in gcmfaces_diags/) consists of a series of physical diagnostics of common interest that are routinely carried out to document and compare simulations of the global ocean. Each set of diagnostics X is entirely encoded (computation and display) in one routine named as `diags_set_X.m` . To define diagnostics X (X being just a placeholder here) users can follow the directions in `diags_set_user.m` or take example of the already included sets of diagnostics in gcmfaces_diags/

The computational loop is operated through `diags_driver.m` with 'X' as the third argument (see `help diags_driver.m` for details) and the results are stored to files in mat/ (see Fig.7). The display phase is done later by calling `diags_display.m` (simple display to screen) or `diags_driver_tex.m` (to generate a pdf file in tex/). An example of the end result, based entirely on already downloaded `gcmfaces` code, can be found here. In the context of state estimation, model-data misfit analyses are also included in the standard analysis (using `MITprof` files and codes).

Gridded physical variables involved in each diagnostic computational loop (e.g. temperature fields 'THETA') are to be provided by users either as MITgcm output (binary files) or in the nctiles format (netcdf files) as depicted in Fig.7. In the nctiles case, the directory and file organization is shown here. In the binary output case, file names and contents are shown there.

Any user can for example regenerate this depiction of the ECCO v4 solution (the gcmfaces 'standard analysis') from the monthly output available here (expectedly organized according to Fig.7) simply by executing diags_driver.m [3] and diags_driver_tex.m [4] in the following sequence :

```
dirModel='release1/';
```

[3] This involves MITprof that also gets installed by this shell script.

[4] User needs to install m_map for mapping and plotting.

```
dirMat='release1/mat/';
dirTex='release1/tex/';
nameTex='standardAnalysis';
%
diags_driver(dirModel,dirMat,1992:2011);%requires gcmfaces and MITprof in path
diags_driver_tex(dirMat,{},dirTex,nameTex);%further requires m_map in path
```

Figure 7: Directory structure as expected by the gcmfaces and MITprof toolboxes. The toolboxes themselves can be relocated anywhere as long as their locations are included in the matlab path. Advanced analysis using diags_driver.m and diags_driver_tex.m will respectively generate the mat/ directory (for intermediate computational results) and the tex/ directory (for the standard analysis document). This diagnostic process relies on the depicted organization of GRID/ (see section 3) and release1/ (see section 5) for automation. User will otherwise be prompted to enter directory names. The nctiles_climatology/ and nctiles/ local subdirectory may contain downloaded copies of these files or those files for example, whereas diags/ may contain binary MITgcm output generated by the user.

```
./
├── gcmfaces/ (matlab toolbox)
│   ├── sample_input/ (binary files)
│   ├── @gcmfaces/ (matlab codes)
│   ├── gcmfaces_calc/ (matlab codes)
│   └── ...
├── MITprof/ (matlab toolbox)
│   ├── profiles_samples/ (netcdf files)
│   ├── profiles_process_main_v2/ (matlab codes)
│   ├── profiles_stats/ (matlab codes)
│   └── ...
├── GRID/ (binary output)
├── release1/
│   ├── diags/ (binary output)
│   ├── nctiles/ (netcdf output)
│   ├── nctiles_climatology/ (netcdf output)
│   ├── MITprof_release1/ (netcdf output)
│   ├── mat/ (created by gcmfaces)
│   ├── tex/ (created by gcmfaces)
│   └── ...
├── other_solution/
│   ├── diags/ (binary output)
│   └── ...
└── ...
```