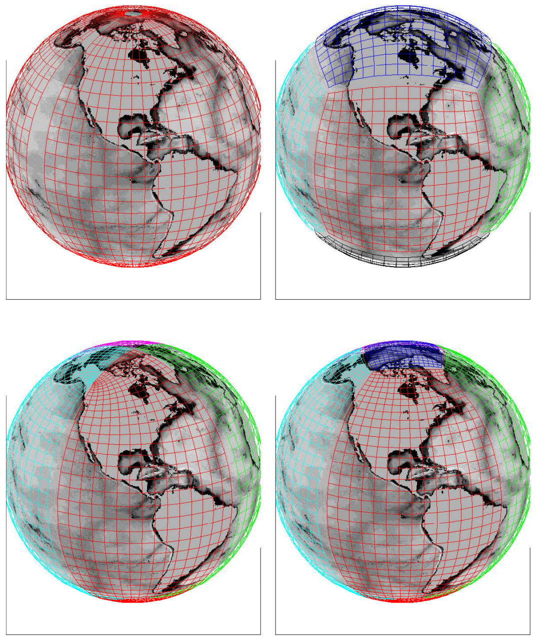
gcmfaces

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

May 7, 2014

Contents

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. Section 2 serves as a broader introduction of gcmfaces ; its design and its basic concepts. Section 3 is dedicated to more advanced functionalities. But first let us explain how to obtain the software, and later keep it up to date.

1 Software Download and Update Procedures

To download and start using `gcmfaces` , first hit the download button at

```
http://mitgcm.org/viewvc/MITgcm/MITgcm_contrib/gael/
          setup_gcmfaces_and_mitprof.csh
```

Move that C-shell script to a dedicated directory, and execute it as

```
source setup_gcmfaces_and_mitprof.csh
```

This will download codes from the `MITgcm` CVS server (see below), wget sample data sets for testing, 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 ever that approach failed on your system, then you want to do download and test manually. Typically, open a terminal window in C shell, and do

```
setenv CVSROOT ':pserver:cvsanon@mitgcm.org:/u/gcmpack'
cvs login
      ( enter the CVS password: "cvsanon" )
cvs co -d gcmfaces MITgcm_contrib/gael/matlab_class
```

Then download, gunzip and untar the sample input data from

```
http://mitgcm.org/~gforget/sample_input.tar.gz
```

Then start matlab from the gcmfaces directory and run `gcmfaces_init.m`

[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.

All past and future evolutions of the software can be tracked using the `cvs` version control system. In order to take advantage of the latest developments, you are advised to keep your copy of `gcmfaces` up to date using the 'cvs update -P -d' command. If you are new to `cvs` , you may want to be careful and read about this command. See for example

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

An important point is that you want to avoid creating conflicts that may ultimately require extra work from you. Those could arise if, since your latest update, you modified your copy of a routine in its original directory, and developers modified it in the `cvs` repository over the same time span. 'cvs update' usually won't be able to patch two modifications together. You risk ending up with a disfunctional patched 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 (1).

2 The Basis of gcmfaces

2.1 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. 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 falls on users. That is the need for new software to analyze the newer 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 broken down 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 can handle various grids in a largely generic way. The same applies within gcmfaces . Second, matlab allows for object oriented programming. As explained hereafter, gcmfaces uses this feature to make grids specifics transparent, but in the lowest level functions such as '+'. Advanced diagnostics can then be implemented in a generic and compact way.

Figure 1: Illustration of four different ways of gridding the earth. Top left: lat-lon grid, which maps the earth to a single rectangular array ('face'). Top right: cube-sphere grid, which maps the earth to the six faces of a cube. Bottom right: lat-lon-cap grid, which maps the earth using five faces (an arctic face, and four mostly lat-lon sectors). Bottom left: quadripolar grid, which maps the earth using four faces. For each plot, the different faces are color-coded, and the ocean topography underlaid.

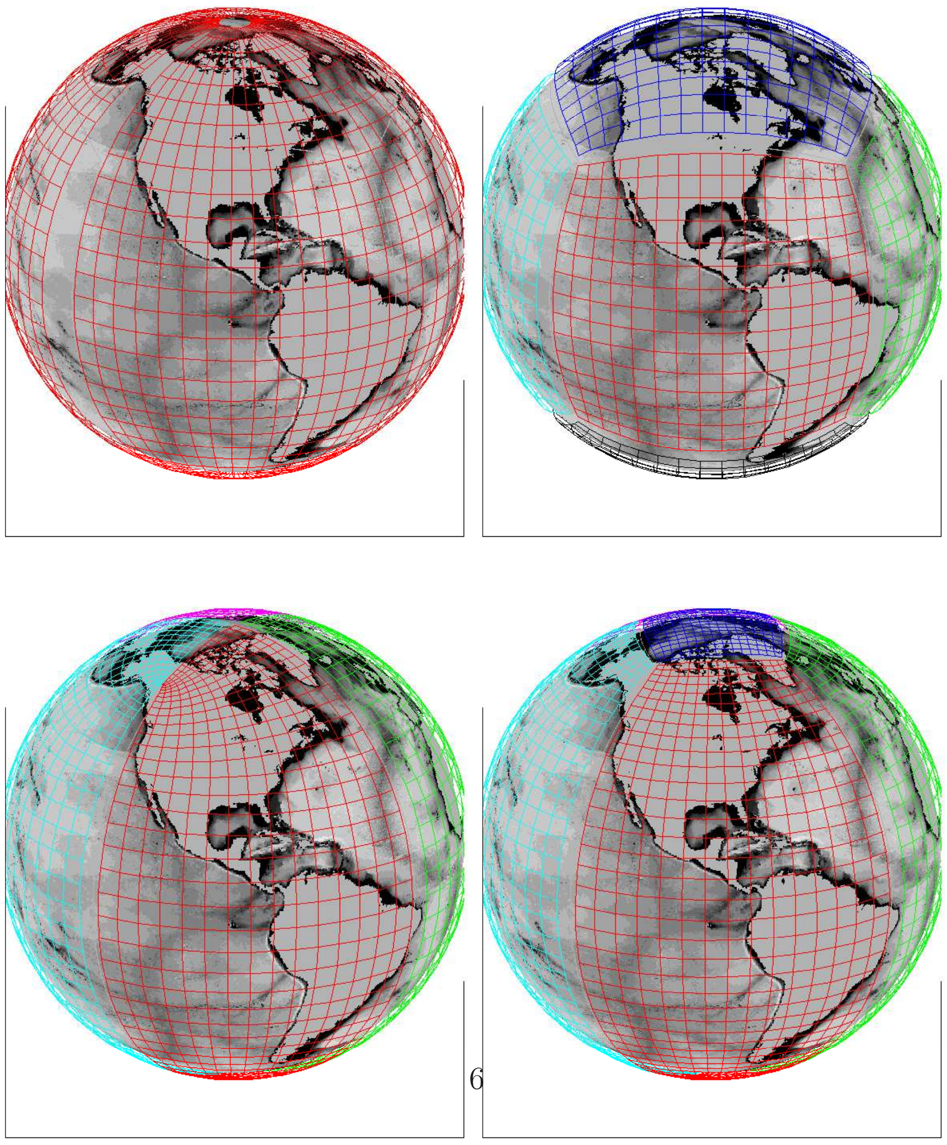
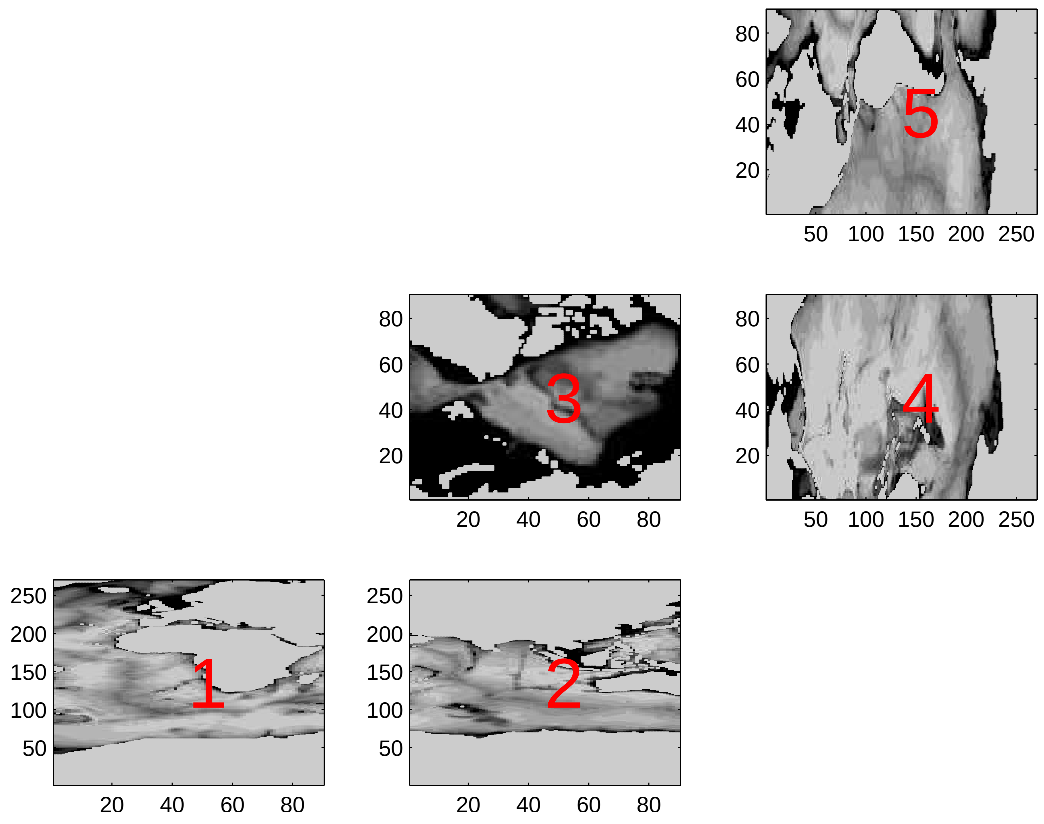


Figure 2: Ocean bottom depth, over a lat-lon-cap grid, displayed in gcmfaces format. In each panel, the red number shows the face number, going from one to five. Indexing on each face is discussed in text.



2.2 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.1). For our purpose though, a major limitation of struct objects is that numerical operations such as '+' cannot be applied to them. Each time we would need to add two such gridded fields, we would thus need to add their respective f1, f2, etc. explicitely. And similarly for any other operation, either simple or complex, which would quickly become very cumbersome.

Table 1: 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 problem, matlab offers the possibility of user defined classes, in which the Tab.1 struct is associated with e.g. a valid '+' operation. Once the code for '+' is provided (`@gcmfaces/plus.m` ; see Tab.2) 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.1 struct associated with Tab.2-like operations.

As a result, any higher level `gcmfaces` programs (see next sections) can use the compact and generic '+' form. Let me emphasize genericity here. The '1+fld' command will be valid regardless of grid specifics, such as the number of faces. It won't require any edit if we were to apply it to a newly specified 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

Table 2: @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;
```

operate face by face. A noteworthy feature however is that max, mean, median, min, std, and sum operators can be applied either globally and face by face. They return global values when passed a sole gcmfaces argument. They return face by face values 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.

2.3 Specifying Grids

So far we have defined gcmfaces objects, and associated low-level operations. To give substance to this framework however, we still need to define a grid. Two grid examples get downloaded along with the `gcmfaces` software (see section 1), which can readily be put to work (see next section). Here we will use the lat-lon-cap grid (Fig.1, bottom right) for illustration, as we get into grid specifics. Let us note however that most conventions are simply inherited from `MITgcm` , so we won't present those in extensive details. Instead we shall emphasize the `gcmfaces` side of things, and point the reader to the `MITgcm` manual for further details on indexing, naming, etc. conventions.

gcmfaces supports C-grids of various sizes, and for various domain decompostions (see Fig.1). The required grid variables (see below) are carried in memory using the mygrid global structure. They must first be read from file by using gcmfaces_IO/grid_load.m which requires three arguments : the grid files location (dirGrid), the domain decomposition (nFaces), and the file format (fileFormat). Those are copied to mygrid (first set of variables in Tab.3) and the grid fields are then accordingly read from file and added to mygrid . Presently, the supported values of nFaces are 1, 4, 5, and 6 (see Fig.1), and the supported file formats are MITgcm binary formats : 'straight' (W2_mapIO=-1), 'cube' (W2_mapIO=1), and 'compact' (W2_mapIO=0).Once the grid fields are in mygrid , they can be accessed in any routine through a call to gcmfaces_global.m That call will also give access to the myenv global structure (environment variables) and add gcmfaces directories to the path (if they were missing).

Tab.3 shows the grid fields that need to be specified as part of mygrid. Their names largely follow MITgcm conventions. In brief, XC, YC and RC denote the longitude, latitude and depth of tracer points (e.g. temperature points). DXC, DYC, DRC and RAC are the corresponding grid spacing fields. Since C-grids are staggered grids, another set of those (XG, YG, RF, DXG, DYG, DRF, RAZ) is necessary to a full specification of the grid. So is the specification of the orientation (AngleCS and AngleSN) of grid lines relative to meridians and parallels. The last set of variables (Depth, hFacC, hFacS and hFacW) is a specification of the GCM ocean/land mask (not its grid per se). Extensive details on C-grid variables and associated conventions can be found in the MITgcm user manual (section 2.11 and 6.2.4).

It is useful to recall indexing and vector conventions though. For this purpose let us consider the lat-lon-cap grid (Fig.1, bottom right) laid out on a plane (Fig.2). Gridded fields represented as gcmfaces objects (Tab.1) basically match that graphical display. On each face the first index goes

Table 3: 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)

from left to right, and the second goes from bottom to top. This becomes clear by matching dimensions in Tab.1 with axes in Fig.2. In addition for a vector field (e.g. a velocity field) the first component (U) points to the right (along horizontal grid lines), whereas the second component (V) points upward. In general a rotation operator is needed to recover northward and eastward components from the so-defined U and V.

Finally let us anticipate on what may be needed to specify a new grid, and take advantage of the `gcmfaces` generic operations. Three things basically: I/O routines to go from files to gcmfaces objects, and vice versa; some format conversion routines to facilitate plotting; a plane lay out of faces such as Fig.2, along with routines to exchange neighboring data amongst faces (see section 3).

3 Higher Level Functions

This section intends to get users started with more advanced analyses. The `gcmfaces_demo.m` routine shall serve as both a stepping stone and a tutorial. It demonstrates a number advanced capabilities that are readily available. Along the way it displays a detailed account of computations and function calls. Below we summarize the so-demonstrated functions and emphasize key features. Then we provide a more thorough inventory of available functions.

3.1 Illustrative Examples

When you downloaded `gcmfaces` (see section 1) some of the features described below have already been tested. Assuming that this first step went smoothly, you are all set for the next step : start `matlab` from the gcmfaces directory and run `gcmfaces_demo.m` As prompted you first specify the demo grid and the amount of explanatory text output. `gcmfaces_demo.m` then carries and depicts a series of computations accordingly.

The first call to `gcmfaces_global.m` sets path and environment variables in `myenv` Then `grid_load.m` loads the chosen grid to `mygrid` This call sequence, as explained before, is the basis for any higher level `gcmfaces` computation. In particular you will find a call to `gcmfaces_global.m` at the start of many routines – eventually including your own.

The first part of `gcmfaces_demo.m` charts a typical analysis of the global ocean circulation. To compute transports through specified transects, we first need to augment `mygrid` with `mygrid.LATS_MASKS` and `mygrid.LINES_MASKS` Indeed, in general, the transects of interest cannot be expected to follow individual grid lines in a simple fashion. Therefore `gcmfaces_lines_zonal.m` and `gcmfaces_lines_transp.m` are used to determine paths over model grid lines that closely follow specified zonal lines and transects respectively. Such a grid line path is shown in Fig.3 for a transect defined as the great circle arc between 45E-85N and 135W-85N.

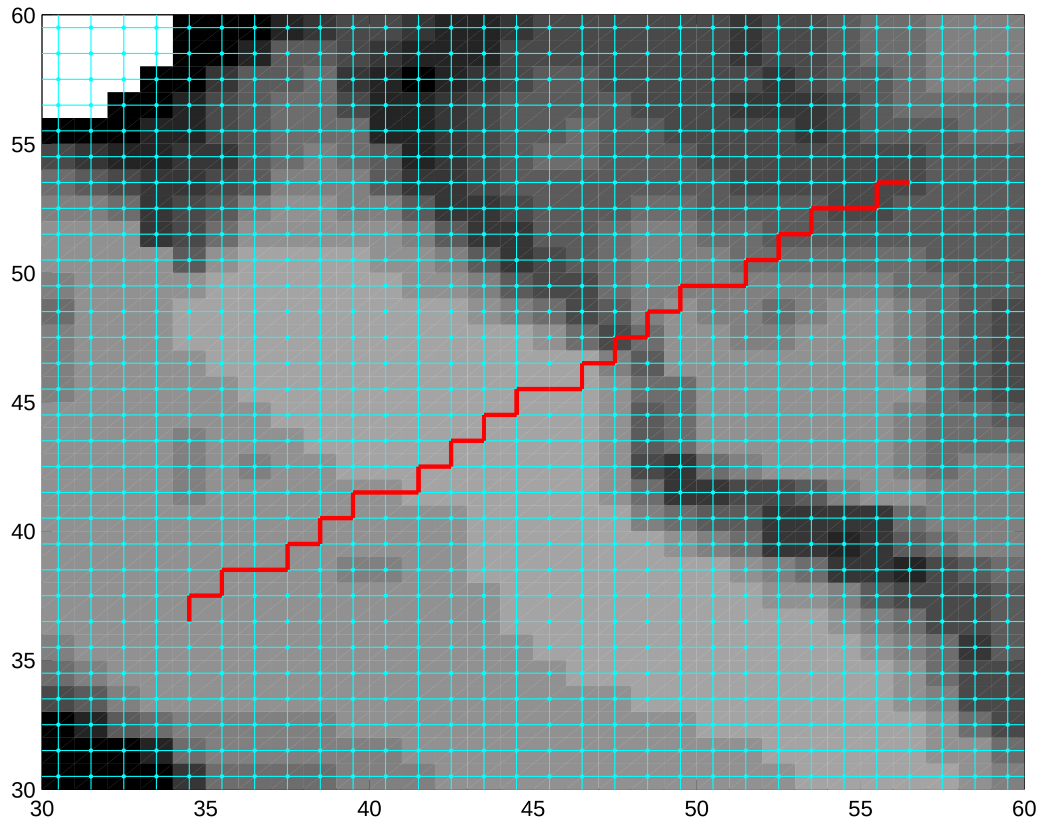


Figure 3: Example of a grid line path (in red) that approximates a transect between 45E-85N and 135W-85N. Location : central part of face 3 from Fig.2. Shading : ocean bottom depth. Blue lines : grid cell edges.

Next velocity and tracer fields are read from file using `rdmds2gcmfaces.m` Then, using `mygrid.LATS_MASKS` and `mygrid.LINES_MASKS` , the following computations are illustrated : 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 stored to disk and later displayed.

The second part of `gcmfaces_demo.m` focuses on data processing features such as bin average, interpolation, smoothing and format conversions. We want to point the reader to `example_smooth.m` in particular. The smoother indeed consists in time stepping a diffusion equation, which involves computing gradients `(calc_T_grad.m)` and divergences `(calcUV_div.m)` Such operations may be of interest to many users. They also give us an opportunity to explain the need for 'exchange' routines.

Assume that you want e.g. to compute spatial derivatives of a gridded tracer field $\mathcal{T}$ in some ocean domain $\mathcal{D}$ delimited by a contour $\mathcal{C}$. It should be clear that you need to know $\mathcal{T}$ not only inside of $\mathcal{D}$ but also for the neighboring grid points that lie right accross $\mathcal{C}$. This thought experiment applies to the specific case where $\mathcal{D}$ is one grid face of Fig.2. In this case, to compute gradients on face 1 e.g., you will need $\mathcal{T}$ from faces 2, 3, and 5 neighbors. Exchanging tracer data between neighboring faces is the purpose of `exch_T_N.m` as illustrated in `calc_T_grad.m` . To exchange velocity data, `exch_UV_N.m` should be used instead (see `calcUV_div.m)`

The third part 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. Here we use `convert2array.m` to assemble the faces data into one array, and then imagesc. Fig.4 is the result for ocean bottom depth. The faces numbers were rotated and placed consistent with the way `convert2array.m` operates for this grid (compare with Fig.2). For basic geographical coordinates displays, we use `convert2pcol.m` then pcolor, leading to Fig.5 Finally, `gcmfaces` relies on `m_map` for geographical projections. The `m_map_gcmfaces` front-end to `m_map` deals with data format conversions, and is capable of producing Fig.6.

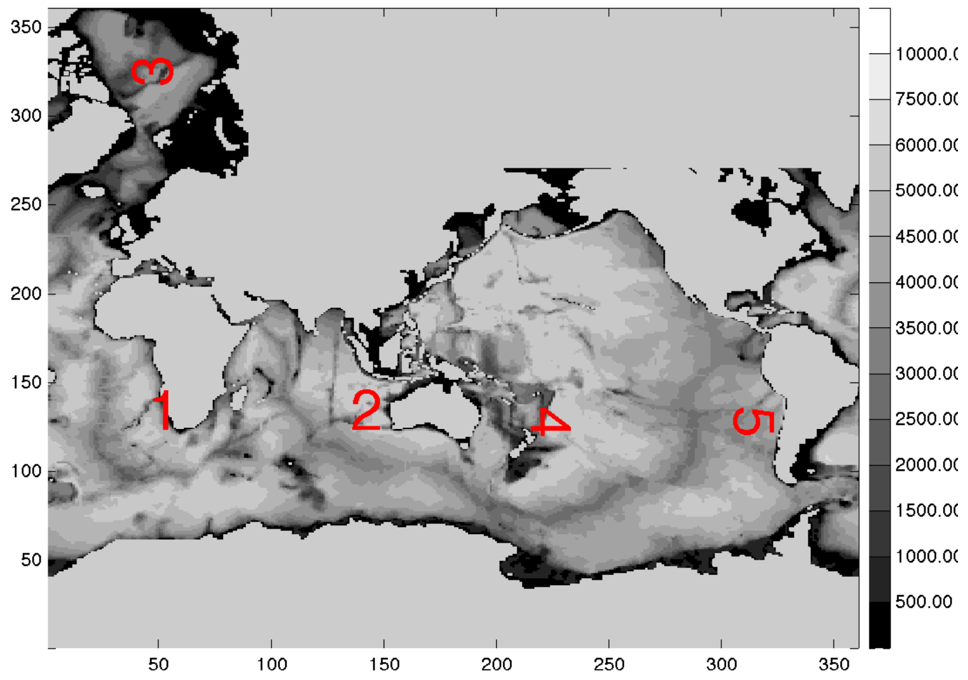


Figure 4: same field as in Fig.2, but once the 5 faces have been assembled into one array, using convert2array.m

3.2 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 is entirely specified (computation and display) in one routine named e.g. as `diags_set_X.m` (where X is a placeholder). The computational phase is done by calling `diags_driver.m` with 'X' as one of the parameters (see help for details) and the results are store to disk. The display phase is done later by calling `diags_driver_tex.m` (to generate

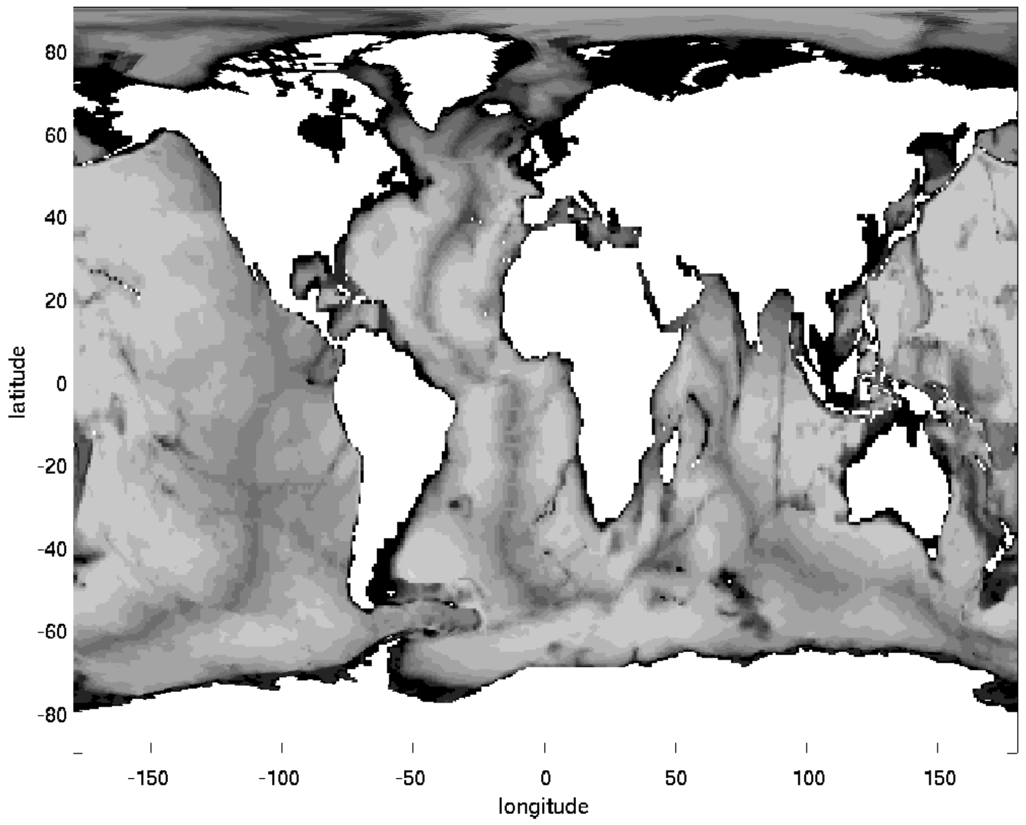


Figure 5: as Fig.2 but in geographical coordinates, using convert2pcol

a tex then pdf file) or `diags_display.m` (simple display to screen). In the context of state estimation, model-data misfit analyses are also included in the standard analysis (codes outside of gcmfaces_diags/). An example of the complete standard analysis can be found @ http://mit.ecco-group.org/opendap/ecco_for_las/ version_4/release1/ancillary_data/standardAnalysis.pdf

Pre-requisites to carrying the standard analysis are :

1. loading grid files listed in Fig.3
2. specifying a few model parameters

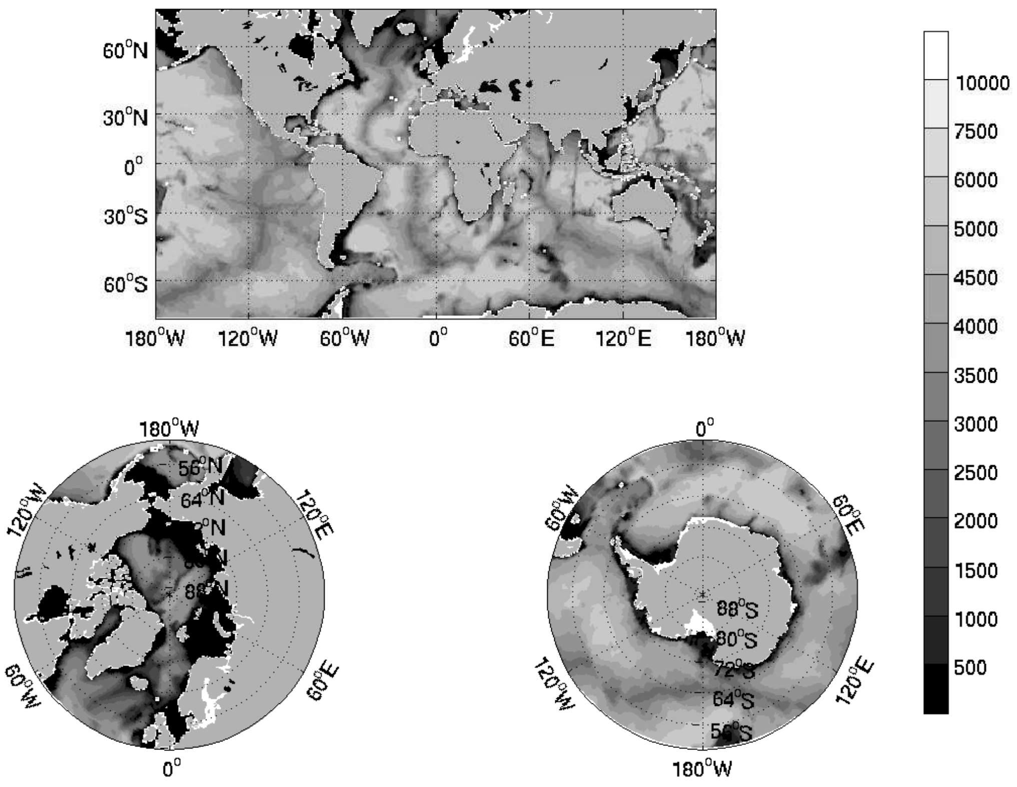


Figure 6: as Fig.2 but in geographical coordinates, using m_map_gcmfaces

3. providing necessary model fields

The first item is done using `grid_load.m` as before. The second item is done interactively : as the user first calls `diags_driver.m` he is asked to select a default parameter set and to edit it if needed. Once the process is complete, mygrid and myparms are stored to disk in a file called diags_grid_parms.mat that is reloaded in later calls to `diags_driver.m`

Model fields involved in each diagnostic computations (e.g. temperature fields 'THETA') are to be provided in one of two supported formats :

1. mds (MITgcm binary output).

2. nctiles (tiled netcdf files).

In the nctiles case the expected file and directory structures are shown @ http://mit.eccogroup.org/opendap/ecco_for_las/version_4/release1/nctiles/ In the mds case directory names and file names are hard-coded for each set of diagnostics. Users can define their own set of diagnostics following the directions in `diags_set_user.m` . A user routine named e.g. as `diags_set_Y.m` could readily be called through `diags_driver.m` etc.

3.3 Functions Inventory

The vast majority of routines include a help section that can be accessed from within matlab. It tyipcally 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 in subdirectories that we document below (Tabs.4-7).

Table 4: Inventory of functions.

./	
gcmfaces_demo.m	demonstrates various capabilities.
gcmfaces_global.m	path and global variables
gcmfaces_init.m	run tests at download
@gcmfaces/	**gcmfaces class definition and methods**
	abs.m, and.m, angle.m, cat.m, cos.m, cumsum.m, cut_T_N.m, diff.m, display.m, eq.m, exch_T_N.m, exch_UV.m, exch_UV_N.m, exp.m, find.m, gcmfaces.m, ge.m, get.m, gt.m, imag.m, isnan.m, le.m, log2.m, lt.m, m_ll2xy.m, max.m, mean.m, median.m, min.m, minus.m, mk3D.m, mrdivide.m, mtimes.m, nanmax.m, nanmean.m, nanmedian.m, nanmin.m, nanstd.m, nansum.m, ne.m, not.m, or.m, plus.m, power.m, rdivide.m, real.m, repmat.m, set.m, sin.m, sqrt.m, squeeze.m, std.m, subsasgn.m, subsref.m, sum.m, tan.m, times.m, uminus.m, uplus.m, zeros.m,

Table 5: Inventory of functions (continued).

ecco_v4/	**routines specific to the analysis of ECCO v4 estimates**
alldiag_load.m	loads basic_diags_ecco results for plot by basic_diags_ecco_disp
basic_diags_ecco.m	compute a set of basic oceano diagnostics
basic_diags_ecco_disp.m	display results of basic_diags_ecco.m
comp_driver.m	wraps up cost and physics computations
plot_driver.m	wraps up cost and physics results display
cost_altimeter.m	compute ecco v4 altimeter cost function for SSH
cost_altimeter_disp.m	display ecco v4 altimeter cost function for SSH
cost_bp.m	compute ecco v4 altimeter cost function for GRACE
cost_sst.m	compute ecco v4 altimeter cost function for SST
cost_summary.m	display costfunction0000 etc.
cost_xx.m	compute ecco v4 altimeter cost function for CONTROLS
gcmfaces_remap.m	remap a lat-lon grid product to a gcmfaces grid
v4_basin.m	obtain the mask of an ocean basin
v4_extract_ll.m	extract the lat-lon part of the grided field
v4_read_bin.m	read binary file (no meta file)
gcmfaces_IO/	**read/write data from/to disk**
	convert2gcmfaces.m, grid_load.m, grid_load_native.m, rdmds.m, rdmds2gcmfaces.m, rdmds2workspace.m, rdmds2workspace_list.m, read2memory.m, write2file.m,

Table 6: Inventory of functions (continued).

gcmfaces_calc/	**physical diagnostics computations**
	calc_MeridionalTransport.m, calc_T_grad.m, calc_UEVNfromUXVY.m, calc_UV_div.m, calc_barostream.m, calc_boxmean_T.m, calc_overturn.m, calc_transports.m, calc_zonmean_T.m, calc_zonmedian_T.m, disp_transport.m, gcmfaces_bindata.m, gcmfaces_edge_mask.m, gcmfaces_lines_transp.m, gcmfaces_lines_zonal.m, gcmfaces_section.m, gcmfaces_subset.m,
gcmfaces_convert/	**format conversions; to and from gcmfaces class**
	convert2arctic.m, convert2array.m, convert2array_cube.m, convert2array_ll.m, convert2array_llc.m, convert2array_llpc.m, convert2cube.m, convert2pcol.m, convert2pcol_cube.m, convert2pcol_ll.m, convert2pcol_llc.m, convert2pcol_llpc.m, convert2southern.m, convert2vector.m,
gcmfaces_exch/	**data exchange between neighboring faces**
	exch_T_N_cube.m, exch_T_N_ll.m, exch_T_N_llc.m, exch_T_N_llpc.m, exch_UV_N_cube.m, exch_UV_N_ll.m, exch_UV_N_llc.m, exch_UV_N_llpc.m, exch_UV_cube.m, exch_UV_ll.m, exch_UV_llc.m, exch_UV_llpc.m,

Table 7: Inventory of functions (continued).

gcmfaces_maps/	**m_map front end and plot related tools**
	gcmfaces_cmap_cbar.m, m_map_1face.m, m_map_1face_uv.m, m_map_gcmfaces.m, m_map_gcmfaces_movie.m, m_map_gcmfaces_uv.m, m_map_gcmfaces_uvrotate.m,
gcmfaces_misc/	**miscellaneous functions**
	ccaa.m, convertR4toR4nonan.m, convertR8toR4.m, density.m, depthStretch.m, depthStretchPlot.m, diff_mat.m, gcmfaces_msg.m, imagescnan.m, input_list_check.m, runmean.m, sym_g.m, write2tex.m,
gcmfaces_smooth/	**smoothing and extrapolation via diffusion equations**
	diffsmooth2D.m, diffsmooth2D_div_inv.m, diffsmooth2D_extrap_fwd.m, diffsmooth2D_extrap_inv.m, diffsmooth2Drotated.m,
sample_analysis/	**demonstration routines, called by gcmfaces_demo.m**
	basic_diags_compute_v3_or_v4.m, basic_diags_display_transport.m, basic_diags_display_v3_or_v4.m, line_greatC_TUV_MASKS_v3.m, line_greatC_TUV_MASKS_v4.m, plot_one_field.m, plot_std_field.m,
sample_processing/	**demonstration data for use in gcmfaces_demo.m**
	example_bin_average.m, example_griddata.m, example_interp.m, example_smooth.m,