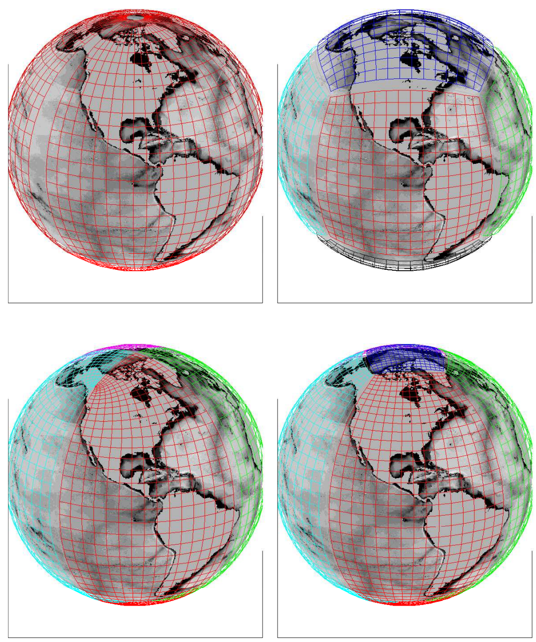
gcmfaces

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 14, 2016

Contents

Summary

`gcmfaces` is a Matlab framework designed to handle gridded earth variables; results of `MITgcm` ocean simulations originally (Forget et al., 2015). It allows users to seamlessly deal with various gridding approaches (e.g. see Fig.2) using compact and generic codes. It includes many basic and more evolved functionalities such as plotting, or computing transports, gradients, and budgets. `MITprof` is a complementary toolbox to handle in-situ ocean observations (Forget et al., 2015). This document provides guidelines to download and update the software (section 1) followed by the `gcmfaces` documentation. Its design and basic features are presented in sections 2 and 3. Higher level functions are illustrated in sections 4 and 5.

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/.

Disclaimer

1 Download And Update

There are two ways to download and start using `gcmfaces` and `MITprof`:

1. download frozen copies: arguably the simplest method that will work in all computing environments (Linux, iOS, MS-windows).

2. use the `MITgcm` CVS server: this is the recommended method under Linux and iOS (assuming CVS was installed) since it has the major advantage that the codes can later easily be updated.

This section documents both methods and the setup of `gcmfaces`.

1.1 download frozen copies

The frozen copies of gcmfaces and MITprof are stored at ftp://mit.ecco-group.org/ecco_for_las/version_4/checkpoints/

Download the latest versions[1], uncompress and untar them, and rename the two directories as 'gcmfaces' and 'MITprof'. When starting Matlab, one will add these two directories to the path as explained in section 1.3.

1.2 use the MITgcm CVS server

Login to the `MITgcm` CVS server as explained in this page then download the up to date versions of `gcmfaces` and `MITprof` by typing

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

All past and future evolutions of the codes can be traced using the `cvs` version control system. To update an existing copy of the codes and

[1] gcmfaces_20160114.tar.gz and c65r_MITprof.tar.gz at the time of writing.

take advantage of the latest developments one typically goes inside a directory and types 'cvs update -P -d' at the command line. If you are new to `cvs` then you may want to read about the update command at http://mitgcm.org/public/using_cvs.html.

1.3 getting started with gcmfaces

Download the LLC90 grid (Forget et al., 2015) directory at ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/nctiles_grid/ as shown in Fig. 1. Then start Matlab and load the grid by typing:

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

%load nctiles_grid in memory:
grid_load;

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

The applications in sections 4 and 5 further require downloading: ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/nctiles_climatology/ and adding the `m_map` plotting toolbox to the Matlab path: https://www.eoas.ubc.ca/ rich/map.html

Figure 1: Directory structure that is consistent with the Matlab commands in Sect. 1.3. The nctiles_climatology/ directory (14G) contains the monthly mean climatology of the ECCO v4, release 1 state estimate (Forget et al., 2015). m_map and nctiles_climatology/ are not necessary in section 1.3 but are used to demonstrate higher-level functions in sections 4 and 5.

```
./
├── gcmfaces/ (Matlab toolbox)
├── MITprof/ (Matlab toolbox)
├── m_map/ (Matlab toolbox)
├── nctiles_grid/ (netcdf files)
└── release1/
    ├── nctiles_climatology/ (netcdf files)
    ├── mat/ (see section 5)
    └── tex/ (see section 5)
```

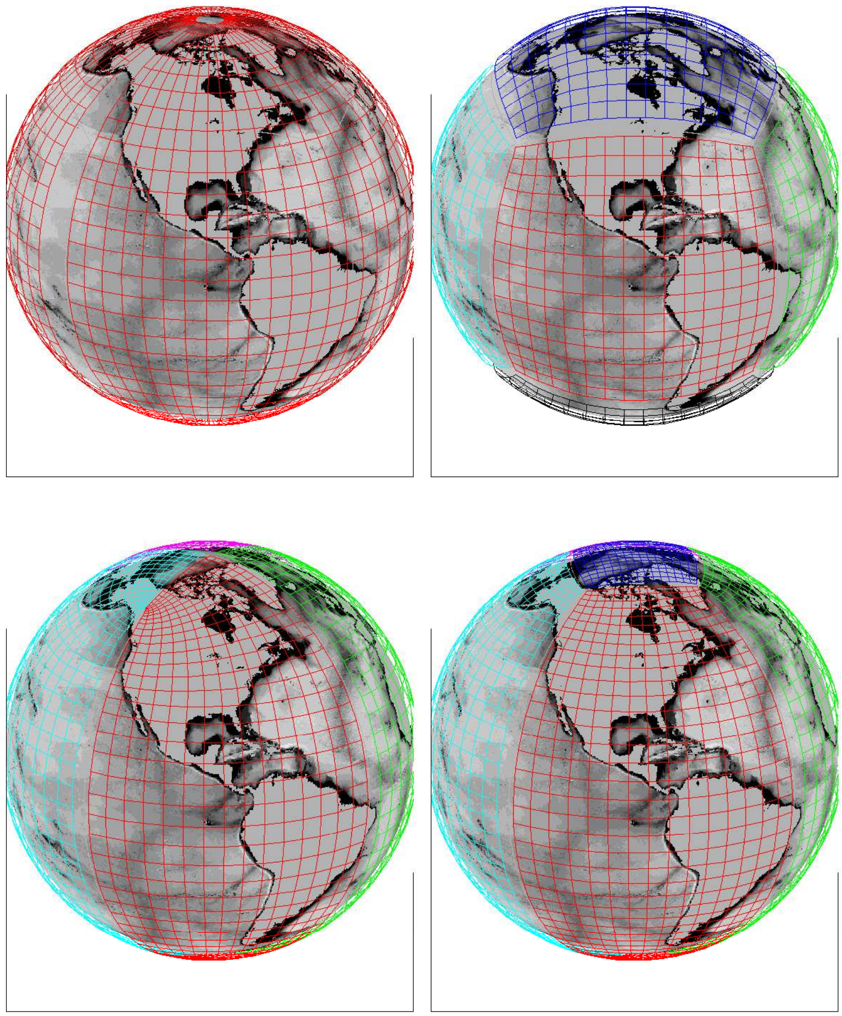


Figure 2: Four different ways of gridding the earth. Top left: lat-lon grid, mapping the earth to a single rectangular array ('face'). Top right: cube-sphere grid, mapping the earth to the six faces of a cube. Bottom right: lat-lon-cap 'LLC' grid (five faces). Bottom left: quadripolar grid (four faces). Faces are color-coded, and the ocean topography underlaid. Only a subset of the grid lines are shown in this depiction.

2 The gcmfaces class

The basic motivation for developing `gcmfaces` was to provide a unified framework that allows for the analysis of earth variables on various grids. Fig. 2 shows four types of grids that are commonly used in ocean general circulation models (GCMs). Despite evident differences in GCM grid designs, such grids can all be represented as sets of connected arrays (or 'faces'). This fact is illustrated in Fig. 3 for the LLC90 grid (bottom right panel in Fig.2) that is used in ECCO v4 (Forget et al., 2015).

The core of `gcmfaces` lies in its definition of a new Matlab data type (or 'class') that represents gridded earth variables as sets of connected arrays (the '@gcmfaces/' subdirectory). An object of the gcmfaces class is stored in memory as shown in Table 1. The gcmfaces class inherits many of its basic operations (e.g., '+') from the 'double' class as illustrated by `@gcmfaces/plus.m` (see Table 2). Objects of the gcmfaces class can thus be manipulated simply through compact and general expressions such as 'a+b' (see section 3.3) that are robust to changes in grid design.

Table 1: Gridded variable represented using the gcmfaces class. In this case the LLC90 grid (Fig.2, bottom right) is used that has five faces (f1 to f5).

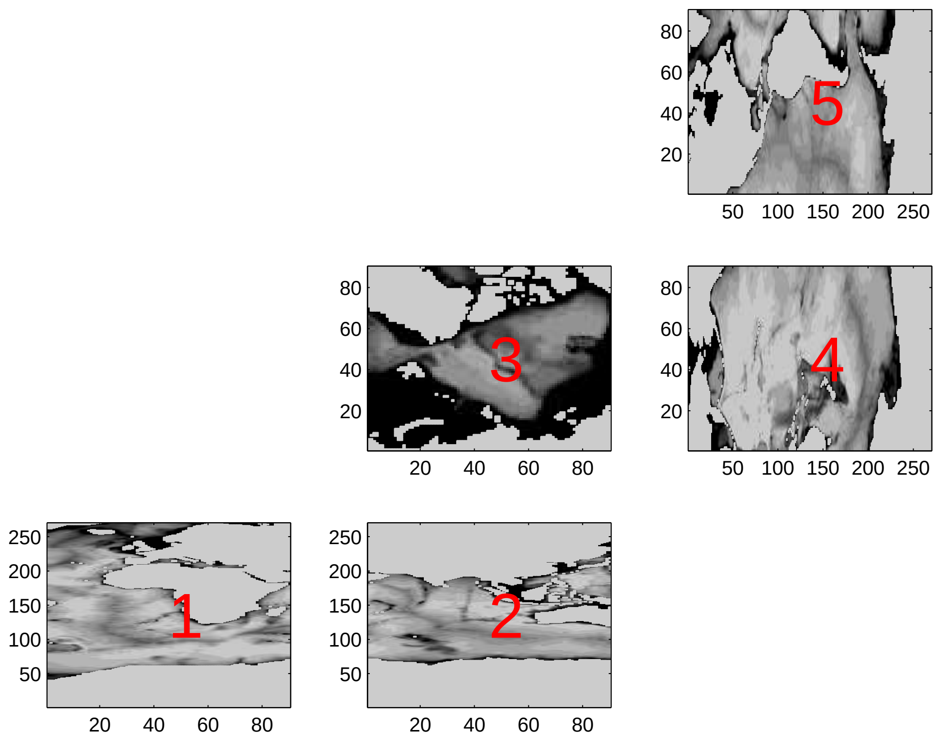
fld =	
nFaces:	5
f1:	[90x270 double]
f2:	[90x270 double]
f3:	[90x90 double]
f4:	[270x90 double]
f5:	[270x90 double]

Table 2: The '+' operation for gcmfaces objects (@gcmfaces/plus.m).

```
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;
```

Figure 3: Ocean topography displayed face by face for the LLC90 grid (Fig.2, bottom right). The face indices (from 1 to 5) are overlaid in red. Within each face, grid point indices increase from left to right and bottom to top in this view that reflects the data organization in memory (Tab. 1).



3 Basic Features

The representation of grid variables in memory is documented in section 3.1. Other key features of `gcmfaces` are the 'exchange' functions that connect faces (section 3.2) and the 'overloading' of common operations (section 3.3). I/O functions are discussed in section 3.4.

3.1 Grid Variables

In practice the `gcmfaces` framework gets activated by loading a grid in memory using the `grid_load.m` function. The default grid (LLC90) can be loaded in memory through a call to `grid_load.m` without any argument (as done in Sect. 1.3). For other grids, `grid_load.m` arguments need to be specified as explained by 'help grid_load.m'. `grid_load.m` stores all grid variables in memory within a global structure named `mygrid` (Tab.3).

`mygrid` can be accessed in Matlab at any point by declaring it as 'global mygrid;' or using `gcmfaces_global.m`. The latter method additionally: (1) issues a warning when 'mygrid has not yet been loaded to memory'; provides a few environment variables via `myenv`; adds gcmfaces directories to the path if needed. It should be stressed that gcmfaces functions often rely on `mygrid` and `myenv`. If they get deleted from memory (e.g., by a 'clear all') then a call to `grid_load.m` will re-activate gcmfaces properly.

The C-grid variables listed in Tab.3 follow the MITgcm naming convention (see sections 2.11 and 6.2.4 in this documentation). In brief, XC, YC and RC denote longitude, latitude and vertical position of tracer variables. 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 C-grid specification where velocity variables are shifted compared with tracer variables.

The indexing and vector conventions also derive from the `MITgcm`. The

Table 3: List of grid variables contained in the mygrid global structure. The naming convention are directly inherited from the MITgcm (see this online documentation).

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)

indexing convention is illustrated for the LLC90 grid in Fig. 3. For a vector field the first component (U) points strait to the right of the page in Fig. 3, whereas the second component (V) points strait to the top of the page. The location of U components are shifted by half a grid point towards the left of the page, while the location of V components are shifted by half a grid point towards the bottom of the page (reflecting the C-grid approach).

3.2 Exchange Functions

Many quantities of interests (e.g., budgets) involve values from neighboring grid points that often need to be 'exchanged' between faces. This is achieved in practice by appending rows and columns at the sides of each face that are obtained from the neighboring faces – appending rows and columns from faces #2, 3, and 5 at the sides of face #1 in the case of Fig. 3 for example. These exchanges are operated by `exch_T_N.m` for tracer fields and by `exch_UV_N.m` for velocity fields. These functions are needed for example to compute temperature gradients (with `calc_T_grad.m`) and flow convergences (with `calc_UV_conv.m`) as illustrated in section 4.

3.3 Overloaded Functions

Fig. 2 depicts the 'overloading' of the '+' operation by `@gcmfaces/plus.m` . In executing commands such as 'fld+1', Matlab will use `@gcmfaces/plus.m` if one of the arguments of '+' (i.e. sum) is of the gcmfaces class. Many common operations and functions are similarly overloaded in the '@gcmfaces/' directory that defines the gcmfaces class and its operations:

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 is worth mentioning the case of `@gcmfaces/subsasgn.m` (subscripted assignment) and `@gcmfaces/subsref.m` (subscripted reference) since they are 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;' respectively call subsref.m and subsasgn.m. If fld is of the gcmfaces class instead 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).
```

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

3.4 I/O Functions

Objects of the gcmfaces class can simply be saved to or read from file in Matlab's own I/O format (.mat files). The other file formats that are currently supported in the gcmfaces framework are: (1) the MITgcm binary formats documented here; (2) the nctiles format used to distribute ECCO v4 fields (Forget et al., 2015). When reading from such files, the provided I/O functions (rdmds2gcmfaces.m and read_nctiles.m, respectively) reformat the input into gcmfaces objects on the fly.

4 Tutorial

Here it is assumed that the user has completed the installation procedure in section 1.3 (including the installation of 'nctiles_climatology/' and 'm_map/').

gcmfaces_demo.m can then be executed that provides examples of a few of the gcmfaces capabilities. As prompted by gcmfaces_demo.m : specify the desired amount of explanatory text output. gcmfaces_demo.m) will then proceed through the examples while displaying explanations in the Matlab command window. The Matlab GUI and debugger can also be used to follow the examples step by step.

The first section of gcmfaces_demo.m illustrates plotting capabilities. gcmfaces relies on m_map (https://www.eoas.ubc.ca/ rich/map.html) for geographical projections through the m_map_gcmfaces front-end that typically produces Fig.4. The convert2pcol function provides an alternative to display results directly via 'pcolor' (Fig. 5). The second section of gcmfaces_demo.m focuses on data processing capabilities such as interpolation and smoothing. example_smooth.m in particular integrates a diffusion equation, which illustrates computations of tracer gradients (calc_T_grad.m) and flux convergences (calc_UV_conv.m) . The third section of gcmfaces_demo.m illustrates computations of oceanic transports and stream-functions (example_transports.m).

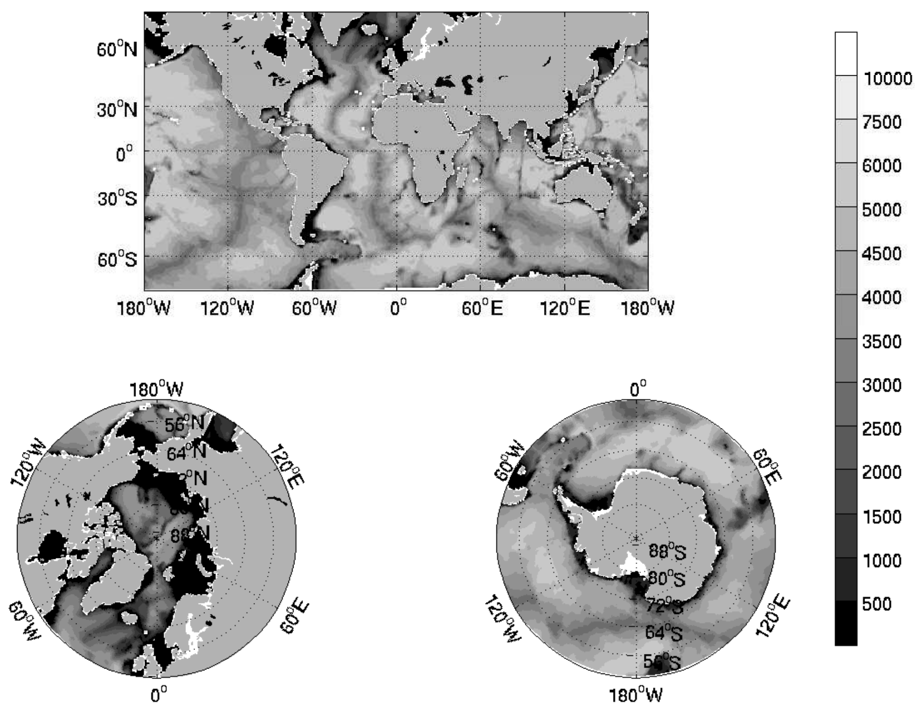


Figure 4: Same as Fig.3 but plotted in geographical coordinates using m_map_gcmfaces.m

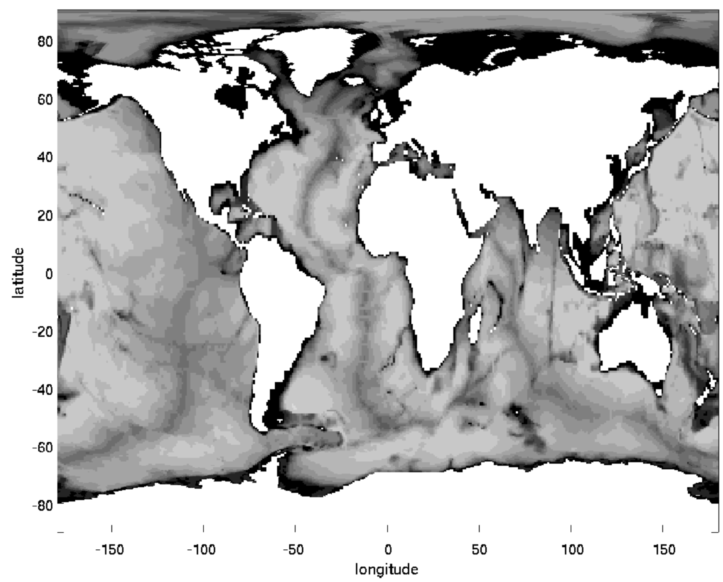


Figure 5: Same as Fig.3 but plotted in geographical coordinates using convert2pcol.m

5 Standard Analysis

The gcmfaces standard analysis consists of an extensive set of physical diagnostics that are routinely monitored in MITgcm simulations and ECCO v4 estimates (Forget et al., 2015). The computational loop is operated by `diags_driver.m` that stores the results in a dedicated directory ('mat/' in Fig.1). The display phase is done afterwards by calling `diags_display.m` (simple display to screen) or `diags_driver_tex.m` (to generate a tex file).

Here it is assumed that the user has completed the installation procedure in section 1.3 (including the installation of 'nctiles_climatology/' and 'm_map/'). The code below then generates mean and variance maps (setDiags='B' encoded in `diags_set_B.m`) from the ECCO v4 monthly mean climatology (12 monthly fields), which should take about 5 minutes:

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

%compute diagnostics:
help diags_driver;
dirModel='release1/';
dirMat=[dirModel 'mat/'];
setDiags='B';
diags_driver(dirModel,dirMat,'climatology',setDiags);

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

Each set of diagnostics (computation and display) is encoded in one routine with a name such as 'diags_set_XX.m' (here 'XX' is just a placeholder). These routines can be found in the 'gcmfaces_diags/' directory. Sets of diagnostics that can be generated using 'nctiles_climatology/' include oceanic transports ('A'), mean and variance maps ('B'), sections and time series ('C'), and mixed layer depths ('MLD').

If the 'setDiags' argument to `diags_driver.m` is omitted then the four diagnostic sets will be generated at once, which should takes about 1/2 hour. As this generates a large number of plots, one may prefer to generate a tex file containing all of the plots, which should take another 10 minutes:

```
%compute more diagnostics:
dirModel='release1/'; dirMat=[dirModel 'mat/'];
diags_driver(dirModel,dirMat,'climatology');

%generate a tex file containing all of the plots:
dirTex=[dirModel 'tex/']; nameTex='standardAnalysis';
diags_driver_tex(dirMat,{},dirTex,nameTex);
```

These diagnostics can also be generated for the full ECCO v4 time series: ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/nctiles/ after downloading this directory (243G) and placing it next to 'nctiles_climatology/' in Fig. 1. Since the 20 year time series consists of 240 monthly records, the computation is usually distributed over multiple processors (e.g. each processor processing one of the years) or done overnight with:

```
 diags_driver(dirModel,dirMat,[1992:2011]);
```