gcmfaces Documentation

Release 0.1.0

Gael Forget

Nov 08, 2018

Contents

1 Getting Started **3**
- 1.1 Install Software . . . 3
- 1.2 Obtain Input Data . . . 3
- 1.3 Activate gcmfaces . . . 4

2 Basic Features **7**
- 2.1 The gcmfaces Class . . . 7
- 2.2 Handling C-Grids . . . 9
- 2.3 Exchange Functions . . . 10
- 2.4 Overloaded Functions . . . 10
- 2.5 Input / Output Files . . . 11

3 Tutorial Examples **13**

4 Standard Analysis **15**

5 Sample grids **17**

6 Indices and tables **19**

Bibliography **21**

Here, you will learn about the *gcmfaces* toolbox that provides a generic treatment of gridded Earth variables in Matlab and Octave.

The *gcmfaces* toolbox handles gridded Earth variables as sets of connected arrays. This object-oriented approach allows users to write generic, compact analysis codes that readily become applicable to a wide variety of grids (e.g., those in Figure 5.1). *gcmfaces* notably allows for analysis of MITgcm output on any of its familiar grids. It was originally developed as part the *ECCO version 4* framework along with the companion *MITprof* toolbox that handles unevenly distributed in-situ ocean observations *[FCH+15]*.

This user manual provides an installation guide for *gcmfaces* and *MITprof* (Section 1), a documentation of the basic *gcmfaces* features (Section 2), and an overview of higher-level *gcmfaces* functionalities for mapping, transport, etc. operations (Section 3 and Section 4).

CHAPTER 1

Getting Started

1.1 Install Software

Download the latest software version from github by typing

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

at the command line or using the github web browser interface. This method allows users to update the software later on and to manage their own, if any, code modifications. Archived frozen versions of the software, which can be cited in publications using permanent digital object identifiers, are also available via zenodo. Additionally, *gcmfaces* relies on the *m_map* toolbox for geographic projections (Figure 3.1), which can be downloaded from this webpage (e.g., m_map1.4.tar.gz).

Octave users will want to replace `git clone ...faces` with `git clone -b octave ...faces` in the above recipe. They will also need to install and load the Octave statistics, io, and netcdf packages.

Note: *MITprof* is not generally needed by *gcmfaces*, but is used in Section 3 and Section 4.

1.2 Obtain Input Data

The *gcmfaces* toolbox allows users to seamlessly deal with various gridding approaches (e.g., all grids distributed via this FTP server) using compact and generic codes as explained in this user guide. Once a grid has been loaded to memory (see below and Section 2.2), *gcmfaces* can be used to analyze ocean model solutions and state estimates on that grid (Section 3 and Section 4).

To get started in Section 1.3 and Section 2, it suffices to download `nctiles_grid/` (145M) either from this ftp server or from this permanent archive. Section 3 and Section 4 use `nctiles_climatology/` (10G) to illustrate higher-level functionalities. One download method, from the command line, is shown in *Demo Directory Downloads*. Commands reported afterwards assume that downloaded contents are organized as shown in *Demo Directories Organization*.

The other input data sets shown in *Demo Directories Organization* (inside of `release2/`) are not be needed unless user wants to reproduce the full set of plots in *[FCH+16]*. The contents of `profiles/` (7G) and `nctiles_remotesensing/` (27G) allow for model-data comparisons, while `nctiles_monthly/` (170G) contains monthly time series of ocean variables over 1992-2011. These can be used to reproduce the plots in *[FCH+16]* via a few function calls as explained at the end of Section 4.

Demo Directory Downloads

```
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
```

Demo Directories Organization

```
gcmfaces/ (Matlab / Octave toolbox)
MITprof/ (Matlab / Octave toolbox)
m_map/ (Matlab / Octave toolbox)
nctiles_grid/ (downloaded data)
release2_climatology/
    nctiles_climatology/ (downloaded data)
    mat/ (created by software)
    tex/ (created by software)
release2/
    nctiles_monthly/ (downloaded data)
    nctiles_remotesensing/ (downloaded data)
    profiles/ (downloaded data)
    mat/ (created by software)
    tex/ (created by software)
```

1.3 Activate gcmfaces

Once `gcmfaces/` and `nctiles_grid/` have been placed in a common directory as shown in *Demo Directories Organization*, open Matlab or Octave from within 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);
```

(continues on next page)

(continued from previous page)

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

CHAPTER 2

Basic Features

The core of *gcmfaces* lies in (1) its representation of connected arrays, or faces, as objects of the class defined in the `@gcmfaces/` directory (see Section 2.1) and (2) its handling of C-Grid specifications via the *mygrid* global structure (Section 2.2). Other basic features include functions that *exchange* data between faces (Section 2.3), or *overload* common operations (Section 2.4), as well as I/O routines (Section 2.5). These and other *gcmfaces* functions are generally documented through help sections that are readily accessible via the Matlab or Octave command window.

2.1 The gcmfaces Class

Figure 5.1 illustrates four types of grids that are have been used in ocean general circulation models. Despite evident design differences, these grids can all be represented as sets of connected arrays, or faces, as shown in Figure 2.1 in the case of the LLC90 grid. *gcmfaces* simply takes advantage of this fact by defining a class for these objects, within `@gcmfaces/`, that represents gridded earth variables generically as sets of connected arrays.

Grid specifics, such as the number of faces and the size of each face, are embedded within the *gcmfaces* objects (see *Gcmfaces object structure*). 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 (Figure 5.1). The *gcmfaces* class inherits many of its basic operations from the *double* class as illustrated for the *Overloaded + function* (see Section 2.4 for details).

Gcmfaces object structure

An Earth variable on the LLC90 grid (Figure 5.1, bottom right) stored as a *gcmfaces* object called *fld* has the data structure depicted below. In this example, *fld* is a two dimensional field, and the five face arrays plotted in Figure 2.1 are denoted as f1 to f5.

```
fld
  nFaces: 5
  f1: [90x270 double]
  f2: [90x270 double]
  f3: [90x90 double]
```

(continues on next page)

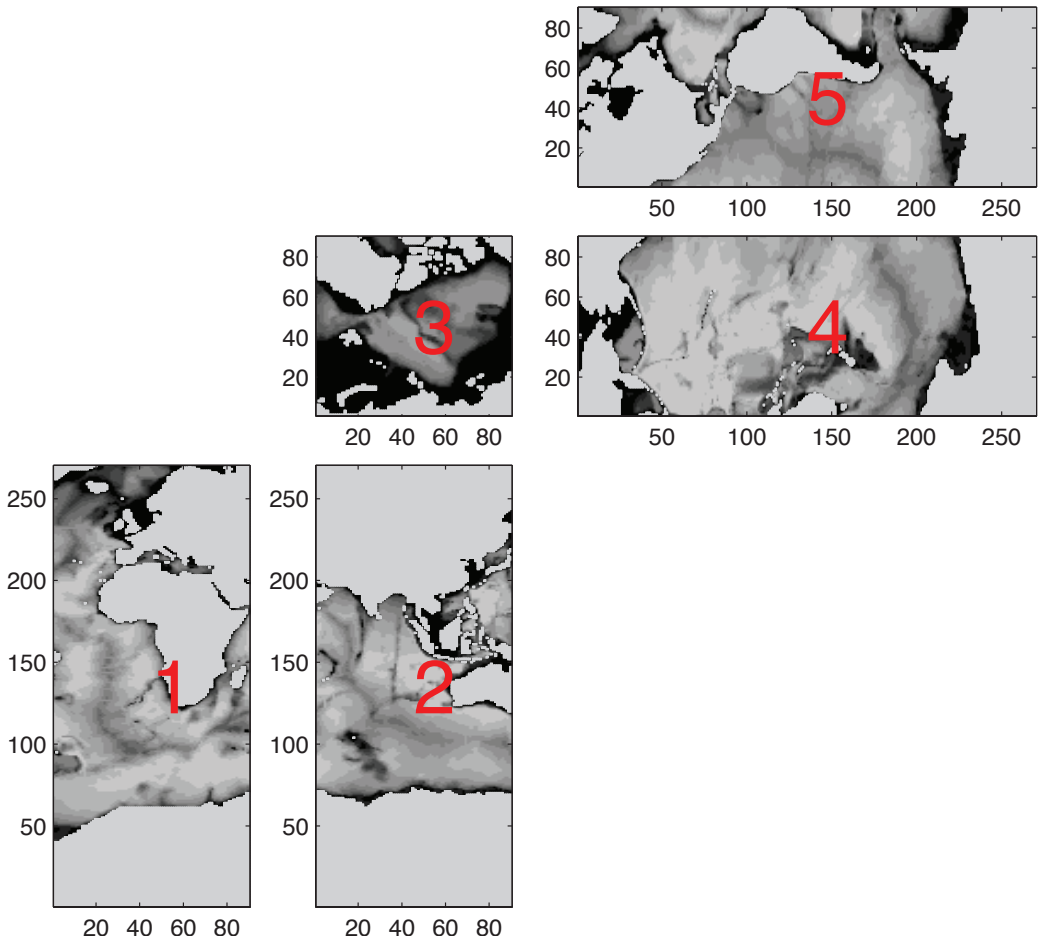

Figure 2.1: Ocean topography on the LLC90 grid (Figure 5.1, bottom right) displayed face by face (going from 1 to 5). This plot generated using example_display(1) illustrates how gcmfaces organizes data in memory (see *Gcmfaces object structure*). Within each face, grid point indices increase from left to right and bottom to top.

(continued from previous page)

```
  f4: [270x90 double]
  f5: [270x90 double]
```

2.2 Handling C-Grids

In practice *gcmfaces* gets activated by adding, to the least, the `@gcmfaces/` directory to the Matlab path and then loading a grid to memory (Section 1.3). The default grid is LLC90, which can be loaded to memory by calling `grid_load.m` without any argument. Section 2.5 and `help grid_load;` provide additional information regarding, respectively, and supported file formats and `grid_load.m` arguments. As an alternative to `grid_load.m`, *MITgcm* input grid files can be read `grid_load_native.m` as shown here (see README and `demo_grids.m`).

Both `grid_load.m` and `grid_load_native.m` store all C-grid variables at once in a global variable named *mygrid* (Table 2.1). *gcmfaces* functions then rely on *mygrid* that they get access to by calling `gcmfaces_global.m` which also returns system information via *myenv*. If these global variables get deleted at some point, for example by a call to `clear all;`, user may need to rerun `grid_load.m` or `grid_load_native.m`. In such situtations, any call to `gcmfaces_global.m` will generate a warning that *mygrid has not yet been loaded to memory*.

Table 2.1: List of grid variables available via the mygrid global variable. The naming convention is directly inherited from the *MITgcm* naming convention[1].

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 Table 2.1 derive from the *MITgcm* naming convention[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 the C-grid approach[1].

Indexing and vector orientation conventions also derive from *MITgcm* conventions[1]. The indexing convention is illustrated in Figure 2.1. 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

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

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 nearby grid points on neighboring faces. In practice rows and columns need to be appended at each face edge that are *exchanged* between neighboring faces – e.g., rows and columns from faces #2, #3, and #5 need to be appended at the face #1 edges in Figure 2.1. Exchanges are operated by `exch_T_N.m` for tracer-type variables and by `exch_UV_N.m` for velocity-type variables. These are notably used to compute gradients (`calc_T_grad.m`) and flow convergences (`calc_UV_conv.m`).

2.4 Overloaded Functions

As in the case of the *Overloaded + function*, common operations and 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.

Overloaded + function

```
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')
```

(continues on next page)

(continued from previous page)

```
   end;
end;
```

2.5 Input / Output Files

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/` to the Matlab path.

Alternatively, gcmfaces variables can be written to files in the *nctiles* format *[FCH+15]*. Illustrations in this user guide rely upon ECCO version 4 fields which are distributed in this format (see Section 1.2; *Demo Directories Organization* and *Demo Directory Downloads*). The associated I/O functions provided in *gcmfaces* (`write2nctiles.m` and `read_nctiles.m`) reformat data on the fly.

Finally, *gcmfaces* can 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 convert *mds* output to *gcmfaces* objects on the fly. The reverse conversion occurs when `convert2gcmfaces.m` is called with a *gcmfaces* input argument. This approach provides a unified framework to analyze MITgcm output or prepare MITgcm input for all known grids (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

CHAPTER 3

Tutorial Examples

To proceed with the tutorial examples, user is expected to have completed the software installation and data downloads from Section 1 (including `nctiles_climatology/` and `m_map/`). The full suite of tutorial examples can then be executed via `gcmfaces_demo.m` by opening Matlab and typing

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

As prompted by `gcmfaces_demo.m`, specify the desired amount of explanatory text output. Various examples then proceed and display comments in the Matlab or Octave command window. The Matlab GUI and debugger can also be used to run each example line by line. This can be useful 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`). The second section focuses on data processing capabilities such as interpolation and smoothing. `example_interp.m` interpolates 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`).

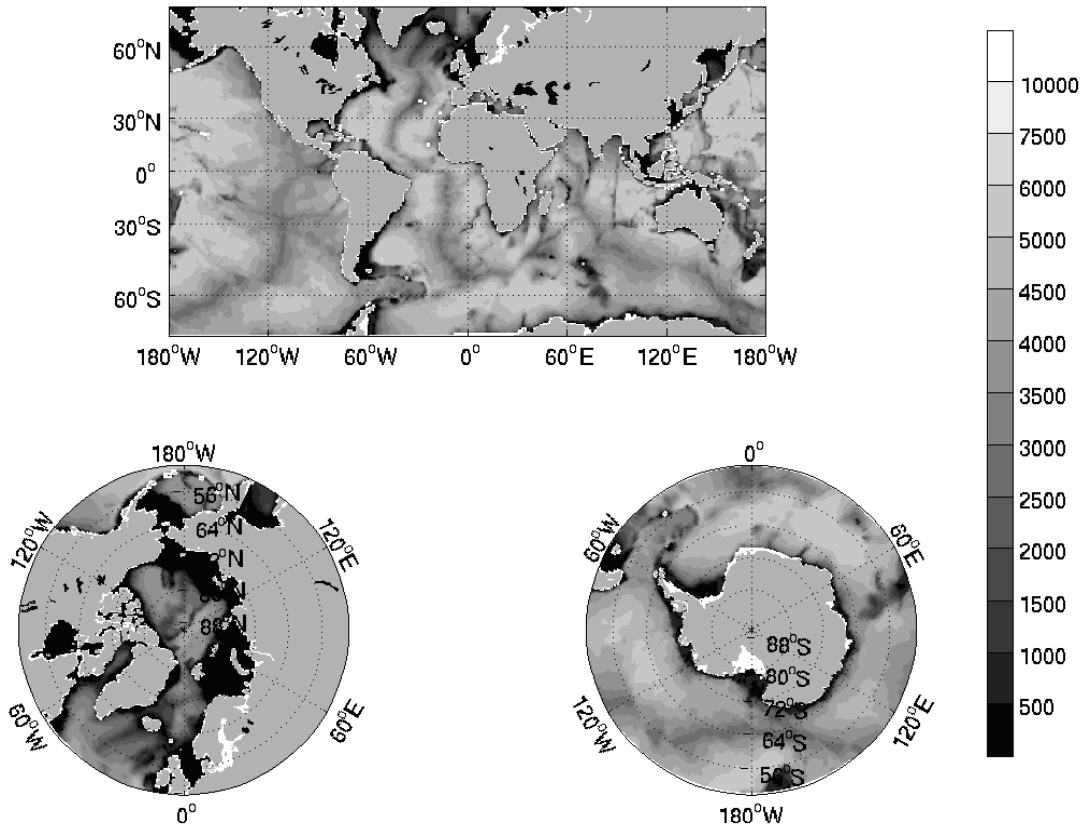

Figure 3.1: Same as Figure 2.1 but plotted in geographical coordinates using `m_map_gcmfaces.m`. This plot is generated by calling example_display(4).

CHAPTER 4

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 *[FCH+15]*, *[FCH+16]*. The computational loop is operated by `diags_driver.m` which stores intermediate results in a dedicated directory (`mat/` in *Demo Directories Organization*). Afterwards, the display phase is operated via `diags_display.m` or `diags_driver_tex.m` as explained below.

In order to proceed, user should have completed the installation procedure in Section 1 and organized directories as shown in *Demo Directories Organization*. They can then, for example, generate and display variance maps from the ECCO v4 monthly mean climatology (12 monthly fields) by opening Matlab and executing `diags_set_B.m` as follows:

```
%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 should take $\approx$ 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 should take $\approx$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 all four diagnostic sets from ECCO v4 r2 climatology takes ≈1/2 hour. Computing them from the 1992-2011 monthly time series (`nctiles_monthly/` in *Demo Directories Organization*) by typing

```
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.

Note: The above `diags_driver` calls rely on default parameters that are adequate for the Section 1 solution, but yours may differ. Using the `doInteractive` option (see `help diags_driver`) is therefore the generally recommended method, since it gives you the opportunity to review and, if needed, edit the relevant parameters.

CHAPTER 5

Sample grids

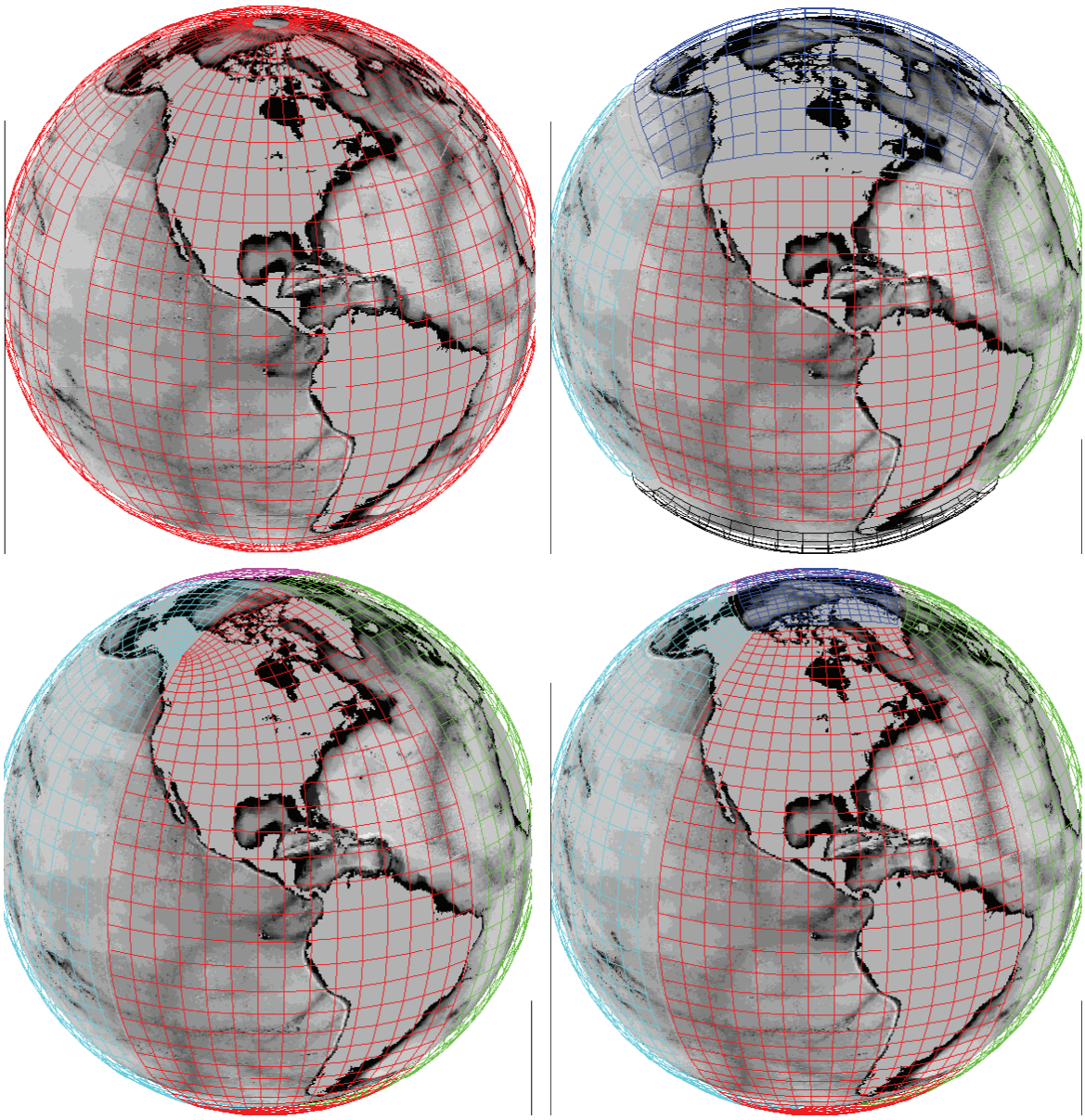

Figure 5.1: Four approaches to gridding the Earth which are all commonly used in numerical models. Top left: lat-lon grid; mapping the Earth to a single rectangular array (*face*). Top right: cube-sphere grid; mapping the earth to the six faces of a cube. Bottom right: lat-lon-cap, *LLC*, grid (five 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.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Bibliography

[FCH+15] G. Forget, J.-M. Campin, P. Heimbach, C. N. Hill, R. M Ponte, and C. Wunsch. ECCO version 4: an integrated framework for non-linear inverse modeling and global ocean state estimation. *Geoscientific Model Development*, 8(10):3071–3104, 2015. URL: http://www.geosci-model-dev.net/8/3071/2015/, doi:10.5194/gmd-8-3071-2015.

[FCH+16] G. Forget, J.-M. Campin, P. Heimbach, C. N. Hill, R. M Ponte, and C. Wunsch. ECCO version 4: second release. 2016. URL: http://hdl.handle.net/1721.1/102062.