ECCO v4 development notes

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

August 17, 2015

abstract

These notes pertain to the ECCO v4 state estimate, model setup, and associated codes (Forget et al., 2015). Section 1 points to the other elements of documentation that are available online, and associated download procedures. Section 2 provides guidance to ECCO v4 users interested in operating the ECCO v4 model set-up and/or reproducing the ECCO v4 solution. Section 3 documents the re-implemented estimation modules of MITgcm. Some of the included material in section 3 is expected to eventually move to the MITgcm manual. Throughout this document I try to rely on pre-existing documents rather than duplicating them. Links to pre-existing documents are indicated by blue colored font (e.g. 'manual' in the previous sentence).

Contents

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 Discussions*, **8 (5)**, 3653–3743, doi:10.5194/gmdd-8-3653-2015, URL http://www.geosci-model-dev-discuss.net/8/3653/2015/.

1 downloads

This section documents locations and directions to download the MITgcm (section 1.1), the ECCO v4 model setup (section 1.2), the ECCO v4 state estimate output (section 1.3), and related diagnostic matlab tools (section 1.4).

1.1 MITgcm

To install the MITgcm:

- Go to the MITgcm web-page @ mitgcm.org
- Install MITgcm using cvs as explained @ cvs
- Run MITgcm using testreport as explained @ manual, howto

Pre-requisites are cvs, gcc, gfortran (or alternatives), and mpi (only for parallel runs). For example, my laptop setup, including mpi and netcdf, involved the following mac ports:

- cvs @1.11.23_1 (active)
- wget @1.14_5+ssl (active)
- gcc48 @4.8.2_0 (active)
- mpich-default @3.0.4_9+gcc48 (active)
- mpich-gcc48 @3.0.4_9+fortran (active)
- netcdf @4.3.0_2+dap+netcdf4 (active)
- netcdf-fortran @4.2_10+gcc48 (active)

Overridding the default mac gcc and mpich with the above requires:

- sudo port select –set gcc mp-gcc48
- sudo port select –set mpich mpich-gcc48-fortran

Using mpi and netcdf within MITgcm requires two environment variables:

- export MPI_INC_DIR=/opt/local/include
- export NETCDF_ROOT=/opt/local

1.2 ECCO v4 setup

Any MITgcm user can easily install the ECCO v4 setups using the setup_these_exps.csh shell script as explained @ README. It downloads global_oce_cs32/ (small setup), global_oce_llc90/ (bigger setup) and model inputs from global_oce_input_fields.tar.gz to a subdirectory called global_oce_tmp_download/. The user then wants to move its contents to MITgcm/verification/ (as shown in Fig.1) in order to allow for automated execution of the short benchmark runs via testreport using genmake2 (see section 2.1). Pre-requisites: having downloaded MITgcm (section 1.1) and mpi libraries (only if user wants to run the bigger global_oce_llc90/).

The short benchmarks are ran on a daily basis to ensure continued compatibility with the up to date MITgcm. While the short benchmarks only go for a few time steps, global_oce_llc90/ also is the basic setup that produces the 1992-2011 ECCO v4 ocean state estimate (Forget et al., 2015) when configured accordingly (as explained in section 2.3). Thus running the short benchmarks (section 2.1) is a useful step towards re-producing the state estimate (section 2.3). It should also be noted that an adjoint version of the short benchmarks also exist that can readily be run by users who access to the TAF compiler.

Figure 1: MITgcm directory structure downloaded using cvs. The ECCO v4 directories indicated with "+" were downloaded separately using setup_these_exps.csh script and moved to MITgcm/verification/.

```
MITgcm/
 |__model/ (core of MITgcm)
 |__pkg/ (MITgcm modules)
 |__verification/
 |   |__testreport (shell script)
 |   |__aim.5l_cs (mitgcm regression test)
 |   |__+ global_oce_cs32/ (for laptops)
 |   |__+ global_oce_llc90/ (for computers)
 |   |__+ global_oce_input_fields/ (inputs)
 |   |__hs94.128x64x5 (mitgcm regression test)
 |   |__...
 |__tools/
     |__genmake2 (shell script)
     |__build_options (wrt compilers)
     |__...
```

1.3 ECCO v4 solution

The state estimate output for ECCO v4-release 1 is available via this server which is linked to ecco-group.org. The various subdirectories contain monthly fields, this documentation of the solution, in situ and model profiles, the grid specifications and ancillary data as explained in README.docx. For example a file (or a subdirectory) can be downloaded at the command line e.g. per

```
wget --recursive ftp://mit.ecco-group.org/ecco_for_las/version_4/release1/README.docx
```

1.4 Diagnostic Tools

To help ECCO v4 and MITgcm users analyze model output obtained either per section 1.3 or per section 2.3, two sets of Matlab tools are made freely available:

- download gcmfaces and MITprof using shell script (or see getting_started.m)
- download MITgcm/utils using cvs (basic functionalities only).

Any user can for example regenerate this documentation of the solution (the gcmfaces 'standard analysis') from the section 1.3 or section 2.3 output (expectedly organized according to Fig.2) simply by executing diags_driver.m [1] and diags_driver_tex.m [2] in the following sequence :

```
dirModel='release1_20150603_c65l/';
dirMat='release1_20150603_c65l/mat/';
dirTex='release1_20150603_c65l/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
```

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

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

Figure 2: Directory structure as expected by 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 standard analysis). This diagnostic process relies on the depicted organization of GRID/ and solution/ for automation (user will otherwise be prompted to enter directory names) and depends on downloaded copies of fields to nctiles/ (local subdirectory).

```
./
├── 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)
├── release 1 solution/
│   ├── diags/ (binary output)
│   ├── nctiles/ (netcdf output)
│   ├── MITprof/ (netcdf output)
│   ├── mat/ (created by gcmfaces)
│   └── tex/ (created by gcmfaces)
├── other solution/
│   ├── diags/ (binary output)
│   └── ...
└── ...
```

2 MITgcm runs

The following procedures, commands and submission scripts allow runs of the ECCO v4 MITgcm setup – either in short regression tests (section 2.1) or for multi-decadal simulations such as the full 20 year state estimate (section 2.3). Pre-requisite for sections 2.1 and 2.3: having downloaded the MITgcm (section 1.1) and the ECCO v4 setups (section 1.2). Pre-requisite for section 2.3: having downloaded forcing fields and a few other binary model inputs (listed below).

2.1 regression tests

Short benchmarks of the MITgcm and ECCO v4 setup are run using testreport command line utility (see Fig.2; howto). Serial runs are executed simply at the command line e.g. per

```
./testreport -t global_oce_cs32
```

or

```
./testreport -skipdir global_oce_llc90
```

The reader is referred to 'testreport –help' and howto for additional explanation about such commands. If everything proceeds as expected then the result of the comparison with the reference result is reported to screen as shown in abbreviated form in Fig. 3. Depending on your machine environment the agreement with the reference result may be lower in which case 'testreport' may indicate 'FAIL' (e.g. see README). Despite the dramatic character of such message, this is generally ok and does not prevent reproducing full model solutions accurately (see section 2.3). If the testreport process gets interrupted then it is often safer to clean up experiment directories (e.g., by executing ./testreport -clean -t global_oce_*) and start over.

```
default 10  ----T-----  ----S-----
G D M    c        m  s        m  s
e p a R  g  m  m  e  .  m  m  e  .
n n k u  2  i  a  a  d  i  a  a  d
2 d e n  d  n  x  n  .  n  x  n  .

Y Y Y Y>14<16 16 16 16 16 16 16 16  pass  global_oce_cs32
```

Figure 3: Abbreviated output of testreport to screen.

The above 'testreport' commands deserve a couple more specific comments. The first command runs the global_oce_cs32/ benchmark solely. The second command will run all MITgcm benchmarks including global_oce_cs32/ but not the global_oce_llc90/ benchmark that requires at least 12 processors in forward (96 in adjoint) and therefore should not be run in serial mode (doing so may crash your laptop). It is thus excluded by using the 'skipdir' option. It should be stressed however that global_oce_cs32/ depends on the files in global_oce_llc90/ (which is the main setup) rather than duplicating them. Therefore global_oce_llc90/ must not be removed from MITgcm/verification for global_oce_cs32/ to work.

Running the short benchmarks with mpi (assuming it has been installed) is equally simple:

```
./testreport -of ../tools/build_options/linux_amd64_ifort+mpi_ice_nas \
-j 4 -MPI 96 -command 'mpiexec -np TR_NPROC ./mitgcmuv' \
-t global_oce_llc90
```

for example will run the first forward benchmark of global_oce_llc90/ on 96 processors using an ifort compiler. Note that the specifics (number of processors and compiler choice) are to be determined by the user and are machine dependent.

Often in massively parallel computing environments, it is common that mpi jobs can only be run within a queuing system. The submission script in Fig.4 (that is also machine specific) provides an example on how to do it. It contains 3 hard-coded switches : fwdORad = 1 (2 for adjoint); numExp = 1 (2 for llc90); excludeMpi = 0 (1 for serial). This script should be located and submitted from MITgcm/verification. It is also common that compute nodes cannot access certain compilers, in which case the user may want to proceed in two steps:

1. compile outside of the queuing system using e.g. per

   ```
   ./testreport -of ../tools/build_options/linux_amd64_ifort+mpi_ice_nas \
   -j 4 -MPI 96 -command 'mpiexec -np TR_NPROC ./mitgcmuv' \
   -t global_oce_llc90 -norun
   ```

2. submit the Fig.4 script, after adding -q to the 'opt' variable to skip compilation.

Running adjoint benchmarks requires access to the TAF compiler. The calls to testreport (see above) then only need to be slightly altered by appending the '-ad' option (for either serial or mpi jobs) and replacing 'mitgcmuv' with 'mitgcmuv_ad' (only for mpi jobs). It should also be noted that, unlike other MITgcm benchmarks, global_oce_cs32/ and global_oce_llc90/ do not include any adjoint specific 'code_ad/' directory as they simply use the forward model 'code/' directory instead. Since testreport relies on the existence of 'code_ad/' for its adjoint option though, it is necessary to soft link 'code/' to 'code_ad/' in both global_oce_cs32/ and global_oce_llc90/ accordingly in order to to run their 'testreport -ad' versions.

Figure 4: Example script to run mpi testreport via a queueing system (machine dependent).

```
#PBS -S /bin/csh
#PBS -l select=1:ncpus=16:model=ivy+4:ncpus=20:model=ivy
#PBS -l walltime=02:00:00
#PBS -q devel
#PBS -m n

#environment variables and libraries
#-----------------------------------------------
limit stacksize unlimited
module purge
module load modules comp-intel/2013.1.117 mpi-sgi/mpt.2.10r6 netcdf/4.0
#
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${HOME}/lib
setenv MPI_IB_TIMEOUT 20
setenv MPI_IB_RAILS 2
setenv MPI_IB_FAILOVER 1
setenv  MPI_CONNECTIONS_THRESHOLD  2049

#local variables and commands
#----------------------------------------
set fwdORad = 1
set numExp  = 1
set excludeMpi = 0
#
if ( ${numExp} == 1) then
  set nameExp = global_oce_cs32
  set NBproc  =  6
else
  set nameExp = global_oce_llc90
  set NBproc  = 96
endif
#
if ( ${excludeMpi} == 1 ) then
  set opt = '-of ../tools/build_options/linux_amd64_ifort -j 4'
else
  set opt = '-of ../tools/build_options/linux_amd64_ifort+mpi_ice_nas -j 4'
endif
#
if ( ${fwdORad} == 1 && ${excludeMpi} == 0 ) then
  ./testreport ${opt} -MPI \
  ${NBproc} -command 'mpiexec -np TR_NPROC ./mitgcmuv' -t ${nameExp}
else if ( ${fwdORad} == 2 && ${excludeMpi} == 0 ) then
  ./testreport ${opt} -MPI \
  ${NBproc} -command 'mpiexec -np TR_NPROC ./mitgcmuv_ad' -ad -t ${nameExp}
else if ( ${fwdORad} == 1 && ${excludeMpi} == 1 ) then
  ./testreport ${opt} -t ${nameExp}
else if ( ${fwdORad} == 2 && ${excludeMpi} == 1 ) then
  ./testreport ${opt} -ad -t ${nameExp}
endif

exit
```

2.2 Iterative optimization test case

The global_oce_cs32/input_OI implements an iterative optimization test case. It case boils down to optimal interpolation (the model dynamics are not involved) solved by a variational method using the MITgcm adjoint (a diffusion equation in this test case). The pre-requisites are:

1. run the adjoint benchmark in global_oce_cs32/ via testreport (see section 2.1).
2. Go to MITgcm/lsopt and compile (see section 3.18 of manual).
3. Go to MITgcm/optim, replace 'natl_box_adjoint' with 'global_oce_cs32' in this Makefile, and compile as explained in section 3.18 of manual. An executable named 'optim.x' should get created in MITgcm/optim. If otherwise, please contact ecco-support@mit.edu
4. go to MITgcm/verification/global_oce_cs32/input_OI and type 'source ./prepare_run'

Then the iterative optimization itself proceeds as follows

1. ./mitgcmuv_ad > output.txt
2. ./optim.x > op.txt
3. increment optimcycle by 1 in data.optim
4. go back to step #1, to run the next iteration
5. type 'grep fc costfunction000*' to display results (Fig. 5).

```
costfunction0000: fc =   4118.1987222194211        0.00000000
costfunction0001: fc =   1523.9310891186672        0.00000000
costfunction0002: fc =   1053.3611790049420        0.00000000
costfunction0003: fc =   790.10479375339185        0.00000000
```

Figure 5: Results of iterative optimization after 3 iterations carried out as explained in section 2.2.

2.3 full ECCO v4 runs

The 1992-2011 ECCO v4 ocean state estimate (Forget et al., 2015) is reproduced on a monhtly basis to ensure continued compatibility with the up to date MITgcm. Re-running the baseline 20 year solution (or any other 20 year of global_oce_llc90/) on 96 processors may take about 8 to 12 hours (depending on the computing environment). Reproducing the state estimate requires additional input to be downloaded (besides section 1.2; see below). Unlike for the short benchmarks of section 2.1, in the case of these longer model runs:

- the model is compiled and run outside of testreport.
- the model is compiled with compiler optimization.

- additional forcing and binary input is necessary.
- additional memory and/or disk space is necessary.

The reader is referred to howto for a general explanation of such practice. The typical compilation sequence for the ECCO v4 forward model (i.e. the model setup in global_oce_llc90/) is shown in Fig.6. The tamc.h_itXX and PROFILES_SIZE.h_itXX headers (see Fig.6) allow for additional time steps and in situ profiles input, respectively. Once done with compilation, the user typically creates and enters a run directory, links the model executable and inputs into place (see Fig.7), and submits a job to the queueing system (see Fig.8). The 'input_itXX/prepare_run' script (Fig.7) makes a local copy of the model executable ('mitgcmuv'), of all namelists ('data*' from the various 'input*/' directories downloaded in section 1.2) that set up the model at run time, and links a few binary inputs such as the grid and bathymetry files. Assuming that the forcing, observations, model parameter adjustments and initial condition directories were populated and the script was edited (user need to specify forcingDir, obsDir, ctrlDir and pickDir in Fig.7) accordingly then it will furthermore link their contents in the run directory. Users interested in obtaining the necessary input files are advised to contact ecco-support@mit.edu regarding:

- 6 hourly forcing files over 1992-2011 (EIG*199? EIG*20??).
- insitu data sets (*feb2013*.nc) used in long benchmark (see below).
- model parameter adjustsments, a.k.a. the control vector (xx_*).
- the initial conditions (pickup*)

Once the model run has completed, one wants to verify that it accurately reproduces the reference result – or detect that a mistake was made. To this end, a mechanism that is analogous to testreport but is geared towards benchmarking long runs was introduced by Forget et al. (2015). It is operated by testreport_ecco.m within Matlab. The pre-requisite is to add the reference result directory 'MITgcm/verification/global_oce_llc90/results_itXX/' to the Matlab path. As explained in Forget et al. (2015) testreport_ecco.m compares time series of global mean variables, and other characteristics of the solution, to the reference state estimate values. The array of tests can be extended to e.g. meridional transports by adding gcmfaces to the Matlab path. The typical call sequence is indicated in the help of testreport_ecco.m and in Fig.9 that also illustrates the typical display of the benchmarking results report to the user screen. The expected level of accuracy for re-runs of the baseline 20 year solution (with an up to date MITgcm code on any given computer) is reached when the displayed values are < -4 (see Forget et al., 2015, for details). In cases when some of the tests were omitted (e.g. because gcmfaces was not in the Matlab path) the display will show NaN for omitted tests. From the generated model output, one may further easily compute and display many diagnostic quantities using the gcmfaces standard analysis for example (see section 1.4).

Figure 6: Compilation directives, outside testreport, for intensive model runs. On a different machine (computer) another build option file such as linux_amd64_gfortran or linux_amd64_ifort11 should be used. To compile the adjoint, users need a TAF license and to replace ‘make -j 4’ with ‘make adall -j 4’. Note : the ‘-mods=../code’ specification can be omitted if the build directory contains the ‘genmake_local’ file).

```
cd verification/global_oce_llc90/build
../../../tools/genmake2 -optfile=\\
../../../tools/build_options/linux_amd64_ifort+mpi_ice_nas -mpi -mods=../code

make depend

\rm tamc.h PROFILES_SIZE.h
cp ../code/tamc.h_itXX tamc.h
cp ../code/PROFILES_SIZE_itXX PROFILES_SIZE.h

make -j 4
```

Figure 7: Example script to setup the 20 year ECCO v4 state estimate. It is implied that user has filled directories /bla, /blaa, /blaaa and /blaaa with appropriate forcing, observational, control vector, and pickup files.

```
#!/bin/csh -f

set forcingDir  = ~/bla
set obsDir      = ~/blaa
set ctrlDir     = ~/blaaa
set pickDir     = ~/blaaaa

source ../input_itXX/prepare_run
cp ../build/mitgcmuv .
\rm pick*ta EIG*
ln -s ${forcingDir}/EIG* .
ln -s ${obsDir}/* .
ln -s ${ctrlDir}/xx* .
ln -s ${pickDir}/pick* .

exit
```

Figure 8: Example script to run the 20 year ECCO v4 state estimate on 96 processors (machine dependent).

```
PBS -S /bin/csh
#PBS -l select=1:ncpus=16:model=ivy+4:ncpus=20:model=ivy
#PBS -l walltime=12:00:00
#PBS -q long

#environment variables and libraries
#----------------------------------------------
limit stacksize unlimited
module purge
module load modules comp-intel/2013.1.117 mpi-sgi/mpt.2.10r6 netcdf/4.0
#
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH}:${HOME}/lib
setenv MPI_IB_TIMEOUT 20
setenv MPI_IB_RAILS 2
setenv MPI_IB_FAILOVER 1
setenv  MPI_CONNECTIONS_THRESHOLD  2049

#run MITgcm
#----------------
mpiexec -np 96 dplace -s1 ./mitgcmuv
exit
```

Figure 9: Calling sequence to be executed form within matlab to verify that their re-run of the 20 year ECO v4 state estimate is acceptably close to the released state estimate.

```
addpath ../results_itXX;%necessary .m and .mat files
mytest0=testreport_ecco([],'release1'); mytest0.info.interactive=0;%initialization
mytest=testreport_ecco(mytest0,'release1',[-1:4],'./',1);%compute the tests
testreport_ecco(mytest,'release1');%display the results
%testreport_write(mytest,'myRun');%save the results to a mat file

The result as displayed to screen should look something like:

--------------------------------------------------------------
             &   jT &   jS &  jTs &      ... &  (reference is)
r4it11.c65l/ & (-6) & (-6) & (-6) &      ...  &  release1
--------------------------------------------------------------
```

3 the *generic* pkg/ecco and pkg/ctrl

State estimation consists in minimizing a least squares distance, $\mathrm{J}(\mathfrak{u})$, that is defined as

$$
\begin{aligned}
\mathrm{J}(\mathfrak{u}) &= \sum_i \alpha_i \times (\mathrm{d}_i^T\, \mathbf{R_i}^{-1}\, \mathrm{d}_i) + \sum_j \beta_j \times (\mathfrak{u}_j^T \mathfrak{u}_j) && (1)\\
\mathrm{d}_i &= \mathcal{P}(\mathrm{m}_i - \mathrm{o}_i) && (2)\\
\mathrm{m}_i &= \mathcal{SDM}(\mathfrak{v}) && (3)\\
\mathfrak{v} &= \mathcal{Q}(\mathfrak{u}) && (4)\\
\mathfrak{u} &= \mathcal{R}(\mathfrak{u}') && (5)
\end{aligned}
$$

where d_i denotes a set of model-data differences, α_i the corresponding multiplier, $\mathbf{R_i}^{-1}$ the corresponding weights, $\mathfrak{u}_j$ a set of non-dimensional controls (of adjustable model parameters), β_j the corresponding multiplier, and additional symbols appearing in Eqs. 2-5 are defined below.

The *generic* implementation of Eqs.1-5 and the adjoint interface within the MITgcm is charted in Fig. 10. A basic presentation of Eqs.1-5 and Fig. 10 can be found in Forget et al. (2015). Details of the implementation within the MITgcm 'pkg/ecco' and 'pkg/ctrl' (a concern for developers mainly) are provided later in sections 3.3 and 3.4. Most importantly, sections 3.1 and 3.2 document the generic features in 'pkg/ecco' and 'pkg/ctrl' and their practical application. The presented features are tested daily via global_oce_cs32/ (section 2.1; adjoint experiment) and tested monthly in real-life conditions via the full ECCO v4 run (section 2.3; forward run), which will also serve for illustration in this section.

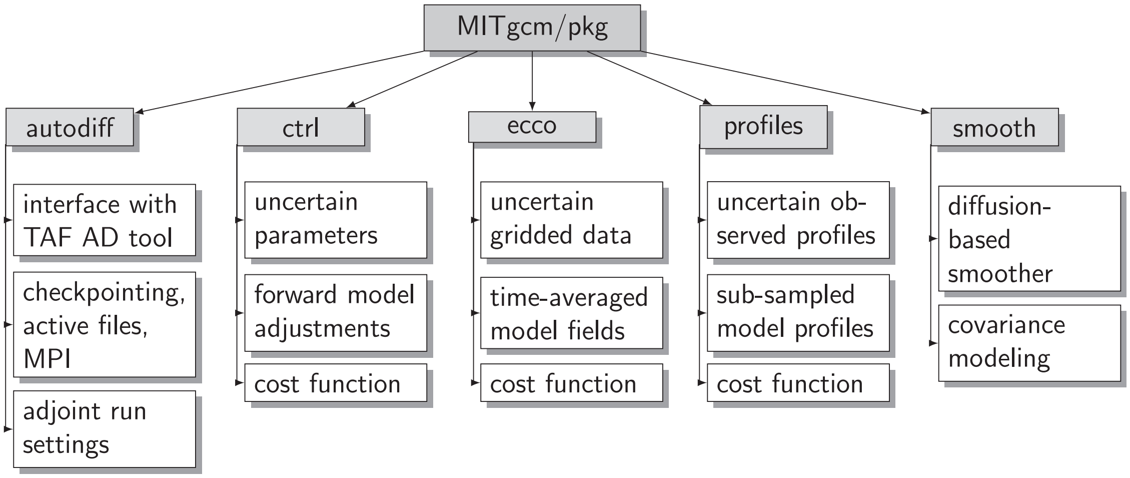

Figure 10: Chart of the organization and roles of MITgcm estimation modules. Additional details are reported in the MITgcm manual, in Forget et al. (2015), and in section 3 of this document.

3.1 usage: pkg/ecco

Model counterparts (m_i) to observational data (o_i) derive from adjustable model parameters ($\mathfrak{v}$; see section 3.2) through the model dynamics ($\mathcal{M}$; see Forget et al. 2015), diagnostic computations ($\mathcal{D}$), and averaging (or subsampling in 'pkg/profiles') in space and time ($\mathcal{S}$). For each cost function term the underlying uncertainty field ($\sqrt{\mathbf{R_i}}$) is specified by 'gencost_errfile'.[3] The corresponding cost function multiplier (α_i) is specified by 'mult_gencost' (it is 1. by default).

The file name for the observational fields (o_i) is specified by 'gencost_datafile'. Normally o_i (and m_i accordingly) is a time series of daily or monthly averages as specified by 'gencost_avgperiod'. In principle any periodicity should be possible but only 'month', 'day', 'step' and 'const' are implemented. The observational time series may be split in yearly files finishing in e.g. '_1992', '_1993', etc. Dense time series of model time steps can also be employed for testing purposes (e.g. in this data.ecco). Climatologies of m_i can be formed from its time series to compare with observational o_i climatologies. This option is activated by the gencost_preproc='clim' specification as illustrated in this data.ecco where 'gencost_preproc_i=9' sets the climatological cycle to 9 records (1 record = 1 time step in this example). Finally the gencost_avgperiod='const' option is adequate when m_i is constant through time once the model initialization phase is complete.[4] Plain model-data misfits ($\mathrm{m}_i - \mathrm{o}_i$) can be penalized directly (i.e. used in Eq. 1 in place of d_i). More generally though penalized misfits (d_i in Eq. 1) derive from $\mathrm{m}_i - \mathrm{o}_i$ through a generic post-processor ($\mathcal{P}$ in Eq. 2). They can thus be smoothed in space at run time by setting gencost_posproc='smooth' for example (see this data.ecco).

The physical variable in m_i is specified at run time via the first characters in 'gencost_barfile' (to match the observed variable specified as o_i) as illustrated in this data.ecco and that data.ecco. The list of implemented variables as of the MITgcm checkpoint c65m is reported in Tab. 1. In cases when two different averages of the same variable may be needed in separate cost function terms (e.g. daily and monthly) or simply for convenience then an extension starting with '_' can be added to 'gencost_barfile' (such as '_day' and '_mon'). In cases when two cost function terms may use the same m_i, the user may specify the same name (via 'gencost_barfile') in both terms. In cases of three dimensional variables (see Tab. 1) the 'gencost_is3d' run-time option is automatically set to .TRUE. (it is .FALSE. by default). The gencost_outputlevel=1 option will output model-data misfit fields for offline analysis and visualization.

[3] The option for time varying error fields remains to be implemented in gencost.

[4] This feature remains to be added to daily benchmark.

Table 1: List of implemented 'gencost_barfile' options that can be specified at run-time in data.ecco (as of the MITgcm checkpoint c65m). An extension starting with '_' can be appended at the end of the variable names for convenience.

variable name	description	remarks
m_eta	sea surface height	free surface + corrections
m_sst	sea surface temperature	first level temperature
m_sss	sea surface salinity	first level salinity
m_bp	bottom pressure	
m_ustress	zonal wind stress	
m_vstress	meridional wind stress	
m_uwind	zonal wind	
m_vwind	meridional wind	
m_atemp	atmospheric temperature	
m_aqh	atmospheric humidity	
m_precip	precipitation	
m_swdown	downward shortwave	
m_lwdown	downward longwave	
m_wspeed	wind speed	
m_siarea	sea-ice concentration	
m_siheff	sea-ice effective thickness	
m_sihsnow	snow effective thickness	
m_theta	temperature	three-dimensional
m_salt	salinity	three-dimensional
m_UE	zonal velocity	three-dimensional
m_VN	meridional velocity	three-dimensional
m_diffkr	diapycnal diffusion	three-dimensional, constant
m_kapgm	bolus velocity parameter	three-dimensional, constant
m_kapredi	isopycnal diffusion	three-dimensional, constant
m_geothermalflux	geothermal heating	'const'
m_bottomdrag	bottom drag	'const'

3.2 usage: pkg/ctrl

Three basic options are implemented for Eqs. 4-5: time variable two dimensional controls (‘gentim2d’), time-invariant 2D controls (‘genarr2d’), and time-invariant 3D controls (‘genarr3d’). The ‘gentim2d’ run-time options are documented below as an example. Corresponding options exist in ‘genarr2d’ and ‘genarr3d’ except for the specifically time variable aspects (see below).

The control problem is non-dimensional by default, as reflected by the omission of weights in control penalties ($\mathfrak{u}_j^T \mathfrak{u}_j$, Eq.1). Non-dimensional controls ($\mathfrak{u}_j$) are scaled to physical units ($\mathfrak{v}_j$) through multiplication by their respective uncertainty fields ($\sigma_{\mathfrak{u}_j}$), as part of the generic pre-processor Q (Eq.4). An adjustable parameter are activated and specified by the first character in ‘xx_gentim2d_file’ (as illustrated in this data.ctrl and that data.ctrl). The list of implemented variables as of the MITgcm checkpoint c65m is reported in Tab. 2.

Table 2: List of implemented ‘xx_gen?????_file’ (with ????? indicated under remarks) options that can be specified at run-time in data.ctrl (as of the MITgcm checkpoint c65m). An extension starting with ‘_‘ can be appended at the end of the variable names for convenience.

variable name	description	remarks
’xx_atemp’	atmospheric temperature	‘gentim2d’
’xx_aqh’	atmospheric humidity	‘gentim2d’
’xx_swdown’	downward shortwave	‘gentim2d’
’xx_lwdown’	downward longwave	‘gentim2d’
’xx_precip’	precipitation	‘gentim2d’
’xx_uwind’	zonal wind	‘gentim2d’
’xx_vwind’	meridional wind	‘gentim2d’
’xx_tauu’	zonal wind stress	‘gentim2d’
’xx_tauv’	meridional wind stress	‘gentim2d’
’xx_etan’	initial free surface height	‘genarr2d’
’xx_theta’	initial temperature	‘genarr3d’
’xx_salt’	initial salinity	‘genarr3d’
’xx_diffkr’	diapycnal diffusion	‘genarr3d’
’xx_kapgm’	bolus velocity parameter	‘genarr3d’
’xx_kapredi’	isopycnal diffusion	‘genarr3d’
’xx_geothermal’	geothermal heating	‘genarr2d’
’xx_bottomdrag’	bottom drag	‘genarr2d’

The corresponding uncertainty must be provided (in the form of weights $1/\sigma^2_{\mathfrak{u}_j}$; via a file name specified by ‘xx_gentim2d_weight’) to scale $\mathfrak{u}_j$ to physical units.[5] Besides the scaling of $\mathfrak{u}_j$ to physical units, generic pre-processor Q can include for example spatial correlation modeling (using an implementation of Weaver and Courtier, 2001). This feature is activated for e.g. the first set of controls by setting xx_gentim2d_preproc(1)=’WC01’.[6] As an alternative, one may set xx_gentim2d_preproc(1)=’smooth’ to apply the smoothing part of Weaver

[5] Options to specify an uncertainty field or constant instead remain to be implemented.

[6] The ctrlSmoothCorrel3D/2D switches and CPPs remain to be fully deprecated.

and Courtier, 2001 but omit the normalization part. Additional specification is possible in some cases (depending on ‘xx_gentim2d_preproc’) via sub-options ‘xx_gentim2d_preproc_i’ (integer),‘...._r’ (real), ‘...._c’ (character string). For example, setting xx_gentim2d_preproc_i(1,1)=2 along with xx_gentim2d_preproc(1)=’WC01’ would apply the second correlation model defined in data.smooth to the first set of controls (the first correlation model would otherwise be used by default). The full list of implemented ‘xx_gentim2d_preproc’ options (as of the MITgcm checkpoint c65m) is reported in Tab. 3.

Table 3: List of implemented ‘gentim2d’ options (top) and associated ‘xx_gentim2d_preproc’ options (bottom) that can be specified at run-time in data.ctrl (as of the MITgcm checkpoint c65m).

parmater name	type	role
xx_gentim2d_file	character	activate an adjustable parameter
xx_gentim2d_weight	character	specify weight field(s)
xx_gentim2d_preproc	character	optional features listed below
xx_gentim2d_bounds	real (five values)	impose bounds (see that data.ctrl)
mult_gentim2d	real	cost function multiplier (1. by default)
gentim2dPrecond	real	preconditioner (1. by default)

xx_gentim2d_preproc	further specifications (see text)	effect (in forward)
’WC01’	xx_gentim2d_preproc_i	activate correlation modeling
’smooth’	xx_gentim2d_preproc_i	activate plain smoothing
’docycle’	xx_gentim2d_preproc_i	average period replication
’rmcycle’	xx_gentim2d_preproc_i	periodic average subtraction
’variaweight’	(none)	time variable weights

In the case of time-variable parameter adjustments, the frequency is specificied by ‘xx_gentim2d_period’. Time variable weights can also be provided by specifying ‘variaweight’ as e.g. ‘xx_gentim2d_preproc(2)’. In this case the ‘xx_gentim2d_weight’ file must contain as many records as the control parameter time series itself (≈ the duration of the run divided by ’xx_gentim2d_period’). In the case when several adjustments are sought in one model parameter (e.g. time mean and time variable forcing adjustments treated separetely) then an extension starting with ‘_‘ can be added to the ‘xx_gentim2d_file’ specification (e.g. ’_mean’ and ’_anom’; see this data.ctrl).

Further time-variable (‘gentim2d’ only) options are available via xx_gentim2d_preproc= ’docycle’ and ’rmcycle’. They can be combined with ‘variaweight’ (that occurs after ‘docycle’ and ‘rmcycle’) to create many ‘gentim2d’ varieties. The example in this data.ctrl specifies that adjustments to atmospheric temperatures are split in three terms: time mean (‘xx_atempA’), *seasonal cycle* anomaly (‘xx_atempB’; of zero time mean), and *interannual* anomaly (‘xx_atempC’; of zero time mean and *seasonal cycle*). In a real-life situation, a seasonal cycle would consist of e.g. 12 monthly averages or 26 bi-weekly averages. In the short global_oce_cs32 benchmark a *seasonal cycle* is represented by a cycle of just two time steps. The three corresponding cycle durations would be specified as xx_gentim2d_preproc_i=12, 26 and 2 respectively. The corresponding time mean would always be specified as xx_gentim2d_preproc_i=1.

With ‘gentim2d’, it can be imposed that adjustments stay bounded via ‘xx_gentim2d_bounds’.

Within ‘genarr2d’ and ‘genarr3d’, the corresponding option rather imposes bounds on adjusted parameters. Another run-time parameter in data.ctrl is ‘mult_gentim2d’ (or the ‘genarr2d’, ‘genarr3d’ version) that sets the multiplier for the corresponding cost function penalty (β_j in Eq. 1; $\beta_j = 1.$ by default). Pre-conditioner $\mathcal{R}$ (Eq. 5) does not appear in the estimation problem itself (Eq.1), as it only serves to push an optimization process preferentially towards certain directions of the control space. It is specified by ‘gentim2dPrecond’ (which is 1. by default).[7]

3.3 implementation: pkg/ecco and pkg/ctrl

The implementation of Eqs. 2 and 3 belongs in ‘pkg/ecco’ and ‘pkg/profiles’[8] whereas Eqs. 4 and 5 belong in ‘pkg/ctrl’. This section depicts the generic features implementations in ‘pkg/ecco’ (first paragraphs) and ‘pkg/ctrl’ (later paragraphs). The maximum numbers of generic cost function terms (NGENCOST), 3D cost function terms (NGENCOST3D), and post-processing options (NGENPPROC) are set at compile time in ecco.h. The maximum numbers of generic time-variable 2D controls (maxCtrlTim2D), time-invariant 2D controls (maxCtrlArr2D), time-invariant 3D controls (maxCtrlArr3D) and pre- or post-processing options (maxCtrlProc) are set at compile time in CTRL_SIZE.h. Other files involved in compiling ‘pkg/ecco’ and ‘pkg/profiles’ are listed in section 3.4. Run-time options for ‘pkg/ecco’ and ‘pkg/profiles’ (that should be the main aspect of interest for most users) are readily documented in sections 3.1-3.2.

The operations in $\mathcal{D}$ and $\mathcal{S}$ (see Eq.3) are mainly carried out as the forward model steps through time, respectively by ecco_phys.F and cost_averagesfields.F. During cost_averagesfields.F, cost_gencost_customize.F maps physical variables to generic arrays (according to ‘gencost_barfile’ specified in data.ecco; see section 3.1) and cost_averagesgeneric.F then proceeds with time-averaging, and periodically outputs the time-averaged m_i to file. Climatologies of m_i can be formed (as an optional feature) from its time series to compare with observational o_i climatologies (see section 3.1). This part of the m_i processing is carried out within cost_generic.F after the full time series has been written to file. Model-data misfits are then computed (Eq. 2) by cost_generic.F that relies on ecco_toolbox.F for elementary operations and on cost_genread.F for re-reading m_i from file. The calls to cost_generic.F are operated in a loop by cost_gencost_all.F.

The overall sequence of operations for one cost function term is charted in Fig.11. The distinction between ‘preproc’ and ‘posproc’ matches that between Eqs. 3 and 2. Most concretely the pre-processing ends and post-processing starts at the computation of $\mathrm{m}_i - \mathrm{o}_i$ using ‘ecco_diffmsk’ in cost_generic.F. Besides the numerous possibilities offered by this generic code, specific cost function terms that do not fit in the Fig.11 chart quite yet can be operated via cost_gencost_all.F and freely take advantage of the rest of the generic capabilities (storage arrays, adjoint checkpoint storage, run-time parameters, etc.). Examples of how to do this include cost_gencost_boxmean.F, cost_gencost_sshv4.F and cost_gencost_seaicev4.F.

In the implementation of Eqs. 4 and 5, generality and versatility is greatly improved by operating virtually all of the pre-processing during model initialization. This is done by ctrl_map_ini_genarr.F (’genarr2d’, ’genarr3d’) and ctrl_map_ini_gentim2d.F (’gentim2d’). By the end of the processing steps, the effective version of the parameter adjustments (’gentim2d’,

[7] The ‘genarr2d’ cases is treated accordingly, but the ‘genarr3dPrecond’ implementation seems incomplete.

[8] Additional documentation of ‘pkg/profiles’ is available in the MITgcm manual and in Forget et al. (2015).

Algorithm 1 Generic cost function algorithm.

```
function COST_GENERIC(...)                          ▷ Argument list defines the cost function
    call ecco_zero                                  ▷ Initialize local array to 0
    call ecco_cprsrl                                ▷ Copy mask to local array
    for irec = 1, nrecloop do                       ▷ Loop over time steps, days or months
        call cost_gencal                            ▷ Get file names, pointers
        Begin cost_genread                          ▷ Read, process model field
            if no preproc then
                call ecco_readbar                   ▷ Read one record
            else if preproc=clim then
                call ecco_readbar within loop       ▷ Average records
            end if
        End cost_genread
        call mdsreadfield                           ▷ Read observational field
        call ecco_diffmsk                           ▷ Compute masked model-data misfit
        if posproc=smooth then
            call smooth_hetero2d                    ▷ Smooth masked misfit
        end if
        call ecco_addcost                           ▷ Add to cost function
    end for
end function
```

Figure 11: Chart of the generic cost function routine in pkg/ecco.

’genarr2d’) or of the adjusted model parameters (’genarr3d’) are written to disk (with ’.effective’ in the file name). Adjusted time-invariant model parameters are generally set during model initialization, so their adjustments are also operated before the model time-stepping starts by ctrl_map_ini_genarr.F (for ’genarr2d’, ’genarr3d’). Adjusted forcing variables are however reset at each time-step during the model run, so their adjustments are operated at the same times by e.g. exf_getsurfacefluxes.F or exf_getffields.F (for ’gentim2d’). The cost function term $\mathbf{u}_j^T \mathbf{u}_j$ is computed once the model run is complete along with the rest of Eq.1 (see section 3.1).

It should be stressed that if a control parameter is activated at run-rime (see section 3.2) but its uncertainty is not specified (or vice versa) then ctrl_readparms.F signals the inconsistency before stopping the model during its initialization (the user may otherwise overlook that his specification would have no effect). It should also be noted that the effective version of ‘gentim2d’ adjustments always consists of a full time series of length ≈ the duration of the run divided by ’xx_gentim2d_period’. It may contain e.g. repeated seasonal mean adjustments or a sequence of interannual adjustments. This choice allows for a uniform, simple and general treatment of all varieties of ‘gentim2d’ adjustments needed during the model integration (that boils down to the temporal interpolation carried out in ctrl_map_gentim2d.F). This approach is particularly advantageous in the context of the checkpointed adjoint model development (see MITgcm manual). It only implies a marginal overhead in disk storage as compared with the adjoint checkpointing output itself or with e.g. six-hourly re-analysis forcing input files.

3.4 Legacy: pkg/ecco and pkg/ctrl

Much of the legacy code that has been distributed as part of ‘pkg/ecco’ and ‘pkg/ctrl’ in the past is now deprecated – it is superseeded by the generic cost function and control codes presented above. Most of the deprecated codes had not been tested or maintained for many years, and consist of variations of the same operations duplicated many times. Another issue was the lack of organization amongst the deprecated codes (unlike in Fig.10). The consensus was that there was no point in keeping them around much longer.

For the time being the deprecated codes still exist but they are not compiled anymore unless the ‘ECCO_CTRL_DEPRECATED’ compile option is added in e.g. ‘ECCO_CPPOPTIONS.h’ (see below for details). To further facilitate the transition from old to new setup, the ctrlUseGen run-time parameter allows a switch between the old and new (generic) treatment of control vectors (assuming that ‘ECCO_CTRL_DEPRECATED’ was defined at compile time). As a side note: there is one non-generic feature that ISN’T deprecated since it has not been re-implemented in generic fashion, which is the control of open boundary conditions.

The deprecation of the legacy codes leads to a vast reduction in the volume of estimation codes (30% of the code treated by automatic differentiation, which includes the entire physical model, was removed in the process), a vast addition of capabilities (new or pre-existing functionalities are now available for any gridded data set), and a greatly improved flexibility (virtually all options can now be switched on/off at run time). Furthermore, the ecco, ctrl and autodiff packages were made independent of each other, and to follow common MITgcm coding practices. For example they can now be switched on/off at run time, independently (by virtue of useECCO, useCTRL, useAUTODIFF).

Compiling options are typically found in the ‘code/’ directory of any given setup of MITgcm (when customized) or in the corresponding MITgcm package (when using defaults). The most obvious difference between the new setup and an old setup is that CPP_OPTIONS.h now disregards ECCO_CPPOPTIONS.h and uses the following instead :

- AUTODIFF_OPTIONS.h contains the few compile directives of pkg/autodiff. The maximum numbers of time steps are set in tamc.h
- ECCO_OPTIONS.h contains compile directives of pkg/ecco. Very few remain necessary, since all generic cost function settings can now be chosen at run time. The maximum numbers of cost terms are set in ecco.h
- CTRL_OPTIONS.h contains compile directives of pkg/ctrl. Very few remain necessary, since all generic control settings can now be chosen at run time. The maximum numbers of controls are set in CTRL_SIZE.h
- along with MOM_COMMON_OPTIONS.h, GMREDI_OPTIONS.h, GGL90_OPTIONS.h, PROFILES_OPTIONS.h, EXF_OPTIONS.h, SEAICE_OPTIONS.h, DIAG_OPTIONS.h

4 notes on pkg/profiles

As of upcoming checkpoint65o, pkg/profiles has been added capabilities to specify variables at run-time. The ‘data.profiles’ in Fig. 12 thus specifies via ‘prof_names’ that ‘prof_T’, ‘prof_S’ and ‘prof_OXY’ are provided in input file ‘wod13_CTD3.nc’. The corresponding model variables are set via ‘prof_namesmod’ to ‘theta’, ‘salt’ and ‘pTracer’ respectively. In this last case the pTracer index is set to 5 using ‘prof_itracer’ (the index for oxygen in ‘pkg/dic’). Another run-time option is available to specify a model variable scaling factor (‘prof_facmod’; equal to 1 by default. Additional variables are available when ‘pkg/ecco’ is compiled, such as ’eta’ (sea surface height), ‘UE’ (zonal velocity) and ‘VN’ (meridional velocity) computed at grid cell centers by ecco_phys.F. The full list of available variables is encoded in profiles_interp.F. If variable are not specified at all at run-time (e.g. as done in that data.profiles) then pkg/profiles reverts to its old behavior (i.e. it uses ‘prof_T’, ‘prof_S’ and ‘theta’, ‘salt’ by default).

As part of the internal revision of ‘pkg/profiles’ that added these capabilities, new run-time parameters (prof_names, prof_namesmod, prof_facmod, prof_itracer) and internal variables (prof_namesmask, prof_namesweight) were added to profiles.h. The specification of array sizes that earlier was in profiles.h along with the array definitions was moved to a separate header file (PROFILES_SIZE.h) as done in other packages. An advantage of this approach is that future internal revisions of ‘pkg/profiles’ (with potential modifications in ‘profiles.h’) won’t require updating compiling options in e.g. this code directory. The internal treatment of file ID’s (routine ‘profiles_findunit.F’ was added) and of interpolation coefficients (‘ALLOW_PROFILES_GENERICGRID’ compile option became obsolete) were also revised. Thus ‘profiles_interp_gg.F’ and other flavors of ‘profiles_interp.F’ could be retired.

```
#
# ******************
# PROFILES cost function
# ******************
 &PROFILES_NML
 profilesDir='profiles',
#
 profilesfiles(1)= 'wod13_CTD3',
 prof_names(1,1)='prof_T',
 prof_namesmod(1,1)='theta',
 prof_names(1,2)='prof_S',
 prof_namesmod(1,2)='salt',
 prof_names(1,3)='prof_OXY',
 prof_namesmod(1,3)='pTracer',
 prof_itracer(1,3)=5,
 #
 &
```

Figure 12: Example: data.profiles