ECCO v4 development notes

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

June 11, 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.

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 2.2). Thus running the short benchmarks (section 2.1) is a useful step towards re-producing the state estimate (section 2.2). 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 2.2, 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 2.2 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 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 2.2). Pre-requisite for sections 2.1 and 2.2: having downloaded the MITgcm (section 1.1) and the ECCO v4 setups (section 1.2). Pre-requisite for section 2.2: 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 2.2). 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 forward global_oce_llc90/ benchmark 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 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). 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.5. The tamc.h_itXX and profiles.h_itXX headers (see Fig.5) 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.6), and submits a job to the queueing system (see Fig.7). User interested in obtaining the additional model input required to reproduce the baseline 1992-2011 solution (i.e. the ECCO v4 state estimate) are advised to contact ecco-support@mit.edu. This model input consists of:

- the forcing files (EIG*199? EIG*20??).
- the initial conditions (pickup*) and control vector adjustsments (xx_*).
- the insitu data sets inputs (*feb2013*.nc) used for benchmarking purposes (see below).

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.8 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 5: 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.h
cp ../code/tamc.h_itXX tamc.h
cp ../code/profiles.h_itXX profiles.h

make -j 4
```

Figure 6: 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 7: 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 8: 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 re-implemented ecco and ctrl packages

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

$$\mathrm{J}(\mathfrak{u}) = \sum_i \alpha_i \times (\mathrm{d}_i^T \, \mathbf{R_i}^{-1} \, \mathrm{d}_i) + \sum_j \beta_j \times (\mathfrak{u}_j^T \mathfrak{u}_j) \quad (1)$$

$$\mathrm{d}_i = \mathcal{P}(\mathrm{m}_i - \mathrm{o}_i) \quad (2)$$

$$\mathrm{m}_i = \mathcal{SDM}(\mathfrak{v}) \quad (3)$$

$$\mathfrak{v} = \mathcal{Q}(\mathfrak{u}) \quad (4)$$

$$\mathfrak{u} = \mathcal{R}(\mathfrak{u}') \quad (5)$$

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, β_j the corresponding multiplier, and additional symbols appearing in Eqs. 2-5 are defined below. The implementation of Eqs.1-5 and the adjoint interface within the MITgcm is charted in Fig. 9. A general presentation of Eqs.1-5 and Fig. 9 can be found in Forget et al. (2015). The focus here is on the underlying code development in 'pkg/ecco' and 'pkg/ctrl' of MITgcm. These features are now tested daily via global_oce_cs32/ (adjoint experiment) that will also serve here for illustration in this document.

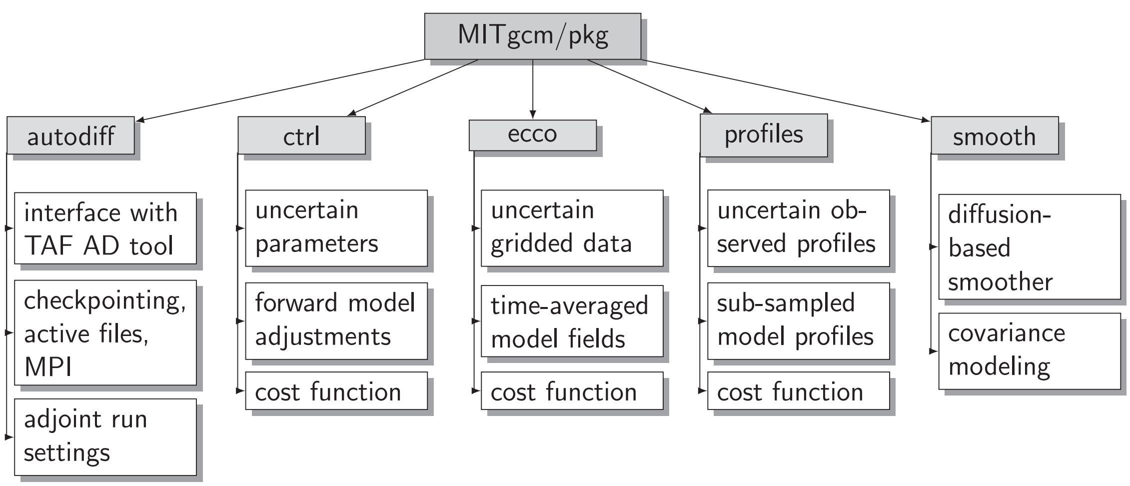

Figure 9: Chart of the organization and roles of MITgcm estimation modules. Additional details are reported in the MITgcm manual, Forget et al. (2015), and section 3 of this document.

3.1 pkg/ecco run-time parameters and algorithm

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 ($\mathcal{S}$). The physical variable in m_i is specified at run time via the first characters in 'gencost_barfile' (to match the observed variable in o_i) as illustrated in this data.ecco and that data.ecco. The list of implemented variables as of the MITgcm checkpoint c65l consists of 'm_eta', 'm_sst', 'm_sss', 'm_bp', 'm_tauZon', 'm_tauMer', 'm_theta', 'm_salt'[3]. In the case of three dimensional variables (e.g. 'm_theta' or 'm_salt') the 'gencost_is3d' run-time option must be set to .TRUE. in data,ecco (it is .FALSE. by default). In cases when two different averages of the same variable may be needed (e.g. daily and monthly) then a suffix starting with '_' can be added (such as '_day' and '_mon'). The file name for the observational fields (o_i) and the model-data uncertainty field ($\sqrt{\mathbf{R_i}}$) are specified at run time via 'gencost_datafile'[4] and 'gencost_errfile'[5] respectively. The cost function multiplier (α_i) further needs to be specified by 'mult_gencost' (it is 0. by default).

Both $\mathcal{D}$ and $\mathcal{S}$ in Eq.3 are mainly carried out as the forward model steps through time, respectively by ecco_phys.F and cost_averagesgeneric.F, and m_i is written to file periodically. m_i and o_i normally are time series of daily or monthly averages[6], as specified at run time via 'gencost_avgperiod'. However dense time series of model time steps can also be employed for testing purposes as illustrated in this data.ecco. Furthermore climatologies of m_i can be formed from its time series by cost_genread.F to allow for comparison with observational o_i climatologies. This part of the m_i processing is carried out within cost_generic.F (see below) after the full time series has been written to file. It is activated via a 'gencost_preproc' option as illustrated in this data.ecco.

Model-data misfits are computed (Eq. 2) upon completion of the forward model simulation 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. 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 misfits to be penalized (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 via 'gencost_posproc' for example (see this data.ecco). The overall sequence of operations for one cost function term is charted in Fig.10. 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.

[3]!!! pending modification ... right now : 'eta', 'sst', 'sss', 'bp', 'tauZon', 'tauMer', 'theta', 'salt'

[4]The observational field time series may be split in yearly files finishing in e.g. '_1992', '_1993', etc.

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

[6]In principle any frequency should be possible but only 'month', 'day', and 'step' are currently implemented.

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

3.2 pkg/ctrl run-time parameters and algorithm

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). Three basic options are implemented: time variable two dimensional controls ('gentim2d'), time-invariant 2D controls ('genarr2d'), and time-invariant 3D controls ('genarr3d'). In all three, non-dimensional controls ($\mathfrak{u}_j$) are scaled to physical units ($\mathfrak{v}_j$) through multiplication by their respective uncertainty fields, as part of the generic pre-processor Q (Eq.4). For any adjustable parameter that is activated at run-time (by setting 'xx_gentim2d_file' in data.ctrl) the corresponding uncertainties must be provided in the form of weights (by setting 'xx_gentim2d_weight' in data.ctrl). Before discussing these specifications in more detail, it should be stressed that if any uncertain parameter is activated but the weight is not specified (or vice versa) then pkg/ctrl signals the inconsistency and then stops the model during its initialization (since nothing would not get adjusted otherwise).

An adjustable model parameter $\mathfrak{u}_j$ gets activated at run time according to the first characters in 'xx_gentim2d_file' (or the 'genarr2d' or 'genarr3d' versions) as illustrated in this data.ctrl and that data.ctrl. In the case when several adjustments are sought in one model parameter (e.g. time mean and time variable forcing adjustments treated separetely) then a suffix starting with '_' can be added to the 'xx_gentim2d_file' specification (such as '_mean' and '_anom'; see this data.ctrl). Each weight binary file (specified via 'xx_gentim2d_weight') must contain at least one map of $\frac{1}{\sigma_u^2}$ where σ_u is the corresponding uncertainty map that will be used to scale $\mathfrak{u}_j$ to physical units (as part of Q in Eq.4).[7] Time variable weights can also be provided by specifying 'variaweight' as a pre-processing option in data.ctrl ('xx_gentim2d_preproc'; further discussed below).

Besides the scaling to physical units, the generic pre-processor Q can include the spatial correlation model and/or a mapping in time such as the cyclic repetition of mean seasonal controls for example. Such pre-processing options are set per 'xx_gentim2d_preproc' run-time parameters. Refined specifications are needed in some cases, which can be done via 'xx_gentim2d_preproc_i' (integer), 'xx_gentim2d_preproc_r' (real), 'xx_gentim2d_preproc_c' (character string). In the case of time-variable adjustable model parameters, the frequency is specificied per 'xx_gentim2d_period' and the multiplier for the cost function penalty (β_j in Eq. 1) is specified per 'mult_gentim2d' (which is 0. by default). In the case of time-invariant 3D model parameters, bounds can further be specified via the 'xx_genarr3d_bounds' run-time parameter.[8] The pre-conditioner $\mathcal{R}$ (in 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 via the 'gentim2dPrecond' (or the 'genarr2d' or 'genarr3d' versions) run-time parameter (which is 1. by default).

In the implementation of Eqs. 4 and 5, generality and versatility is greatly improved by operating virtually all of the pre-processing at initialization time. 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', 'genarr2d') or of the adjusted model parameters ('genarr3d') are written to disk (with '.effective' in the file name). Time-invariant model parameters are set during model initialization, so their potential adjustments are generally also made before the model time-stepping starts ('genarr2d', 'genarr3d').

[7] Eventually options should be added to specify an uncertainty field or constant instead

[8] In principle this should be possible also for 'genarr2d' and 'gentim2d' but this is not yet implemented.

Forcing variables are however reset at each time-step, so their potential adjustment via 'gentim2d' are made during the model run. In this case it should be noted that the effective version always consists of a full time series, whether it contains e.g. repeated seasonal mean adjustments or series of interannual anomalies. This choice allows for a uniform, simple and general treatment of all varieties of time-variable parameter adjustments needed during the model integration (the temporal interpolation carried out in ctrl_map_gentim2d.F). This approach is particularly advantageous in the context of the checkpointed adjoint model development, and generally implies a marginal overhead in disk storage as compared with e.g the adjoint checkpoints (see MITgcm manual) or the forcing fields thermselves. The cost function that is commonly included to penalize parameter adjustments ($\mathfrak{u}_j^T \mathfrak{u}_j$, Eq.1) is itself computed once the model run has completed, along with the other cost function terms (section 3.1).

3.3 Compiling options for pkg/ecco, pkg/ctrl, etc.

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.9). 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