Jet Propulsion Laboratory
California Institute of Technology

Running the ECCO Model and Adjoint

Ou Wang, Ian Fenty, and Ichiro Fukumori

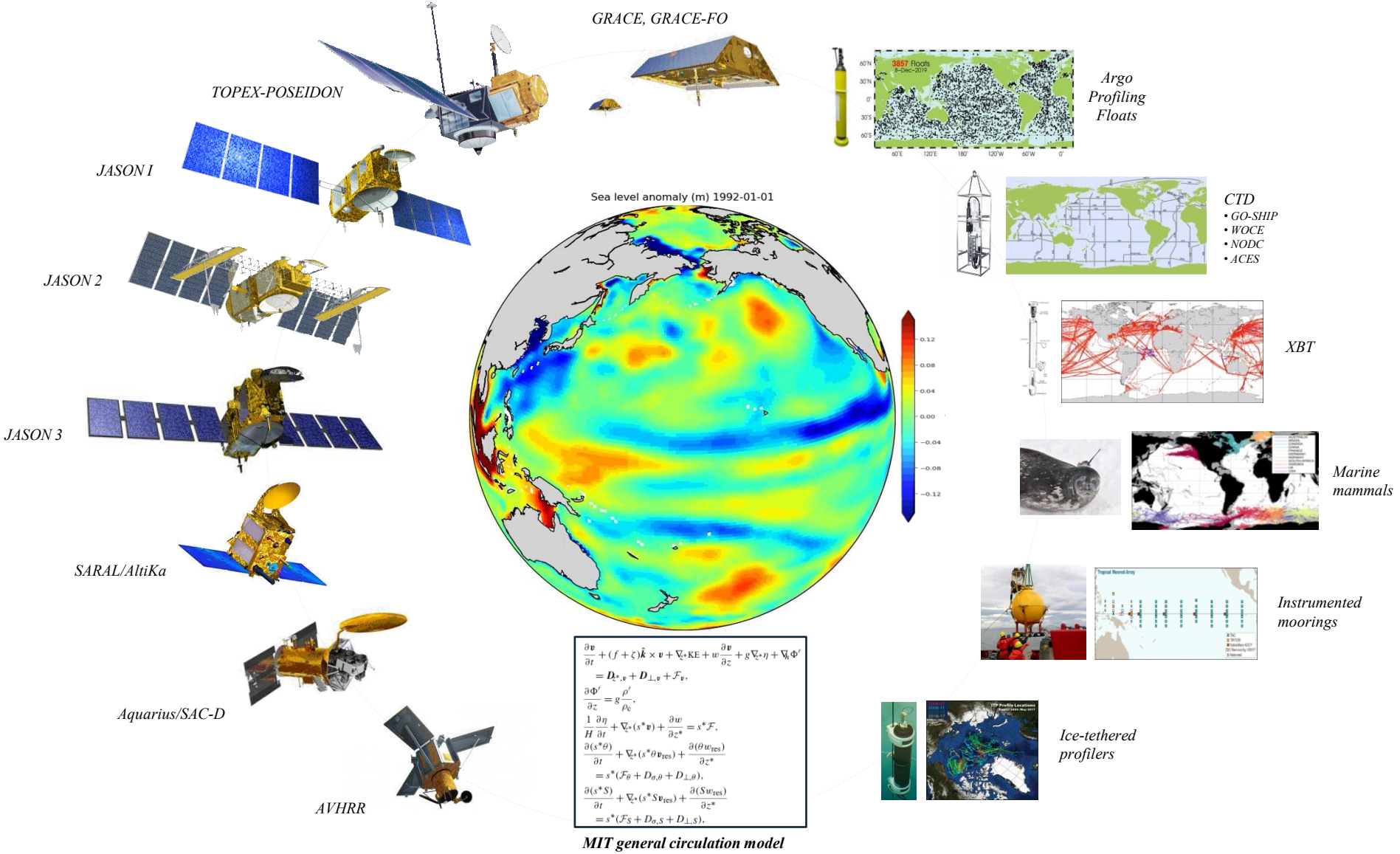
Jet Propulsion Laboratory, California Institute of Technology

ECCO Summer School 2025 Pacific Grove, California May 19-30, 2025

© 2025 All rights reserved

Estimating the Circulation & Climate of the Ocean (ECCO) ocean state estimates:

Synthesis of global ocean data with MITgcm using an adjoint-based inverse estimation method



ECCO Ocean State Estimates: Synthesis of global ocean and sea-ice observations with MITgcm using an adjoint-based inverse estimation method

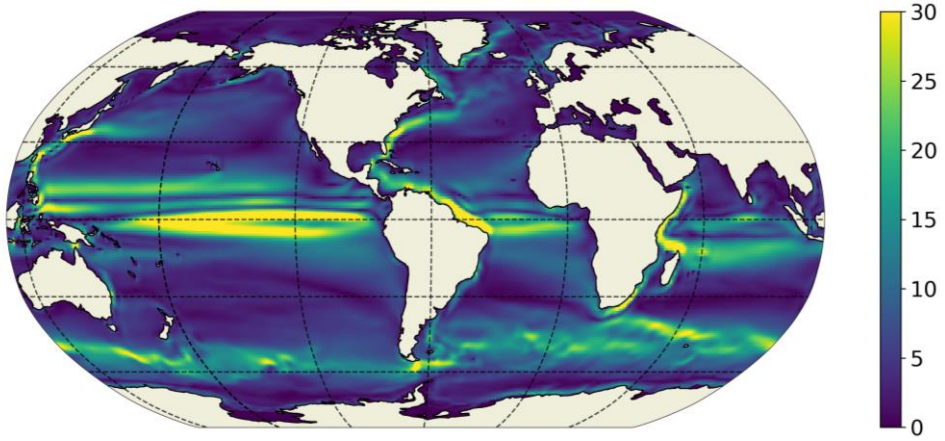
Key Components

- Observations to constrain the model along with their **uncertainties**
- Model (MITgcm)
    - Grid
    - Bathymetry
    - Surface forcing and other (e.g., geothermal forcing, ice-shelf melt)
    - Mixing coefficients
    - **Uncertainty for control**
- Algorithmic Differentiation Tools (e.g., TAF, Tapenade) to generate adjoint code

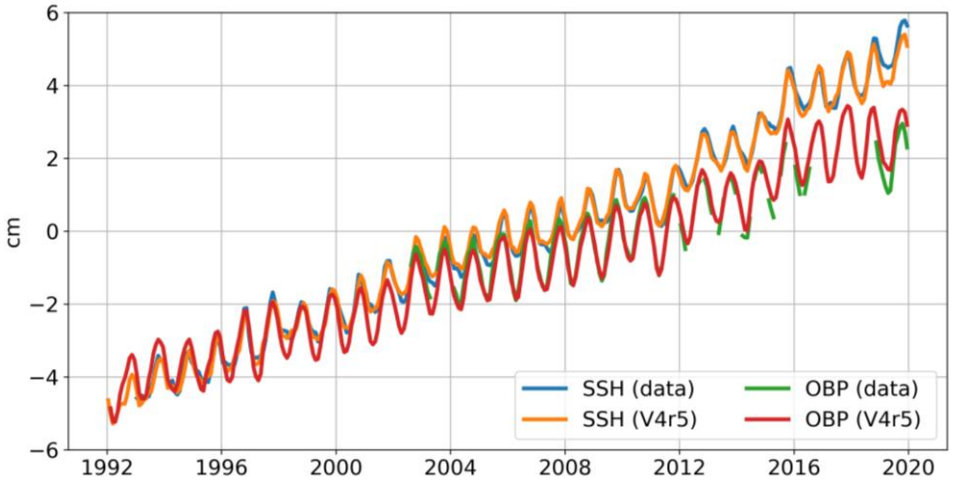
ECCO Version 4

- Version 4 Release 5 (V4r5, latest release) and Version 4 Release 4 (V4r4)
- Multi-decadal global ocean and sea-ice state estimate; observation constrained for 1992-2019 (V4r5) and simulation only for 2020-present
- Constrained by satellite altimetry, GRACE, Aquarius, AVHRR, ARGO, CTD, XBT, satellite sea-ice measurements (concentration, freeboard)...;
- Including ice-shelves and ice-fronts around Antarctic;
- Adjusting atmospheric forcing, mixing parameters, initial conditions, and ice-shelf heat transfer coefficient (only for V4r5) (*controls*);
- Model: non-linear free surface boundary and real freshwater boundary conditions;
- A physically consistent solution.

1992-2019 mean surface current speed (cm/s)

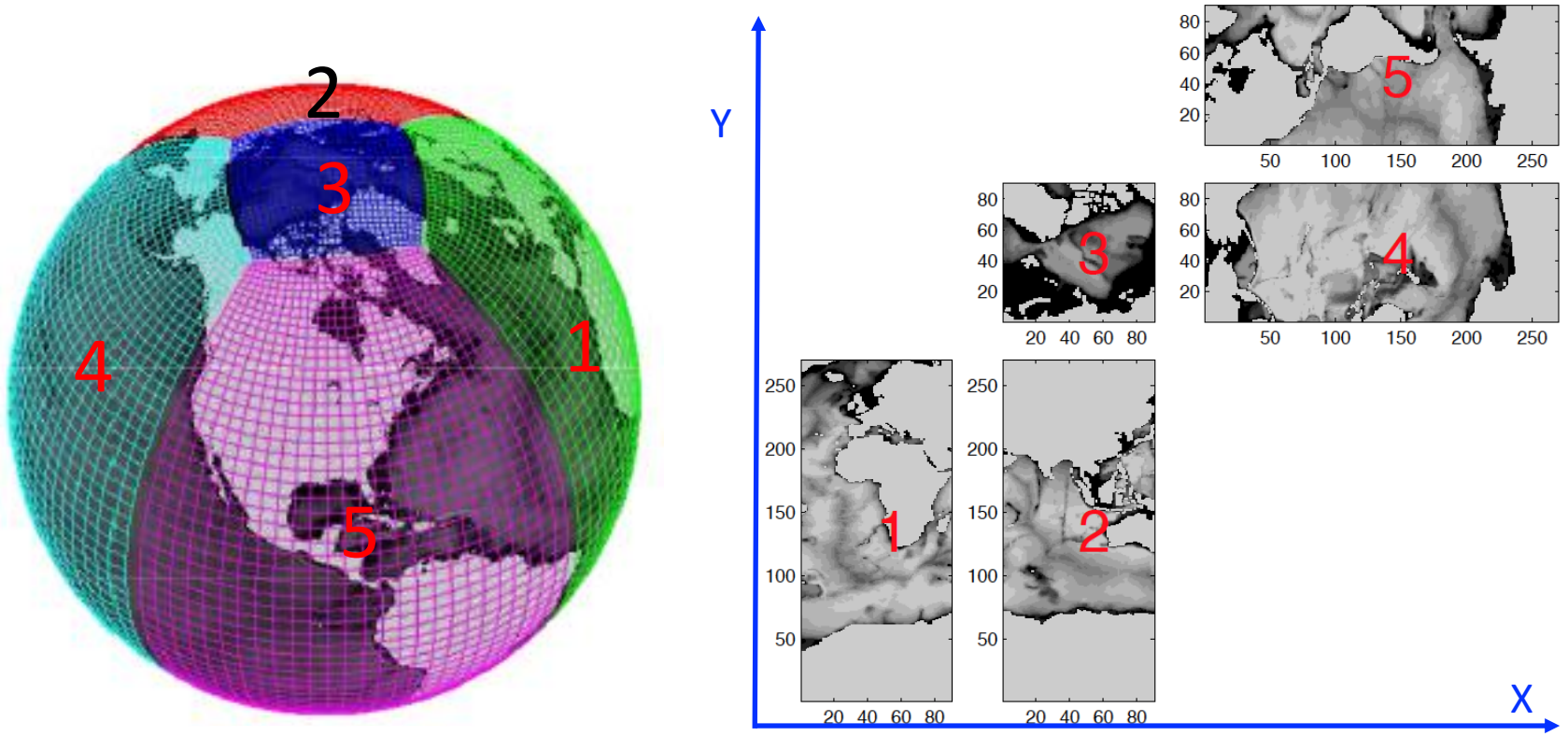


Global mean ssh and obp (cm)



ECCO V4

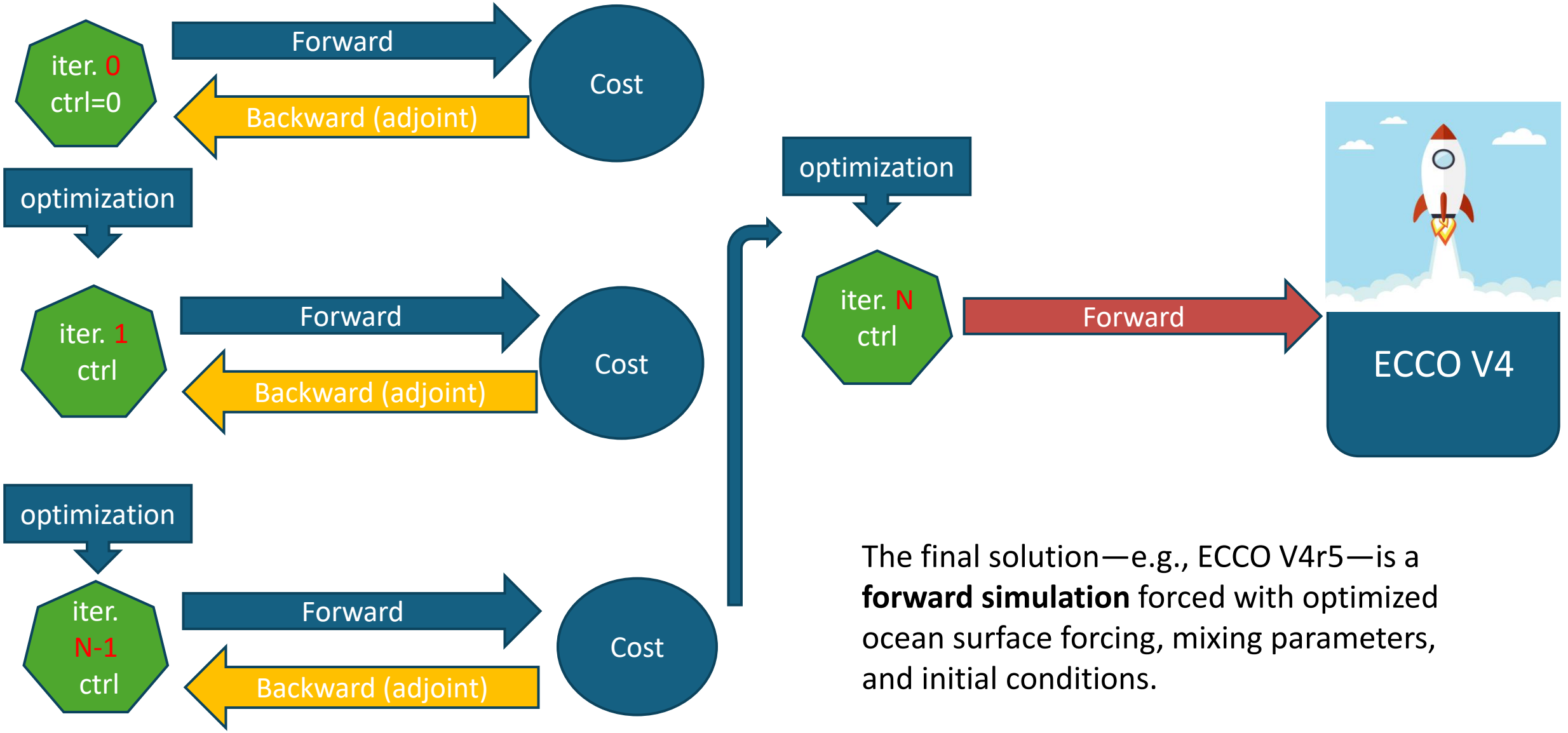
Model Grid: Lat-Lon-Cap90 (LLC90)



Horizontal resolution	22km to 111km
Vertical resolution	10m to 457m from surface to bottom @ 6145m

Forget et al. (2015)

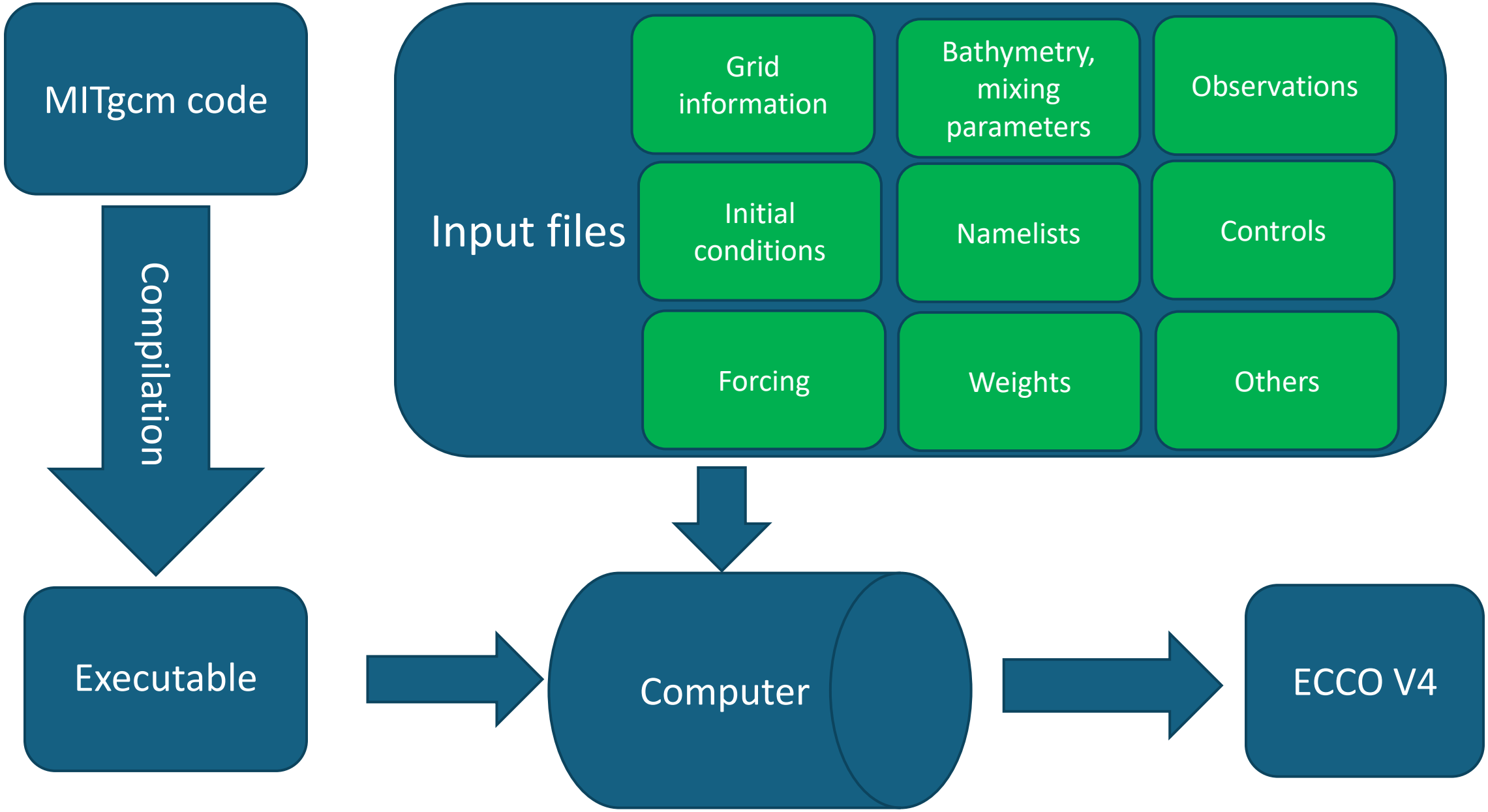
An iterative process to obtain optimized controls



The final solution—e.g., ECCO V4r5—is a **forward simulation** forced with optimized ocean surface forcing, mixing parameters, and initial conditions.

Reproducing ECCO V4 (**Forward Simulation**)

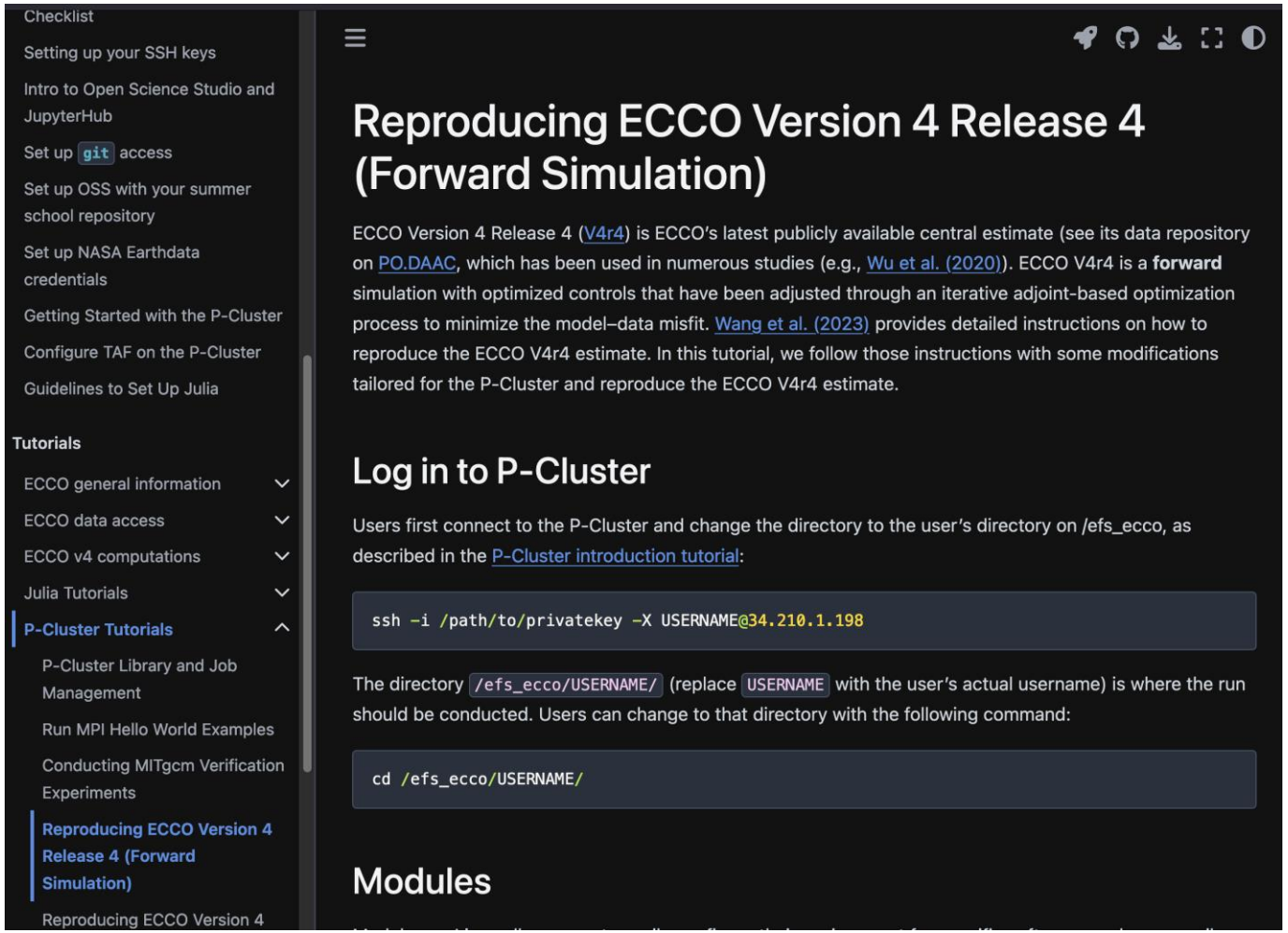
- Why to reproduce ECCO V4
  - Generate different model outputs
    - Sampling frequency from monthly to weekly
    - Output other model fields
  - Forward sensitivity experiments using different forcing, mixing, etc.
    - Climatological forcing
    - Increased/reduced mixing parameters
- Steps to Conduct Forward Run
  - Download the Model Code
  - Obtain Input Files
  - Compile the Code
  - Submit the Forward Run Script


MITgcm code
Compilation
Executable
Input files
Grid information
Bathymetry, mixing parameters
Observations
Initial conditions
Namelists
Controls
Forcing
Weights
Others
Computer
ECCO V4

Tutorial for Reproducing ECCO V4r4

Part of Summer School Tutorials

https://ecco-summer-school.github.io/ecco-2025/tutorials/pcluster/reproducing_v4r4.html



- Using MITgcm version *checkpoint66g* and V4r4-specifc code
- Run with 96 CPUs (For llc90 grid, the globe can be split into 117 30×30 tiles; 21 tiles are over land and therefore skipped).
- Tailored for the P-Cluster
  - Suitable modules (e.g., compilers) have been configured
  - Input files have been pre-downloaded to the P-Cluster
- A general reproduction document for ECCO V4 is available on Zenodo (DOI: 10.5281/zenodo.7789915), , including instructions for downloading the input files.

Tutorial for Reproducing ECCO V4

Part of Summer School Tutorials

https://ecco-summer-school.github.io/ecco-2025/tutorials/pcluster/reproducing_v4r5.html

- Essentially the same procedure as reproducing ECCO V4r4
- Using MITgcm version *checkpoint68g* and V4r5-specifc code
- Run with 113 CPUs; more CPUs are used than in V4r4 because V4r5 includes ice sheets around Antarctica, which introduce more wet tiles.
- Code and input files are all available on the P-Cluster

Steps to Reproduce ECCO V4
(*All commands are available in the reproduction tutorials*)

Log in to the P-Cluster

ssh -i /path/to/privatekey -X USERNAME@34.210.1.198
cd /efs_ecco/USERNAME

Modules

Necessary modules should be automatically loaded. If not, modify your shell configuration file, such as .bashrc, following the instruction described in the tutorial of Getting Started with the P-Cluster

Steps to Reproduce ECCO V4 (continued)

Get the MITgcm model and V4 specific code and namelist files

- The code and namelist files have been downloaded to the P-Cluster. Users can copy them directly to their preferred local directory (see below; replace USERNAME with the user's own username.)
- They are also available on GitHub; see the tutorial for details.
- The namelist files contain run-time parameters.

```
rsync -av /efs_ecco/ECCO/V4/r4/WORKINGDIR /efs_ecco/USERNAME/r4/
```

Current directory structure of /efs_ecco/USERNAME/r4/:

```
WORKINGDIR
├── ECCO-v4-Configurations
├── ECCOV4
│   └── release4
│       ├── code
│       └── namelist
└── MITgcm
```

Steps to Reproduce ECCO V4 (continued)

Input files for atmospheric forcing, initial conditions, and others

- Have also been pre-downloaded into the P-Cluster
- Total data volume is a few hundreds Gigabytes
- **No need to copy them to the user's working directory. The run script will use a symbolic link to access the files.**

```
cd /efs_ecco/USERNAME/r4/
ln -s /efs_ecco/ECCO/V4/r4/input .
```

Current directory structure of /efs_ecco/USERNAME/r4/:

```
┌── WORKINGDIR
│   ├── ECCO-v4-Configurations
│   ├── ECCOV4
│   │   └── release4
│   │       ├── code
│   │       └── namelist
│   └── MITgcm
└── input
```

Steps to Reproduce ECCO V4 (continued)

Compile to generate the executable

For simplicity, assume current directory is **/efs_ecco/USERNAME/r4/**

```
cd WORKINGDIR/ECCOV4/release4
mkdir build
cd build
export ROOTDIR=../../../MITgcm
../../../MITgcm/tools/genmake2 -mods=../code -optfile=../code/linux_ifort_impi_aws_sysmodule -mpi
make depend
make all
cd ..
```

Create a build directory

Create an environment variable that will be used by MITgcm

../../../MITgcm/tools/genmake2 -mods=../code -optfile=../code/linux_ifort_impi_aws_sysmodule -mpi

- Generate a Makefile to be used by make (a build automation tool that generates executable)
- The file linux_ifort_impi_aws_sysmodule, which specifies compilation flags, libraries, etc. is tailored for the P-Cluster.
- The -mpi option indicates that the code will be compiled as an MPI job to run in parallel using multiple processors.

What does Makefile look like?

```
ROOTDIR     = ../../../MITgcm
BUILDDIR    = .
SOURCEDIRS  =  ../code $(ROOTDIR)/pkg/…

EXEDIR      = .
EXECUTABLE  = $(EXEDIR)/mitgcmuv
TOOLSDIR    = $(ROOTDIR)/tools
OADTOOLS    =

ENABLED_PACKAGES =  -DALLOW_CAL -DALLOW_COST…
DISABLED_PACKAGES =  -UALLOW_ADMTLM -UALLOW_AIM_V23…

# Fortran compiler
FC = mpiifort
# Fortran compiler
F90C =
# C compiler
CC = mpiicc
 …
```

Steps to Reproduce ECCO V4 (continued)

- **make depend**: determines code decencies: head files included
- **make all**: generates the executable (named mitgcmuv in build/)

After compilation, directory structure of /efs_ecco/USERNAME/r4/ is as follows:

```
WORKINGDIR
├── ECCO-v4-Configurations
├── ECCOV4
│   └── release4
│       ├── code
│       ├── namelist
│       └── build
└── MITgcm
input
```

mitgcmuv in build

Steps to Reproduce ECCO V4 (continued)

Run

- An example run script is provided in /efs_ecco/ECCO/V4/r4/scripts
- The P-Cluster uses Slurm as its batch job scheduler
- The example script conducts a three-month model integration from January 1, 1992, to March 31, 1992. The script can be modified to perform 26-year (1992-2017) runs over the full ECCO V4r4 model integration period
- sbatch submits a batch job to Slurm, based on the job configuration specified in the script. The job script requests 3 nodes, with each node running 36 tasks.
- Once submitted, Slurm will display a message with the job ID, such as

  Submitted batch job 1181
- Users can monitor the job status by issuing the command squeue

cd WORKINGDIR/ECCOV4/release4
cp -p /efs_ecco/ECCO/V4/r4/scripts/run_script_slurm.bash .
sbatch run_script_slurm.bash

Example run script

/efs_ecco/ECCO/V4/r4/scripts/run_script_slurm.bash

```
#!/bin/bash
#SBATCH -J ECCOv4r4                                  ← Job name
#SBATCH --nodes=3                                    ← 3 nodes
#SBATCH --ntasks-per-node=36                         ← 36 tasks per node
#SBATCH --time=24:00:00                              ← Requesting 24-hour wall time
#SBATCH --exclusive                                  ← Other users cannot use idle CPUs in the 3 nodes
#SBATCH --partition=sealevel-c5n18xl-demand          ← Partition (i.e., queue) name
#SBATCH --mem-per-cpu=1GB                            ← Each CPU has 1GB memory.
#SBATCH -o ECCOv4r4-%j-out                           ← Standard output file is ECCOv4r4-NNN-out, where NNN is the job ID.
#SBATCH -e ECCOv4r4-%j-out                           ← Standard error file is set to be the same as the Standard output file.

# Initialize and set up the environment.             ← Set up environment
umask 022
ulimit -s unlimited
source /etc/profile
source /shared/spack/share/spack/setup-env.sh
source /usr/share/modules/init/sh
```

Example run script (continued)

/efs_ecco/ECCO/V4/r4/scripts/run_script_slurm.bash

```
# Load modules
module purge
module load intel-oneapi-compilers-2021.2.0-gcc-11.1.0-adt4bgf
module load intel-oneapi-mpi-2021.2.0-gcc-11.1.0-ibxno3u
module load netcdf-c-4.8.1-gcc-11.1.0-6so76nc
module load netcdf-fortran-4.5.3-gcc-11.1.0-d35hzyr
module load hdf5-1.10.7-gcc-9.4.0-vif4ht3
module list

# Set environment variables
export FORT_BUFFERED=1
export MPI_BUFS_PER_PROC=128
export MPI_DISPLAY_SETTINGS=""
```

← Load modules

← Set up more environment variables

Example run script (continued)

/efs_ecco/ECCO/V4/r4/scripts/run_script_slurm.bash

Create run directory

```
mkdir -p "${basedir}/run"
cd "${basedir}/run" || exit  # Change directory and exit if it fails
```

← Create run directory

← Change into it

Link input files

```
ln -s ../namelist/* .
ln -s ${inputdir}/input_init/* .
…
ln -s ${inputdir}/input_forcing/control_weights/* .
ln -s ${inputdir}/native_grid_files/tile*.mitgrid .
```

Modify some namelist files, e.g., change the number of time steps to 3-mont

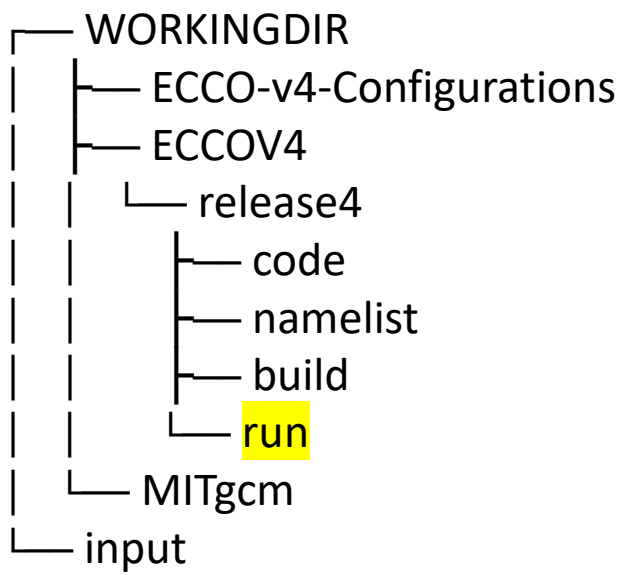
```
unlink data
cp -p ../namelist/data .
sed -i '/#nTimeSteps=2160,/ s/^#//; /nTimeSteps=227903,/ s/^/#/' data
```

Run the mitgcmuv executable with MPI using the specified number of processes

```
mpirun -np "${nprocs}" ./mitgcmuv
```

Steps to Reproduce ECCO V4 (continued)

After the run starts, directory structure of /efs_ecco/USERNAME/r4/ is as follows:



- Use squeue to check the job status.
- After the job completes, check whether it ended normally.
  - Yes, if the last line of run/STDOUT.0000 is
    PROGRAM MAIN: Execution ended Normally
  - Otherwise, the job does not end normally.

Results
WORKINGDIR/run

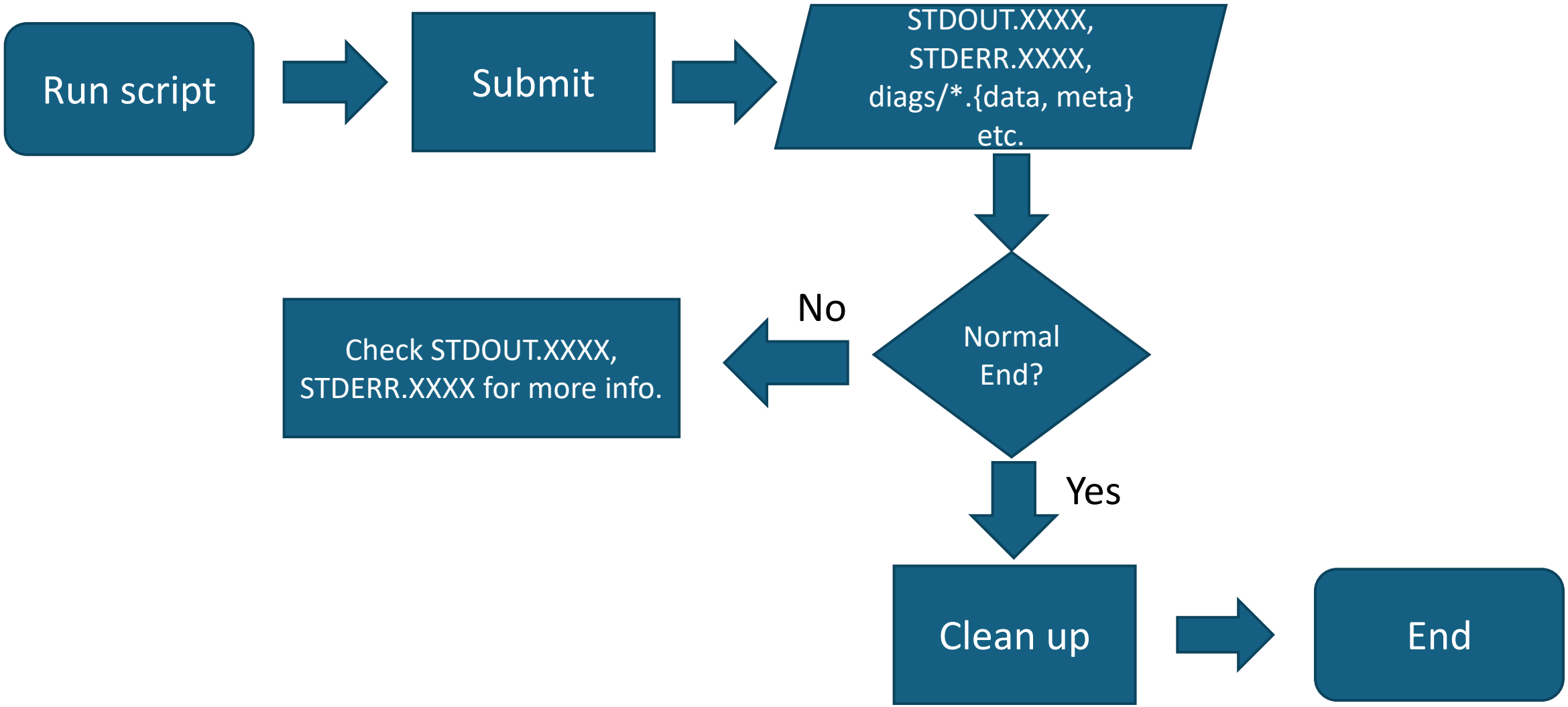
Expected files

files	
STDOUT.XXXX	model configuration, monitored statistics of model state variables
STDERR.XXXX	any warnings
diags/*.{data,meta}	outputs of the model state in binary format
m_*.{data,meta}, misfit*.{data,meta}, …	outputs from pkg/ecco for cost calculation
xx*	control adjustments

Successful if the last line of STDOUT.0000 is:

PROGRAM MAIN: Execution ended Normally

Flowchart for Conducting the Run



Namelist (runtime parameters)

filename	
data	Core runtime parameters
data.cal	Specify model start time
data.pkg	Individual package switch
data.diagnostic	Diagnostics variables and output frequencies
data.exf	Forcing files and formats
data.profiles	In situ files
data.ecco	Cost terms
data.gmredi	GM-Redi parameters
data.seaice	Sea-ice parameters
data.ctrl	Control variables, weights
data.exch2	Tile exchange and blank tile parameters

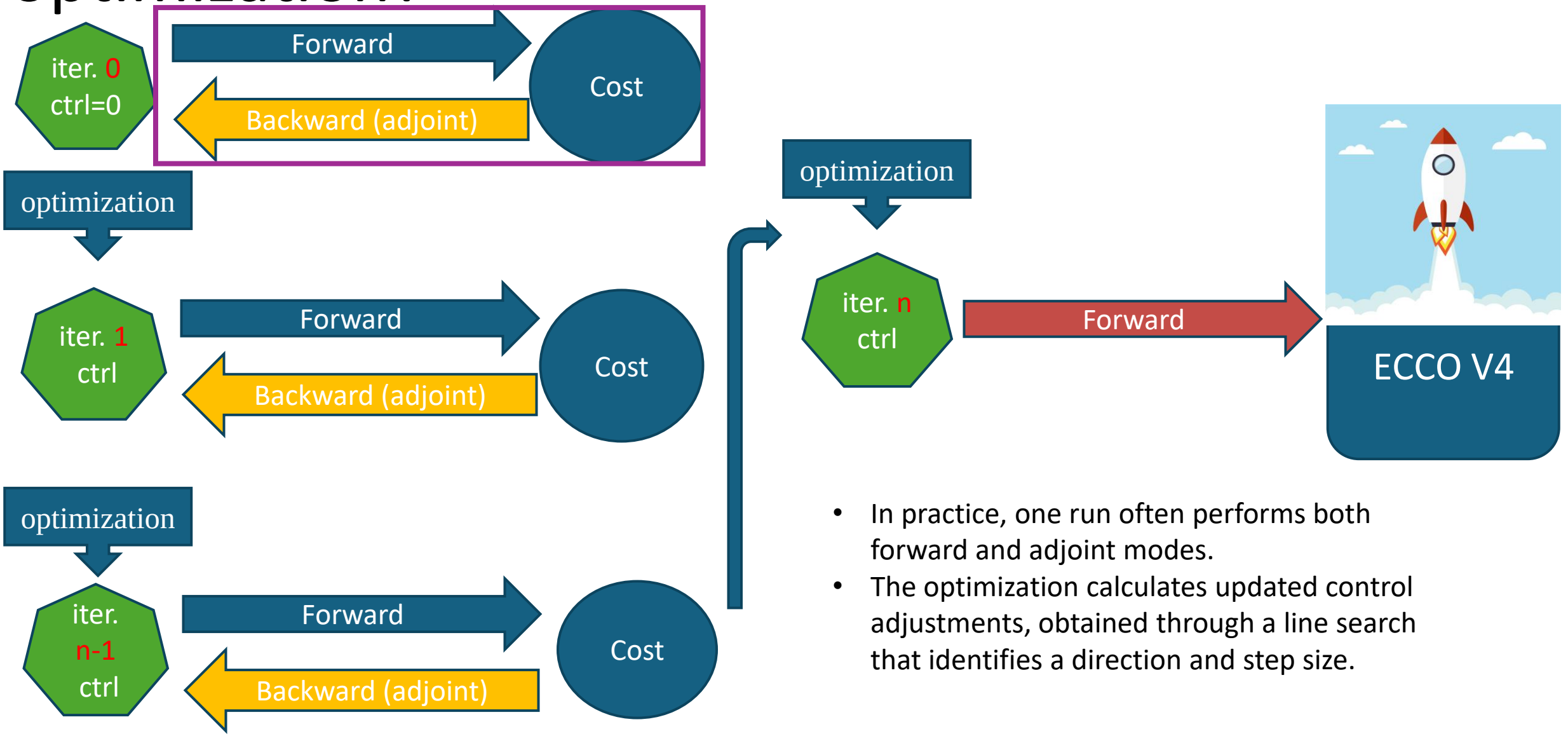
Example modifications to namelist files

filename	
data	nTimeSteps=227903,
	Change the value (# of time steps) changes the model integration period. For example, the model runs for 3 months with: nTimeSteps=2160,
data.diagnostic	frequency(91) = 2635200.0, fields(1,91) = 'SSH', filename(91) = 'diags/SSH_mon_mean/SSH_mon_mean',
	The above outputs monthly-mean SSH with *frequency* specifies averaging period (in seconds). The following changes the namelists above to output weekly-mean OBP: frequency(91) = 604800.0, fields(1,91) = 'OBP', filename(91) = 'diags/OBP_week_mean/OBP_week_mean',
data.exf	atempfile = 'eccov4r4_tmp2m_degC',
	The above specifies the air temperature forcing. Change it to some modified forcing: atempfile = 'eccov4r4_tmp2m_degC_modified',

Things to know to run it on another computer

- Have necessary modules installed
  - Fortran compiler
  - MPI
  - netCDF
- Compilation option file
  - linux_ifort_impi_aws_sysmodule may not work
  - Start from one from the list in MITgcm/tools/build_options/. Pick up one having the same operating system, machine name and compiler in the filename as yours.
- Run out memory?
  - Request more CPUs than the number of tiles

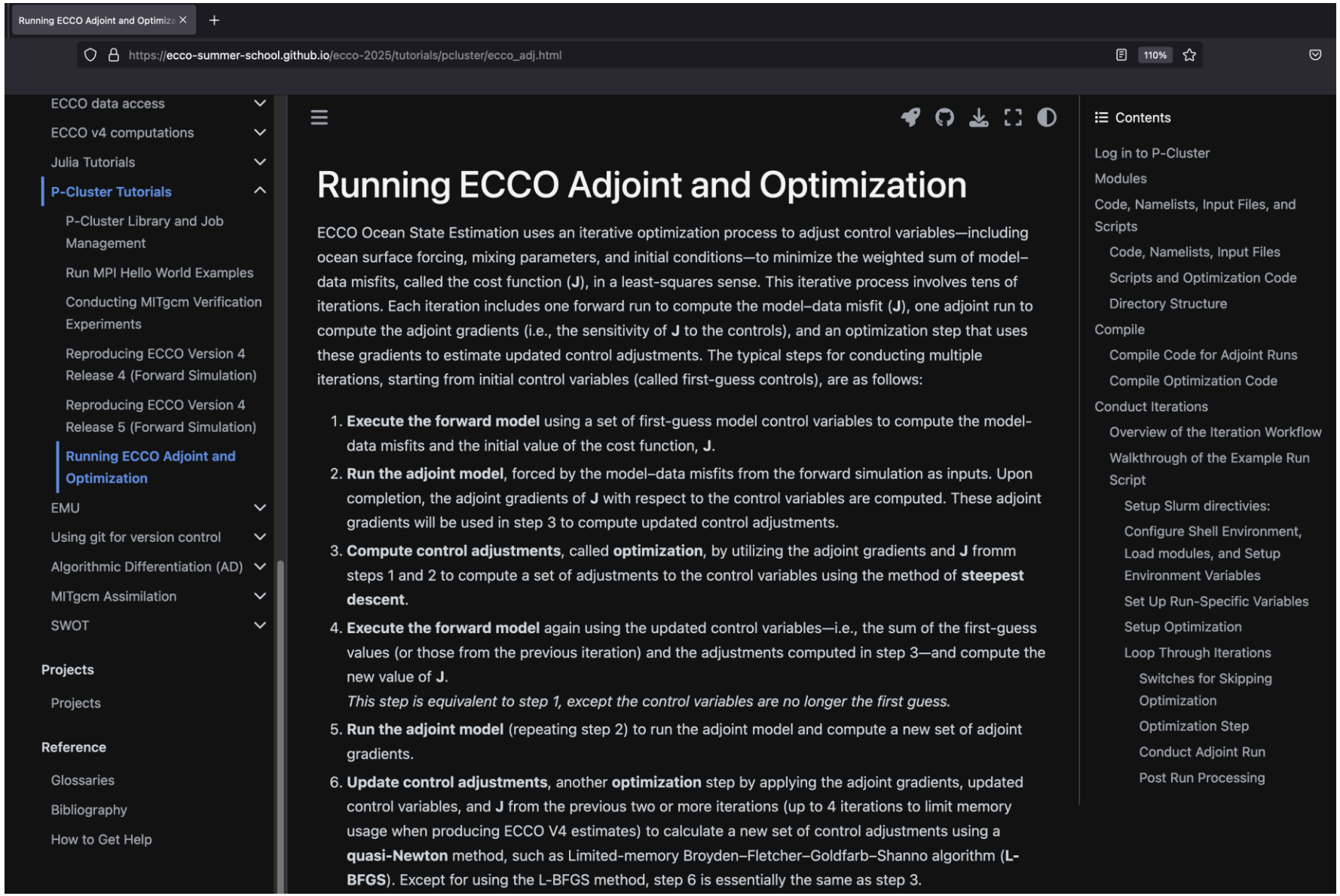
How to run adjoint and conduct optimization?



- In practice, one run often performs both forward and adjoint modes.
- The optimization calculates updated control adjustments, obtained through a line search that identifies a direction and step size.

Tutorials for Running ECCO Adjoint

https://ecco-summer-school.github.io/ecco-2025/tutorials/pcluster/ecco_adj.html



Flowchart diagram for the iterative optimization process

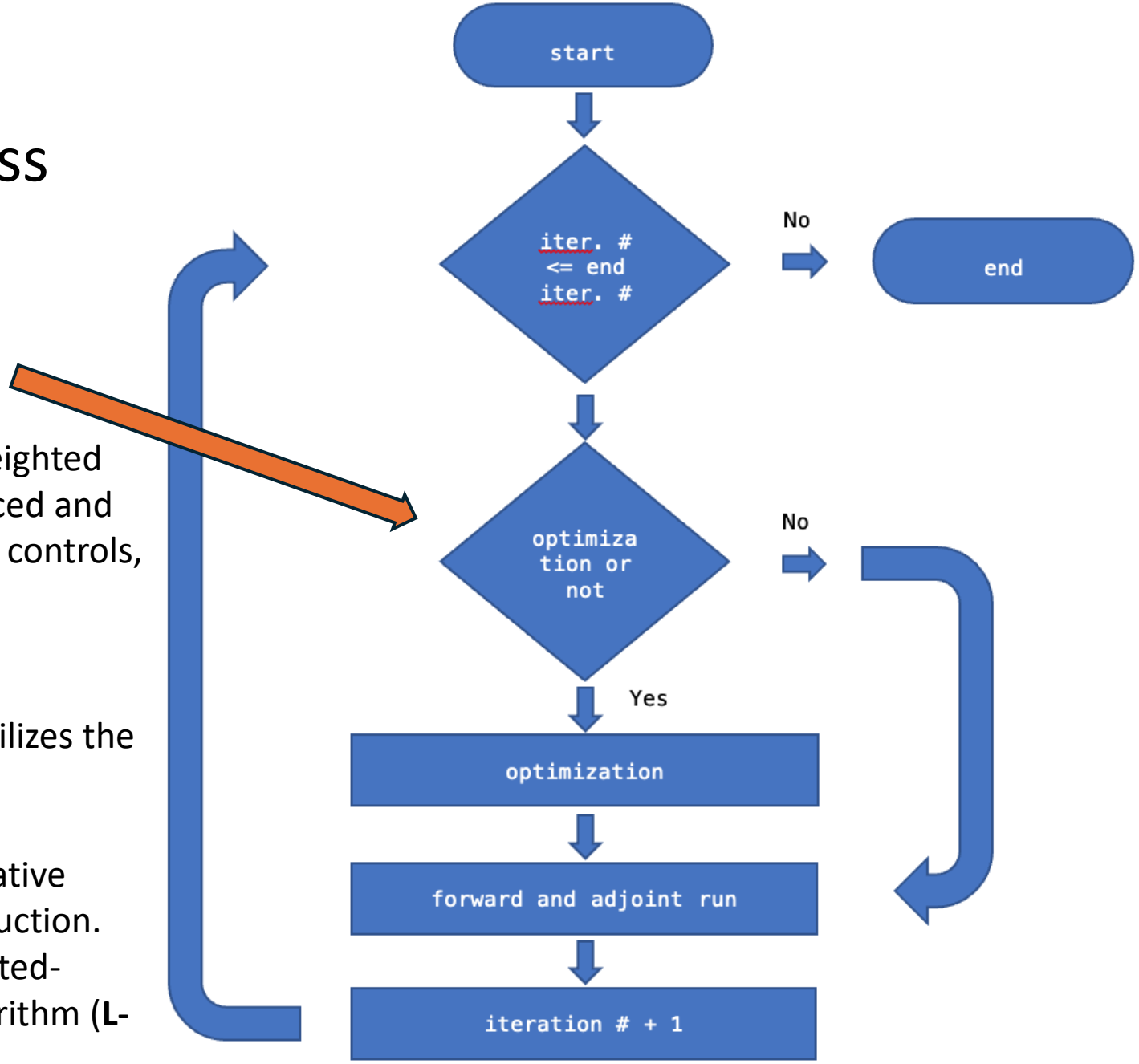
What is the optimization step?

A line search identifies a direction along which the weighted squared sum of mode-data difference (J) can be reduced and then calculates a step size by a specific amount to the controls, in order to to reduce J.

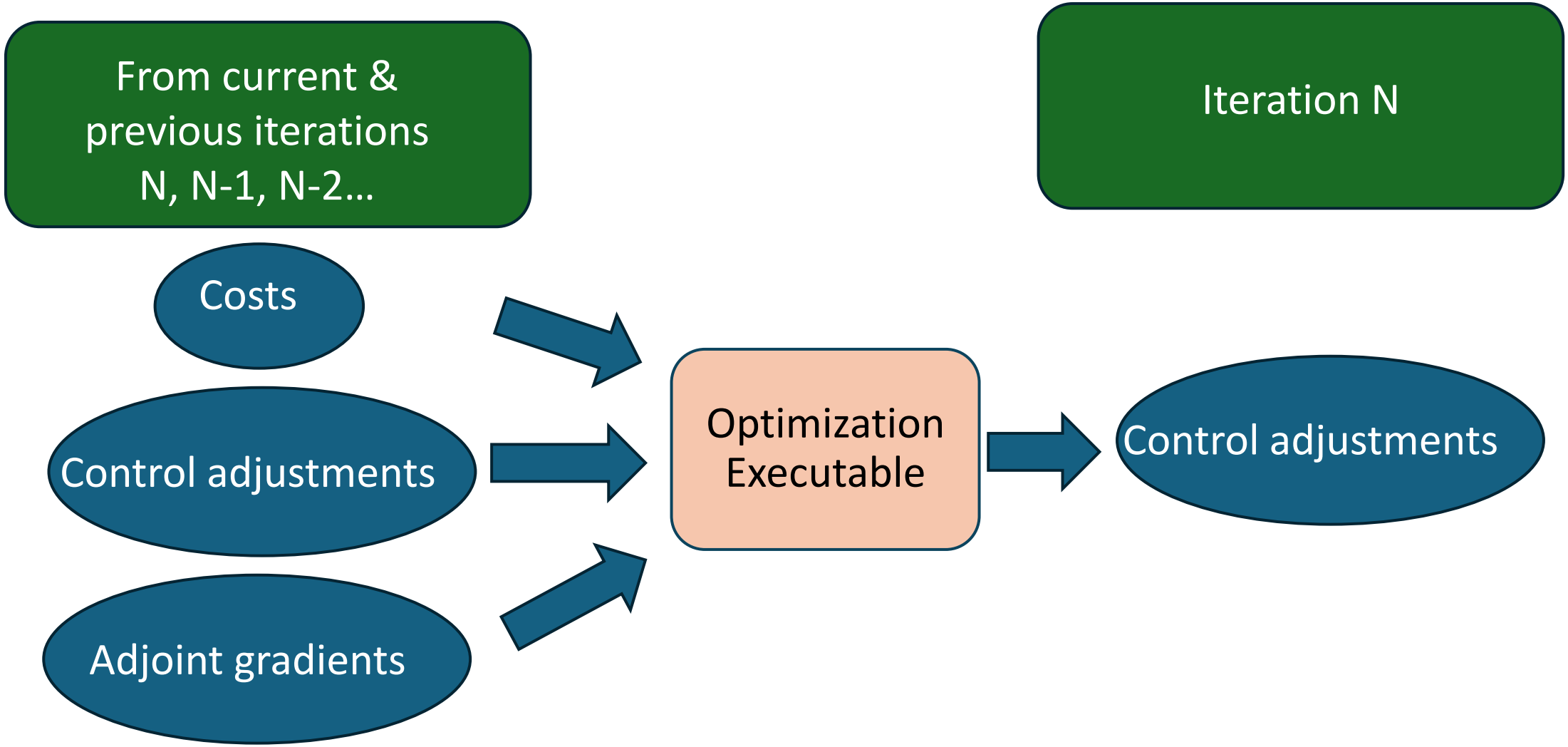
Two linear search methods:

- Steepest descent: a first-derivative method that utilizes the gradient as the search direction

- Quasi-Newton method that uses the second-derivative (curvature) information often achieves faster J reduction. ECCO V4 uses one of Quasi-Newton methods: Limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (**L-BFGS**).



During optimization step …

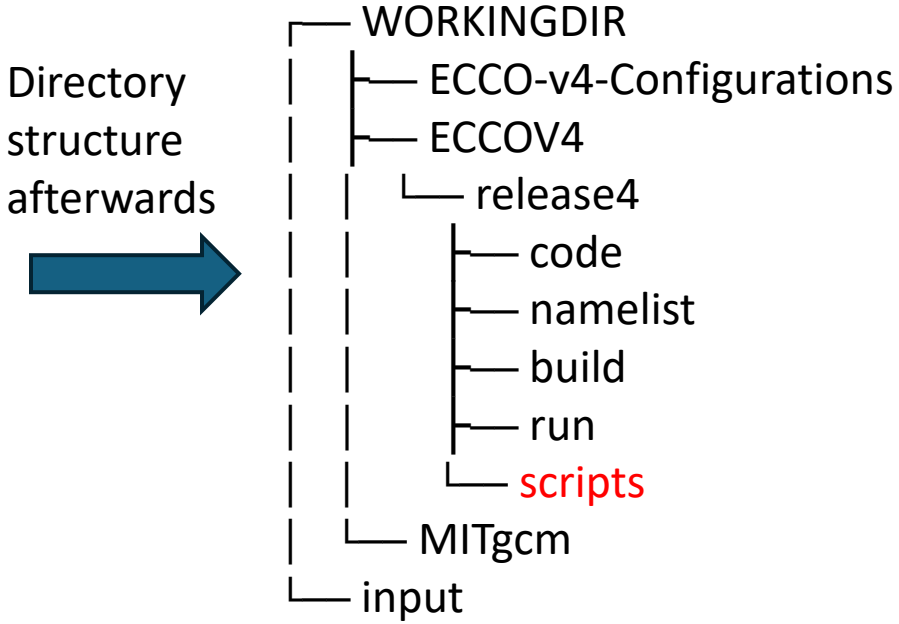


Practical Steps

- Follow the same steps as the forward simulation to obtain the code, input files, etc.
- Compilation is similar to that for the forward simulation, but with a few important differences — including sending the Fortran code to TAF to generate the adjoint code.
- Obtain more scripts that are needed for conducting a few iterations.

```
cd WORKINGDIR
cp -r "ECCO-v4-Configurations/ECCOv4 Release
4/scripts/" ECCOV4/release4/
```

Directory structure afterwards

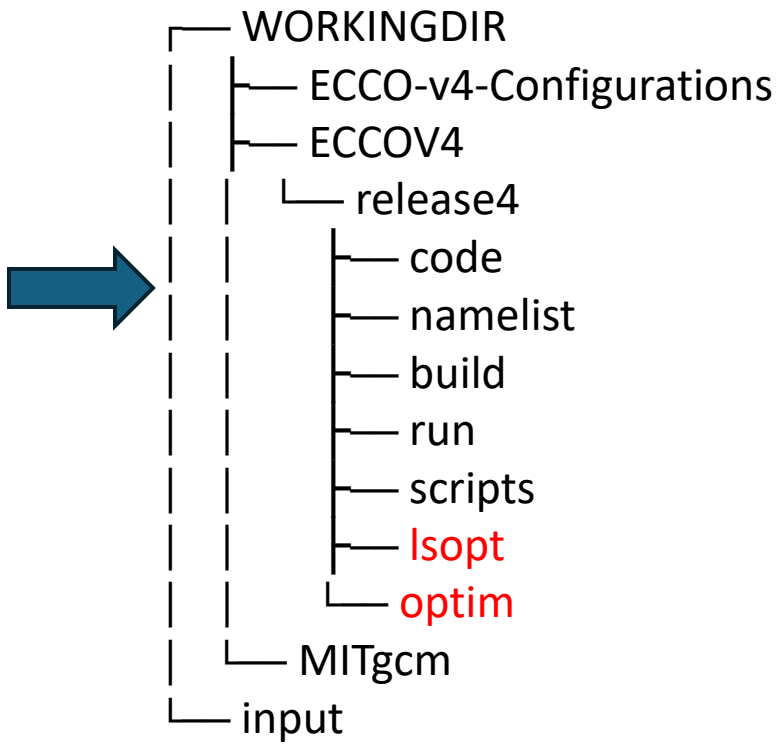


Obtain Optimization Code

Assume current directory is **/efs_ecco/USERNAME/r4/**

```
cd WORKINGDIR
cp -r "ECCO-v4-Configurations/ECCOv4 Release 4/optimization/lsopt/" ECCOV4/release4/
cp -r "ECCO-v4-Configurations/ECCOv4 Release 4/optimization/optim/" ECCOV4/release4/
```

Directory structure with the two new directories are highlighted in red



Compile Optimization Code

```
cd WORKINGDIR/ECCOV4/release4
cd lsopt
make clean
make
cd ../optim
make clean
make
cd ..
```

- The executable would be optim.x in ***WORKINGDIR/ECCOV4/release4/*optim/**.
- It will be used for linear search during the optimization step.

Compilation for adjoint run

Compile to generate the executable

For simplicity, assume current directory is **/efs_ecco/USERNAME/r4/**

- Create a build_ad directory to differentiate it from the build directory used for the forward simulation.
- Generate the adjoint code by sending the Fortran code to TAF — this is done in the step make -j16 adtaf.
- Generate the executable by make –j16 adall; the executable will be build_ad/mitgcm_uv

```
cd WORKINGDIR/ECCOV4/release4
mkdir build_ad
cd build_ad
export ROOTDIR=../../../MITgcm
../../../MITgcm/tools/genmake2 -mods=../code -optfile=../code/linux_ifort_impi_aws_sysmodule -mpi
make depend
make adtaf
make adall
cd ..
```

make adtaf: send code to FastOpt for TAF to generate the adjoint code

- Makefile defines how to prepare the code (not all subroutines need to be sent to TAF) and provides instructions for sending it to TAF
- ad_input_code.f: packaged code sent to TAF
- ad_input_code_ad.f: adjoint code returned by TAF
- ad_taf_output.f: Same as ad_input_code_ad.f but with some small changes

make adtaf: sample on-screen messages are as follows

staf -server fastopt.net -f77 -reverse -i4 -r4 -intrinsic system,flush -l taf_ad.log -toplevel 'the_main_loop' -input 'xx_theta_dummy … xx_genarr2d_dummy,xx_genarr3d_dummy,xx_gentim2d_dummy,… xx_vwind_mean_dummy' -output 'fc' ad_input_code.f

Transformation of Algorithms in Fortran (TAF)
Copyright 2000-2019 FastOpt GmbH, Hamburg, Germany
All rights reserved.
URL: http://www.FastOpt.de, Email: info@FastOpt.de
script to access TAF remotely version 5.2

Processing files at fastopt.net, please wait.

reverse: tell TAF to generate adjoint code

'the_main_loop': routine to be processed

xx_gentim2d_dummy: control

'fc':cost function (**J**)

Transformation of Algorithms in Fortran (TAF) Version 6.8.2
Copyright 2000-2025 FastOpt GmbH, Hamburg, Germany
All rights reserved.
URL: http://www.FastOpt.de, Email: info@FastOpt.de

TAF needed 2.43910E+02 seconds

ls -l ad_input_code_ad.f
-rw-r--r-- 1 owang owang 11902820 May 21 05:37 ad_input_code_ad.f
cat ad_input_code_ad.f | sed -f ../../../MITgcm/tools/adjoint_sed > ad_taf_output.f

make adall: use ad_taf_output.f and other subroutines that were not sent to TAF for translation to generate the executable:
mitgcmuv_ad

Example subroutines (S/R) not “TAFed”:

- S/Rs to output diagnostics
- S/Rs that are equivalent of an operation by a symmetric matrix (a transpose of a symmetric matrix is itself)
- S/Rs too complicated for TAF; human intervention is necessary!

Code snippet to illustrate forward code and TAF-generated adjoint code

Original MITgcm Code (Forward)

```
      do k = 1, ksize
        do j = 1-oly, sny+oly
          do i = 1-olx, snx+olx
            ab_gtr(i,j) = ab0*gtracer(i,j,k)+ab1*gtrnm(i,j,k,bi,bj,m1)
     $+ab2*gtrnm(i,j,k,bi,bj,m2)
            gtrnm(i,j,k,bi,bj,m2) = gtracer(i,j,k)+ab_gtr(i,j)
          end do
        end do
      end do
```

An example to help explain the TAF translation

Rewrite forward code:

```
gtrnm(i,j,k,bi,bj,m2) = gtracer(i,j,k)+ab_gtr(i,j)
```

To:

```
gtrnm(i,j,k,bi,bj,m2) = 0.d0
gtrnm(i,j,k,bi,bj,m2) = gtrnm(i,j,k,bi,bj,m2)+gtracer(i,j,k)
gtrnm(i,j,k,bi,bj,m2) = gtrnm(i,j,k,bi,bj,m2)+ab_gtr(i,j)
```

TAF-generated adjoint code

```
      do k = ksize, 1, -1
        do j = 1-oly, sny+oly
          do i = 1-olx, snx+olx
            ab_gtr_ad(i,j) = ab_gtr_ad(i,j)+gtrnm_ad(i,j,k,bi,bj,m2)
            gtracer_ad(i,j,k) = gtracer_ad(i,j,k)+gtrnm_ad(i,j,k,bi,bj,m2)
            gtrnm_ad(i,j,k,bi,bj,m2) = 0.d0
            gtracer_ad(i,j,k) = gtracer_ad(i,j,k)+ab_gtr_ad(i,j)*ab0
            gtrnm_ad(i,j,k,bi,bj,m1) = gtrnm_ad(i,j,k,bi,bj,m1)+
     $ab_gtr_ad(i,j)*ab1
            gtrnm_ad(i,j,k,bi,bj,m2) = gtrnm_ad(i,j,k,bi,bj,m2)+
     $ab_gtr_ad(i,j)*ab2
            ab_gtr_ad(i,j) = 0.d0
          end do
        end do
      end do
```

TAF Store directives: An example

Original MITgcm Code (Forward)

```
      DO k=Nr,2,-1
CADJ STORE salt (:,:,k-1,bi,bj) = comlev1_bibj_k,
CADJ &     key=kkey, kind = isbyte
        CALL FIND_RHO_2D(
     I    iMin, iMax, jMin, jMax, k,
     I    theta(1-OLx,1-OLy,k-1,bi,bj),
     I    salt (1-OLx,1-OLy,k-1,bi,bj),
     O    rhoKm1,
     I    k-1, bi, bj, myThid )
       ENDDO
```

TAF-translated Code (Forward)

```
      DO k=Nr,2,-1
         do ip2 = 1-oly, sny+oly
           do ip1 = 1-olx, snx+olx
             comlev1_bibj_k_salt_109h(ip1,ip2,kkey) = salt(ip1,
    $ip2,k-1,bi,bj)
           end do
         end do
         call find_rho_2d( imin,imax,jmin,jmax,k,theta(1-olx,1-
    $oly,k-1,bi,bj),salt(1-olx,1-oly,k-1,bi,bj),rhokm1,help_l,bi,bj,
    $mythid )
       ENDDO
```

TAF-translated Code (Adjoint)

```
      DO k=2, Nr
         do ip2 = 1-oly, sny+oly
           do ip1 = 1-olx, snx+olx
             salt(ip1,ip2,k-1,bi,bj) = comlev1_bibj_k_salt_109h(
    $ip1,ip2,kkey)
           end do
         end do
         call find_rho_2d_ad( imin,imax,jmin,jmax,k,theta(1-olx,
    $1-oly,k-1,bi,bj),theta_ad(1-olx,1-oly,k-1,bi,bj),salt(1-olx,1-oly,
    $k-1,bi,bj),salt_ad(1-olx,1-oly,k-1,bi,bj),rhokm1_ad,bi,bj )
       ENDDO
```

- TAF store directives inserted because salt gets overwritten during time stepping.
- TAF translates the store directive to Fortran code:
  comlev1_bibj_k_salt_109h(ip1,ip2,kkey) = salt(ip1,ip2,k-1,bi,bj)
- Somewhere later (not shown), comlev1_bibj_k_salt_109h gets written to disk.
- The adjoint code loads it back from disk and assign salt to it.

Conduct three iterations from cold start with a simple cost (all control adjustments start from 0)

cd WORKINGDIR/ECCOV4/release4
cp -p /efs_ecco/ECCO/V4/r4/scripts/run_script_slurm_autoopt_coldstart_v4r4.bash .
sbatch run_script_slurm_autoopt_coldstart_v4r4.bash

Directory name	Description
v4r4_coldstart.iter0	run directory for iteration 0
v4r4_coldstart.iter1	run directory for iteration 1
v4r4_coldstart.iter2	run directory for iteration 2
ctrlvec.v4r4_coldstart	files for control adjustments (generated by optimization) and adjoint gradients (generated by each iteration and copied over)
optim.v4r4_coldstart	Where the optimization is conducted, using input files loaded from ctrlvec.v4r4_coldstart and output the new control adjustments for the next iteration to ctrlvec.v4r4_coldstart. Information of previous iterations is saved in OPWARMD & OPWARMI.

A few more details:

- A fractional cost reduction target specified (0.4%)The script figures out the actual target cost value
- ecco_cost (adjoint gradients in packed format) generated (and also ecco_ctrl (control adjustments for iteration 0) at the end of each iteration;

  Examples:
  - ecco_cost_MIT_CE_000.opt0000 and ecco_ctrl_MIT_CE_000.opt0000
  - Unpacked control files xx.*.data,
  - Unpacked adjoint gradients: adxx.*.data
- The ecco_cost and ecco_ctrl files from previous iterations are used as inputs for the optimization to generate new control adjustments for the next iteration, such as ecco_ctrl_MIT_CE_000.opt0001
- The new control adjustments are used as part of the input for the next iteration, and the iterative process continues

Results: Cost vs. Iteration

Iteration Number	Cost	Cost Ratio w.r.t. Iteraion0
0	**2079042.47585259**	1
1	2070774.52785874	0.996
2	1284971.72606061	0.618

Steepest descent

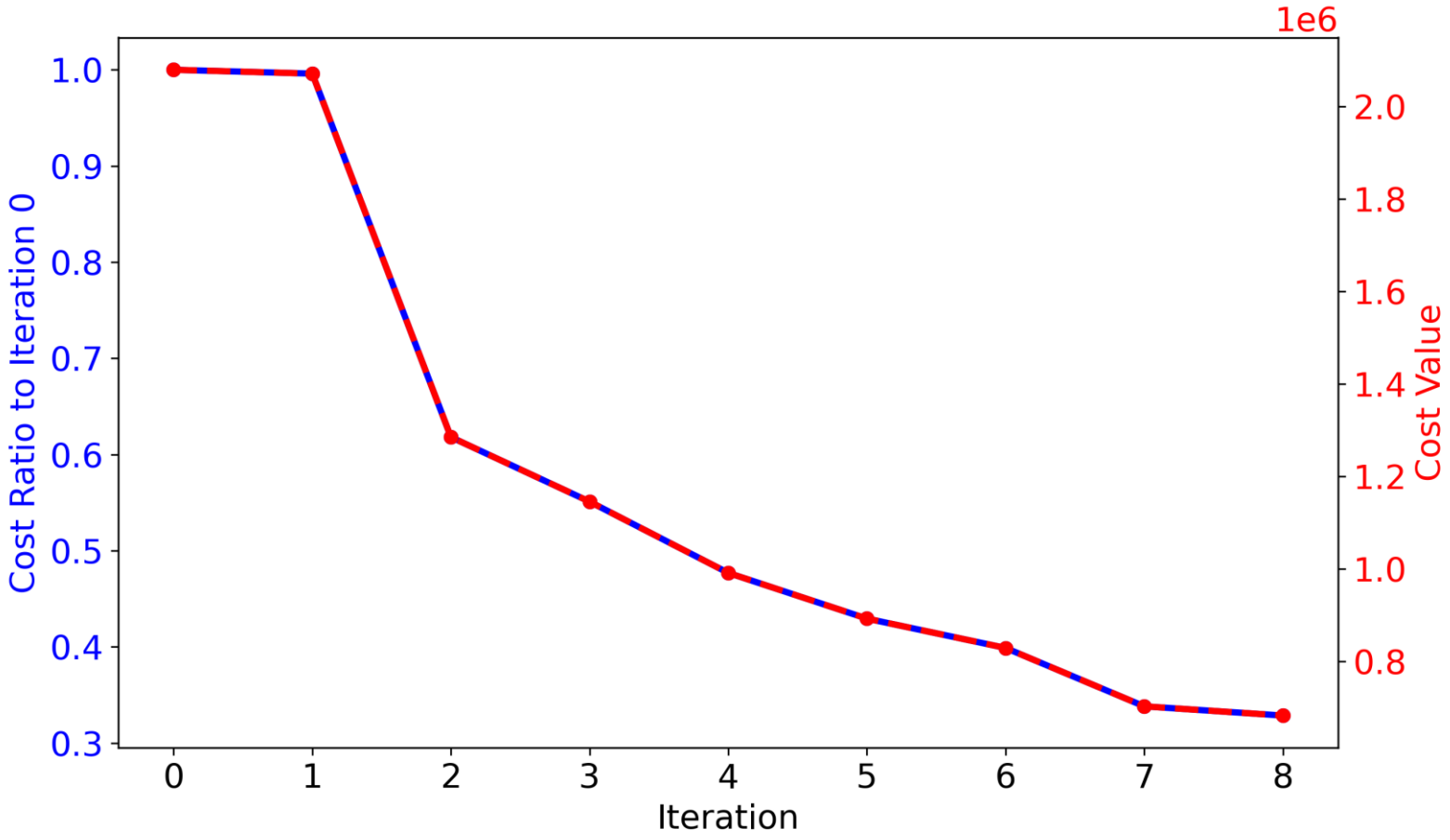
Quasi-Newton

- For steepest descent, we specify a cost reduction target of 0.004. The actual cost reduction from iterations 0 to 1 is also 0.004, matching the specified cost reduction target.

Cost values in the table above are the **fc** values in the cost function file in each run directory, e.g., v4r4_coldstart.iter0/costfunction0000 (see first few lines below):

fc = **2079042.47585259** 0.0000000E+00
siv4-conc (gencost 1) = 0.787294673837671D+05 0.130851000000000D+06
siv4-deconc (gencost 2) = 0.124716306787977D+05 0.236400000000000D+04
siv4-exconc (gencost 3) = 0.335788181304271D+04 0.557200000000000D+04
…

Cost vs. Iterations



- Iteration 8 cost is 30% of iteration 0
- Small cost reduction from iteration 0 to 1 is consistent with specified cost reduction target
- Large cost reduction from iteration 1 to 2, when line search was switched from steepest descent to Quasi-Newton

Thank you!

Questions:
ecco-support@mit.edu
(please subscribe via
http://mailman.mit.edu/mailman/listinfo/ecco-support)