MITgcm SEAICE manual

Tim Hill

August 17, 2018

Contents

1 Introduction

This document aims to serve as a starting point for running the MITgcm in configurations with ice. I have the model at a functional point and hope to leave some resources so it is easy to pick up where I left off.

The first section serves as an introduction to MITgcm and should be a good resource to carry out the first few functional model runs, including running with the ice package.

The next few sections outline some specific things I learned about the model, and may be useful to someone trying to extend the model past what is outlined here. The tools I created for visualizing results are described. Some specific model configurations that did not work are explained, and problems I encountered with the model or with the Graham environment are discussed.

2 Setting up the MITgcm

The most useful references for the MITgcm are:

- The fluids wiki has a wealth iof information about computing resources, running on the SHARCNET cluster, and has an MITgcm tutorial (discussed later).
- The MITgcm homepage
- The new MITgcm documentation is on read the docs, but is incomplete at the time of writing. The old documentation is still accessible but has some unfortunate qualities, including embedding tabels as images so they are not searchable.
- The MITgcm github repo is where I usually look at and search through the model code if I need to.

2.1 Basic tutorial

The Fluids wiki tutorial is a good tutorial to start with the model. The tutorial was originally written by Emily Tyhurst, and I have corrected a few mistakes in the tutorial.

The tutorial is largely self-contained, but I want to say a word about directory structure. MITgcm expects your case files to be contained in the same directory as the model. For example, if you make a folder called `simulation1` for your simulation, this folder might be in the same directory as the MITgcm directories `doc/`, `eesupp/`, `pkg/`, `model/`, etc., in which case the directory structure would look like

```
/home/user/MITgcm

      simulation1/

      doc/
      eesupp
      jobs/
      lsopt/
      model/
```

```
        optim/
        pkg/
        tools/
        utils/
        verification/
        ...
```

I had a folder I called `MITgcmdata/` which was a git repo that contained all my model runs. Then my directory tree was

```
/home/user/MITgcm

        MITgcmdata/
            simulation1
            simulation2
            ...

        doc/
        eesupp
        jobs/
        lsopt/
        model/
        optim/
        pkg/
        tools/
        utils/
        verification/
        ...
```

Having a structure like this makes compiling the model much easier. You can in theory compile the model from anywhere as long as you point the `genmake2` script to the model directories, but in practice this has been harder than expected.

2.2 SEAICE tutorial

The SEAICE package 3.1.2 provides a dynamic and thermodynamic sea-ice model for the MITgcm. This package is general purpose enough to model freshwater and ocean ice. The package usually does a good job (at least qualitatively) of recreating dynamics and ice quantities reasonably physically.

Write the rest of this up for small processor numbers (ie HOOD)

3 More details about running MITgcm

This section tries to describe aspects of running the MITgcm in more detail than the previous section. Hopefully this will be a good resource when trying to extend the model past what is included in the tutorials.

3.1 Available packages

MITgcm is structured as a core model + packages. The only packages within my scope are the seaice, external forcing, calendar, and diagnostic packages. For each package, it must be enabled at both compile-time and at run-time.

At compile-time, the package must be included in the `packages.conf` file. Then at run-time the package must be included in the packages namelist in `data.pkg`. For example, see listing 1.

```
# Packages
 &PACKAGES
 useEXF=.TRUE.,
 useCAL=.TRUE.,
# useTHSICE=.TRUE.,
 useSEAICE=.TRUE.,
 useDIAGNOSTICS=.TRUE.
 &
```

Listing 1: Sample data.pkg file. This would enable the EXF, CAL, SEAICE, and DIAGNOSTICS packages, and disable the THSICE package.

The rest of this section describes the additional packages available for MITgcm.

3.1.1 DIAGNOSTICS

The diagnostics package exists to make it easier to control the output files generated by the model. It is easy to choose which fields are outputted and how frequently files are saved. As usual, the package requires compile-time and run-time options.

Compile time options

At compile time (probably in your `code` directory), you should have the file `DIAGNOSTICS_SIZE.h` (example below). This can be copied from the DIAGNOSTICS package code, `pkg/diagnostics/DIAGNOSTICS_SIZE.h`.

```
C     Diagnostics Array Dimension
C     ---------------------------
C     ndiagMax   :: maximum total number of available diagnostics
C     numlists   :: maximum number of diagnostics list (in data.diagnostics)
C     numperlist :: maximum number of active diagnostics per list (data.
     diagnostics)
C     numLevels  :: maximum number of levels to write    (data.diagnostics)
C     numdiags   :: maximum size of the storage array for active 2D/3D
     diagnostics
C     nRegions   :: maximum number of regions (statistics-diagnostics)
C     sizRegMsk  :: maximum size of the regional-mask (statistics-diagnostics)
C     nStats     :: maximum number of statistics (e.g.: aver,min,max ...)
C     diagSt_size:: maximum size of the storage array for statistics-diagnostics
C Note : may need to increase "numdiags" when using several 2D/3D diagnostics,
C  and "diagSt_size" (statistics-diags) since values here are deliberately small.
      INTEGER    ndiagMax
      INTEGER    numlists, numperlist, numLevels
      INTEGER    numdiags
      INTEGER    nRegions, sizRegMsk, nStats
      INTEGER    diagSt_size
      PARAMETER( ndiagMax = 500 )
      PARAMETER( numlists = 25, numperlist = 50, numLevels=2*Nr )
      PARAMETER( numdiags = 25*Nr )
      PARAMETER( nRegions = 0 , sizRegMsk = 1 , nStats = 4 )
      PARAMETER( diagSt_size = 10*Nr )


CEH3 ;;; Local Variables: ***
```

```
27 CEH3 ;;; mode:fortran ***
28 CEH3 ;;; End: ***
```

Listing 2: Example `DIAGNOSTICS_SIZE.h` file

Parameter `numlists` (line 20) and `numdiags` (line 21) can be adjusted if you need more diagnostics.

Run time options

The diagnostics you want to output, and how frequently to write files are specified at runtime in file `data.diagnostics`. Specify the field names in `field` and `filename` lines, and set the frequency. See the following example. The file needs namelists `&DIAGNOSTICS_LIST` (for writing an enitre field to file) and `DIAG_STATIS_PARMS` (for writing per-level statistics), even if one of them is empty.

```
1  # Diagnostic Package Choices
2  #-----------------
3  # for each output-stream:
4  #  filename(n) : prefix of the output file name (only 8.c long) for outp.stream n
5  #  frequency(n):< 0 : write snap-shot output every |frequency| seconds
6  #               > 0 : write time-average output every frequency seconds
7  #  timePhase(n)     : write at time = timePhase + multiple of |frequency|
8  #  averagingFreq(n) : frequency (in s) for periodic averaging interval
9  #  averagingPhase(n): phase     (in s) for periodic averaging interval
10 #  repeatCycle(n)   : number of averaging intervals in 1 cycle
11 #  levels(:,n) : list of levels to write to file (Notes: declared as REAL)
12 #                when this entry is missing, select all common levels of this
       list
13 #  fields(:,n) : list of diagnostics fields (8.c) (see "available_diagnostics.log
       "
14 #                file for the list of all available diag. in this particular
       config)
15 #-----------------
16  &DIAGNOSTICS_LIST
17 # diag_mnc     = .FALSE.,
18 # dumpAtLast   = .TRUE.,
19 #-----------------------------
20   frequency(1) = -3600.0,
21   timePhase(1) = 0,
22   fields(1, 1) = 'THETA',
23   filename( 1) = 'T',
24 #-----------------
25   frequency(2) = -3600.0,
26   timePhase(2) = 0,
27   fields(1, 2) = 'UVEL',
28   filename( 2) = 'U',
29 #-----------------
30  ...
31   &
32
33   &DIAG_STATIS_PARMS
34   &
```

Listing 3: Example `data.diagnostics` file

For a real model run we would likely want to include more diagnostics by continuing to add to the namelist `DIAGNOSTICS_LIST` in the same way. The full list of diagnostics is listed in the documentation for each package, as well as in the `available_diagnostics.log` file created when you run with

the diagnostics package. See the examples in the tutorials for a full working example.

3.1.2 SEAICE package

The SEAICE package includes physical parameterizations for ice dynamics and simple thermodynamics. The dynamical model is described in more detail in section 7.6. This section focuses on how to set up and use the SEAICE package.

SEAICE has both compile-time and run-time options, like other packages.

Compile-time options

The compile-time options come from the two header files `SEAICE_OPTIONS.h` and `SEAICE_SIZE.h`. See the files in the SEAICE tutorial, verification experiments, or in the package code `pkg/seaice/SEAICE_OPTIONS.h`.

Run-time options

Run-time options are specified in the file `data.seaice`. The SEAICE package has some questionable default values (see this GitHub issue). The simplest file that fixes the default values is listing 4. This file would be suitable for starting a simulation with no ice, since the initial ice area fraction, thickness, and snow files are not specified.

```
# SEAICE parameters
 &SEAICE_PARM01
# There are some problems with SEAICE pkg defaults, see https://github.com/MITgcm/MITgcm/issues/107
 SEAICEuseDYNAMICS     = .TRUE.,
 SEAICEscaleSurfStress = .TRUE.,
 SEAICE_useMultDimSnow = .TRUE.,
 SEAICEetaZmethod      = 3,
 SEAICE_drag           = 0.001,
 MIN_LWDOWN            = 0.0,
 SEAICE_area_reg       = 1.0E-5,
 SEAICE_hice_reg       = 1.0E-5,
# These three settings let ice be advected physically by the wind/surface currents
 SEAICEadvHEFF         = .TRUE.,
 SEAICEadvAREA         = .TRUE.,
# Always either use 33 or 77!
 SEAICEadvScheme       = 33,
 &

 &SEAICE_PARM03
 &
```

Listing 4: Recommended parameters for data.seaice file

Initial ice conditions can be specified as a constant value, or with an input file. To specify an initial constant value, set the keywords `SEAICE_initialHEFF`,

SEAICE_initialAREA. To set with a file, set the input files HeffFile, AreaFile, HsnowFile, HsaltFile to initialize the initial ice thickness, fractional ice covered area, snow thickness, and salinity. Keywords should be in namelist SEAICE_PARM01. See the SEAICE tutorial for an example with initial ice configuration (Case 2).

3.1.3 EXF package

The external forcing (EXF) package allows the model to be forced with real (or simulated) observations of meteorological data. This package has temporal interpolation abilities, which would be convenient when working with real observations.

As with all packages, EXF requires compile-time and run-time options.

Compile-time options

The compile-time options are specified in file EXF_OPTIONS.h. See the file in the SEAICE tutorial, or copy from the exf package, pkg/exf/EXF_OPTIONS.h.

Run-time options

The run-time options are specified in file data.exf, see example in listing 5. Fields that should be included to force the model are in table 1

Field name	Description	Typical range
uwind	Surface (10-m) zonal winds, $\mathrm{m\,s^{-1}}$	$\lvert\texttt{uwind}\rvert \leq 10$
vwind	Surface (10-m) merodional winds, $\mathrm{m\,s^{-1}}$	$\lvert\texttt{vwind}\rvert \leq 10$
atemp	Surface (2-m) air temperature, K	$200 \leq \texttt{atemp} \leq 300$
aqh	Surface (2-m) specific humidity, $\mathrm{kg\,kg^{-1}}$	$0 \leq \texttt{aqh} \leq 0.02$
swdown	Downward shortwave radiation, $\mathrm{W\,m^{-1}}$	$50 \leq \texttt{swdown} \leq 450$
lwdown	Downward longwave radiation, $\mathrm{W\,m^{-1}}$	$50 \leq \texttt{lwdown} \leq 450$

Table 1: Required fields to force MITgcm

Initial state files are set in EXF_NML_02 in data.exf with the keyword [fieldname]file. For example, uwindfile = to set the initial zonal winds.

The EXF package has a built in method to handle different time intervals beween meteorological records and model time steps. The time interval between meteorological records in seconds is specified for each field with the namelist entry `[field]period` in `EXF_NML_02`, for any of the above fields. The time for the meteorological records to repeat themselves is set with the keyword `repeatPeriod` in `EXF_NML_01`.

```
# ********************
# External Forcing Data
# ********************
 &EXF_NML_01
#
 useExfCheckRange  = .TRUE.,
 useExfCheckRange  = .TRUE.,
 repeatPeriod      = 86400.0, # time until met. records repeat themselves
 exf_iprec         = 64,
 exf_yftype        = 'RL',
 &
# ********************
 &EXF_NML_02
 uwindstartdate1   = 20050401,
 uwindstartdate2   = 00000,
 uwindperiod       = 3600.0,  # time between consecutive entries in uwindfile

 vwindstartdate1   = 20050401,
 vwindstartdate2   = 00000,
 vwindperiod       = 3600.0

 atempstartdate1   = 20050401,
 atempstartdate2   = 000000,
 atempperiod       = 3600.0,

 aqhstartdate1     = 20050401,
 aqhstartdate2     = 000000,
 aqhperiod         = 3600.0,

 swdownstartdate1  = 20050401,
 swdownstartdate2  = 000000,
 swdownperiod      = 3600.0,

 lwdownstartdate1  = 20050401,
 lwdownstartdate2  = 000000,
 lwdownperiod      = 3600.0,

 uwindfile         ='x_wind.bin',
 vwindfile         ='y_wind.bin',
 atempfile         ='airtemp.bin',
 aqhfile           ='aqh.bin',
 swdownfile        ='swdown.bin',
 lwdownfile        ='lwdown.bin',
 &

 &EXF_NML_03
 &

 &EXF_NML_04
 &

 &EXF_NML_OBCS
 &
```

Listing 5: Sample `data.exf` file. This instance has meteorological records every 3600 s (1 h).

The EXF package can also handle the case when the meteorological records start from a different time than your model run. In this case the `startDate_1` and `startDate_2` fields become important. In `data.cal`, set

the model start time using these fields as follows:

- `startDate_1`: Format `YYYYMMDD` specifies the date to start
- `startDate_2`: Format `hhmmss` specifies the time to start

Then in `data.exf`, specify the start time of the meteorological records in the same way. You can set this globally, or for each individual field.

3.2 Core model parameters and recommendations

This section discusses choosing an equation of state, using checkpoint files, time-stepping, binary input/output, and other parameters for the core model.

3.2.1 Equation of state

The equation of state relates temperature, density, and salinity of water. For freshwater, the maximum density of is at 4 °C, so a linear equation of state is a poor choice with temperatures near 4 °C. The equation of state of Jackett and McDougall (1995, [?]) is a good compromise between simplicity (doesn't add an extra pickup file) and accuracy. Use this equation of state by including `eosType='JMD95Z',` in the `PARM01` namelist of `data`. See the discussion in the documentation about EOS type.

3.2.2 Checkpoint files

The model will inevitably fail sometimes. This can lose a lot of computation time especially when running large jobs on Graham. Therefore, it is best practice to save checkpoint files periodically. MITgcm has rolling checkpoint files and permanent checkpoint files. Rolling checkpoint files save the model current state with one of two file names ("ckptA" and "ckptB"), overwriting the oldest file to save new checkpoints. Permanent checkpoint files are saved with the model iteration numebr and therefore do not get overwritten. The frequency at which to save checkpoint files is controlled by the keywords `chkptFreq` for rolling checkpoints and `pchkptFreq` for permament checkpoints in the `PARM03` namelist of the `data` file.

3.2.3 Time stepping

The time step and simulation length parameters are all in the `PARM03` namelist of the `data` file. Specify the start and end time with the `startTime` and `endTime` keywords. The model time step is given by the `deltaT` keyword.

Alternatively, different time steps can be specified for the momenutm and tracer equations by setting `deltaTmom` and `deltaTtracer` individually. If not using the diagnostics package, the file output frequency should be specified as `dumpFreq` here too. If using the diagnostics package, set this to 0.

3.2.4 File input and output

Input and output files to the model are binary files in big-endian format. I usually use double-precision floats for input, by specifying `readBinaryPrec=64,` in `PARM01`. This corresponds to data type `">f8"` in python's numpy package. See listing 6 for a pythonic way to set input data for the model.

```
import numpy as np

T = np.zeros((100, 100, 50), dtype = '>f8', order = 'F')
for z in range(50):
  T[:, :, z] = 4 * (z / 50)

with open('initial_temperature', "wb") as fid:
  T.tofile(fid)
```

Listing 6: Python code for making input files to MITgcm

Model output is by default a 32-bit float data type. You shouldn't have to do much with the binary output data if you convert the data to netCDF. But sometimes it can be nice to check a field's values interactively as the model runs. Listing 7 has python code to open the output data.

```
import numpy as np
T = np.fromfile("T.0000001080.data", ">f4")
# ... do something with T
```

Listing 7: Python code for opening MITgcm binary output files

3.2.5 Gridding parameters

The size of the model grid must be specified at compile-time and at run-time. This includes setting the number of processors. Each compiled executable must always be run with the same number of grid points and processors, but can be run for different resolutions on the same grid.

Compile-time size options

The model grid first must be specified in file `SIZE.h`. See listing 8 for an example file. The overall grid is segmented into a few different subgrids. The full domain is split into tiles in X and Y. Then you specify the number of tiles per processor in X and Y (`nSx`, `nSy`), and the number of X, Y points per tile (`sNx`, `sNy`), and the number of processors in X and Y (`nPx`, `nPy`). Therefore,

you need to satisfy `Nx = sNx * nSx * nPx` and `Ny = sNy * nSy * nPy`. The amount each tile overlaps is given by `OLx` and `OLy`. The values you need here depend on which packages you use, and sometimes the advection schemes. The values in the example file are suitable for the SEAICE package with advection scheme 33.

```
C $Header: /u/gcmpack/MITgcm/verification/exp0/code/SIZE.h_mpi,v 1.1 2003/09/10
     02:25:46 edhill Exp $
C $Name:  $
C
C     *==========================================================*
C     | SIZE.h Declare size of underlying computational grid.
C     *==========================================================*
C     | The design here supports a three-dimensional model grid
C     | with indices I,J and K. The three-dimensional domain
C     | is comprised of nPx*nSx blocks (or tiles) of size sNx
C     | along the first (left-most index) axis, nPy*nSy blocks
C     | of size sNy along the second axis and one block of size
C     | Nr along the vertical (third) axis.
C     | Blocks/tiles have overlap regions of size OLx and OLy
C     | along the dimensions that are subdivided.
C     *==========================================================*
C     \ev
C
C     Voodoo numbers controlling data layout:
C     sNx :: Number of X points in tile.
C     sNy :: Number of Y points in tile.
C     OLx :: Tile overlap extent in X.
C     OLy :: Tile overlap extent in Y.
C     nSx :: Number of tiles per process in X.
C     nSy :: Number of tiles per process in Y.
C     nPx :: Number of processes to use in X.
C     nPy :: Number of processes to use in Y.
C     Nx  :: Number of points in X for the full domain.
C     Ny  :: Number of points in Y for the full domain.
C     Nr  :: Number of points in vertical direction.
      INTEGER sNy
      INTEGER OLx
      INTEGER OLy
      INTEGER nSx
      INTEGER nSy
      INTEGER nPx
      INTEGER nPy
      INTEGER Nx
      INTEGER Ny
      INTEGER Nr
      PARAMETER (
     &           sNx =   40,
     &           sNy =   40,
     &           OLx =   3,
     &           OLy =   3,
     &           nSx =   1,
     &           nSy =   1,
     &           nPx =   10,
     &           nPy =   10,
     &           Nx  =   400,
     &           Ny  =   400,
     &           Nr  =   50)

C     MAX_OLX  - Set to the maximum overlap region size of any array
C     MAX_OLY    that will be exchanged. Controls the sizing of exch
C                routine buufers.
      INTEGER MAX_OLX
      INTEGER MAX_OLY
      PARAMETER ( MAX_OLX = OLx,
     &            MAX_OLY = OLy )
```

Listing 8: Sample SIZE.h configuration file

Run-time options

The resolution is specified at run time in the `data` file, namelist `PARM04`. Depending on your preference for using r or z for your vertical coordinate, this namelist might look like listing 9. The grids are specified as `(number of points) * (distance between points)`. Alternatively, you can specify a list of distances. This might be convenient for the vertical coordinate to space grid points closer to the surface.

```
1 &PARM04
2  usingCartesianGrid=.TRUE.,
3  usingSphericalPolarGrid=.FALSE.,
4  delX=400*12.50,
5  delY=400*12.50,
6  delZ=50*0.2,
```

Listing 9: Sample grid namelist data

4 What didn't work

4.1 Advection schemes

MITgcm has many choices for advection schemes for various parts of the model. The SEAICE package is particularly sensitive to the choice of advection scheme. Choosing the wrong advection scheme can cause an unphysical amount of ice to form, cause ice to form above 0 C, or fail to move ice under wind or surface current forcing. The recommended schemes are 33 and 77, with all available schemes listed in table 2 and described in the documentation.

The SEAICE advection scheme is specified by including the line (for example) `SEAICEadvScheme = 33` in the `SEAICE_PARM01` namelist of `data.seaice`.

4.2 Surface winds

At some point we tried running the model with no wind forcing. I specified a binary file with all points identically 0, and input this into the model by including `uwindfile = 'const0.bin'` and `vwindfile = 'const0.bin'` in `EXF_NML_02` of `data.exf`. In this case I started with an ice covered lake, and was trying to melt the ice with downward solar radiation and warm air temperature. There was no heat transfer between the atmosphere and ice, so the ice never melted. My theory about why involves the possible configurations you can use the EXF package in. The package needs certain combinations of fields to be specified to compute the heat fluxes and other fields that it

Token	Advection Scheme
1	1st-order upwind
2	2nd-order centered difference
3	3rd-order upwind
4	3th-order centered difference
7	7th-order one step method with monotonicity preserving limiter
20	2nd-order direct space and time (Lax-Wendroff)
30	3rd-order direct space and time
33	3rd-order flux-limited direct space and time
40	Piecewise parabolic method with "null" limiter
41	Piecewise parabolic method with "mono" limiter
42	Piecewise parabolic method with "weno" limiter
50	Piecewise quartic method with "null" limiter
51	Piecewise quartic method with "mono" limiter
52	Piecewise quartic method with "weno" limiter
77	Non-linear flux limiter
80	2nd-order moment advection scheme (Prather, 1986)
81	2nd-order moment advection scheme, Prather Limiter

Table 2: Standard model parameters for case 1: Ice formation

couples to the core model. Of course this isn't said in the documentation, but the following table is included in the header file `EXF_OPTIONS.h`.

```
C     #  |TEMP  |DOWN  |BULK  |EVAP  |TURB  |             actions
C    ---|------|------|------|------|------|-------------------------------------
C    (1)|  -   |  -   |  -   |  -   |  -   | Read-in hflux, swflux and sflux.
C       |      |      |      |      |      |
C    (2)|  -   | def  |  -   |  -   |  -   | Read-in hflux, swdown and sflux.
C       |      |      |      |      |      | Compute swflux.
C       |      |      |      |      |      |
C    (3)| def  | def  | def  |  -   |  -   | Read-in atemp, aqh, swdown, lwdown,
C       |      |      |      |      |      |  precip, and runoff.
C       |      |      |      |      |      | Compute hflux, swflux and sflux.
C       |      |      |      |      |      |
C    (4)| def  |  -   | def  |  -   |  -   | Read-in atemp, aqh, swflux, lwflux,
C       |      |      |      |      |      |  precip, and runoff.
C       |      |      |      |      |      | Compute hflux and sflux.
C       |      |      |      |      |      |
C    (5)| def  | def  |  -   | def  | def  | Read-in hs, hl, swdown, lwdown,
C       |      |      |      |      |      |  evap, precip and runoff.
C       |      |      |      |      |      | Compute hflux, swflux and sflux.
C       |      |      |      |      |      |
C    (6)| def  |  -   |  -   | def  | def  | Read-in hs, hl, swflux, lwflux,
C       |      |      |      |      |      |  evap, precip and runoff.
C       |      |      |      |      |      | Compute  hflux and sflux.
C
C    ==========================================================================
```

Listing 10: Available configurations for running the EXF package

My theory is that have zero wind speed intereferes with one of these configurations, and the package can't compute all the required fields to couple

to the core model.

We got over this problem by specifying ε-winds. By specifying a very small wind speed $\mathcal{O}(10^{-16})$ $\mathrm{m\,s^{-1}}$, the model behaves properly and computes all heat fluxes.

5 Results: cases studied

5.1 Ice advection

This case was designed to verify:

- Ice will be moved by wind and surface currents
- Water will be cooled by a low air temperature
- Ice will melt in warm water and form in cold water

The setup for this experiment is specified in table 3. A square lake (6 km x 6 km x 10 m) at initially constant 2 °C, with air temperature −5 °C. There is $2.5\,\mathrm{m\,s^{-1}}$ wind in the positive x direction in the "South" half of the lake. Two small pieces of ice are initialized. One is directly in the wind, and the other is out of the wind. The chunk in the wind should be advected quickly by the wind, and the other should be slower since it needs to wait for surface currents to build up and push it.

Paramter Name	**Parameter Value**	**Comment**
`Nx`	60	15 x 4 processors
`Ny`	60	15 x 4 processors
`tempAdvScheme`	33	High order advection scheme
`eosType`	Linear	Linear EOS with incorrent slope: caused cold water at 0 °C to drop to the bottom, below warmer water. This EOS is not recommended.
`delX`	`60*100.0`	6 km lake, 100 m resolution
`delY`	`60*100.0`	6 km lake, 100 m resolution
`delZ`	`20*0.5`	10 m deep lake, 0.5 m resolution

`uwind`	$\begin{cases} 2.5 \text{ m/s} & y \leq 3\text{km} \\ 0 \text{ m/s} & y > 3\text{km} \end{cases}$	
`vwind`	0	
`atemp`	$-5°\text{C}$	Constant cold air temperature
`aqh`	$0.01\,\text{kg}\,\text{kg}^{-1}$	Specific humidity.
`swdown`	0	Minimum shortwave downward radiation
`lwdown`	$50\ \text{W}\ \text{m}^{-2}$	Minimum longwave downward radiation
`SEAICEuseDYNAMICS`	`.TRUE.`	Allow SEAICE dynamics
`SEAICE_area_reg`	`1.0E-5`	Ice fraction less than this will be treated as zero ice for next time step
`SEAICE_hice_reg`	`1.0E-5`	Height cutoff
`SEAICEadvHEFF`	`.TRUE.`	Advect effective height
`SEAICEadvAREA`	`.TRUE.`	Advect seaice area fraction
Temperature	$2\,°\text{C}$	Temperature stratified lake
Initial SEAICE area fraction	0.7	Two small chunks of ice
Initial SEAICE thickness	0.7 m	Two small chunks of ice

Table 3: Parameter values for case 1: ice advection

5.1.1 Results

This case was first run with the default ice advection scheme (`SEAICEadvScheme=2`), which gives unphysical results - ice forms when the surface temperature is above $0\,°\text{C}$ (figure 1). I ran the same case with advection schemes 7, 33, and 77 (figures 2, 3). Scheme 7 is also clearly unphysical. In this case the overlap extent in X and Y was not enough, causing the gridding pattern.

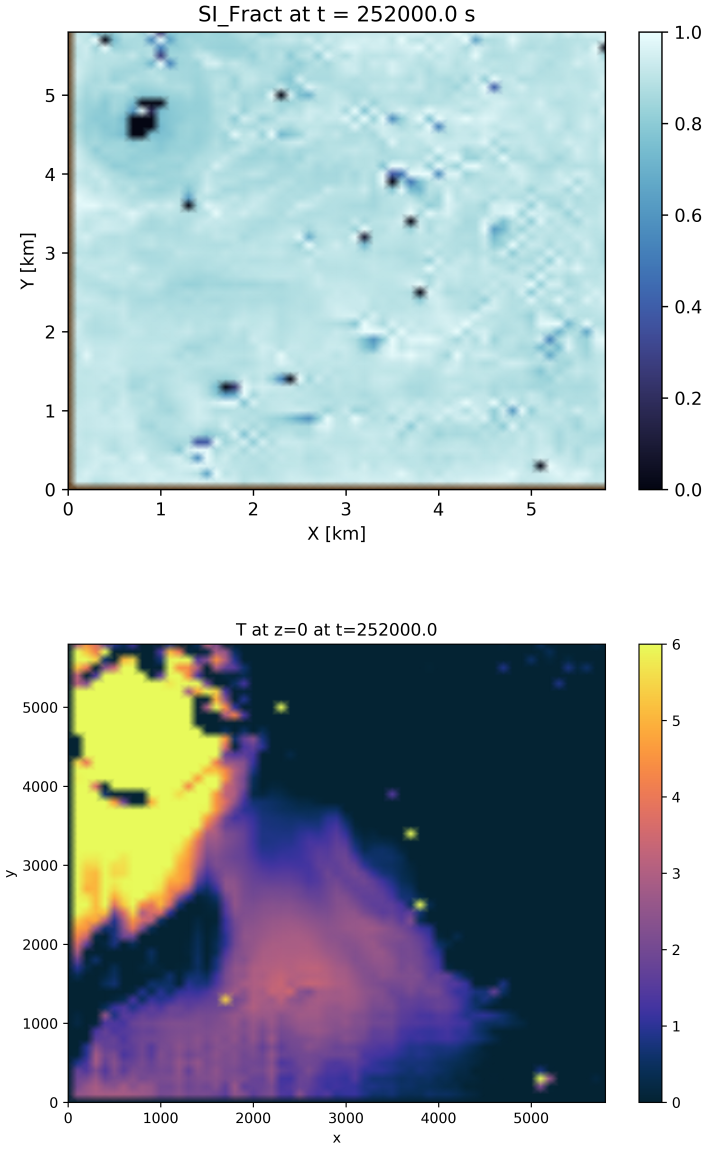

Figure 1: Ice area fraction and temperature fields for advection sceheme 2 (second order centered difference scheme). Note surface temperature much above freezing where ice has newly formed.

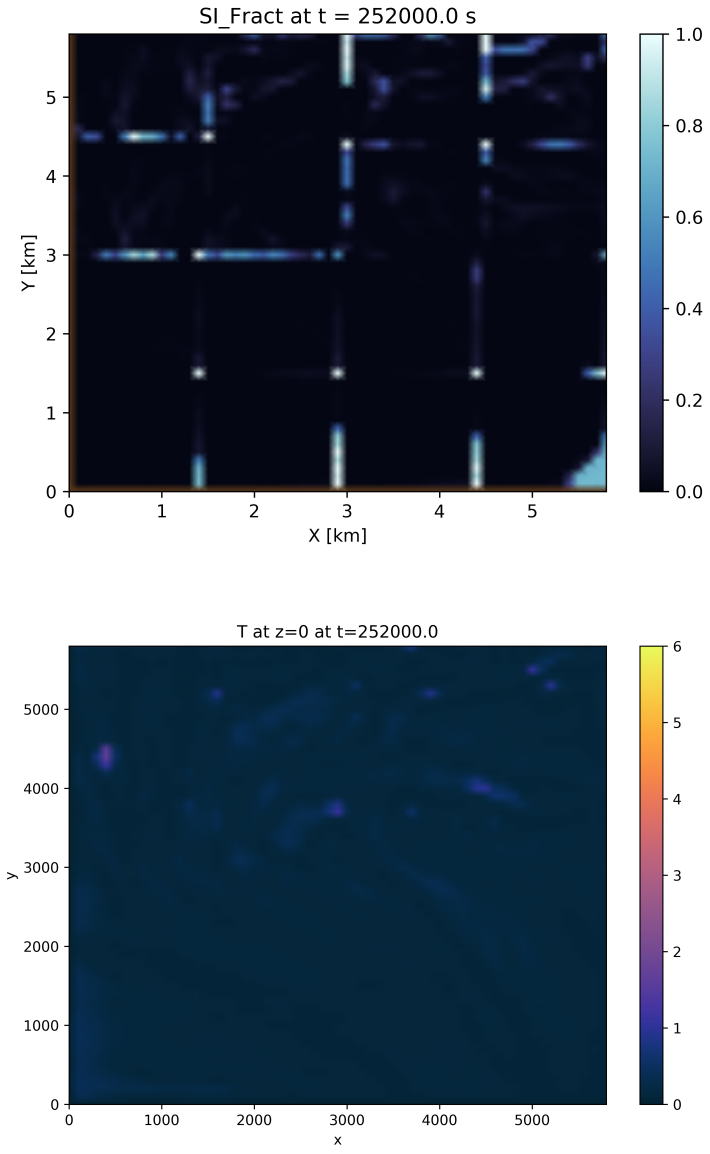

Figure 2: Ice area fraction and temperature fields for advection sceheme 7. The obvious gridding pattern comes from the wrong overlap extent in X and Y.

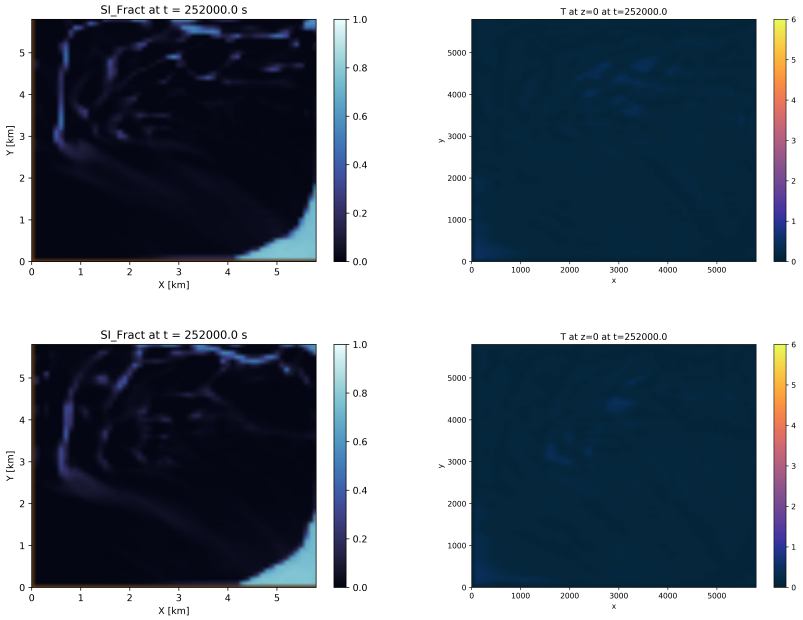

Figure 3: Ice area fraction and temperature fields for advection scheme 33 (top row) and 77 (bottom row). These agree qualitatively very well, and agree with physical intuition.

The difference between the top and bottom of each panel in figure 3 is due to the wind. Ice forms in the top half where the water is calmer (because there is no wind there), gets pushed by the surface currents, and then in the bottom half is pushed quickly by the wind. It collects in the bottom right corner.

5.1.2 What was learned

From this simulation, we learned and see the following points

- With a good advection scheme (33 or 77, high dimensional with flux limiter) the ice is advected physically by the wind and surface currents.
- The water was cooled by the cold air
- Ice melted at first because of the warm water. Ice forms later on when the surface temperature is at or below 0 (assuming we have a good advection scheme).

- The Linear equation of state is a poor choice. Looking at temperature cross-sections the cold water falls to the bottom, and water closer to 4 °C rises to the top. This is opposite from what should happen. A more accurate EOS would form ice faster since cold water would stay at the top.

5.2 Ice melting

We wanted to design a scenario which would isolate how the model melts ice. The lake is initialized with constant ice cover, with warm air and moderate downward solar radiation (see complete details in table 4. There are Eastward winds in the South half of the lake. We expect the ice to melt and break up. In particular, we want to see:

- Ice to melt until it starts to break up
- The ice to start being moved around once it has space to move (once there is some open water)
- Ice to completely melt, water to be heated by the radiation and air

Paramter Name	**Parameter Value**	**Comment**
`Nx`	600	50 x 12 processors
`Ny`	600	50 x 12 processors
`tempAdvScheme`	33	High order advection scheme
`nonHydrostatic`	`.FALSE.`	Use hydrostatic configuration
`eosType`	JMD95z	Polynomial approximation to true equation of state. This EOS respects the max in density at 4°C
`delX`	`600*10.0`	6 km lake, 100 m resolution
`delY`	`600*10.0`	6 km lake, 100 m resolution
`delZ`	`100*0.1`	10 m deep lake, 0.5 m resolution
`uwind`	$\begin{cases} 1.0 \text{ m/s} & y \leq 3\text{km} \\ 0 \text{ m/s} & y > 3\text{km} \end{cases}$	
`vwind`	0	

`atemp`	10°C	Constant cold air temperature
`aqh`	0.01	Approximately 50% constant humidity
`swdown`	300 W m^{-2}	Minimum shortwave downward radiation
`lwdown`	300 W m^{-2}	Minimum longwave downward radiation
`SEAICEuseDYNAMICS`	`.TRUE.`	Allow SEAICE dynamics
`SEAICE_area_reg`	`1.0E-5`	Ice fraction less than this will be treated as zero ice for next time step
`SEAICE_hice_reg`	`1.0E-5`	Height cutoff
`SEAICEadvHEFF`	`.TRUE.`	Advect effective height
`SEAICEadvAREA`	`.TRUE.`	Advect seaice area fraction
`SEAICEadvScheme`	33	Advection scheme 33: Flux limited direct space and time advection scheme
Temperature	$\begin{cases} 0°C & z = 0 \text{ m} \\ 4°C & z = -10 \text{ m} \end{cases}$	Temperature stratified lake
Initial SEAICE area fraction	0.9	Constant ice covered fraction
Initial SEAICE thickness	0.1	Constant ice thickness

Table 4: Parameter values for case 2: ice melt

The model was run with these parameters, as well as:

- Change `nonHydrostatic` to `.TRUE.` to test sensitivity to hydrostatic/non-hydrostatic configuration
- Double the number of grid points in X and Y to test resolution sensitivity
- Set surface winds to 0 to test effect of wind

5.2.1 Results

The first results are for the case outlined in table 4. This is hydrostatic with 12.5 m horizontal resolution and 0.5 $\mathrm{m\,s^{-1}}$ winds in the South half. See pseudocolour plots of the ice thickness and temperature in figure 5, and the ice mass time-series in figure 4. Psuedocolour plots aren't presented for other cases with wind, since they are qualitatively similar.

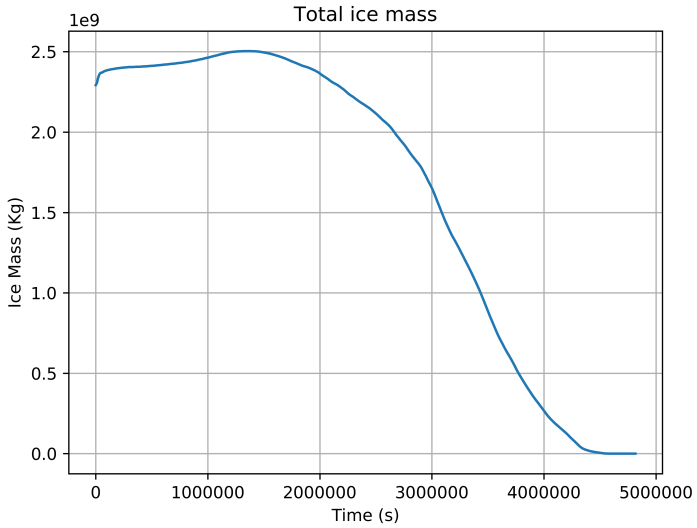

Figure 4: Total ice mass with parameters from table 4. The ice initially grows in the cold water, and later is melted by the warm air temperature and solar radiation.

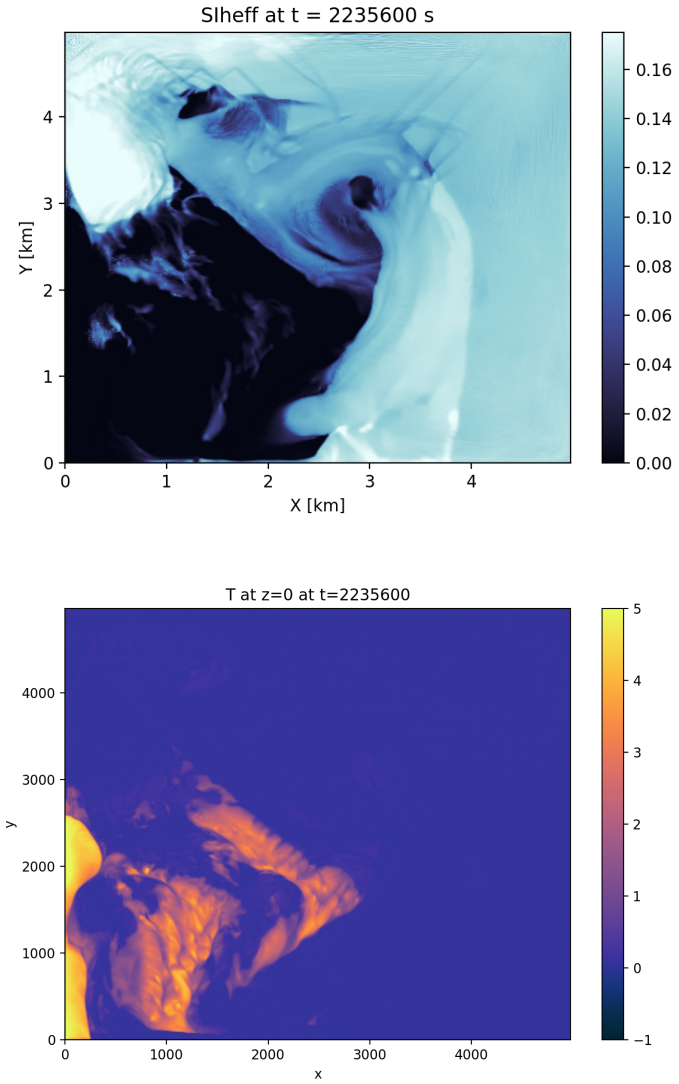

Figure 5: Effective ice thickness (top) and surface temperature (bottom) with parameters as in table 4. The ice has melted in the bottom left corner from the wind. Notice the ridge built up to the right of the open water, and the cracking in the ice above the open water.

Parameter comparisons

We want to see how sensitive the results are to changes in some parameters. We compare

- Hydrostatic and non-hydrostatic model configurations
- Higher resolution
- Removing surface winds

The model runs much faster in hydrostatic configuration than non-hydrostatic, and eventually we will need to use the hydrostatic configuration to run a full lake-scale model. Therefore, we'd like to know how much of a difference this approximation makes for our cases. Similarly, we want to understand what impact resolution will have in our simulations. Therefore, we run 4 cases: high and low resolution for both hydrostatic and non-hydrostatic configurations. We start with 12.5 m horizontal resolution, and increase to 6.25 m resolution. Total kinetic energy is compared in figure 6. The case with no surface wind is not shown since the kinetic energy is significantly lower.

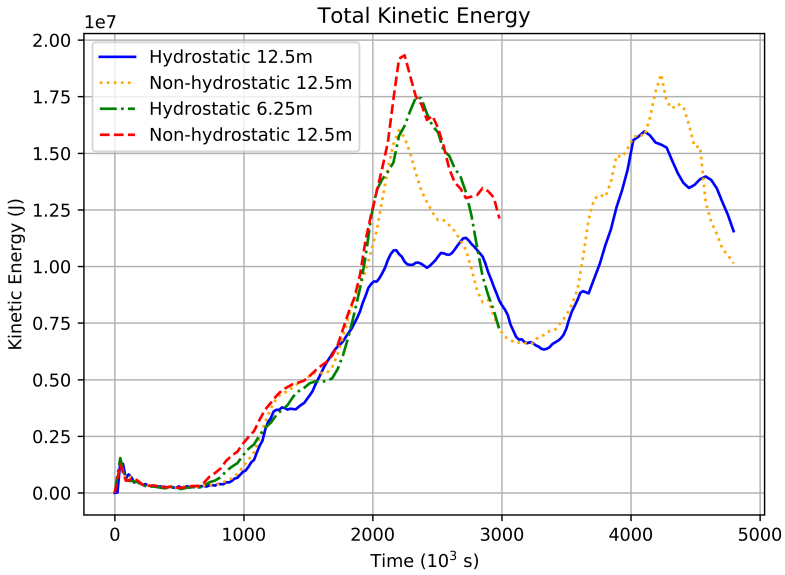

Figure 6: Comparison of total kinetic energy for different model configurations.

The high-resolution simulations have good agreement between hydrostatic and non-hydrostatic configurations, and agree well with the lower resolution non-hydrostatic configuration. The low resolution hydrostatic configuration underestimates the first maximum of kinetic energy compared to the other simulations.

We can compare the mean ice thickness and total internal energy in the water between model configurations. We compare the above simulations, as well as the simulation with no surface winds. Intuitively removing the winds should change the rate of heat transfer and ice break-up, so we investigate this. The results are in figure 7.

In each case, the ice melts most rapidly at about 5 weeks in to the simulation. Call this the break-up time. All simulations have relatively good agreement on ice thickness until the break-up time. The case with no wind takes longer to start melting, and then melts faster, than the other cases. At break-up time the case with no wind seems to be able to transfer heat from the water into the ice to melt it more efficiently than the other cases. This might explain the dip in internal energy compared to the other cases.

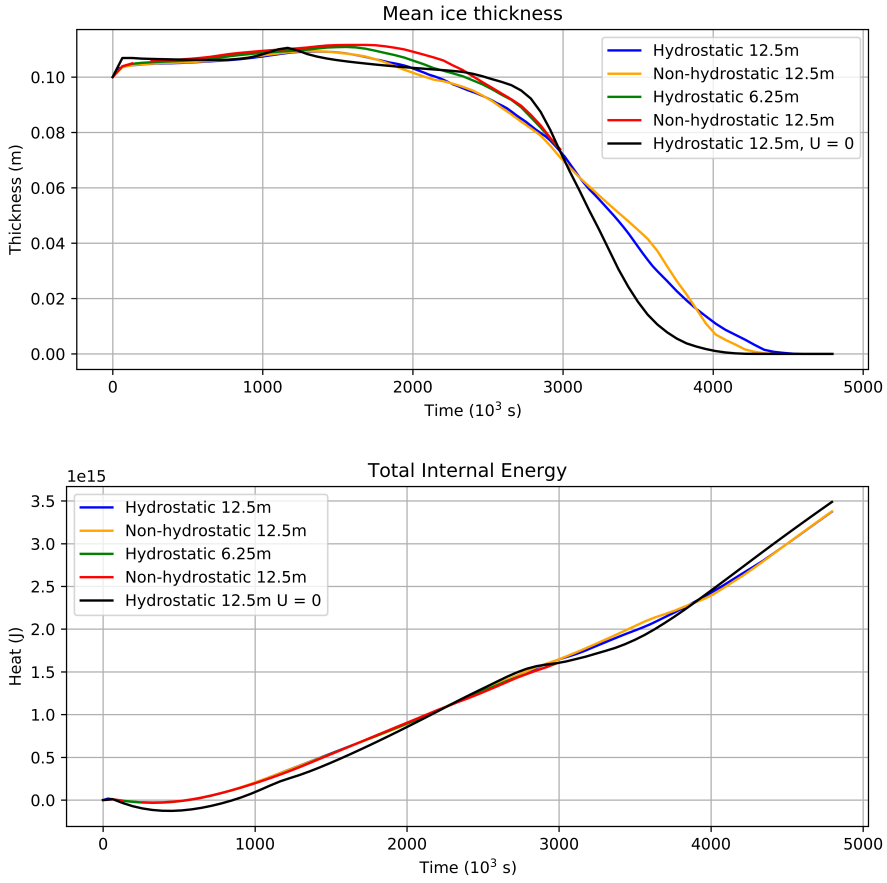

Figure 7: Comparison of mean ice thickness (top) and total internal energy of the water (bottom) between model configurations.

5.2.2 What was learned

From running the main case, we can say the model melts ice reasonably physically. The ice melts until it gets thin enough to break up, at which point it cracks and starts being pushed around by the wind and surface currents. The ice continues to move and melt, eventually melting entirely. The remaining water is heated by the downward radiation and warm air temperature.

The total kinetic energy varies significantly between different parameter sets. In particular, the kinetic energy is significantly different between

hydrostatic and non-hydrostatic configurations at lower resolutions. This could be a problem for lake scale simulations, where the resolution would be relatively large. However, we expect that for large resolutions the difference between hydrostatic and non-hydrostatic configurations should be small. Nevertheless, even with the large difference in kinetic energy, the total internal energy and average ice thickness are relatively insensitive to changing the hydrostatic flag. They are more sensitive to the inclusion of surface winds, which may be expected.

Regarding running the model, I had to include a very small wind speed instead of truly 0 winds. This is discussed in more detail in section 4.2. These cases were also run without all the recommended modifications to the default parameters given in listing 4. These results should be checked with those improved parameters and flags.

5.3 Solar forcing

We are interested in how the model simulates solar forcing over open water and ice covered water. This is clearly important to lake dynamics, and likely needs to be studied in more detail than I have done so far.

To study the different solar forcing for ice-covered and open water, I initialize a lake to be completely covered in 1 m thick ice in the North half, and open water in the South half. This is an idealized interface between an ice sheet and open water in a real lake. I set the air temperature to be 0 °C, and send down moderate solar radiation. I continue to use Eastward winds in only the South half of the lake. See more details on the setup in table 5.

From this case, I want to see

- More warming in the open water than under ice covered water, since the ice should reflect most of the incoming radiation and absorb heat from the water in melting
- A layer of cold water staying under the ice since the water at 0 °C is less dense than sun-heated water closer to 4 °C

Paramter Name	**Parameter Value**	**Comment**
`Nx`	600	50 x 12 processors
`Ny`	600	50 x 12 processors
`tempAdvScheme`	33	High order advection scheme
`nonHydrostatic`	`.FALSE.`	Hydrostatic model configuration

`eosType`	JMD95z	Polynomial approximation to true equation of state. This EOS respects the max in density at 4°C
`delX`	`600*10.0`	6 km lake, 100 m resolution
`delY`	`600*10.0`	6 km lake, 100 m resolution
`delZ`	`100*0.1`	10 m deep lake, 0.5 m resolution
`uwind`	$\begin{cases} 0.5 \text{ m/s} & y \leq 3\text{km} \\ 0 \text{ m/s} & y > 3\text{km} \end{cases}$	
`vwind`	0	
`atemp`	0°C	Constant cold air temperature
`aqh`	0.01	Approximately 50% constant humidity
`swdown`	300 W m^{-2}	Minimum shortwave downward radiation
`lwdown`	300 W m^{-2}	Minimum longwave downward radiation
`SEAICEuseDYNAMICS`	`.TRUE.`	Allow SEAICE dynamics
`SEAICE_area_reg`	`1.0E-5`	Ice fraction less than this will be treated as zero ice for next time step
`SEAICE_hice_reg`	`1.0E-5`	Height cutoff
`SEAICEadvHEFF`	`.TRUE.`	Advect effective height
`SEAICEadvAREA`	`.TRUE.`	Advect seaice area fraction
`SEAICEadvScheme`	33	Advection scheme 33: Flux limited direct space and time advection scheme
Temperature	$\begin{cases} 0°\text{C} & z = 0 \text{ m} \\ 4°\text{C} & z = -10 \text{ m} \end{cases}$	Temperature stratified lake
Initial SEAICE area fraction	$1.0, y \geq 3$ km	Constant ice in North half

Initial SEAICE thickness	$1.0, y \geq 3$ km	Constant ice in North half

Table 5: Parameter values for case 3: solar forcing

This case was run in hydrostatic and non-hydrostatic configuration.

5.3.1 Results

A plot of the surface temperature is shown in figure 8. Figure 9 shows constant temperature contours. Note the warm water pushing the 0 °C contour up under the ice in the East end of the lake. The ripples in the contours are cased by velocity shear. Since the wind has a smooth transition (hyperbolic tangent function), the wind at the ice edge is half the magnitude of the maximum wind in the South half. The surface currents are then in the negative X direction against the ice and positive X in the bulk of the open water, creating shear.

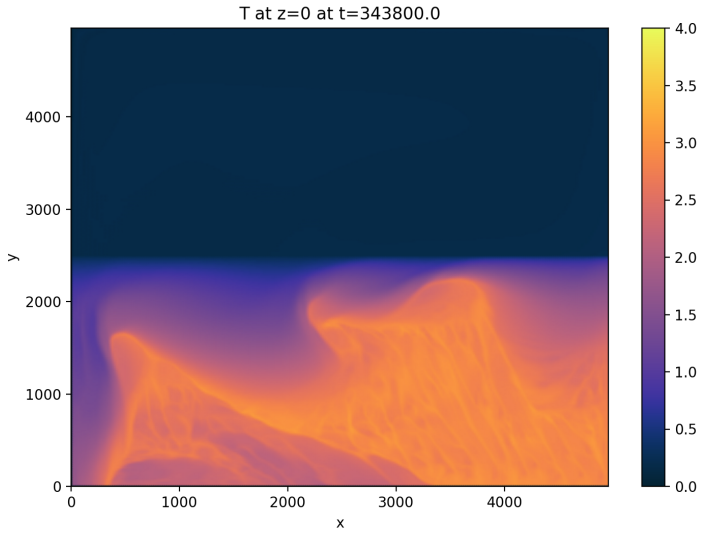

Figure 8: Surface temperature at the end of the model run. The surface layer under the ice remains at exactly 0 °C, while the open water is warmed by the sun.

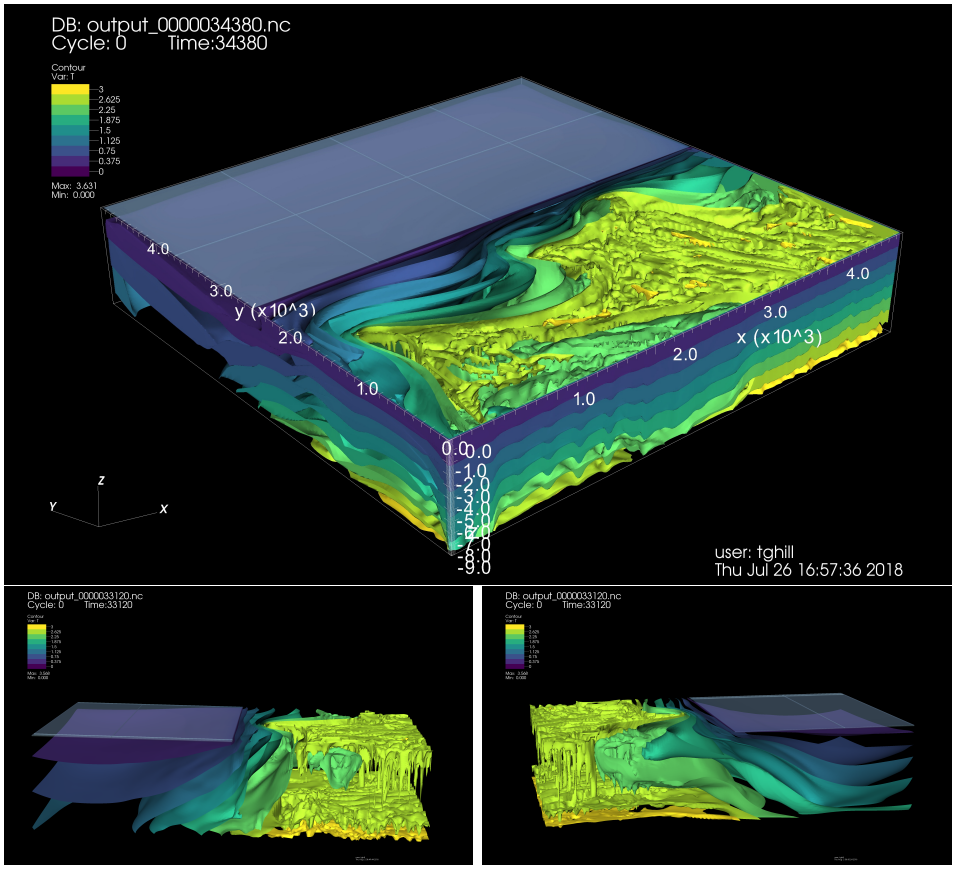

Figure 9: Constant temperature surfaces for hydrostatic configuration. The top panel shows the entire domain. The bottom panels show two perspectives of a cut of the Eastern half of the domain. The forward face in the bottom left is the center of the domain. The forward face in the bottom right is the Eastern edge of the domain.

This case was also run in non-hydrostatic configuration. A comparison of the kinetic energy, internal energy, and ice mass are shown in figure **??**. The kinetic energies agree very well, and the internal energy and ice mass are even closer. This might not be the case for higher resolutions, but this was not studied. This case has a horizontal resolution of 20 m and a vertical resolution of 20 cm. If the resolution is increased to closer to 5 m as in the ice melting high resolution cases the curves might diverge.

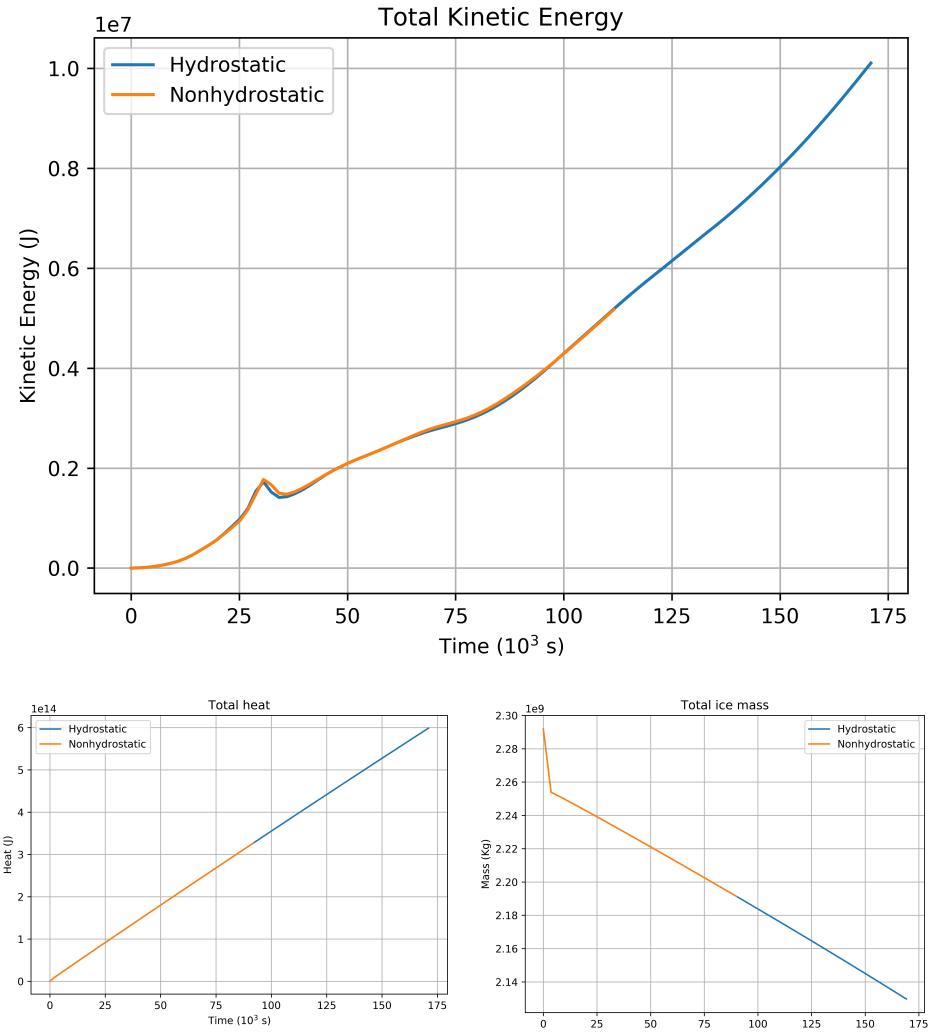

Figure 10: Comparison of total kinetic energy (top), internal energy in the water (bottom left) and ice mass (bottom right) for hydrostatic and non-hydrostatic configurations.

5.3.2 What was learned

From this case, we see that the treatment of solar radiation is consistent with our intuition. At the resolution used, the difference between hydrostatic and non-hydrostatic configurations is very small. This was also a good test case

for 3D visualization techniques (figure 9).

It might be worthwhile to run this case at higher resolution on Graham to see if at higher resolutions the non-hydrostatic configuration has higher kinetic energy from convection than the hydrostatic case.

5.4 Periodic forcing

So far, all cases have used a similar domain, and have had constant forcing. We try running the model with a much larger domain and daily cycles in the forcing fields.

TALK ABOUT DOMAIN AND FORCING.

Meteorological forcing

We use a sinusoidal daily cycle in as a very rough approximation to the daily cycle of the forcing (figure 11).

5.4.1 Results

We expect a roughly sinusoidal signal in the output fields in response to the input forcing (fig 11). The mean value of the output fields is shown in figure 12. Notice the strange shape of the wind stress curve, and that the peak positive U value occurs before midday. We would expect the peak positive surface currents to occur at or slightly after the peak positive wind speed, but the currents appear to lead the wind.

5.4.2 Run with constant temperature

We want to see what happens when we run this case with no temperature stratification. The response curves above have the strange "capped" structure, and we want to know if this goes away or stays when we take out the stratification. The capped structure could be due to the wind creating an internal seiche, and removing the stratification would remove the seiche.

5.5 1D comparison

Show results to CLIMo, talk about what we've changed.

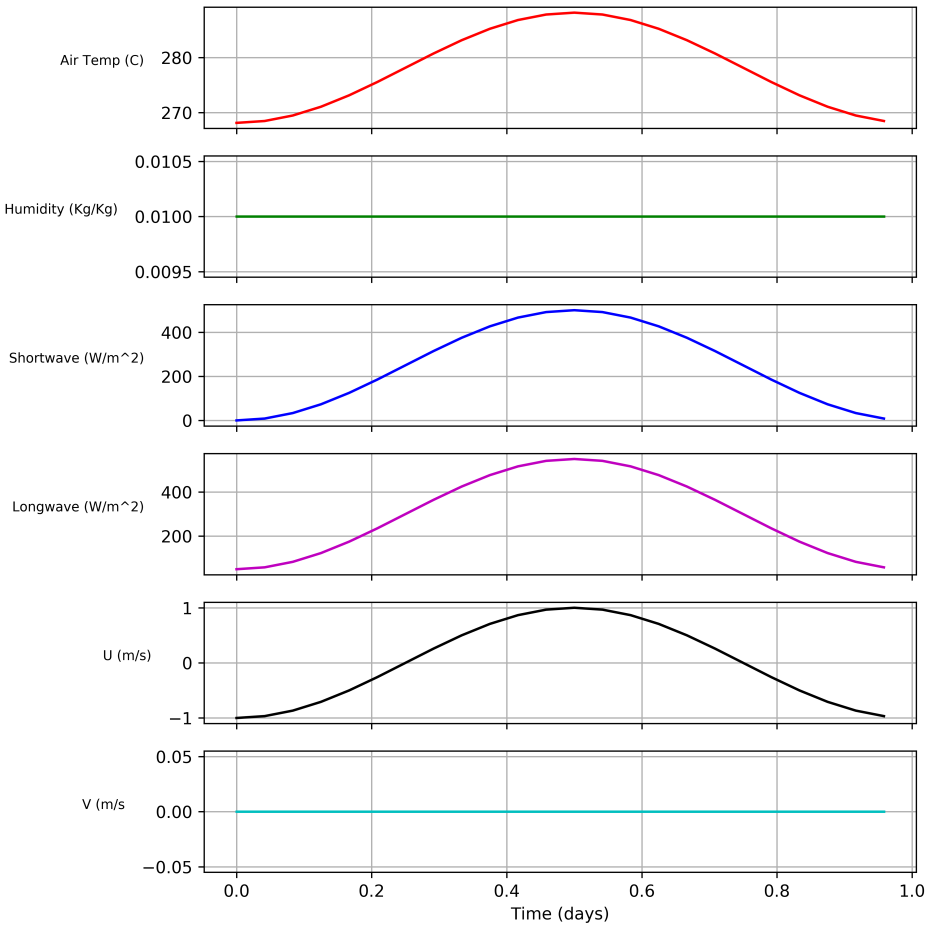

Figure 11: Time-dependent forcing values. Fields are sampled every hour (same frequency as output files are generated).

6 Recurring issues

6.1 NetCDF/HDF5 libraries

I have periodically had problems with the NetCDF/HDF5 libraries. These problems usually came from running out of disk space or disk quotas, but sometimes show up when the wrong modules are loaded. When processing the model binary output files to netCDF files, sometimes the process fails with a generic netCDF or HDF library error. This may be caused by one of

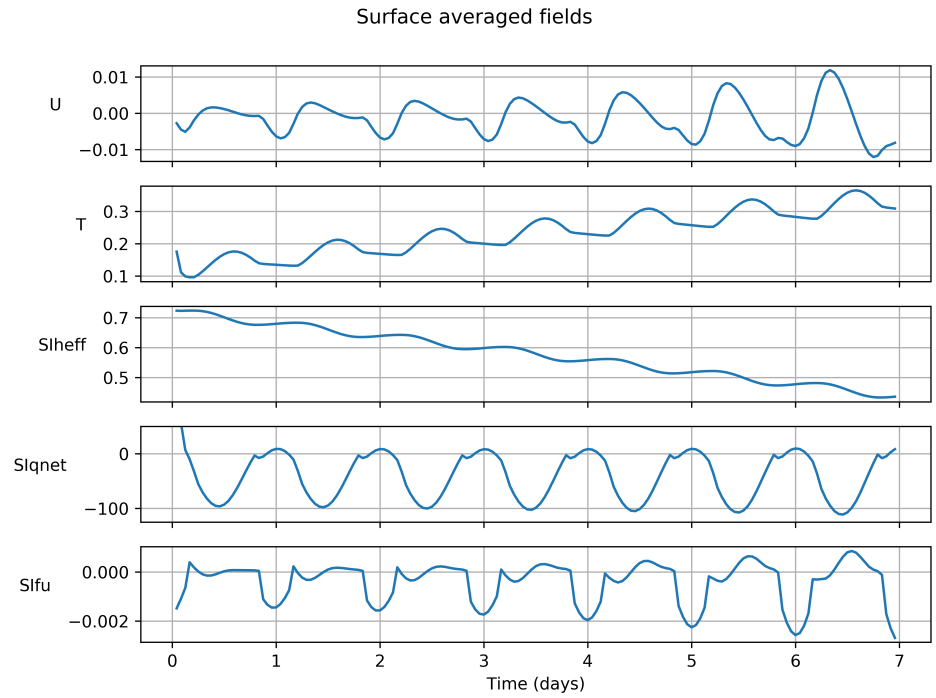

Figure 12: Mean value of output fields in response to periodic forcing. From top to bottom: mean surface U currents, mean surface temperature, mean effective ice thickness, mean heat transfer from the lake into the ice, mean surface wind stress (U direction)

the two issues below.

6.1.1 Disk space or quota

The first time this issue came up, I had run out of my disk quota on Graham. Try running the bash command `ls -l`. You want to see something like `drwxr-s--- 3 tghill ctb-mmstastn  4096 Jul 24 14:08 include` for each line. Of the two names, the second should be `ctb-mmstastn`, because then your file storage will count towards the contributed resource space, instead of your own (or elsewhere). If a file shows up with group ownership that is your username (or another supervisor who has smaller storage space), you can change the ownership by running `chown [username]:ctb-mmstastn` (or a different group name).

If you're running MITgcm a lot on Hood, ask Robyn for access to a larger

storage drive to store your data on. The home disk is relatively small.

6.1.2 Modules

The loaded modules on Graham can also present problems. I haven't been able to determine which modules don't work together when running or compiling the model, but to install and use NetCDF tools, the NetCDF and HDF modules should be loaded. You can seach for the current versions with the command `module spider netcdf`, `module spider hdf5`. If you're compiling and running in paralle, you might need to load the mpi versions. The list of modules below has worked for me for compiling, running, and converting output to netCDF.

- StdEnv/2016.4
- nixpkgs/16.09
- imkl/11.3.4.258
- intel/2016.4
- openmpi/2.1.1
- hdf5/1.8.18
- netcdf-mpi/4.1.3

6.2 MITgcm python utils

A python `FutureWarning` comes up when running the netCDF conversion package from `gcmpy` with python 3. This might become an Exception in python 3.7. This is caused by a problem in the MITgcmutils package that ships with MITgcm.

The problem line was fixed in commit 36b33d0d, see this pull request. Any model code before this change should be updated to the most recent version with this issue fixed.

7 Physics of the model

The model uses physical parameterizations that are usually documented to some extent in the MITgcm documentation. As part of getting the model up and running, I looked into some of the parameterizations and compared to what we expect.

7.1 Radiative cooling

We looked into the radiative cooling of ice when starting to compare MITgcm ice growth rates to the 1-D model CLIMo (Canadian Lake Ice Model).

The radiative cooling should follow the Stefan-Boltzmann law for total radiative power M

$$M = \varepsilon \sigma T^4 \tag{1}$$

Where M is the radiant emittance ($\mathrm{W\,m^{-2}}$), ε is the emissivity, and $\sigma = 5.670\,373 \times 10^{-8}\,\mathrm{W\,m^{-2}\,K^{-4}}$ is the Stefan-Boltzmann constant.

The peak wavelength of the spectral distribution should follow Wien's law

$$\lambda_{max} = \frac{b}{T} \tag{2}$$

Where $b = 2.898 \times 10^{-3}\,\mathrm{m\,K}$ is Wien's displacement constant and T is the temperature. For 0 °C, $\lambda_{max} \approx 10^{-5}\,\mathrm{m}$ is deep in the infrared (longwave). Therefore, all the radiant emittance should be captured as upward longwave radiation.

The emittance will depend on the value of emissivity used. The SEAICE packages takes in ice emissivity with the variable `SEAICE_emissivity` which is actually $\sigma\varepsilon$. The default value is $\sigma\varepsilon = 5.50 \times 10^{-8}\,\mathrm{W\,m^{-2}\,K^{-4}}$. Computing the expected emittance,

$$M = 306.17\,\mathrm{W\,m^{-2}}$$

The model shows very good agreement with this value (fig. **??**)

7.2 Water surface energy budget

As one way to compare the effect of parameters and meteorological forcing values, we compute the total internal energy in heat of the water, as well as the heat fluxes across the water boundary. We expect that

$$\frac{dQ}{dt} = \Phi_{net} = \Phi_{ice} + \Phi_{atm} \tag{3}$$

The model outputs $-\Phi_{net}$ as the diagnostic `SIqnet` (heat flux from the water into ice if ice covered, atmosphere if open water). To check this, we compute the internal energy of the water as

$$Q = c_w \sum_{x,y,z} \rho \Delta V \Delta T \tag{4}$$

Where the sum is over all grid cells, ΔV is the volume of each cell, and ΔT is the temperature at time t minus the initial temperature.

The derivative is taken using the most basic scheme,

$$\frac{dQ}{dt} \approx \frac{Q(t_{n+1}) - Q(t_n)}{t_{n+1} - t_n}$$

Finally, a plot of $\frac{dQ}{dt} + \texttt{SIqnet}$ should be near zero (not identically zero because of the derivative scheme) for all times. We have found this to be the case.

7.3 Surface albedo

In comparing the MITgcm ice growth rate to CLIMo, I investigated the albedo parameterization of MITgcm. The model classifies ice according to the temperature of the surface water T_{surf} comapred to the freezing/melting temperature T_{melt}

$$\text{ice} = \begin{cases} \text{dry}, & T_{surf} < T_{melt} \\ \text{wet}, & T_{surf} \geq T_{melt} \end{cases}$$

The default ice albedo is

$$\alpha_i = \begin{cases} 0.75, T_{surf} < T_{melt} \\ 0.66, T_{surf} \geq T_{melt} \end{cases}$$

The default snow albedo is

$$\alpha_s = \begin{cases} 0.84, T_{surf} < T_{melt} \\ 0.70, T_{surf} \geq T_{melt} \end{cases}$$

And the values are controlled by the input variables `SEAICE_dryIceAlb`, `SEAICE_wetIceAlb`, `SEAICE_drySnowAlb`, and `SEAICE_wetSnowAlb`.

7.4 Thermodynamics

The SEAICE package is mainly intended for ice dynamics, but includes basic thermodynamics. The THSICE package is be fully compatible with SEAICE and include improved thermodynamics. However, the THSICE package is not well suited for freshwater simulations, so for the purpose of this document it is recommended to continue to use only the SEAICE package for dynamics and thermodynamics.

The THSICE package is intended to improve the thermodynamics of saltwater ice. When ice forms over salt water, some pockets of salt water called brine pockets get trapped in the ice. These brine pockets provide interesting thermodynamics. When the ice is warmed, some of the ice around the brine pockets will be melted. This meltwater must be freshwater, so it dilutes the salt in the brine pocket. This can sufficiently dilute the brine pocket so that the pocket freezes completely, releasing latent heat to the surrounding ice [**?**].

Clearly this process is irrelevant for freshwater lake ice. While the three-layer formulation of [**?**] is an improvement over what is described in the SEAICE documentation, the implementation does not work for freshwater. The model tends to NaN once ice starts to form. Moreover, the SEAICE documentation says "NOTE: THIS SECTION IS TERRIBLY OUT OF DATE" about the thermodynamics description in the documentation. Therefore, we can't really compare the packages anyways.

7.5 Total kinetic energy

I compute the total kinetic energy of the water as an additional way to compare the differences between different parameters and configurations. The total kinetic energy is computed as

$$\text{TKE} = \frac{1}{2} \sum_{x,y,z} \rho \left(u^2 + v^2 + w^2 \right) \Delta V \tag{5}$$

Where $\rho = \rho_0 + \rho'$ is the total (reference + anomaly) density of the gri cell, u, v, w are the x, y, z velocities, and ΔV is the volume of the grid cell.

7.6 Dynamical ice model

The equation governing the dynamical evolution of the ice is the conservation of momentum equation

$$m \frac{D\mathbf{u}}{Dt} = -mf\mathbf{k} \times \mathbf{u} - m\nabla\phi(0) + \mathbf{F} + \tau_{\mathbf{air}} + \tau_{\mathbf{ocean}} \tag{6}$$

Where $m = m_i + m_s$ is the total mass of ice and snow per unit area, $\phi(0)$ is the sea surface height potential, $\mathbf{F} = \nabla \cdot \sigma$ is the divergence of the ice stress tensor, and τ_{air}, τ_{ocean} are the wind-ice and ocean-ice stresses. The wind and ocean stresses are parameterized as

$$\begin{aligned} \tau_{\mathbf{air}} &= \rho_{air} C_{air} \left\| \mathbf{U}_{air} - \mathbf{u} \right\| R_{air} \left(\mathbf{U}_{air} - \mathbf{u} \right) \\ \tau_{\mathbf{water}} &= \rho_{water} C_{water} \left\| \mathbf{U}_{water} - \mathbf{u} \right\| R_{water} \left(\mathbf{U}_{water} - \mathbf{u} \right) \end{aligned} \tag{7}$$

Where $\mathbf{U_i}$ are the surface winds of the atmosphere and water, C_i are air and water drag coefficients, ρ_i are reference densities, and R_i are rotation matrices.

7.7 Radiation

The model can be forced with net or downward shortwave and longwave radiation. Longwave radiation is absorbed by the surface layer, while shortwave radiation penetrates into the interior of the water. This section describes the penetrating shortwave radiation. The equations here come from [**?**] and the source code, particularly `pkg/exf/exf_radiation.F`, `model/src/swfrac.F` and `model/src/external_forcing.F`.

First, consider the amount of shortwave radiation that is transmitted through the surface. The reflected shortwave radiation is the product of the surface albedo and the incoming shortwave radiation, αI_{down}. Then, the amount absorbed into the water is

$$I_0 = (1 - \alpha) I_{down} \tag{8}$$

The radiation transmission model [**?**] splits the radiation into two wavelength bands. Each decays exponentially

$$\frac{I}{I_0} = Re^{-z/\zeta_1} + (1 - R)e^{-z/\zeta_2} \tag{9}$$

Where I is the radiation at depth z (positive from the surface), and $R = 0.62$, $\zeta_1 = 0.60\,\mathrm{m}$, $\zeta_2 = 20\,\mathrm{m}$ are constants that come from Jerlov water type IA. This water type represents clear ocean water, and is hard-coded into MITgcm.

This treatment should work well for relatively deep water, but for shallow lakes this is a problem. For example, with a 2 m deep lake approximately 37% of the radiation makes it to the bottom of the lake. This radiation should then be absorbed by the bottom and eventually heat the lake from the bottom, but this is not accounted for.

7.8 Humidity

The EXF package uses specific humidity, which is the ratio of the mass of water vapour to the total mass of a unit volume of air. This is very different from relative humidity, which we are more familiar with from everyday weather. [Include how to convert between them]

8 Visualization tools

Talk about VisIt scripting, or point to wiki? Figure this out!

9 Conclusions

Talk about where I think the model is, and what went well and not so well.

10 Next steps

Talk about what to do next.

10.1 Comparison to 1D model

Talk about what to change wit h CLIMo comparison.