
PISM, a Parallel Ice Sheet Model

version 1.0

the PISM authors

Oct 20, 2017

CONTENTS

1	Acknowledgements	3
2	Installing PISM	5
2.1	Required tool and libraries	5
2.2	Installation Cookbook	6
2.2.1	Installing prerequisites from Debian packages	6
2.2.2	Installing required libraries on macOS	7
2.2.3	Installing prerequisites from sources	7
2.2.4	Building PETSc	8
2.2.5	Building PISM	9
2.2.6	Common build problems and solutions	10
2.3	Quick tests of the installation	11
2.4	Next steps	12
2.5	Rebuilding PISM documentation	12
2.5.1	Manual	12
2.5.2	Source Code Browser	13
2.5.3	Documentation for PISM's Python bindings and inversion tools	13
2.5.4	Re-building documentation without PISM's prerequisites	13
3	PISM User's Manual	15
3.1	Getting started: a Greenland ice sheet example	15
3.1.1	Input data	16
3.1.2	First run	17
3.1.3	Watching the first run	18
3.1.4	Second run: a better ice-dynamics model	20
3.1.5	Third run: higher resolution	22
3.1.6	Fourth run: paleo-climate model spin-up	23
3.1.7	Grid sequencing	25
3.1.8	An ice dynamics parameter study	29
3.2	Ice dynamics, the PISM view	33
3.2.1	Two stress balance models: SIA and SSA	33
3.2.2	A hierarchy of simplifying assumptions for grounded ice flow	34
3.2.3	Evolutionary versus diagnostic modeling	35
3.2.4	Climate inputs, and their interface with ice dynamics	36
3.3	Initialization and bootstrapping	39
3.3.1	Initialization from a saved model state	39
3.3.2	Bootstrapping	40
3.4	Modeling choices	41
3.4.1	Model domain, grid, and time	42
3.4.2	Ice dynamics and thermodynamics	50

3.4.3	The subglacier	57
3.4.4	Marine ice sheet modeling	65
3.4.5	Disabling sub-models	71
3.4.6	Dealing with more difficult modeling choices	71
3.5	Practical usage	72
3.5.1	Handling NetCDF files	72
3.5.2	Input and output	72
3.5.3	Saving time series of scalar diagnostic quantities	74
3.5.4	Saving time series of spatially-varying diagnostic quantities	76
3.5.5	Saving re-startable snapshots of the model state	77
3.5.6	Run-time diagnostic viewers	78
3.5.7	PISM's configuration parameters and how to change them	79
3.5.8	Regridding	80
3.5.9	Signals, to control a running PISM model	82
3.5.10	Understanding adaptive time-stepping	83
3.5.11	Balancing the books	84
3.5.12	PETSc options for PISM users	86
3.5.13	Utility and test scripts	87
3.5.14	Using PISM for flow-line modeling	87
3.5.15	Managing source code modifications	88
3.6	Simplified geometry experiments	89
3.6.1	EISMINT II	90
3.6.2	MISMIP	91
3.6.3	MISMIP3d	95
3.7	Verification	97
3.7.1	Sample convergence plots	99
3.8	Validation case studies	102
3.8.1	An SIA flow model for a table-top laboratory experiment	102
3.8.2	An SSA flow model for the Ross Ice Shelf in Antarctica	104
3.9	Example: A regional model of the Jakobshavn outlet glacier in Greenland	107
3.9.1	Get the drainage basin delineation tool	108
3.9.2	Preprocess the data and get the whole ice sheet model file	108
3.9.3	Identify the drainage basin for the modeled outlet glacier	109
3.9.4	Cut out the computational domain for the regional model	110
3.9.5	Quick start	110
3.9.6	Spinning-up the regional model on a 5 km grid	110
3.9.7	Century run on a 2 km grid	112
3.9.8	Plotting results	112
3.10	Configuration parameters	113
3.11	Diagnostic quantities	145
3.11.1	Spatially-variable fields	145
3.11.2	Scalar time-series	159
4	Climate forcing	165
4.1	Managing model time	166
4.1.1	Periodic climate data	166
4.1.2	Using time bounds in forcing data	166
4.2	Examples and corresponding options	167
4.2.1	One way coupling to a climate model	167
4.2.2	Using climate anomalies	167
4.2.3	SeaRISE-Greenland	168
4.2.4	SeaRISE-Greenland paleo-climate run	168
4.2.5	Antarctic paleo-climate runs	168
4.3	Testing if forcing data is used correctly	169

4.3.1	Visualizing climate inputs, without ice dynamics	169
4.3.2	Using low-resolution test runs	169
4.3.3	Visualizing the climate inputs in the Greenland case	170
4.4	Surface mass and energy process model components	172
4.4.1	The “invisible” model	172
4.4.2	Reading top-surface boundary conditions from a file	172
4.4.3	Elevation-dependent temperature and mass balance	174
4.4.4	Temperature-index scheme	175
4.4.5	PIK	178
4.4.6	Scalar temperature offsets	178
4.4.7	Lapse rate corrections	178
4.4.8	Mass flux adjustment	179
4.4.9	Using climate data anomalies	179
4.4.10	The caching modifier	180
4.5	Atmosphere model components	180
4.5.1	Reading boundary conditions from a file	181
4.5.2	Cosine yearly cycle	181
4.5.3	SeaRISE-Greenland	181
4.5.4	PIK	182
4.5.5	One weather station	182
4.5.6	Scalar temperature offsets	182
4.5.7	Scalar precipitation offsets	183
4.5.8	Scalar precipitation scaling	183
4.5.9	Precipitation correction using scalar temperature offsets	183
4.5.10	Temperature and precipitation lapse rate corrections	183
4.5.11	Using climate data anomalies	184
4.6	Ocean model components	184
4.6.1	Constant in time and space	185
4.6.2	Reading forcing data from a file	185
4.6.3	PIK	186
4.6.4	Basal melt rate and temperature from thermodynamics in boundary layer	186
4.6.5	Scalar sea level offsets	186
4.6.6	Scalar sub-shelf temperature offsets	187
4.6.7	Scalar sub-shelf mass flux offsets	187
4.6.8	Scalar sub-shelf mass flux fraction offsets	187
4.6.9	Scalar melange back pressure fraction	188
4.6.10	The caching modifier	188
5	Technical notes	189
5.1	CF standard names used by PISM	189
5.1.1	Existing standard names	189
5.1.2	<i>Proposed</i> standard names	190
5.1.3	Final technical notes	191
5.2	On the vertical coordinate in PISM, and a critical change of variable	191
5.3	Using Schoof’s parameterized bed roughness technique in PISM	194
5.3.1	An explanation	194
5.3.2	Theory	195
5.3.3	Practical application, and Taylor approximation	196
5.4	PISM coding guidelines	198
5.4.1	File names	199
5.4.2	Source and header files	199
5.4.3	Inline functions and methods	199
5.4.4	Including header files	199
5.4.5	Naming	200

5.4.6	Namespaces	200
5.4.7	Formatting	201
5.4.8	Error handling	203
5.4.9	Function design	205
5.5	How do I...?	206
5.5.1	Creating and using configuration flags and parameters	206
5.5.2	Creating and using additional variables	207
5.6	Using IceModelVec2T to load space- and time-dependent forcing fields	211
5.6.1	Rationale	212
5.6.2	Setup	212
5.6.3	Actual use	213
5.6.4	Putting it all together to implement an “ocean model”	214
6	Authorship	217
	Bibliography	219

This *Manual* was built using revision v1.0-3-gc3d2ee74f committed on 2017-10-20 09:43:31 -0800.

ACKNOWLEDGEMENTS

Development of PISM is supported by NASA grant NNX17AG65G and NSF grants PLR-1603799 and PLR-1644277. See [Table 1.1](#) for the full list of grants that supported PISM development.

The Snow, Ice, and Permafrost group at the Geophysical Institute is the home for the University of Alaska PISM developers; find us in Elvey 410D. The Arctic Region Supercomputing Center (now RCS) has provided significant computational resources and technical help in the development of PISM.

Thanks for comments/questions from many PISM users around the world, including these not already listed as PISM authors:

Guðfinna Aðalgeirsdóttir, Antje Fitzner, Nick Golledge, Tore Hattermann, Moritz Huetten, Thomas Kleiner, Leo van Kampenhout, Marianne Madsen, Malou Maris, Tim Morey, Mirena Olaizola, Christian Rodehacke, Nathan Shemonski, Sebastian Simonsen, Anne Solgaard, Ben Sperisen, Synne Høyer Svendsen, Martin Truffer, Shuting Yang, Ryan Woodard

for helpful comments and questions on PISM and this *Manual*. Dave Covey, Don Bahls, and Greg Newby have supported our hardware, software, and computations. Bob Bindschadler, Sophie Nowicki, Jesse Johnson, and others in the SeaRISE group have motivated and assisted PISM development in many ways.

PISM includes the source code for three external libraries we rely on:

- [CalCalcs](#), Copyright © 2010 by David W. Pierce, GPL
- [VDT](#), Copyright © 2012 Danilo Piparo, Vincenzo Innocente, Thomas Hauth (CERN), LGPL
- [cubature.c](#), Copyright © 2005 Steven G. Johnson, GPL

Table 1.1: Past and current grants supporting PISM development.

Start year	End year	Agency	Number	Title
2002	2008	NASA	NAG5-11371	Ice Sheet Modeling, A science team component of Polar Radar for Ice Sheet Measurements (PRISM)
2009	2013	NASA	NNX09AJ38G	A high resolution Parallel Ice Sheet Model including fast, sliding flow: advanced development and application
2013	2017	NASA	NNX13AM16G	Understanding measured variability in the Greenland ice sheet using the Parallel Ice Sheet Model (PISM)
2013	2017	NASA	NNX13AK27G	Challenging the Parallel Ice Sheet Model with reproducing the present-day mass loss signal from the Jakobshavn basin, Greenland
2016	2017	NASA	NNX16AQ40G	Improving Greenland-wide ice discharge projections for the 21st century and beyond
2016	2019	NSF	PLR-1603799	Collaborative Research: Understanding the controls on spatial and temporal variability in ice discharge using a Greenland-wide ice sheet model.
2017	2020	NSF	PLR-1644277	Collaborative Research: Feedbacks between Orographic Precipitation and Ice Dynamics.
2017	2020	NASA	NNX17AG65G	Using ICESat/OIB elevation and satellite-derived velocity changes to constrain time-varying basal motion.

INSTALLING PISM

The fastest path to a fully functional PISM installation is to use a Linux system with a Debian-based package system (e.g. [Ubuntu](#)): Start by following subsection *Installing prerequisites from Debian packages*, then *Building PETSc*, then *Building PISM* to install PISM itself.

2.1 Required tool and libraries

This table lists required dependencies for PISM alphabetically.

Required Library	Comment
CMake	version ≥ 3.1
FFTW	version ≥ 3.1
GSL	version ≥ 1.15
MPI	any recent version
NetCDF ¹	version ≥ 4.1
PETSc ²	version ≥ 3.5
UDUNITS	any recent version

Before installing these “by hand”, check sections *Installing prerequisites from Debian packages* and *Installing required libraries on macOS* for specific how-to.

In particular, if multiple MPI implementations (e.g. MPICH and Open-MPI) are installed then PETSc can under some situations “get confused” and throw MPI-related errors. Even package systems have been known to allow this confusion.

Optional libraries listed below are needed for certain PISM features, namely cell-area correction and parallel I/O. These libraries are recommended, but not strictly required:

Recommended Library	Comment
PROJ.4	Used to compute cell areas and cell bounds
PnetCDF	Can be used for parallel I/O

[Python](#) is needed both in the PETSc installation process and in scripts related to PISM pre- and post-processing, while [Git](#) is usually needed to download the PISM code. Both should be included in any Linux/Unix distribution.

The following Python packages are needed to do all the examples in the *User’s Manual* (which run Python scripts):

¹ Note that PISM uses ncgen (provided by NetCDF) on the system where PISM is *compiled*.

² “PETSc” is pronounced “pet-see”.

Table 2.1: Python packages

Library	Comment
NumPy	used in <i>most</i> scripts
matplotlib	used in some scripts
netcdf4-python	used in <i>most</i> scripts

2.2 Installation Cookbook

Installing PISM requires getting its prerequisites (see [Required tool and libraries](#)) and then building PISM itself (see [Building PISM](#)). Sections below cover some cases; please [e-mail us](#) if you need help.

2.2.1 Installing prerequisites from Debian packages

You should be able to use your package manager to get the prerequisites for PISM. Install the following packages using `apt-get` or `synaptic` or similar. All of these are recommended as they satisfy requirements for building or running PISM.

Table 2.2: Debian packages

Package name	Comments
<code>cmake</code>	required to configure PISM
<code>libfftw3-dev</code>	required by PISM
<code>g++</code>	required to build PISM
<code>libgs10-dev</code>	required by PISM
<code>netcdf-bin</code>	required: <code>ncgen</code> is used during the build process
<code>libnetcdf-dev</code>	required by PISM
<code>libudunits2-dev</code>	required by PISM
<code>cdo</code>	used in some pre-processing scripts
<code>cmake-curses-gui</code>	a text-based easy interface for CMake
<code>git</code>	used to get PISM source code
<code>nco</code>	used in many pre-processing scripts
<code>ncview</code>	view fields in NetCDF files
<code>libproj-dev</code>	used to compute ice area and volume
<code>python-dev</code>	(helps with scripts... perhaps not essential)
<code>python-pyproj</code>	used in some pre-processing scripts
<code>python-netcdf4</code>	used in most post-processing scripts
<code>libx11-dev</code>	X windows is useful to get graphics through PETSc
<code>libblas-dev</code>	BLAS is required by PETSc
<code>liblapack-dev</code>	LAPACK is required by PETSc
<code>openmpi-bin</code>	MPI is required to run PISM in parallel
<code>libopenmpi-dev</code>	MPI is required to run PISM in parallel

You may be able to install these by running

```
sudo apt-get install cmake libfftw3-dev g++ libgs10-dev netcdf-bin \  
libnetcdf-dev libudunits2-dev cdo cmake-curses-gui \  
git nco ncview libproj-dev python-dev python-pyproj \  
python-netcdf4 libx11-dev libblas-dev liblapack-dev \  
openmpi-bin libopenmpi-dev
```


(You may need to change this command to match your package system.)

Once done, see [Building PETSc](#) to install PETSc from source and then [Building PISM](#) for building PISM itself.

2.2.2 Installing required libraries on macOS

Follow these steps to install PISM’s prerequisites on the Mac OS X operating system.

1. As PISM is distributed as source code only, you will need software developer’s tools, [XCode](#) and the *X window system server*, [XQuartz](#).
2. The use of [MacPorts](#) (or [Fink](#), or [Homebrew](#)) is recommended, as it significantly simplifies installing many open-source libraries. These instructions assume that you use [MacPorts](#). Download a package from the [MacPorts](#), install, and set the environment:

```
export PATH=/opt/local/bin:/opt/local/sbin:$PATH
```

3. It may not be necessary to install Python, as it is bundled with the operating system. Some PISM scripts use SciPy; it can be installed using MacPorts or by downloading the [Enthought Python Distribution](#).
4. This MacPorts command should install all of PISM’s required libraries:

```
sudo port install git cmake fftw-3 gsl mpich-default netcdf udunits2 libproj4 ncview
```

5. At this point, all the PISM prerequisites except PETSc are installed. Proceed to [Building PETSc](#).
6. Now you can build PISM as described in section [Building PISM](#).

2.2.3 Installing prerequisites from sources

1. You will need [Python](#) and [Git](#) installed. To use the (recommended) graphical output of PISM you will need an [X Window server](#).
2. Generally the “header files” for its prerequisite libraries are required for building PISM. (This means that the “developer’s versions” of the libraries are needed if the libraries are downloaded from package repositories like Debian’s; see [Required tool and libraries](#).)
3. PISM uses [NetCDF](#) as an input and output file format. If it is not already present, install it using the instructions at the web-page or using a package management system.
4. PISM uses the [GNU Scientific Library](#) for certain numerical calculations and special functions. If it is not already present, install it using the instructions at the web-page or using a package management system.
5. PISM uses the [FFTW library](#) for the deformation of the solid earth (bed) under ice loads. Install FFTW version 3.1 or later, or check that it is installed already.
6. You will need a version of [MPI](#). Your system may have an existing MPI installation, in which case it should probably be used when building PETSc. The goal is to have the PETSc installation use the same version of MPI which is called by the `mpiexec` or `mpirun` executable.

If you had to install an MPI library “by hand” you will want to add the MPI bin directory to your path so that you can run parallel programs using the `mpiexec` or `mpirun` command. For example, you can add it with the statement

```
export PATH=/home/user/mympi/bin:$PATH
```

(for Bash shell).

Such a statement can, of course, appear in your `.bashrc` (or `.profile`) file so that there is no need to retype it each time you use MPI.

7. PISM uses `UDUNITS` to convert units of physical quantities read from input files and written to output files. Follow instructions on its website to install.

2.2.4 Building PETSc

PISM is built on top of `PETSc`, which is actively developed and an up-to-date PETSc distribution is unlikely to be available in package repositories. Download the PETSc source by grabbing the current gzipped tarball at:

<http://www.mcs.anl.gov/petsc/download/index.html>

(Use version 3.5 or newer; see *Required tool and libraries* for details.) The “lite” form of the tarball is fine if you are willing to depend on an Internet connection for accessing PETSc documentation.

You should configure and build PETSc as described on the PETSc installation page, but it might be best to read the following comments on the PETSc configure and build process first:

1. Untar in your preferred location and enter the new PETSc directory. Note PETSc should *not* be configured using root privileges. When you run the configure script the following options are recommended; note PISM uses shared libraries by default:

```
export PETSC_DIR=$PWD
export PETSC_ARCH=opt
./config/configure.py --with-shared-libraries \
                    --with-debugging=0 \
                    --with-fc=0
```

You need to define the environment variables `PETSC_DIR` and `PETSC_ARCH`¹ – one way is shown here – *before* running the configuration script. Turning off the inclusion of debugging code and symbols can give a significant speed improvement, but some kinds of development will benefit from setting `--with-debugging=1`. Using shared libraries may be unwise on certain clusters; check with your system administrator. PISM does not use PETSc’s Fortran API, so the Fortran compiler is disabled by `--with-fc=0`.

2. It is sometimes convenient to have PETSc grab a local copy of BLAS and LAPACK rather than using the system-wide version. So one may add “`--download-f2cblaslapack=1`” to the other configure options.
3. If there is an existing MPI installation, for example at `/home/user/mympi/`, one can point PETSc to it by adding the option “`--with-mpi-dir=/home/user/mympi/`”. The path used in this option must have MPI executables `mpicxx` and `mpicc`, and either `mpiexec` or `mpirun`, in sub-directory `bin/` and MPI library files in sub-directory `lib/`.

Alternatively, use MPI’s compiler wrappers to specify an MPI library when installing PETSc, for example:

```
CC=mpicc CXX=mpicxx ./config/configure.py --with-shared-libraries \
                    --with-debugging=0 \
                    --with-fc=0
```

If you get messages suggesting that PETSc cannot configure using your existing MPI, you might want to try adding the `--download-mpich=1` (or `--download-openmpi=1`) option to PETSc’s configure command.

4. Configuration of PETSc for a batch system requires special procedures described at the PETSc documentation site. One starts with a configure option `--with-batch=1`. See the “Installing on machine requiring cross compiler or a job scheduler” section of the [PETSc installation page](#).

¹ The `PETSC_ARCH` variable is just a string you can use to choose different PETSc configurations and does not have any other significance.

5. Configuring PETSc may take a moment even when everything goes smoothly. A value for the environment variable PETSC_ARCH will be reported at the end of the configure process; take note of this value. One may always reconfigure with additional PETSC_ARCH as needed.
6. After `configure.py` finishes, you will need to make `all test` in the PETSc directory and watch the result. If the X Windows system is functional some example viewers will appear; as noted you will need the X header files for this to work.

2.2.5 Building PISM

To make sure that the key PETSc and MPI prerequisites work properly together, so that you can run PISM in parallel, you might want to make sure that the correct `mpiexec` can be found, by setting your `PATH`. For instance, if you used the option `--download-mpich=1` in the PETSc configure, the MPI bin directory will have a path like `$PETSC_DIR/$PETSC_ARCH/bin`. Thus the following lines might appear in your `.bashrc` or `.profile`, if not there already:

```
export PETSC_DIR=/home/user/petsc-3.7.6/
export PETSC_ARCH=opt
export PATH=$PETSC_DIR/$PETSC_ARCH/bin/:$PATH
```

From now on we will assume that the PETSC_ARCH and PETSC_DIR variables are set.

Follow these steps to build PISM:

1. Get the latest source for PISM using the [Git](#) version control system:

Check [PISM's website](#) for the latest version of PISM. Run

```
git clone git://github.com/pism/pism.git pism-stable
```

A directory called “pism-stable” will be created. Note that in the future when you enter that directory, `git pull` will update to the latest revision of PISM.¹

2. Build PISM:²

```
mkdir -p pism-stable/build
cd pism-stable/build
PISM_INSTALL_PREFIX=~/.pism cmake ..
make install
```

Here `pism-stable` is the directory containing PISM source code while `~/.pism` is the directory PISM will be installed into. All the temporary files created during the build process will be in `pism-stable/build` created above.

You might need to add `CC` and `CXX` to the `cmake` command:

```
PISM_INSTALL_PREFIX=~/.pism CC=mpicc CXX=mpicxx cmake ..
```

Whether this is necessary or not depends on your MPI setup.

Commands above will configure PISM to be installed in `~/.pism/bin` and `~/.pism/lib/` then compile and install all its executables and scripts.

If your operating system does not support shared libraries³, then set `Pism_LINK_STATICALLY` to “ON”. This can be done by either running

¹ Of course, after `git pull` you will make `-C build install` to recompile and re-install PISM.

² Please report any problems you meet at these build stages by [sending us](#) the output.

³ This might be necessary if you're building on a Cray XT5 or a Sun Opteron Cluster, for example.

```
cmake -DPism_LINK_STATICALLY=ON ..
```

or by using `ccmake`⁴ run

```
ccmake ..
```

and then change `Pism_LINK_STATICALLY` (and then press `c` to “configure” and `g` to “generate Makefiles”). Then run `make install`.

Object files created during the build process (located in the `build` sub-directory) are not automatically deleted after installing PISM, so run “`make clean`” if space is an issue. You can also delete the build directory altogether if you are not planning on re-compiling PISM.

Note: When using Intel’s compiler high optimization settings such as `-O3`, `-fp-model precise` may be needed to get reproducible model results. Set it using `ccmake` or by setting `CFLAGS` and `CXXFLAGS` environment variables when building PISM’s prerequisites and PISM itself.

```
export CFLAGS="-fp-model precise"
export CXXFLAGS="-fp-model precise"
cmake [other options] ..
```

Note: To achieve best performance it can be useful to tell the compiler to target the “native” architecture. (This gives it permission to use CPU instructions that may not work on older CPUs.)

```
export CFLAGS="-march=native"
export CXXFLAGS="-march=native"
cmake [other options] ..
```

-
3. PISM executables can be run most easily by adding the `bin/` sub-directory in your selected install path (`~/pism/bin` in the example above) to your `PATH`. For instance, this command can be done in the `Bash` shell or in your `.bashrc` file:

```
export PATH=~/.pism/bin:$PATH
```

4. Now see section *Quick tests of the installation* or *Getting started: a Greenland ice sheet example* to continue.

2.2.6 Common build problems and solutions

We recommend using `ccmake`, the text-based CMake interface to adjust PISM’s build parameters. One can also set CMake cache variables using the `-D` command-line option (`cmake -Dvariable=value`) or by editing `CMakeCache.txt` in the build directory.

Here are some issues we know about.

- Sometimes, if a system has more than one MPI installation CMake finds the wrong one. To tell it which one to use, set `MPI_LIBRARY` and related variables by using `ccmake`. You can also set environment variables `CC` and `CXX` to point to MPI wrappers:

```
CC=mpicc CXX=mpicxx cmake path/to/pism-source
```

⁴ Install the `cmake-curses-gui` package to get `ccmake` on [Ubuntu](#).

It is also possible to guide CMake’s configuration mechanism by setting `MPI_COMPILER` to the compiler (such as `mpicc`) corresponding to the MPI installation you want to use, setting `MPI_LIBRARY` to `MPI_LIBRARY-NOTFOUND` and re-running CMake.

- If you are compiling PISM on a system using a cross-compiler, you will need to disable CMake’s tests trying to determine if PETSc is installed properly. To do this, set `PETSC_EXECUTABLE_RUNS` to “yes”.

To tell CMake where to look for libraries for the target system, see [CMake cross compiling](#) and the paragraph about `CMAKE_FIND_ROOT_PATH` in particular.

- Note that the PISM build system uses `ncgen` from the NetCDF package to generate the configuration file `pism_config.nc`. This means that a working NetCDF installation is required on both the “host” and the “target” systems when cross-compiling PISM.
- Some systems support static libraries only. To build PISM statically and tell CMake not to try to link to shared libraries, set `Pism_LINK_STATICALLY` to `ON` using `ccmake`.
- You can set `Pism_LOOK_FOR_LIBRARIES` to “OFF” to disable all heuristics and set compiler flags by hand. See [HPC builds](#) for examples.

2.3 Quick tests of the installation

Once you’re done with the installation, a few tests can confirm that PISM is functioning correctly.

1. Try a MPI four process verification run:

```
mpiexec -n 4 pismv -test G -y 200
```

If you see some output and a final `Writing model state to file 'unnamed.nc'` then PISM completed successfully. At the end of this run you get measurements of the difference between the numerical result and the exact solution. See [Verification](#) for more on PISM verification.

The above “-n 4” run should work even if there is only one actual processor (core) on your machine. (In that case MPI will just run multiple processes on the one processor.) This run will also produce a NetCDF output file `unnamed.nc`, which can be read and viewed by NetCDF tools.

2. Try an EISMINT II run using the PETSc viewers (under the X window system):

```
pisms -y 5000 -view thk,temppabase,velsurf_mag
```

When using such viewers and `mpiexec` the additional final option `-display :0` is sometimes required to enable MPI to use X, like this:

```
mpiexec -n 2 pisms -y 5000 -view thk,temppabase,velsurf_mag -display :0
```

Also `-drawpause 0.1` or similar may be needed if the figures are refreshing too fast.

3. Run a basic suite of software tests. To do this, make sure that `NCO` and Python packages `NumPy` and `netcdf4-python` are installed. Also, the CMake flag `Pism_BUILD_EXTRA_EXECS` should be `ON`. Then run:

```
make          # do this if you changed something with CMake
make test
```

in the build directory. The message at the bottom should say “100% tests passed, 0 tests failed out of XX” or similar. Feel free to [send us](#) the output of `make test`. if any failed tests cannot be resolved.

2.4 Next steps

Start with the section *Getting started: a Greenland ice sheet example*.

Completely up-to-date documentation can be built from source; see *Rebuilding PISM documentation* for details.

A final reminder with respect to installation: Let's assume you have checked out a copy of PISM using Git, *as described above*. You can then update your copy of PISM to the latest version by running `git pull` in the PISM directory and make `install` in your build directory.

2.5 Rebuilding PISM documentation

You might want to rebuild the documentation from source, as PISM and its documentation evolve together. These tools are required:

Tool	Comment
Sphinx	needed to rebuild this Manual
sphinxcontrib.bibtex	needed to rebuild this Manual and the documentation of PISM's Python bindings (below)
LaTeX	needed to rebuild the PDF version of this Manual, the PISM <i>Source Code Browser</i> , and the documentation of PISM's Python bindings
dvipng	needed to rebuild the documentation of PISM's Python bindings
Latexmk	needed to rebuild the PDF version of this Manual
doxygen	required to rebuild the PISM <i>Source Code Browser</i>
graphviz	required to rebuild the PISM <i>Source Code Browser</i>

Note that if you install Sphinx using [MacPorts](#), you will install a version that corresponds to your Python version, and its executables will have names with suffixes corresponding to this version, e.g. `sphinx-build-2.7` rather than `sphinx-build` for Python 2.7. You will want to set up aliases so that the standard names work as well. To do this, run

```
sudo port select sphinx py27-sphinx
```

(replacing `py27-sphinx` with `py26-sphinx` for Python 2.6, etc.) If you opt not to do this, you can tell CMake the name of your Sphinx executable using

```
cmake -DSPHINX_EXECUTABLE=sphinx-build-2.7 ...
```

for example.

2.5.1 Manual

To rebuild this manual, change to the PISM build directory and run

```
make manual_html # for the HTML version of the manual
make manual_pdf  # for the PDF version of the manual
```

The main page for this manual is then in `doc/sphinx/html/index.html` inside your build directory.

The PDF manual will be in `doc/sphinx/pism_manual.pdf` in your build directory.

2.5.2 Source Code Browser

To rebuild the PISM *Source Code Browser*, change to the PISM build directory and run

```
make browser
```

The main page for the documentation is then in `doc/browser/html/index.html` inside your build directory.

2.5.3 Documentation for PISM's Python bindings and inversion tools

These bindings make scripting and interactive use of PISM possible, but many PISM users will not need them. Installing them is required to use PISM for inversion of surface velocities for basal shear stress and ice hardness. Building their documentation is strongly-recommended before use.

To build the documentation of PISM's Python bindings, change to the PISM build directory and run

```
make pismpython_docs
```

The main page for the documentation is then in `doc/pismpython/html/index.html` inside your build directory. The documentation build can take some time while it builds a large number of small images from LaTeX formulas.

2.5.4 Re-building documentation without PISM's prerequisites

To build documentation on a system without PISM's prerequisite libraries (such as MPI and PETSc), assuming that PISM sources are in `~/pism-stable`, do the following:

```
cd ~/pism-stable
mkdir doc-build # create a build directory
cd doc-build
cmake ../doc
```

then commands “`make manual_html`”, “`make manual_pdf`” and others (see above) will work as expected.

PISM USER'S MANUAL

Welcome! All information about PISM is online at the home page

<http://www.pism-docs.org>

Please see the extensive lists of PISM publications and applications at that page.

This User's Manual gives examples of how to run PISM using publicly-available data for: the whole Greenland ice sheet, the Jakobshavn outlet glacier in Greenland, the Ross ice shelf in Antarctica, and a number of simplified geometry tests. It documents all the PISM's command-line options and configuration parameters. It summarizes the continuum models used by PISM, and it illustrates how PISM's numerical approximations are verified.

See the *Installation Manual* for how to download the PISM source code and install it, along with needed libraries. The *Climate Forcing Manual* extends this Manual to cover additional couplings to atmosphere and ocean models and data.

Users who want to understand more deeply how PISM is designed, or who want to extend it, will need to go beyond what is described here. See the *Source Code Browser*, which is online for the latest stable version. It can be generated from source code as described in *Rebuilding PISM documentation*. It gives a complete view of the class/object structure of the PISM source code.

Warning: PISM is an ongoing research project. Ice sheet modeling requires many choices. Please don't trust the results of PISM or any other ice sheet model without a fair amount of exploration. Also, please don't expect all your questions to be answered here. [Write to us](#) with questions.

3.1 Getting started: a Greenland ice sheet example

This introduction is intended to be interactive and participatory, and it should work on *your personal machine* as well as on a supercomputer. Please try the commands and view the resulting files. Do the runs with your own values for the options. We can't hide the fact that PISM has lots of "control knobs," but fiddling with them will help you get going. Give it a try!

We get started with an extended example showing how to generate initial states for prognostic model experiments on the Greenland ice sheet. Ice sheet and glacier model studies often involve modeling present and past states using actions like the ones demonstrated here. Our particular choices made here are motivated by the evaluation of initialization methods in [55].

We use data assembled by the *Sea-level Response to Ice Sheet Evolution (SeaRISE)* assessment process [51]. SeaRISE is a community-organized assessment process providing an upper bound on ice sheet contributions to sea level in the next 100–200 years, especially for the IPCC AR5 report in 2013.

This example is a hands-on first look at PISM. It is not an in-depth tutorial, and some details of what is happening are only explained later in this Manual, which thoroughly discusses PISM options, nontrivial modeling choices, and how to preprocess input data.

The basic runs here, mostly on coarse 20 and 10 km grids, can be done on a typical workstation or laptop. PISM is, however, designed to make high resolution (e.g. 5 km to ~ 500 m grids for whole-Greenland ice sheet modeling) possible by exploiting large-scale parallel processing. See [55], [112], [74], among other published high-resolution PISM examples.

3.1.1 Input data

The NetCDF data used to initialize SeaRISE runs is [freely-available online](#).

To download the specific file we want, namely `Greenland_5km_v1.1.nc`, and preprocess it for PISM, do:

```
cd pism/examples/std-greenland
./preprocess.sh
```

The script `preprocess.sh` requires `wget` and also the [NetCDF Operators](#). It downloads the version 1.1 of the SeaRISE “master” present-day data set, which contains ice thickness and bedrock topography from BEDMAP [113], and modeled precipitation and surface mass balance rates from RACMO [52], among other fields.

In particular, it creates three new NetCDF files which can be read by PISM. The spatially-varying fields, with adjusted metadata, go in `pism_Greenland_5km_v1.1.nc`. The other two new files contain famous time-dependent paleoclimate records from ice and seabed cores: `pism_dT.nc` has the GRIP temperature record [4] and `pism_dSL.nc` has the SPECMAP sea level record [5].

Any of these NetCDF files can be viewed with `ncview` or other NetCDF visualization tools; see Table 3.21. An application of IDV to the master data set produced Fig. 3.1, for example. Use `ncdump -h` to see the metadata and history of the files.

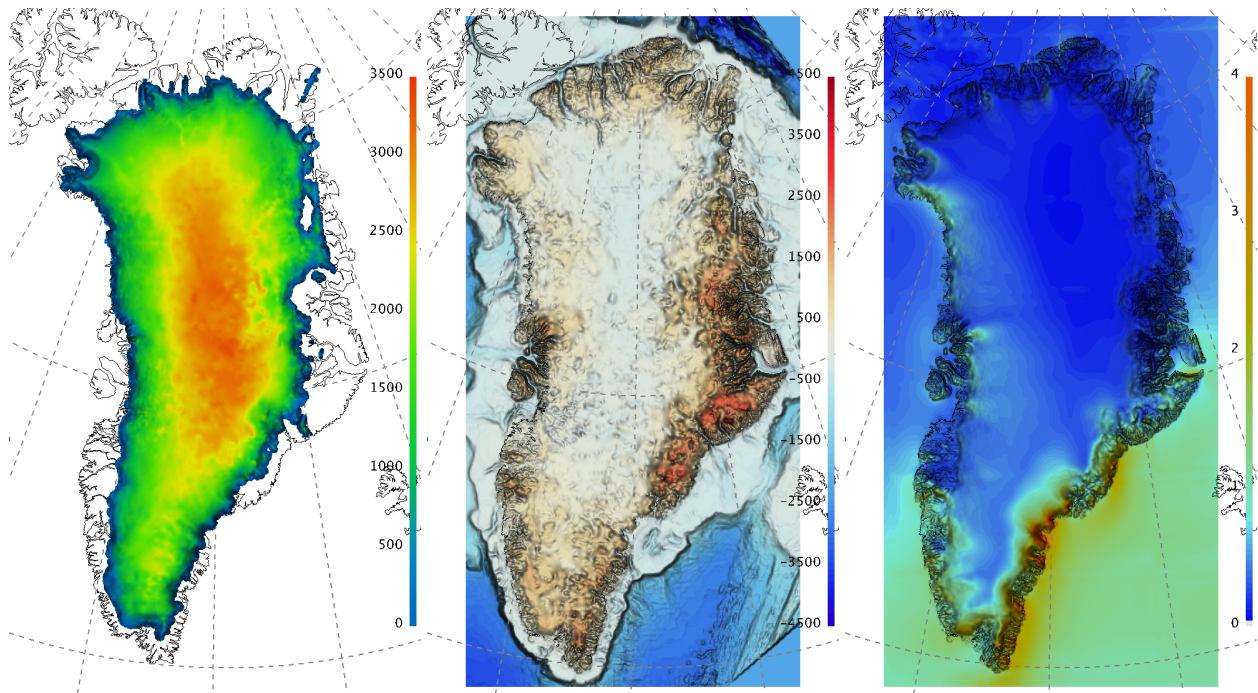


Fig. 3.1: The input file contains present-day ice thickness (left; m), bedrock elevation (center; m), and present-day precipitation (right; $m/year$ ice equivalent) for SeaRISE-Greenland. These are fields `thk`, `topg`, and `precipitation`, respectively, in `pism_Greenland_5km_v1.1.nc`.

3.1.2 First run

Like many Unix programs, PISM allows a lot of command-line options. In fact, because the variety of allowed ice sheet, shelf, and glacier configurations, and included sub-models, is so large, command-line options are covered in sections *Initialization and bootstrapping* through *Practical usage* of this manual.¹ In practice one often builds scripts to run PISM with the correct options, which is what we show here. The script we use is “spinup.sh” in the `examples/std-greenland/` subdirectory of `pism/`.

Note that initializing ice sheets, usually called “spin-up”, can be done by computing approximate steady states with constant boundary data, or, in some cases, by integrating paleo-climatic and long-time-scale information, also applied at the ice sheet boundary, to build a model for the present state of the ice sheet. Both of these possibilities are illustrated in the `spinup.sh` script. The spin-up stage of using an ice sheet model may actually require more processor-hours than follow-on “experiment” or “forecast” stages.

To see what can be done with the script, read the usage message it produces:

```
./spinup.sh
```

The simplest spin-up approach is to use a “constant-climate” model. We take this approach first. To see a more detailed view of the PISM command for the first run, do:

```
PISM_DO=echo ./spinup.sh 4 const 10000 20 sia g20km_10ka.nc
```

Setting the environment variable `PISM_DO` in this way tells `spinup.sh` just to print out the commands it is about to run instead of executing them. The “proposed” run looks like this:

```
mpiexec -n 4 pismr \
-i pism_Greenland_5km_v1.1.nc -bootstrap -Mx 76 -My 141 \
-Mz 101 -Mbz 11 -z_spacing equal -Lz 4000 -Lbz 2000 -skip -skip_max 10 \
-grid.correct_cell_areas false -periodicity none -ys -10000 -ye 0 \
-surface given -surface_given_file pism_Greenland_5km_v1.1.nc \
-calving ocean_kill -ocean_kill_file pism_Greenland_5km_v1.1.nc \
-sia_e 3.0 \
-ts_file ts_g20km_10ka.nc -ts_times -10000:yearly:0 \
-extra_file ex_g20km_10ka.nc -extra_times -10000:100:0 \
-extra_vars diffusivity,temppabase,tempicethk_basal,bmelt,tillwat,velsurf_mag,mask,thk,topg,
-usurf \
-o g20km_10ka.nc
```

Let’s briefly deconstruct this run.

At the front is “`mpiexec -n 4 pismr`”. This means that the PISM executable `pismr` is run in parallel using four processes (usually one per CPU core) under the *Message Passing Interface*. Though we are assuming you have a workstation or laptop with at least 4 cores, this example will work with 1 to about 50 processors, with reasonably good scaling in speed. Scaling can be good with more processors if we run at higher spatial resolution [17], [111]. The executable name “`pismr`” stands for the standard “run” mode of PISM (in contrast to specialized modes described later in sections *Verification* and *Simplified geometry experiments*).

Next, the proposed run uses option `-bootstrap` to start the run by “bootstrapping.” This term describes the creation, by heuristics and highly-simplified models, of the mathematical initial conditions required for a deterministic, time-dependent ice dynamics model. Then the options describe a 76×141 point grid in the horizontal, which gives 20 km grid spacing in both directions. Then there are choices about the vertical extent and resolution of the computational grid; more on those later. After that we see a description of the time axis, with a start and end time given: “`-ys -10000 -ye 0`”.

Then we get the instructions that tell PISM to read the upper surface boundary conditions (i.e. climate) from a file: “`-surface given -surface_given_file pism_Greenland_5km_v1.1.nc`”. For more on these choices,

¹ Moreover, every configuration parameter can be set using a command-line option with the same name.

see subsection *Climate inputs, and their interface with ice dynamics*, and also the *Climate Forcing Manual*.

Then there are a couple of options related to ice dynamics. First is a minimal calving model which removes ice at the calving front location given by a thickness field in the input file (“-calving ocean_kill”); see section *Calving* for this and other calving options). Then there is a setting for enhanced ice softness (“-sia_e 3.0”). See section *Ice rheology* for more on this enhancement parameter, which we also return to later in section *An ice dynamics parameter study*.

Then there are longish options describing the fields we want as output, including scalar time series (“-ts_file ts_g20km_10ka.nc -ts_times -10000:yearly:0”; see section *Practical usage*) and space-dependent fields (“-extra_file ...”; again see section *Practical usage*), and finally the named output file (“-o g20km_10ka.nc”).

Note that the modeling choices here are reasonable, but they are not the only way to do it! The user is encouraged to experiment; that is the point of a model.

Now let’s actually get the run going:

```
./spinup.sh 4 const 10000 20 sia g20km_10ka.nc &> out.g20km_10ka &
```

The terminating “&”, which is optional, asks the system to run the command in the background so we can keep working in the current shell. Because we have re-directed the text output (“&> out.g20km_10ka”), PISM will show what it is doing in the text file out.g20km_10ka. Using less is a good way to watch such a growing text-output file. This run should take around 20 minutes.

3.1.3 Watching the first run

As soon as the run starts it creates time-dependent NetCDF files ts_g20km_10ka.nc and ex_g20km_10ka.nc. The latter file, which has spatially-dependent fields at each time, is created after the first 100 model years, a few wall clock seconds in this case. The command -extra_file ex_g20km_10ka.nc -extra_times -10000:100:0 adds a spatially-dependent “frame” at model times -9900, -9800, ..., 0.

To look at the spatial-fields output graphically, do:

```
ncview ex_g20km_10ka.nc
```

We see that ex_g20km_10ka.nc contains growing “movies” of the fields chosen by the -extra_vars option. A frame of the ice thickness field thk is shown in Fig. 3.2 (left).

The time-series file ts_g20km_10ka.nc is also growing. It contains spatially-averaged “scalar” diagnostics like the total ice volume or the ice-sheet-wide maximum velocity (variable ice_volume_glacierized and max_hor_vel, respectively). It can be viewed by running

```
ncview ts_g20km_10ka.nc
```

The growing time series for ice_volume_glacierized is shown in Fig. 3.2 (right). Recall that our intention was to generate a minimal model of the Greenland ice sheet in approximate steady-state with a steady (constant-in-time) climate. The measurable steadiness of the ice_volume_glacierized time series is a possible standard for steady state (see [14], for example).

At the end of the run the output file g20km_10ka.nc is generated. Fig. 3.3 shows some fields from this file. In the next subsections we consider their “quality” as model results. To see a report on computational performance, we do:

```
ncdump -h g20km_10ka.nc |grep history
:history = "user@machine 2017-10-04 19:16:08 AKDT: PISM done. Performance stats: 0.1784 wall_
-clock hours, 0.7136 proc.-hours, 14005.0054 model years per proc.-hour.\n",
```

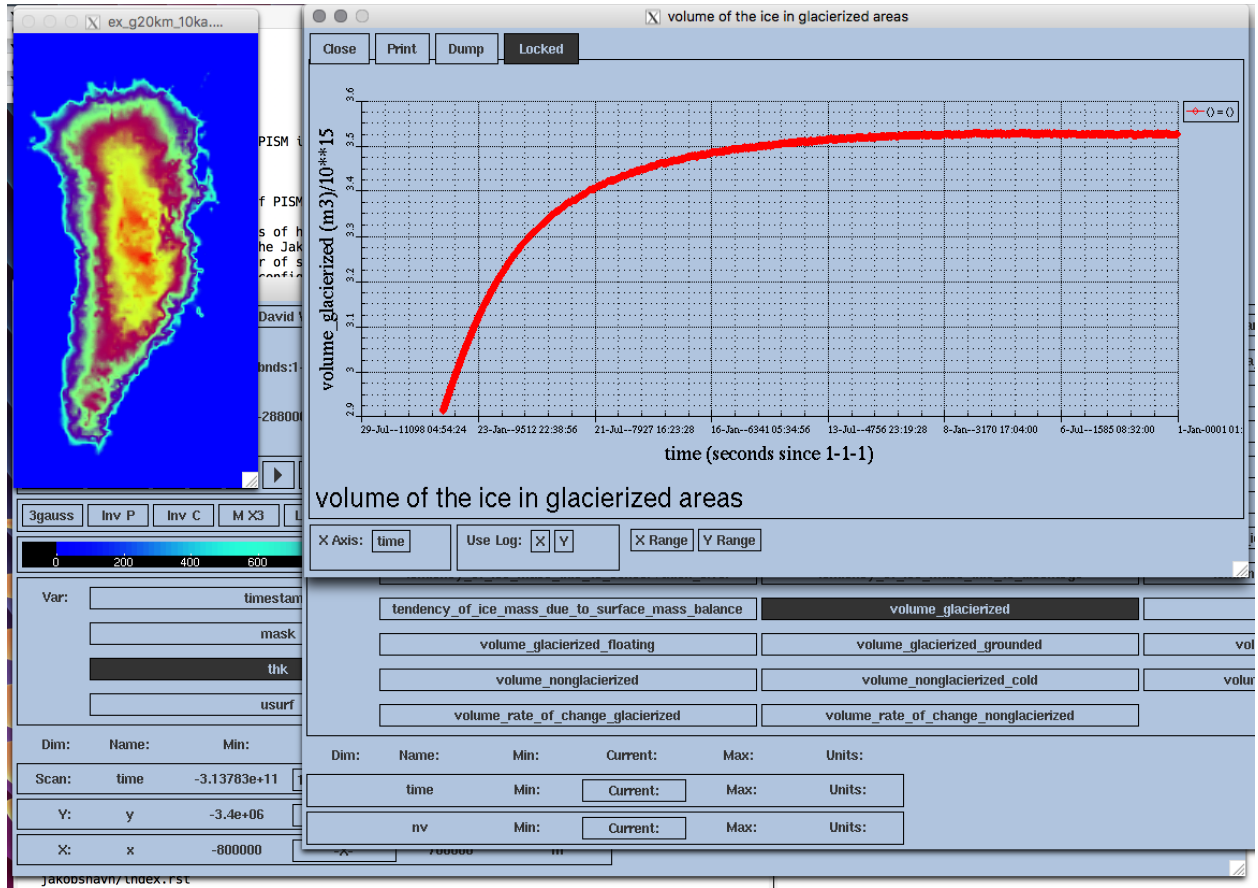


Fig. 3.2: Two views produced by ncview during a PISM model run.

Left thk, the ice sheet thickness, a space-dependent field, from file ex_g20km_10ka.nc.

Right ice_volume_glacierized, the total ice sheet volume time-series, from file ts_g20km_10ka.nc.

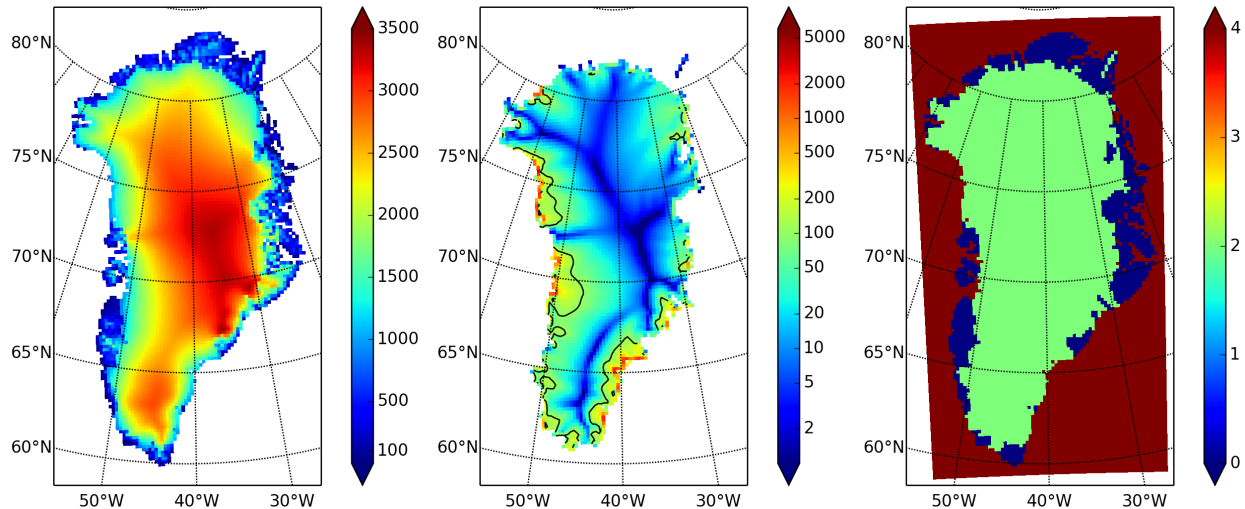


Fig. 3.3: Fields from output file `g20km_10ka.nc`.

Left `usurf`, the ice sheet surface elevation in meters.

Middle `velsurf_mag`, the surface speed in meters/year, including the 100 m/year contour (solid black).

Right `mask`, with 0 = ice-free land, 2 = grounded ice, 4 = ice-free ocean.

3.1.4 Second run: a better ice-dynamics model

It is widely-understood that ice sheets slide on their bases, especially when liquid water is present at the base (see [40], [32], among others). An important aspect of modeling such sliding is the inclusion of membrane or “longitudinal” stresses into the stress balance [17]. The basic stress balance in PISM which involves membrane stresses is the Shallow Shelf Approximation (SSA) [30]. The stress balance used in the previous section was, by contrast, the (thermomechanically-coupled) non-sliding, non-membrane-stress Shallow Ice Approximation (SIA) [18], [14]. The preferred ice dynamics model within PISM, that allows both sliding balanced by membrane stresses and shear flow as described by the SIA, is the SIA+SSA “hybrid” model [17], [25]. For more on stress balance theories see section *Ice dynamics, the PISM view* of this Manual.

The practical issue with models of sliding is that a distinctly-uncertain parameter space must be introduced. This especially involves parameters controlling the amount and pressure of subglacial water (see [55], [78], [81], [54], among others). In this regard, PISM uses the concept of a saturated and pressurized subglacial till with a modeled distribution of yield stress [17], [33]. The yield stress arises from the PISM model of the production of subglacial water, which is itself computed through the conservation of energy model [23]. We use such models in the rest of this Getting Started section.

While the `spinup.sh` script has default sliding-related parameters, for demonstration purposes we change one parameter. We replace the default power $q = 0.25$ in the sliding law (the equation which relates both the subglacial sliding velocity and the till yield stress to the basal shear stress which appears in the SSA stress balance) by a less “plastic” and more “linear” choice $q = 0.5$. See section *Controlling basal strength* for more on sliding laws. To see the run we propose, do

```
PISM_DO=echo PARAM_PPQ=0.5 ./spinup.sh 4 const 10000 20 hybrid g20km_10ka_hy.nc
```

Now remove “`PISM_DO=echo`” and redirect the text output into a file to start the run:

```
PARAM_PPQ=0.5 ./spinup.sh 4 const 10000 20 hybrid g20km_10ka_hy.nc &> out.g20km_10ka_hy &
```

This run should take 10 minutes or less.¹

When this run is finished it produces `g20km_10ka_hy.nc`. As before do

```
ncdump -h g20km_10ka_hy.nc |grep history
```

to see performance results for your machine.

The results of this run are shown in Fig. 3.4. We show the basal sliding speed field `velbase_mag` in this Figure, where Fig. 3.3 had the mask, but the reader can check that `velbase_mag` is zero in the nonsliding SIA-only result `g20km_10ka.nc`.

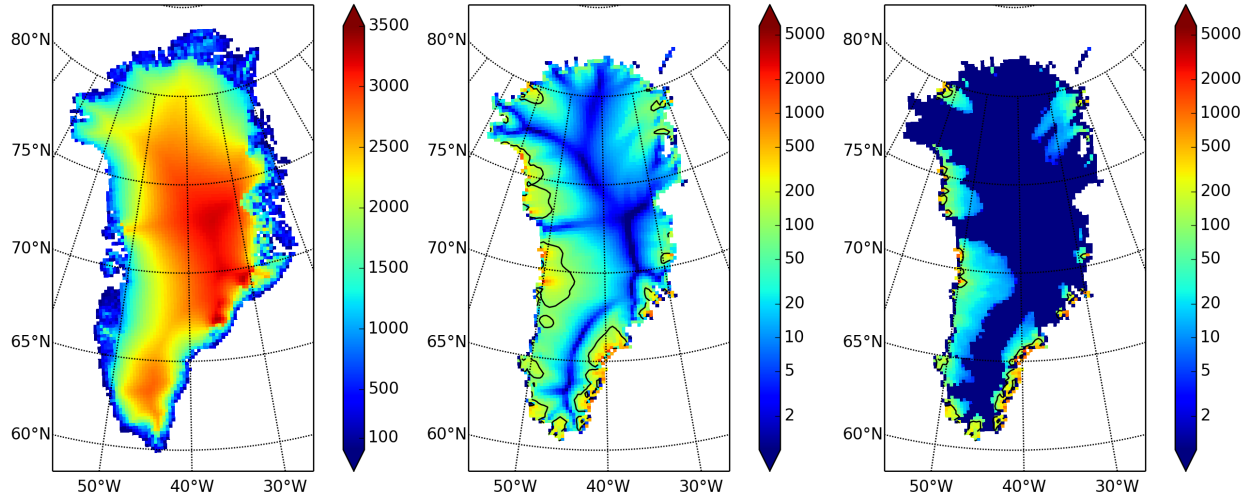


Fig. 3.4: Fields from output file `g20km_10ka_hy.nc`.

Left `usurf`, the ice sheet surface elevation in meters.

Middle `velsurf_mag`, the surface speed in m/year, including the 100 m/year contour (solid black).

Right the sliding speed `velbase_mag`, shown the same way as `velsurf_mag`.

The hybrid model includes sliding, and it is important to evaluate that aspect of the output. However, though it is critical to the response of the ice to changes in climate, basal sliding velocity is essentially unobservable in real ice sheets. On the other hand, because of relatively-recent advances in radar and image technology and processing [117], the surface velocity of an ice sheet can be measured.

So, how good is our model result `velsurf_mag`? Fig. 3.5 compares the radar-observed `surfvelmag` field in the downloaded SeaRISE-Greenland data file `Greenland_5km_v1.1.nc` with the just-computed PISM result. The reader might agree with these broad qualitative judgements:

- the model results and the observed surface velocity look similar, and
- slow near-divide flow is generally in the right areas and of generally the right magnitude, but
- the observed Northeast Greenland ice stream is more distinct than in the model.

We can compare these PISM results to other observed-vs-model comparisons of surface velocity maps, for example Figure 1 in [116] and Figure 8 in [115]. Only ice-sheet-wide parameters and models were used here in PISM, that is, each location in the ice sheet was modeled by the same physics. By comparison, those published comparisons

¹ Regarding the relative speeds of the runs that produce `g20km_10ka.nc` and `g20km_10ka_hy.nc`, note that the computation of the SSA stress balance is substantially more expensive than the SIA in a per-step sense. However, the SSA stress balance in combination with the mass continuity equation causes the maximum diffusivity in the ice sheet to be substantially lower during the run. Because the maximum diffusivity controls the time-step in the PISM adaptive time-stepping scheme [18], the number of time steps is reduced in the hybrid run. To see this contrast use `ncview ts_g20km_10ka.nc` to view variables `max_diffusivity` and `dt`.

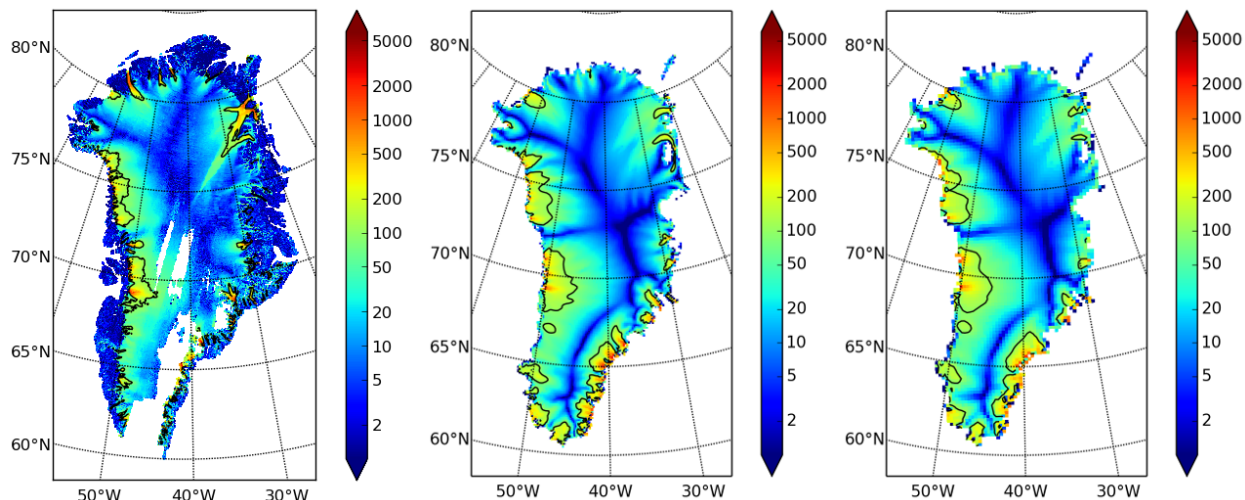


Fig. 3.5: Comparing observed and modeled surface speed.

All figures have a common scale (m/year), with 100 m/year contour shown (solid black).

Left surfvelmag, the observed values from SeaRISE data file Greenland_5km_v1.1.nc.

Middle velsurf_mag from g20km_10ka_hy.nc.

Right velsurf_mag from g10km_10ka_hy.nc.

involved tuning a large number of spatially-variable subglacial parameters to values which would yield close match to observations of the surface velocity. Such tuning techniques, called “inversion” or “assimilation” of the surface velocity data, are also possible in PISM,² but the advantage of having few parameters in a model is well-known: the results reflect the underlying model, not the flexibility of many parameters.

We have only tried two of the many models possible in PISM, and we are free to identify and adjust important parameters. The first parameter change we consider, in the next subsection, is one of the most important: grid resolution.

3.1.5 Third run: higher resolution

Now we change one key parameter, the grid resolution. Model results differ even when the only change is the resolution. Using higher resolution “picks up” more detail in the bed elevation and climate data.

If you can let it run overnight, do

```
PARAM_PPQ=0.5 ./spinup.sh 4 const 10000 10 hybrid g10km_10ka_hy.nc &> out.g10km_10ka_hy &
```

This run might take 4 to 6 hours. However, supposing you have a larger parallel computer, you can change “mpirun -n 4” to “mpirun -n N” where N is a substantially larger number, up to 100 or so with an expectation of reasonable scaling on this grid [17], [111].

Some fields from the result g10km_10ka_hy.nc are shown in Fig. 3.6. Fig. 3.5 also compares observed velocity to the model results from 20 km and 10 km grids. As a different comparison, Fig. 3.7 shows ice volume time series ice_volume_glacierized for 20 km and 10 km runs done here. We see that this result depends on resolution, in particular because higher resolution grids allow the model to better resolve the flux through topographically-controlled outlet glaciers (compare [119]). However, because the total ice sheet volume is a highly-averaged quantity, the ice_volume_glacierized difference from 20 km and 10 km resolution runs is only about one part in 60 (about

² See [118] (inversion of DEMs for basal topography) and [56] (inversion surface velocities for basal shear stress) for PISM-based inversion methods and analysis.

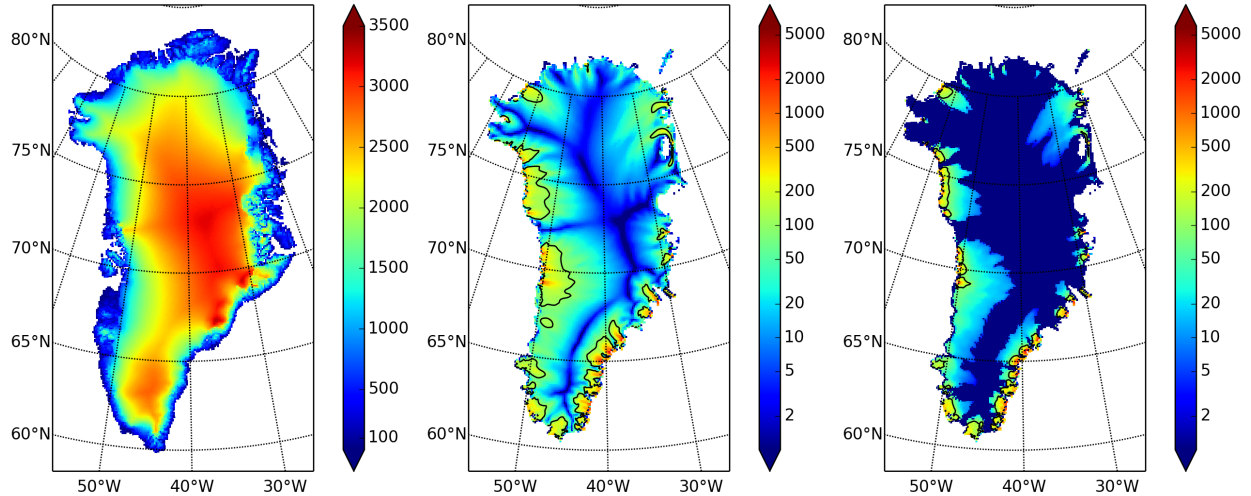


Fig. 3.6: Fields from output file `g10km_10ka_hy.nc`.
Compare to Fig. 3.4, which only differs by resolution.

Left `usurf` in meters.

Middle `velsurf_mag` in m/year.

Right `velbase_mag` in m/year.

1.5%) at the final time. By contrast, as is seen in the near-margin ice in various locations shown in Fig. 3.5, the ice velocity at a particular location may change by 100% when the resolution changes from 20 km to 10 km.

Roughly speaking, the reader should only consider trusting model results which are demonstrated to be robust across a range of model parameters, and, in particular, which are shown to be relatively-stable among relatively-high resolution results for a particular case. Using a supercomputer is justified merely to confirm that lower-resolution runs were already “getting” a given feature or result.

3.1.6 Fourth run: paleo-climate model spin-up

At this point we have barely mentioned one of the most important players in an ice sheet model: the surface mass balance (SMB) model. Specifically, an SMB model combines precipitation (e.g. [120] for present-day Greenland) and a model for melt. Melt models are always based on some approximation of the energy available at the ice surface [15]. Previous runs in this section used a “constant-climate” assumption, which specifically meant using the modeled present-day SMB rates from the regional climate model RACMO [52], as contained in the SeaRISE-Greenland data set `Greenland_5km_v1.1.nc`.

While a physical model of ice dynamics only describes the movement of the ice, the SMB (and the sub-shelf melt rate) are key inputs which directly determine changes in the boundary geometry. Boundary geometry changes then influence the stresses seen by the stress balance model and thus the motion.

There are other methods for producing SMB than using present-day modeled values. We now try such a method, a “paleo-climate spin-up” for our Greenland ice sheet model. Of course, direct measurements of prior climates in Greenland are not available as data! There are, however, estimates of past surface temperatures at the locations of ice cores (see [4] for GRIP), along with estimates of past global sea level [5] which can be used to determine where the flotation criterion is applied—this is how PISM’s mask variable is determined. Also, models have been constructed for how precipitation differs from the present-day values [2]. For demonstration purposes, these are all used in the next run. The relevant options are further documented in the *Climate Forcing Manual*.

As noted, one must compute melt in order to compute SMB. Here this is done using a temperature-index, “positive degree-day” (PDD) model [15]. Such a PDD model has parameters for how much snow and/or ice is melted when

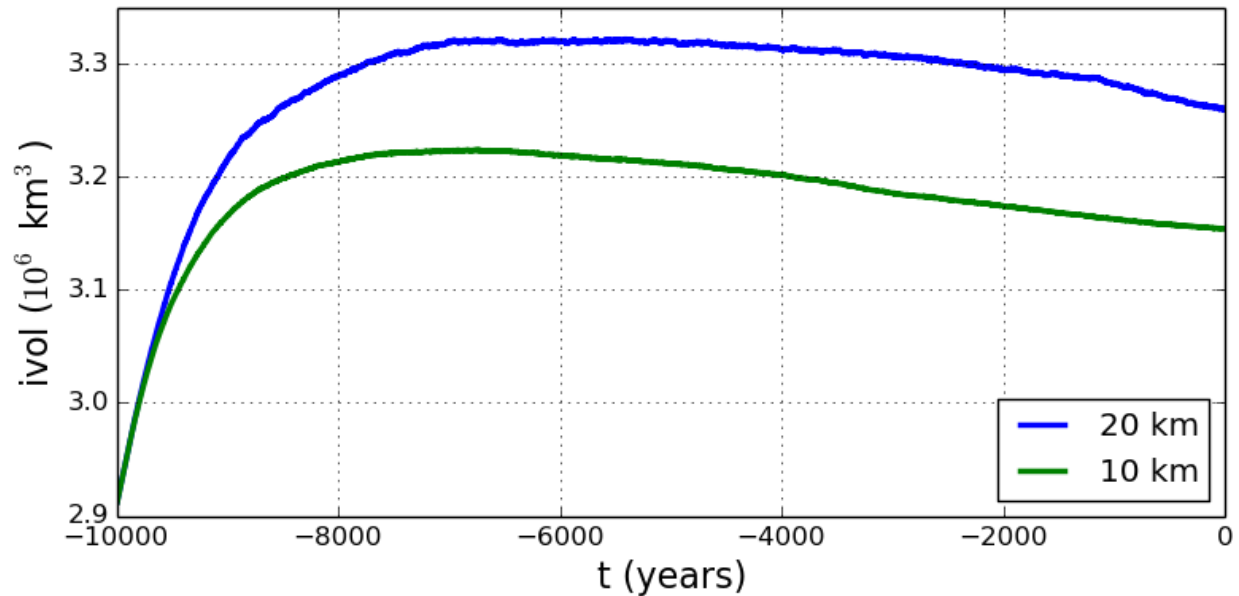


Fig. 3.7: Time series of modeled ice sheet volume `ice_volume_glacierized` on 20km and 10km grids. The present-day ice sheet has volume about $2.9 \times 10^6 \text{ km}^3$ [113], the initial value seen in both runs.

surface temperatures spend time near or above zero degrees. Again, see the *Climate Forcing Manual* for relevant options.

To summarize the paleo-climate model applied here, temperature offsets from the GRIP core record affect the snow energy balance, and thus the rates of melting and runoff calculated by the PDD model. In warm periods there is more marginal ablation, but precipitation may also increase (according to a temperature-offset model [2]). Additionally sea level undergoes changes in time and this affects which ice is floating. Finally we add an earth deformation model, which responds to changes in ice load by changing the bedrock elevation [84].

To see how all this translates into PISM options, run

```
export PARAM_PPQ=0.5
export REGRIDFILE=g20km_10ka_hy.nc
PISM_DO=echo ./spinup.sh 4 paleo 25000 20 hybrid g20km_25ka_paleo.nc
```

You will see an impressively-long command, which you can compare to the *first one*. There are several key changes. First, we do not start from scratch but instead from a previously computed near-equilibrium result:

```
-regrid_file g20km_10ka_hy.nc -regrid_vars litho_temp,thk,enthalpy,tillwat,bmelt
```

For more on regridding see section *Regridding*. Then we turn on the earth deformation model with option `-bed_def 1c`; see section *Earth deformation models*. After that the atmosphere and surface (PDD) models are turned on and the files they need are identified:

```
-atmosphere searise_greenland,delta_T,paleo_precip -surface pdd \
-atmosphere_paleo_precip_file pism_dT.nc -atmosphere_delta_T_file pism_dT.nc
```

Then the ocean model, which provides both a subshelf melt rate and a time-dependent sea level to the ice dynamics core, is turned on with `-ocean constant,delta_SL` and the file it needs is identified with `-ocean_delta_SL_file pism_dSL.nc`. For all of these “forcing” options, see the *Climate Forcing Manual*. The remainder of the options are similar or identical to the run that created `g20km_10ka_hy.nc`.

To actually start the run, which we rather arbitrarily start at year -25000 , essentially at the LGM, do:

```
export PARAM_PPQ=0.5
export REGRIDFILE=g20km_10ka_hy.nc
./spinup.sh 4 paleo 25000 20 hybrid g20km_25ka_paleo.nc &> out.g20km_25ka_paleo &
```

This run should only take one or two hours, noting it is at a coarse 20 km resolution.

The fields `usurf`, `velsurf_mag`, and `velbase_mag` from file `g20km_25ka_paleo.nc` are sufficiently similar to those shown in Fig. 3.4 that they are not shown here. Close inspection reveals differences, but of course these runs only differ in the applied climate and run duration and not in resolution or ice dynamics parameters.

To see the difference between runs more clearly, Fig. 3.8 compares the time-series variable `ice_volume_glacierized`. We see the effect of option `-regrid_file g20km_10ka_hy.nc -regrid_vars . . , thk, . . .`, which implies that the paleo-climate run starts with the ice geometry from the end of the constant-climate run.

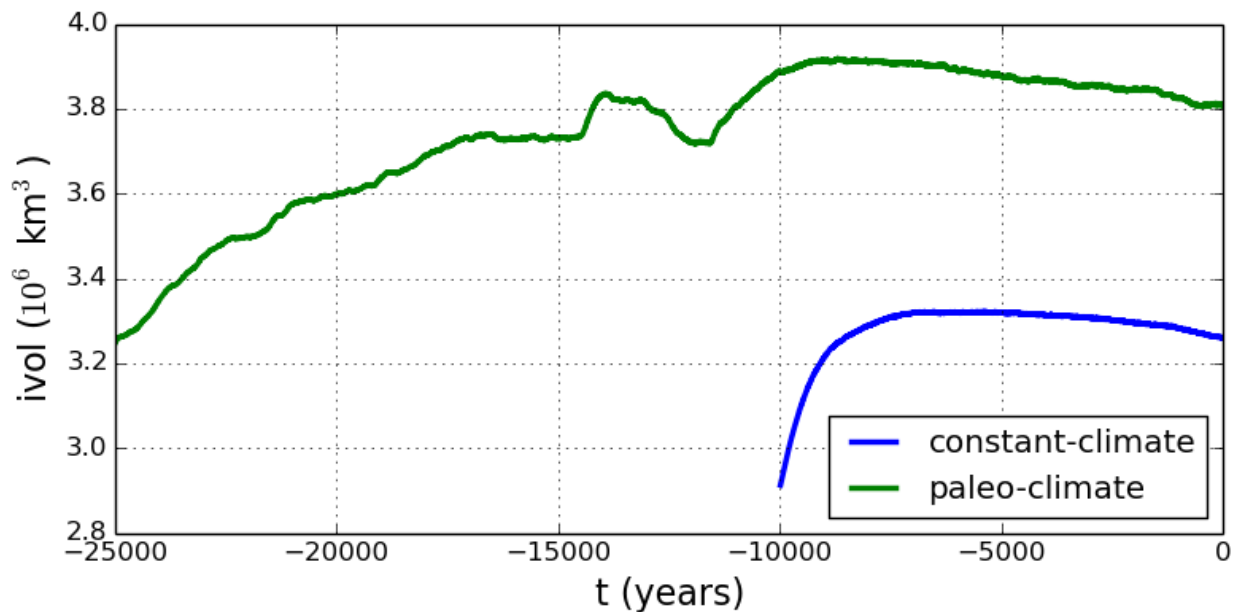


Fig. 3.8: Time series of modeled ice sheet volume `ice_volume_glacierized` from constant-climate (blue; `ts_g20km_10ka_hy.nc`) and paleo-climate (green; `ts_g20km_25ka_paleo.nc`) spinup runs. Note that the paleo-climate run started with the ice geometry at the end of the constant-climate run.

Another time-series comparison, of the variable `ice_volume_glacierized_temperate`, the total volume of temperate (at 0°C) ice, appears in Fig. 3.9. The paleo-climate run shows the cold period from ≈ -25 ka to ≈ -12 ka. Both constant-climate and paleo-climate runs then come into rough equilibrium in the holocene. The bootstrapping artifact, seen at the start of the constant-climate run, which disappears in less than 1000 years, is avoided in the paleo-climate run by starting with the constant-climate end-state. The reader is encouraged to examine the diagnostic files `ts_g20km_25ka_paleo.nc` and `ex_g20km_25ka_paleo.nc` to find more evidence of the (modeled) climate impact on the ice dynamics.

3.1.7 Grid sequencing

The previous sections were not very ambitious. We were just getting started! Now we demonstrate a serious PISM capability, the ability to change, specifically to *refine*, the grid resolution at runtime.

One can of course do the longest model runs using a coarse grid, like the 20 km grid used first. It is, however, only possible to pick up detail from high quality data, for instance bed elevation and high-resolution climate data, using

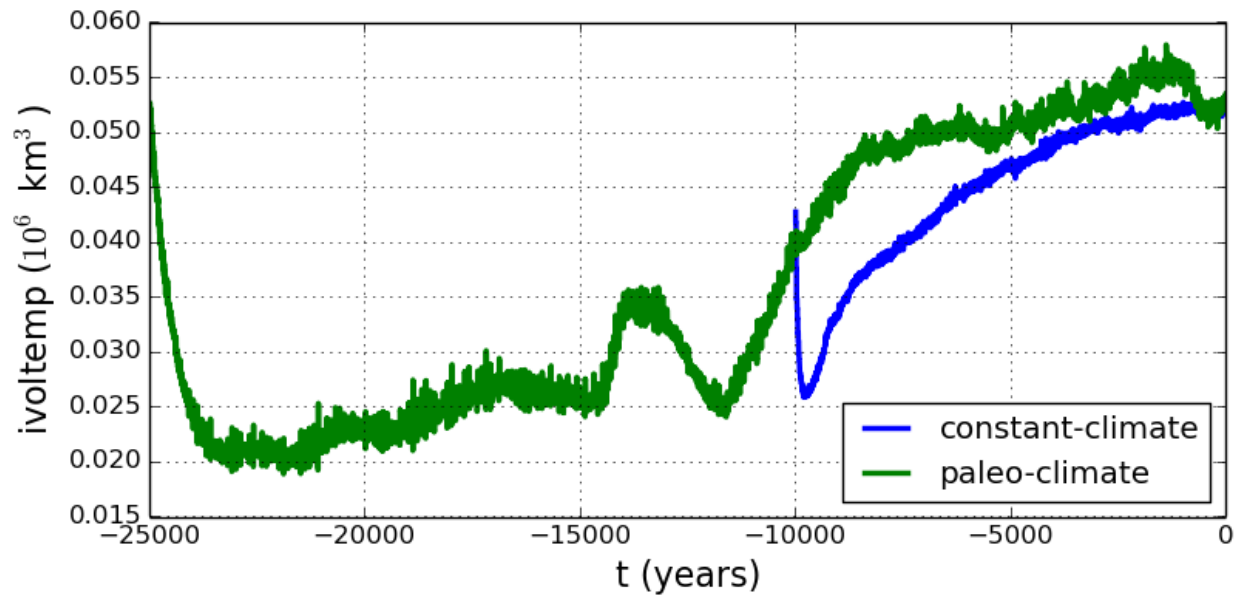


Fig. 3.9: Time series of temperate ice volume `ice_volume_glacierized_temperate` from constant-climate (blue; `ts_g20km_10ka_hy.nc`) and paleo-climate (green; `ts_g20km_25ka_paleo.nc`) spinup runs. The cold of the last ice age affects the fraction of temperate ice. Note different volume scale compared to that in Fig. 3.8; only about 1% of ice is temperate (by volume).

high grid resolution.

A 20 or 10 km grid is inadequate for resolving the flow of the ice sheet through the kind of fjord-like, few-kilometer-wide topographical confinement which occurs, for example, at Jakobshavn Isbrae in the west Greenland ice sheet [110], an important outlet glacier which both flows fast and drains a large fraction of the ice sheet. One possibility is to set up an even higher-resolution PISM regional model covering only one outlet glacier, but this requires decisions about coupling to the whole ice sheet flow. (See section *Example: A regional model of the Jakobshavn outlet glacier in Greenland*.) But here we will work on high resolution for the whole ice sheet, and thus all outlet glaciers.

Consider the following command and compare it to the *first one*:

```
mpiexec -n 4 pismr -i pism_Greenland_5km_v1.1.nc -bootstrap -Mx 301 -My 561 \
-Mz 201 -Mbz 21 -z_spacing equal -Lz 4000 -Lbz 2000 -ys -200 -ye 0 \
-regrid_file g20km_10ka_hy.nc -regrid_vars litho_temp,thk,enthalpy,tillwat,bmelt ...
```

Instead of a 20 km grid in the horizontal (`-Mx 76 -My 141`) we ask for a 5 km grid (`-Mx 301 -My 561`). Instead of vertical grid resolution of 40 m (`-Mz 101 -z_spacing equal -Lz 4000`) we ask for a vertical resolution of 20 m (`-Mz 201 -z_spacing equal -Lz 4000`).¹ Most significantly, however, we say `-regrid_file g20km_10ka_hy.nc` to regrid — specifically, to bilinearly-interpolate — fields from a model result computed on the coarser 20 km grid. The regridded fields (`-regrid_vars litho_temp,...`) are the evolving mass and energy state variables which are already approximately at equilibrium on the coarse grid. Because we are bootstrapping (i.e. using the `-bootstrap` option), the other variables, especially the bedrock topography `topg` and the climate data, are brought in to PISM at “full” resolution, that is, on the original 5 km grid in the data file `pism_Greenland_5km_v1.1.nc`.

This technique could be called “grid sequencing”.² The result of the above command will be to compute the near-equilibrium result on the fine 5 km grid, taking advantage of the coarse-gridded computation of approximate equilibrium, and despite a run of only 200 model years (`-ys -200 -ye 0`). How close to equilibrium we get depends on

¹ See subsections *Bootstrapping*, *Computational box*, and *Spatial grid* for more about determining the computation domain and grid at bootstrapping.

² It is not quite “multigrid.” That would both involve refinement and coarsening stages in computing the fine grid solution.

both durations, i.e. on both the coarse and fine grid run durations, but certainly the computational effort is reduced by doing a short run on the fine grid. Note that in the previous subsection we also used regridding. In that application, however, `-regrid_file` only “brings in” fields from a run on the same resolution.

Generally the fine grid run duration in grid sequencing should be at least $t = \Delta x / v_{\min}$ where Δx is the fine grid resolution and v_{\min} is the lowest ice flow speed that we expect to be relevant to our modeling purposes. That is, the duration should be such that slow ice at least has a chance to cross one grid cell. In this case, if $\Delta x = 5$ km and $v_{\min} = 25$ m/year then we get $t = 200$ a. Though we use this as the duration, it is a bit short, and the reader might compare $t = 500$ results (i.e. using $v_{\min} = 10$ m/year).

Actually we will demonstrate how to go from 20 km to 5 km in two steps, $20 \text{ km} \rightarrow 10 \text{ km} \rightarrow 5 \text{ km}$, with durations of 10 ka, 2 ka, and 200 a, respectively. The 20 km coarse grid run is already done; the result is in `g20km_10ka_hy.nc`. So we run the following script which is `gridseq.sh` in `examples/std-greenland/`. It calls `spinup.sh` to collect all the right PISM options:

```
#!/bin/bash
NN=4
export PARAM_PPQ=0.5
export REGRIDFILE=g20km_10ka_hy.nc
export EXSTEP=100
./spinup.sh $NN const 2000 10 hybrid g10km_gridseq.nc
export REGRIDFILE=g10km_gridseq.nc
export EXSTEP=10
./spinup.sh $NN const 200 5 hybrid g5km_gridseq.nc
```

Environment variable `EXSTEP` specifies the time in years between writing the spatially-dependent, and large-file-size-generating, frames for the `-extra_file ...` diagnostic output.

Warning: The 5 km run requires 8 Gb of memory at minimum!

If you try it without at least 8 Gb of memory then your machine will “bog down” and start using the hard disk for swap space! The run will not complete and your hard disk will get a lot of wear! (If you have less than 8 Gb memory, comment out the last three lines of the above script by putting the “#” character at the beginning of the line so that you only do the $20 \text{ km} \rightarrow 10 \text{ km}$ refinement.)

Run the script like this:

```
./gridseq.sh &> out.gridseq &
```

The 10 km run takes under two wall-clock hours (8 processor-hours) and the 5 km run takes about 6 wall-clock hours (24 processor-hours).

Fig. 3.10, showing only a detail of the western coast of Greenland, with several outlet glaciers visible, suggests what is accomplished: the high resolution runs have separated outlet glacier flows, as they are in reality. Note that all of these results were generated in a few wall clock hours on a laptop! The surface speed `velsurf_mag` from files `g10km_gridseq.nc` and `g5km_gridseq.nc` is shown (two right-most subfigures). In the two left-hand subfigures we show the same field from NetCDF files `g40km_10ka_hy.nc` and `g20km_10ka_hy.nc`; the former is an added 40 km result using an obvious modification of the run in section *Second run: a better ice-dynamics model*.

Fig. 3.11, which shows time series of ice volume, also shows the cost of high resolution, however. The short 200 a run on the 5 km grid took about 3 wall-clock hours compared to the 10 minutes taken by the 10 ka run on a 20 km grid. The fact that the time series for ice volume on 10 km and 5 km grids are not very “steady” also suggests that these runs should actually be longer.

In this vein, if you have an available supercomputer then a good exercise is to extend our grid sequencing example to 3 km or 2 km resolutions [55]; these grids are already supported in the script `spinup.sh`. Note that the vertical grid also generally gets refined as the horizontal grid is refined.

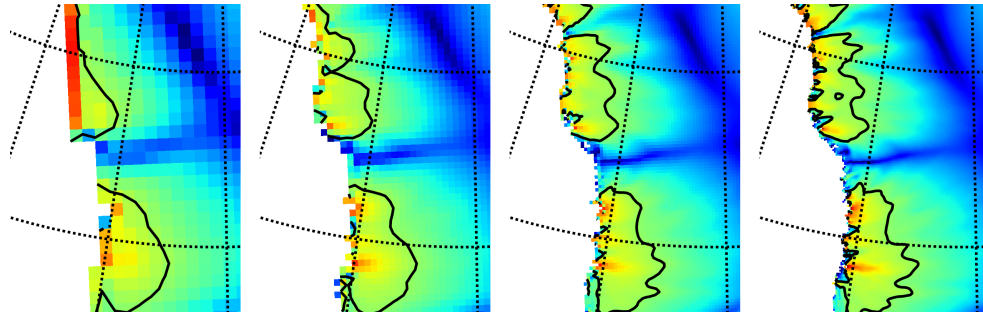


Fig. 3.10: Detail of field `velsurf_mag` showing the central western coast of Greenland, including Jakobshavn Isbrae (lowest major flow), from runs of resolution 40, 20, 10, 5 km (left-to-right). Color scheme and scale, including 100 m/year contour (solid black), are all identical to `velsurf_mag` Figures Fig. 3.4, Fig. 3.5, and Fig. 3.6.

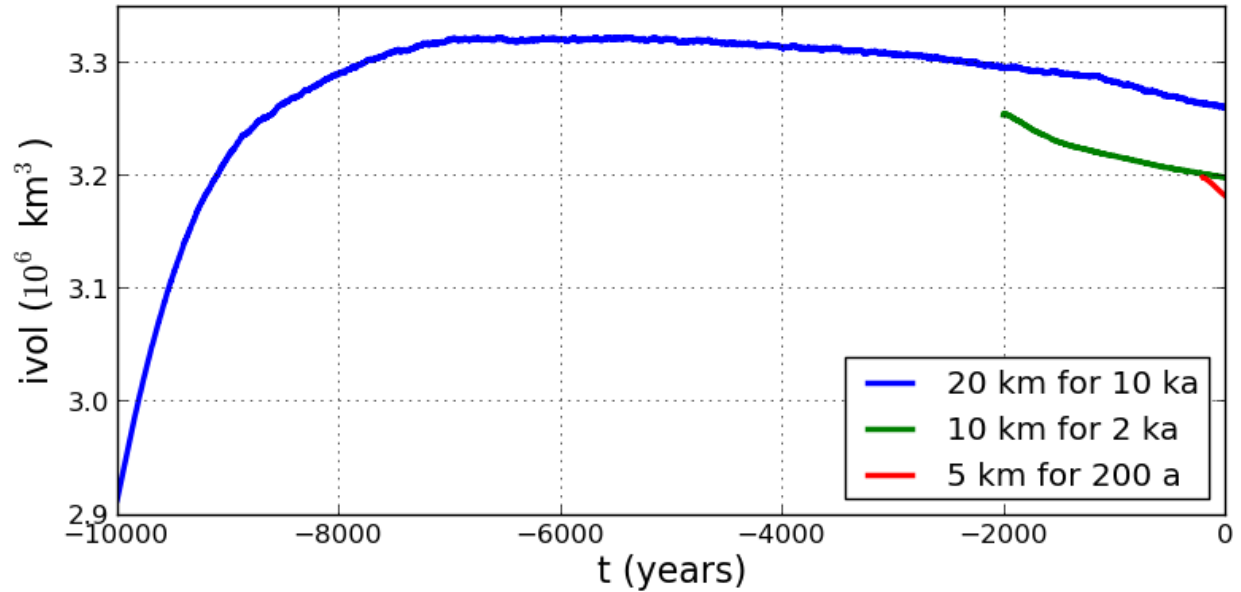


Fig. 3.11: Time series of ice volume `ice_volume_glacierized` from the three runs in our grid sequencing example: 20 km for 10 ka = `ts_g20km_10ka_hy.nc`, 10 km for 2 ka = `ts_g10km_gridseq.nc`, and 5 km for 200 a = `ts_g5km_gridseq.nc`.

Going to a 1km grid is possible, but you will start to see the limitations of distributed file systems in writing the enormous NetCDF files in question [111]. Notice that a factor-of-five refinement in all three dimensions, e.g. from 5 km to 1 km in the horizontal, and from 20 m to 4 m in the vertical, generates an output NetCDF file which is 125 times larger. Since the already-generated 5 km result `g5km_gridseq.nc` is over 0.5 Gb, the result is a very large file at 1 km.

On the other hand, on fine grids we observe that *memory* parallelism, i.e. spreading the stored model state over the separated memory of many nodes of supercomputers, is as important as the usual *computation* (CPU) parallelism.

This subsection has emphasized the “P” in PISM, the nontrivial parallelism in which the solution of the conservation equations, especially the stress balance equations, is distributed across processors. An easier and more common mode of parallelism is to distribute distinct model runs, each with different parameter values, among the processors. For scientific purposes, such parameter studies, whether parallel or not, are at least as valuable as individual high-resolution runs.

3.1.8 An ice dynamics parameter study

The readers of this manual should not assume the PISM authors know all the correct parameters for describing ice flow. While PISM must have *default* values of all parameters, to help users get started,¹ it has more than two hundred user-configurable parameters. The goal in this manual is to help the reader adjust them to their desired values. While “correct” values may never be known, or may not exist, examining the behavior of the model as it depends on parameters is both a nontrivial and an essential task.

For some parameters used by PISM, changing their values within their ranges of experimental uncertainty is unlikely to affect model results in any important manner (e.g. `constants.sea_water.density`). For others, however, for instance for the exponent in the basal sliding law, changing the value is highly-significant to model results, as we’ll see in this subsection. This is also a parameter which is very uncertain given current glaciological understanding [67].

To illustrate a parameter study in this Manual we restrict consideration to just two important parameters for ice dynamics,

- q = `basal_resistance.pseudo_plastic.q`: exponent used in the sliding law which relates basal sliding velocity to basal shear stress in the SSA stress balance; see subsection *Controlling basal strength* for more on this parameter, and
- e = `stress_balance.sia.enhancement_factor`: values larger than one give flow “enhancement” by making the ice deform more easily in shear than is determined by the standard flow law [64], [62]; applied only in the SIA stress balance; see section *Ice rheology* for more on this parameter.

By varying these parameters over full intervals of values, say $0.1 \leq q \leq 1.0$ and $1 \leq e \leq 6$, we could explore a two-dimensional parameter space. But of course each (q, e) pair needs a full computation, so we can only sample this two-dimensional space. Furthermore we must specify a concrete run for each parameter pair. In this case we choose to run for 1000 model years, in every case initializing from the stored state `g10km_gridseq.nc` generated in the previous section *Grid sequencing*.

The next script, which is `param.sh` in `examples/std-greenland/`, gets values $q \in \{0.1, 0.5, 1.0\}$ and $e \in \{1, 3, 6\}$ in a double `for`-loop. It generates a run-script for each (q, e) pair. For each parameter setting it calls `spinup.sh`, with the environment variable `PISM_D0=echo` so that `spinup.sh` simply outputs the run command. This run command is then redirected into an appropriately-named `.sh` script file:

```
#!/bin/bash

NN=4
DURATION=1000
START=g10km_gridseq.nc
for PPQ in 0.1 0.5 1.0 ; do
```

¹ See *Configuration parameters* for the full list.

```
for SIAE in 1 3 6 ; do
  export PISM_DO=echo REGRIDFILE=$START PARAM_PPQ=$PPQ PARAM_SIAE=$SIAE
  ./spinup.sh $NN const $DURATION 10 hybrid p10km_${PPQ}_${SIAE}.nc &> p10km_${PPQ}_${SIAE}.sh
done
done
```

Notice that, because the stored state `g10km_gridseq.nc` used $q = 0.5$ and $e = 3$, one of these runs simply continues with no change in the physics.

To set up and run the parameter study, without making a mess from all the generated files, do:

```
cd examples/std-greenland/          # g10km_gridseq.nc should be in this directory
mkdir paramstudy
cd paramstudy
ln -s ../g10km_gridseq.nc          # these four lines make links to ...
ln -s ../pism_Greenland_5km_v1.1.nc #
ln -s ../spinup.sh                 #
ln -s ../param.sh                  # ... existing files in examples/std-greenland/
./param.sh
```

The result of the last command is to generate nine run scripts,

```
p10km_0.1_1.sh p10km_0.1_3.sh p10km_0.1_6.sh
p10km_0.5_1.sh p10km_0.5_3.sh p10km_0.5_6.sh
p10km_1.0_1.sh p10km_1.0_3.sh p10km_1.0_6.sh
```

The reader should inspect a few of these scripts. They are all very similar, of course, but, for instance, the `p10km_0.1_1.sh` script uses options `-pseudo_plastic_q 0.1` and `-sia_e 1`.

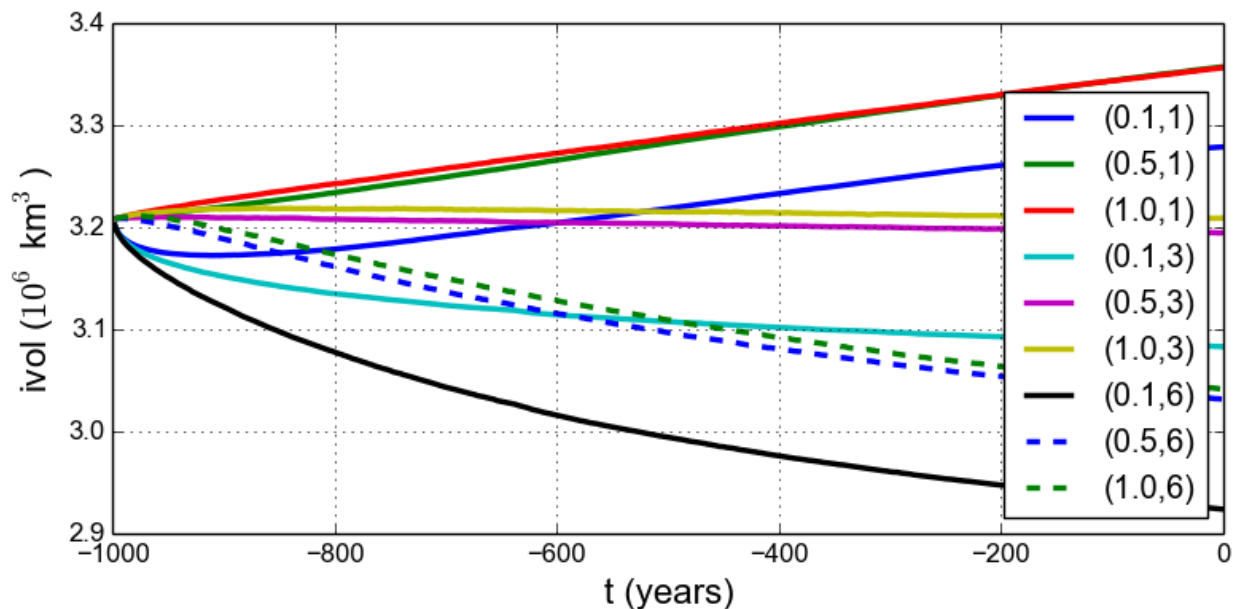


Fig. 3.12: Time series of ice volume `ice_volume_glacierized` from nine runs in our parameter study example, with parameter choices (q, e) given.

We have not yet run PISM, but only asked one script to create nine others. We now have the option of running them sequentially or in parallel. Each script itself does a parallel run, over the $NN=4$ processes specified by `param.sh` when

generating the run scripts. If you have $4 \times 9 = 36$ cores available then you can do the runs fully in parallel (this is `runparallel.sh` in `examples/std-greenland/`):

```
#!/bin/bash

for scriptname in $(ls p10km*sh) ; do
    echo ; echo "starting ${scriptname} ..."
    bash $scriptname &> out.${scriptname} & # start immediately in background
done
```

Otherwise you should do them in sequence (this is `runsequential.sh` in `examples/std-greenland/`):

```
#!/bin/bash

for scriptname in $(ls p10km*sh) ; do
    echo ; echo "starting ${scriptname} ..."
    bash $scriptname # will wait for completion
done
```

On the same old 2012-era 4 core laptop, `runsequential.sh` took a total of just under 7 hours to complete the whole parameter study. The runs with $q = 0.1$ (the more “plastic” end of the basal sliding spectrum) took up to four times longer than the $q = 0.5$ and $q = 1.0$ runs. Roughly speaking, values of q which are close to zero imply a subglacial till model with a true yield stress, and the result is that even small changes in overall ice sheet state (geometry, energy, ...) will cause *some* location to exceed its yield stress and suddenly change flow regime. This will shorten the time steps. By contrast, the e value is much less significant in determining run times.

On a supercomputer, the `runparallel.sh` script generally should be modified to submit jobs to the scheduler. See example scripts `advanced/paramspawn.sh` and `advanced/paramsubmit.sh` for a parameter study that does this. (But see your system administrator if you don’t know what a “job scheduler” is!) Of course, if you have a supercomputer then you can redo this parameter study on a 5 km grid.

Results from these runs are seen in [Fig. 3.12](#) and [Fig. 3.13](#). In the former we see that the (0.5, 3) run simply continues the previous initialization run. In some other graphs we see abrupt initial changes, caused by abrupt parameter change, e.g. when the basal sliding becomes much more plastic ($q = 0.1$). In all cases with $e = 1$ the flow slows and the sheet grows in volume as discharge decreases, while in all cases with $e = 6$ the flow accelerates and the sheet shrinks in volume as discharge increases.

In [Fig. 3.13](#) we can compare the surface speed model results to observations. Roughly speaking, the ice softness parameter e has effects seen most-clearly by comparing the interior of the ice sheet; scan left-to-right for the $e = 1, 3, 6$ subfigures. The basal sliding exponent q has effects seen most-clearly by comparing flow along the very steep margin, especially in the southern half of the ice sheet; scan top-to-bottom for $q = 0.1, 0.5, 1.0$, going from nearly-plastic at top to linear at bottom.

From such figures we can make an informal assessment and comparison of the results, but objective assessment is important. Example objective functionals include:

1. compute the integral of the square (or other power) of the difference between the model and observed surface velocity [\[55\]](#), or
2. compute the model-observed differences between the histogram of the number of cells with a given surface speed [\[114\]](#).

Note that these functionals are measuring the effects of changing a small number of parameters, namely two parameters in the current study. So-called “inversion” might use the same objective functionals but with a much larger parameter space. Inversion is therefore capable of achieving much smaller objective measures [\[56\]](#), [\[115\]](#), [\[116\]](#), though at the cost of less understanding, perhaps, of the meaning of the optimal parameter values.

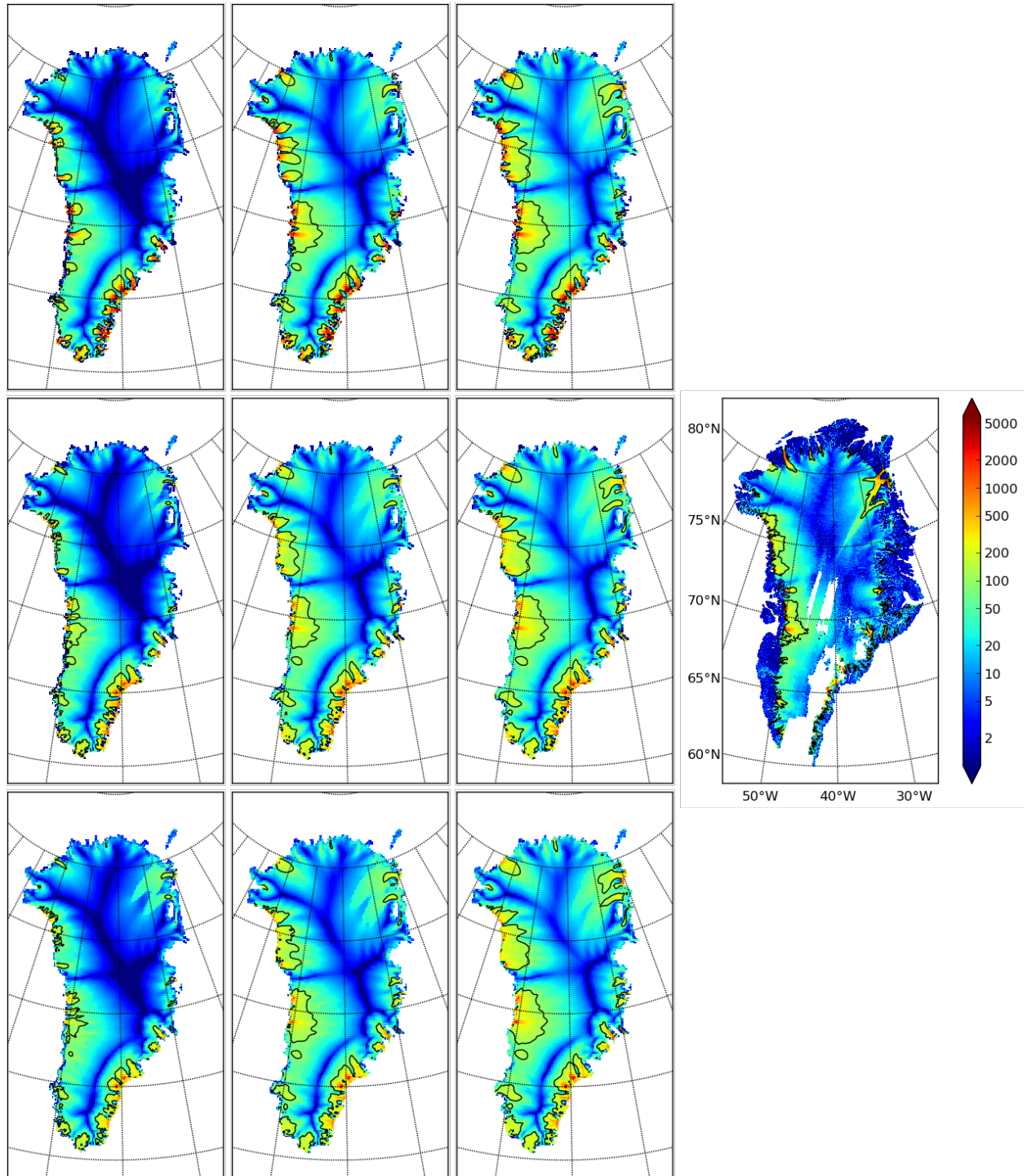


Fig. 3.13: Surface speed `velsurf_mag` from a 10 km grid parameter study. Right-most subfigure is observed data from `Greenland_5km_v1.1.nc`. Top row: $q = 0.1$ and $e = 1, 3, 6$ (left-to-right). Middle row: $q = 0.5$. Bottom row: $q = 1.0$. All subfigures have common color scale (velocity m/year), as shown in the right-most figure, with 100 m/year contour shown in all cases (solid black).

3.2 Ice dynamics, the PISM view

This section describes PISM’s high-level view of ice dynamics and the separation of ice dynamics from climate inputs.

Please see *Modeling choices* so learn how to control PISM’s sub-models.

3.2.1 Two stress balance models: SIA and SSA

At each time-step of a typical PISM run, the geometry, temperature, and basal strength of the ice sheet are included into stress (momentum) balance equations to determine the velocity of the flowing ice. The “full” stress balance equations for flowing ice form a non-Newtonian Stokes model [19]. PISM does not attempt to solve the Stokes equations themselves, however. Instead it can numerically solve, in parallel, two different shallow approximations which are well-suited to ice sheet and ice shelf systems:

- the non-sliding shallow ice approximation (SIA) [29], also called the “lubrication approximation” [19], which describes ice as flowing by shear in planes parallel to the geoid, with a strong connection of the ice base to the bedrock, and
- the shallow shelf approximation (SSA) [30], which describes a membrane-type flow of floating ice [31], or of grounded ice which is sliding over a weak base [32], [33].

The SIA equations are easier to solve numerically than the SSA, and easier to parallelize, because they are local in each column of ice. Specifically, they describe the vertical shear stress as a local function of the driving stress [34]. They can confidently be applied to those grounded parts of ice sheets for which the basal ice is frozen to the bedrock, or which is minimally sliding, and where the bed topography is relatively slowly-varying in the map-plane [19]. These characteristics apply to the majority (by area) of the Greenland and Antarctic ice sheets.

We solve the SIA with a non-sliding base because the traditional [24], [35], [36] addition of ad hoc “sliding laws” into the SIA stress balance, and especially schemes which “switch on” at the pressure-melting temperature [14], have bad continuum [37] and numerical (see [17], appendix B) modeling consequences.

The SSA equations can confidently be applied to large floating ice shelves, which have small depth-to-width ratio and negligible basal resistance [31], [38]. The flow speeds in ice shelves are frequently an order-of-magnitude higher than in the non-sliding, grounded parts of ice sheets.

Terrestrial ice sheets also have fast-flowing grounded parts, however, called “ice streams” or “outlet glaciers” [39]. Such features appear at the margin of, and sometimes well into the interior of, the Greenland [40] and Antarctic [41] ice sheets. Describing these faster-flowing grounded parts of ice sheets requires something more than the non-sliding SIA. This is because adjacent columns of ice which have different amounts of basal resistance exert strong “longitudinal” or “membrane” stresses [33] on each other.

In PISM the SSA may be used as a “sliding law” for grounded ice which is already modeled everywhere by the non-sliding SIA [17], [25]. For grounded ice, in addition to including shear in planes parallel to the geoid, we must balance the membrane stresses where there is sliding. This inclusion of a membrane stress balance is especially important when there are spatial and/or temporal changes in basal strength. This “sliding law” role for the SSA is in addition to its more obvious role in ice shelf modeling. The SSA plays both roles in a PISM whole ice sheet model in which there are large floating ice shelves (e.g. as in Antarctica [42], [6], [25]; see also *An SSA flow model for the Ross Ice Shelf in Antarctica*).

The “SIA+SSA hybrid” model is recommended for most whole ice sheet modeling purposes because it seems to be a good compromise given currently-available data and computational power. A related hybrid model described by Pollard and deConto [43] adds the shear to the SSA solution in a slightly-different manner, but it confirms the success of the hybrid concept.

By default, however, PISM does not turn on (activate) the SSA solver. This is because a decision to solve the SSA must go with a conscious user choice about basal strength. The user must both use a command-line option to turn on the SSA (e.g. option `-stress_balance ssa`; see section *Choosing the stress balance*) and also make choices in

input files and runtime options about basal strength (see section *Controlling basal strength*). Indeed, uncertainties in basal strength boundary conditions usually dominate the modeling error made by not including higher-order stresses in the balance.

When the SSA model is applied a parameterized sliding relation must be chosen. A well-known SSA model with a linear basal resistance relation is the Siple Coast (Antarctica) ice stream model by MacAyeal [32]. The linear sliding law choice is explained by supposing the saturated till is a linearly-viscous fluid. A free boundary problem with the same SSA balance equations but a different sliding law is the Schoof [33] model of ice streams, using a plastic (Coulomb) sliding relation. In this model ice streams appear where there is “till failure” [34], i.e. where the basal shear stress exceeds the yield stress. In this model the location of ice streams is not imposed in advance.

As noted, both the SIA and SSA models are *shallow* approximations. These equations are derived from the Stokes equations by distinct small-parameter arguments, both based on a small depth-to-width ratio for the ice sheet. For the small-parameter argument in the SIA case see [19]. For the corresponding SSA argument, see [30] or the appendices of [33]. Schoof and Hindmarsh [44] have analyzed the connections between these shallowest models and higher-order models, while [45] discusses ice dynamics and stress balances comprehensively. Note that SIA, SSA, and higher-order models all approximate the pressure as hydrostatic.

Instead of a SIA+SSA hybrid model as in PISM, one might use the Stokes equations, or a “higher-order” model (i.e. less-shallow approximations [26], [27]), but this immediately leads to a resolution-versus-stress-inclusion tradeoff. The amount of computation per map-plane grid location is much higher in higher-order models, although careful numerical analysis can generate large performance improvements for such equations [46].

Time-stepping solutions of the mass conservation and energy conservation equations, which use the ice velocity for advection, can use any of the SIA or SSA or SIA+SSA hybrid stress balances. No user action is required to turn on these conservation models. They can be turned off by user options `-no_mass` (ice geometry does not evolve) or `-energy none` (ice enthalpy and temperature does not evolve), respectively.

3.2.2 A hierarchy of simplifying assumptions for grounded ice flow

Table 3.1 describes a hierarchy of models, listed roughly in order of increasing effectiveness in modeling grounded ice sheets with fast flow features. This is also the order of increasing need for data to serve as boundary and initial conditions, however, as also described in the Table.

Table 3.1: Hierarchy of flow models in PISM for the grounded parts of ice sheets. Listed from most to fewest simplifying assumptions *and* from least to greatest need for boundary data. The *italicized* models are planned for future versions of PISM but are not implemented so far.

Model	Assumptions	Required data
<i>perfectly-plastic ice</i>	<i>steady state</i> ; ice has shear stresses below a pre-determined ice “yield stress”; also needs pre-determined location of ice sheet margin	<ul style="list-style-type: none"> • bed elevation
<i>balance velocities</i>	<i>steady state</i> ; ice flows down surface gradient [20]	<i>same as above, plus:</i> <ul style="list-style-type: none"> • surface mass balance • (initial) ice thickness
isothermal SIA	non-sliding lubrication flow, fixed softness [21], [22]	<i>same as above, but time-dependence is allowed</i>
thermo-coupled SIA	non-sliding lubrication flow, temperature-dependent softness [18], [14]	<i>same as above, plus:</i> <ul style="list-style-type: none"> • surface temperature • geothermal flux
polythermal SIA	allows liquid water fraction in temperate ice; conserves energy better [23], [24]	<i>same as above</i>
SIA + SSA hybrid	SSA as a sliding law for thermo-coupled SIA [17], [25]; polythermal by default	<i>same as above, plus:</i> <ul style="list-style-type: none"> • model for subglacial water • model for basal resistance
<i>Blatter-Pattyn</i>	“higher-order”, bridging stresses [26], [27], [28]	<i>same as above</i>

It may also be helpful to view the implemented stress balances as PISM software components (C++ classes). Fig. 3.14 shows the sequences of actions taken by the SIA-only, SSA-only, and SIA+SSA hybrid model components. In each case a membrane stress solution is generated first, then a distribution of vertical shear in the column of ice is generated second, and finally a use of incompressibility computes the vertical component of the velocity. The nonsliding SIA-only model has a trivialized membrane stress solution. The SSA-only model has a trivialized computation of vertical shear.

3.2.3 Evolutionary versus diagnostic modeling

The main goal of a numerical ice sheet model like PISM is to be a dynamical system which evolves as similarly as possible to the modeled ice sheet. Such a goal assumes one has the “right” climate inputs and parameter choices at each time step. It also assumes one has the “right” initial conditions, such as an adequate description of the present state of the ice sheet, but this assumption is rarely satisfied. Instead a variety of heuristics must be used to minimally-initialize an ice sheet model. For options associated to establishing mathematical initial conditions when first starting PISM, see section *Initialization and bootstrapping*.

Inside PISM are evolution-in-time partial differential equations which are solved by taking small time steps. “Small” may vary from thousandths to tens of model years, in practice, depending primarily on grid resolution, but also on modeled ice geometry and flow speed. Time steps are chosen adaptively in PISM, according to the stability criteria of the combined numerical methods [17], [18].

However, especially for ice streams and shelves, non-time-stepping “diagnostic” solution of the stress balance partial differential equations might be the desired computation, and PISM can also produce such “diagnostic” velocity fields. Such computations necessarily assume that the ice geometry, viscosity, and boundary stresses are known. Because of

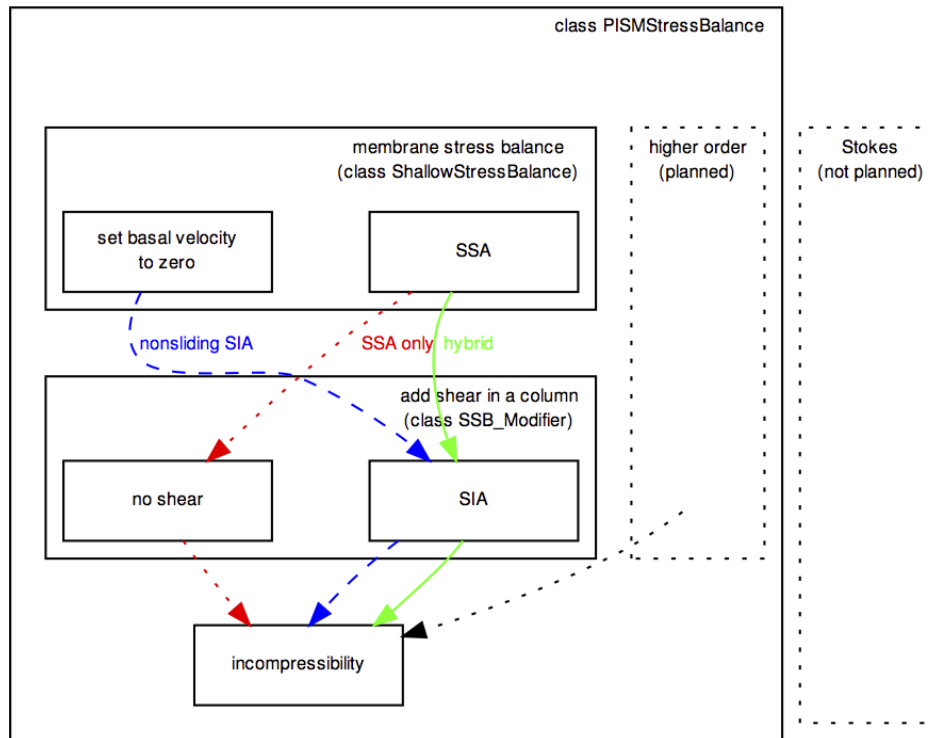


Fig. 3.14: The SIA-only, SSA-only, and SIA+SSA hybrid models represent different “routes” through stress balance PISM components. In each case the inputs are ice geometry and boundary stresses, and the final output is a three-dimensional velocity field within the ice.

the slowness of the ice, in the sense that inertia can be neglected in the stress balance [19], such computations can determine the ice velocity.

Sections *Getting started: a Greenland ice sheet example* and *An SSA flow model for the Ross Ice Shelf in Antarctica* give examples illustrating evolutionary and diagnostic modes of PISM, respectively. The first describes time-stepping evolution models for the Greenland ice sheet, while the second describes a diagnostic SSA model for the Ross ice shelf.

3.2.4 Climate inputs, and their interface with ice dynamics

Because PISM’s job is to approximate ice flow, its “world view” is centered around ice dynamics. The discussion of boundary conditions in this Manual is thus ice-dynamics-centric. On the other hand, there is no constraint on the nature of, or completeness of, climate models which could be coupled to PISM. This section therefore explains a PISM organizing principle, namely that *climate inputs affect ice dynamics by a well-defined interface*.

Almost no attempt is made here to describe the physics of the climate around ice sheets, so see [16] for terminology and [15] for a review of how surface melt can be modeled. See the *Climate Forcing Manual* for much more information on PISM’s climate-coupling-related options and on the particular fields which are shared between the ice dynamics core and the climate model. Table 3.2 lists fields which are needed as boundary conditions at the interfaces.

All PISM ice sheet models have some kind of interface green in Fig. 3.15) to a subaerial surface processes layer containing snow, firn, and liquid (or refrozen) runoff. The surface layer is assumed to cover the whole surface of the ice, and all grounded areas that the ice might occupy, including ablation areas and ice-free land. We also always have an interface (blue) to the ocean, but this interface is inactive if there is no floating ice.

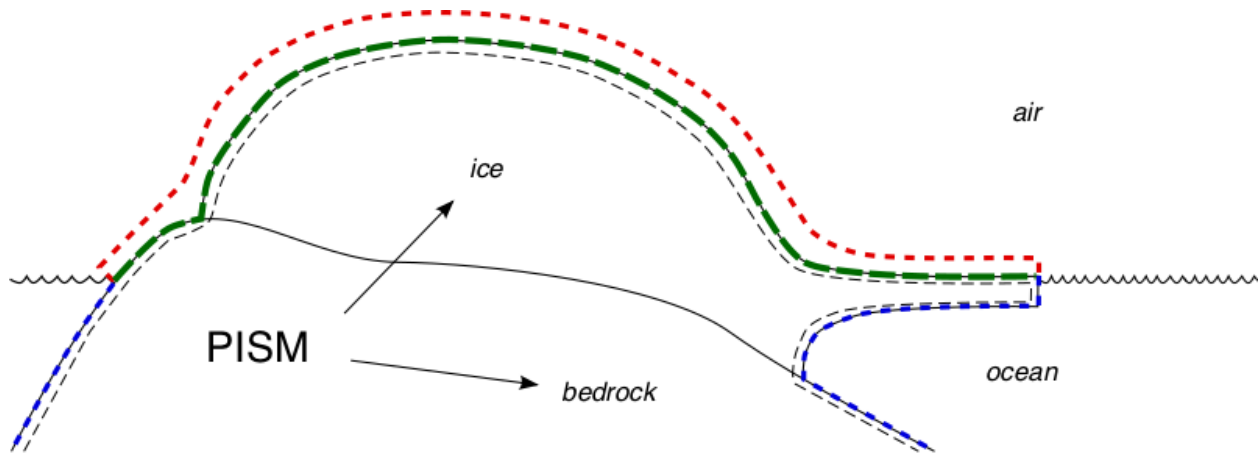


Fig. 3.15: PISM’s view of interfaces between an ice sheet and the outside world

Table 3.2: Boundary conditions required by PISM’s ice dynamics core; see Fig. 3.15. The optional red interface is absent if PISM does not “own” the surface processes layer.

Boundary surface	Fields (conditions)
upper surface of the surface processes layer (red)	<i>optional</i> ; typically: air temperature, precipitation
top ice surface, but below firn (green)	<i>required</i> : boundary temperature (or enthalpy), mass flux (SMB) into the ice
ice shelf basal surface (blue)	<i>required</i> : mass flux into the ocean, boundary temperature
bottom surface of thermally-modeled bedrock layer (not shown)	<i>required</i> : geothermal flux

The surface processes layer might be very simple. It might either read the important fields from a file or otherwise transfer them from a separate (non-PISM) climate model. If, however, the surface processes layer is “owned” by the PISM model then there is an additional interface (red) to the atmosphere above. In no case does PISM “own” the atmosphere; if it has an interface to the atmosphere at all then it reads atmosphere fields from a file or otherwise transfers them from a climate model.

Regarding the base of the ice, the temperature of a layer of bedrock in contact with grounded ice is generally included in PISM’s conservation of energy model; see subsections *Computational box* and *Spatial grid*. Also, as described in section *Earth deformation models*, PISM can apply an optional bed deformation component approximating the movement of the Earth’s crust and upper mantle in response to changing ice load. In these senses everything below the black dashed line in Fig. 3.15 is always “owned” by PISM.

The PISM ice dynamics core would like to get the required fields listed in Table 3.2 directly from observations or measurements, or directly from a GCM. In many realistic modeling situations, however, PISM code must be used for all or part of the surface processes modeling necessary to provide the ice-dynamics core with the needed fields. Due to differences in model resolutions and required down-scaling, this need for some PISM-based boundary-processes modelling may occur even in some cases where PISM is coupled to a GCM. Thus, to be able to use the data that is available, a PISM run might use components that are responsible for modeling surface (snow) processes or sub-shelf/ocean interaction. These components might be very minimal, merely turning data that we already have into data in the right units and with the right metadata.

Thus we have PISM’s design: the ice-dynamics PISM core does not contain any parameterization or other model for boundary mass or energy fluxes into or out of the ice. These boundary parameterizations and models are present in the PISM source code, however, as instances of `pism::Component` classes. This simplifies customizing and debugging PISM’s climate inputs, and it promotes code reuse. It isolates the code that needs to be changed to couple PISM to different climate models.

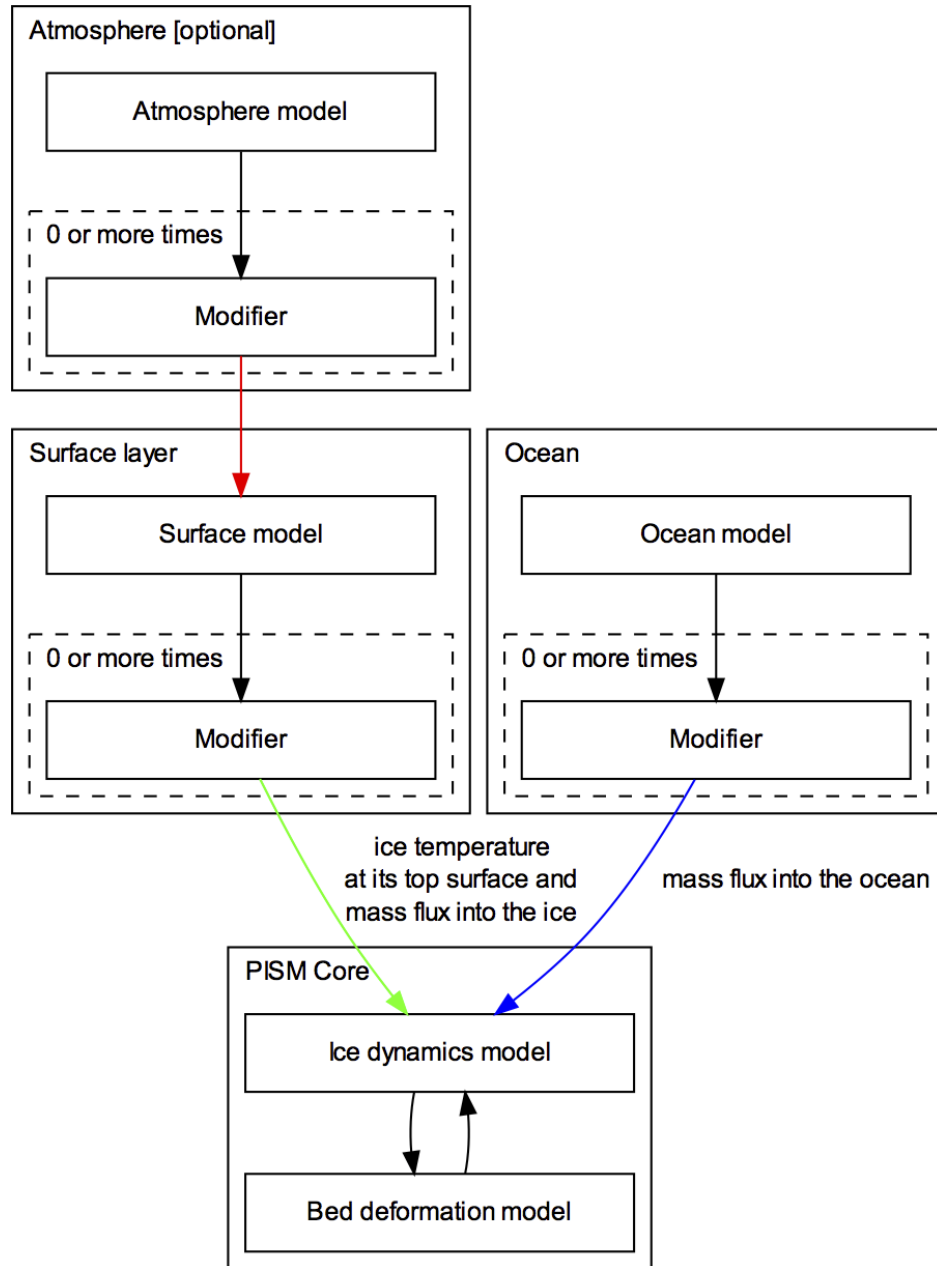


Fig. 3.16: PISM climate input data flow. Colored arrows correspond to interfaces in Fig. 3.15.

The classes `pism::SurfaceModel`, `pism::AtmosphereModel`, and `pism::OceanModel` are all derived from `pism::Component`. Corresponding to the red dashed line in Fig. 3.15, a `pism::AtmosphereModel` might not even be present in some PISM configurations. While they are required, `pism::SurfaceModel` and `pism::OceanModel` may contain (hide) anything from nearly-trivial parameterizations of ice surface temperatures and mass fluxes to a GCM of great complexity.

The “modifiers” in Fig. 3.16 adjust the climate model inputs. Modifiers can be chained together so that multiple modifications are made to the outputs of the original component. For example, ice-core-derived air temperature offsets, used to model the space-time distribution of paleo-climatic surface temperature, is an example of an implemented modifier. Please see the *Climate Forcing Manual* for a list of climate components and modifiers included in PISM source code and other details. Users wishing to customize PISM’s climate inputs and/or couple PISM to a climate model should additionally see the [PISM Source Browser](#) and the documentation therein.

Fig. 3.16 illustrates the data flow needed by the ice dynamics core. The data flow in the other direction, i.e. needed by the model to which PISM is coupled, depends on particular modeling choices, but great flexibility is allowed.

Why describe all this structure here? On the one hand, some users may be interested in coupling PISM to other models. On the other hand, the PISM authors do not claim expertise in modeling atmosphere, ocean, or even snow processes. This separation has a definite code-reliability purpose. PISM users are ultimately responsible for providing the climate inputs they intend.

3.3 Initialization and bootstrapping

There are three ways to start PISM:

- option `-i` reads a previously-saved “complete” PISM model state in a NetCDF file, or
- option `-i -bootstrap` reads an “incomplete” NetCDF file and uses heuristics to fill in needed fields, or
- one of the executables `pisms` or `pismv` is used to initialize simplified-geometry experiments or verification tests from formulas in the source code, and thus no input file is required.

One of the first two choices is required when using the executable `pismr`. Modeling usually starts with the `-i input.nc -bootstrap` option because real ice sheet observations are never complete initial conditions. Runs with multiple stages often use the `-i` option after the first stage.

3.3.1 Initialization from a saved model state

“Initialization” has the specific, simple meaning in PISM that option “`-i`” was used. If a previous PISM run has saved a NetCDF file using “`-o`” then that file will contain complete initial conditions for continuing the run. The output file from the last run can be loaded with “`-i`”:

```
pisms -eisII A -y 100 -o foo.nc
pisms -eisII A -i foo.nc -y 100 -o bar.nc
```

As noted verification tests (section *Verification*) and simplified-geometry experiments (section *Simplified geometry experiments*) do not need input files at all because they initialize from formulas in the source code. They can, however, be continued from saved model states using `-i`. Specifying the simplified geometry experiment or verification test *is*, however, necessary if the run is to continue with the climate inputs for that experiment or test. For example, based on the above `pisms` runs, it is valid to do

```
pismr -i foo.nc -y 100 -o bar.nc
```

but the climate and other parameters use PISM default values, and thus are not (necessarily) the values specified in EISMINT II.

As a technical note about saved states, a PISM run with `-stress_balance ssa` also saves the last SSA velocities to the output file in variables `u_ssa` and `v_ssa`. The presence of these velocities adds efficiency in restarting because an initial estimate speeds up the solution of the SSA stress balance equations. If you want to use `-i` but also ignore these velocities then use option `-dontreadSSAvels`.

-i file format

PISM produces CF-1.5 compliant NetCDF files. The easiest way to learn the output format *and* the `-i` format is to do a simple run and then look at the metadata in the resulting file, like this:

```
pisms -eisII A -y 10 -o foo.nc
ncdump -h foo.nc | less
```

Note that variables in the output file have a `pism_intent` attribute. When `pism_intent` is `diagnostic`, the variable can be deleted from the file without affecting whether PISM can use it as a `-i` input file. Variables with `pism_intent` is `model_state`, by contrast, must be present when using `-i`.

The automatically-produced time variable has a `units` attribute like "seconds since 1-1-1" because the CF metadata conventions require a reference date.

By default PISM ignores this reference date except when it is used in unit conversions based on a calendar (see below).

3.3.2 Bootstrapping

“Bootstrapping” in PISM means starting a modeling run with less than sufficient data, and letting essentially heuristic models fill in needed fields. These heuristics are applied before the first time step is taken, so they are part of an initialization process. Bootstrapping uses the option `-bootstrap`; see section [First run](#) for an example.

The need for an identified stage like “bootstrapping” comes from the fact that initial conditions for the evolution equations describing an ice sheet are not all observable. As a principal example of this problem, these initial conditions include the temperature within the ice. Glaciological observations, specifically remote-sensed observations which cover a large fraction or all of an ice sheet, never include this temperature field in practice.

Ice sheet models often need to do something like this to get “reasonable” initial fields within the ice:

1. start only with (potentially) observable quantities like surface elevation, ice thickness, ice surface temperature, surface mass balance, and geothermal flux,
2. “bootstrap” as defined here, using heuristics to fill in temperatures at depth and to give a preliminary estimate of the basal sliding condition and the three-dimensional velocity field, and
3. (a) *either* do a long run, often holding the current geometry and surface conditions steady, to evolve toward a steady state which has compatible temperature, stress, and velocity fields,
(b) *or* do a long run using an additional (typically spatially-imprecise) historical record from an ice core or a sea bed core (or both), to apply forcing to the surface temperature or sea level (for instance), but with the same functional result of filling in temperature, stress, and velocity fields.

When using `-bootstrap` you will need to specify both grid dimensions (using `-Mx`, `-My` and `-Mz`; see section [Spatial grid](#)) and the height of the computational box for the ice with `-Lz` (section [Computational box](#)). The data read from the file can determine the horizontal extent of the model, if options `-Lx`, `-Ly` are not set. The additional required specification of vertical extent by `-Lz` is reasonably natural because typical data used in “bootstrapping” are two-dimensional. Using `-bootstrap` without specifying all four options `-Mx`, `-My`, `-Mz`, `-Lz` is an error.

If `-Lx` and `-Ly` specify horizontal grid dimensions smaller than in the bootstrapping file, PISM will cut out the center portion of the domain. Alternatively, options `-x_range` and `-y_range` each take a list of two numbers, a list of minimum and maximum x and y coordinates, respectively (in meters), which makes it possible to select a subset that is not centered in the bootstrapping file’s grid.

For the key issue of what heuristic is used to determine the temperatures at depth, there are two methods. The default method uses ice thickness, surface temperature, surface mass balance, and geothermal flux. The temperature is set to the solution of a steady one-dimensional differential equation in which conduction and vertical advection are in balance, and the vertical velocity linearly-interpolates between the surface mass balance rate at the top and zero at the bottom. The non-default method, set with option `-boot_temperature_heuristic quartic_guess`, was the default in older PISM versions (stable0.5 and earlier); it does not use the surface mass balance and instead makes a more-heuristic estimate of the vertical temperature profile based only on the ice thickness, surface temperature, and geothermal flux.

-bootstrap file format

Allowed formats for a bootstrapping file are relatively simple to describe.

1. NetCDF variables should have the `units` containing a [UDUNITS](#)-compatible string. If this attribute is missing, PISM will assume that a field uses MKS units.¹
2. NetCDF coordinate variables should have `standard_name` or `axis` attributes. These are used to determine which *spatial* dimension a NetCDF dimension corresponds to; for example see `ncdump -h` output from a file produced by PISM. The `x` and `y` dimensions need not be called “`x`” and “`y`”.
3. Coordinate variables have to be strictly-increasing.
4. Three-dimensional variables will be ignored in bootstrapping.
5. The `standard_name` attribute is used, when available, to identify a variable, so variable names need not match corresponding variables in a PISM output file. See the [CF standard names used by PISM](#) for a list of CF standard names used in PISM.

Specifically, the bed elevation (topography) is read by `standard_name = bedrock_altitude` and the ice thickness by `standard_name = land_ice_thickness`.

6. Any two-dimensional variable except bed topography and ice thickness may be missing. For missing variables some heuristic will be applied. See [Table 3.1](#) for a sketch of the data necessary for bootstrapping; see `src/base/iMbootstrap.cc` for all further details.
7. Surface elevation is ignored if present. Users with surface elevation and bed elevation data should compute the ice thickness variable, put it in the bootstrapping file, and set its `standard_name` to `land_ice_thickness`.

3.4 Modeling choices

PISM consists of several sub-models corresponding to various physical processes and parameterizations. In short, these are

1. Ice dynamics and thermodynamics (stress balance, ice rheology, mass and energy conservation, ice age)
2. Subglacial processes (hydrology, basal strength, bed deformation)
3. Marine ice-sheet modeling (parameterization of calving processes, calving front advance and retreat, iceberg removal)

All these sub-models are controlled by command-line options and configuration parameters.¹

In additions to this, one has to choose the computational grid, the modeling domain, and so on.

This section describes how to understand and make these modeling choices.

¹ PISM automatically converts data present in an input file to MKS. This means that having ice thickness in feet or temperature in Fahrenheit is allowed.

¹ See [Configuration parameters](#) for the full list of configuration parameters and corresponding options.

3.4.1 Model domain, grid, and time

This section describes PISM’s computational domain and grid and the way to control it.

Computational box

PISM does all simulations in a computational box which is rectangular in the PISM coordinates. The coordinate system has horizontal coordinates x, y and a vertical coordinate z . The z coordinate is measured positive upward from the base of the ice.¹ The vector of gravity is in the negative z direction. The surface $z = 0$ is the base of the ice, however, and thus is usually not horizontal in the sense of being parallel to the geoid. The surface $z = 0$ is the base of the ice both when the ice is grounded and when the ice is floating.

When the ice is grounded, the true physical vertical coordinate z' , namely the coordinate measure relative to a reference geoid, is given by $z' = z + b(x, y)$ where $b(x, y)$ is the bed topography. The surface $z' = h(x, y)$ is the surface of the ice. In the grounded case the equation $h(x, y) = H(x, y) + b(x, y)$ always applies if $H(x, y)$ is the thickness of the ice.

In the floating case, the physical vertical coordinate is

$$z' = z - \frac{\rho_i}{\rho_s} H(x, y)$$

where ρ_i is the density of ice and ρ_s the density of sea water. Again $z = 0$ is the base of the ice, which is the surface

$$z' = -\frac{\rho_i}{\rho_s} H(x, y).$$

The surface of the ice is

$$h(x, y) = \left(1 - \frac{\rho_i}{\rho_s}\right) H(x, y).$$

The *flotation criterion* $-\frac{\rho_i}{\rho_s} H(x, y) > b(x, y)$ applies.

The computational box can extend downward into the bedrock. As $z = 0$ is the base of the ice, the bedrock corresponds to negative z values regardless of its true (i.e. z') elevation.

The extent of the computational box, along with its bedrock extension downward, is determined by four numbers L_x , L_y , L_z , and L_{bz} (see Fig. 3.17 and Table 3.3). The first two of these are half-widths and have units of kilometers when set by command-line options or displayed.

Table 3.3: Options defining the extent of PISM’s computational box

Option	Description
-Lx (km)	Half-width of the computational domain (in the x -direction)
-Ly (km)	Half-width of the computational domain (in the y -direction)
-Lz (meters)	Height of the computational domain; must exceed maximum ice thickness
-Lbz (meters)	Depth of the computational domain in the bedrock thermal layer
-x_range A,B (meters)	Specify the range of x coordinates. Use this to select a subset of an input grid that isn’t in the center of a domain.
-y_range A,B (meters)	Specify the range of y coordinates.

See [Grid registration](#) for details about the interpretation of L_x , L_y , and the way the grid spacing is computed.

¹ See [On the vertical coordinate in PISM, and a critical change of variable](#) for details.

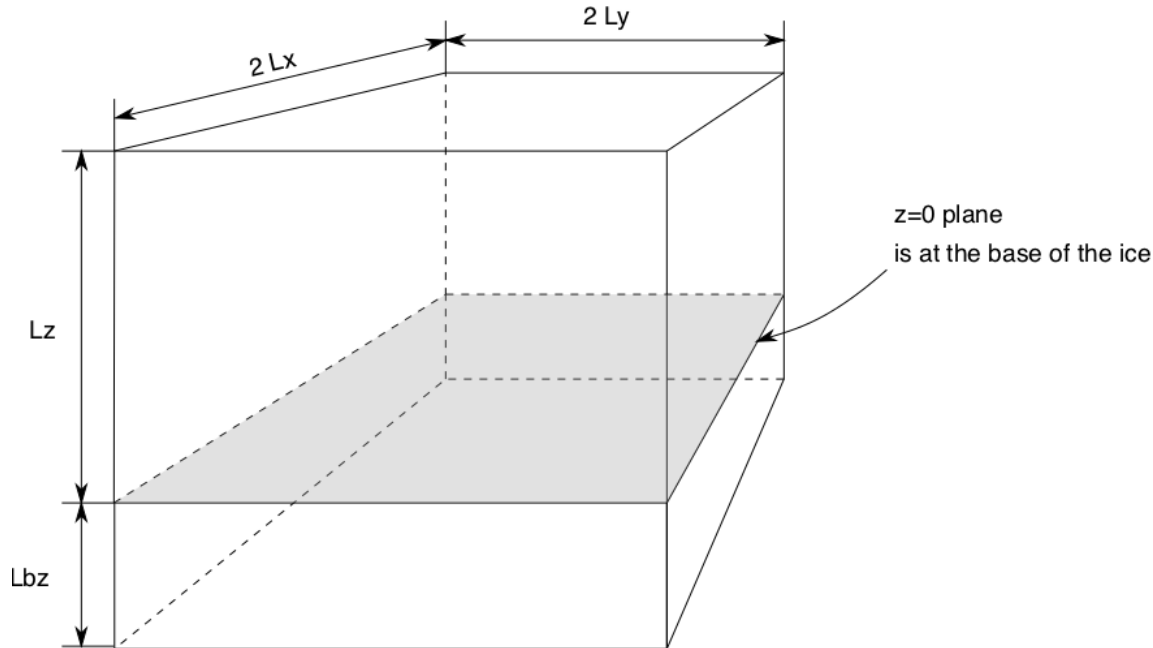


Fig. 3.17: PISM's computational box

Contents

- *Spatial grid*
 - *Grid registration*
 - *Grid projections*
 - *Parallel domain distribution*

Spatial grid

The PISM grid covering the computational box is equally spaced in horizontal (x and y) directions. Vertical spacing in the ice is quadratic by default but optionally equal spacing can be chosen; choose with options `-z_spacing [quadratic, equal]` at bootstrapping. The grid read from a “-i” input file is used as is. The bedrock thermal layer model always uses equal vertical spacing.

The grid is described by four numbers, namely the number of grid points M_x in the x direction, the number M_y in the y direction, the number M_z in the z direction within the ice, and the number M_{bz} in the z direction within the bedrock thermal layer. These are specified by options `-Mx`, `-My`, `-Mz`, and `-Mbz`, respectively. The defaults are 61, 61, 31, and 1, respectively. Note that M_x , M_y , M_z , and M_{bz} all indicate the number of grid *points* so the number of grid *spaces* are one less, thus 60, 60, 30, and 0 in the default case.

The lowest grid point in a column of ice, at $z = 0$, coincides with the highest grid point in the bedrock, so M_{bz} must always be at least one. Choosing $M_{bz} > 1$ is required to use the bedrock thermal model. When a thermal bedrock layer is used, the distance L_{bz} is controlled by the `-Lbz` option. Note that M_{bz} is unrelated to the bed deformation model (glacial isostasy model); see section [Earth deformation models](#).

In the quadratically-spaced case the vertical spacing near the ice/bedrock interface is about four times finer than it would be with equal spacing for the same value of M_z , while the spacing near the top of the computational box is correspondingly coarser. For a detailed description of the spacing of the grid, see the documentation on

`IceGrid::compute_vertical_levels()` in the [PISM class browser](#).

The user should specify the grid when using `-bootstrap` or when initializing a verification test (section [Verification](#)) or a simplified-geometry experiment (section [Simplified geometry experiments](#)). If one initializes PISM from a saved model state using `-i` then the input file determines all grid parameters. For instance, the command

```
pismr -i foo.nc -y 100
```

should work fine if `foo.nc` is a PISM output file. Because `-i` input files take precedence over options,

```
pismr -i foo.nc -Mz 201 -y 100
```

will give a warning that “PISM WARNING: ignoring command-line option ‘-Mz’”.

Grid registration

PISM’s horizontal computational grid is uniform and (by default) cell-centered.¹

This is not the only possible interpretation, but it is consistent with the finite-volume handling of mass (ice thickness) evolution in PISM.

Consider a grid with minimum and maximum x coordinates x_{\min} and x_{\max} and the spacing Δx . The cell-centered interpretation implies that the domain extends *past* x_{\min} and x_{\max} by one half of the grid spacing, see [Fig. 3.18](#).

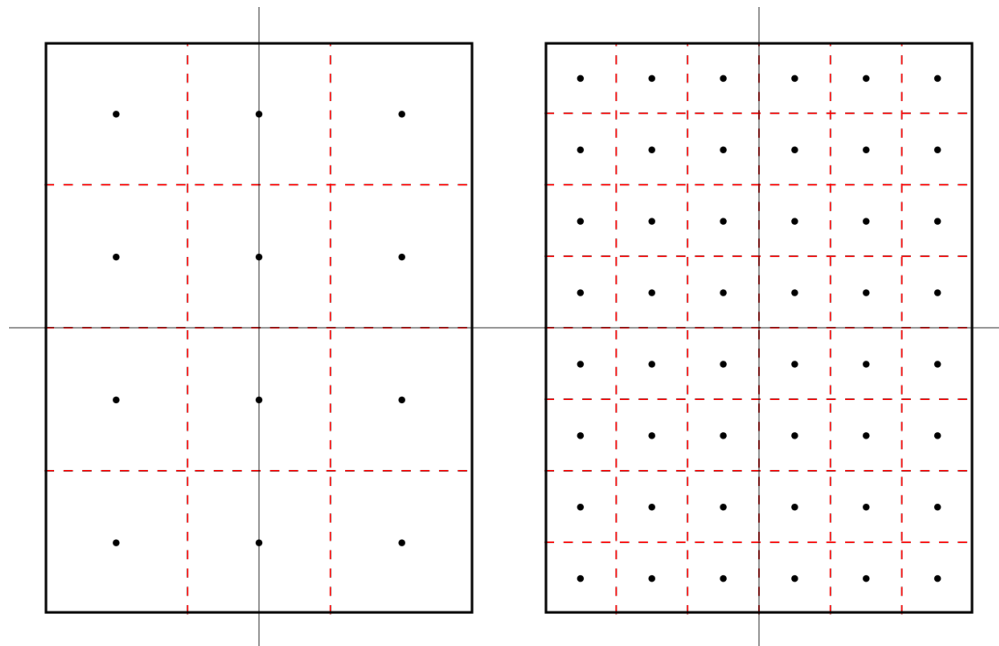


Fig. 3.18: Computational grids using the default (center) grid registration.

Left: a coarse grid. *Right:* a finer grid covering the same domain.

The solid black line represents the domain boundary, dashed red lines are cell boundaries, black circles represent grid points.

When getting the size of the domain from an input file, PISM will compute grid parameters as follows:

$$\Delta x = x_1 - x_0$$

$$L_x = \frac{1}{2}((x_{\max} - x_{\min}) + \Delta x).$$

¹ This is consistent with the [CF Conventions](#) document for data-sets without cell bounds: “If bounds are not provided, an application might reasonably assume the gridpoints to be at the centers of the cells, but we do not require that in this standard.”

This is not an issue when re-starting from a PISM output file but can cause confusion when specifying grid parameters at bootstrapping and reading in fields using “regridding.”

For example:

```
> pisms -grid.registration center \
-Lx 10 -Mx 4 \
-y 0 -verbose 1 \
-o grid-test.nc
> ncdump -v x grid-test.nc | tail -2 | head -1
x = -7500, -2500, 2500, 7500 ;
```

Note that we specified the domain half width of 10 km and selected 4 grid points in the x direction. The resulting x coordinates range from -7500 meters to 7500 meters with the grid spacing of 5 km.

In summary, with the default (center) grid registration

$$\begin{aligned}\Delta x &= \frac{2L_x}{M_x}, \\ x_{\min} &= x_c - L_x + \frac{1}{2}\Delta x, \\ x_{\max} &= x_c + L_x - \frac{1}{2}\Delta x,\end{aligned}\tag{3.1}$$

where x_c is the x -coordinate of the domain center.

Note: One advantage of this approach is that it is easy to build a set of grids covering a given region such that grid cells nest within each other as in Fig. 3.18. In particular, this makes it easier to create a set of surface mass balance fields for the domain that use different resolutions but *have the same total SMB*.

Compare this to

```
> pisms -grid.registration corner \
-Lx 10 -Mx 5 \
-y 0 -verbose 1 \
-o grid-test.nc
> ncdump -v x grid-test.nc | tail -2 | head -1
x = -10000, -5000, 0, 5000, 10000 ;
```

Here the grid spacing is also 5 km, although there are 5 grid points in the x direction and x coordinates range from -10000 to 10000 .

With the “corner” grid registration

$$\begin{aligned}\Delta x &= \frac{2L_x}{M_x - 1}, \\ x_{\min} &= x_c - L_x, \\ x_{\max} &= x_c + L_x.\end{aligned}\tag{3.2}$$

See Fig. 3.19 for an illustration.

To switch between (3.1) and (3.2), set the configuration parameter `grid.registration`.

Grid projections

PISM can use the PROJ.4 library (see *Required tool and libraries*) and projection information to compute

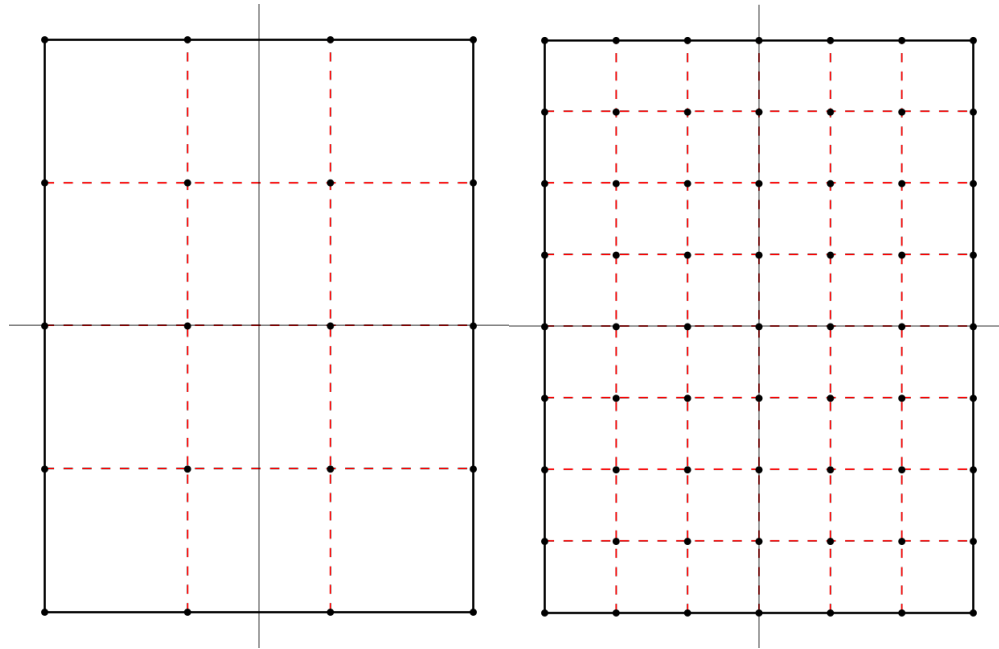


Fig. 3.19: Computational grids using the corner grid registration.
Left: a coarse grid. Right: a finer grid covering the same domain.

- a more accurate estimate of cell areas, improving the accuracy of reported areas and volumes,
- latitudes and longitudes of grid points (variables `lat` and `lon`), and
- latitudes and longitudes of cell corners (variables `lat_bnds` and `lon_bnds`).

To use this feature, compile PISM with PROJ.4 and add the global attribute `proj4` containing the parameter string describing the projection to the input file.

For example, the input file `pism_Greenland_5km_v1.1.nc` in *Getting started: a Greenland ice sheet example* has the following:

```
> ncdump -h pism_Greenland_5km_v1.1.nc | grep :proj4
:proj4 = "+proj=stere +lat_0=90 +lat_ts=71 +lon_0=-39 +k=1 +x_0=0 +y_0=0 +ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs" ;
```

The spinup run in that example disables cell area correction to avoid the dependency on PROJ.4 (look for `-grid.correct_cell_areas false` in the command). If we remove this option, PISM will report the following.

```
> pismr -i pism_Greenland_5km_v1.1.nc \
  -bootstrap -Mx 76 -My 141 -Mz 101 -Mbz 11 ... \
  -grid.correct_cell_areas true ... -o output.nc
...
* Got projection parameters "+proj=stere +lat_0=90 +lat_ts=71 +lon_0=-39 +k=1 +x_0=0 +y_0=0
+ellps=WGS84 +towgs84=0,0,0,0,0,0,0 +units=m +no_defs" from "pism_Greenland_5km_v1.1.nc".
* Computing cell areas using projection parameters...
* Computing longitude and latitude using projection parameters...
...
... done with run
Writing model state to file `output.nc'...
```

If the `proj4` attribute contains the string `"+init=epsg:XXXX"` where `XXXX` is 3413, 3031, or 26710, PISM will also create a CF-conforming mapping variable describing the projection in use.

“Mapping” variables following CF metadata conventions in input files are copied to output files (including `-extra_files`) but are **not** used to compute corrected cell areas and latitude/longitude coordinates.

To simplify post-processing and analysis with CDO PISM adds the PROJ.4 string (if known) to the mapping variable, putting it in the `proj4_params` attribute.

```
> ncdump -h g20km_10ka_hy.nc | grep mapping:

mapping:ellipsoid = "WGS84" ;
mapping:grid_mapping_name = "polar_stereographic" ;
mapping:false_easting = 0. ;
mapping:false_northing = 0. ;
mapping:latitude_of_projection_origin = 90. ;
mapping:standard_parallel = 71. ;
mapping:straight_vertical_longitude_from_pole = -39. ;
mapping:proj4_params = "+proj=stere +lat_0=90 +lat_ts=71 +lon_0=-39 +k=1 +x_0=0 +y_0=0
↪+ellps=WGS84 +towgs84=0,0,0,0,0,0 +units=m +no_defs" ;
```

Parallel domain distribution

When running PISM in parallel with `mpirun -n N`, the horizontal grid is distributed across N processes². PISM divides the grid into N_x parts in the x direction and N_y parts in the y direction. By default this is done automatically, with the goal that $N_x \times N_y = N$ and N_x is as close to N_y as possible. Note that N should, therefore, be a composite (not prime) number.

Users seeking to override this default can specify N_x and N_y using the `-Nx` and `-Ny` command-line options.

Once N_x and N_y are computed, PISM computes sizes of sub-domains $M_{x,i}$ so that $\sum_{i=1}^{N_x} M_{x,i} = M_x$ and $M_{x,i} - \lfloor M_x/N_x \rfloor < 1$. To specify strip widths $M_{x,i}$ and $M_{y,i}$, use command-line options `-procs_x` and `-procs_y`. Each option takes a comma-separated list of numbers as its argument. For example,

```
mpirun -n 3 pisms -Mx 101 -My 101 \
          -Nx 1 -procs_x 101 \
          -Ny 3 -procs_y 20,61,20
```

splits a 101×101 grid into 3 strips along the x axis.

To see the parallel domain decomposition from a completed run, see the `rank` variable in the output file, e.g. using `-o_size big`. The same `rank` variable is available as a spatial diagnostic field (section *Saving time series of spatially-varying diagnostic quantities*).

Model time

Table 3.4 gives the command-line options which control PISM time. If option `-ys` is absent then the start year is read from the input file (if present) or it defaults to zero. The default value for the end year is the start year plus the given (`-y`) run length. If both `-ys` and `-ye` are used then the run length is set to the difference. Using all three of `-ys`, `-y` and `-ys` is not allowed; this generates an error message.

Table 3.4: Command-line options controlling PISM time

Option	Description
<code>-y</code> (years)	Number of model years to run.
<code>-ys</code> (years)	Model year at which to start the run. Also resets the model time, ignoring any time in the input file.
<code>-ye</code> (years)	Model year at which to end the run.

² In most cases one process corresponds to one “core” of your computer.

Calendars

Most of PISM, and its ice dynamics core in particular, only needs to know the length of the current time-step. Internally PISM stores time in “seconds since a specified moment” and thus PISM generally does not use or need a calendar.¹ We refer to PISM internal time as *model time*.

One can select a calendar for more precise control of the model time, however. A “calendar” is a concept that is part of the [CF Conventions](#). Choosing a calendar is appropriate for runs for specific temporal periods like “the 18th-century” or “1989–2010”. The calendar is generally needed because specific knowledge of lengths of months and years is required to use climate data properly or to facilitate model validation.

PISM uses [CalCalcs](#) by David W. Pierce to perform calendric computations. This lets us support all the [calendars](#) defined by the CF Metadata Conventions document except for the 366_day (all_leap) calendar.

Time units in PISM’s output files always contain a reference date because it is required by the CF metadata conventions.

By default PISM uses the 365_day calendar. This is appropriate for runs that do not require precise application of forcing data or reporting on particular dates (paleo-climate runs, for example). In this mode PISM ignores the reference date in time unit specifications (such as “days since 1969-7-20”), though the value set using `time.reference_date` configuration parameter is saved in (is passed forward into) output files.

The calendar setting also affects the year length. In particular, with the default choice of the calendar (365_day), `-y 10` sets the run duration to

$$10 \times 365 \times 24 \times 60 \times 60 = 315360000$$

seconds, not $10 \times 365.25 \times 24 \times 60 \times 60$ or similar.

Note: This does **not** affect unit conversion: the factor used to convert m/s to $m/year$ does not depend on the calendar choice.

Table 3.5: Calendars supported by PISM. Please see CalCalcs documentation for details

Keyword	Meaning
gregorian or standard	Mixed Gregorian/Julian calendar used today.
proleptic_gregorian	Gregorian calendar extended to dates before 1582-10-15.
noleap or 365_day	Calendar with fixed-length 365-day years
360_day	Calendar with fixed-length 360-day years divided into 30-day months
julian	Julian calendar
none	no calendar

Selecting a non-trivial (Gregorian, Proleptic Gregorian, or Julian) calendar using the `time.calendar` configuration parameter or the `-calendar` command-line option enables calendar-based time management; see [Table 3.5](#). The implications of selecting such a calendar are:

- PISM uses the `units` attribute of coordinate variables *literally* (including the reference date) in unit conversions. Please make sure that the `time` variable in all forcing files has the `units` attribute such as “days since 2012-1-1”. PISM will stop with an error message if a time variable does not have a reference date in its unit specification.
- It is important to use units that are a fixed multiple of “seconds”, such as “minutes since 1989-1-1” or “days since 1999-12-31” and avoid “months” and “years”. (PISM uses UDUNITS-2 to convert units, and in UDUNITS one month is always interpreted as $\frac{1}{12} \cdot 365.242198781$ days.) Please see the [CF Conventions](#) document for details.

¹ Note seconds are part of SI units.

- PISM uses dates in standard output:

```
...
  time interval (length)  [2012-01-01, 2021-12-31]  (10.000 years)
...
S 2012-05-26:  0.00011    0.6306    0.000000000    0.00000
v$Eh m (dt=0.10000)
S 2012-07-01:  0.00014    0.6306    0.000000000    0.00000
```

Just like in the no-calendar mode, run length, run start and run end times are specified using `-y`, `-ys` and `-ye` command-line options, respectively. Arguments of these options are interpreted in a slightly different manner, though:

- the run length option `-y` takes an *integer* argument, interpreted as the number of *calendar* years
- options `-ys` and `-ye` take *dates* as arguments.

For example, either of the following commands sets up a run covering the 21st century:

```
pismr -calendar gregorian -ys 2001-1-1 -y 100 ...
pismr -calendar standard -ys 2001-1-1 -ye 2101-1-1 ...
```

(These option combinations are equivalent.)

It is also possible to run PISM for the duration of the available forcing data using the `-time_file` option. The command

```
pismr -calendar gregorian -time_file forcing.nc
```

will extract the reference date and run length from `forcing.nc`, respecting time bounds.

When a non-trivial calendar is selected, spatial and scalar time-series can be saved daily, monthly or yearly using these calendric computations. See sections *Saving time series of scalar diagnostic quantities* and *Saving time series of spatially-varying diagnostic quantities*.

Re-starting an interrupted run using `-time_file`

If a run using `-time_file` gets interrupted but manages to save a backup, re-starting with `-time_file` will attempt to re-do the entire run because options `-y`, `-ys`, and `-ye` are ignored:

```
# This run gets killed but leaves backup.nc:
pismr -i input.nc -time_file time.nc -o output.nc
# This WILL NOT start from the time saved in backup.nc
# and continue until the end time in time.nc
pismr -i backup.nc -time_file time.nc -o output.nc
```

In this case we want to set the start time of the run from `backup.nc`, but use the end time from `time.nc`. To achieve this, use the option `-time_file_continue_run`.

```
# This run gets killed but leaves backup.nc:
pismr -i input.nc -time_file time.nc -o output.nc
# This WILL continue until the end time in time.nc, starting from backup.nc
pismr -i backup.nc -time_file time.nc -o output.nc -time_file_continue_run
```

Diagnostic computations

A “diagnostic” computation can be defined as one where the internal state does not evolve. The internal state of PISM is the set of variables read by “-i”. You can ask PISM to do a diagnostic computation by setting the run duration to a small number such as 0.001 years (about 9 hours). The duration to use depends on the modeling setup, but should be smaller than the maximum time-step allowed by PISM’s stability criteria. Such short runs can also be used to look at additional fields corresponding to the current model state.

As an example, consider these two runs:

```
pisms -y 6000 -o foo.nc
pismr -i foo.nc -y 0.001 -o bar.nc -o_size big
```

The result of the second (short) run is a NetCDF file `bar.nc` which contains the full three-dimensional velocity field in the scalar NetCDF variables `uvel`, `vvel`, and `wvel`, as well as many other variables. The file `foo.nc` does not contain many of these fields because it was written with the default output size of `medium`. The “-y 0.001” run has diagnostically “filled-in” all the fields which PISM can model at a time step, but the run duration was chosen so as to avoid significant model state evolution during the run.

This diagnostic mode is often associated to the modeling of ice shelves and ice streams. section [An SSA flow model for the Ross Ice Shelf in Antarctica](#) describes using a short “diagnostic” run to model the Ross ice shelf [57]. Verification tests I and J, section [Verification](#), are diagnostic calculations using the SSA.

The NetCDF model state saved by PISM at the end of an *evolution* run (i.e. with “-y Y” for $Y > 0$) does not, under the default `-o_size medium` output size, contain the three-dimensional velocity field. Instead, it contains just a few more variables than those which are needed to restart the run with `-i`. One can force PISM to save all the supported diagnostic quantities at the end of a time-stepping run using the option `-o_size big`. Or one can go back and do a “-y small_number” diagnostic run using `-o_size big`.

3.4.2 Ice dynamics and thermodynamics

Choosing the stress balance

The basic stress balance used for all grounded ice in PISM is the non-sliding, thermomechanically-coupled SIA [18]. For the vast majority of most ice sheets, as measured by area or volume, this is an appropriate model, which is an $O(\epsilon^2)$ approximation to the Stokes model if ϵ is the depth-to-length ratio of the ice sheet [19].

The shallow shelf approximation (SSA) stress balance applies to floating ice. See the Ross ice shelf example in section [An SSA flow model for the Ross Ice Shelf in Antarctica](#) for an example in which the SSA is only applied to floating ice.

The SSA is also used in PISM to describe the sliding of grounded ice and the formation of ice streams [17]. Specifically for the SSA with “plastic” (Coulomb friction) basal resistance, the locations of ice streams are determined as part of a free boundary problem of Schoof [33], a model for emergent ice streams within a ice sheet and ice shelf system. This model explains ice streams through a combination of plastic till failure and SSA stress balance.

This SSA description of ice streams is, however, also the preferred “sliding law” for the SIA [17], [25]. The SSA should be combined with the SIA, in this way, in preference to classical SIA sliding laws which make ice basal velocity a local function of the basal value of the driving stress. The resulting combination of SIA and SSA is a “hybrid” approximation of the Stokes model [25]. Option `-stress_balance ssa+sia` turns on this “hybrid” model. In this use of the SSA as a sliding law, floating ice is also subject to the SSA.

Of course there is more to the use of a stress balance than just turning it on! At all grounded points a yield stress, or a pseudo-yield-stress in the case of power law sliding (section [Controlling basal strength](#)), is computed from the amount of stored basal water and from a (generally) spatially-varying till strength. The amount of stored basal water is modeled by the subglacial hydrology mode choice (section [Subglacial hydrology](#)) based on the basal melt rate which is, primarily, thermodynamically-determined (subsection [Controlling basal strength](#)).

Table 3.6 describes the basic choice of stress balance. If the SSA stress balance is used, a choice of two solvers is available, namely `-ssa_method fd` (default) or `-ssa_method fem`. See Table 3.7, which describes additional controls on the numerical solution of the stress balance equations. If option `-ssa_method fd` is chosen then several more controls on numerics are available; see Table 3.8. If the ice sheet being modeled has any floating ice then the user is advised to read section *PIK options for marine ice sheets* on modeling marine ice sheets.

Table 3.6: The basic choice of stress balance

Option	Description
<code>-stress_balance none</code>	Turn off ice flow completely.
<code>-stress_balance sia</code> (default)	Grounded ice flows by the non-sliding SIA. Floating ice essentially doesn't flow, so this model is not recommended for marine ice sheets.
<code>-stress_balance ssa</code>	Use the SSA model exclusively. Horizontal ice velocity is constant throughout ice columns.
<code>-stress_balance prescribed_sliding</code>	Use the constant-in-time prescribed sliding velocity field read from a file set using <code>-prescribed_sliding_file</code> , variables <code>ubar</code> and <code>vbar</code> . Horizontal ice velocity is constant throughout ice columns.
<code>-stress_balance ssa+sia</code>	The recommended sliding law, which gives the SIA+SSA hybrid stress balance. Combines SSA-computed velocity, using pseudo-plastic till, with SIA-computed velocity according to the combination in [25]; similar to [17]. Floating ice uses SSA only.
<code>-stress_balance prescribed_sliding+sia</code>	Use the constant-in-time prescribed sliding velocity in combination with the non-sliding SIA.

Table 3.7: Choice of, and controls on, the numerical SSA stress balance.

Option	Description
<code>-ssa_method [fd fem]</code>	Both finite difference (<code>fd</code> ; the default) and finite element (<code>fem</code>) versions of the SSA numerical solver are implemented in PISM. The <code>fd</code> solver is the only one which allows PIK options (section <i>PIK options for marine ice sheets</i>). <code>fd</code> uses Picard iteration [17], while <code>fem</code> uses a Newton method. The <code>fem</code> solver has surface velocity inversion capability [56].
<code>-ssa_eps</code> (10^{13})	The numerical schemes for the SSA compute an effective viscosity ν which depends on strain rates and ice hardness (thus temperature). The minimum value of the effective viscosity times the thickness (i.e. νH) largely determines the difficulty of solving the numerical SSA. This constant is added to keep νH bounded away from zero: $\nu H \rightarrow \nu H + \epsilon_{\text{SSA}}$, where ϵ_{SSA} is set using this option. Units of <code>ssa_eps</code> are Pa m s. Set to zero to turn off this lower bound.
<code>-ssa_view_nuh</code>	View the product νH for your simulation as a runtime viewer (section <i>Run-time diagnostic viewers</i>). In a typical Greenland run we see a wide range of values for νH from $\sim 10^{14}$ to $\sim 10^{20}$ Pa m s.

Table 3.8: Controls on the numerical iteration of the `-ssa_method fd` solver

Option	Description
<code>-ssa_maxi</code> (300)	Set the maximum allowed number of Picard (nonlinear) iterations in solving the shallow shelf approximation.
<code>-ssa_rtol</code> (10^{-4})	The Picard iteration computes a vertically-averaged effective viscosity which is used to solve the equations for horizontal velocity. Then the new velocities are used to recompute an effective viscosity, and so on. This option sets the relative change tolerance for the effective viscosity. The Picard iteration stops when successive values $v^{(k)}$ of the vertically-averaged effective viscosity satisfy $\ (v^{(k)} - v^{(k-1)})H\ _1 \leq Z\ v^{(k)}H\ _1$ where $Z = \text{ssa_rtol}$.
<code>-ssa_fd_ksp_rtol</code> (10^{-5})	Set the relative change tolerance for the iteration inside the Krylov linear solver used at each Picard iteration.

Ice rheology

Contents

- *Ice rheology*
 - *Choosing the flow laws for SIA and SSA stress balances*
 - *Choose enhancement factor and exponent*

The “rheology” of a viscous fluid refers to the relation between the applied stress and the resulting deformation, the strain rate. The models of ice rheology available in PISM are all isotropic [34]. A rheology in this class is described by a “flow law”, which is, in the most general case in PISM, a function $F(\sigma, T, \omega, P, d)$ in the “constitutive relation” form

$$D_{ij} = F(\sigma, T, \omega, P, d) \sigma'_{ij}. \quad (3.3)$$

Here D_{ij} is the strain rate tensor, σ'_{ij} is the stress deviator tensor, T is the ice temperature, ω is the liquid water fraction, P is the pressure, d is the grain size, and $\sigma^2 = \frac{1}{2}\|\sigma'_{ij}\|_F^2 = \frac{1}{2}\sigma'_{ij}\sigma'_{ij}$ defines the second invariant σ of the stress deviator tensor.

Form (3.3) of the flow law is used in the SIA, but the “viscosity” form of a flow law, found by inverting the constitutive relation (3.3), is needed for ice shelf and ice stream (SSA) flow [17]:

$$\sigma'_{ij} = 2\nu(D, T, \omega, P, d) D_{ij} \quad (3.4)$$

Here $\nu(D, T, \omega, P, d)$ is the “effective viscosity” and $D^2 = \frac{1}{2}D_{ij}D_{ij}$.

Most of the flow laws in PISM are of Glen-Nye single-power type. For example,

$$F(\sigma, T) = A(T)\sigma^{n-1} \quad (3.5)$$

is the common temperature-dependent Glen law [62], [18] (which has no dependence on liquid water fraction, pressure, or grain size). If the ice softness $A(T) = A_0$ is constant then the law is isothermal, whereas if there is dependence on temperature then $A(T)$ is usually a generalization of “Arrhenius” form

$$A(T) = A \exp(-Q/(RT)).$$

The more elaborate Goldsby-Kohlstedt law [63] is a function $F(\sigma, T, P, d)$, but in this case the function F cannot be factored into a product of a function of T, P, d and a single power of σ , as in form (3.5).

There is only one choice for the flow law which takes full advantage of the enthalpy mode of PISM, which is the thermodynamical modeling (i.e. conservation of energy) default. Namely the Glen-Paterson-Budd-Lliboutry-Duval flow law [23], [64], [62], which is a function $F(\sigma, T, \omega, P)$. This law is the only one in the literature where the ice softness depends on both the temperature and the liquid water fraction, so it parameterizes the (observed) softening of pressure-melting-temperature ice as its liquid fraction increases. One can use this default polythermal law or one may choose among a number of “cold ice” laws listed in Table 3.9 which do not use the liquid water fraction.

All flow law parameters can be changed using configuration parameters; see section *PISM’s configuration parameters and how to change them* and the implementation of flow laws in the [Source Code Browser](#). Note that different flow laws have different numbers of parameters, but all have at least two parameters (e.g. A_0 and n in `isothermal_glen`). One can create a new, and reasonably arbitrarily, scalar function F by modifying source code; see source files in `src/base/rheology/`.

Choosing the flow laws for SIA and SSA stress balances

Command-line options `-sia_flow_law` and `-ssa_flow_law` choose which flow law is used by the SIA and SSA stress balances, respectively. Allowed arguments are listed in Table 3.9 below. Viscosity form (3.4) is not known for the Goldsby-Kohlstedt law [63], so option “`-ssa_flow_law gk`” is an error.

Table 3.9: Single-power flow laws. Choose the ice rheology using `-sia_flow_law` and `-ssa_flow_law` and one of the names in this table. Flow law choices other than `gpbld` do not use the liquid water fraction ω but only the temperature T .

Name	Comments and References
<code>gpbld</code>	Glen-Paterson-Budd-Lliboutry-Duval law [64], the enthalpy-based default in PISM [23]. Extends the Paterson-Budd law (below) to positive liquid water fraction. If $A_c(T)$ is from Paterson-Budd then this law returns $A(T, \omega) = A_c(T)(1 + C\omega),$ where ω is the liquid water fraction, C is a configuration parameter <code>flow_law.gpbld.water_frac_coeff</code> [default $C = 181.25$], and ω is capped at level <code>flow_law.gpbld.water_frac_observed_limit</code> .
<code>gpbld3</code>	Same as <code>gpbld</code> , but with the fixed Glen exponent $n = 3$. This flow law implementation is <i>significantly faster</i> thanks to a number of low-level optimizations. It is also less flexible (all constants are hard-wired and cannot be changed using configuration parameters).
<code>pb</code>	Paterson-Budd law, the cold-mode default. Fixed Glen exponent $n = 3$. Has a split “Arrhenius” term $A(T) = A \exp(-Q/RT^*)$ where $A = 3.615 \times 10^{-13} \text{ s}^{-1} \text{ Pa}^{-3}, Q = 6.0 \times 10^4 \text{ J mol}^{-1}$ if $T^* < 263 \text{ K}$ and $A = 1.733 \times 10^3 \text{ s}^{-1} \text{ Pa}^{-3}, Q = 13.9 \times 10^4 \text{ J mol}^{-1}$ if $T^* > 263 \text{ K}$. Here T^* is pressure-adjusted temperature [62].
<code>arr</code>	<i>Cold</i> part of Paterson-Budd. Regardless of temperature, the A and Q values for $T^* < 263 \text{ K}$ in the Paterson-Budd law apply. This is the flow law used in the thermomechanically-coupled exact solutions run by <code>pismv -test F</code> and <code>pismv -test G</code> [18], [65].
<code>arrwarm</code>	<i>Warm</i> part of Paterson-Budd. Regardless of temperature, the A and Q values for $T^* > 263 \text{ K}$ in Paterson-Budd apply.
<code>hooke</code>	Hooke law with $A(T) = A \exp(-Q/(RT^*) + 3C(T_r - T^*)^k).$ Fixed Glen exponent $n = 3$ and constants as in [66], [36].
<code>isothermal_glen</code>	The isothermal Glen flow law. Here $F(\sigma) = A_0 \sigma^{n-1}$ with inverse $\nu(D) = \frac{1}{2} B_0 D^{(1-n)/(2n)}$ where A_0 is the ice softness and $B_0 = A_0^{-1/n}$ is the ice hardness.
<code>gk</code>	This law has a combination of exponents from $n = 1.8$ to $n = 4$ [63]. It can only be used by the SIA stress balance. Because it has more than one power, option <code>-sia_n</code> has no effect, though <code>-sia_e</code> works as expected. This law does not use the liquid water fraction, but only the temperature.

Choose enhancement factor and exponent

An enhancement factor can be added to any flow law through a runtime option. Single-power laws also permit control of the flow law exponent through a runtime option.

Options `-sia_e` and `-ssa_e` set flow enhancement factors for the SIA and SSA respectively. Option `-sia_e` sets “ e ” in $D_{ij} = e F(\sigma, T, \omega, P, d) \sigma'_{ij}$, in equation (3.3). Option `-ssa_e` sets “ e ” in the viscosity form so that $\sigma'_{ij} = e^{-1/n} 2 \nu(D, T, \omega, P, d) D_{ij}$.

Options `-sia_n` and `-ssa_n` set the exponent when a single-power flow law is used (see Table 3.9). Simply changing to a different value from the default $n = 3$ is not recommended without a corresponding change to the enhancement factor, however. This is because the coefficient and the power are non-trivially linked when a power law is fit to

experimental data [67], [62].

Here is a possible approach to adjusting both the enhancement factor and the exponent. Suppose σ_0 is preferred as a scale (reference) for the driving stress that appears in both SIA and SSA models. Typically this is on the order of one bar or 10^5 Pa. Suppose one wants the same amount of deformation D_0 at this reference driving stress as one changes from the old exponent n_{old} to the new exponent n_{new} . That is, suppose one wants both

$$D_0 = E_{old} A \sigma_0^{n_{old}} \quad \text{and} \quad D_0 = E_{new} A \sigma_0^{n_{new}}$$

to be true with a new enhancement factor E_{new} . Eliminating D_0 and solving for the new enhancement factor gives

$$E_{new} = E_{old} \sigma_0^{n_{old}-n_{new}}. \quad (3.6)$$

It follows, for example, that if one has a run with values

```
-sia_e 3.0 -sia_n 3.0
```

then a new run with exponent $n = 6.0$ and the same deformation at the reference driving stress of 10^5 Pa will use

```
-sia_e 3.0e-15 -sia_n 6.0
```

because $E_{new} = 3.0\sigma_0^{3-6} = 3.0 \times (10^5)^{-3}$ from equation (3.6).

A corresponding formula applies to `-ssa_e` if the `-ssa_n` value changes.

Table 3.10: For all flow laws, an enhancement factor can be added by a runtime option. For the single-power flow laws in Table 3.9, the (Glen) exponent can be controlled by a runtime option.

Option	Configuration parameter	Comments
<code>-sia_e (1.0)</code>	<code>stress_balance.sia.enhancement_factor</code>	Note (see the supplement of [55]) used 3.0 for Greenland ice sheet simulations while [6] used 4.5 for simulations of the Antarctic ice sheet with PISM-PIK.
<code>-sia_n (3.0)</code>	<code>stress_balance.sia.Glen_exponent</code>	See text and eqn (3.6) to also set <code>-sia_e</code> if <code>-sia_n</code> changes.
<code>-ssa_e (1.0)</code>	<code>stress_balance.ssa.enhancement_factor</code>	Note [6] used 0.512 for simulations of the Antarctic ice sheet with PISM-PIK.
<code>-ssa_n (3.0)</code>	<code>stress_balance.ssa.Glen_exponent</code>	See text and eqn (3.6) to also set <code>-ssa_e</code> if <code>-ssa_n</code> changes.

Surface gradient method

PISM computes surface gradients to determine the “driving stress”

$$(\tau_{d,x}, \tau_{d,y}) = -\rho g H \nabla h,$$

where H is the ice thickness, and h is the ice surface elevation. The driving stress enters into both the SIA and SSA stress balances, but in the former the driving stress is needed on a staggered grid, while in the latter the driving stress is needed on the regular grid.

Surface gradients are computed by finite differences in several slightly-different ways. There are options for choosing which method to use **in the SIA model**, but to the best of our knowledge there is no theoretical advice on the best, most robust mechanism.

The SSA model uses centered finite differences, switching from centered to one-sided near the ice margin and does not recognize choices other than `-gradient` eta.

There are three `-gradient` methods in PISM:

Table 3.11: Options controlling the surface gradient computation in the SIA code

Option	Description
<code>-gradient mahaffy</code>	This most “standard” way computes the surface slope onto the staggered grid for the SIA [68]. It makes $O(\Delta x^2, \Delta y^2)$ errors. For computations of driving stress on the regular grid, centered differencing is used instead.
<code>-gradient haseloff</code>	This is the default method. It only differs from <code>mahaffy</code> at ice-margin locations, where the slope is approximated using one-sided finite differences in cases where an adjacent ice-free bedrock surface elevation is above the ice elevation.
<code>-gradient eta</code>	<p>In this method we first transform the thickness H by $\eta = H^{(2n+2)/n}$ and then differentiate the sum of the thickness and the bed using centered differences:</p> $\nabla h = \nabla H + \nabla b = \frac{n}{(2n+2)} \eta^{(-n-2)/(2n+2)} \nabla \eta + \nabla b.$ <p>Here b is the bed elevation and h is the surface elevation. This transformation sometimes has the benefits that the surface values of the horizontal velocity and vertical velocity, and the driving stress, are better behaved near the margin. See [21] for technical explanation of this transformation and compare [69]. The actual finite difference schemes applied to compute the surface slope are similar to option <code>mahaffy</code>.</p>

Note: The `-gradient eta` may improve the model performance near *grounded* margins but should not be used in simulations of marine ice sheets.

Modeling conservation of energy

In normal use PISM solves the conservation of energy problem within the ice, the thin subglacial layer, and a layer of thermal bedrock. For the ice and the subglacial layer it uses an enthalpy-based scheme [23] which allows the energy to be conserved even when the temperature is at the pressure-melting point.

Ice at the melting point is called “temperate” ice. Part of the thermal energy of temperate ice is in the latent heat of the liquid water stored between the crystals of the temperate ice. Part of the thermal energy of the whole glacier is in the latent heat of the liquid water under the glacier. The enthalpy scheme correctly models these storehouses of thermal energy, and thus it allows polythermal and fully-temperate glaciers to be modeled [61].

The state of the full conservation of energy model includes the 3D enthalpy variable plus the 2D `bwat` and `tillwat` subglacial hydrology state variables (subsection *Subglacial hydrology*), all of which are seen in output files. The important basal melt rate computation involves all of these energy state variables, because the basal melt rate (`bmelt` in output files) comes from conserving energy across the ice-bedrock layer [23]. Fields `temp`, `liqfrac`, and `temp_pa` seen in output files are all actually diagnostic outputs because all of these can be recovered from the enthalpy and the ice geometry.

Because this part of PISM is just a conservation law, there is little need for the user to worry about controlling it. If desired, however, conservation of energy can be turned off entirely with `-energy none`. The default enthalpy-based conservation of energy model (i.e. `-energy enthalpy`) can be replaced by the temperature-based (i.e. “cold ice”) method used in [17] and verified in [18] by setting option `-energy cold`.

The thermal bedrock layer model is turned off by setting `-Mbz 1` (i.e. zero spaces) while it is turned on by choosing a depth and number of points, as in `-Lbz 1000 -Mbz 21`, for example, which gives a layer depth of 1000 m and grid spaces of 50 m (= 1000/20). The input geothermal flux (`bheatflx` in output files) is applied at the bottom of the bedrock thermal layer if such a layer is present and otherwise it is applied at the base of the ice.

Computing ice age

By default, PISM does not compute the age of the ice because it does not directly impact ice flow when using the default flow laws. It is very easy to turn on. Just set `-age`. A 3D variable age will appear in output files. It is read at input if `-age` is set and otherwise it is ignored even if present in the input file. If `-age` is set and the variable age is absent in the input file then the initial age is set to zero.

The age of the ice can be used in two parameterizations in the SIA stress balance model:

1. Ice grain size parameterization based on data from [58] and [59] (Vostok core data). In PISM, only the Goldsby-Kohlstedt flow law (see *Ice rheology*) uses the grain size.
2. The flow enhancement factor can be coupled to the age of the ice as in [60]: during Eemian and Holocene e is set to
 - `stress_balance.sia.enhancement_factor_interglacial` during Eemian and Holocene,
 - `stress_balance.sia.enhancement_factor` otherwise.

See `time.eemian_start`, `time.eemian_end`, and `time.holocene_start`.

3.4.3 The subglacier

This section describes how to control PISM's sub-models of subglacial processes: the basal strength, the subglacial hydrology, and bed deformation.

Two of these sections (*Controlling basal strength* and *Parameterization of bed roughness in the SIA*) can be thought of as parts of PISM's stress balance model: they cover PISM's treatment of basal boundary conditions: the former for stress balance models including longitudinal stresses, and the latter for the non-sliding SIA flow.

Controlling basal strength

When using option `-stress_balance ssa+sia`, the SIA+SSA hybrid stress balance, a model for basal resistance is required. This model for basal resistance is based, at least conceptually, on the hypothesis that the ice sheet is underlain by a layer of till [78]. The user can control the parts of this model:

- the so-called sliding law, typically a power law, which relates the ice base (sliding) velocity to the basal shear stress, and which has a coefficient which is or has the units of a yield stress,
- the model relating the effective pressure on the till layer to the yield stress of that layer, and
- the model for relating the amount of water stored in the till to the effective pressure on the till.

This subsection explains the relevant options.

The primary example of `-stress_balance ssa+sia` usage is in section *Getting started: a Greenland ice sheet example* of this Manual, but the option is also used in sections *MISMIP*, *MISMIP3d*, and *Example: A regional model of the Jakobshavn outlet glacier in Greenland*.

In PISM the key coefficient in the sliding is always denoted as yield stress τ_c , which is `tauc` in PISM output files. This parameter represents the strength of the aggregate material at the base of an ice sheet, a poorly-observed mixture of pressurized liquid water, ice, granular till, and bedrock bumps. The yield stress concept also extends to the power law form, and thus most standard sliding laws can be chosen by user options (below). One reason that the yield stress is a useful parameter is that it can be compared, when looking at PISM output files, to the driving stress (`taud_mag` in PISM output files). Specifically, where `tauc < taud_mag` you are likely to see sliding if option `-stress_balance ssa+sia` is used.

A historical note on modeling basal sliding is in order. Sliding can be added directly to a SIA stress balance model by making the sliding velocity a local function of the basal value of the driving stress. Such an SIA sliding mechanism

appears in ISMIP-HEINO [79] and in EISMINT II experiment H [14], among other places. This kind of sliding is *not* recommended, as it does not make sense to regard the driving stress as the local generator of flow if the bed is not holding all of that stress [17], [37]. Within PISM, for historical reasons, there is an implementation of SIA-based sliding only for verification test E; see section [Verification](#). PISM does *not* support this SIA-based sliding mode in other contexts.

Choosing the sliding law

In PISM the sliding law can be chosen to be a purely-plastic (Coulomb) model, namely,

$$|\tau_b| \leq \tau_c \quad \text{and} \quad \tau_b = -\tau_c \frac{\mathbf{u}}{|\mathbf{u}|} \quad \text{if and only if} \quad |\mathbf{u}| > 0. \quad (3.7)$$

Equation (3.7) says that the (vector) basal shear stress τ_b is at most the yield stress τ_c , and that only when the shear stress reaches the yield value can there be sliding. The sliding law can, however, also be chosen to be the power law

$$\tau_b = -\tau_c \frac{\mathbf{u}}{u_{\text{threshold}}^q |\mathbf{u}|^{1-q}}, \quad (3.8)$$

where $u_{\text{threshold}}$ is a parameter with units of velocity (see below). Condition (3.7) is studied in [33] and [80] in particular, while power laws for sliding are common across the glaciological literature (e.g. see [67], [45]). Notice that the coefficient τ_c in (3.8) has units of stress, regardless of the power q .

In both of the above equations (3.7) and (3.8) we call τ_c the *yield stress*. It corresponds to the variable `tauc` in PISM output files. We call the power law (3.8) a “pseudo-plastic” law with power q and threshold velocity $u_{\text{threshold}}$. At the threshold velocity the basal shear stress τ_b has exact magnitude τ_c . In equation (3.8), q is the power controlled by `-pseudo_plastic_q`, and the threshold velocity $u_{\text{threshold}}$ is controlled by `-pseudo_plastic_uthreshold`. The plastic model (3.7) is the $q = 0$ case of (3.8).

See [Table 3.12](#) for options controlling the choice of sliding law. The purely plastic case is the default; just use `-stress_balance ssa+sia` to turn it on. (Or use `-stress_balance ssa` if a model with no vertical shear is desired.)

Warning: Options `-pseudo_plastic_q` and `-pseudo_plastic_uthreshold` have no effect if `-pseudo_plastic` is not set.

Table 3.12: Sliding law command-line options

Option	Description
<code>-pseudo_plastic</code>	Enables the pseudo-plastic power law model. If this is not set the sliding law is purely-plastic, so <code>pseudo_plastic_q</code> and <code>pseudo_plastic_uthreshold</code> are inactive.
<code>-plastic_reg (m/a)</code>	Set the value of ϵ regularization of the plastic law, in the formula $\tau_b = -\tau_c \mathbf{u} / \sqrt{ \mathbf{u} ^2 + \epsilon^2}$. The default is 0.01 m/a. This parameter is inactive if <code>-pseudo_plastic</code> is set.
<code>-pseudo_plastic_q</code>	Set the exponent q in (3.8). The default is 0.25.
<code>-pseudo_plastic_uthreshold (m/a)</code>	Set $u_{\text{threshold}}$ in (3.8). The default is 100 m/a.

Equation (3.8) is a very flexible power law form. For example, the linear case is $q = 1$, in which case if $\beta = \tau_c / u_{\text{threshold}}$ then the law is of the form

$$\tau_b = -\beta \mathbf{u}$$

(The “ β ” coefficient is also called β^2 in some sources (see [32], for example).) If you want such a linear sliding law, and you have a value $\beta = \text{beta}$ in Pa s m^{-1} , then use this option combination:

```
-pseudo_plastic \
-pseudo_plastic_q 1.0 \
-pseudo_plastic_uthreshold 3.1556926e7 \
-yield_stress constant -tauc beta
```

This sets $u_{\text{threshold}}$ to 1 m s^{-1} but using units m a^{-1} .

More generally, it is common in the literature to see power-law sliding relations in the form

$$\tau_b = -C|\mathbf{u}|^{m-1}\mathbf{u},$$

where C is a constant, as for example in sections *MISMIP* and *MISMIP3d*. In that case, use this option combination:

```
-pseudo_plastic \
-pseudo_plastic_q m \
-pseudo_plastic_uthreshold 3.1556926e7 \
-yield_stress constant \
-tauc C
```

Determining the yield stress

Other than setting it to a constant, which only applies in some special cases, the above discussion does not determine the yield stress τ_c . As shown in Table 3.13, there are two schemes for determining τ_c in a spatially-variable manner:

- `-yield_stress mohr_coulomb` (the default) determines the yields stress by models of till material property (the till friction angle) and of the effective pressure on the saturated till, or
- `-yield_stress constant` allows the yield stress to be supplied as time-independent data, read from the input file.

In normal modelling cases, variations in yield stress are part of the explanation of the locations of ice streams [33]. The default model `-yield_stress mohr_coulomb` determines these variations in time and space. The value of τ_c is determined in part by a subglacial hydrology model, including the modeled till-pore water amount `tillwat` (section *Subglacial hydrology*), which then determines the effective pressure N_{til} (see below). The value of τ_c is also determined in part by a material property field $\phi = \text{tillphi}$, the “till friction angle”. These quantities are related by the Mohr-Coulomb criterion [67]:

$$\tau_c = c_0 + (\tan \phi) N_{\text{til}}. \quad (3.9)$$

Here c_0 is called the “till cohesion”, whose default value in PISM is zero (see [33], formula (2.4)) but which can be set by option `-till_cohesion`.

Option combination `-yield_stress constant -tauc X` can be used to fix the yield stress to have value $\tau_c = X$ at all grounded locations and all times if desired. This is unlikely to be a good modelling choice for real ice sheets.

Table 3.13: Command-line options controlling how yield stress is determined

Option	Description
<code>-yield_stress mohr_coulomb</code>	The default. Use equation (3.9) to determine τ_c . Only effective if <code>-stress_balance ssa</code> or <code>-stress_balance ssa+sia</code> is also set.
<code>-till_cohesion</code>	Set the value of the till cohesion (c_0) in the plastic till model. The value is a pressure, given in Pa.
<code>-tauc_slippery_grounding_lines</code>	If set, reduces the basal yield stress at grounded-below-sea-level grid points one cell away from floating ice or ocean. Specifically, it replaces the normally-computed τ_c from the Mohr-Coulomb relation, which uses the effective pressure from the modeled amount of water in the till, by the minimum value of τ_c from Mohr-Coulomb, i.e. using the effective pressure corresponding to the maximum amount of till-stored water. Does not alter the reported amount of till water, nor does this mechanism affect water conservation.
<code>-plastic_phi (degrees)</code>	Use a constant till friction angle. The default is 30° .
<code>-topg_to_phi (list of 4 numbers)</code>	Compute ϕ using equation (3.10).
<code>-yield_stress constant</code>	Keep the current values of the till yield stress τ_c . That is, do not update them by the default model using the stored basal melt water. Only effective if <code>-stress_balance ssa</code> or <code>-stress_balance ssa+sia</code> is also set.
<code>-tauc</code>	Directly set the till yield stress τ_c , in units Pa, at all grounded locations and all times. Only effective if used with <code>-yield_stress constant</code> , because otherwise τ_c is updated dynamically.

We find that an effective, though heuristic, way to determine $\phi = \text{tillphi}$ in (3.9) is to make it a function of bed elevation [55], [6], [25]. This heuristic is motivated by hypothesis that basal material with a marine history should be weak [35]. PISM has a mechanism setting $\phi = \text{tillphi}$ to be a *piecewise-linear* function of bed elevation. The option is

```
-topg_to_phi phimin,phimax,bmin,bmax
```

Thus the user supplies 4 parameters: ϕ_{\min} , ϕ_{\max} , b_{\min} , b_{\max} , where b stands for the bed elevation. To explain these, we define $M = (\phi_{\max} - \phi_{\min}) / (b_{\max} - b_{\min})$. Then

$$\phi(x, y) = \begin{cases} \phi_{\min}, & b(x, y) \leq b_{\min}, \\ \phi_{\min} + (b(x, y) - b_{\min}) M, & b_{\min} < b(x, y) < b_{\max}, \\ \phi_{\max}, & b_{\max} \leq b(x, y). \end{cases} \quad (3.10)$$

It is worth noting that an earth deformation model (see section [Earth deformation models](#)) changes $b(x, y) = \text{topg}$ used in (3.10), so that a sequence of runs such as

```
pismr -i foo.nc -bed_def lc -stress_balance ssa+sia -topg_to_phi 10,30,-50,0 ... -o bar.nc
pismr -i bar.nc -bed_def lc -stress_balance ssa+sia -topg_to_phi 10,30,-50,0 ... -o baz.nc
```

will use *different* tillphi fields in the first and second runs. PISM will print a warning during initialization of the second run:

```
* Initializing the default basal yield stress model...
option -topg_to_phi seen; creating tillphi map from bed elev ...
PISM WARNING: -topg_to_phi computation will override the 'tillphi' field
               present in the input file 'bar.nc'!
```

Omitting the `-topg_to_phi` option in the second run would make PISM continue with the same tillphi field which was set in the first run.

Determining the effective pressure

When using the default option `-yield_stress mohr_coulomb`, the effective pressure on the till N_{til} is determined by the modeled amount of water in the till. Lower effective pressure means that more of the weight of the ice is carried by the pressurized water in the till and thus the ice can slide more easily. That is, equation (3.9) sets the value of τ_c proportionately to N_{til} . The amount of water in the till is, however, a nontrivial output of the hydrology (section *Subglacial hydrology*) and conservation-of-energy (section *Modeling conservation of energy*) submodels in PISM.

Following [81], based on laboratory experiments with till extracted from an ice stream in Antarctica, [82] propose the following parameterization which is used in PISM. It is based on the ratio $s = W_{til}/W_{til}^{max}$ where $W_{til} = \text{tillwat}$ is the effective thickness of water in the till and $W_{til}^{max} = \text{hydrology.tillwat_max}$ is the maximum amount of water in the till (see section *Subglacial hydrology*):

$$N_{til} = \min \left\{ P_o, N_0 \left(\frac{\delta P_o}{N_0} \right)^s 10^{(e_0/C_c)(1-s)} \right\} \quad (3.11)$$

Here P_o is the ice overburden pressure, which is determined entirely by the ice thickness and density, and the remaining parameters are set by options in Table 3.14. While there is experimental support for the default values of C_c , e_0 , and N_0 , the value of $\delta = \text{basal_yield_stress.mohr_coulomb.till_effective_fraction_overburden}$ should be regarded as uncertain, important, and subject to parameter studies to assess its effect.

Table 3.14: Command-line options controlling how till effective pressure N_{til} in equation (3.9) is determined

Option	Description
<code>-till_reference_void_ratio</code>	$= e_0$ in (3.11), dimensionless, with default value 0.69 [81]
<code>-till_compressibility_coefficient</code>	$= C_c$ in (3.11), dimensionless, with default value 0.12 [81]
<code>-till_effective_fraction_overburden</code>	$= \delta$ in (3.11), dimensionless, with default value 0.02 [82]
<code>-till_reference_effective_pressure</code>	$= N_0$ in (3.11), in Pa, with default value 1000.0 [81]

Subglacial hydrology

At the present time, two simple subglacial hydrology models are implemented *and documented* in PISM, namely `-hydrology null` and `-hydrology routing`; see Table 3.15 and [82]. In both models, some of the water in the subglacial layer is stored locally in a layer of subglacial till by the hydrology model. In the `routing` model water is conserved by horizontally-transporting the excess water (namely `bwat`) according to the gradient of the modeled hydraulic potential. In both hydrology models a state variable `tillwat` is the effective thickness of the layer of liquid water in the till; it is used to compute the effective pressure on the till (see the previous subsection). The pressure of the transportable water `bwat` in the `routing` model does not relate directly to the effective pressure on the till.

Table 3.15: Command-line options to choose the hydrology model

Option	Description
<code>-hydrology null</code>	The default model with only a layer of water stored in till. Not mass conserving in the map-plane but much faster than <code>-hydrology routing</code> . Based on “undrained plastic bed” model of [87]. The only state variable is <code>tillwat</code> .
<code>-hydrology routing</code>	A mass-conserving horizontal transport model in which the pressure of transportable water is equal to overburden pressure. The till layer remains in the model, so this is a “drained and conserved plastic bed” model. The state variables are <code>bwat</code> and <code>tillwat</code> .

See Table 3.16 for options which apply to all hydrology models. Note that the primary water source for these models is the energy conservation model which computes the basal melt rate `basal_melt_rate_grounded`. There is, however, also option `-hydrology_input_to_bed_file` which allows the user to *add* water directly into the subglacial

layer, in addition to the computed `basal_melt_rate_grounded` values. Thus `-hydrology_input_to_bed_file` allows the user to model drainage directly to the bed from surface runoff, for example. Also option `-hydrology_bmelt_file` allows the user to replace the computed `basal_melt_rate_grounded` rate by values read from a file, thereby effectively decoupling the hydrology model from the ice dynamics (esp. conservation of energy).

Table 3.16: Subglacial hydrology command-line options which apply to all hydrology models

Option	Description
<code>-hydrology_bmelt_file</code>	Specifies a NetCDF file which contains a time-independent field <code>basal_melt_rate_grounded</code> which has units of water thickness per time. This rate <i>replaces</i> the conservation-of-energy computed rate <code>basal_melt_rate_grounded</code> .
<code>-hydrology_const_bmelt (m/s)</code>	If <code>-hydrology_use_const_bmelt</code> is set then use this to set the constant rate (water thickness per time).
<code>-hydrology_input_to_bed_file</code>	Specifies a NetCDF file which contains a time-dependent field <code>inputtobed</code> which has units of water thickness per time. This rate is <i>added to</i> the <code>basal_melt_rate_grounded</code> rate.
<code>-hydrology_input_to_bed_period (a)</code>	The period, in years, of <code>-hydrology_input_to_bed_file</code> data.
<code>-hydrology_input_to_bed_reference_year (a)</code>	The reference year for periodizing the <code>-hydrology_input_to_bed_file</code> data.
<code>-hydrology_tillwat_max (m)</code>	Maximum effective thickness for water stored in till.
<code>-hydrology_tillwat_decay_rate (m/a)</code>	Water accumulates in the till at the basal melt rate <code>basal_melt_rate_grounded</code> , minus this rate.
<code>-hydrology_use_const_bmelt</code>	Replace the conservation-of-energy basal melt rate <code>basal_melt_rate_grounded</code> with a constant.

The default model: `-hydrology null`

In this model the water is *not* conserved but it is stored locally in the till up to a specified amount; option `-hydrology_tillwat_max` sets that amount. The water is not conserved in the sense that water above the `hydrology_tillwat_max` level is lost permanently. This model is based on the “undrained plastic bed” concept of [87]; see also [17].

In particular, denoting `tillwat` by W_{til} , the till-stored water layer effective thickness evolves by the simple equation

$$\frac{\partial W_{til}}{\partial t} = \frac{m}{\rho_w} - C \quad (3.12)$$

where $m = \text{basal_melt_rate_grounded}$ ($\text{kg m}^{-2} \text{s}^{-1}$), ρ_w is the density of fresh water, and $C = \text{hydrology_tillwat_decay_rate}$. At all times bounds $0 \leq W_{til} \leq W_{til}^{max}$ are satisfied.

This `-hydrology null` model has been extensively tested in combination with the Mohr-Coulomb till (section *Controlling basal strength* above) for modelling ice streaming (see [55] and [17], among others).

The mass-conserving model: `-hydrology routing`

In this model the water *is* conserved in the map-plane. Water does get put into the till, with the same maximum value `hydrology_tillwat_max`, but excess water is horizontally-transported. An additional state variable `bwat`, the effective thickness of the layer of transportable water, is used by `routing`. This transportable water will flow in

the direction of the negative of the gradient of the modeled hydraulic potential. In the `routing` model this potential is calculated by assuming that the transportable subglacial water is at the overburden pressure [88]. Ultimately the transportable water will reach the ice sheet grounding line or ice-free-land margin, at which point it will be lost. The amount that is lost this way is reported to the user.

In this model `tillwat` also evolves by equation (3.12), but several additional parameters are used in determining how the transportable water `bwat` flows in the model; see Table 3.17. Specifically, the horizontal subglacial water flux is determined by a generalized Darcy flux relation [78], [89]

$$\mathbf{q} = -k W^\alpha |\nabla\psi|^{\beta-2} \nabla\psi \quad (3.13)$$

where \mathbf{q} is the lateral water flux, $W = \text{bwat}$ is the effective thickness of the layer of transportable water, ψ is the hydraulic potential, and k, α, β are controllable parameters (Table 3.17).

In the `routing` model the hydraulic potential ψ is determined by

$$\psi = P_o + \rho_w g(b + W) \quad (3.14)$$

where $P_o = \rho_i g H$ is the ice overburden pressure, g is gravity, ρ_i is ice density, ρ_w is fresh water density, H is ice thickness, and b is the bedrock elevation.

For most choices of the relevant parameters and most grid spacings, the `routing` model is at least two orders of magnitude more expensive computationally than the `null` model. This follows directly from the CFL-type time-step restriction on lateral flow of a fluid with velocity on the order of centimeters to meters per second (i.e. the subglacial liquid water `bwat`). (By comparison, much of PISM ice dynamics time-stepping is controlled by the much slower velocity of the flowing ice.) Therefore the user should start with short runs of order a few model years. The option `-report_mass_accounting` is also recommended, so as to see the time-stepping behavior at `stdout`. Finally, daily or even hourly reporting for scalar and spatially-distributed time-series to see hydrology model behavior, especially on fine grids (e.g. < 1 km).

Table 3.17: Command-line options specific to hydrology model routing

Option	Description
<code>-hydrology_hydraulic_conductivity k</code>	$= k$ in formula (3.13).
<code>-hydrology_null_strip (km)</code>	In the boundary strip water is removed and this is reported. This option specifies the width of this strip, which should typically be one or two grid cells.
<code>-hydrology_gradient_power_in_flux β</code>	$= \beta$ in formula (3.13).
<code>-hydrology_thickness_power_in_flux α</code>	$= \alpha$ in formula (3.13).
<code>-report_mass_accounting</code>	At each major (ice dynamics) time-step, the duration of hydrology time steps is reported, along with the amount of subglacial water lost to ice-free land, to the ocean, and into the “null strip”.

Earth deformation models

The option `-bed_def [iso, 1c]` turns one of the two available bed deformation models.

The first model `-bed_def iso`, is instantaneous pointwise isostasy. This model assumes that the bed at the starting time is in equilibrium with the load. Then, as the ice geometry evolves, the bed elevation is equal to the starting bed elevation minus a multiple of the increase in ice thickness from the starting time:

$$b(t, x, y) = b(0, x, y) - f [H(t, x, y) - H(0, x, y)] .$$

Here f is the density of ice divided by the density of the mantle, so its value is determined by setting the values of `bed_deformation.mantle_density` and `constants.ice_density` in the configuration file; see *PISM's configuration parameters and how to change them*. For an example and verification, see Test H in `:ref'sec-verification`.

The second model `-bed_def lc` is much more physical. It is based on papers by Lingle and Clark [83] and Bueler and others [84]. It generalizes and improves the most widely-used earth deformation model in ice sheet modeling, the flat earth Elastic Lithosphere Relaxing Asthenosphere (ELRA) model [85]. It imposes essentially no computational burden because the Fast Fourier Transform is used to solve the linear differential equation [84]. When using this model in PISM, the rate of bed movement (uplift) and the viscous plate displacement are stored in the PISM output file and then used to initialize the next part of the run. In fact, if gridded “observed” uplift data is available, for instance from a combination of actual point observations and/or paleo ice load modeling, and if that uplift field is put in a NetCDF variable with standard name `tendency_of_bedrock_altitude` in the input file, then this model will initialize so that it starts with the given uplift rate.

Here are minimal example runs to compare these models:

```
mpiexec -n 4 pisms -eisII A -y 8000 -o eisIIA_nobd.nc
mpiexec -n 4 pisms -eisII A -bed_def iso -y 8000 -o eisIIA_bdiso.nc
mpiexec -n 4 pisms -eisII A -bed_def lc -y 8000 -o eisIIA_bdlc.nc
```

Compare the `topg`, `usurf`, and `dbdt` variables in the resulting output files. See also the comparison done in [84].

To include “measured” uplift rates during initialization, use the option `-uplift_file` to specify the name of the file containing the field `dbdt` (CF standard name: `tendency_of_bedrock_altitude`).

Use the `-topg_delta_file` option to apply a correction to the bed topography field read in from an input file. This sets the bed topography b at the beginning of a run as follows:

$$b = b_0 + \Delta b. \quad (3.15)$$

Here b_0 is the bed topography (`topg`) read in from an input file and Δb is the `topg_delta` field read in from the file specified using this option.

A correction like this can be used to get a bed topography field at the end of a paleo-climate run that is closer to observed present day topography. The correction is computed by performing a “preliminary” run and subtracting modeled bed topography from present day observations. A subsequent run with this correction should produce a bed elevations that are closer to observed values.

Parameterization of bed roughness in the SIA

Schoof [86] describes how to alter the SIA stress balance to model ice flow over bumpy bedrock topography. One computes the amount by which bumpy topography lowers the SIA diffusivity. An internal quantity used in this method is a smoothed version of the bedrock topography. As a practical matter for PISM, this theory improves the SIA’s ability to handle bed roughness because it parameterizes the effects of “higher-order” stresses which act on the ice as it flows over bumps. For additional technical description of PISM’s implementation, see *Using Schoof’s parameterized bed roughness technique in PISM*.

There is only one associated option: `-bed_smoother_range` gives the half-width of the square smoothing domain in meters. If zero is given, `-bed_smoother_range 0` then the mechanism is turned off. The mechanism is on by default using executable `pismr`, with the half-width set to 5 km (`-bed_smoother_range 5.0e3`), giving Schoof’s recommended smoothing size of 10 km [86].

This mechanism is turned off by default in executables `pisms` and `pismv`.

Under the default setting `-o_size medium`, PISM writes fields `topgsmooth` and `schoofs_theta` from this mechanism. The thickness relative to the smoothed bedrock elevation, namely `topgsmooth`, is the difference between the unsmoothed surface elevation and the smoothed bedrock elevation. It is *only used internally by this mechanism*, to compute a modified value of the diffusivity; the rest of PISM does not use this or any other smoothed bed. The

field `schoofs_theta` is a number θ between 0 and 1, with values significantly below zero indicating a reduction in diffusivity, essentially a drag coefficient, from bumpy bed.

3.4.4 Marine ice sheet modeling

PISM is often used to model whole ice sheets surrounded by ocean, with attached floating ice shelves, or smaller regions like outlet glaciers flowing into embayments and possibly generating floating tongues. This section explains the geometry and stress balance mechanisms in PISM that apply to floating ice, at the vertical calving faces of floating ice, or at marine grounding lines. The physics at calving fronts is very different from elsewhere on an ice sheet, because the flow is nothing like the lubrication flow addressed by the SIA, and nor is the physics like the sliding flow in the interior of an ice domain. The needed physics at the calving front can be thought of as boundary condition modifications to the mass continuity equation and to the SSA stress balance equation. The physics of grounding lines are substantially handled by recovering sub-grid information through interpolation.

PIK options for marine ice sheets

Contents

- *PIK options for marine ice sheets*
 - *Stress condition at calving fronts*
 - *Partially-filled cells at the boundaries of ice shelves*
 - *Iceberg removal*
 - *Sub-grid treatment of the grounding line position*

References [73], [70], [25] by the research group of Prof. Anders Levermann at the Potsdam Institute for Climate Impact Research (“PIK”), Germany, describe most of the mechanisms covered in this section. These are all improvements to the grounded, SSA-as-a-sliding law model of [17]. These improvements make PISM an effective Antarctic model, as demonstrated by [74], [6], [75], among other publications. These improvements had a separate existence as the “PISM-PIK” model from 2009–2010, but since PISM stable0.4 are part of PISM itself.

A summary of options to turn on most of these “PIK” mechanisms is in Table 3.18. More information on the particular mechanisms is given in sub-sections *Stress condition at calving fronts* through *Sub-grid treatment of the grounding line position* that follow the Table.

Table 3.18: Options which turn on PIK ice shelf front and grounding line mechanisms. A calving law choice is needed in addition to these options.

Option	Description
<code>-cfbc</code>	apply the stress boundary condition along the ice shelf calving front [25]
<code>-kill_icebergs</code>	identify and eliminate free-floating icebergs, which cause well-posedness problems for the SSA stress balance solver [25]
<code>-part_grid</code>	allow the ice shelf front to advance by a part of a grid cell, avoiding the development of unphysically-thinned ice shelves [73]
<code>-subgl</code>	apply interpolation to compute basal shear stress and basal melt near the grounding line [76]
<code>-no_subgl_basal_melt</code>	don’t apply interpolation to compute basal melt near the grounding line if <code>-subgl</code> is set [76]
<code>-pik</code>	equivalent to option combination <code>-cfbc -kill_icebergs -part_grid -subgl</code>

Note: When in doubt, PISM users should set option `-pik` to turn on all of mechanisms in Table 3.18. The user should also choose a calving model from Table 3.20. However, the `-pik` mechanisms will not be effective if the non-default FEM stress balance `-ssa_method fem` is chosen.

Stress condition at calving fronts

The vertically integrated force balance at floating calving fronts has been formulated by [31] as

$$\int_{z_s - \frac{\rho}{\rho_w} H}^{z_s + (1 - \frac{\rho}{\rho_w}) H} \sigma \cdot \mathbf{n} \, dz = \int_{z_s - \frac{\rho}{\rho_w} H}^{z_s} \rho_w g (z - z_s) \mathbf{n} \, dz. \quad (3.16)$$

with \mathbf{n} being the horizontal normal vector pointing from the ice boundary oceanward, σ the *Cauchy* stress tensor, H the ice thickness and ρ and ρ_w the densities of ice and seawater, respectively, for a sea level of z_s . The integration limits on the right hand side of equation (3.16) account for the pressure exerted by the ocean on that part of the shelf, which is below sea level (bending and torque neglected). The limits on the left hand side change for water-terminating outlet glacier or glacier fronts above sea level according to the bed topography. By applying the ice flow law (section *Ice rheology*), equation (3.16) can be rewritten in terms of strain rates (velocity derivatives), as one does with the SSA stress balance itself.

Note that the discretized SSA stress balance, in the default finite difference discretization chosen by `-ssa_method fd`, is solved with an iterative matrix scheme. If option `-cfbc` is set then, during matrix assembly, those equations which are for fully-filled grid cells along the ice domain boundary have terms replaced according to equation (3.16), so as to apply the correct stresses [73], [25].

Partially-filled cells at the boundaries of ice shelves

Albrecht et al [73] argue that the correct movement of the ice shelf calving front on a finite-difference grid, assuming for the moment that ice velocities are correctly determined (see below), requires tracking some cells as being partially-filled (option `-part_grid`). If the calving front is moving forward, for example, then the neighboring cell gets a little ice at the next time step. It is not correct to add that little mass as a thin layer of ice which fills the cell's horizontal extent, as that would smooth the steep ice front after a few time steps. Instead the cell must be regarded as having ice which is comparably thick to the upstream cells, but where the ice only partially fills the cell.

Specifically, the PIK mechanism turned on by `-part_grid` adds mass to the partially-filled cell which the advancing front enters, and it determines the coverage ratio according to the ice thickness of neighboring fully-filled ice shelf cells. If option `-part_grid` is used then the PISM output file will have field `ice_area_specific_volume` which tracks the amount of ice in the partially-filled cells as a “thickness”, or, more appropriately, “volume per unit area”. When a cell becomes fully-filled, in the sense that the `ice_area_specific_volume` reaches the average of the ice thickness in neighboring ice-filled cells, then the residual mass is redistributed to neighboring partially-filled or empty grid cells.

The stress balance equations determining the velocities are only sensitive to “fully-filled” cells. Similarly, advection is controlled only by values of velocity in fully-filled cells. Adaptive time stepping (specifically: the CFL criterion) limits the speed of ice front propagation so that at most one empty cell is filled, or one full cell emptied, per time step by the advance or retreat, respectively, of the calving front.

Iceberg removal

Any calving mechanism (see section *Calving*) removes ice along the seaward front of the ice shelf domain. This can lead to isolated cells either filled or partially-filled with floating ice, or to patches of floating ice (icebergs) fully

surrounded by ice free ocean neighbors. This ice is detached from the flowing and partly-grounded ice sheet. That is, calving can lead to icebergs.

In terms of our basic model of ice as a viscous fluid, however, the stress balance for an iceberg is not well-posed because the ocean applies no resistance to balance the driving stress. (See [33].) In this situation the numerical SSA stress balance solver will fail.

Option `-kill_icebergs` turns on the mechanism which cleans this up. This option is therefore generally needed if there is nontrivial calving or significant variations in sea level during a simulation. The mechanism identifies free-floating icebergs by using a 2-scan connected-component labeling algorithm. It then eliminates such icebergs, with the corresponding mass loss reported as a part of the 2D discharge flux diagnostic (see section *Saving time series of spatially-varying diagnostic quantities*).

Sub-grid treatment of the grounding line position

The command-line option `-subgl` turns on a parameterization of the grounding line position based on the “LI” parameterization described in [77] and [76]. With this option PISM computes an extra flotation mask, available as the `cell_grounded_fraction` output variable, which corresponds to the fraction of the cell that is grounded. Cells that are ice-free or fully floating are assigned the value of 0 while fully-grounded icy cells get the value of 1. Partially grounded cells, the ones which contain the grounding line, get a value between 0 and 1. The resulting field has two uses:

- It is used to scale the basal friction in cells containing the grounding line in order to avoid an abrupt change in the basal friction from the “last” grounded cell to the “first” floating cell. See the source code browser for the detailed description and section *MISMIP3d* for an application.
- It is used to adjust the basal melt rate in cells containing the grounding line: in such cells the basal melt rate is set to $M_{b,\text{adjusted}} = \lambda M_{b,\text{grounded}} + (1 - \lambda)M_{b,\text{shelf-base}}$, where λ is the value of the flotation mask. Use `-no_subgl_basal_melt` to disable this.

Flotation criterion, mask, and sea level

The most basic decision about marine ice sheet dynamics made internally by PISM is whether a ice-filled grid cell is floating. That is, PISM applies the “flotation criterion” [25] at every time step and at every grid location to determine whether the ice is floating on the ocean or not. The result is stored in the `mask` variable. The `mask` variable has `pism_intent = diagnostic`, and thus it does *not* need to be included in the input file set using the `-i` option.

The possible values of the mask are given in Table 3.19. The mask does not *by itself* determine ice dynamics. For instance, even when ice is floating (mask value `MASK_FLOATING`), the user must turn on the usual choice for ice shelf dynamics, namely the SSA stress balance, by using options `-stress_balance ssa` or `-stress_balance ssa+sia`.

Table 3.19: The PISM mask, in combination with user options, determines the dynamical model.

Mask value	Meaning
0 = <code>MASK_ICE_FREE_BEDROCK</code>	ice free bedrock
2 = <code>MASK_GROUNDED</code>	ice is grounded
3 = <code>MASK_FLOATING</code>	ice is floating (the SIA is never applied; the SSA is applied if the <code>ssa</code> or <code>ssa+sia</code> stress balance model is selected)
4 = <code>MASK_ICE_FREE_OCEAN</code>	ice-free ocean

Assuming that the geometry of the ice is allowed to evolve (which can be turned off by option `-no_mass`), and assuming an ocean exists so that a sea level is used in the flotation criterion (which can be turned off by option `-dry`),

then at each time step the mask will be updated.

Calving

Contents

- *Calving*
 - *Eigen calving*
 - *Von Mises stress calving*
 - *Additional calving methods*

The Table 3.20 summarizes options controlling calving parameterizations implemented in PISM.

Table 3.20: Options for the calving models in PISM.

Option	Description
-calving_cfl	Apply CFL-type criterion to reduce (limit) PISM's time step using the horizontal calving rate computed by <code>eigen_calving</code> or <code>vonmises_calving</code> .
-calving eigen_calving	Physically-based calving parameterization [70], [25]. Wherever the product of principal strain rates is positive, the calving rate is proportional to this product.
-eigen_calving_K (ms)	Sets the proportionality parameter K in ms.
-calving vonmises_calving	Physically-based calving parameterization [71] that uses the tensile von Mises stresses.
-vonmises_calving_sigma_max (Pa)	Sets the maximum tensile stress $\tilde{\sigma}$ in Pa.
-calving thickness_calving	Calve all near-terminus ice which is thinner than ice threshold thickness H_{cr} .
-thickness_calving_threshold (m)	Sets the thickness threshold H_{cr} in meters.
-thickness_calving_threshold_file	Specifies the file containing the variable <code>calving_threshold</code> to be used as the spatially-variable thickness threshold
-calving float_kill	All floating ice is calved off immediately.
-float_kill_margin_only	At each time step, calve cells at the ice margin only instead of removing all floating ice.
-float_kill_calve_near_grounding_line	Calve floating ice near the grounding line (this is the default). Disable using <code>-float_kill_calve_near_grounding_line off</code> .
-calving ocean_kill	All ice flowing into grid cells marked as "ice free ocean", according to the ice thickness in the provided file, is calved.
-ocean_kill_file	Sets the file with the <code>thk</code> field used to compute maximum ice extent.

To select several calving mechanisms, use a comma-separated list of keywords mentioned in Table 3.20:

```
-calving eigen_calving,thickness_calving,ocean_kill,vonmises_calving
```

Eigen calving

PISM-PIK introduced a physically-based 2D-calving parameterization [70]. This calving parameterization is turned on in PISM by option `-calving eigen_calving`. Average calving rates, c , are proportional to the product of principal components of the horizontal strain rates, $\dot{\epsilon}_{\pm}$, derived from SSA-velocities

$$c = K \dot{\epsilon}_{+} \dot{\epsilon}_{-} \quad \text{and} \quad \dot{\epsilon}_{\pm} > 0. \quad (3.17)$$

The rate c is in m s^{-1} , and the principal strain rates $\dot{\epsilon}_{\pm}$ have units s^{-1} , so K has units m s . The constant K incorporates material properties of the ice at the front. It can be set using the `-eigen_calving_K` option or a configuration parameter (`calving.eigen_calving.K` in `src/pism_config.cdl`).

The actual strain rate pattern strongly depends on the geometry and boundary conditions along the confinements of an ice shelf (coast, ice rises, front position). The strain rate pattern provides information in which regions preexisting fractures are likely to propagate, forming rifts (in two directions). These rifts may ultimately intersect, leading to the release of icebergs. This (and other) ice shelf calving models are not intended to resolve individual rifts or calving events, but it produces structurally-stable calving front positions which agree well with observations. Calving rates balance calving-front ice flow velocities on average.

The partially-filled grid cell formulation (section *Partially-filled cells at the boundaries of ice shelves*) provides a framework suitable to relate the calving rate produced by `eigen_calving` to the mass transport scheme at the ice shelf terminus. Ice shelf front advance and retreat due to calving are limited to a maximum of one grid cell length per (adaptive) time step. The calving rate (velocity) from `eigen_calving` can be used to limit the overall timestep of PISM—thus slowing down all of PISM—by using `-calving_cfl`. This “CFL”-type time-step limitation is definitely recommended in high-resolution runs which attempt to model calving position accurately. Without this option, under certain conditions where PISM’s adaptive time step happens to be long enough, dendritic structures can appear at the calving front because the calving mechanism cannot “keep up” with the computed calving rate.

Von Mises stress calving

Warning: This code is experimental and has not yet been thoroughly tested, use at your own risk.

While `eigen-calving` (section *Eigen calving*) is appropriate for Antarctic ice shelves, it does not work for outlet glaciers that flow in narrow fjords. Along valleys with nearly parallel walls, the transverse component of the velocity is close to zero, and the transversal strain rate is therefore also close to zero and noisy.

Instead of the product of the eigen strain rates, [71] proposes a calving law where the calving rate c is a functionally related to tensile stresses:

$$c = |\mathbf{u}| \frac{\tilde{\sigma}}{\sigma_{max}}, \quad (3.18)$$

where $\tilde{\sigma}$ is the tensile von Mises stress and σ_{max} is a threshold that has units Pa . It can be set as a configuration parameter (`calving.vonmises.sigma_max` in `src/pism_config.cdl`). As the tensile fracture strength is much smaller than the compressive fracture strength, the effective tensile strain rate is defined as

$$\tilde{\epsilon}_e = \left(\frac{1}{2} \left(\max(0, \dot{\epsilon}_{+})^2 + \max(0, \dot{\epsilon}_{-})^2 \right) \right)^{1/2}. \quad (3.19)$$

Following [71] $\tilde{\sigma}$ is given by

$$\tilde{\sigma} = \sqrt{3} B \tilde{\epsilon}_e^{1/n}, \quad (3.20)$$

where B is the ice hardness.

Similar to `eigen_calving`, the calving rate from `vonmises_calving` can be used to limit the overall timestep of PISM — thus slowing down all of PISM — by using `-calving_cfl`.

Additional calving methods

PISM also includes three more basic calving mechanisms (Table 3.20). The option `-calving thickness_calving` is based on the observation that ice shelf calving fronts are commonly thicker than about 150–250 m (even though the physical reasons are not clear yet). Accordingly, any floating ice thinner than H_{cr} is removed along the front, at a rate at most one grid cell per time step. The value of H_{cr} can be set using the `-thickness_calving_threshold` option or the `calving.thickness_calving.threshold` configuration parameter.

To set a spatially-variable ice thickness threshold, use the option `-thickness_calving_threshold_file` or the parameter `calving.thickness_calving.threshold_file`. This file should contain the variable `calving_threshold` in meters (or other compatible units).

Option `-calving float_kill` removes (calves), at each time step of the run, any ice that satisfies the flotation criterion. Use of this option implies that there are no ice shelves in the model at all.

Use the option `-float_kill_margin_only` to restrict this to cells at the ice margin.

Sometimes it is useful to preserve a one-cell-wide shelf near the grounding line. To do this, set `calving.float_kill.calve_near_grounding_line` to false.

Option `-calving ocean_kill` chooses the calving mechanism removing ice in the “open ocean”. It requires the option `-ocean_kill_file`, which specifies the file containing the ice thickness field `thk`. (This can be the input file specified using `-i`.) Any locations which were ice-free (`thk == 0`) and which had bedrock elevation below sea level (`topg < 0`), in the provided data set, are marked as ice-free ocean. The resulting mask is not altered during the run, and is available as diagnostic field `ocean_kill_mask`. At these places any floating ice is removed at each step of the run. Ice shelves can exist in locations where a positive thickness was supplied in the provided data set.

Modeling melange back-pressure

Equation (3.16) above, describing the stress boundary condition for ice shelves, can be written in terms of velocity components:

$$\begin{aligned} 2\nu H(2u_x + u_y)\mathbf{n}_x + 2\nu H(u_y + v_x)\mathbf{n}_y &= \int_b^h (p_{ice} - p_{ocean})dz \mathbf{n}_x, \\ 2\nu H(u_y + v_x)\mathbf{n}_x + 2\nu H(2v_y + u_x)\mathbf{n}_y &= \int_b^h (p_{ice} - p_{ocean})dz \mathbf{n}_y. \end{aligned} \quad (3.21)$$

Here ν is the vertically-averaged ice viscosity, b is the ice base elevation, h is the ice top surface elevation, and p_{ocean} and p_{ice} are pressures of the column of sea water and ice, respectively.

We call the integral on the right hand side of (3.21) the “pressure difference term”. To model the effect of melange [72] on the stress boundary condition, we assume that the melange back-pressure $p_{melange}$ does not exceed $p_{ice} - p_{ocean}$. Therefore we introduce $\lambda \in [0, 1]$ (the melange back pressure fraction) such that

$$p_{melange} = \lambda(p_{ice} - p_{ocean}).$$

Then melange pressure is added to the ordinary ocean pressure so that the pressure difference term scales with λ :

$$\begin{aligned} \int_b^h (p_{ice} - (p_{ocean} + p_{melange})) dz &= \int_b^h (p_{ice} - (p_{ocean} + \lambda(p_{ice} - p_{ocean}))) dz \\ &= (1 - \lambda) \int_b^h (p_{ice} - p_{ocean}) dz. \end{aligned} \quad (3.22)$$

This formula replaces the integral on the right hand side of (3.21).

The resulting stress boundary condition at the shelf front is

$$\begin{aligned} 2\nu H(2u_x + u_y)\mathbf{n}_x + 2\nu H(u_y + v_x)\mathbf{n}_y &= (1 - \lambda) \int_b^h (p_{\text{ice}} - p_{\text{ocean}}) dz \mathbf{n}_x, \\ 2\nu H(u_y + v_x)\mathbf{n}_x + 2\nu H(2v_y + u_x)\mathbf{n}_y &= (1 - \lambda) \int_b^h (p_{\text{ice}} - p_{\text{ocean}}) dz \mathbf{n}_y. \end{aligned} \quad (3.23)$$

By default, λ is set to zero, but PISM implements a scalar time-dependent “melange back pressure fraction offset” forcing in which λ can be read from a file. Please see the [Climate Forcing Manual](#) for details.

3.4.5 Disabling sub-models

Certain major model components, unlike more peripheral ones like bed deformation or calving, are “on” by default. They do not need to be turned on explicitly. For example, the SIA computation is so common that it would be a hassle to require an option to turn it on every time you need it.

But sometimes one wants to disable particular components, during model spin-up, for example. PISM has the following “off” switches:

- `-no_mass` disables the mass-continuity (conservation of mass) step
- `-energy none` disables the conservation of energy computation
- `-energy cold` makes PISM use temperature instead of enthalpy in the energy conservation code
- `-stress_balance none` disables the stress balance computation (useful for testing surface mass balance inputs)
- `-dry` essentially disables ocean models: ice is always considered to be grounded, the sub-shelf melt rate and temperature is not used, and the calving-front boundary condition is computed ignoring the water pressure exerted on the vertical face at a (possibly submerged) terminus.

3.4.6 Dealing with more difficult modeling choices

Most uses of an ice sheet model depend on careful modeling choices in situations where there are considerable uncertainties *and* the model results depend strongly on those choices. There may be, at the present state of knowledge, *no clear default values* that PISM can provide. Furthermore, the available PISM options and sub-models are known to *not* be sufficient for all users. Thus there are modelling situations for which we know the user may have to do a great deal more hard work than just choose among PISM runtime options.

Here are example cases where users have worked hard:

- User made use of available data in order to choose parameters for existing PISM models. These parameters then override PISM defaults.

Example

Use regional atmosphere model output to identify PDD parameters suitable for modeling surface mass balance on a particular ice sheet. Then supply these parameters to PISM by a `-config_override` file.

- User wrote code, including code which modified current PISM internals, either to add additional processes or to “correct” PISM default process models.

Example

Add a new sub-ice-shelf melt model by modifying C++ code in the `src/coupler/` directory.

- User simplified the model in use, instead of the default which was more elaborate.

Example

Instead of using the PISM default mechanism connecting basal melt rate and basal strength, bypass this mechanism by generating a map of yield stress `tauc` directly and supplying it as input.

3.5 Practical usage

Running PISM requires many practical decisions, from pre-processing input data and selecting diagnostic quantities to save to monitoring running simulations and managing code modifications; see the following sub-sections for details.

3.5.1 Handling NetCDF files

PISM takes one or more NetCDF files as input, performs some computation, and then produces one or more NetCDF files as output. However, other tools are usually needed to help to extract meaning from NetCDF files, and yet more NetCDF tools help with creating PISM input files or post-processing PISM output files.

Here we list a number of NetCDF tools that can be useful in preparing input data for use with PISM and post-processing results; see [Table 3.21](#).

Table 3.21: A selection of tools for viewing and modifying NetCDF files.

Tool	Function
<code>ncdump</code> (part of NetCDF)	dump binary NetCDF as <code>.cdl</code> (text) file
<code>ncgen</code> (part of NetCDF)	convert <code>.cdl</code> file to binary NetCDF
<code>ncview</code>	quick graphical view
<code>CDO</code>	Climate Data Operators; command-line tools, including conservative re-mapping
<code>IDV</code>	more complete visualization
<code>NCO</code>	NetCDF Operators; command-line tools for pre- and post-processing
<code>NCL</code>	NCAR Command Language
<code>PyNGL</code>	Python version of NCL

The PISM authors use `ncview` and “`ncdump -h`” for quick visualization and metadata examination. `NCO` has powerful command-line manipulation of NetCDF files, but requires some learning. Another such command-line tool is `CDO`, but to use `CDO` on PISM files first run the script `nc2cdo.py`, from the `util/` PISM directory, on the file to fix the metadata so that `CDO` will understand the mapping. Finally, Python scripts using the `netcdf4-python` package (see [Installing PISM](#)) are often the best way to non-trivially change a NetCDF file or make publishable figures from it. MATLAB also has good NetCDF I/O capabilities.

See [Table 3.1](#) in section *A hierarchy of simplifying assumptions for grounded ice flow* for an overview on the data necessary for modeling. For more information on the format of input files for PISM, see section *Initialization and bootstrapping*.

3.5.2 Input and output

PISM is a program that reads NetCDF files and then outputs NetCDF files. [Table 3.22](#) summarizes command-line options controlling the most basic ways to input and output NetCDF files when starting and ending PISM runs.

Table 3.22: Basic NetCDF input and output options

Option	Description
<code>-i</code>	Chooses a PISM output file (NetCDF format) to initialize or restart from. See section <i>Initialization and bootstrapping</i> .
<code>-bootstrap</code>	Bootstrap from the file set using <code>-i</code> using heuristics to “fill in” missing fields. See section <i>Initialization and bootstrapping</i> .
<code>-dontreadSSAvels</code>	Turns off reading the <code>ubar_ssa</code> and <code>vbar_ssa</code> velocities saved by a previous run using the <code>ssa</code> or <code>ssa+sia</code> stress balance (see section <i>Choosing the stress balance</i>).
<code>-o</code>	Chooses the output file name. Default name is <code>unnamed.nc</code> .
<code>-o_size size_keyword</code>	Chooses the size of the output file to produce. Possible sizes are <ul style="list-style-type: none"> • <code>none</code> (<i>no</i> output file at all), • <code>small</code> (only variables necessary to restart PISM), • <code>medium</code> (the default, includes diagnostic quantities listed in the configuration parameter <code>output.sizes.medium</code>, if they are available in the current PISM setup), • <code>big_2d</code> (same as <code>medium</code>, plus variables listed in <code>output.sizes.big_2d</code>), and • <code>big</code> (same as <code>big_2d</code>, plus variables listed in <code>output.sizes.big</code>).

Table 3.23 lists the controls on what is printed to the standard output. Note the `-help` and `-usage` options for getting help at the command line.

Table 3.23: Options controlling PISM’s standard output

Option	Description
<code>-help</code>	Brief descriptions of the many PISM and PETSc options. The run occurs as usual according to the other options. (The option documentation does not get listed if the run didn’t get started properly.) Use with a pipe into <code>grep</code> to get usefully-filtered information on options, for example <code>pisms -help grep cold</code> .
<code>-info</code>	Gives information about PETSc operations during the run.
<code>-list_diagnostics</code>	Prints a list of all available diagnostic outputs (time series and spatial) for the run with the given options. Stops run after printing the list.
<code>-log_summary</code>	At the end of the run gives a performance summary and also a synopsis of the PETSc configuration in use.
<code>-options_left</code>	At the end of the run shows an options table which will indicate if a user option was not read or was misspelled.
<code>-usage</code>	Short summary of PISM executable usage, without listing all the options, and without doing the run.
<code>-verbose</code>	Increased verbosity of standard output. Usually given with an integer level; 0,1,2,3,4,5 are allowed. If given without argument then sets level 3, while <code>-verbose 2</code> is the default (i.e. equivalent to no option). At the extremes, <code>-verbose 0</code> produces no stdout at all, <code>-verbose 1</code> prints only warnings and a few high priority messages, and <code>-verbose 5</code> spews a lot of usually-undesirable stuff. <code>-verbose 3</code> output regarding initialization may be useful.
<code>-version</code>	Show version numbers of PETSc and PISM.

The following sections describe more input and output options, especially related to saving quantities during a run, or adding to the “diagnostic” outputs of PISM.

PISM file I/O performance

When working with fine grids¹, the time PISM spends writing output files, spatially-varying diagnostic files, or backup files can become significant.

It turns out that it is a lot faster to read and write files using the `t,y,x,z` storage order, as opposed to the more convenient (e.g. for NetCDF tools) `t,z,y,x` order. The reason is that PISM uses the `y,x,z` order internally,² and therefore writing an array in a different order is an inherently-expensive operation.

You can, however, choose any one of the three supported output orders using the `-o_order` option with one of `xyz`, `yxz`, and `zyx` as the argument.

To transpose dimensions in an existing file, use the `ncpdq` (“permute dimensions quickly”) tool from the [NCO](#) suite. For example, run

```
ncpdq -a t,z,zb,y,x bad.nc good.nc
```

to turn `bad.nc` (with any inconvenient storage order) into `good.nc` using the `t,z,y,x` order.

PISM also supports NetCDF-4 parallel I/O, which gives better performance in high-resolution runs and avoids NetCDF-3 file format limitations. (In a NetCDF-3 file a variable record cannot exceed 4 gigabytes.) Build PISM with parallel NetCDF-4 and use `-o_format netcdf4_parallel` to enable this code.

In addition to `-o_format netcdf4_parallel` and `netcdf3` (default) modes, PISM can be built with `PnetCDF` for best I/O performance. The option `-o_format pnetcdf` turns “on” `PnetCDF` I/O code. (`PnetCDF` seems to be somewhat fragile, though, so use at your own risk.)

3.5.3 Saving time series of scalar diagnostic quantities

It is also possible to save time-series of certain scalar diagnostic quantities using a combination of the options `-ts_file`, `-ts_times`, and `-ts_vars`. For example,

```
pismr -i foo.nc -y 1e4 -o output.nc -ts_file time-series.nc \  
-ts_times 0:1:1e4 -ts_vars ice_volume_glacierized,ice_area_glacierized_grounded
```

will run for 10000 years, saving total ice volume and grounded ice area to `time-series.nc` yearly. See tables [Table 3.24](#) for the list of options and [Scalar time-series](#) for the full list of supported time-series.

Note that, similarly to the snapshot-saving code (section [Saving re-startable snapshots of the model state](#)), this mechanism does not affect adaptive time-stepping. Here, however, PISM will save exactly the number of time-series records requested.

Omitting the `-ts_vars` option makes PISM save *all* available variables listed in [Scalar time-series](#). Because scalar time-series take minimal storage space, compared to spatially-varying data, this is usually a reasonable choice. Run PISM with the `-list_diagnostics` option to see the list of all available time-series.

If the file `foo.nc`, specified by `-ts_file foo.nc`, already exists then by default the existing file will be moved to `foo.nc~` and the new time series will go into `foo.nc`. To append the time series onto the end of the existing file, use option `-ts_append`.

PISM buffers time-series data and writes it at the end of the run, once 10000 values are stored, or when an `-extra_file` is saved, whichever comes first. Sending an `USR1` (or `USR2`) signal to a PISM process flushes these buffers, making it possible to monitor the run. (See section [Signals, to control a running PISM model](#) for more about PISM’s signal handling.)

¹ For example, resolutions of 2km and higher on the whole-Greenland scale.

² This is not likely to change.

Table 3.24: Command-line options controlling saving scalar time-series

Option	Description
-ts_file	Specifies the file to save to.
-ts_times	Specifies times to save at as a MATLAB-style range $a : \Delta t : b$, a comma-separated list, or a keyword (hourly, daily, monthly, yearly). See section <i>Saving time series of spatially-varying diagnostic quantities</i> .
-ts_vars	Comma-separated list of variables. Omitting this option is equivalent to listing the <i>all</i> variables.
-ts_append	Append time series to file if it already exists. No effect if file does not yet exist.

Besides the above information on usage, here are comments on the physical significance of several scalar diagnostics:

- For each variable named `..._flux`, positive values mean ice sheet mass gain.
- PISM reports ice volume, ice mass, and several other quantities for “glacierized” areas. These quantities do not include contributions from areas where the ice thickness is equal to or below the value of the configuration parameter `output.ice_free_thickness_standard` (in meters). Corresponding quantities without the suffix *do* include areas with a thin, “seasonal” ice cover.
- The `sub_shelf_ice_flux` may be non-zero even if `ice_area_glacierized_shelf` (floating ice area) is zero. This is due to the fact that during time-stepping fluxes are computed before calving is applied, and the ice area is computed *after* calving. Hence ice that is calved off experiences top-surface and basal fluxes, but does not contribute to the reported area. This is a small error that approaches zero as the grid is refined. In this case `sub_shelf_ice_flux` should be added to the calving flux during post-processing.¹
- Ice volume and area are computed and then split among floating and grounded portions:

$$\begin{array}{lll} \text{ice_volume_glacierized} & \mapsto & (\text{ice_volume_glacierized_shelf}, \\ \text{ice_volume_glacierized_grounded}) & \text{while} & \text{ice_area_glacierized} \mapsto \\ (\text{ice_area_glacierized_shelf}, \text{ice_area_glacierized_grounded}). \end{array}$$
The volumes have units m^3 and the areas have units m^2 .
- The thermodynamic state of the ice sheet can be assessed, in part, by the amount of cold or temperate (“temp”) ice. Thus there is another splitting: `ice_volume_glacierized` \mapsto (`ice_volume_glacierized_cold`, `ice_volume_glacierized_temperate`) and `ice_area_glacierized` \mapsto (`ice_area_glacierized_cold_base`, `ice_area_glacierized_temperate_base`).
- If a PISM input file contains the proj4 global attribute with a PROJ.4 string defining the projection then PISM computes corrected cell areas using this information, grid parameters, and the WGS84 reference ellipsoid. This yields areas and volumes with greater accuracy.
- The sea-level-relevant ice volume `slvol` is the total grounded ice volume minus the amount of ice, that, in liquid form, would fill up the regions with bedrock below sea level, if this ice were removed. That is, `slvol` is the sea level rise potential of the ice sheet at that time. The result is reported in sea-level equivalent, i.e. meters of sea level rise.
- Fields `max_diffusivity` and `max_hor_vel` relate to PISM time-stepping. These quantities appear in per-time-step form in the standard output from PISM (i.e. at default verbosity). `max_diffusivity` determines the length of the mass continuity sub-steps for the SIA stress balance (sub-)model. `max_hor_vel` determines the CFL-type restriction for mass continuity and conservation of energy contributions of the SSA stress balance (i.e. sliding) velocity.

¹ This will be fixed in a later release of PISM.

3.5.4 Saving time series of spatially-varying diagnostic quantities

Sometimes it is useful to have PISM save a handful of diagnostic *maps* at some interval like every 10 years or even every month. One can use snapshots (section [Saving re-startable snapshots of the model state](#)), but doing so can easily fill your hard-drive because snapshots are complete (i.e. re-startable) model states. Sometimes you want a *subset* of model variables saved frequently in an output file.

Use options `-extra_file`, `-extra_times`, and `-extra_vars` for this. For example,

```
pismr -i foo.nc -y 10000 -o output.nc -extra_file extras.nc \
      -extra_times 0:10:1e4 -extra_vars velsurf_mag,velbase_mag
```

will run for 10000 years, saving the magnitude of horizontal velocities at the ice surface and at the base of ice every 10 years. Times are specified using a comma-separated list or a MATLAB-style range. See [Table 3.25](#) for all the options controlling this feature. The section [Spatially-variable fields](#) list all the variable choices.

Note that options `-extra_times`, `-save_times`, `-ts_times` take *dates* if a non-trivial calendar is selected. For example,

```
pismr ... -extra_times 10           # every 10 years
pismr ... -extra_times 2days       # every 2 days
pismr ... -calendar gregorian -extra_times 1-1-1:daily:11-1-1 # daily for 10 years
pismr ... -calendar gregorian -extra_times daily -ys 1-1-1 -ye 11-1-1
pismr ... -calendar gregorian -extra_times 2hours -ys 1-1-1 -ye 1-2-1
```

The step in the range specification can have the form `Nunit`, for example `5days`. Units based on “months” and “years” are not supported if a non-trivial calendar is selected.

In addition to specifying a constant step in `-extra_times a:step:b` one can save every hour, day, month, or every year by using `hourly`, `daily`, `monthly` or `yearly` instead of a number; for example

```
pismr -i foo.nc -y 100 -o output.nc -extra_file extras.nc \
      -extra_times 0:monthly:100 -extra_vars dHdt
```

will save the rate of change of the ice thickness every month for 100 years. With `-calendar none` (the default), “monthly” means “every $\frac{1}{12}$ of the year”, and “yearly” is “every $3.14 \dots \times 10^7$ ” seconds, otherwise PISM uses month lengths computed using the selected calendar.

It is frequently desirable to save diagnostic quantities at regular intervals for the whole duration of the run; options `-extra_times`, `-ts_times`, and `-save_times` provide a shortcut. For example, use `-extra_times yearly` to save at the end of every year.

This is especially useful when using a climate forcing file to set run duration:

```
pismr -i foo.nc -surface given -surface_given_file climate.nc \
      -calendar gregorian -time_file climate.nc \
      -extra_times monthly -extra_file ex.nc -extra_vars thk
```

will save ice thickness at the end of every month while running PISM for the duration of climate forcing data in `climate.nc`.

Times given using `-extra_times` describe the reporting intervals by giving the endpoints of these reporting intervals. The save itself occurs at the end of each interval. This implies, for example, that `0:1:10` will produce 10 records at times 1, ..., 10 and *not* 11 records.

If the file `foo.nc`, specified by `-extra_file foo.nc`, already exists then by default the existing file will be moved to `foo.nc~` and the new time series will go into `foo.nc`. To append the time series onto the end of the existing file, use option `-extra_append`.

The list of available diagnostic quantities depends on the model setup. For example, a run with only one vertical grid level in the bedrock thermal layer will not be able to save `litho_temp`, an SIA-only run does not use a basal yield stress model and so will not provide `tauc`, etc. To see which quantities are available in a particular setup, use the `-list_diagnostics` option, which prints the list of diagnostics and stops.

The `-extra_file` mechanism modifies PISM's adaptive time-stepping scheme so as to step to, and save at, *exactly* the times requested. By contrast, as noted in subsection *Saving time series of scalar diagnostic quantities*, the `-ts_file` mechanism does not alter PISM's time-steps and instead uses linear interpolation to save at the requested times in between PISM's actual time-steps.

Table 3.25: Command-line options controlling extra diagnostic output

Option	Description
<code>-extra_file</code>	Specifies the file to save to; should be different from the output (<code>-o</code>) file.
<code>-extra_times</code>	Specifies times to save at either as a MATLAB-style range $a : \Delta t : b$ or a comma-separated list.
<code>-extra_vars</code>	Comma-separated list of variables
<code>-extra_split</code>	Save to separate files, similar to <code>-save_split</code> .
<code>-extra_append</code>	Append variables to file if it already exists. No effect if file does not yet exist, and no effect if <code>-extra_split</code> is set.

3.5.5 Saving re-startable snapshots of the model state

Sometimes you want to check the model state every 1000 years, for example. One possible solution is to run PISM for a thousand years, have it save all the fields at the end of the run, then restart and run for another thousand, and etc. This forces the adaptive time-stepping mechanism to stop *exactly* at multiples of 1000 years, which may be desirable in some cases.

If saving exactly at specified times is not critical, then use the `-save_file` and `-save_times` options. For example,

```
pismr -i foo.nc -y 10000 -o output.nc -save_file snapshots.nc \
      -save_times 1000:1000:10000
```

starts a PISM evolution run, initializing from `foo.nc`, running for 10000 years and saving snapshots to `snapshots.nc` at the first time-step after each of the years 1000, 2000, ..., 10000.

We use a MATLAB-style range specification, $a : \Delta t : b$, where $a, \Delta t, b$ are in years. The time-stepping scheme is not affected, but as a consequence we do not guarantee producing the exact number of snapshots requested if the requested save times have spacing comparable to the model time-steps. This is not a problem in the typical case in which snapshot spacing is much greater than the length of a typical time step.

It is also possible to save snapshots at intervals that are not equally-spaced by giving the `-save_times` option a comma-separated list. For example,

```
pismr -i foo.nc -y 10000 -o output.nc -save_file snapshots.nc \
      -save_times 1000,1500,2000,5000
```

will save snapshots on the first time-step after years 1000, 1500, 2000 and 5000. The comma-separated list given to the `-save_times` option can be at most 200 numbers long.

If `snapshots.nc` was created by the command above, running

```
pismr -i snapshots.nc -y 1000 -o output_2.nc
```

will initialize using the last record in the file, at about 5000 years. By contrast, to restart from 1500 years (for example) it is necessary to extract the corresponding record using `ncks`


```
ncks -d t,1500years snapshots.nc foo.nc
```

and then restart from `foo.nc`. Note that `-d t,N` means “extract the N -th record” (counting from zero). So, this command is equivalent to

```
ncks -d t,1 snapshots.nc foo.nc
```

Also note that the second snapshot will probably be *around* 1500 years and `ncks` handles this correctly: it takes the record closest to 1500 years.

By default re-startable snapshots contain only the variables needed for restarting PISM. Use the command-line option `-save_size` to change what is saved.

Another possible use of snapshots is for restarting runs on a batch system which kills jobs which go over their allotted time. Running PISM with options `-y 1500 -save_times 1000:100:1400` would mean that if the job is killed before completing the whole 1500 year run, we can restart from near the last multiple of 100 years. Restarting with option `-ye` would finish the run on the desired year.

When running PISM on such a batch system it can also be useful to save re-startable snapshots at equal wall-clock time (as opposed to model time) intervals by adding the “`-backup_interval (hours)`” option.

Caution: If the wall-clock limit is equal to N times backup interval for a whole number N PISM will likely get killed while writing the last backup.

It is also possible to save snapshots to separate files using the `-save_split` option. For example, the run above can be changed to

```
pismr -i foo.nc -y 10000 -o output.nc -save_file snapshots \
      -save_times 1000,1500,2000,5000 -save_split
```

for this purpose. This will produce files called `snapshots-year.nc`. This option is generally faster if many snapshots are needed, apparently because of the time necessary to reopen a large file at each snapshot when `-save_split` is not used. Note that tools like `NCO` and `ncview` usually behave as desired with wildcards like “`snapshots-*.nc`”.

Table 3.26 lists the options related to saving snapshots of the model state.

Table 3.26: Command-line options controlling saving snapshots of the model state.

Option	Description
<code>-save_file</code>	Specifies the file to save to.
<code>-save_times</code>	Specifies times at which to save snapshots, by either a MATLAB-style range $a : \Delta t : b$ or a comma-separated list.
<code>-save_split</code>	Separate the snapshot output into files named <code>snapshots-year.nc</code> . Faster if you are saving more than a dozen or so snapshots.
<code>-save_size [none,small,medium,big,big_2d]</code>	Similar to <code>o_size</code> , changes the “size” of the file (or files) written; the default is “small”

3.5.6 Run-time diagnostic viewers

Basic graphical views of the changing state of a PISM ice model are available at the command line by using options listed in Table 3.27. All the quantities listed in *Spatially-variable fields* are available. Additionally, a couple of

diagnostic quantities are *only* available as run-time viewers; these are shown in table [Table 3.28](#).

Table 3.27: Options controlling run-time diagnostic viewers

Option	Description
<code>-view</code>	Turns on map-plane views of one or several variables, see Spatially-variable fields .
<code>-view_size</code> (number)	desired viewer size, in pixels
<code>-display</code>	The option <code>-display :0</code> seems to frequently be needed to let PETSc use Xwindows when running multiple processes. It must be given as a <i>final</i> option, after all the others.

The option `-view` shows map-plane views of 2D fields and surface and basal views of 3D fields (see [Spatially-variable fields](#)); for example:

```
pismr -i input.nc -y 1000 -o output.nc -view thk,tempsurf
```

shows ice thickness and ice temperature at the surface.

Table 3.28: Special run-time-only diagnostic viewers

Option	Description
<code>-ssa_view_nuh</code>	log base ten of nuH, only available if the finite-difference SSA solver is active.
<code>-ssa_nuh_viewer_size</code> (number)	Adjust the viewer size.
<code>-ksp_monitor_draw</code>	Iteration monitor for the Krylov subspace routines (KSP) in PETSc. Residual norm versus iteration number.

3.5.7 PISM's configuration parameters and how to change them

PISM's behavior depends on values of many flags and physical parameters (see [Configuration parameters](#) for details). Most of parameters have default values¹ which are read from the configuration file `pism_config.nc` in the `lib` sub-directory.

It is possible to run PISM with an alternate configuration file using the `-config` command-line option:

```
pismr -i foo.nc -y 1000 -config my_config.nc
```

The file `my_config.nc` has to contain *all* of the flags and parameters present in `pism_config.nc`.

The list of parameters is too long to include here; please see the [Configuration parameters](#) for an automatically-generated table describing them.

Some command-line options *set* configuration parameters; some PISM executables have special parameter defaults. To examine what parameters were used in a particular run, look at the attributes of the `pism_config` variable in a PISM output file.

Managing parameter studies

Keeping all PISM output files in a parameter study straight can be a challenge. If the parameters of interest were controlled using command-line options then one can use `ncdump -h` and look at the `history` global attribute.

Alternatively, one can change parameter values by using an “overriding” configuration file. The `-config_override` command-line option provides this alternative. A file used with this option can have a subset of the configuration flags

¹ For `pismr`, grid parameters `Mx`, `My`, that must be set at bootstrapping, are exceptions.

and parameters present in `pism_config.nc`. Moreover, PISM adds the `pism_config` variable with values used in a run to the output file, making it easy to see which parameters were used.

Here’s an example. Suppose we want to compare the dynamics of an ice-sheet on Earth to the same ice-sheet on Mars, where the only physical change was to the value of the acceleration due to gravity. Running

```
pismr -i input.nc -y 1e5 -o earth.nc <other PISM options>
```

produces the “Earth” result, since PISM’s defaults correspond to this planet. Next, we create `mars.cdl` containing the following:

```
netcdf mars {
    variables:
        byte pism_overrides;
        pism_overrides:constants.standard_gravity = 3.728;
        pism_overrides:constants.standard_gravity_doc = "m s-2; standard gravity on Mars";
}
```

Notice that the variable name is `pism_overrides` and not `pism_config` above. Now

```
ncgen -o mars_config.nc mars.cdl
pismr -i input.nc -y 1e5 -config_override mars_config.nc -o mars.nc <other PISM options>
```

will create `mars.nc`, the result of the “Mars” run. Then we can use `ncdump` to see what was different about `mars.nc`:

```
ncdump -h earth.nc | grep pism_config: > earth_config.txt
ncdump -h mars.nc | grep pism_config: > mars_config.txt
diff -U 1 earth_config.txt mars_config.txt
--- earth_config.txt 2015-05-08 12:44:43.000000000 -0800
+++ mars_config.txt 2015-05-08 12:44:51.000000000 -0800
@@ -734,3 +734,3 @@
         pism_config:ssafd_relative_convergence_units = "1" ;
-        pism_config:constants.standard_gravity_doc = "acceleration due to gravity on Earth,
-geoid" ;
+        pism_config:constants.standard_gravity_doc = "m s-2; standard gravity on Mars" ;
+        pism_config:constants.standard_gravity_type = "scalar" ;
@@ -1057,3 +1057,3 @@
         pism_config:ssafd_relative_convergence = 0.0001 ;
-        pism_config:constants.standard_gravity = 9.81 ;
+        pism_config:constants.standard_gravity = 3.728 ;
+        pism_config:start_year = 0. ;
```

Saving PISM’s configuration for post-processing

In addition to saving `pism_config` in the output file, PISM automatically adds this variable to all files it writes (snap shots, time series of scalar and spatially-varying diagnostic quantities, and backups). This may be useful for post-processing and analysis of parameter studies as the user has easy access to all configuration options, model choices, etc., without the need to keep run scripts around.

3.5.8 Regridding

FIXME: mention that `-regrid_file` without `-regrid_vars` re-grids model state variables for all selected sub-models.

It is common to want to interpolate a coarse grid model state onto a finer grid or vice versa. For example, one might want to do the EISMINT II experiment on the default grid, producing output `foo.nc`, but then interpolate both the ice thickness and the temperature onto a finer grid. The basic idea of “regridding” in PISM is that one starts over from the

beginning on the finer grid, but one extracts the desired variables stored in the coarse grid file and interpolates these onto the finer grid before proceeding with the actual computation.

The transfer from grid to grid is reasonably general — one can go from coarse to fine or vice versa in each dimension x, y, z — but the transfer must always be done by *interpolation* and never *extrapolation*. (An attempt to do the latter will always produce a PISM error.)

Such “regridding” is done using the `-regrid_file` and `-regrid_vars` commands as in this example: }

```
pisms -eisII A -Mx 101 -My 101 -Mz 201 -y 1000 \
    -regrid_file foo.nc -regrid_vars thk,temp -o bar.nc
```

By specifying regridded variables “`thk,temp`”, the ice thickness and temperature values from the old grid are interpolated onto the new grid. Here one doesn’t need to regrid the bed elevation, which is set identically zero as part of the EISMINT II experiment A description, nor the ice surface elevation, which is computed as the bed elevation plus the ice thickness at each time step anyway.

A slightly different use of regridding occurs when “bootstrapping”, as described in section *Initialization and bootstrapping* and illustrated by example in section *Getting started: a Greenland ice sheet example*.

See Table 3.29 for the regriddable variables using `-regrid_file`. Only model state variables are regriddable, while climate and boundary data generally are not explicitly regriddable. (Bootstrapping, however, allows the same general interpolation as this explicit regrid.)

Table 3.29: Regriddable variables. Use `-regrid_vars` with these names.

Name	Description
age	age of ice
bwat	effective thickness of subglacial melt water
bmelt	basal melt rate
dbdt	bedrock uplift rate
litho_temp	lithosphere (bedrock) temperature
mask	grounded/dragging/floating integer mask, see section <i>Flotation criterion, mask, and sea level</i>
temp	ice temperature
thk	land ice thickness
topg	bedrock surface elevation
enthalpy	ice enthalpy

Here is another example: suppose you have an output of a PISM run on a fairly coarse grid (stored in `foo.nc`) and you want to continue this run on a finer grid. This can be done using `-regrid_file` along with `-bootstrap`:

```
pismr -i foo.nc -bootstrap -Mx 201 -My 201 -Mz 21 -Lz 4000 \
    -regrid_file foo.nc -regrid_vars litho_temp,enthalpy -y 100 -o bar.nc \
    -surface constant
```

In this case all the model-state 2D variables present in `foo.nc` will be interpolated onto the new grid during bootstrapping, which happens first, while three-dimensional variables are filled using heuristics mentioned in section *Initialization and bootstrapping*. Then temperature in bedrock (`litho_temp`) and ice enthalpy (`enthalpy`) will be interpolated from `foo.nc` onto the new grid during the regridding stage, overriding values set at the bootstrapping stage. All of this, bootstrapping and regridding, occurs before the first time step.

By default PISM checks the grid overlap and stops if the current computational domain is not a subset of the one in a `-regrid_file`. It is possible to disable this check and allow constant extrapolation: use the option `-allow_extrapolation`.

For example, in a PISM run the ice thickness has to be lower than the vertical extent of the computational domain. If the ice thickness exceeds `Lz` PISM saves the model state and stops with an error message.

```
pismr -i input.nc -bootstrap -Mz 11 -Lz 1000 -z_spacing equal \
-y 3e3 \
-o too-short.nc
PISM ERROR: Ice thickness exceeds the height of the computational box (1000.0000 m).
The model state was saved to 'too-short_max_thickness.nc'.
To continue this simulation, run with
-i too-short_max_thickness.nc -bootstrap -regrid_file too-short_max_thickness.nc \
-allow_extrapolation -Lz N [other options]
where N > 1000.0000.
```

Regridding with extrapolation makes it possible to extend the vertical grid and continue a simulation like this one — just follow the instructions provided in the error message.

3.5.9 Signals, to control a running PISM model

Ice sheet model runs sometimes take a long time, so the state of a run may need checking. Sometimes the run needs to be stopped, but with the possibility of restarting. PISM implements these behaviors using “signals” from the POSIX standard, included in Linux and most flavors of Unix. Table 3.30 summarizes how PISM responds to signals. A convenient form of kill, for Linux users, is `pkill` which will find processes by executable name. Thus “`pkill -USR1 pismr`” might be used to send all PISM processes the same signal, avoiding an explicit list of *PIDs*.

Table 3.30: Signalling running PISM processes. “*PIDs*” stands for list of all identifiers of the PISM processes.

Command	Signal	PISM behavior
<code>kill -KILL PIDs</code>	SIGKILL	Terminate with extreme prejudice. PISM cannot catch it and no state is saved.
<code>kill -TERM PIDs</code>	SIGTERM	End process(es), but save the last model state in the output file, using <code>-o</code> name or default name as normal. Note that the history string in the output file will contain an “EARLY EXIT caused by signal SIGTERM” indication.
<code>kill -USR1 PIDs</code>	SIGUSR1	Process(es) will continue after saving the model state at the end of the current time step, using a file name including the current model year. Time-stepping is not altered. Also flushes output buffers of scalar time-series.
<code>kill -USR2 PIDs</code>	SIGUSR2	Just flush time-series output buffers.

Here is an example. Suppose we start a long verification run in the background, with standard out redirected into a file:

```
pismv -test G -Mz 101 -y 1e6 -o testGmillion.nc >> log.txt &
```

This run gets a Unix process id, which we assume is “8920”. (Get it using `ps` or `pgrep`.) If we want to observe the run without stopping it we send the USR1 signal:

```
kill -USR1 8920
```

(With `pkill` one can usually type “`pkill -usr1 pismv`”.) Suppose it happens that we caught the run at year 31871.5. Then, for example, a NetCDF file `pismv-31871.495.nc` is produced. Note also that in the standard out log file `log.txt` the line

```
caught signal SIGUSR1: Writing intermediate file ... and flushing time series.
```

appears around that time step. Suppose, on the other hand, that the run needs to be stopped. Then a graceful way is

```
kill -TERM 8920
```

because the model state is saved and can be inspected.

3.5.10 Understanding adaptive time-stepping

At each time step the PISM standard output includes “flags” and then a summary of the model state using a few numbers. A typical example is

```
v$Eh diffusivity (dt=0.83945 in 2 substeps; av dt_sub_mass_cont=0.41972)
S -124791.571: 3.11640 2.25720 3.62041 18099.93737
y SSA: 3 outer iterations, ~17.0 KSP iterations each
```

The characters “v\$Eh” at the beginning of the flags line, the first line in the above example, give a very terse description of which physical processes were modeled in that time step. Here “v” means that a stress balance was solved to compute the velocity. Then the enthalpy was updated (“E”) and the ice thickness and surface elevation were updated (“h”). The rest of the flags line looks like

```
diffusivity (dt=0.83945 in 2 substeps; av dt_sub_mass_cont=0.41972)
```

Recall that the PISM time step is determined by an adaptive mechanism. Stable mass conservation and conservation of energy solutions require such an adaptive time-stepping scheme [18]. The first character we see here, namely “diffusivity”, is the adaptive-timestepping “reason” flag. See Table 3.31. We also see that there was a major time step of 0.83945 model years divided into 2 substeps of about 0.42 years. The `-skip` option enables this mechanism, while `-skip_max` sets the maximum number of such substeps. The adaptive mechanism may choose to take fewer substeps than `-skip_max` so as to satisfy certain numerical stability criteria, however.

The second line in the above, the line which starts with “S”, is the summary. Its format, and the units for these numbers, is simple and is given by a couple of lines printed near the beginning of the standard output for the run:

P	YEAR:	ivol	iarea	max_diffusivity	max_hor_vel
U	years	10 ⁶ _km ³	10 ⁶ _km ²	m ² s ⁻¹	m/year

That is, in each summary we have the total ice volume, total ice area, maximum diffusivity (of the SIA mass conservation equation), and maximum horizontal velocity (i.e. $\max(\max(|u|), \max(|v|))$).

The third line of the above example shows that the SSA stress balance was solved. Information on the number of nonlinear (outer) and linear (inner) iterations is provided [17].

Table 3.31: Meaning of the adaptive time-stepping “reason” flag in the standard output flag line.

PISM output	Active adaptive constraint or PISM sub-system that limited time-step size
3D CFL	three-dimensional CFL for temperature/age advection [18]
diffusivity	diffusivity for SIA mass conservation [18], [95]
end of the run	end of prescribed run time
max	maximum allowed Δt applies; set with <code>-max_dt</code>
internal (derived class)	maximum Δt was temporarily set by a derived class
2D CFL	2D CFL for mass conservation in SSA regions (upwinded; [17])
-ts... reporting	the <code>-ts_times</code> option and the configuration flag <code>time_stepping.hit_ts_times</code> ; see section <i>Saving time series of scalar diagnostic quantities</i>
-extra... reporting	the <code>-extra_times</code> option; see section <i>Saving time series of spatially-varying diagnostic quantities</i>
surface	a surface or an atmosphere model
ocean	an ocean model
hydrology	a hydrology model stability criterion, see section <i>Subglacial hydrology</i>
BTU	time-the bedrock thermal layer model, see section <i>Modeling conservation of energy</i>
eigencalving	the eigen-calving model, see section <i>Calving</i>

Table 3.32: Options controlling time-stepping

Option	Description
<code>-adapt_ratio</code>	Adaptive time stepping ratio for the explicit scheme for the mass balance equation.
<code>-max_dt (years)</code>	The maximum time-step in years. The adaptive time-stepping scheme will make the time-step shorter than this as needed for stability, but not longer.
<code>-skip</code>	Enables time-step skipping, see below.
<code>-skip_max</code>	Number of mass-balance steps, including SIA diffusivity updates, to perform before temperature, age, and SSA stress balance computations are done. This is only effective if the time step is being limited by the diffusivity time step restriction associated to mass continuity using the SIA. The maximum recommended value for <code>-skip_max</code> is, unfortunately, dependent on the context. The temperature field should be updated when the surface changes significantly, and likewise the basal sliding velocity if it comes (as it should) from the SSA calculation.
<code>-timestep_hit_multiples (years)</code>	Hit multiples of the number of model years specified. For example, if stability criteria require a time-step of 11 years and the <code>-timestep_hit_multiples 3</code> option is set, PISM will take a 9 model year long time step. This can be useful to enforce consistent sampling of periodic climate data.

3.5.11 Balancing the books

2D diagnostics

PISM provides the following 2D diagnostics to keep track of mass conservation.¹

¹ See *Diagnostic quantities* for the full list of diagnostics.

All these are computed as time-averaged fluxes over reporting intervals. Positive values correspond to mass gain.

At every grid point (point-wise) we have

```
tendency_of_ice_amount = (tendency_of_ice_amount_due_to_flow +
                           tendency_of_ice_amount_due_to_conservation_error +
                           tendency_of_ice_amount_due_to_surface_mass_balance +
                           tendency_of_ice_amount_due_to_basal_mass_balance +
                           tendency_of_ice_amount_due_to_discharge)
```

Here “ice amount” is “ice mass per unit area”, units of kg/m^2 .

Alternatively one can request the same 2D diagnostics in terms of ice mass:

```
tendency_of_ice_mass = (tendency_of_ice_mass_due_to_flow +
                         tendency_of_ice_mass_due_to_conservation_error +
                         tendency_of_ice_mass_due_to_surface_mass_balance +
                         tendency_of_ice_mass_due_to_basal_mass_balance +
                         tendency_of_ice_mass_due_to_discharge)
```

Use a shortcut

```
pismr -extra_file ex.nc -extra_times N -extra_vars amount_fluxes,...
```

to save fluxes in terms of “ice amount” and

```
pismr -extra_file ex.nc -extra_times N -extra_vars mass_fluxes,...
```

to save fluxes in terms of “ice mass.”

Comments

- `tendency_of_ice_mass_due_to_flow` is the change in ice mass corresponding to flux divergence
- `tendency_of_ice_mass_due_to_conservation_error` is the artificial change in ice mass needed to preserve non-negativity of ice thickness.
- `tendency_of_ice_mass_due_to_surface_mass_balance` is the change due to the surface mass balance; note that this is *not* the same as the provided SMB: in ablation areas this is the *effective* mass balance taking into account the amount of ice present
- `tendency_of_ice_mass_due_to_basal_mass_balance` is the *effective* change due to basal (grounded and sub-shelf) melting
- `tendency_of_ice_mass_due_to_discharge` combines changes due to calving and frontal melt

Scalar diagnostics

Diagnostics listed above are also available as scalar diagnostics, integrated over the whole computational domain.

```
tendency_of_ice_mass = (tendency_of_ice_mass_due_to_influx +
                         tendency_of_ice_mass_due_to_conservation_error +
                         tendency_of_ice_mass_due_to_basal_mass_balance +
                         tendency_of_ice_mass_due_to_surface_mass_balance +
                         tendency_of_ice_mass_due_to_discharge)
```

Comments

- `tendency_of_ice_mass_due_to_influx` is the integral of $-\nabla \cdot Q$ over the computational domain. This should be zero (up to the effect of rounding errors) in simulations that *do not* use Dirichlet boundary conditions for ice thickness. Prescribing ice thickness creates sources and sinks, and this diagnostic describes their influence.
- `tendency_of_ice_mass_due_to_conservation_error` should be zero (or close to zero) in most simulations

3.5.12 PETSc options for PISM users

All PETSc programs including PISM accept command line options which control how PETSc distributes jobs among parallel processors, how it solves linear systems, what additional information it provides, and so on. The PETSc manual [94] is the complete reference on these options. We list some here that are useful to PISM users. They can be mixed in any order with PISM options.

Both for PISM and PETSc options, there are ways of avoiding the inconvenience of long commands with many runtime options. Obviously, and as illustrated by examples in the previous sections, shell scripts can be set up to run PISM. But PETSc also provides two mechanisms to give runtime options without retyping at each run command.

First, the environment variable `PETSC_OPTIONS` can be set. For example, a sequence of runs might need the same refined grid, and you might want to know if other options are read, ignored, or misspelled. Set (in Bash):

```
export PETSC_OPTIONS="-Mx 101 -My 101 -Mz 51 -options_left"
```

The runs

```
pismv -test F -y 100
pismv -test G -y 100
```

then have the same refined grid in each run, and the runs report on which options were read.

Alternatively, the file `.petsrc` is always read, if present, from the directory where PISM (i.e. the PETSc program) is started. It can have a list of options, one per line. In theory, these two PETSc mechanisms (`PETSC_OPTIONS` and `.petsrc`) can be used together.

Now we address controls on how PETSc solves systems of linear equations, which uses the PETSc “KSP” component (Krylov methods). Such linear solves are needed each time the nonlinear SSA stress balance equations are used (e.g. with the option `-stress_balance ssa -ssa_method fd`).

Especially for solving the SSA equations with high resolution on multiple processors, it is recommended that the option `-ssafd_ksp_rtol` be set lower than its default value of 10^{-5} . For example,

```
mpiexec -n 8 ssa_testi -Mx 3 -My 769 -ssa_method fd
```

may fail to converge on a certain machine, but adding `“-ssafd_ksp_rtol 1e-10”` works fine.

There is also the question of solver *type*, using option `-ssafd_ksp_type`. Based on one processor evidence from `ssa_testi`, the following are possible choices in the sense that they work and allow convergence at some reasonable rate: `cg`, `bicg`, `gmres`, `bcgs`, `cgs`, `tfqmr`, `tcqmr`, and `cr`. It appears `bicg`, `gmres`, `bcgs`, and `tfqmr`, at least, are all among the best. The default is `gmres`.

Actually the KSP uses preconditioning. This aspect of the solve is critical for parallel scalability, but it gives results which are dependent on the number of processors. The preconditioner type can be chosen with `-ssafd_pc_type`. Several choices are possible, but for solving the ice stream and shelf equations we recommend only `bjacobi`, `ilu`, and `asm`. Of these it is not currently clear which is fastest; they are all about the same for `ssa_testi` with high tolerances

(e.g. `-ssa_rtol 1e-7 -ssaafd_ksp_rtol 1e-12`). The default (as set by PISM) is `bjacobi`. To force no preconditioning, which removes processor-number-dependence of results but may make the solves fail, use `-ssaafd_pc_type none`.

For the full list of PETSc options controlling the SSAFD solver, run

```
ssa_testi -ssa_method fd -help | grep ssaafd_ | less
```

3.5.13 Utility and test scripts

In the `test/` and `util/` subdirectories of the PISM directory the user will find some python scripts and one Matlab script, listed in Table 3.33. The python scripts are all documented at the *Packages* tab on the [PISM Source Code Browser](#). The Python scripts all take option `--help`.

Table 3.33: Some scripts which help in using PISM

Script	Function
<code>test/vfnw.py</code>	Organizes the process of verifying PISM. Specifies standard refinement paths for each of the tests (section <i>Verification</i>).
<code>test/vnreport.py</code>	Automates the creation of convergence graphs like figures Fig. 3.25 – Fig. 3.29.
<code>util/fill_missing.py</code>	Uses an approximation to Laplace’s equation $\nabla^2 u = 0$ to smoothly replace missing values in a two-dimensional NetCDF variable. The “hole” is filled with an average of the boundary non-missing values. Depends on <code>netcdf4-python</code> and <code>scipy</code> Python packages.
<code>util/flowline.py</code>	See section <i>Using PISM for flow-line modeling</i> .
<code>util/flowlineslab.py</code>	See section <i>Using PISM for flow-line modeling</i> .
<code>util/check_stationarity.py</code>	Evaluate stationarity of a variable in a PISM <code>-ts_file</code> output.
<code>util/nc2cdo.py</code>	Makes a netCDF file ready for Climate Data Operators (CDO).
<code>util/nc2mat.py</code>	Reads specified variables from a NetCDF file and writes them to an output file in the MATLAB binary data file format <code>.mat</code> , supported by MATLAB version 5 and later. Depends on <code>netcdf4-python</code> and <code>scipy</code> Python packages.
<code>util/nccmp.py</code>	A script comparing variables in a given pair of NetCDF files; used by PISM software tests.
<code>util/pism_config_editor.py</code>	Makes modifying or creating PISM configuration files easier.
<code>util/pism_matlab.m</code>	An example MATLAB script showing how to create a simple NetCDF file PISM can bootstrap from.
<code>util/PISMNC.py</code>	Used by many Python example scripts to generate a PISM-compatible file with the right dimensions and time-axis.

3.5.14 Using PISM for flow-line modeling

As described in sections *Computational box* and *Spatial grid*, PISM is a three-dimensional model. Moreover, parameters `Mx` and `My` have to be greater than or equal to three, so it is not possible to turn PISM into a 2D (flow-line) model

by setting `Mx` or `My` to 1.

There is a way around this, though: by using the `-periodicity` option to tell PISM to make the computational grid *y*-periodic and providing initial and boundary conditions that are functions of *x* only one can ensure that there is no flow in the *y*-direction. (Option `-periodicity` takes an argument specifying the direction: `none`, `x`, `y` and `xy` — for “periodic in both X- and Y-directions”.)

In this case `Mx` can be any number; we want to avoid unnecessary computations, though, so “`-Mx 3`” is the obvious choice.

One remaining problem is that PISM still expects input files to contain both *x* and *y* dimensions. To help with this, PISM comes with a Python script `flowline.py` that turns NetCDF files with *N* grid points along a flow line into files with 2D fields containing $N \times 3$ grid points.¹

Here’s an example which uses the script `util/flowlineslab.py` to create a minimal, and obviously unrealistic, dataset. A file `slab.nc` is created by `util/flowlineslab.py`, but it is not ready to use with PISM. Proceed as follows, after checking that `util/` is on your path:

```
flowlineslab.py                                # creates slab.nc with only an x-direction
flowline.py -o slab-in.nc --expand -d y slab.nc
```

produces a PISM-ready `slab-in.nc`. Specifically, `flowline.py` “expands” its input file in the *y*-direction. Now we can “bootstrap” from `slab-in.nc`:

```
mpiexec -n 2 pismr -surface given -i slab-in.nc -bootstrap -periodicity y \
-Mx 201 -My 3 -Lx 1000 -Ly 4 -Lz 2000 -Mz 11 -y 10000 -o pism-out.nc
```

To make it easier to visualize data in the file created by PISM, “collapse” it:

```
flowline.py -o slab-out.nc --collapse -d y pism-out.nc
```

3.5.15 Managing source code modifications

“Practical usage” may include editing the source code to extend, fix or replace parts of PISM.

We provide both user-level (this manual) and developer-level documentation. Please see source code browsers at <http://www.pism-docs.org> for the latter.

- To use your (modified) version of PISM, you will need to follow the compilation from sources instructions in the *Installation Manual*.
- It is a good idea to enable “debugging” settings when modifying PISM.

Benefits include

- better error messages during compilation,
- a number of sanity checks that are disabled by default,
- the ability to use debuggers such as `gdb` or `lldb`.

Set `CMAKE_BUILD_TYPE` to “Debug” in `ccmake` to enable debugging settings, then run `make` to re-compile.

Warning: Debugging settings disable code optimization, making PISM significantly slower. Please make sure that `CMAKE_BUILD_TYPE` is set to “Release” and `Pism_DEBUG` is set to “OFF” when compiling PISM for “real” runs.

¹ This script requires the `numpy` and `netCDF4` Python modules. Run `flowline.py --help` for a full list of options.

- We find it very useful to be able to check if a recent source code change broke something. PISM comes with “regression tests”, which check if certain parts of PISM perform the way they should.¹

Run “make test” in the build directory to run PISM’s regression tests.

Note, though, that while a test failure usually means that the new code needs more work, passing all the tests does not guarantee that everything works as it should. We are constantly adding new tests, but so far only a subset of PISM’s functionality can be tested automatically.

- We strongly recommend using a version control system to manage code changes. Not only is it safer than the alternative, it is also more efficient.

3.6 Simplified geometry experiments

There have been three stages of ice sheet model intercomparisons based on simplified geometry experiments since the early 1990s [97].

EISMINT I (European Ice Sheet Modeling INiTiative) [22]¹ was the first of these and involved only the isothermal shallow ice approximation (SIA). Both fixed margin and moving margin experiments were performed in EISMINT I, and various conclusions were drawn about the several numerical schemes used in the intercomparison. EISMINT I is superseded, however, by verification using the full variety of known exact solutions to the isothermal SIA [21]. The “rediscovery”, since EISMINT I, of the Halfar similarity solution with zero accumulation [98], and verification runs using that solution, already suffices to measure the isothermal SIA performance of PISM more precisely than would be allowed by comparison to EISMINT I results.

EISMINT II [14] pointed out interesting and surprising properties of the thermocoupled SIA. References [18], [99], [100], [36], [101], [17] each interpret the EISMINT II experiments and/or describe attempts to add more complete physical models to “fix” the (perceived and real) shortfalls of ice sheet model behavior on EISMINT II experiments. We believe that the discussion in [102], [36], [18] adequately explains the “spokes” in EISMINT II experiment F as a genuine fluid instability, while [37] and Appendix B of [17] adequately cautions against the continuum model that generates the “spokes” in EISMINT II experiment H. Thus we can move on from that era of controversy. In any case, PISM has built-in support for all of the published and unpublished EISMINT II experiments; these are described in the next subsection.

The ISMIP (Ice Sheet Model Intercomparison Project)² round of intercomparisons covers 2008–2013 (at least). There are four components of ISMIP substantially completed, namely HOM = Higher Order Models [103], [104], HEINO = Heinrich Event INtercOMparison [105], [79], MISMIP (below), and MISMIP3d (also below).

PISM participated in HEINO, but this ability is unmaintained. We believe the continuum problem described by HEINO, also used in EISMINT II experiment H (above), is not meaningfully approximate-able because of a required discontinuous jump in the basal velocity field. The continuum problem predicts infinite vertical velocity because of this jump ([17], Appendix B). Details of the numerical schemes and their results are irrelevant if the continuum model makes such a prediction. PISM offers the physical continuum model described in [17], an SIA+SSA hybrid, as an alternative to the continuum model used in ISMIP-HEINO and EISMINT II experiment H. Indeed the SIA+SSA hybrid is offered as a unified shallow model for real ice sheets (section *Ice dynamics, the PISM view*).

There is no current plan to support ISMIP-HOM [103], [104], but comparison of shallow PISM results to exact Stokes solutions is a goal for PISM evaluation.

A third and fourth ISMIP parts are the two parts of the Marine Ice Sheet Model Intercomparison Project, MISMIP [106] and MISMIP3D [107]. These experiments are supported in PISM, as described in subsections *MISMIP* and *MISMIP3d* below.

¹ This automates running verification tests described in section *Verification*, for example.

² See <http://homepages.vub.ac.be/~phuybrec/eismint.html>

² See <http://homepages.vub.ac.be/~phuybrec/ismip.html>

3.6.1 EISMINT II

There are seven experiments described in the published EISMINT II writeup [14]. They are named A, B, C, D, F, G, and H. They have these common features:

- runs are for 2×10^5 years, with no prescribed time step;
- a 61×61 horizontal grid on a square domain (1500 km side length) is prescribed;
- surface inputs (temperature and mass balance) have angular symmetry around the grid center;
- the bed is flat and does not move (no isostasy);
- the temperature in the bedrock is not modeled;
- only the cold (not polythermal) thermomechanically-coupled SIA is used [14]; and
- basal melt rates do not affect the evolution of the ice sheet.

The experiments differ from each other in their various combinations of surface temperature and mass balance parameterizations. Experiments H and G involve basal sliding, under the physically-dubious SIA sliding rubric ([17], Appendix B), while the others don't. Four experiments start with zero ice (A,F,G,H), while the other experiments (B,C,D) start from the final state of experiment A.

In addition to the seven experiments published in [14], there were an additional five experiments described in the EISMINT II intercomparison description [96], labeled E, I, J, K, and L. These experiments share most features listed above, but with the following differences. Experiment E is the same as experiment A except that the peak of the accumulation, and also the low point of the surface temperature, are shifted by 100 km in both x and y directions; also experiment E starts with the final state of experiment A. Experiments I and J are similar to experiment A but with non-flat “trough” bed topography. Experiments K and L are similar to experiment C but with non-flat “mound” bed topography.

See Table 3.34 for how to run all EISMINT II experiments in PISM. Experiments E – L are only documented in [96].

Table 3.34: Running the EISMINT II experiments in PISM. Use `-skip`
`-skip_max 5`, on the 61×61 default grid, for significant speedup.

Command: “pisms +”	Relation to experiment A
<code>-eisII A -Mx 61 -My 61 -Mz 61 -Lz 5000 -y 2e5 -o eisIIA.nc</code>	
<code>-eisII B -i eisIIA.nc -y 2e5 -o eisIIB.nc</code>	warmer
<code>-eisII C -i eisIIA.nc -y 2e5 -o eisIIC.nc</code>	less snow (lower accumulation)
<code>-eisII D -i eisIIA.nc -y 2e5 -o eisIID.nc</code>	smaller area of accumulation
<code>-eisII F -Mx 61 -My 61 -Mz 81 -Lz 6000 -y 2e5 -o eisIIF.nc</code>	colder; famous spokes [18]
<code>-eisII G -Mx 61 -My 61 -Mz 201 -Lz 5000 -y 2e5 -o eisIIG.nc</code>	sliding (regardless of temperature)
<code>-eisII H -Mx 61 -My 61 -Mz 201 -Lz 5000 -y 2e5 -o eisIIH.nc</code>	melt-temperature activated sliding
<code>-eisII E -i eisIIA.nc -y 2e5 -o eisIIE.nc</code>	shifted climate maps
<code>-eisII I -Mx 61 -My 61 -Mz 61 -Lz 5000 -y 2e5 -o eisIII.nc</code>	trough topography
<code>-eisII J -i eisIII.nc -y 2e5 -o eisIIJ.nc</code>	trough topography and less snow
<code>-eisII K -Mx 61 -My 61 -Mz 61 -Lz 5000 -y 2e5 -o eisIIK.nc</code>	mound topography
<code>-eisII L -i eisIIK.nc -y 2e5 -o eisIIL.nc</code>	mound topography and less snow

The vertical grid is not specified in EISMINT II, but a good simulation of the thermomechanically-coupled conditions near the base of the ice requires relatively-fine resolution there. We suggest using the default unequally-spaced grid. With 61 levels it gives a grid spacing of $\sim 20m$ in the ice layer closest to the bed, but more vertical levels are generally

better. Alternatively these experiments can be done with an equally-spaced grid; in this case we suggest using enough vertical levels to give 20 m spacing, for example. When there is sliding, even more vertical resolution is recommended (see Table 3.34). Also, the vertical extent must be sufficient so that when the ice thickness grows large, especially before thermo-softening brings it back down, the vertical grid is tall enough to include all the ice. Table 3.34 therefore includes suggested settings of `-Lz`; experiment F is different because ice thickness increases with colder temperatures.

These SIA-only simulations parallelize well. Very roughly, for the standard 61×61 horizontal grid, wall-clock-time speedups will occur up to about 30 processors. Runs on finer (horizontal) grids will benefit from even more processors. Also, the “skip” mechanism which avoids updating the temperature at each time step is effective, so options like `-skip_max 5` are recommended.

The EISMINT II experiments can be run with various modifications of the default settings. For instance, a twice-finer grid in the horizontal is “`-Mx 121 -My 121`”. Table 3.35 lists some optional settings which are particular to the EISMINT II experiments.

Table 3.35: Changing the default settings for EISMINT II

Option	Default values [experiments]	Units	Meaning
<code>-eisII</code>	A		Choose single character name of EISMINT II [14] simplified geometry experiment. See Table 3.34.
<code>-Mmax</code>	0.5 [ABDEFGHIK], 0.25 [CJL]	<i>m/year</i>	max value of accumulation rate
<code>-Rel</code>	450 [ABEFGHIK], 425 [CDJL]	km	radial distance to equilibrium line
<code>-Sb</code>	10^{-2} [all]	<i>(m/year)/km</i>	radial gradient of accumulation rate
<code>-ST</code>	1.67×10^{-2} [all]	K/km	radial gradient of surface temperature
<code>-Tmin</code>	238.15 [ACDEGHIJKL], 243.15 [B], 223.15 [F]	K	max of surface temperature
<code>-bmr_in_cont</code>			Include the basal melt rate in the mass continuity computation; overrides EISMINT II default.

See subdirectory `examples/eismintII/` for a simple helper script `runexp.sh`.

3.6.2 MISMIP

This intercomparison addresses grounding line dynamics by considering an idealized one-dimensional stream-shelf system. In summary, a flowline ice stream and ice shelf system is modeled, the reversibility of grounding line movement under changes in the ice softness is tested, different sliding laws are tested, and the behavior of grounding lines on reverse-slope beds is tested. The intercomparison process is described at the website

<http://homepages.ulb.ac.be/~fpattyn/mismip/>

Find a full text description there, along with the published report on the results [106]; that paper includes results from PISM version 0.1. These documents are essential reading for understanding MISMIP results generally, and for appreciating the brief discussion in this subsection.

PISM’s version of MISMIP includes an attached ice shelf even though modeling the shelf is theoretically unnecessary in the flow line case. The analysis in [108] shows that the only effect of an ice shelf, in the flow line case, is to transfer the force imbalance at the calving front directly to the ice column at the grounding line. Such an analysis does not apply to ice shelves with two horizontal dimensions; real ice shelves have “buttressing” and “side drag” and other forces not present in the flow line [109]. See the next subsection on MISMIP3d and the Ross ice shelf example in section *An SSA flow model for the Ross Ice Shelf in Antarctica*, among other examples.

We must adapt the usual 3D PISM model to two horizontal dimensions, i.e. to do flow-line problems (see section [Using PISM for flow-line modeling](#)). The flow direction for MISIMP is taken to be “x”. We periodize the cross-flow direction “y”, and use the minimum number of points in the y-direction. This number turns out to be “-My 3”; fewer points than this in the cross-flow direction confuses the finite difference scheme.

PISM can do MISIMP experiments with either of two applicable ice dynamics models. Model 1 is a pure SSA model; “category 2” in the MISIMP classification. Model 2 combines SIA and SSA velocities as described in [\[25\]](#); “category 3” because it resolves “vertical” shear (i.e. using SIA flow).

There are many runs for a complete MISIMP intercomparison submission. Specifically, for a given model there are 62 runs for each grid choice, and three (suggested) grid choices, so a full suite is $3 \times 62 = 186$ runs.

The coarsest grid (“mode 1”) has 12 km spacing. The finest grid, “mode 2” with 1.2 km spacing, accounts for all the compute time, however; in the MISIMP description it is 1500 grid spaces in the flow line direction (= 3001 grid *points* in PISM’s doubled computational domain). In between is “mode 3”, a mode interpretable by the intercomparison participant, and here we just use a 6 km grid.

The implementation of MISIMP in PISM conforms to the intercomparison description, but that document specifies

... we require that the rate of change of grounding line position be 0.1 m/a or less, while the rate of change of ice thickness at each grid point at which ice thickness is defined must be less than 10^{-4} m/a...

as a standard for “steady state”. The scripts here do not implement this stopping criterion. However, we report enough information, in PISM output files with scalar and spatially-variable time-series, to compute a grounding line rate or the time at which the thickness rate of change drops below 10^{-4} m/a.

See

```
examples/misimp/misimp2d/README.md
```

for usage of the scripts that run MISIMP experiments in PISM. For example, as described in this `README.md`, the commands

```
./run.py -e 1a --mode=1 > experiment-1a-mode-1.sh
bash experiment-1a-mode-1.sh 2 >& out.1a-mode-1 &
./plot.py ABC1_1a_M1_A7.nc -p -o profileA7.png
```

first generate a bash script, then use it to do a run which takes about 20 minutes, and then generate an image in `.png` format. Note that step 7 is in the middle of the experiment. It is shown in [Fig. 3.20](#) (left).

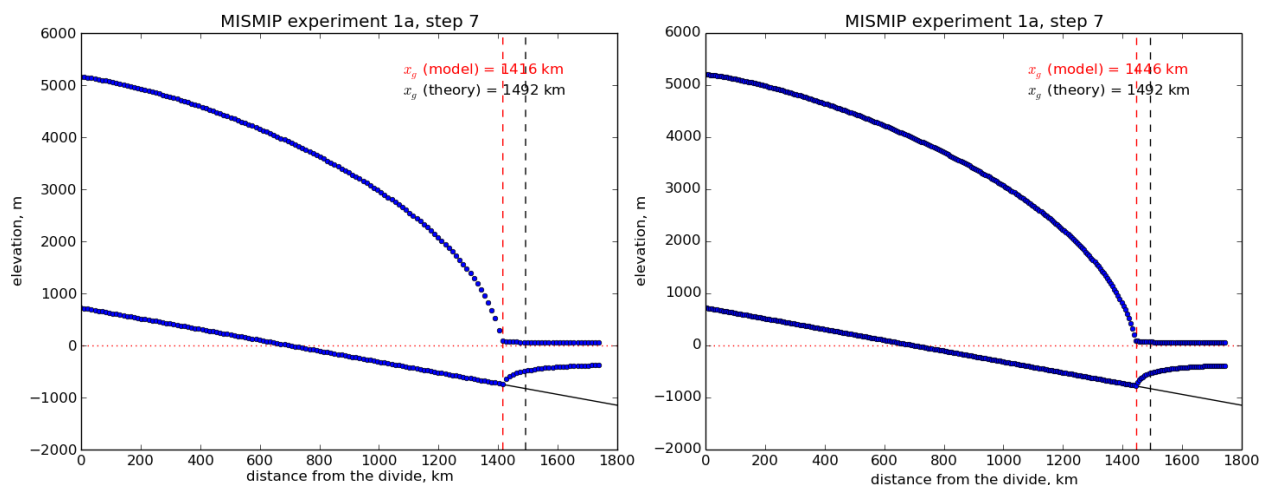


Fig. 3.20: A marine ice sheet profile in the MISIMP intercomparison; PISM model 1, experiment 1a, at step 7. Left: grid mode 1 (12 km grid). Right: grid mode 3 (6 km grid).

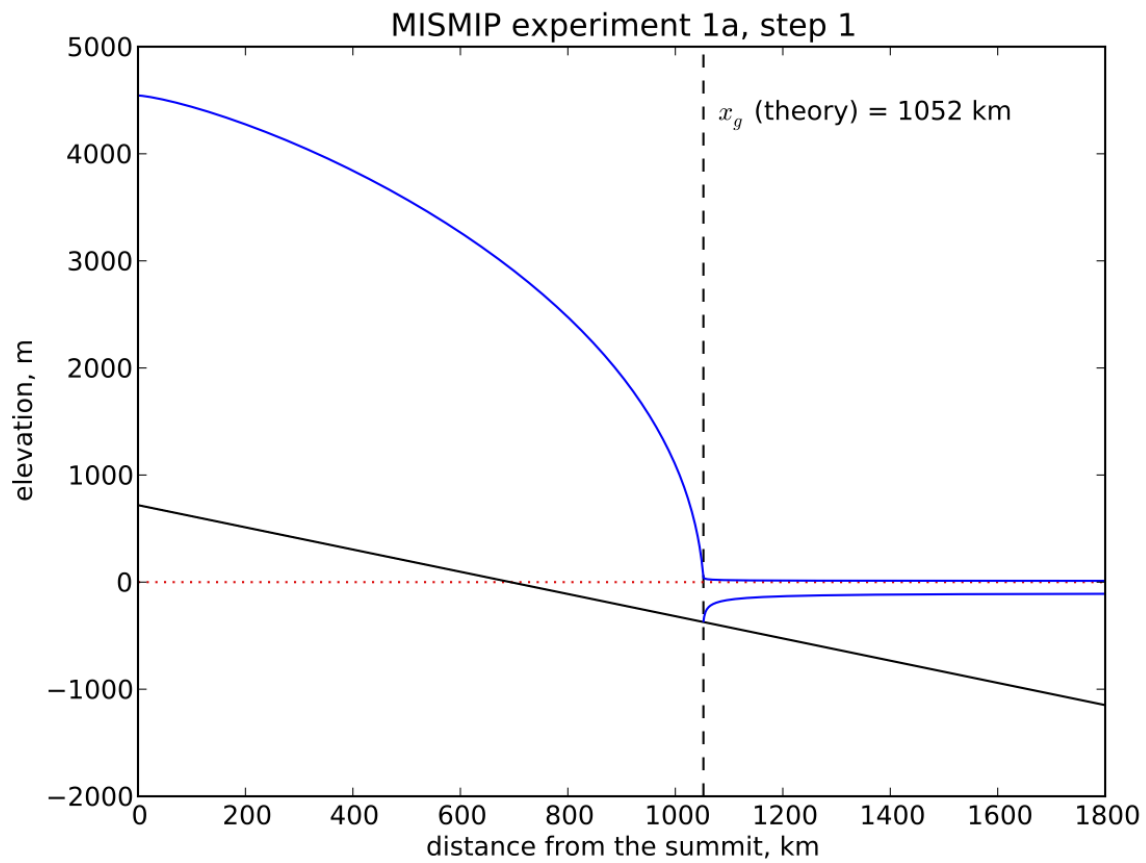


Fig. 3.21: Analytical profile for steady state of experiment 1a, step 1, from theory in [108]. This is a boundary layer asymptotic matching result, but not the exact solution to the equations.

The script `MISMIP.py` in `examples/mismip/mismip2d` has the ability to compute the profile from the Schoof's [108] asymptotic-matching boundary layer theory. This script is a Python translation, using `scipy` and `pylab`, of the provided MATLAB codes. For example,

```
python MISMIP.py -o mismip_analytic.png
```

produces a .png image file with Fig. 3.21. By default `run.py` uses the asymptotic-matching thickness result from the [108] theory to initialize the initial ice thickness, as allowed by the MISMIP specification.

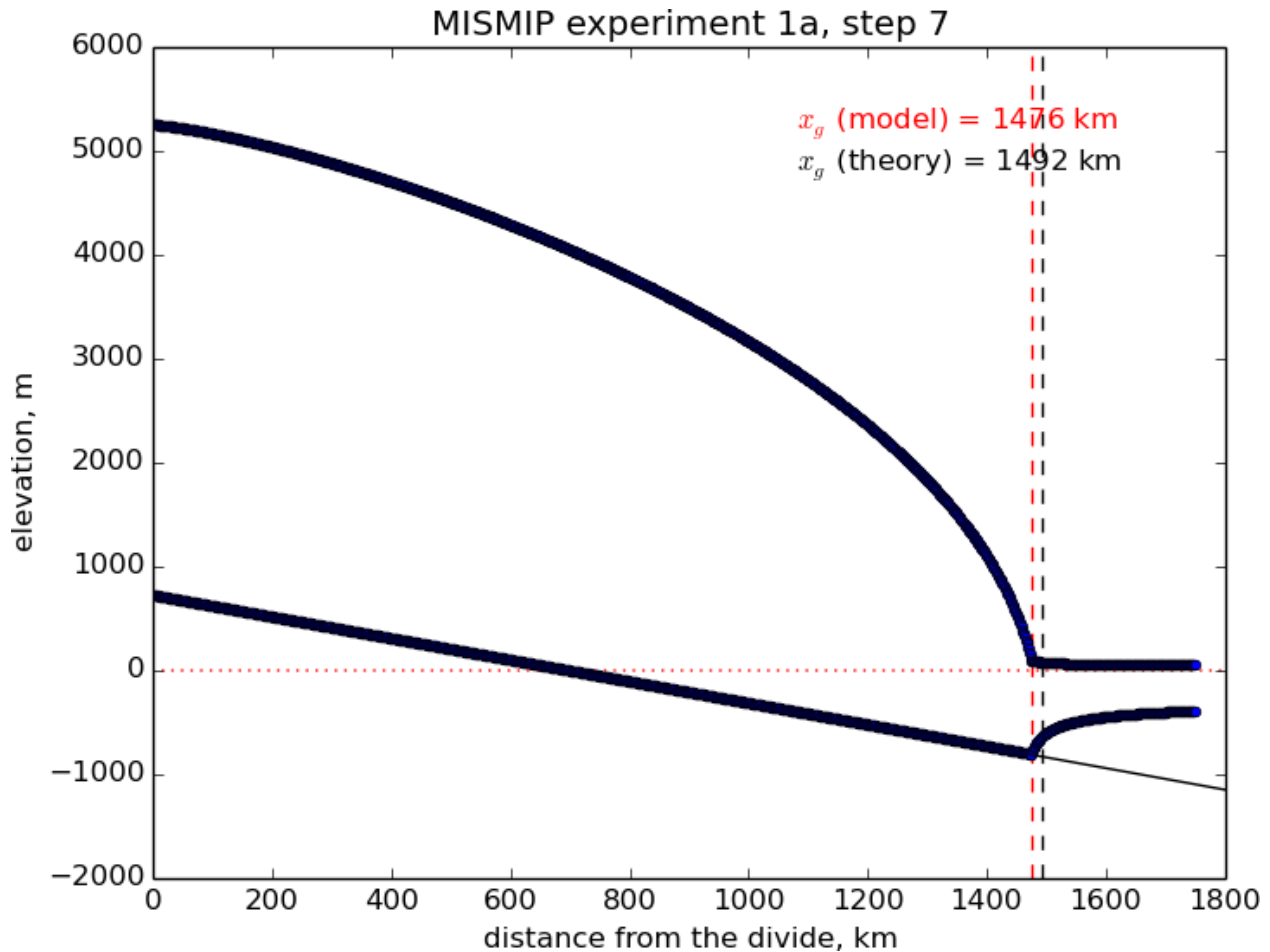


Fig. 3.22: Results from MISMIP grid mode 2, with 1.2 km spacing, for steady state of experiment 1a: profile at step 7 (compare Fig. 3.20).

Generally the PISM result does not put the grounding line in the same location as Schoof's boundary layer theory, and at least at coarser resolutions the problem is with PISM's numerical solution, not with Schoof's semi-analytic theory. The result improves under grid refinement, however. Results from grid mode 3 with 6 km spacing, instead of 12 km in mode 1, are the right part of Fig. 3.20. The corresponding results from grid mode 2, with 1.2 km spacing, are in Figure Fig. 3.22. Note that the difference between the numerical grounding line location and the semi-analytical location has been reduced from 76 km for grid mode 1 to 16 km for grid mode 2 (a factor of about 5), by using a grid refinement from 12 km to 1.2 km (a factor of about 10).

3.6.3 MISMIP3d

The ice2sea MISMIP3d intercomparison is a two-horizontal-dimensional extension of the flowline case described above. As before, in MISMIP3d the grounding line position and its reversibility under changes of physical parameters is analyzed. Instead of changing the ice softness, however, the spatial distribution and magnitude of basal friction is adjusted between experiments. The applied basal friction perturbation of the basal friction is a localized gaussian “bump” and thus a curved grounding line is obtained. In contrast to the flowline experiments, no (semi-)analytical solutions are available to compare to the numerical results.

A full description of the MISMIP3d experiments can be found at

<http://homepages.ulb.ac.be/~fpattyn/mismip3d/>

and the results are published in [107].

A complete set of MISMIP3d experiments consists of three runs: Firstly, a flowline solution on a linearly-sloped bed, similar to the flowline MISMIP experiments of the previous section, is run into a steady state (“standard experiment Stnd”). Then the localized sliding perturbation is applied (“perturbation experiment”) causing the grounding line to shift and lose symmetry. Two different amplitudes of the perturbation are considered (“P10” and “P75”). Finally, beginning from the final state of the perturbation experiment, the sliding perturbation is removed and the system is run again into steady state (“reversibility experiment”). The resulting geometry, in particular the grounding line position, is expected to be close to that of the standard experiment. Expecting such reversibility assumes that a particular stationary ice geometry only depends on its physical parameters and boundary conditions and not on how it is dynamically reached.

For these experiments in PISM, a Python script generates a shell script which has the commands and options for running a MISMIP3d experiment. The python script is `createscript.py` in the folder `examples/mismip/mismip3d/`. Run

```
./createscript.py -h
```

to see a usage message. A `README.md` gives a tutorial on how to use `createscript.py` and do the runs themselves.

For the flowline Stnd experiment, as in the MISMIP case, a computational domain with three grid points in the direction orthogonal to the ice flow (arbitrarily chosen as y-direction) is chosen by `createscript.py`. For the perturbation and reversibility experiments a domain is defined which is symmetric along the ice-divide (mirror symmetry) and along the center line of the ice flow, while the side boundaries are periodic, which corresponds to a free-slip condition for the flow in x-direction. Though this choice of the symmetric computational domain increases computational cost, it allows us to use standard PISM without fixing certain boundary conditions in the code. (That is, it avoids the issues addressed in the regional mode of PISM; see section *Example: A regional model of the Jakobshavn outlet glacier in Greenland.*)

PISM participated in the MISMIP3d intercomparison project [107] using version `pism0.5`, and the exact results can be reproduced using that version. PISM’s results, and the role of resolution and the new subgrid grounding line interpolation scheme are discussed in [76].

We observed a considerable improvement of the results with respect to the absolute grounding line positions compared to other models (e.g. the FE reference model Elmer/Ice) and to the reversibility when applying the subgrid grounding line interpolation method; see Fig. 3.23. Furthermore, we observed that only using SSA yields almost the same results as the full hybrid SIA+SSA computation for the MISMIP3D (and also the MISMIP) experiments, but, when not applying the SIA computation, after a considerably shorter computation time (about 10 times shorter). We explain the small and almost negligible SIA velocities for the MISMIP(3D) experiments with the comparably small ice surface gradients in the MISMIP3d ice geometries. See Fig. 3.24 for a comparison of SSA and SIA velocities in the MISMIP3D geometry. Note that both Figures Fig. 3.23 and Fig. 3.24 were generated with resolution of $\Delta x = \Delta y = 1$ km.

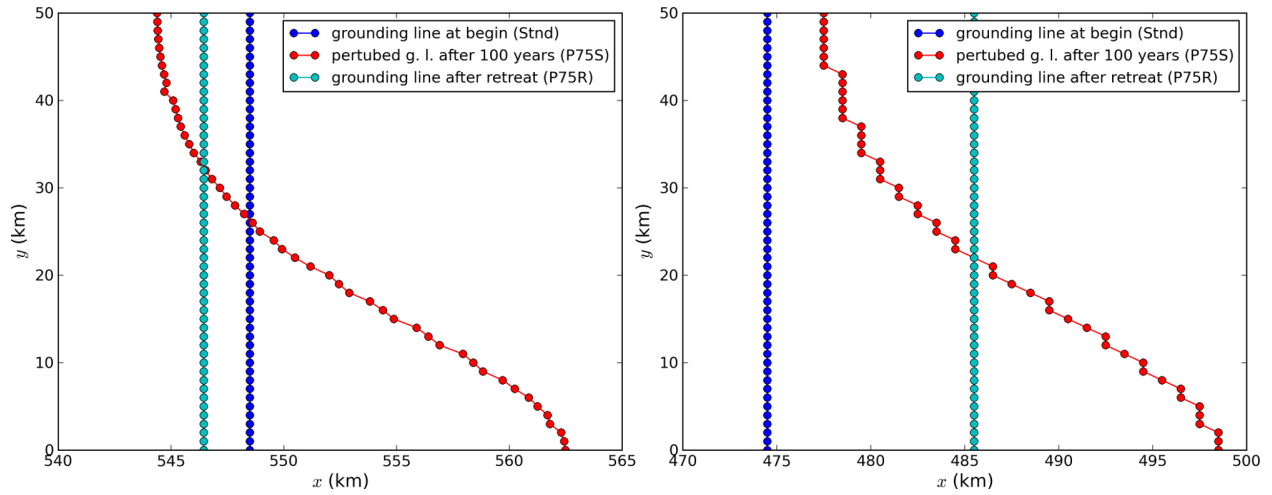


Fig. 3.23: Comparison between the grounding lines of the higher-amplitude (“P75”) MISIP3d experiments performed with PISM when using the subgrid grounding line interpolation method (left) or not using it (right). In both cases the SIA+SSA hybrid is used.

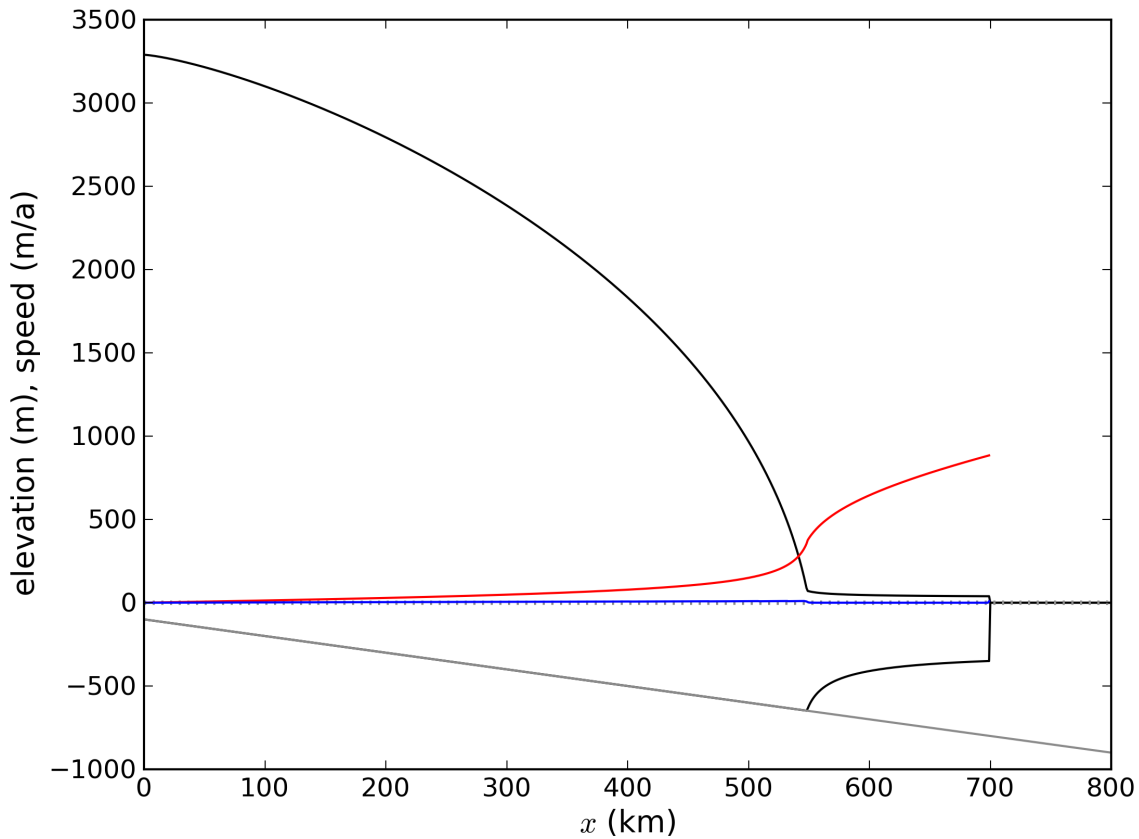


Fig. 3.24: The SIA velocities are negligible in the MISIP3d standard experiment (“Std”). The steady state ice geometry is plotted (black) together with the computed SSA velocity (red) and SIA velocity (blue). The SIA velocity reaches its maximum value of about 10 m/a at the grounding line, about two orders of magnitude less than the maximum of the SSA velocity.

3.7 Verification

Two types of errors may be distinguished: modeling errors and numerical errors. Modeling errors arise from not solving the right equations. Numerical errors result from not solving the equations right. The assessment of modeling errors is *validation*, whereas the assessment of numerical errors is called *verification*. . . Validation makes sense only after verification, otherwise agreement between measured and computed results may well be fortuitous.

P. Wesseling, (2001) *Principles of Computational Fluid Dynamics*, pp. 560–561 [122]

Verification is the essentially mathematical task of checking that the predictions of the numerical code are close to the predictions of a continuum model, the one which the numerical code claims to approximate. It is a crucial task for a code as complicated as PISM. In verification there is no comparison between model output and observations of nature. Instead, one compares exact solutions of the continuum model, in circumstances in which they are available, to their numerical approximations.

Reference [121] gives a broad discussion of verification and validation in computational fluid dynamics. See [21] and [18] for discussion of verification issues for the isothermal and thermomechanically coupled shallow ice approximation (SIA), respectively, and for exact solutions to these models, and [17], [33] for verification using an exact solution to the SSA equations for ice streams.

In PISM there is a separate executable `pismv` which is used for SIA-related verification, and there are additional scripts for SSA-related verification. The source codes which are verified by `pismv` are, however, exactly the same source files as are run by the normal PISM executable `pismr`. In technical terms, `pismv` runs a derived class of the PISM base class.

Table 3.36: Exact solutions for verification

Test	Continuum model tested	Reference	Comments
A	isothermal SIA, steady, flat bed, constant accumulation	[21]	
B	isothermal SIA, flat bed, zero accumulation	[21], [98]	similarity solution
C	isothermal SIA, flat bed, growing accumulation	[21]	similarity solution
D	isothermal SIA, flat bed, oscillating accumulation	[21]	uses compensatory accumulation
E	isothermal SIA; as A, but with sliding in a sector	[21]	uses compensatory accumulation
F	thermomechanically coupled SIA (mass and energy conservation), steady, flat bed	[65], [18]	uses compensatory accumulation and heating
G	thermomechanically coupled SIA; as F but with oscillating accumulation	[65], [18]	ditto
H	bed deformation coupled with isothermal SIA	[84]	joined similarity solution
I	stream velocity computation using SSA (plastic till)	[33], [17]	
J	shelf velocity computation using SSA	(source code)	
K	pure conduction in ice and bedrock	[131]	
L	isothermal SIA, steady, non-flat bed	(source code)	numerical ODE solution

Table 3.37: Canonical PISM verification runs using the exact solutions listed in Table 3.36.

Test	Example invocation
A	<code>pismv -test A -Mx 61 -My 61 -Mz 11 -y 25000</code>
B	<code>pismv -test B -Mx 61 -My 61 -Mz 11 -ys 422.45 -y 25000</code>
C	<code>pismv -test C -Mx 61 -My 61 -Mz 11 -y 15208.0</code>
D	<code>pismv -test D -Mx 61 -My 61 -Mz 11 -y 25000</code>
E	<code>pismv -test E -Mx 61 -My 61 -Mz 11 -y 25000</code>
F	<code>pismv -test F -Mx 61 -My 61 -Mz 61 -y 25000</code>
G	<code>pismv -test G -Mx 61 -My 61 -Mz 61 -y 25000</code>
H	<code>pismv -test H -Mx 61 -My 61 -Mz 11 -y 40034 -bed_def iso</code>
I	<code>ssa_testi -ssa_method fd -Mx 5 -My 500 -ssa_rtol 1e-6 -ssaafd_ksp_rtol 1e-11</code>
J	<code>ssa_testj -ssa_method fd -Mx 60 -My 60 -ssaafd_ksp_rtol 1e-12</code>
K	<code>pismv -test K -Mx 6 -My 6 -Mz 401 -Mbz 101 -y 130000</code>
L	<code>pismv -test L -Mx 61 -My 61 -Mz 31 -y 25000</code>

Table 3.38: `pismv` command-line options

Option	Description
<code>-test</code>	Choose verification test by single character name; see Table 3.36.
<code>-no_report</code>	Do not report errors at the end of a verification run.
<code>-eo</code>	Only evaluate the exact solution; no numerical approximation at all.
<code>-report_file</code>	Save error report to a netCDF file.
<code>-append</code>	Append to a report file.

Table 3.36 summarizes the many exact solutions currently available in PISM. Most of these exact solutions are solutions of *free boundary problems* for partial differential equations; only Tests A, E, J, K are fixed boundary value problems.

Table 3.37 shows how to run each of them on a coarse grids. Note that tests I and J require special executables `ssa_testi`, `ssa_testj` which are built with configuration flag `Pism_BUILD_EXTRA_EXECS` equal to ON. Table 3.38 gives the special verification-related options of the `pismv` executable.

Numerical errors are not, however, the dominant reasons why ice sheet models give imperfect results. The largest sources of errors include those from using the wrong (e.g. over-simplified or incorrectly-parameterized) continuum model, and from observational or pre-processing errors present in input data. Our focus here on numerical errors has a model-maintenance goal. It is *easier* to maintain code by quantitatively confirming that it produces small errors in cases where those can be measured, rather than “eyeballing” results to see that they are “right” according to human judgment.

The goal of verification is not generally to see that the error is zero at any particular resolution, or even to show that the error is small in a predetermined absolute sense. Rather the goals are

- to see that the error *is* decreasing,
- to measure the rate at which it decreases, and
- to develop a sense of the magnitude of numerical error before doing realistic ice sheet model runs.

Knowing the error decay rate may give a prediction of how fine a grid is necessary to achieve a desired smallness for the numerical error.

Therefore one must “go down” a grid refinement “path” and measure numerical error for each grid [121]. The refinement path is defined by a sequence of spatial grid cell sizes which decrease toward the refinement limit of zero size [132]. In PISM the timestep Δt is determined adaptively by a stability criterion (see section *Understanding adaptive time-stepping*). In PISM one specifies the number of grid points, thus the grid cell sizes because the overall dimensions

of the computational box are normally fixed; see section *Computational box*. By “measuring the error for each grid” we mean computing a norm (or norms) of the difference between the numerical solution and the exact solution.

For a grid refinement path example, in tests of the thermomechanically-coupled SIA model one refines in three dimensions, and these runs produced Figures 13, 14, and 15 of [18]:

```
pismv -test G -max_dt 10.0 -y 25000 -Mx 61 -My 61 -Mz 61 -z_spacing equal
pismv -test G -max_dt 10.0 -y 25000 -Mx 91 -My 91 -Mz 91 -z_spacing equal
pismv -test G -max_dt 10.0 -y 25000 -Mx 121 -My 121 -Mz 121 -z_spacing equal
pismv -test G -max_dt 10.0 -y 25000 -Mx 181 -My 181 -Mz 181 -z_spacing equal
pismv -test G -max_dt 10.0 -y 25000 -Mx 241 -My 241 -Mz 241 -z_spacing equal
pismv -test G -max_dt 10.0 -y 25000 -Mx 361 -My 361 -Mz 361 -z_spacing equal
```

The last two runs require a supercomputer! In fact the $361 \times 361 \times 361$ run involves more than 100 million unknowns, updated at each of millions of time steps. Appropriate use of parallelism (`mpirun -n NN pismv`) and of the `-skip` modification to adaptive timestepping accelerates such fine-grid runs; see section *Understanding adaptive time-stepping*.

Figures Fig. 3.25 through Fig. 3.29 in *Sample convergence plots* show a sampling of the results of verifying PISM using the tests described above. These figures were produced automatically using Python scripts `test/vfnw.py` and `test/vnreport.py`. See section *Utility and test scripts*.

These figures *do not* show outstanding rates of convergence, relative to textbook partial differential equation examples. For the errors in tests B and G, see the discussion of free margin shape in [21]. For the errors in test I, the exact continuum solution is not very smooth at the free boundary [33].

3.7.1 Sample convergence plots

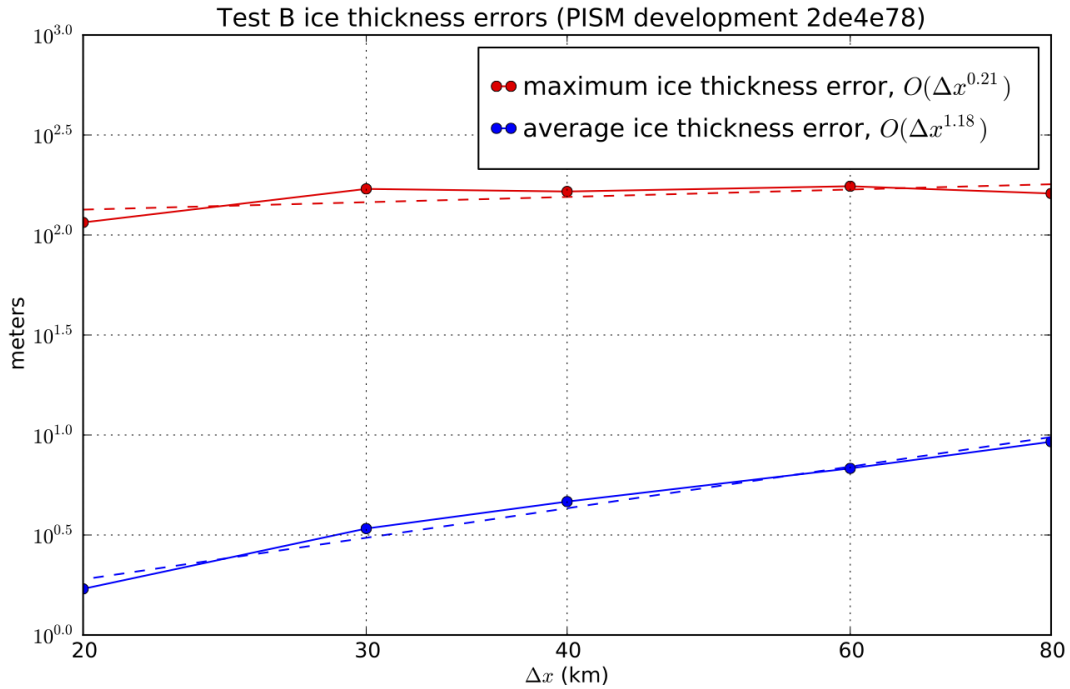


Fig. 3.25: Numerical thickness errors in test B. See [21] for discussion.

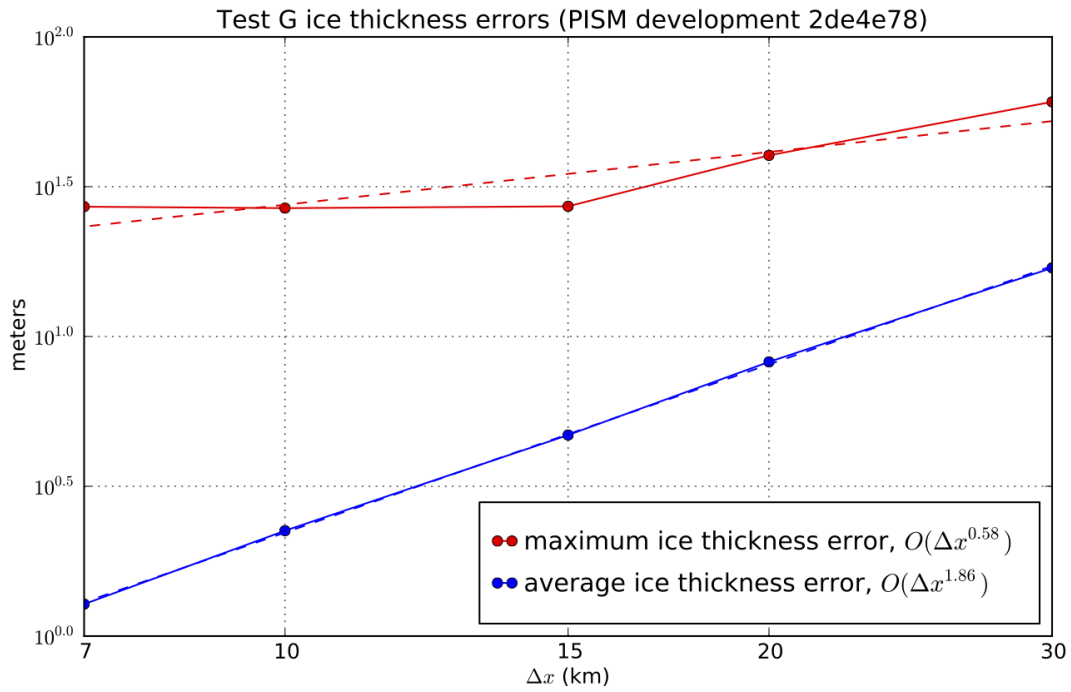


Fig. 3.26: Numerical thickness errors in test G. See [18] and [21].

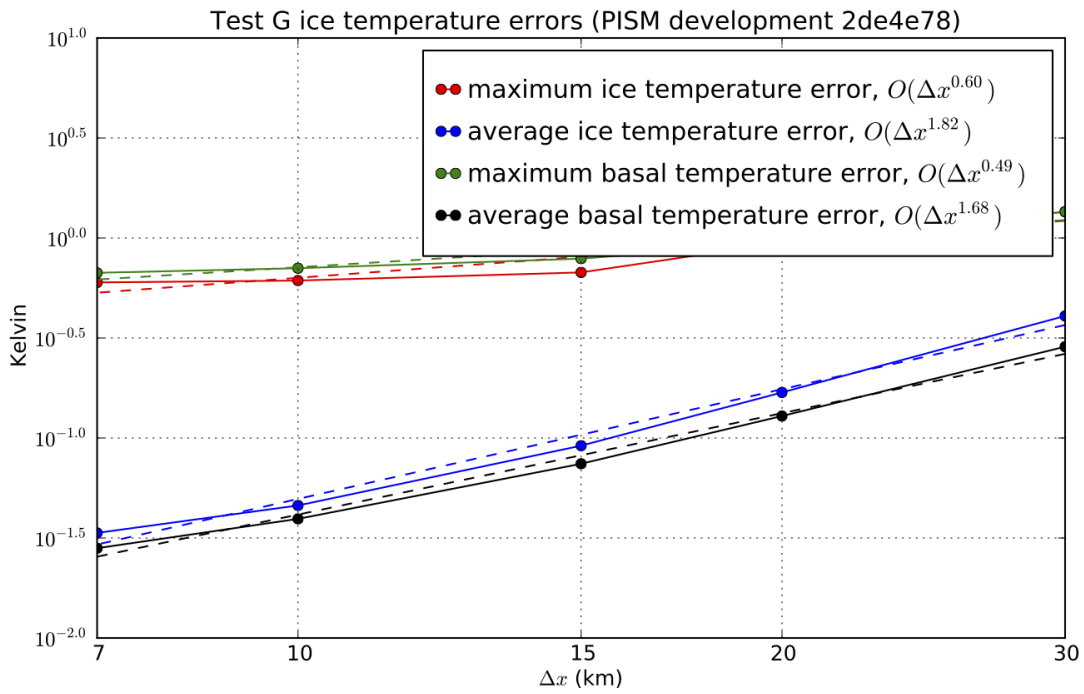


Fig. 3.27: Numerical temperature errors in test G. See [18].

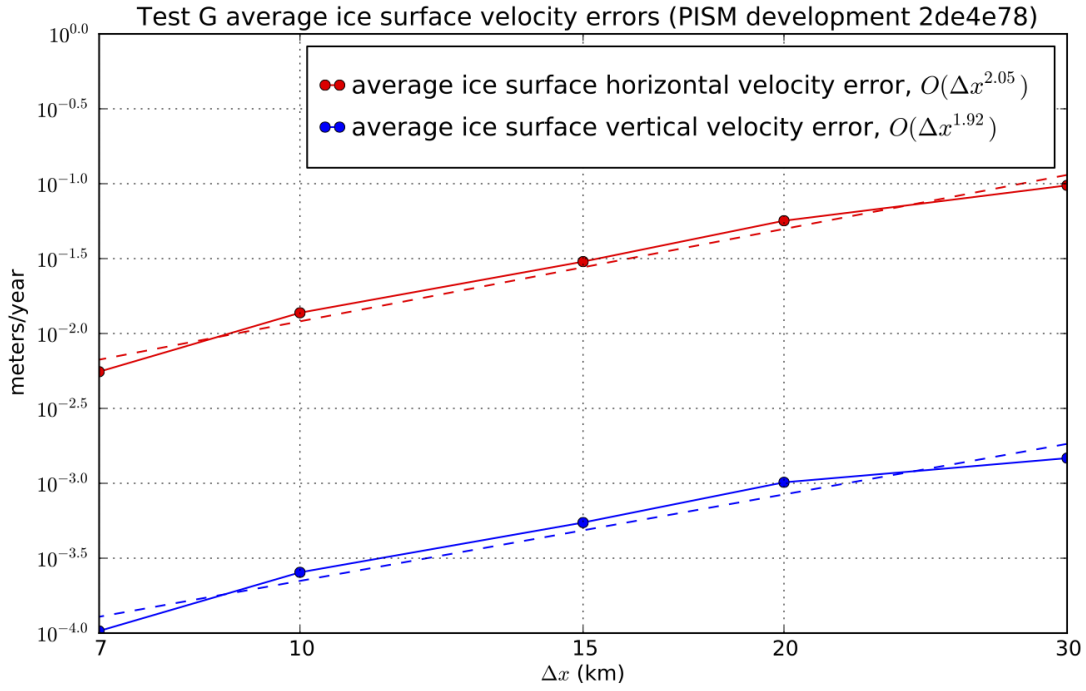


Fig. 3.28: Numerical errors in computed surface velocities in test G.

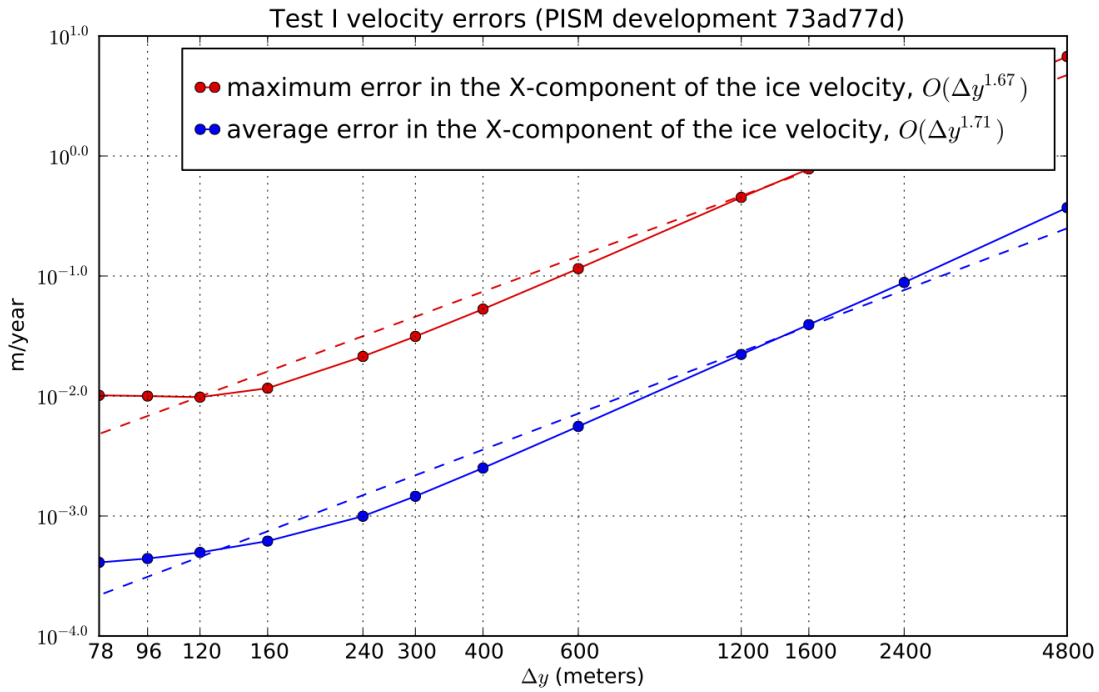


Fig. 3.29: Numerical errors in horizontal velocities in test I, an ice stream. See [33], [17].

3.8 Validation case studies

“Validation” describes the comparison of numerical model output with physical observations in cases where the observations are sufficiently-complete and of sufficient quality so that the performance of the numerical model can be assessed [121], [122]. Roughly speaking, validation can happen when the observations or data are better than the model, so the comparison measures the quality of the numerical model and not merely errors in, or incompleteness of, the data. Because of the difficulty of measuring boundary conditions for real ice flows, this situation is not automatic in glaciology, or even common.¹ Nonetheless we try two cases, first where PISM is applied on millimeter scale to model a laboratory experiment, and second for a large-scale ice flow in which all uncertainties of bedrock topography, basal sliding, and subglacial hydrology are removed, namely a present-day ice shelf.

3.8.1 An SIA flow model for a table-top laboratory experiment

Though there are additional complexities to the flow of real ice sheets, an ice sheet is a shear-thinning fluid with a free surface. PISM ought to be able to model such flows in some generality. We test that ability here by comparing PISM’s isothermal SIA numerical model to a laboratory observations of a 1% Xanthan gum suspension in water in a table-top, moving-margin experiment by R. Sayag and M. Worster [123], [124]. The “gum” fluid is more shear-thinning than ice, and it has much lower absolute viscosity values, but it has the same density. This flow has total mass ~ 1 kg, compared to $\sim 10^{18}$ kg for the Greenland ice sheet.

We compare our numerical results to the “constant-flux” experiment from [123]. Fig. 3.30 shows the experimental setup by reproducing Figures 2(c) and 2(d) from that reference. A pump pushes the translucent blue-dyed fluid through a round 8 mm hole in the middle of a clear table-top at a mass rate of about 3 gm/s. The downward-pointing camera, which produced the right-hand figure, allows measurement of the location of margin of the “ice cap”, and in particular of its radius. The measured radii data are the black dots in Fig. 3.31.

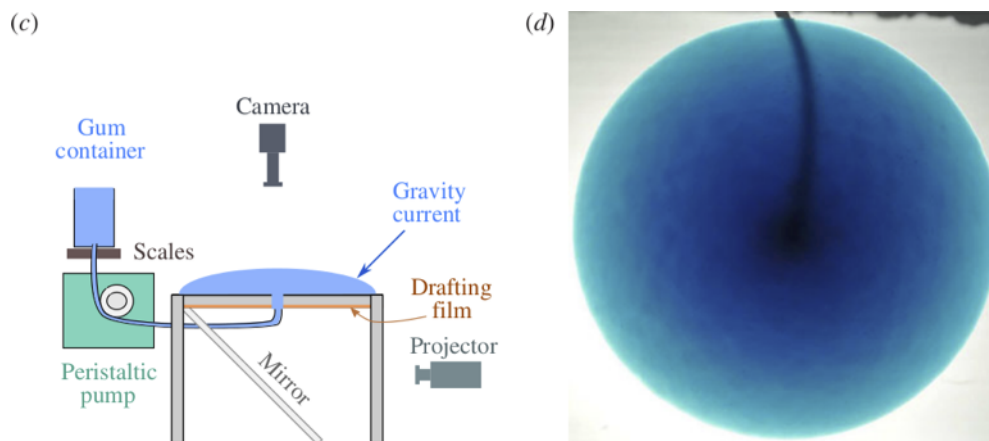


Fig. 3.30: Reproduction of Figures 2(c) and 2(d) from [123]. Left: experimental apparatus used for “constant-flux release” experiment. Right: snapshot of constant-flux experiment (plan view), showing an axisymmetric front.

The closest glaciological analog would be an ice sheet on a flat bed fed by positive basal mass balance (i.e. “refreeze”) underneath the dome, but with zero mass balance elsewhere on the lower and upper surfaces. However, noting that the mass-continuity equation is vertically-integrated, we may model the input flux (mass balance) as arriving at the top of the ice sheet, to use PISM’s climate-input mechanisms. The flow though the input hole is simply modeled as constant across the hole, so the input “climate” uses `-surface` given with a field `climatic_mass_balance`, in the bootstrapping file, which is a positive constant in the hole and zero outside. While our replacement of flow into the base by mass balance at the top represents a very large change in the vertical component of the velocity field, we still see good agreement in the overall shape of the “ice sheet”, and specifically in the rate of margin advance.

¹ Which explains the rise of “simplified geometry intercomparisons”; see section *Simplified geometry experiments*.

Sayag & Worster estimate Glen exponent $n = 5.9$ and a softness coefficient $A = 9.7 \times 10^{-9} \text{ Pa}^{-5.9} \text{ s}^{-1}$ for the flow law of their gum suspension, using regression of laboratory measurements of the radius. (Compare PISM defaults $n = 3$ and $A \approx 4 \times 10^{-25} \text{ Pa}^{-3} \text{ s}^{-1}$ for ice.) Setting the Sayag & Worster values is one of several changes to the configuration parameters, compared to PISM ice sheet defaults, which are done in part by overriding parameters at run time by using the `-config_override` option. See `examples/labgum/preprocess.py` for the generation of a configuration `.nc` file with these settings.

To run the example on the default 10 mm grid, first do

```
./preprocess.py
```

and then do a run for 746 model seconds [123] on the 10 mm grid on a $520 \text{ mm} \times 520 \text{ mm}$ square domain using 4 processors:

```
./rungum.sh 4 52 &> out.lab52 &
```

This run generates text file `out.lab52`, diagnostic files `ts_lab52.nc` and `ex_lab52.nc`, and final output `lab52.nc`. This run took about 5 minutes on a 2013 laptop, thus roughly real time! When it is done, you can compare the modeled radius to the experimental data:

```
./showradius.py -o r52.png -d constantflux3.txt ts_lab52.nc
```

You can also redo the whole thing on higher resolution grids (here: 5 and 2.5 mm), here using 6 MPI processes if the runs are done simultaneously, and when it is done after several hours, make a combined figure just like Fig. 3.31:

```
./preprocess.py -Mx 104 -o initlab104.nc
./preprocess.py -Mx 208 -o initlab208.nc
./rungum.sh 2 104 &> out.lab104 &
./rungum.sh 4 208 &> out.lab208 &
./showradius.py -o foo.png -d constantflux3.txt ts_lab*.nc
```

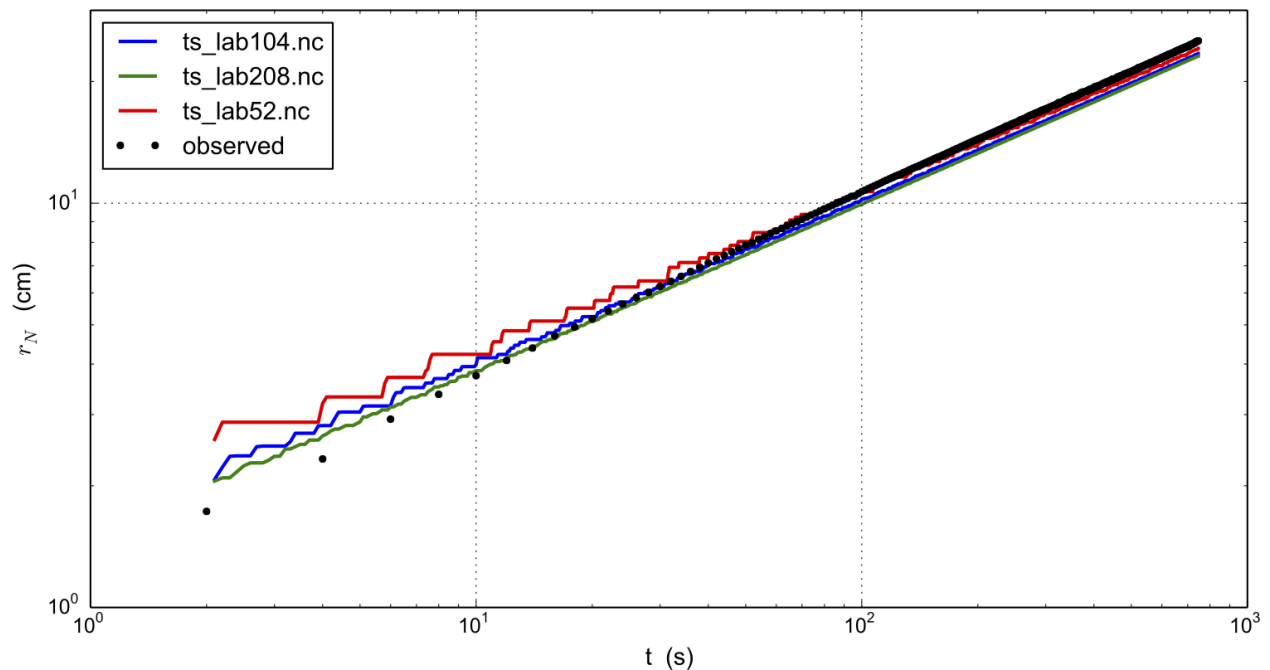


Fig. 3.31: Radius $r_N(t)$ for runs with 10 mm (`ts_lab52.nc`), 5 mm (`ts_lab104.nc`), and 2.5 mm (`ts_lab208.nc`) grids, compared to observations from Sayag & Worster’s [123] table-top “ice cap” (gravity current) made from a 1% Xanthan gum suspension, as shown in Figure Fig. 3.30.

We see that on the coarsest grid the modeled volume has “steps” because the margin advances discretely. Note we are computing the radius by first computing the fluid-covered area a on the cartesian grid, and then using $a = \pi r^2$ to compute the radius.

Results are better on finer grids, especially at small times, because the input hole has radius of only 8 mm. Furthermore this “ice cap” has radius comparable to the hole for the first few model seconds. The early evolution is thus distinctly non-shallow, but we see that increasing the model resolution reduces most of the observation-model difference. In fact there is little need for “higher-order” stresses because the exact similarity solution of the shallow continuum equations, used by Sayag & Worster, closely-fits the data even for small radius and time (see [123], Figure 4).

In any case, the large-time observations are very closely-fit by the numerical results at all grid resolutions. We have used the Glen-law parameters n, A as calculated by Sayag & Worster, but one could do parameter-fitting to get the “best” values if desired. In particular, roughly speaking, n controls the slope of the results in Fig. 3.31 and A controls their vertical displacement.

3.8.2 An SSA flow model for the Ross Ice Shelf in Antarctica

As part of the EISMINT series of intercomparisons, MacAyeal and others [57] successfully validated early-1990s ice shelf numerical models using velocity data for the Ross ice shelf. The data were from the RIGGS survey [125], acquired in the period 1973–1978 and measured at a few hundred locations in a grid across the shelf. Substantial modelling developments followed EISMINT-Ross, including inverse modeling to recover depth-averaged viscosity [126] and parameter-sensitivity studies [127]. Previous PISM versions set up the EISMINT-Ross flow model and performed the diagnostic computation, with RIGGS data for validation.

However, availability of rich new data sets for ice sheet modeling, including the ALBMAP v1 [128] ice sheet geometry, bedrock, and climate data set, and the radar-derived (InSAR) MEaSUREs Antarctica Velocity Map [129], allows us to use more complete, recent, and higher-resolution data for the same basic job. Furthermore one can extend the diagnostic Ross ice shelf calculation both to other ice shelves around Antarctica and to time-evolving (“prognostic”) cases using the eigencalving [70] mechanisms.

The scripts in this subsection are found in directory `examples/ross/`. In summary, the script `preprocess.py` downloads easily-available data and builds a NetCDF input file for PISM. For the diagnostic computation we document first, the script `run_diag.sh` (in subdirectory `examples/ross/diagnostic/`) runs PISM. The script `plot.py` shows a comparison of observations and model results, as in Fig. 3.32.

Preprocessing input data

NSIDC requires registration to download the MEaSUREs InSAR-Based Antarctica Ice Velocity Map; please register, log in, and get the *first*, 900 m version of it (`antarctica_ice_velocity_900m.nc`) here:

<https://n5eil01u.ecs.nsidc.org/MEASURES/NSIDC-0484.001/1996.01.01/>

Put this file in `examples/ross`.

The script `preprocess.py` then downloads ALBMAP using `wget`; these two files total around 350 Mb. Then it uses `NCO` to cut out the relevant portion of the grid and `CDO` to conservatively-interpolate the high-resolution (900 m) velocity data onto the coarser (5 km) geometry grid used in ALBMAP. The script `nc2cdo.py` from directory `util/`, prepares the NetCDF file for the application of CDO, which requires complete projection information. Run

```
cd examples/ross/  
./preprocess.py
```

to download ALBMAP and finish the pre-processing.

The NetCDF file `Ross_combined.nc` produced by `preprocess.py` contains ice thickness, bed elevations, surface temperature, net accumulation, as well as latitude and longitude values. All of these are typical of ice sheet modelling

data, both in diagnostic runs and as needed to initialize and provide boundary conditions for prognostic (evolutionary) runs; see below for the prognostic case with these data. The `_combined` file also has variables `u_ssa_bc` and `v_ssa_bc` for the boundary values used in the (diagnostic and prognostic) computation of velocity. They are used at all grounded locations and at ice shelf cells that are immediate neighbors of grounded ice. The variable `bc_mask` specifies these locations. Finally the variables `u_ssa_bc`, `v_ssa_bc`, which contain observed values, are used after the run to compare to the computed interior velocities.

Diagnostic computation of ice shelf velocity

The diagnostic velocity computation bootstraps from `Ross_combined.nc` and performs a short run; in the 211×211 grid case we demonstrate below, the key parts of the PISM command are

```
pismr -i ../Ross_combined.nc -bootstrap -Mx 211 -My 211 -Mz 3 -Lz 3000 -z_spacing equal \
-surface given -stress_balance ssa -energy none -no_mass -yield_stress constant -tauc 1e6 \
-pik -ssa_dirichlet_bc -y 1.0 -ssa_e 0.6 -ssafd_ksp_monitor
```

The computational grid here is the “native” 5 km data grid used in ALBMAP. Regarding the options,

- The maximum thickness of the ice is 2766 m so we choose a height for the computational box large enough to contain the ice (i.e. `-Lz 3000`). Vertical grid resolution is, however, unimportant in this case because we use the SSA stress balance only, and the temperature set at bootstrapping suffices to determine the ice softness; thus the options `-Mz 3 -z_spacing equal`.
- Option `-stress_balance ssa` selects the SSA stress balance and turns off the SIA stress balance computation, since our goal is to model the ice shelf. It also side-steps a technical issue: PISM uses periodic boundary conditions at domain boundaries and most fields in this setup are not periodic. Turning off SIA avoids operations such as differencing surface elevation across the domain edges. For a more complete solution to this technical issue see section *Example: A regional model of the Jakobshavn outlet glacier in Greenland* about a regional model using PISM’s “regional mode” `pismr -regional` and the option `-no_model_strip`.
- Option `-y 1.0 -no_mass -energy none` chooses a “diagnostic” run: in absence of geometry evolution and stability restrictions of the energy balance model a one-year-long run will be covered by exactly one time step.
- Option `-pik` is equivalent to `-cfbc -part_grid -kill_icebergs -subgl` in this non-evolving example. Note that `-kill_icebergs` removes effectively-detached bits of ice, especially in McMurdo sound area, so that the SSA problem is well-posed for the grounded-ice-sheet-connected ice shelf.
- Option `-ssa_dirichlet_bc` forces the use of fields `u_ssa_bc`, `v_ssa_bc`, `bc_mask` described above. The field `bc_mask` is 1 at boundary condition locations, and 0 elsewhere. For the prognostic runs below, the ice thickness is also fixed at boundary condition locations, so as to prescribe ice flux as an ice shelf input.
- Options `-yield_stress constant -tauc 1e6` essentially just turn off the grounded-ice evolving yield stress mechanism, which is inactive anyway, and force a high resistance under grounded ice so it does not slide.
- Option `-ssa_e 0.6` is the single tuned parameter; this value gives good correlation between observed and modeled velocity magnitudes.
- Option `-ssafd_ksp_monitor` provides feedback on the linear solver iterations “underneath” the nonlinear (shear-thinning) SSA solver iteration.

There is no need to type in the above command; just run

```
cd diagnostic/
./run_diag.sh 2 211 0.6
```

Note `run_diag.sh` accepts three arguments: `run_diag.sh N Mx E` does a run with `N` MPI processes, an `Mx` by `Mx` grid, and option `-ssa_e E`. The choices above give a run which only takes a few seconds, and it produces output file `diag_Mx211.nc`.

There are many reasonable choices for the effective softness of an ice shelf, as ice density, temperature, and the presence of fractures all influence the effective softness. Using an enhancement factor `-ssa_e 0.6` acknowledges that the physical justification for tuning the ice softness is uncertain. One could instead use the temperature itself or the ice density¹ as tuning parameters, and these are worthwhile experiments for the interested PISM user.

The script `plot.py` takes PISM output such as `diag_Mx211.nc` to produce Fig. 3.32. The run shown in the figure used an enhancement factor of 0.6 as above. The thin black line outlines the floating shelf, which is the actual modeling domain here. To generate this figure yourself, run

```
../plot.py diag_Mx211.nc
```

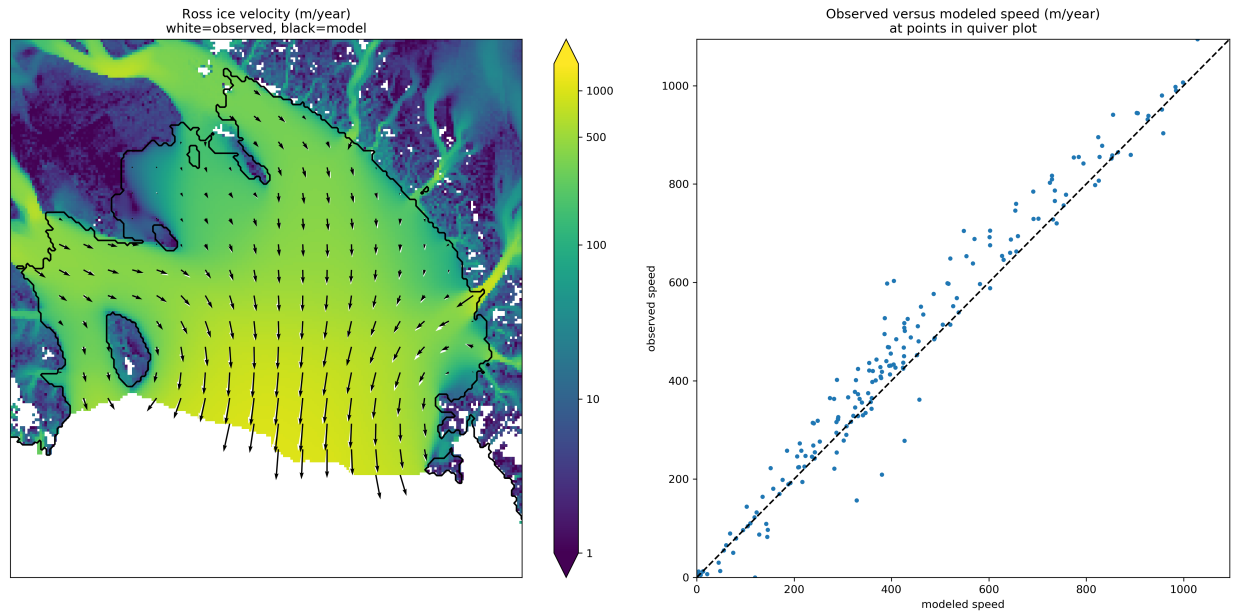


Fig. 3.32: *Left*: Color is speed in m/a. Arrows are observed (white) and modeled (black) velocities. *Right*: Comparison between modeled and observed speeds at points plotted on the left.

Extending this example to other ice shelves

The SSA diagnostic solution described in this section can be easily applied to other ice shelves in Antarctica, such as the Filchner-Ronne Ice Shelf modeled using PISM in [130], for example.

Simply choose a different rectangular domain, within the area covered by the whole-Antarctic data-sets used here, at the preprocessing stage. In particular you should modify the lines “`ncks -O -d x1,439,649 -d y1,250,460 ...`” (for ALBMAP data) and “`ncks -d x,2200,3700 -d y,3500,4700 ...`” (for MEaSUREs velocity data) in the script `examples/ross/preprocess.py`.

Prognostic modelling using eigencalving

Next we summarize how you can create an evolving-geometry model of the Ross ice shelf with constant-in-time inflow across the fixed grounding line. See `README.md` and `run_prog.sh` in `examples/ross/prognostic/`. This example also demonstrates the `-calving eigen_calving` model for a moving calving front [70].

¹ High accumulation rates, cold firn with minimal compression, and basal freeze-on of marine ice may all generate significant variation in shelf density.

Start by running `preprocess.py` in `examples/ross/` as described above. If you have already done the diagnostic example above, then this stage is complete.

Then change to the `prognostic/` directory and run the default example:

```
cd examples/ross/prognostic/
./run_prog.sh 4 211 0.6 100
```

This 100 model year run on 4 processes and a 5 km grid took about forty minutes on a 2016 laptop. It starts with a bootstrapping stage which does a `-y 1.0` run, which generates `startfile_Mx211.nc`. It then re-initializes to start the prognostic run itself. See the `README.md` for a bit more on the arguments taken by `run_prog.sh` and on viewing the output files.

The PISM command done here is (essentially, and without showing diagnostic output choices)

```
pismr -i startfile_Mx211.nc -surface given -stress_balance ssa \
-yield_stress constant -tauc 1e6 -pik -ssa_dirichlet_bc -ssa_e 0.6 \
-y 100 -o prog_Mx211_yr100.nc -o_size big \
-calving eigen_calving,thickness_calving -eigen_calving_K 1e17 \
-calving_cfl -thickness_calving_threshold 50.0
```

Several of these options are different from those used in the diagnostic case. First, while the command `-pik` is the same as before, now each part of its expansion, namely `-cfbc` `-part_grid` `-kill_icebergs` `-subgl`, is important. As the calving front evolves (i.e. regardless of the calving law choices), option `-part_grid` moves the calving front by one grid cell only when the cell is full of the ice flowing into it; see [73]. The option `-kill_icebergs` is essential to maintain well-posedness of the SSA velocity problem at each time step [25]. See section *PIK options for marine ice sheets*.

Option combination

```
-calving eigen_calving,thickness_calving -eigen_calving_K 1e17 \
-calving_cfl -thickness_calving_threshold 50.0
```

specifies that ice at the calving front will be removed if either a criterion on the product of principal stresses is satisfied [70], namely `eigen_calving` with the given constant K , or if the ice thickness goes below the given threshold of 50 meters. See section *Calving*.

3.9 Example: A regional model of the Jakobshavn outlet glacier in Greenland

Jakobshavn Isbrae is a fast-flowing outlet glacier in western Greenland that drains approximately 7% of the area of the Greenland ice sheet. It experienced a large acceleration following the loss of its floating tongue in the 1990s [47], an event which seems to have been driven by warmer ocean temperatures [48]. Because it is thick, has a steep surface slope, has a deep trough in its bedrock topography (Figure Fig. 3.33), and has a thick layer of low-viscosity temperate ice at its base [49], this ice flow is different from the ice streams in West Antarctica or Northeast Greenland [39].

This section describes how to build a PISM regional model of this outlet glacier [50] using scripts from `examples/jako/`. The same strategy should work for other outlet glaciers. We also demonstrate the PISM regional mode `pismr -regional`, and Python `drainage-basin-delineation` tools which can be downloaded from the PISM source code website. Such regional models allow modest-size computers to run high resolution models¹ and large ensembles. Regional analysis is justified if detailed data is available for the region.

The geometric data used here is the SeaRISE [51] 1 km dataset for the whole Greenland ice sheet. It contains bedrock topography from recent CReSIS radar in the Jakobshavn area. We also use the SeaRISE 5 km data set which has

¹ PISM can also do 1 km runs for the whole Greenland ice sheet; see this [news item](#).

climatic mass balance from the Greenland-region climate model RACMO [52].

A regional ice flow model generally needs ice flow and stress boundary conditions. For this we use a 5 km grid, whole ice sheet, spun-up model state from PISM, described in Section *Getting started: a Greenland ice sheet example* of this *Manual*. You can download the large NetCDF result from the PISM website, or you can generate it by running a Section *Getting started: a Greenland ice sheet example* script.

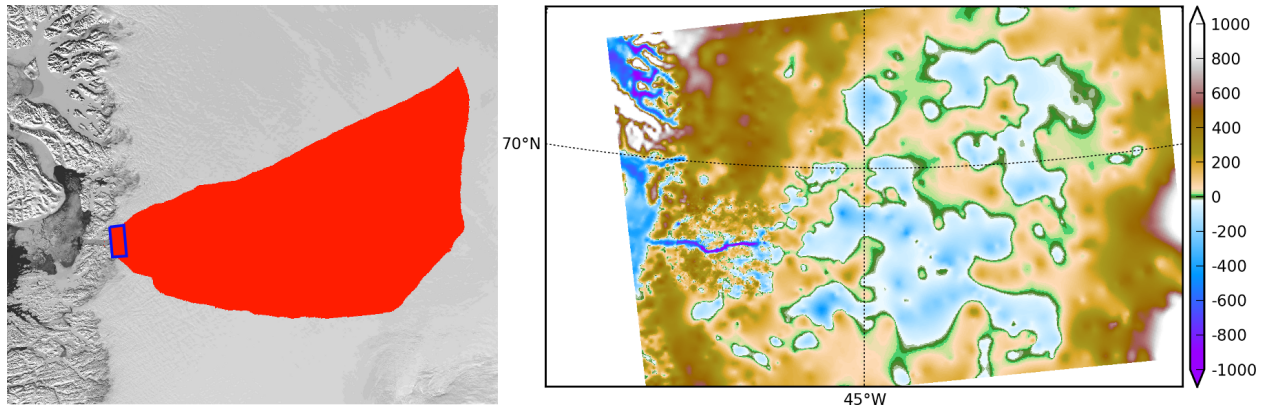


Fig. 3.33: A `regional-tools` script computes a drainage basin mask from the surface DEM (left; Modis background) and from a user-identified terminus rectangle (blue). The regional model can exploit high-resolution bedrock elevations inland from Jakobshavn fjord (right; meters asl).

3.9.1 Get the drainage basin delineation tool

The drainage basin tool `regional-tools` is at <https://github.com/pism/regional-tools>. Get it using `git` and set it up as directed in its `README.md`. Then come back to the `examples/jako/` directory and link the script. Here is the quick summary:

```
cd ~/usr/local/                                # the location you want
git clone https://github.com/pism/regional-tools.git
cd regional-tools/
python setup.py install                        # may add "sudo" or "--user"
cd PISM/examples/jako/
ln -s ~/usr/local/regional-tools/pism_regional.py . # symbolic link to tool
```

3.9.2 Preprocess the data and get the whole ice sheet model file

Script `preprocess.sh` downloads and cleans the 1 km SeaRISE data, an 80 Mb file called `Greenland1km.nc`.² The script also downloads the SeaRISE 5 km data set `Greenland_5km_v1.1.nc`, which contains the RACMO surface mass balance field (not present in the 1 km data set). If you have already run the example in Section *Getting started: a Greenland ice sheet example* then you already have this file and you can link to it to avoid downloading:

```
ln -s ../std-greenland/Greenland_5km_v1.1.nc
```

The same script also preprocesses a pre-computed 5 km grid PISM model result `g5km_gridseq.nc` for the whole ice sheet. This provides the boundary conditions, and the thermodynamical initial condition, for the regional flow model we are building. If you have already generated it by running the script in section *Grid sequencing* then link to it,

² If this file is already present then no actual download occurs, and preprocessing proceeds. Thus: Do not worry about download time if you need to preprocess again. The same comment applies to other downloaded files.


```
ln -s ../std-greenland/g5km_gridseq.nc
```

Otherwise running `preprocess.sh` will download it. Because it is about 0.6 Gb this may take some time.

So now let's actual run the preprocessing script:

```
./preprocess.sh
```

Files `gr1km.nc`, `g5km_climate.nc`, and `g5km_bc.nc` will appear. These can be examined in the usual ways, for example:

```
ncdump -h gr1km.nc | less      # read metadata
ncview gr1km.nc               # view fields
```

The boundary condition file `g5km_bc.nc` contains thermodynamical spun-up variables (`enthalpy`, `bmelt`, `bwat`) and boundary values for the sliding velocity (`u_ssa_bc`, `v_ssa_bc`); these have been extracted from `g5km_gridseq.nc`.

None of the above actions is specific to Jakobshavn, though all are specific to Greenland. If your goal is to build a regional model of another outlet glacier in Greenland, then you may be able to use `preprocess.sh` as is. The SeaRISE 1 km data set has recent CReSIS bed topography data only for the vicinity of the Jakobshavn outlet, however, and it is otherwise just BEDMAP. Because outlet glacier flows are bed-topography-dominated, additional bed elevation data should be sought.

3.9.3 Identify the drainage basin for the modeled outlet glacier

Here we are going to extract a “drainage basin mask” from the surface elevation data (DEM) in `gr1km.nc`. The goal is to determine, in part, the locations outside of the drainage basin where boundary conditions taken from the precomputed whole ice sheet run can be applied to modeling the outlet glacier flow itself.

The basin mask is determined by the gradient flow of the surface elevation. Thus generating the mask uses a highly-simplified ice dynamics model (namely: ice flows down the surface gradient). Once we have the mask, we will apply the full PISM model in the basin interior marked by the mask. Outside the basin mask we will apply simplified models or use the whole ice sheet results as boundary conditions.

The script `pism_regional.py` computes the drainage basin mask based on a user choice of a “terminus rectangle”; see Figure Fig. 3.33. There are two ways to use this script:

- To use the graphical user interface (GUI) mode.

Run

```
python pism_regional.py
```

Select `gr1km.nc` to open. Once the topographic map appears in the Figure window, you may zoom enough to see the general outlet glacier area. Then select the button “Select terminus rectangle”. Use the mouse to select a small rectangle around the Jakobshavn terminus (calving front), or around the terminus of another glacier if you want to model that. Once you have a highlighted rectangle, select a “border width” of at least 50 cells.³ Then click “Compute the drainage basin mask.” Because this is a large data set there will be some delay. (Multi-core users will see that an automatic parallel computation is done.) Finally click “Save the drainage basin mask” and save with your preferred name; we will assume it is called `jakomask.nc`. Then quit.

- To use the command-line interface.

The command-line interface of `pism_regional.py` allows one to re-create the mask without changing the terminus rectangle choice. (It also avoids the slowness of the GUI mode for large data sets.) In fact, for repeatability, we will assume you have used this command to calculate the drainage basin:

³ This recommendation is somewhat Jakobshavn-specific. We want our model to have an ice-free down flow (western) boundary on the resulting computational domain for the modeled region.

```
python pism_regional.py -i gr1km.nc -o jakomask.nc -x 360,382 -y 1135,1176 -b 50
```

This call generates the red region in [Fig. 3.33](#). Options `-x A,B` `-y C,D` identify the grid index ranges of the terminus rectangle, and option `-b` sets the border width. To see more script options, run with `--help`.

3.9.4 Cut out the computational domain for the regional model

We still need to “cut out” from the whole ice sheet geometry data `gr1km.nc` the computational domain for the regional model. The climate data file `g5km_climate.nc` and the boundary condition file `g5km_bc.nc` do not need this action because PISM’s coupling and SSA boundary condition codes already handle interpolation and/or subsampling for such data.

You may have noticed that the text output from running `pism_regional.py` included a `cutout` command which uses `ncks` from the NCO tools. This command also appears as a global attribute of `jakomask.nc`:

```
ncdump -h jakomask.nc | grep cutout
```

Copy and run the command that appears, something like

```
ncks -d x,299,918 -d y,970,1394 gr1km.nc jako.nc
```

This command is also applied to the mask file; note the option `-A` for “append”:

```
ncks -A -d x,299,918 -d y,970,1394 jakomask.nc jako.nc
```

Now look at `jako.nc`, for example with “`ncview -minmax all jako.nc`”. This file is the full geometry data ready for a regional model. The field `ftt_mask` identifies the drainage basin, outside of which we will use simplified time-independent boundary conditions. Specifically, outside of the `ftt_mask` area, but within the computational domain defined by the extent of `jako.nc`, we will essentially keep the initial thickness. Inside the `ftt_mask` area all fields will evolve normally.

3.9.5 Quick start

The previous steps starting with the command “`./preprocess.sh`” above, then using the command-line version of `pism_regional.py`, and then doing the `ncks` cut-out steps, are all accomplished in one script,

```
./quickjakosetup.sh
```

Running this takes about a minute on a fast laptop, assuming data files are already downloaded.

3.9.6 Spinning-up the regional model on a 5 km grid

To run the PISM regional model we will need to know the number of grid points in the 1 km grid in `jako.nc`. Do this:

```
ncdump -h jako.nc | head
netcdf jako {
  dimensions:
    y = 425 ;
    x = 620 ;
  ...
```

The grid has spacing of 1 km, so our computational domain is a 620 km by 425 km rectangle. A 2 km resolution, century-scale model run is easily achievable on a desktop or laptop computer, and that is our goal below. A lower 5 km

resolution spin-up run, matching the resolution of the 5 km whole ice sheet state computed earlier, is also achievable on a small computer; we do that first.

The boundary condition fields in `g5km_bc.nc`, from the whole ice sheet model result `g5km_gridseq.nc`, may or may not, depending on modeller intent, be spun-up adequately for the purposes of the regional model. For instance, the intention may be to study equilibrium states with model settings special to the region. Here, however we assume that some regional spin-up is needed, if for no other reason that the geometry used here (from the SeaRISE 1km data set) differs from that in the whole ice sheet model state.

We will get first an equilibrium 5 km regional model, and then do a century run of a 2 km model based on that. While determining “equilibrium” requires a decision, of course, a standard satisfied here is that the ice volume in the region changes by less than 0.1 percent in the final 100 model years. See `ice_volume_glacierized` in `ts_spunjako_0.nc` below.

The 5 km grid⁴ uses `-Mx 125 -My 86`. So now we do a basic run using 4 MPI processes:

```
./spinup.sh 4 125 86 &> out.spin5km &
```

You can read the stdout log file while it runs: “`less out.spin5km`”. The run takes about 4.4 processor-hours on a 2016 laptop. It produces three files which can be viewed (e.g. with `ncview`): `spunjako_0.nc`, `ts_spunjako_0.nc`, and `ex_spunjako_0.nc`. Some more comments on this run are appropriate:

- Generally the regridding techniques used at the start of this spin-up run are recommended for regional modeling. Read the actual run command by

```
PISM_D0=echo ./spinup.sh 4 125 86 | less
```

- We use `-i jako.nc -bootstrap`, so we get to choose our grid, and (as usual in PISM with `-bootstrap`) the fields are interpolated to our grid.
- A modestly-fine vertical grid with 20 m spacing is chosen, but even finer is recommended, especially to resolve the temperate ice layer in these outlet glaciers.
- There is an option `-no_model_strip 10` asking `pismr -regional` to put a 10 km strip around edge of the computational domain. This strip is entirely outside of the drainage basin defined by `ftt_mask`. In this strip the thermodynamical spun-up variables `bmelt`, `tillwat`, `enthalpy`, `litho_temp` from `g5km_bc.nc` are held fixed and used as boundary conditions for the conservation of energy model. A key part of putting these boundary conditions into the model strip are the options

```
-regrid_file g5km_bc.nc -regrid_vars bmelt,tillwat,enthalpy,litho_temp,vel_ssa_bc
```

- Dirichlet boundary conditions `u_ssa_bc`, `v_ssa_bc` are also regridded from `g5km_bc.nc` for the sliding SSA stress balance, and the option `-ssa_dirichlet_bc` then uses them during the run. The SSA equations are solved as usual except in the `no_model_strip` where these Dirichlet boundary conditions are used. Note that the velocity tangent to the north and south edges of the computational domain is significantly nonzero, which motivates this usage.
- The calving front of the glacier is handled by the following option combination:

```
-calving ocean_kill -ocean_kill_file jako.nc -pik
```

This choice uses the present-day ice extent, defined by SeaRISE data in `Greenland1km.nc`, to determine the location of the calving front. Recalling that `-pik` includes `-cfbc`, we are applying a PIK mechanism for the stress boundary condition at the calving front. The other PIK mechanisms are largely inactive because of `-calving ocean_kill`, but they should do no harm (see section *PIK options for marine ice sheets*).

⁴ Calculate $620/5 + 1$ and $425/5 + 1$, for example.

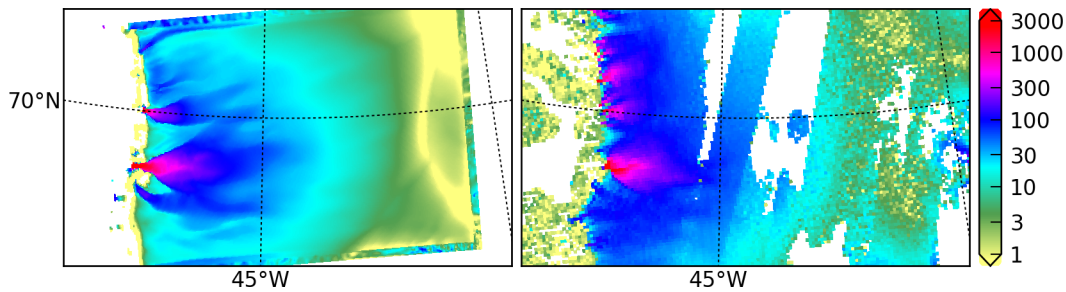


Fig. 3.34: Left: modeled surface speed at the end of a 2 km grid, 100 model year, steady present-day climate run. Right: observed surface speed, an average of four winter velocity maps (2000, 2006–2008) derived from RADARSAT data, as included in the SeaRISE 5 km data set [53], for the same region. Scales are in meters per year.

3.9.7 Century run on a 2 km grid

Now that we have a spun-up state, here is a 100 model year run on a 2 km grid with a 10 m grid in the vertical:

```
./century.sh 4 311 213 spunjako_0.nc &> out.2km_100a &
```

This run requires at least 6 GB of memory, and it takes about 16 processor-hours.

It produces a file `jakofine_short.nc` almost immediately and then restarts from it because we need to regrid fields from the end of the previous 5 km regional run (in `spunjako_0.nc`) and then to “go back” and regrid the SSA boundary conditions from the 5 km whole ice sheet results `g5km_bc.nc`. At the end of the run the final file `jakofine.nc` is produced. Also there is a time-series file `ts_jakofine.nc` with monthly scalar time-series and a spatial time-dependent file `ex_jakofine.nc`. The surface speed at the end of this run is shown in Fig. 3.34, with a comparison to observations.

Over this 100 year period the flow appears to be relatively steady state. Though this is not surprising because the climate forcing and boundary conditions are time-independent, a longer run reveals ongoing speed variability associated to subglacially-driven sliding cyclicity; compare [54].

The ice dynamics parameters chosen in `spinup.sh` and `century.sh`, especially the combination

```
-topg_to_phi 15.0,40.0,-300.0,700.0 -till_effective_fraction_overburden 0.02 \
-pseudo_plastic -pseudo_plastic_q 0.25 -tauc_slippery_grounding_lines
```

are a topic for a parameter study (compare [55]) or a study of their relation to inverse modeling results (e.g. [56]).

3.9.8 Plotting results

Fig. 3.34 was generated using `pypismtools`, `NCO` and `CDO`. Do

```
ncpdq -a time,z,y,x spunjako_0.nc jako5km.nc
nc2cdo.py jako5km.nc
cdo remapbil,jako5km.nc Greenland_5km_v1.1.nc Greenland_5km_v1.1_jako.nc # FIXME: if fails,
↪ proceed?
ncap2 -O -s "velsurf_mag=surfvelmag*1.;" Greenland_5km_v1.1_jako.nc \
Greenland_5km_v1.1_jako.nc
basemap-plot.py -v velsurf_mag --singlerow -o jako-velsurf_mag.png jakofine.nc \
Greenland_5km_v1.1_jako.nc
```

To choose a colormap `foo.cpt` add option `--colormap foo.cpt` in the last command. For this example `PyPISMTools/colormaps/Full_saturation_spectrum_CCW.cpt` was used.

3.10 Configuration parameters

Every parameter corresponds to a command-line option; some options map to a combination of parameters.

There are five kinds of options and parameters:

1. Strings (mostly file names),
2. Scalars (a number representing a physical quantify, usually with units),
3. Integers (an integer number, usually corresponding to a count),
4. Booleans (True/False, on/off, yes/no),
5. Keywords (one of a given set of strings).

Each parameter can be set using the command-line option consisting of a dash followed by the parameter name. For example,

```
-constants.standard_gravity 10
```

sets the acceleration due to gravity (parameter `constants.standard_gravity`) to 10. Options listed below are *shortcuts*, added for convenience.

The following are equivalent and choose the temperature-based (as opposed to enthalpy-based) energy balance model:

```
-energy.temperature_based
-energy.temperature_based on
-energy.temperature_based yes
-energy.temperature_based true
-energy.temperature_based True
```

The following are also equivalent: they disable updating geometry by performing a step of the mass-continuity equation:

```
-geometry.update.enabled off
-geometry.update.enabled no
-geometry.update.enabled false
-geometry.update.enabled False
-no_geometry.update.enabled
```

The `-no_` prefix is still supported for compatibility with older scripts, but will be removed in a later PISM version.

1. `age.enabled` (*boolean*)

Value no

Option `-age`

Description Solve age equation (advection equation for ice age).

2. `age.initial_value` (*scalar*)

Value 0 (years)

Description Initial age of ice

3. `atmosphere.fausto_air_temp.c_ma` (*scalar*)

Value -0.718900 (Kelvin / degree_north)

Description latitude-dependence coefficient for formula (1) in [1]

4. `atmosphere.fausto_air_temp.c_mj` (*scalar*)

- Value** -0.158500 (Kelvin / degree_north)
- Description** latitude-dependence coefficient for formula (2) in [1]
5. `atmosphere.fausto_air_temp.d_ma` (*scalar*)
- Value** 314.980000 (Kelvin)
- Description** 41.83+273.15; base temperature for formula (1) in [1]
6. `atmosphere.fausto_air_temp.d_mj` (*scalar*)
- Value** 287.850000 (Kelvin)
- Description** = 14.70+273.15; base temperature for formula (2) in [1]
7. `atmosphere.fausto_air_temp.gamma_ma` (*scalar*)
- Value** -0.006309 (Kelvin / meter)
- Description** = -6.309 / 1km; mean slope lapse rate for formula (1) in [1]
8. `atmosphere.fausto_air_temp.gamma_mj` (*scalar*)
- Value** -0.005426 (Kelvin / meter)
- Description** = -5.426 / 1km; mean slope lapse rate for formula (2) in [1]
9. `atmosphere.fausto_air_temp.kappa_ma` (*scalar*)
- Value** 0.067200 (Kelvin / degree_west)
- Description** longitude-dependence coefficient for formula (1) in [1]
10. `atmosphere.fausto_air_temp.kappa_mj` (*scalar*)
- Value** 0.051800 (Kelvin / degree_west)
- Description** longitude-dependence coefficient for formula (2) in [1]
11. `atmosphere.fausto_air_temp.summer_peak_day` (*integer*)
- Value** 196
- Description** day of year for July 15; used in corrected formula (4) in [1]
12. `atmosphere.precip_exponential_factor_for_temperature` (*scalar*)
- Value** 0.070417 (Kelvin-1)
- Description** = 0.169/2.4; in SeaRISE-Greenland formula for paleo-precipitation from present; a 7.3% change of precipitation rate for every one degC of temperature change [2]
13. `basal_resistance.beta_ice_free_bedrock` (*scalar*)
- Value** 1.800000e+09 (Pascal second meter-1)
- Description** value is for ice stream E from [90]; thus sliding velocity, but we hope it doesn't matter much; at 100 m/year the linear sliding law gives 57040 Pa basal shear stress
14. `basal_resistance.plastic.regularization` (*scalar*)
- Value** 0.010000 (meter / year)
- Option** -plastic_reg
- Description** Set the value of ϵ regularization of plastic till; this is the second ϵ in formula (4.1) in [33]
15. `basal_resistance.pseudo_plastic.enabled` (*boolean*)

- Value** no
- Option** -pseudo_plastic
- Description** Use the pseudo-plastic till model (basal sliding law).
16. basal_resistance.pseudo_plastic.q (*scalar*)
- Value** 0.250000 (pure number)
- Option** -pseudo_plastic_q
- Description** The exponent of the pseudo-plastic basal resistance model
17. basal_resistance.pseudo_plastic.sliding_scale_factor (*scalar*)
- Value** -1 (1)
- Option** -sliding_scale_factor_reduces_tauc
- Description** divides pseudo-plastic τ_{auc} (yield stress) by given factor; this would increase sliding by given factor in absence of membrane stresses; not used if negative or zero; not used by default
18. basal_resistance.pseudo_plastic.u_threshold (*scalar*)
- Value** 100 (meter / year)
- Option** -pseudo_plastic_uthreshold
- Description** threshold velocity of the pseudo-plastic sliding law
19. basal_yield_stress.add_transportable_water (*boolean*)
- Value** no
- Option** -tauc_add_transportable_water
- Description** If 'yes' then the water amount in the transport system is added to τ_{wat} in determining τ_{auc} (in the Mohr-Coulomb relation). Normally only the water in the till is used.
20. basal_yield_stress.constant.value (*scalar*)
- Value** 200000 (Pascal)
- Option** -tauc
- Description** fill value for yield stress for basal till (plastic or pseudo-plastic model); note 2×10^5 Pa = 2.0 bar is quite strong and little sliding should occur without an explicit τ_{auc} choice altering this default
21. basal_yield_stress.ice_free_bedrock (*scalar*)
- Value** 1000000 (Pascal)
- Option** -high_tauc
- Description** the 'high' yield stress value used in grounded ice-free areas.
22. basal_yield_stress.model (*keyword*)
- Value** mohr_coulomb
- Choices** constant, mohr_coulomb
- Option** -yield_stress
- Description** The basal yield stress model to use when sliding is active.
23. basal_yield_stress.mohr_coulomb.till_cohesion (*scalar*)

- Value** 0 (Pascal)
- Option** -till_cohesion
- Description** cohesion of till; = c_0 in most references; note Schoof uses zero but Paterson pp 168–169 gives range 0–40 kPa; but Paterson notes that ‘... all the pairs c_0 and ϕ in the table would give a yield stress for Ice Stream B that exceeds the basal shear stress there...’
24. basal_yield_stress.mohr_coulomb.till_compressibility_coefficient (*scalar*)
- Value** 0.120000 (pure number)
- Option** -till_compressibility_coefficient
- Description** coefficient of compressibility of till; value from [81]
25. basal_yield_stress.mohr_coulomb.till_effective_fraction_overburden (*scalar*)
- Value** 0.020000 (pure number)
- Option** -till_effective_fraction_overburden
- Description** δ in notes; $N_0 = \delta P_o$ where P_o is overburden pressure; N_0 is reference (low) value of effective pressure (i.e. normal stress); N_0 scales with overburden pressure unlike [81]; default value from Greenland and Antarctic model runs
26. basal_yield_stress.mohr_coulomb.till_log_factor_transportable_water (*scalar*)
- Value** 0.100000 (meters)
- Option** -till_log_factor_transportable_water
- Description** If basal_yield_stress.add_transportable_water = yes then the water amount in the transport system is added to tillwat in determining τ_{auc} . Normally only the water in the till is used. This factor multiplies the logarithm in that formula.
27. basal_yield_stress.mohr_coulomb.till_phi_default (*scalar*)
- Value** 30 (degrees)
- Option** -plastic_phi
- Description** fill value for till friction angle
28. basal_yield_stress.mohr_coulomb.till_reference_effective_pressure (*scalar*)
- Value** 1000 (Pascal)
- Description** reference effective pressure N_0 ; value from [81]
29. basal_yield_stress.mohr_coulomb.till_reference_void_ratio (*scalar*)
- Value** 0.690000 (pure number)
- Option** -till_reference_void_ratio
- Description** void ratio at reference effective pressure N_0 ; value from [81]
30. basal_yield_stress.mohr_coulomb.topg_to_phi.enabled (*boolean*)
- Value** no
- Description** If THE OPTION -topg_to_phi IS SET THEN THIS WILL BE SET TO ‘yes’, and then MohrCoulombYieldStress will initialize the tillphi field using a piece-wise linear function of depth described by four parameters.
31. basal_yield_stress.mohr_coulomb.topg_to_phi.phi_max (*scalar*)
- Value** 15 (degrees)

- Description** upper value of the till friction angle; see the implementation of MohrCoulombYieldStress
32. `basal_yield_stress.mohr_coulomb.topg_to_phi.phi_min` (*scalar*)
- Value** 5 (degrees)
- Description** lower value of the till friction angle; see the implementation of MohrCoulombYieldStress
33. `basal_yield_stress.mohr_coulomb.topg_to_phi.topg_max` (*scalar*)
- Value** 1000 (meters)
- Description** the elevation at which the upper value of the till friction angle is used; see the implementation of MohrCoulombYieldStress
34. `basal_yield_stress.mohr_coulomb.topg_to_phi.topg_min` (*scalar*)
- Value** -1000 (meters)
- Description** the elevation at which the lower value of the till friction angle is used; see the implementation of MohrCoulombYieldStress
35. `basal_yield_stress.slippery_grounding_lines` (*boolean*)
- Value** no
- Option** -tauc_slippery_grounding_lines
- Description** If yes, at icy grounded locations with bed elevations below sea level, within one cell of floating ice or ice-free ocean, make `tauc` as low as possible from the Mohr-Coulomb relation. Specifically, at such locations replace the normally-computed `tauc` from the Mohr-Coulomb relation, which uses the effective pressure from the modeled amount of water in the till, by the minimum value of `tauc` from Mohr-Coulomb, i.e. by using the effective pressure corresponding to the maximum amount of till-stored water. Does not alter the modeled or reported amount of till water, nor does this mechanism affect water conservation.
36. `bed_deformation.lc.elastic_model` (*boolean*)
- Value** no
- Option** -bed_def_lc_elastic_model
- Description** Use the elastic part of the Lingle-Clark bed deformation model.
37. `bed_deformation.lc.grid_size_factor` (*integer*)
- Value** 4
- Description** The spectral grid size is $(Z*(\text{grid.Mx} - 1) + 1, Z*(\text{grid.My} - 1) + 1)$ where Z is given by this parameter. See [83], [84]
38. `bed_deformation.mantle_density` (*scalar*)
- Value** 3300 (kg meter-3)
- Description** half-space (mantle) density used by the bed deformation model. See [83], [84]
39. `bed_deformation.lithosphere_flexural_rigidity` (*scalar*)
- Value** 5.000000e+24 (Newton meter)
- Description** lithosphere flexural rigidity used by the bed deformation model. See [83], [84]
40. `bed_deformation.mantle_viscosity` (*scalar*)
- Value** 1.000000e+21 (Pascal second)

- Description** half-space (mantle) viscosity used by the bed deformation model. See [83], [84]
41. `bed_deformation.model` (*keyword*)
- Value** none
- Choices** none, iso, lc
- Option** -bed_def
- Description** Selects a bed deformation model to use. ‘iso’ is point-wise isostasy, ‘lc’ is the Lingle-Clark model (see [83], requires FFTW3).
42. `bed_deformation.update_interval` (*scalar*)
- Value** 10 (years)
- Description** Interval between bed deformation updates
43. `bed_deformation.bed_topography_delta_file` (*string*)
- Value** no default
- Option** -topg_delta_file
- Description** The name of the file to read the topg_delta from. This field is added to the bed topography during initialization.
44. `bed_deformation.bed_uplift_file` (*string*)
- Value** no default
- Option** -uplift_file
- Description** The name of the file to read the uplift (dbdt) from. Leave empty to read it from an input file or a regridding file.
45. `bootstrapping.defaults.bed` (*scalar*)
- Value** 1 (meters)
- Description** bed elevation value to use if topg (bedrock_altitude) variable is absent in bootstrapping file
46. `bootstrapping.defaults.bmelt` (*scalar*)
- Value** 0 (meter / second)
- Description** basal melt rate value to use if variable bmelt is absent in bootstrapping file
47. `bootstrapping.defaults.bwat` (*scalar*)
- Value** 0 (meters)
- Description** till water thickness value to use if variable tillwat is absent in bootstrapping file
48. `bootstrapping.defaults.bwp` (*scalar*)
- Value** 0 (Pascal)
- Description** basal water pressure value to use if variable bwp is absent in bootstrapping file; most hydrology models do not use this value because bwp is diagnostic
49. `bootstrapping.defaults.enwat` (*scalar*)
- Value** 0 (meters)
- Description** effective englacial water thickness value to use if variable enwat is absent in bootstrapping file

50. `bootstrapping.defaults.geothermal_flux` (*scalar*)
Value 0.042000 (Watt meter-2)
Description geothermal flux value to use if bheatflx variable is absent in bootstrapping file
51. `bootstrapping.defaults.ice_thickness` (*scalar*)
Value 0 (meters)
Description thickness value to use if thk (land_ice_thickness) variable is absent in bootstrapping file
52. `bootstrapping.defaults.tillwat` (*scalar*)
Value 0 (meters)
Description till water thickness value to use if variable tillwat is absent in bootstrapping file
53. `bootstrapping.defaults.uplift` (*scalar*)
Value 0 (meter / second)
Description uplift value to use if dbdt variable is absent in bootstrapping file
54. `bootstrapping.temperature_heuristic` (*keyword*)
Value smb
Choices smb, quartic_guess
Option -boot_temperature_heuristic
Description The heuristic to use to initialize ice temperature during bootstrapping: 'smb' uses the new method using the surface mass balance, surface temperature, and the geothermal flux, 'quartic_guess' uses the old method using the surface temperature and the geothermal flux.
55. `calving.eigen_calving.K` (*scalar*)
Value 0 (meter second)
Option -eigen_calving_K
Description Set proportionality constant to determine calving rate from strain rates. Note references [70], [6] use K in range 10^9 to 3×10^{11} m a, that is, 3×10^{16} to 10^{19} m s.
56. `calving.float_kill.margin_only` (*boolean*)
Value no
Option -float_kill_margin_only
Description Apply float_kill at ice margin cells only.
57. `calving.float_kill.calve_near_grounding_line` (*boolean*)
Value yes
Option -float_kill_calve_near_grounding_line
Description Calve floating ice near the grounding line.
58. `calving.front_retreat.wrap_around` (*boolean*)
Value false
Option -calving_wrap_around
Description If true, wrap around domain boundaries. This may be needed in some regional synthetic geometry setups.

59. calving.front_retreat.use_cfl (*boolean*)

Value false

Option -calving_cfl

Description apply CFL criterion for eigen-calving rate front retreat

60. calving.methods (*string*)

Value no default

Option -calving

Description comma-separated list of calving methods; one or more of 'eigen_calving', 'ocean_kill', 'float_kill', 'thickness_calving'

61. calving.thickness_calving.threshold (*scalar*)

Value 50 (meters)

Option -thickness_calving_threshold

Description When terminal ice thickness of floating ice shelf is less than this threshold, it will be calved off.

62. calving.thickness_calving.threshold_file (*string*)

Value no default

Option -thickness_calving_threshold_file

Description Name of the file containing the spatially-variable thickness calving threshold.

63. calving.vonmises.sigma_max (*scalar*)

Value 1000000 (Pa)

Option -vonmises_calving_sigma_max

Description Set maximum tensile stress. Note references [71] use 1.0e6 Pa.

64. climate_forcing.buffer_size (*integer*)

Value 60

Description number of 2D climate forcing records to keep in memory; = 5 years of monthly records

65. climate_forcing.evaluations_per_year (*integer*)

Value 52

Description length of the time-series used to compute temporal averages of forcing data (such as mean annual temperature)

66. constants.fresh_water.density (*scalar*)

Value 1000 (kg meter-3)

Description density of fresh water

67. constants.fresh_water.latent_heat_of_fusion (*scalar*)

Value 334000 (Joule / kg)

Description latent heat of fusion for water [61]

68. constants.fresh_water.melting_point_temperature (*scalar*)

Value 273.150000 (Kelvin)

- Description** melting point of pure water
69. `constants.fresh_water.specific_heat_capacity` (*scalar*)
- Value** 4170 (Joule / (kg Kelvin))
- Description** at melting point T_0 [61]
70. `constants.ice.beta_Clausius_Clapeyron` (*scalar*)
- Value** 7.900000e-08 (Kelvin / Pascal)
- Description** Clausius-Clapeyron constant relating melting temperature and pressure: $\beta = dT/dP$ [91]
71. `constants.ice.density` (*scalar*)
- Value** 910 (kg meter-3)
- Description** ρ_i ; density of ice in ice sheet
72. `constants.ice.grain_size` (*scalar*)
- Value** 1 (mm)
- Option** `-ice_grain_size`
- Description** Default constant ice grain size to use with the Goldsby-Kohlstedt [63] flow law
73. `constants.ice.specific_heat_capacity` (*scalar*)
- Value** 2009 (Joule / (kg Kelvin))
- Description** specific heat capacity of pure ice at melting point T_0
74. `constants.ice.thermal_conductivity` (*scalar*)
- Value** 2.100000 (Joule / (meter Kelvin second))
- Description** = W m-1 K-1; thermal conductivity of pure ice
75. `constants.ideal_gas_constant` (*scalar*)
- Value** 8.314410 (Joule / (mol Kelvin))
- Description** ideal gas constant
76. `constants.sea_water.density` (*scalar*)
- Value** 1028 (kg meter-3)
- Description** density of sea water
77. `constants.sea_water.specific_heat_capacity` (*scalar*)
- Value** 3985 (Joule / (kg Kelvin))
- Description** at 35 psu, value taken from http://www.kayelaby.npl.co.uk/general_physics/2_7/2_7_9.html
78. `constants.standard_gravity` (*scalar*)
- Value** 9.810000 (meter second-2)
- Description** acceleration due to gravity on Earth geoid
79. `energy.allow_temperature_above_melting` (*boolean*)
- Value** no

Description If set to ‘yes’, allow temperatures above the pressure-melting point in the cold mode temperature code. Used by some verification tests.

80. `energy.basal_melt.use_grounded_cell_fraction` (*boolean*)

Value true

Option `-subgl_basal_melt`

Description If `geometry.grounded_cell_fraction` is set, use the fractional floatation mask to interpolate the basal melt rate at the grounding line between grounded and floating values.

81. `energy.bedrock_thermal_conductivity` (*scalar*)

Value 3 (Joule / (meter Kelvin second))

Description = W m⁻¹ K⁻¹; for bedrock used in thermal model [3]

82. `energy.bedrock_thermal_density` (*scalar*)

Value 3300 (kg meter⁻³)

Description for bedrock used in thermal model

83. `energy.bedrock_thermal_specific_heat_capacity` (*scalar*)

Value 1000 (Joule / (kg Kelvin))

Description for bedrock used in thermal model [3]

84. `energy.drainage_maximum_rate` (*scalar*)

Value 1.584438e-09 (second⁻¹)

Description 0.05 year⁻¹; maximum rate at which liquid water fraction in temperate ice could possibly drain; see [23]

85. `energy.drainage_target_water_fraction` (*scalar*)

Value 0.010000 (1)

Description liquid water fraction (omega) above which drainage occurs, but below which there is no drainage; see [23]

86. `energy.enabled` (*boolean*)

Value yes

Description Solve energy conservation equations.

87. `energy.enthalpy_cold_bulge_max` (*scalar*)

Value 60270 (Joule / kg)

Description = (2009 J kg⁻¹ K⁻¹) * (30 K); maximum amount by which advection can reduce the enthalpy of a column of ice below its surface enthalpy value

88. `energy.max_low_temperature_count` (*integer*)

Value 10

Option `-max_low_temps`

Description Maximum number of grid points with ice temperature below `energy.minimum_allowed_temperature`.

89. `energy.minimum_allowed_temperature` (*scalar*)

Value 200 (Kelvin)

- Option** `-low_temp`
- Description** Minimum allowed ice temperature
90. `energy.temperate_ice_enthalpy_conductivity_ratio` (*scalar*)
- Value** 0.100000 (pure number)
- Description** K in cold ice is multiplied by this fraction to give K0 in [23]
91. `energy.temperature_based` (*boolean*)
- Value** no
- Description** Use cold ice (i.e. not polythermal) methods.
92. `energy.temperature_dependent_thermal_conductivity` (*boolean*)
- Value** no
- Option** `-vark`
- Description** If yes, use `varkenthSystemCtx` class in the energy step. It is base on formula (4.37) in [45]. Otherwise use `enthSystemCtx`, which has temperature-independent thermal conductivity set by constant `ice.thermal_conductivity`.
93. `enthalpy_converter.T_reference` (*scalar*)
- Value** 223.150000 (Kelvin)
- Description** = T_0 in enthalpy formulas in [23]
94. `enthalpy_converter.relaxed_is_temperate_tolerance` (*scalar*)
- Value** 0.001000 (Kelvin)
- Description** Tolerance within which ice is treated as temperate (cold-ice mode and diagnostics).
95. `flow_law.Hooke.A` (*scalar*)
- Value** 4.421650e-09 (Pascal-3 second-1)
- Description** $A_{\text{Hooke}} = (1/B_0)^n$ where $n=3$ and $B_0 = 1.928 \text{ a}^{1/3} \text{ Pa}$. See [66]
96. `flow_law.Hooke.C` (*scalar*)
- Value** 0.166120 (Kelvin^{flow_law.Hooke.k})
- Description** See [66]
97. `flow_law.Hooke.Q` (*scalar*)
- Value** 78800 (Joule / mol)
- Description** Activation energy, see [66]
98. `flow_law.Hooke.Tr` (*scalar*)
- Value** 273.390000 (Kelvin)
- Description** See [66]
99. `flow_law.Hooke.k` (*scalar*)
- Value** 1.170000 (pure number)
- Description** See [66]
100. `flow_law.Paterson_Budd.A_cold` (*scalar*)
- Value** 3.610000e-13 (Pascal-3 / second)

- Description** Paterson-Budd A_cold, see [62]
101. `flow_law.Paterson_Budd.A_warm` (*scalar*)
- Value** 1730 (Pascal-3 / second)
- Description** Paterson-Budd A_warm, see [62]
102. `flow_law.Paterson_Budd.Q_cold` (*scalar*)
- Value** 60000 (Joule / mol)
- Description** Paterson-Budd Q_cold, see [62]
103. `flow_law.Paterson_Budd.Q_warm` (*scalar*)
- Value** 139000 (Joule / mol)
- Description** Paterson-Budd Q_warm, see [62]
104. `flow_law.Paterson_Budd.T_critical` (*scalar*)
- Value** 263.150000 (Kelvin)
- Description** Paterson-Budd critical temperature, see [62]
105. `flow_law.Schoof_regularizing_length` (*scalar*)
- Value** 1000 (km)
- Description** Regularizing length (Schoof definition)
106. `flow_law.Schoof_regularizing_velocity` (*scalar*)
- Value** 1 (meter / year)
- Description** Regularizing velocity (Schoof definition)
107. `flow_law.gpbld.water_frac_coeff` (*scalar*)
- Value** 181.250000 (pure number)
- Description** coefficient in Glen-Paterson-Budd flow law for extra dependence of softness on liquid water fraction (omega) [45], [64]
108. `flow_law.gpbld.water_frac_observed_limit` (*scalar*)
- Value** 0.010000 (1)
- Description** maximum value of liquid water fraction omega for which softness values are parameterized by [64]; used in Glen-Paterson-Budd-Lliboutry-Duval flow law; compare [23]
109. `flow_law.isothermal_Glen.ice_softness` (*scalar*)
- Value** 3.168900e-24 (Pascal-3 second-1)
- Description** ice softness used by IsothermalGlenIce [22]
110. `fracture_density.constant_fd` (*boolean*)
- Value** no
- Option** -constant_fd
- Description** FIXME
111. `fracture_density.constant_healing` (*boolean*)
- Value** no
- Option** -constant_healing

- Description** Constant healing
112. `fracture_density.enabled` (*boolean*)
- Value** no
- Option** -fractures
- Description** Calculation of fracture density according to stresses and strain rate field.
113. `fracture_density.fd2d_scheme` (*boolean*)
- Value** no
- Option** -scheme_fd2d
- Description** FIXME
114. `fracture_density.fracture_weighted_healing` (*boolean*)
- Value** no
- Option** -fracture_weighted_healing
- Description** Fracture weighted healing
115. `fracture_density.include_grounded_ice` (*boolean*)
- Value** no
- Option** -do_frac_on_grounded
- Description** model fracture density in grounded areas
116. `fracture_density.lefm` (*boolean*)
- Value** no
- Option** -lefm
- Description** FIXME
117. `fracture_density.max_shear_stress` (*boolean*)
- Value** no
- Option** -max_shear
- Description** Use the max. shear stress criterion.
118. `fracture_density.phi0` (*scalar*)
- Value** 0 (1)
- Option** -phi0
- Description** FIXME
119. `fracture_density.softening_lower_limit` (*scalar*)
- Value** 1 (1)
- Option** -fracture_softening
- Description** epsilon in equation (6) in Albrecht and Levermann, 'Fracture-induced softening for large-scale ice dynamics'
120. `fracture_density.write_fields` (*boolean*)
- Value** no

Option -write_fd_fields

Description Writing of fracture density related fields to nc-file.

121. geometry.grounded_cell_fraction (*boolean*)

Value false

Option -subgl

Description Linear interpolation scheme ('LI' in Gladstone et al. 2010) expanded to two dimensions is used if switched on in order to evaluate the position of the grounding line on a subgrid scale.

122. geometry.ice_free_thickness_standard (*scalar*)

Value 0.010000 (meters)

Description If ice is thinner than this standard then the mask is set to MASK_ICE_FREE_BEDROCK or MASK_ICE_FREE_OCEAN.

123. geometry.part_grid.enabled (*boolean*)

Value no

Option -part_grid

Description apply partially filled grid cell scheme

124. geometry.remove_icebergs (*boolean*)

Value no

Option -kill_icebergs

Description identify and kill detached ice-shelf areas

125. geometry.update.enabled (*boolean*)

Value yes

Option -mass

Description Solve the mass conservation equation

126. geometry.update.use_basal_melt_rate (*boolean*)

Value yes

Option -bmr_in_cont

Description Include basal melt rate in the continuity equation

127. grid.allow_extrapolation (*boolean*)

Value no

Option -allow_extrapolation

Description Allow extrapolation during regridding.

128. grid.Lbz (*scalar*)

Value 1000 (meters)

Description Thickness of the thermal bedrock layer. (Inactive if grid.Mbz < 2)

129. grid.Lx (*scalar*)

Value 1500000 (meters)

Description Default computational box is 3000 km x 3000 km (= 2 Lx x 2 Ly) in horizontal.

130. `grid.Ly` (*scalar*)
Value 1500000 (meters)
Description Default computational box is 3000 km x 3000 km (= 2 Lx x 2 Ly) in horizontal.
131. `grid.Lz` (*scalar*)
Value 4000 (meters)
Option -Lz
Description Height of the computational domain.
132. `grid.Mbz` (*integer*)
Value 1
Description Number of thermal bedrock layers; 1 level corresponds to no bedrock.
133. `grid.Mx` (*integer*)
Value 61
Option -Mx
Description Number of grid points in the x direction.
134. `grid.My` (*integer*)
Value 61
Option -My
Description Number of grid points in the y direction.
135. `grid.Mz` (*integer*)
Value 31
Option -Mz
Description Number of vertical grid levels in the ice.
136. `grid.correct_cell_areas` (*boolean*)
Value yes
Description Compute corrected cell areas using WGS84 datum (for ice area and volume computations).
137. `grid.ice_vertical_spacing` (*keyword*)
Value quadratic
Choices quadratic, equal
Option -z_spacing
Description vertical spacing in the ice
138. `grid.lambda` (*scalar*)
Value 4 (pure number)
Description Vertical grid spacing parameter. Roughly equal to the factor by which the grid is coarser at an end away from the ice-bedrock interface.
139. `grid.max_stencil_width` (*integer*)
Value 2

- Description** Maximum width of the finite-difference stencil used in PISM.
140. `grid.registration` (*keyword*)
- Value** center
- Choices** center, corner
- Description** horizontal grid registration
141. `grid.periodicity` (*keyword*)
- Value** xy
- Choices** none, x, y, xy
- Option** -periodicity
- Description** horizontal grid periodicity
142. `hydrology.cavitation_opening_coefficient` (*scalar*)
- Value** 0.500000 (meter-1)
- Option** -hydrology_cavitation_opening_coefficient
- Description** c_1 in notes; coefficient of cavitation opening term in evolution of layer thickness in hydrology::Distributed
143. `hydrology.const_bmelt` (*scalar*)
- Value** 3.168876e-10 (meter / second)
- Option** -hydrology_const_bmelt
- Description** default value is equivalent to 1 cm per year of melt; only used if hydrology.use_const_bmelt = 'yes'
144. `hydrology.creep_closure_coefficient` (*scalar*)
- Value** 0.040000 (pure number)
- Option** -hydrology_creep_closure_coefficient
- Description** c_2 in notes; coefficient of creep closure term in evolution of layer thickness in hydrology::Distributed
145. `hydrology.gradient_power_in_flux` (*scalar*)
- Value** 1.500000 (pure number)
- Option** -hydrology_gradient_power_in_flux
- Description** power β in Darcy's law $q = -kW^\alpha |\nabla\psi|^{\beta-2} \nabla\psi$, for subglacial water layer; used by hydrology::Routing and hydrology::Distributed
146. `hydrology.hydraulic_conductivity` (*scalar*)
- Value** 0.001000 ($m^{2\beta-\alpha} s^{2\beta-3} kg^{1-\beta}$)
- Option** -hydrology_hydraulic_conductivity
- Description** = k in notes; lateral conductivity, in Darcy's law, for subglacial water layer; units depend on powers alpha = hydrology.thickness_power_in_flux and beta = hydrology.potential_gradient_power_in_flux; used by hydrology::Routing and hydrology::Distributed
147. `hydrology.maximum_time_step` (*scalar*)
- Value** 1 (years)

- Description** maximum allowed time step length used by hydrology::Routing and hydrology::Distributed
148. `hydrology.model` (*keyword*)
- Value** `null`
- Choices** `null`, `routing`, `distributed`
- Option** `-hydrology`
- Description** Basal hydrology sub-model.
149. `hydrology.null_diffuse_till_water` (*boolean*)
- Value** `no`
- Description** Diffuse stored till water laterally. See equation (11) of [17]
150. `hydrology.null_diffusion_distance` (*scalar*)
- Value** 20000 (meters)
- Description** diffusion distance for till water thickness; see equation (11) in [17]; only active if hydrology.null_diffuse_till_water is set
151. `hydrology.null_diffusion_time` (*scalar*)
- Value** 1000 (years)
- Description** diffusion time for till water thickness; see equation (11) in [17]; only active if hydrology.null_diffuse_till_water is set
152. `hydrology.null_strip_width` (*scalar*)
- Value** -1 (meters)
- Description** if negative then mechanism is inactive; width of strip around computational domain in which water velocity and water amount are set to zero; used by hydrology::Routing and hydrology::Distributed
153. `hydrology.regularizing_porosity` (*scalar*)
- Value** 0.010000 (pure number)
- Option** `-hydrology_regularizing_porosity`
- Description** ϕ_0 in notes; regularizes pressure equation by multiplying time derivative term
154. `hydrology.roughness_scale` (*scalar*)
- Value** 0.100000 (meters)
- Option** `-hydrology_roughness_scale`
- Description** W_r in notes; roughness scale determining maximum amount of cavitation opening in hydrology::Distributed
155. `hydrology.thickness_power_in_flux` (*scalar*)
- Value** 1.250000 (1)
- Option** `-hydrology_thickness_power_in_flux`
- Description** power α in Darcy's law $q = -kW^\alpha |\nabla\psi|^{\beta-2} \nabla\psi$, for subglacial water layer; used by hydrology::Routing and hydrology::Distributed
156. `hydrology.tillwat_decay_rate` (*scalar*)

- Value** 3.168876e-11 (meter / second)
Option -hydrology_tillwat_decay_rate
Description default value is equivalent to 1 mm per year; rate at which tillwat is reduced to zero, in absence of other effects like input
157. hydrology.tillwat_max (*scalar*)
Value 2 (meters)
Option -hydrology_tillwat_max
Description maximum effective thickness of the water stored in till
158. hydrology.use_const_bmelt (*boolean*)
Value no
Option -hydrology_use_const_bmelt
Description if 'yes', subglacial hydrology model sees basal melt rate which is constant and given by hydrology.const_bmelt
159. inverse.design.ch1 (*scalar*)
Value 0 (1)
Option -inv_design_ch1
Description weight of derivative part of an H1 norm for inversion design variables
160. inverse.design.cL2 (*scalar*)
Value 1 (1)
Option -inv_design_cL2
Description weight of derivative-free part of an H1 norm for inversion design variables
161. inverse.design.func (*keyword*)
Value sobolevH1
Choices sobolevH1, tv
Option -inv_design_func
Description functional used for inversion design variables
162. inverse.design.param (*keyword*)
Value exp
Choices ident, trunc, square, exp
Option -inv_design_param
Description parameterization of design variables used during inversion
163. inverse.design.param_hardav_eps (*scalar*)
Value 10000 (Pascal second^(1/3))
Description tiny vertically-averaged hardness used as a substitute for 0 in some tauc parameterizations
164. inverse.design.param_hardav_scale (*scalar*)
Value 1.000000e+08 (Pascal second^(1/3))

- Description** typical size of ice hardness
165. `inverse.design.param_tauc_eps` (*scalar*)
- Value** 100 (Pascal)
- Description** tiny yield stress used as a substitute for 0 in some `tauc` parameterizations
166. `inverse.design.param_tauc_scale` (*scalar*)
- Value** 100000 (Pascal)
- Description** typical size of yield stresses
167. `inverse.design.param_trunc_hardav0` (*scalar*)
- Value** 1000000 (Pascal second^(1/3))
- Description** transition point of change to linear behaviour for design variable parameterization type 'trunc'
168. `inverse.design.param_trunc_tauc0` (*scalar*)
- Value** 1000 (Pascal)
- Description** transition point of change to linear behaviour for design variable parameterization type 'trunc'
169. `inverse.log_ratio_scale` (*scalar*)
- Value** 10 (pure number)
- Option** `-inv_log_ratio_scale`
- Description** Reference scale for log-ratio functionals
170. `inverse.ssa.hardav_max` (*scalar*)
- Value** 1.000000e+10 (Pascal second^(1/3))
- Description** Maximum allowed value of `hardav` for inversions with bound constraints
171. `inverse.ssa.hardav_min` (*scalar*)
- Value** 0 (Pascal second^(1/3))
- Description** Minimum allowed value of `hardav` for inversions with bound constraints
172. `inverse.ssa.length_scale` (*scalar*)
- Value** 50000 (meters)
- Description** typical length scale for rescaling derivative norms
173. `inverse.ssa.method` (*keyword*)
- Value** `tikhonov_lmvm`
- Choices** `sd`, `nlcg`, `ign`, `tikhonov_lmvm`, `tikhonov_cg`, `tikhonov_blvm`, `tikhonov_lcl`, `tikhonov_gn`
- Option** `-inv_method`
- Description** algorithm to use for SSA inversions
174. `inverse.ssa.tauc_max` (*scalar*)
- Value** 5.000000e+07 (Pascal)
- Description** Maximum allowed value of `tauc` for inversions with bound constraints

175. `inverse.ssa.tauc_min` (*scalar*)
Value 0 (Pascal)
Description Minimum allowed value of `tauc` for inversions with bound constraints
176. `inverse.ssa.tv_exponent` (*scalar*)
Value 1.200000 (pure number)
Option `-inv_ssa_tv_exponent`
Description Lebesgue exponent for pseudo-TV norm
177. `inverse.ssa.velocity_eps` (*scalar*)
Value 0.100000 (meter / year)
Description tiny size of ice velocities during inversion
178. `inverse.ssa.velocity_scale` (*scalar*)
Value 100 (meter / year)
Description typical size of ice velocities expected during inversion
179. `inverse.state_func` (*keyword*)
Value `meansquare`
Choices `meansquare`, `log_ratio`, `log_relative`
Option `-inv_state_func`
Description functional used for inversion design variables
180. `inverse.target_misfit` (*scalar*)
Value 100 (meter / year)
Option `-inv_target_misfit`
Description desired root misfit for SSA inversions
181. `inverse.tikhonov.atol` (*scalar*)
Value 1.000000e-10 (meter / year)
Option `-tikhonov_atol`
Description absolute threshold for Tikhonov stopping criterion
182. `inverse.tikhonov.penalty_weight` (*scalar*)
Value 1 (1)
Option `-tikhonov_penalty`
Description penalty parameter for Tikhonov inversion
183. `inverse.tikhonov.ptol` (*scalar*)
Value 0.100000 (pure number)
Option `-tikhonov_ptol`
Description threshold for reaching desired misfit for adaptive Tikhonov algorithms
184. `inverse.tikhonov.rtol` (*scalar*)
Value 0.050000 (1)

- Option** `-tikhonov_rtol`
Description relative threshold for Tikhonov stopping criterion
185. `ocean.always_grounded` (*boolean*)
Value `no`
Option `-dry`
Description Dry (ocean-less) simulation; ice is considered grounded regardless of ice thickness, bed elevation, and sea level.
186. `ocean.pik_melt_factor` (*scalar*)
Value `0.005000` (1)
Option `-meltfactor_pik`
Description dimensionless tuning parameter in the ‘-ocean pik’ ocean heat flux parameterization; see [6]
187. `ocean.sub_shelf_heat_flux_into_ice` (*scalar*)
Value `0.500000` (W meter-2)
Description = J meter-2 second-1; naively chosen default value for heat from ocean; see comments in `pism::ocean::Constant::shelf_base_mass_flux()`.
188. `ocean.three_equation_model_clip_salinity` (*boolean*)
Value `yes`
Option `-clip_shelf_base_salinity`
Description Clip shelf base salinity so that it is in the range [4, 40] k/kg. See [8].
189. `output.backup_interval` (*scalar*)
Value `1` (hours)
Option `-backup_interval`
Description wall-clock time between automatic backups
190. `output.backup_size` (*keyword*)
Value `small`
Choices `none, small, medium, big_2d, big`
Option `-backup_size`
Description The ‘size’ of a backup file. See configuration parameters `output.sizes.medium`, `output.sizes.big_2d`, `output.sizes.big`
191. `output.fill_value` (*scalar*)
Value `-2.000000e+09` (none)
Description _FillValue used when saving diagnostic quantities
192. `output.format` (*keyword*)
Value `netcdf3`
Choices `netcdf3, netcdf4_parallel, pnetcdf`
Option `-o_format`

Description The I/O format used for spatial fields; 'netcdf3' is the default, 'netcd4_parallel' is available if PISM was built with parallel NetCDF-4, and 'pnetcdf' is available if PISM was built with PnetCDF.

193. `output.ice_free_thickness_standard` (*scalar*)

Value 10 (meters)

Description If ice is thinner than this standard then a grid cell is considered ice-free for purposes of reporting glacierized area, volume, etc.

194. `output.runtime.area_scale_factor_log10` (*integer*)

Value 6

Option `-summary_area_scale_factor_log10`

Description an integer; log base 10 of scale factor to use for area (in km²) in summary line to stdout

195. `output.file_name` (*string*)

Value unnamed.nc

Option `-o`

Description The file to save final model results to.

196. `output.runtime.time_unit_name` (*string*)

Value year

Description Time units used when printing model time, time step, and maximum horizontal velocity at summary to stdout. Must be valid udunits for time. (E.g. choose from year,month,day,hour,minute,second.)

197. `output.runtime.time_use_calendar` (*boolean*)

Value yes

Description Whether to use the current calendar when printing model time in summary to stdout.

198. `output.runtime.viewer.size` (*integer*)

Value 320

Option `-view_size`

Description default diagnostic viewer size (number of pixels of the longer side)

199. `output.runtime.viewer.variables` (*string*)

Value no default

Option `-view`

Description comma-separated list of map-plane diagnostic quantities to view at runtime

200. `output.runtime.volume_scale_factor_log10` (*integer*)

Value 6

Option `-summary_vol_scale_factor_log10`

Description an integer; log base 10 of scale factor to use for volume (in km³) in summary line to stdout

201. `output.save_size` (*keyword*)

Value small

Choices none, small, medium, big_2d, big

Option -save_size

Description The ‘size’ of a snapshot file. See configuration parameters output.sizes.medium, output.sizes.big_2d, output.sizes.big

202. output.size (*keyword*)

Value medium

Choices none, small, medium, big_2d, big

Option -o_size

Description The ‘size’ of an output file. See configuration parameters output.sizes.medium, output.sizes.big_2d, output.sizes.big

203. output.sizes.big (*string*)

Value cts, liqfrac, temp, temp_pa, uvel, vvel, wvel, wvel_rel

Description Comma-separated list of variables to write to the output (in addition to model_state variables and variables listed in output.sizes.medium and output.sizes.big_2d) if ‘big’ output size is selected. Does not include fields written by sub-models.

204. output.sizes.big_2d (*string*)

Value age, bfriact, bheatflx, bmelt, bwp, bwprel, cell_area, dbdt, effbwp, enthalpybase, enthalpysurf, flux_divergence, hardav, hydroinput, lat, litho_temp, lon, nuH, ocean_kill_mask, rank, tempbase, tempicethk, tempicethk_basal, temppabase, tempsurf, thk, thksmooth, tillphi, topg, velbar, velbase, wallmelt, wvelbase

Description Comma-separated list of variables to write to the output (in addition to model_state variables and variables listed in output.sizes.medium) if ‘big_2d’ output size is selected. Does not include fields written by boundary models.

205. output.sizes.medium (*string*)

Value bwat, bwatvel, climatic_mass_balance, diffusivity, enthalpy, flux, flux_mag, ice_surface_temp, liqfrac, mask, schoofs_theta, strain_rates, taub_mag, tauc, taud_mag, temp_pa, tillwat, topgsmooth, usurf, velbar_mag, velbase_mag, velsurf, velsurf_mag, wvelsurf

Description Comma-separated list of variables to write to the output (in addition to model_state variables) if ‘medium’ output size (the default) is selected. Does not include fields written by sub-models.

206. output.timeseries.buffer_size (*integer*)

Value 10000

Description Number of scalar diagnostic time-series records to hold in memory before writing to disk. (PISM writes this many time-series records to reduce I/O costs.) Send the USR2 signal to flush time-series.

207. output.timeseries.variables (*string*)

Value no default

Option -ts_vars

- Description** Requested scalar (time-series) diagnostics. Leave empty to save all available diagnostics.
208. `output.timeseries.append` (*boolean*)
- Value** `false`
- Option** `-ts_append`
- Description** If true, append to the scalar time series output file.
209. `output.timeseries.filename` (*string*)
- Value** *no default*
- Option** `-ts_file`
- Description** Name of the file to save scalar time series to. Leave empty to disable reporting scalar time-series.
210. `output.variable_order` (*keyword*)
- Value** `yxz`
- Choices** `xyz`, `yxz`, `zyx`
- Option** `-o_order`
- Description** Variable order to use in output files.
211. `regional.no_model_strip` (*scalar*)
- Value** `5` (km)
- Option** `-no_model_strip`
- Description** Default width of the ‘no model strip’ in regional setups.
212. `regional.no_model_yield_stress` (*scalar*)
- Value** `1000` (kPa)
- Description** High yield stress used in the ‘no_model_mask’ area in the regional mode.
213. `regional.zero_gradient` (*boolean*)
- Value** `false`
- Option** `-zero_grad_where_no_model`
- Description** Use zero ice thickness and ice surface gradient in the no_model_mask area.
214. `run_info.institution` (*string*)
- Value** *no default*
- Option** `-institution`
- Description** Institution name. This string is written to output files as the ‘institution’ global attribute.
215. `run_info.title` (*string*)
- Value** *no default*
- Option** `-title`
- Description** Free-form string containing a concise description of the current run. This string is written to output files as the ‘title’ global attribute.
216. `stress_balance.calving_front_stress_bc` (*boolean*)

- Value** no
- Option** -cfbc
- Description** Apply CFBC condition as in [73], [25]. May only apply to some stress balances; e.g. SSAFD as of May 2011. If not set then a strength-extension is used, as in [17].
217. `stress_balance.ice_free_thickness_standard` (*scalar*)
- Value** 10 (meters)
- Description** If ice is thinner than this standard then a cell is considered ice-free for purposes of computing ice velocity distribution.
218. `stress_balance.model` (*keyword*)
- Value** sia
- Choices** none, prescribed_sliding, sia, ssa, prescribed_sliding+sia, ssa+sia
- Option** -stress_balance
- Description** Stress balance model
219. `stress_balance.sia.Glen_exponent` (*scalar*)
- Value** 3 (pure number)
- Option** -sia_n
- Description** Glen exponent in ice flow law for SIA
220. `stress_balance.sia.bed_smoother_range` (*scalar*)
- Value** 5000 (meters)
- Option** -bed_smoother_range
- Description** half-width of smoothing domain for stressbalance::BedSmoother, in implementing [86] bed roughness parameterization for SIA; set value to zero to turn off mechanism
221. `stress_balance.sia.e_age_coupling` (*boolean*)
- Value** no
- Option** -e_age_coupling
- Description** Couple the SIA enhancement factor to age as in [24].
222. `stress_balance.sia.enhancement_factor` (*scalar*)
- Value** 1 (1)
- Option** -sia_e
- Description** Flow enhancement factor for SIA
223. `stress_balance.sia.enhancement_factor_interglacial` (*scalar*)
- Value** 1 (1)
- Option** -sia_e_interglacial
- Description** Flow enhancement factor for SIA; used for ice accumulated during interglacial periods.
224. `stress_balance.sia.flow_law` (*keyword*)
- Value** gpbld
- Choices** arr, arrwarm, gk, gpbld, hooke, isothermal_glen, pb, gpbld3

- Option** -sia_flow_law
Description The SIA flow law.
225. stress_balance.sia.grain_size_age_coupling (*boolean*)
Value no
Option -grain_size_age_coupling
Description Use age of the ice to compute grain size to use with the Goldsby-Kohlstedt [63] flow law
226. stress_balance.sia.surface_gradient_method (*keyword*)
Value haseloff
Choices eta, haseloff, mahaffy
Option -gradient
Description method used for surface gradient calculation at staggered grid points
227. stress_balance.ssa.Glen_exponent (*scalar*)
Value 3 (pure number)
Option -ssa_n
Description Glen exponent in ice flow law for SSA
228. stress_balance.ssa.compute_surface_gradient_inward (*boolean*)
Value no
Description If yes then use inward first-order differencing in computing surface gradient in the SSA objects.
229. stress_balance.ssa.dirichlet_bc (*boolean*)
Value no
Option -ssa_dirichlet_bc
Description apply SSA velocity Dirichlet boundary condition
230. stress_balance.ssa.enhancement_factor (*scalar*)
Value 1 (1)
Option -ssa_e
Description Flow enhancement factor for SSA
231. stress_balance.ssa.enhancement_factor_interglacial (*scalar*)
Value 1 (1)
Option -ssa_e_interglacial
Description Flow enhancement factor for SSA; used for ice accumulated during interglacial periods.
232. stress_balance.ssa.epsilon (*scalar*)
Value 1.000000e+13 (Pascal second meter)
Option -ssa_eps

Description Initial amount of regularization in computation of product of effective viscosity and thickness (νH). This default value for νH comes e.g. from a hardness for the Ross ice shelf (\bar{B}) = 1.9e8 Pa s^{1/3} [57] and a typical strain rate of 0.001 1/year for the Ross ice shelf, giving $\nu = (\bar{B})/(2 \cdot 0.001^{2/3}) = 9.49\text{e}+14$ Pa s ~ 30 MPa year, the value in [92], but with a tiny thickness H of about 1 cm.

233. `stress_balance.ssa.fd.brutal_sliding` (*boolean*)

Value false

Option -brutal_sliding

Description Enhance sliding speed brutally.

234. `stress_balance.ssa.fd.brutal_sliding_scale` (*scalar*)

Value 1 (1)

Option -brutal_sliding_scale

Description Brutal SSA Sliding Scale

235. `stress_balance.ssa.fd.lateral_drag.enabled` (*boolean*)

Value false

Description set viscosity at ice shelf margin next to ice free bedrock as friction parameterization

236. `stress_balance.ssa.fd.lateral_drag.viscosity` (*scalar*)

Value 5.000000e+15 (Pascal second)

Option -nu_bedrock

Description Staggered Viscosity used as side friction parameterization.

237. `stress_balance.ssa.fd.max_iterations` (*integer*)

Value 300

Option -ssa_maxi

Description Maximum number of iterations for the ice viscosity computation, in the SSAFD object

238. `stress_balance.ssa.fd.nuH_iter_failure_underrelaxation` (*scalar*)

Value 0.800000 (pure number)

Option -ssaafd_nuH_iter_failure_underrelaxation

Description In event of 'Effective viscosity not converged' failure, use outer iteration rule $\text{nuH} <- \text{nuH} + f(\text{nuH} - \text{nuH_old})$, where f is this parameter.

239. `stress_balance.ssa.fd.relative_convergence` (*scalar*)

Value 0.000100 (1)

Option -ssa_rtol

Description Relative change tolerance for the effective viscosity in the SSAFD object

240. `stress_balance.ssa.fd.replace_zero_diagonal_entries` (*boolean*)

Value yes

Description Replace zero diagonal entries in the SSAFD matrix with `basal_resistance.beta_ice_free_bedrock` to avoid solver failures.

241. `stress_balance.ssa.flow_law` (*keyword*)

- Value** gpbld
- Choices** arr, arrwarm, gpbld, hooke, isothermal_glen, pb, gpbld3
- Option** -ssa_flow_law
- Description** The SSA flow law.
242. `stress_balance.ssa.method` (*keyword*)
- Value** fd
- Choices** fd, fem
- Option** -ssa_method
- Description** Algorithm for computing the SSA solution.
243. `stress_balance.ssa.strength_extension.constant_nu` (*scalar*)
- Value** 9.486807e+14 (Pascal second)
- Description** The SSA is made elliptic by use of a constant value for the product of viscosity (ν) and thickness (H). This value for ν comes from hardness (\bar{B})= $1.9e8 \text{ Pa s}^{1/3}$ [57] and a typical strain rate of 0.001 year⁻¹: $\nu = (\bar{B})/(2 \cdot 0.001^{2/3})$. Compare the value of $9.45e14 \text{ Pa s} = 30 \text{ MPa year}$ in [92].
244. `stress_balance.ssa.strength_extension.min_thickness` (*scalar*)
- Value** 50 (meters)
- Description** The SSA is made elliptic by use of a constant value for the product of viscosity (ν) and thickness (H). At ice thicknesses below this value the product $\nu \cdot H$ switches from the normal vertical integral to a constant value. The geometry itself is not affected by this value.
245. `stress_balance.vertical_velocity_approximation` (*keyword*)
- Value** centered
- Choices** centered, upstream
- Option** -vertical_velocity_approximation
- Description** Vertical velocity FD approximation. “Upstream” uses first-order finite difference to compute u_x and v_y . Uses basal velocity to make decisions.
246. `surface.force_to_thickness.alpha` (*scalar*)
- Value** 0.010000 (year⁻¹)
- Description** exponential coefficient in force-to-thickness mechanism
247. `surface.force_to_thickness.ice_free_alpha_factor` (*scalar*)
- Value** 1 (1)
- Description** `surface.force_to_thickness.alpha` is multiplied by this factor in areas that are ice-free according to the target ice thickness and `surface.force_to_thickness.ice_free_thickness_threshold`
248. `surface.force_to_thickness.ice_free_thickness_threshold` (*scalar*)
- Value** 1 (meters)
- Description** threshold of ice thickness in the force-to-thickness target field. Used to determine whether to use `surface.force_to_thickness.ice_free_alpha_factor`.
249. `surface.force_to_thickness.start_time` (*scalar*)
- Value** -4.540000e+09 (years)

- Description** Starting time for the “force to thickness” modifier; the default is “start from the creation of the Earth.”
250. `surface.given.smb_max` (*scalar*)
- Value** 91000 (kg m⁻² year⁻¹)
- Description** Maximum climatic mass balance value (used to check input data). Corresponds to 100 m/year ice equivalent.
251. `surface.pdd.air_temp_all_precip_as_rain` (*scalar*)
- Value** 275.150000 (Kelvin)
- Description** threshold temperature above which all precipitation is rain; must exceed `surface.pdd.air_temp_all_precip_as_snow` to avoid division by zero, because difference is in a denominator
252. `surface.pdd.air_temp_all_precip_as_snow` (*scalar*)
- Value** 273.150000 (Kelvin)
- Description** threshold temperature below which all precipitation is snow
253. `surface.pdd.balance_year_start_day` (*integer*)
- Value** 274
- Description** day of year for October 1st, beginning of the balance year in northern latitudes.
254. `surface.pdd.factor_ice` (*scalar*)
- Value** 0.008791 (meter / (Kelvin day))
- Description** EISMINT-Greenland value [3]; = (8 mm liquid-water-equivalent) / (pos degree day)
255. `surface.pdd.factor_snow` (*scalar*)
- Value** 0.003297 (meter / (Kelvin day))
- Description** EISMINT-Greenland value [3]; = (3 mm liquid-water-equivalent) / (pos degree day)
256. `surface.pdd.fausto.T_c` (*scalar*)
- Value** 272.150000 (Kelvin)
- Description** = -1 + 273.15; for formula (6) in [1]
257. `surface.pdd.fausto.T_w` (*scalar*)
- Value** 283.150000 (Kelvin)
- Description** = 10 + 273.15; for formula (6) in [1]
258. `surface.pdd.fausto.beta_ice_c` (*scalar*)
- Value** 0.015000 (meter / (Kelvin day))
- Description** water-equivalent thickness; for formula (6) in [1]
259. `surface.pdd.fausto.beta_ice_w` (*scalar*)
- Value** 0.007000 (meter / (Kelvin day))
- Description** water-equivalent thickness; for formula (6) in [1]
260. `surface.pdd.fausto.beta_snow_c` (*scalar*)
- Value** 0.003000 (meter / (Kelvin day))

- Description** water-equivalent thickness; for formula (6) in [1]
261. `surface.pdd.fausto.beta_snow_w` (*scalar*)
- Value** 0.003000 (meter / (Kelvin day))
- Description** water-equivalent thickness; for formula (6) in [1]
262. `surface.pdd.fausto.latitude_beta_w` (*scalar*)
- Value** 72 (degree_north)
- Description** latitude below which to use warm case, in formula (6) in [1]
263. `surface.pdd.firn_compaction_to_accumulation_ratio` (*scalar*)
- Value** 0.750000 (1)
- Description** How much firn as a fraction of accumulation is turned into ice
264. `surface.pdd.firn_depth_file` (*string*)
- Value** *no default*
- Option** `-pdd_firn_depth_file`
- Description** The name of the file to read the firn_depth from.
265. `surface.pdd.temperature_standard_deviation_file` (*string*)
- Value** *no default*
- Option** `-pdd_sd_file`
- Description** The name of the file to read air_temp_sd from.
266. `surface.pdd.interpret_precip_as_snow` (*boolean*)
- Value** no
- Description** Interpret precipitation as snow fall.
267. `surface.pdd.max_evals_per_year` (*integer*)
- Value** 52
- Description** maximum number of times the PDD scheme will ask for air temperature and precipitation to build location-dependent time series for computing (expected) number of positive degree days and snow accumulation; the default means the PDD uses weekly samples of the annual cycle; see also `surface.pdd.std_dev`
268. `surface.pdd.positive_threshold_temp` (*scalar*)
- Value** 273.150000 (Kelvin)
- Description** temperature used to determine meaning of ‘positive’ degree day
269. `surface.pdd.refreeze` (*scalar*)
- Value** 0.600000 (1)
- Description** EISMINT-Greenland value [3]
270. `surface.pdd.refreeze_ice_melt` (*boolean*)
- Value** yes
- Description** If set to ‘yes’, refreeze `surface.pdd.refreeze` fraction of melted ice, otherwise all of the melted ice runs off.

271. `surface.pdd.std_dev` (*scalar*)
Value 5 (Kelvin)
Description std dev of daily temp variation; = EISMINT-Greenland value [3]
272. `surface.pdd.std_dev_lapse_lat_base` (*scalar*)
Value 72 (degree_north)
Description std_dev is a function of latitude, with value `surface.pdd.std_dev` at this latitude; this value only active if `surface.pdd.std_dev_lapse_lat_rate` is nonzero
273. `surface.pdd.std_dev_lapse_lat_rate` (*scalar*)
Value 0 (Kelvin / degree_north)
Description std_dev is a function of latitude, with rate of change with respect to latitude given by this constant
274. `surface.pdd.std_dev_param_a` (*scalar*)
Value -0.150000 (pure number)
Description Parameter a in $\text{Sigma} = a \cdot T + b$, with T in degrees C. Used only if `surface.pdd.std_dev_use_param` is set to yes.
275. `surface.pdd.std_dev_param_b` (*scalar*)
Value 0.660000 (Kelvin)
Description Parameter b in $\text{Sigma} = a \cdot T + b$, with T in degrees C. Used only if `surface.pdd.std_dev_use_param` is set to yes.
276. `surface.pdd.std_dev_use_param` (*boolean*)
Value no
Description Parameterize standard deviation as a linear function of air temperature over ice-covered grid cells. The region of application is controlled by `geometry.ice_free_thickness_standard`.
277. `surface.pressure` (*scalar*)
Value 0 (Pascal)
Description atmospheric pressure; = pressure at ice surface
278. `time.calendar` (*keyword*)
Value 365_day
Choices standard, gregorian, proleptic_gregorian, noleap, 365_day, 360_day, julian, none
Option -calendar
Description The calendar to use.
279. `time.dimension_name` (*string*)
Value time
Description The name of the time dimension in PISM output files.
280. `time.eemian_end` (*scalar*)
Value -114500 (years)
Description End of the Eemian interglacial period. See [60].

281. `time.eemian_start` (*scalar*)
Value -132000 (years)
Description Start of the Eemian interglacial period. See [60].
282. `time.holocene_start` (*scalar*)
Value -11000 (years)
Description Start of the Holocene interglacial period. See [60].
283. `time.reference_date` (*string*)
Value 1-1-1
Description year-month-day; reference date used for calendar computations and in PISM output files
284. `time.run_length` (*scalar*)
Value 1000 (years)
Description Default run length
285. `time.start_year` (*scalar*)
Value 0 (years)
Description Start year.
286. `time_stepping.adaptive_ratio` (*scalar*)
Value 0.120000 (1)
Option `-adapt_ratio`
Description Adaptive time stepping ratio for the explicit scheme for the mass balance equation; [18], inequality (25)
287. `time_stepping.count_steps` (*boolean*)
Value no
Option `-count_steps`
Description If yes, `IceModel::run()` will count the number of time steps it took. Sometimes useful for performance evaluation. Counts all steps, regardless of whether processes (mass continuity, energy, velocity, ...) occurred within the step.
288. `time_stepping.hit_extra_times` (*boolean*)
Value yes
Option `-extra_force_output_times`
Description Modify the time-stepping mechanism to hit times requested using `-extra_times`.
289. `time_stepping.hit_multiples` (*scalar*)
Value 0 (years)
Option `-timestep_hit_multiples`
Description Hit every X years, where X is specified using this parameter. Use 0 to disable
290. `time_stepping.hit_save_times` (*boolean*)
Value no

Option `-save_force_output_times`

Description Modify the time-stepping mechanism to hit times requested using `-save_times`.

291. `time_stepping.hit_ts_times` (*boolean*)

Value `no`

Description Modify the time-stepping mechanism to hit times requested using `-ts_times`.

292. `time_stepping.maximum_time_step` (*scalar*)

Value `60` (years)

Option `-max_dt`

Description Maximum allowed time step length

293. `time_stepping.skip.enabled` (*boolean*)

Value `no`

Option `-skip`

Description Use the temperature, age, and SSA stress balance computation skipping mechanism.

294. `time_stepping.skip.max` (*integer*)

Value `10`

Option `-skip_max`

Description Number of mass-balance steps, including SIA diffusivity updates, to perform before a the temperature, age, and SSA stress balance computations are done

3.11 Diagnostic quantities

The availability of a diagnostic depends on modeling choices. For example, the bed uplift rate `dbdt` is available only if a bed deformation model is selected.

Some scalar diagnostics come in two versions: the one with the suffix `_glacierized` and the one without. Here the former account for the ice `output.ice_free_thickness_standard` meters or thicker (10 meters by default) and the latter include all ice regardless of the thickness. “Glacierized” versions were added to make it easier to analyze changes in glacier volumes and areas and exclude changes in the seasonal snow cover.

Contents

- *Diagnostic quantities*
 - *Spatially-variable fields*
 - *Scalar time-series*

3.11.1 Spatially-variable fields

1. `air_temp_mean_july`

Units Kelvin

Description mean July near-surface air temperature used in the cosine yearly cycle

2. `air_temp_sd`
Units Kelvin
Description standard deviation of near-surface air temperature
3. `air_temp_snapshot`
Units Kelvin
Description instantaneous value of the near-surface air temperature
4. `basal_mass_flux_floating`
Units kg m⁻² year⁻¹
Description average basal mass flux over the reporting interval (floating areas)
Comment positive flux corresponds to ice gain
5. `basal_mass_flux_grounded`
Units kg m⁻² year⁻¹
Description average basal mass flux over the reporting interval (grounded areas)
Comment positive flux corresponds to ice gain
6. `basal_melt_rate_grounded`
Units m year⁻¹
Description ice basal melt rate from energy conservation, in ice thickness per time (valid in grounded areas)
Comment positive basal melt rate corresponds to ice loss
7. `bedtoptemp`
Units Kelvin
Description temperature at the top surface of the bedrock thermal layer
8. `beta`
Units Pa s / m
Description basal drag coefficient
9. `bfrict`
Units W m⁻²
Description basal frictional heating
10. `bheatflx`
Units mW m⁻²
Description upward geothermal flux at the bottom bedrock surface
Comment positive values correspond to an upward flux
11. `bmelt`
Units m year⁻¹
Description ice basal melt rate from energy conservation and subshelf melt, in ice thickness per time
Standard name `land_ice_basal_melt_rate`

- Comment** positive basal melt rate corresponds to ice loss
12. `bwat`
- Units** m
- Description** thickness of transportable water in subglacial layer
13. `bwatvel`
- `bwatvel[0]`
 - Units** m year⁻¹
 - Description** velocity of water in subglacial layer, i-offset
 - `bwatvel[1]`
 - Units** m year⁻¹
 - Description** velocity of water in subglacial layer, j-offset
14. `bwpp`
- Units** Pa
- Description** pressure of transportable water in subglacial layer
15. `bwpprel`
- Units** —
- Description** pressure of transportable water in subglacial layer as fraction of the overburden pressure
16. `calving_threshold`
- Units** m
- Description** threshold used by the ‘calving at threshold’ calving method
17. `cell_area`
- Units** km²
- Description** cell areas
- Comment** values are equal to $dx \cdot dy$ if projection parameters are not available; otherwise WGS84 ellipsoid is used
18. `cell_grounded_fraction`
- Units** —
- Description** fractional grounded/floating mask (floating=0, grounded=1)
19. `climatic_mass_balance`
- Units** kg m⁻² year⁻¹
- Description** surface mass balance (accumulation/ablation) rate
- Standard name** `land_ice_surface_specific_mass_balance_flux`
20. `cts`
- Units** —
- Description** $cts = E/E_s(p)$, so cold-temperate transition surface is at $cts = 1$
21. `dHdt`

Units m year-1

Description ice thickness rate of change

Standard name tendency_of_land_ice_thickness

22. dbdt

Units mm year-1

Description bedrock uplift rate

Standard name tendency_of_bedrock_altitude

23. deviatoric_stresses

- sigma_xx

Units Pa

Description deviatoric stress in X direction

- sigma_yy

Units Pa

Description deviatoric stress in Y direction

- sigma_xy

Units Pa

Description deviatoric shear stress

24. diffusivity

Units m² s⁻¹

Description diffusivity of SIA mass continuity equation

25. diffusivity_staggered

- diffusivity_i

Units m² s⁻¹

Description diffusivity of SIA mass continuity equation on the staggered grid (i-offset)

- diffusivity_j

Units m² s⁻¹

Description diffusivity of SIA mass continuity equation on the staggered grid (j-offset)

26. effbwp

Units Pa

Description effective pressure of transportable water in subglacial layer (overburden pressure minus water pressure)

27. effective_air_temp

Units Kelvin

Description effective mean-annual near-surface air temperature

28. effective_precipitation

Units kg m⁻² year⁻¹

- Description** effective precipitation rate
29. `effective_viscosity`
- Units** kPascal second
- Description** effective viscosity of ice
30. `eigen_calving_rate`
- Units** m year⁻¹
- Description** horizontal calving rate due to eigen-calving
31. `enthalpy`
- Units** J kg⁻¹
- Description** ice enthalpy (includes sensible heat, latent heat, pressure)
32. `enthalpybase`
- Units** J kg⁻¹
- Description** ice enthalpy at the base of ice
33. `enthalpysurf`
- Units** J kg⁻¹
- Description** ice enthalpy at 1m below the ice surface
34. `firn_depth`
- Units** m
- Description** firn cover depth
35. `flux`
- `uflux`
 - Units** m² year⁻¹
 - Description** Vertically integrated horizontal flux of ice in the X direction
 - `vflux`
 - Units** m² year⁻¹
 - Description** Vertically integrated horizontal flux of ice in the Y direction
36. `flux_divergence`
- Units** m year⁻¹
- Description** flux divergence
37. `flux_mag`
- Units** m² year⁻¹
- Description** magnitude of vertically-integrated horizontal flux of ice
38. `flux_staggered`
- Units** m² year⁻¹
- Description** fluxes through cell interfaces (sides) on the staggered grid
39. `frontal_melt_rate`

- Units** m year-1
Description horizontal front retreat rate due to melt
40. `h_x`
- `h_x_i`
Units —
Description the x-component of the surface gradient, i-offset
 - `h_x_j`
Units —
Description the x-component of the surface gradient, j-offset
41. `h_y`
- `h_y_i`
Units —
Description the y-component of the surface gradient, i-offset
 - `h_y_j`
Units —
Description the y-component of the surface gradient, j-offset
42. `hardav`
Units Pa s0.333333
Description vertical average of ice hardness
43. `hardness`
Units Pa s0.333333
Description ice hardness computed using the SIA flow law
44. `height_above flotation`
Units m
Description ice thickness in excess of the maximum floating ice thickness
Comment shows how close to flotation the ice is at a given location
45. `hfgeoubed`
Units mW m-2
Description upward geothermal flux at the top bedrock surface
Standard name upward_geothermal_heat_flux_at_ground_level
Comment positive values correspond to an upward flux
46. `hydrobmelt`
Units m year-1
Description the version of bmelt seen by the hydrology model
47. `hydroinput`
Units m year-1

- Description** total water input into subglacial hydrology layer
48. ice_area_specific_volume
- Units** m³/m²
- Description** ice-volume-per-area in partially-filled grid cells
- Comment** this variable represents the amount of ice in a partially-filled cell and not the corresponding geometry, so thinking about it as ‘thickness’ is not helpful
49. ice_mass
- Units** kg
- Description** mass per cell
50. ice_surface_liquid_water_fraction
- Units** 1
- Description** ice liquid water fraction at the ice surface
51. ice_surface_temp
- Units** Kelvin
- Description** ice temperature at the ice surface
52. lat
- Units** degree_north
- Description** latitude
- Standard name** latitude
53. liqfrac
- Units** 1
- Description** liquid water fraction in ice (between 0 and 1)
54. lon
- Units** degree_east
- Description** longitude
- Standard name** longitude
55. mask
- Units** —
- Description** ice-type (ice-free/grounded/floating/ocean) integer mask
56. melange_back_pressure_fraction
- Units** 1
- Description** dimensionless pressure fraction at calving fronts due to presence of melange
57. nuH
- nuH[0]
- Units** kPa s m
- Description** ice thickness times effective viscosity, i-offset

- nuH[1]
Units kPa s m
Description ice thickness times effective viscosity, j-offset
- 58. ocean_kill_mask
Units —
Description mask specifying fixed calving front locations
- 59. ocean_pressure_difference
Units —
Description ocean pressure difference at calving fronts
- 60. pressure
Units Pa
Description pressure in ice (hydrostatic)
- 61. rank
Units 1
Description processor rank
- 62. saccum
Units kg m-2 year-1
Description accumulation (precipitation minus rain), averaged over the reporting interval
- 63. schoofs_theta
Units 1
Description multiplier ‘theta’ in Schoof’s (2003) theory of bed roughness in SIA
- 64. sea_level
Units meters
Description sea level elevation, relative to the geoid
- 65. sftflf
Units 1
Description fraction of a grid cell covered by floating ice
Standard name floating_ice_sheet_area_fraction
- 66. sftgif
Units 1
Description fraction of a grid cell covered by ice (grounded or floating)
Standard name land_ice_area_fraction
- 67. sftgrf
Units 1
Description fraction of a grid cell covered by grounded ice
Standard name grounded_ice_sheet_area_fraction

68. shelfbmassflux

Units $\text{kg m}^{-2} \text{s}^{-1}$

Description mass flux at the basal surface of ice shelves

69. shelfbtemp

Units Kelvin

Description ice temperature at the basal surface of ice shelves

70. smelt

Units $\text{kg m}^{-2} \text{year}^{-1}$

Description surface melt, averaged over the reporting interval

71. snow_depth

Units m

Description snow cover depth (set to zero once a year)

72. srunoff

Units $\text{kg m}^{-2} \text{year}^{-1}$

Description surface runoff, averaged over the reporting interval

73. ssa_bc_mask

Units —

Description Dirichlet boundary mask

74. ssa_bc_vel

- u_ssa_bc

Units m year^{-1}

Description X-component of the SSA velocity boundary conditions

- v_ssa_bc

Units m year^{-1}

Description Y-component of the SSA velocity boundary conditions

75. strain_rates

- eigen1

Units s^{-1}

Description first eigenvalue of the horizontal, vertically-integrated strain rate tensor

- eigen2

Units s^{-1}

Description second eigenvalue of the horizontal, vertically-integrated strain rate tensor

76. strainheat

Units mW m^{-3}

Description rate of strain heating in ice (dissipation heating)

77. surface_layer_mass

Units kg

Description mass of the surface layer (snow and firn)

78. `surface_layer_thickness`

Units meters

Description thickness of the surface layer (snow and firn)

79. `taub`

- `taub_x`

Units Pa

Description X-component of the shear stress at the base of ice

Comment this field is purely diagnostic (not used by the model)

- `taub_y`

Units Pa

Description Y-component of the shear stress at the base of ice

Comment this field is purely diagnostic (not used by the model)

80. `taub_mag`

Units Pa

Description magnitude of the basal shear stress at the base of ice

Standard name `magnitude_of_land_ice_basal_drag`

Comment this field is purely diagnostic (not used by the model)

81. `tauc`

Units Pa

Description yield stress for basal till (plastic or pseudo-plastic model)

82. `taud`

- `taud_x`

Units Pa

Description X-component of the driving shear stress at the base of ice

Comment this is the driving stress used by the SSA solver

- `taud_y`

Units Pa

Description Y-component of the driving shear stress at the base of ice

Comment this is the driving stress used by the SSA solver

83. `taud_mag`

Units Pa

Description magnitude of the driving shear stress at the base of ice

Comment this is the magnitude of the driving stress used by the SSA solver

84. `tauxz`

- Units** Pa
Description shear stress xz component (in shallow ice approximation SIA)
85. tauyz
Units Pa
Description shear stress yz component (in shallow ice approximation SIA)
86. temp
Units K
Description ice temperature
Standard name land_ice_temperature
87. temp_pa
Units deg_C
Description pressure-adjusted ice temperature (degrees above pressure-melting point)
88. tempbase
Units K
Description ice temperature at the base of ice
Standard name land_ice_basal_temperature
89. tempicethk
Units m
Description temperate ice thickness (total column content)
90. tempicethk_basal
Units m
Description thickness of the basal layer of temperate ice
91. tempabase
Units Celsius
Description pressure-adjusted ice temperature at the base of ice
92. tempsurf
Units K
Description ice temperature at 1m below the ice surface
Standard name temperature_at_ground_level_in_snow_or_firn
93. tendency_of_ice_amount
Units kg m⁻² year⁻¹
Description rate of change of the ice amount
94. tendency_of_ice_amount_due_to_basal_mass_flux
Units kg m⁻² year⁻¹
Description average basal mass flux over reporting interval
Comment positive flux corresponds to ice gain

95. tendency_of_ice_amount_due_to_conservation_error

Units kg m-2 year-1

Description average mass conservation error flux over reporting interval

Comment positive flux corresponds to ice gain

96. tendency_of_ice_amount_due_to_discharge

Units kg m-2 year-1

Description discharge (calving and frontal melt) flux

Standard name land_ice_specific_mass_flux_due_to_calving_and_ice_front_melting

Comment positive flux corresponds to ice gain

97. tendency_of_ice_amount_due_to_flow

Units kg m-2 year-1

Description rate of change of ice amount due to flow

Comment positive flux corresponds to ice gain

98. tendency_of_ice_amount_due_to_surface_mass_flux

Units kg m-2 year-1

Description average surface mass flux over reporting interval

Comment positive flux corresponds to ice gain

99. tendency_of_ice_mass

Units Gt year-1

Description rate of change of the ice mass

100. tendency_of_ice_mass_due_to_basal_mass_flux

Units Gt year-1

Description average basal mass flux over reporting interval

Standard name tendency_of_land_ice_mass_due_to_basal_mass_balance

Comment positive flux corresponds to ice gain

101. tendency_of_ice_mass_due_to_conservation_error

Units Gt year-1

Description average mass conservation error flux over reporting interval

Comment positive flux corresponds to ice gain

102. tendency_of_ice_mass_due_to_discharge

Units Gt year-1

Description discharge (calving and frontal melt) flux

Comment positive flux corresponds to ice gain

103. tendency_of_ice_mass_due_to_flow

Units Gt year-1

Description rate of change of ice mass due to flow

- Comment** positive flux corresponds to ice gain
104. tendency_of_ice_mass_due_to_surface_mass_flux
- Units** Gt year-1
- Description** average surface mass flux over reporting interval
- Comment** positive flux corresponds to ice gain
105. thk
- Units** m
- Description** land ice thickness
- Standard name** land_ice_thickness
106. thksmooth
- Units** m
- Description** thickness relative to smoothed bed elevation in Schoof's (2003) theory of bed roughness in SIA
107. tillphi
- Units** degrees
- Description** friction angle for till under grounded ice sheet
108. tillwat
- Units** m
- Description** effective thickness of subglacial water stored in till
109. topg
- Units** m
- Description** bedrock surface elevation
- Standard name** bedrock_altitude
110. topg_sl_adjusted
- Units** meters
- Description** sea-level adjusted bed topography (zero is at sea level)
111. topgsmooth
- Units** m
- Description** smoothed bed elevation in Schoof's (2003) theory of bed roughness in SIA
112. usurf
- Units** m
- Description** ice upper surface elevation
- Standard name** surface_altitude
113. uvel
- Units** m year-1
- Description** horizontal velocity of ice in the X direction

Standard name land_ice_x_velocity

114. velbar

- ubar

Units m year-1

Description vertical mean of horizontal ice velocity in the X direction

Standard name land_ice_vertical_mean_x_velocity

- vbar

Units m year-1

Description vertical mean of horizontal ice velocity in the Y direction

Standard name land_ice_vertical_mean_y_velocity

115. velbar_mag

Units m year-1

Description magnitude of vertically-integrated horizontal velocity of ice

116. velbase

- uvelbase

Units m year-1

Description x-component of the horizontal velocity of ice at the base of ice

Standard name land_ice_basal_x_velocity

- vvelbase

Units m year-1

Description y-component of the horizontal velocity of ice at the base of ice

Standard name land_ice_basal_y_velocity

117. velbase_mag

Units m year-1

Description magnitude of horizontal velocity of ice at base of ice

118. velsurf

- uvelsurf

Units m year-1

Description x-component of the horizontal velocity of ice at ice surface

Standard name land_ice_surface_x_velocity

- vvelsurf

Units m year-1

Description y-component of the horizontal velocity of ice at ice surface

Standard name land_ice_surface_y_velocity

119. velsurf_mag

Units m year-1

- Description** magnitude of horizontal velocity of ice at ice surface
120. `vonmises_calving_rate`
- Units** m year-1
- Description** horizontal calving rate due to von Mises calving
121. `vonmises_stress`
- Units** Pascal
- Description** tensile von Mises stress
122. `vvel`
- Units** m year-1
- Description** horizontal velocity of ice in the Y direction
- Standard name** `land_ice_y_velocity`
123. `wallmelt`
- Units** m year-1
- Description** wall melt into subglacial hydrology layer from (turbulent) dissipation of energy in transportable water
124. `wvel`
- Units** m year-1
- Description** vertical velocity of ice, relative to geoid
125. `wvel_rel`
- Units** m year-1
- Description** vertical velocity of ice, relative to base of ice directly below
126. `wvelbase`
- Units** m year-1
- Description** vertical velocity of ice at the base of ice, relative to the geoid
- Standard name** `land_ice_basal_upward_velocity`
127. `wvelsurf`
- Units** m year-1
- Description** vertical velocity of ice at ice surface, relative to the geoid
- Standard name** `land_ice_surface_upward_velocity`

3.11.2 Scalar time-series

1. `basal_mass_flux_floating`

Units kg year-1

Description total sub-shelf ice flux

Comment positive means ice gain
2. `basal_mass_flux_grounded`

- Units** kg year-1
Description total over grounded ice domain of basal mass flux
Comment positive means ice gain
3. dt
Units year
Description mass continuity time step
4. hydro_ice_free_land_loss
Units kg s-1
Description rate of liquid water loss from subglacial hydrology into cells with mask as ice free land
5. hydro_negative_thickness_gain
Units kg s-1
Description rate of non-conserving liquid water gain from subglacial hydrology transportable water thickness coming out negative during time step, and being projected up to zero
6. hydro_null_strip_loss
Units kg s-1
Description rate of liquid water loss from subglacial hydrology into cells inside the null strip
7. hydro_ocean_loss
Units kg s-1
Description rate of liquid water loss from subglacial hydrology into cells with mask as ocean
8. ice_area_glacierized
Units m2
Description glacierized area
9. ice_area_glacierized_cold_base
Units m2
Description glacierized area where basal ice is cold
10. ice_area_glacierized_floating
Units m2
Description area of ice shelves in glacierized areas
11. ice_area_glacierized_grounded
Units m2
Description area of grounded ice in glacierized areas
12. ice_area_glacierized_temperate_base
Units m2
Description glacierized area where basal ice is temperate
13. ice_enthalpy
Units J

- Description** enthalpy of the ice, including seasonal cover
14. ice_enthalpy_glacierized
- Units** J
- Description** enthalpy of the ice in glacierized areas
15. ice_mass
- Units** kg
- Description** mass of the ice, including seasonal cover
16. ice_mass_glacierized
- Units** kg
- Description** mass of the ice in glacierized areas
17. ice_volume
- Units** m³
- Description** volume of the ice, including seasonal cover
18. ice_volume_cold
- Units** m³
- Description** volume of cold ice, including seasonal cover
19. ice_volume_glacierized
- Units** m³
- Description** volume of the ice in glacierized areas
20. ice_volume_glacierized_cold
- Units** m³
- Description** volume of cold ice in glacierized areas
21. ice_volume_glacierized_floating
- Units** m³
- Description** volume of ice shelves in glacierized areas
22. ice_volume_glacierized_grounded
- Units** m³
- Description** volume of grounded ice in glacierized areas
23. ice_volume_glacierized_temperate
- Units** m³
- Description** volume of temperate ice in glacierized areas
24. ice_volume_temperate
- Units** m³
- Description** volume of temperate ice, including seasonal cover
25. limnsw
- Units** kg

- Description** mass of the ice not displacing sea water
26. `liquified_ice_flux`
- Units** m³ / year
- Description** rate of ice loss due to liquefaction, averaged over the reporting interval
- Comment** positive means ice loss
27. `max_diffusivity`
- Units** m² s⁻¹
- Description** maximum diffusivity
28. `max_hor_vel`
- Units** m year⁻¹
- Description** maximum abs component of horizontal ice velocity over grid in last time step during time-series reporting interval
29. `slvol`
- Units** m
- Description** total sea-level relevant ice IN SEA-LEVEL EQUIVALENT
30. `tendency_of_ice_mass`
- Units** kg year⁻¹
- Description** rate of change of the mass of ice, including seasonal cover
31. `tendency_of_ice_mass_due_to_basal_mass_balance`
- Units** kg year⁻¹
- Description** total over ice domain of bottom surface ice mass flux
- Comment** positive means ice gain
32. `tendency_of_ice_mass_due_to_conservation_error`
- Units** kg year⁻¹
- Description** total numerical flux needed to preserve non-negativity of ice thickness
- Comment** positive means ice gain
33. `tendency_of_ice_mass_due_to_discharge`
- Units** kg year⁻¹
- Description** discharge (calving & icebergs) flux
- Comment** positive means ice gain
34. `tendency_of_ice_mass_due_to_influx`
- Units** kg year⁻¹
- Description** rate of change of the mass of ice due to influx (i.e. prescribed ice thickness)
35. `tendency_of_ice_mass_due_to_surface_mass_balance`
- Units** kg year⁻¹
- Description** total over ice domain of top surface ice mass flux

Comment positive means ice gain

36. tendency_of_ice_mass_glacierized

Units kg year-1

Description rate of change of the ice mass in glacierized areas

37. tendency_of_ice_volume

Units m3 year-1

Description rate of change of the ice volume, including seasonal cover

38. tendency_of_ice_volume_glacierized

Units m3 year-1

Description rate of change of the ice volume in glacierized areas

CLIMATE FORCING

PISM has a well-defined separation of climate forcing from ice dynamics. This manual is about the climate forcing interface.

By contrast, most options documented in the *PISM User's Manual* control the ice dynamics part. The User's Manual does, however, give an *overview of PISM's surface (atmosphere) and ocean (sub-shelf) interfaces*. At these interfaces the surface mass and energy balances are determined and/or passed to the ice dynamics code.

To get started with climate forcing usage we need to introduce some language to describe parts of PISM. In this manual a *component* is a piece of PISM code, usually a C++ class. A combination of components (or, in some cases, one component) makes up a “model” — an implementation of a physical/mathematical description of a system.

PISM's climate forcing code has two kinds of components.

- Ones that can be used as “stand-alone” models, such as the implementation of the PDD scheme (section *Temperature-index scheme*). These are *model components*.
- Ones implementing “corrections” of various kinds, such as lapse rate corrections (sections *Lapse rate corrections* and *Temperature and precipitation lapse rate corrections*) or ice-core derived offsets (sections *Scalar temperature offsets* and *Scalar sea level offsets*, for example). These are called *modifier components* or *modifiers*.

Model components and modifiers can be chained as shown in Fig. 3.16. For example,

```
-ocean constant,delta_SL -ocean_delta_SL_file delta_SL.nc
```

combines the component providing constant (both in space and time) ocean boundary conditions with a modifier that applies scalar sea level (“SL”) offsets. This combination one of the many ocean models that can be chosen using components as building blocks.

Section *Examples and corresponding options* gives examples of combining components to choose models. Before that we address how PISM handles model time (Section *Managing model time*).

Summary of the main idea in using this manual

Setting up PISM's climate interface *requires* selecting one surface and one ocean component. The surface component may use an atmosphere component also; see Fig. 3.16. Command-line options `-atmosphere`, `-surface` and `-ocean` each take a comma-separated list of keywords as an argument; the first keyword *has* to correspond to a model component, the rest can be “modifier” components. Any of these options can be omitted to use the default atmosphere, surface or ocean model components, but one has to explicitly choose a model component to use a modifier. Model components and modifiers are chained as in Fig. 3.16.

4.1 Managing model time

Most of PISM only needs to know how long the current time step is. The climate forcing (reporting) code, on the other hand, uses time in a precise manner to provide (and report) the correct values at the right time. For example: the February mass balance should be used for 28 days (except during leap years) and not $365/12 = 30.4167$ days.

4.1.1 Periodic climate data

All components reading time-dependent forcing data from files can interpret it as “periodic”. The length of the period (in years) is specified using a `..._period` option. For example, to prescribe a periodic climate which has the same values each year but which includes inter-annual variations, using the `-surface_given` option, set:

```
-surface_given -surface_given_period 1 -surface_given_file forcing.nc
```

Each component has a unique command-line option prefix for a `..._period` option. Please refer to corresponding sections for allowed prefixes.

If forcing data has the period other than one year it is also necessary to specify the “starting time” using the `..._reference_year` option.

For example, to use a 20 year long climate record as periodic climate starting at the beginning of the model year 10, do

```
-surface_given -surface_given_period 20 -surface_given_file forcing.nc \  
-surface_given_reference_year 10
```

Note that the reference year is given in *model years*, not calendar years.

The time variable in a forcing file that is to be used as periodic should start at 0. (In other words, time in a file with periodic forcing data is *time since the beginning of a period*.) Please see the [Model time](#) for a discussion of time units appropriate in forcing files.

4.1.2 Using time bounds in forcing data

PISM interprets climate forcing data as piecewise-constant in time. A forcing file is required to contain time bounds corresponding to each record.

PISM follows the CF (Climate and Forecasting) meta-data conventions. The `ncdump -h` output from a conforming file would look similar to:

```
netcdf forcing {  
dimensions:  
    time = UNLIMITED ; // (214 currently)  
    nv = 2 ;  
variables:  
    double time(time) ;  
        time:units = "seconds since 2000-1-1" ;  
        time:axis = "T" ;  
        time:bounds = "time_bounds" ;  
        time:calendar = "gregorian" ;  
        time:long_name = "time" ;  
    double nv(nv) ;  
    double time_bounds(time, nv) ;
```


The `time_bounds` variable stores the starting and the ending time for each interval in the forcing. This variable is assumed to have the same units as the `time` variable it is associated with, which is why its arguments are not set in this example.

Please see the [CF Conventions](#) document for details.

4.2 Examples and corresponding options

This section gives a very brief overview of some coupling options. Please see sections referenced below for more information.

4.2.1 One way coupling to a climate model

One-way coupling of PISM to a climate model can be achieved by reading a NetCDF file with time- and space-dependent climate data produced by a climate model.

There are two cases:

- coupling to a climate model that includes surface (firn, snow) processes
- coupling to a climate model providing near-surface air temperature and precipitation

Reading ice surface temperature and mass balance

This is the simplest case. It is often the preferred case, for example when the climate model in use has high quality surface mass and energy sub-models which are then preferred to the highly simplified (e.g. temperature index) surface models in PISM.

Variables `climatic_mass_balance`, `ice_surface_temp`

Options `-surface given -surface_given_file forcing.nc`

See also *Reading top-surface boundary conditions from a file*

Reading air temperature and precipitation

As mentioned above, if a climate model provides near-surface air temperature and precipitation, these data need to be converted into top-of-the-ice temperature and climatic mass balance.

One way to do that is by using a temperature index (PDD) model component included in PISM. This component has adjustable parameters; default values come from [3].

Variables `precipitation`, `air_temp`

Options `-atmosphere given -atmosphere_given_file forcing.nc -surface pdd`

See also *Reading boundary conditions from a file, Temperature-index scheme*

If melt is negligible `-surface pdd` should be replaced with `-surface simple` (see section *The “invisible” model*).

4.2.2 Using climate anomalies

Prognostic modeling experiments frequently use time- and space-dependent air temperature and precipitation anomalies.

Variables `precipitation`, `air_temp`, `precipitation_anomaly`, `air_temp_anomaly`

Options -atmosphere given,anomaly, -atmosphere_given_file forcing.nc,
-atmosphere_anomaly_file anomalies.nc, -surface simple

See also *Reading boundary conditions from a file, Using climate data anomalies, The “invisible” model*

The simple surface model component re-interprets precipitation as climatic mass balance, which is useful in cases when there is no melt (Antarctic simulations is an example).

Simulations of the Greenland ice sheet typically use -surface pdd instead of -surface simple.

4.2.3 SeaRISE-Greenland

The SeaRISE-Greenland setup uses a parameterized near-surface air temperature [1] and a constant-in-time precipitation field read from an input (-i) file. A temperature-index (PDD) scheme is used to compute the climatic mass balance.

Variables precipitation, lat, lon

Options -atmosphere searise_greenland -surface pdd

See also *SeaRISE-Greenland, Temperature-index scheme*

The air temperature parameterization is a function of latitude (lat), longitude (lon) and surface elevation (dynamically updated by PISM).

4.2.4 SeaRISE-Greenland paleo-climate run

The air temperature parameterization in the previous section is appropriate for present day modeling. PISM includes some mechanisms allowing for corrections taking into account differences between present and past climates. In particular, one can use ice-core derived scalar air temperature offsets [4], precipitation adjustments [2], and sea level offsets from SPECMAP [5].

Variables precipitation, delta_T, delta_SL, lat, lon

Options -atmosphere searise_greenland,delta_T -atmosphere_delta_T_file delta_T.
nc -surface pdd -ocean constant,delta_SL -ocean_delta_SL_file delta_SL.nc

See also *SeaRISE-Greenland, Scalar temperature offsets, Temperature-index scheme, Constant in time and space, Scalar sea level offsets*

Note that the temperature offsets are applied to *air* temperatures at the *atmosphere level*. This ensures that ΔT influences the PDD computation.

4.2.5 Antarctic paleo-climate runs

Variables climatic_mass_balance, air_temp, delta_T, delta_SL

Options -surface given,delta_T -surface_delta_T_file delta_T.nc -ocean constant,
delta_SL -ocean_delta_SL_file delta_SL.nc

See also *Reading top-surface boundary conditions from a file, Scalar temperature offsets, Constant in time and space, Scalar sea level offsets*

4.3 Testing if forcing data is used correctly

It is very important to ensure that selected forcing options produce the result you expect: we find that the ice sheet response is very sensitive to provided climate forcing, especially in short-scale simulations.

This section describes how to use PISM to inspect climate forcing.

4.3.1 Visualizing climate inputs, without ice dynamics

Recall that internally in PISM there is a *separation of climate inputs from ice dynamics*. This makes it possible to turn “off” the ice dynamics code to visualize the climate mass balance and temperature boundary conditions produced using a combination of options and input files. This is helpful during the process of creating PISM-readable data files, and modeling with such.

To do this, use the option `-test_climate_models` (which is equivalent to `-stress_balance none` and `-energy none`) together with PISM’s reporting capabilities (`-extra_file`, `-extra_times`, `-extra_vars`).

Turning “off” ice dynamics saves computational time while allowing one to use the same options as in an actual modeling run. Note that `-test_climate_models` does *not* disable geometry updates, so one can check if surface elevation feedbacks modeled using lapse rates (and similar) work correctly. Please use the `-no_mass` command-line option to fix ice geometry. (This may be necessary if the mass balance rate data would result in extreme ice sheet growth that is not balanced by ice flow in this setup.)

As an example, set up an ice sheet state file and check if climate data is read in correctly:

```
mpiexec -n 2 pisms -eisII A -y 1000 -o state.nc
pismr -i state.nc -surface given -extra_times 0.0:0.1:2.5 \
      -extra_file movie.nc -extra_vars climatic_mass_balance,ice_surface_temp \
      -ys 0 -ye 2.5
```

Using `pisms` merely generates demonstration climate data, using EISMINT II choices [14]. The next run extracts the surface mass balance `climatic_mass_balance` and surface temperature `ice_surface_temp` from `state.nc`. It then does nothing interesting, exactly because a constant climate is used. Viewing `movie.nc` we see these same values as from `state.nc`, in variables `climatic_mass_balance`, `ice_surface_temp`, reported back to us as the time- and space-dependent climate at times `ys:dt:ye`. It is a boring “movie.”

A more interesting example uses a *positive degree-day scheme*. This scheme uses a variable called `precipitation`, and a calculation of melting, to get the surface mass balance `climatic_mass_balance`.

Assuming that `g20km_10ka.nc` was created *as described in the User’s Manual*, running

```
pismr -test_climate_models -no_mass -i g20km_10ka.nc \
      -atmosphere searise_greenland -surface pdd \
      -ys 0 -ye 1 -extra_times 0:1week:1 \
      -extra_file foo.nc \
      -extra_vars climatic_mass_balance,ice_surface_temp,air_temp_snapshot,precipitation
```

produces `foo.nc`. Viewing in with `ncview` shows an annual cycle in the variable `air_temp` and a noticeable decrease in the surface mass balance during summer months (see variable `climatic_mass_balance`). Note that `ice_surface_temp` is constant in time: this is the temperature *at the ice surface but below firn* and it does not include seasonal variations [15].

4.3.2 Using low-resolution test runs

Sometimes a run like the one above is still too costly. In this case it might be helpful to replace it with a similar run on a coarser grid, with or without the option `-test_climate_models`. (Testing climate inputs usually means checking

if the timing of modeled events is right, and high spatial resolution is not essential.)

The command

```
pismr -i g20km_pre100.nc -bootstrap -Mx 51 -My 101 -Mz 11 \  
-atmosphere searise_greenland \  
-surface pdd -ys 0 -ye 2.5 \  
-extra_file foo.nc -extra_times 0:0.1:2.5 \  
-extra_vars climatic_mass_balance,air_temp_snapshot,smelt,srunoff,saccum \  
-ts_file ts.nc -ts_times 0:0.1:2.5 \  
-o bar.nc
```

will produce `foo.nc` containing a “movie” very similar to the one created by the previous run, but including the full influence of ice dynamics.

In addition to `foo.nc`, the latter command will produce `ts.nc` containing scalar time-series. The variable `surface_ice_flux` (the *total over the ice-covered area* of the surface mass flux) can be used to detect if climate forcing is applied at the right time.

4.3.3 Visualizing the climate inputs in the Greenland case

Assuming that `g20km_pre100.nc` was produced by the run described in section [Getting started: a Greenland ice sheet example](#), one can run the following to check if the PDD model in PISM (see section [Temperature-index scheme](#)) is “reasonable”:

```
pismr -i g20km_pre100.nc -atmosphere searise_greenland,paleo_precip \  
-surface pdd -atmosphere_paleo_precip_file pism_dT.nc \  
-extra_times 0:1week:3 -ys 0 -ye 3 \  
-extra_file pddmovie.nc -o_order zyx \  
-extra_vars climatic_mass_balance,air_temp_snapshot
```

This produces the file `pddmovie.nc` with several variables: `climatic_mass_balance` (instantaneous net accumulation (ablation) rate), `air_temp_snapshot` (instantaneous near-surface air temperature), `precipitation` (mean annual ice-equivalent precipitation rate) and some others.

The variable `precipitation` does not evolve over time because it is part of the SeaRISE-Greenland data and is read in from the input file.

The other two variables were used to create figure [Fig. 4.1](#), which shows the time-series of the accumulation rate (top graph) and the air temperature (bottom graph) with the map view of the surface elevation on the left.

Here are two things to notice:

1. The summer peak day is in the right place. The default for this value is July 15 (day 196, at approximately $196/365 \approx 0.54$ year). (If it is important, the peak day can be changed using the configuration parameter `atmosphere.fausto_air_temp.summer_peak_day`).
2. Lows of the surface mass balance rate `climatic_mass_balance` correspond to positive degree-days in the given period, because of highs of the air temperature. Recall the air temperature graph does not show random daily variations. Even though it has the maximum of about 266 Kelvin, the parameterized instantaneous air temperature can be above freezing. A positive value for positive degree-days is expected [\[10\]](#).

We can also test the surface temperature forcing code with the following command.

```
pismr -i g20km_pre100.nc -surface simple \  
-atmosphere searise_greenland,delta_T \  
-atmosphere_delta_T_file pism_dT.nc \  
-extra_times 100 -ys -125e3 -ye 0 \  
-extra_vars ice_surface_temp \  

```

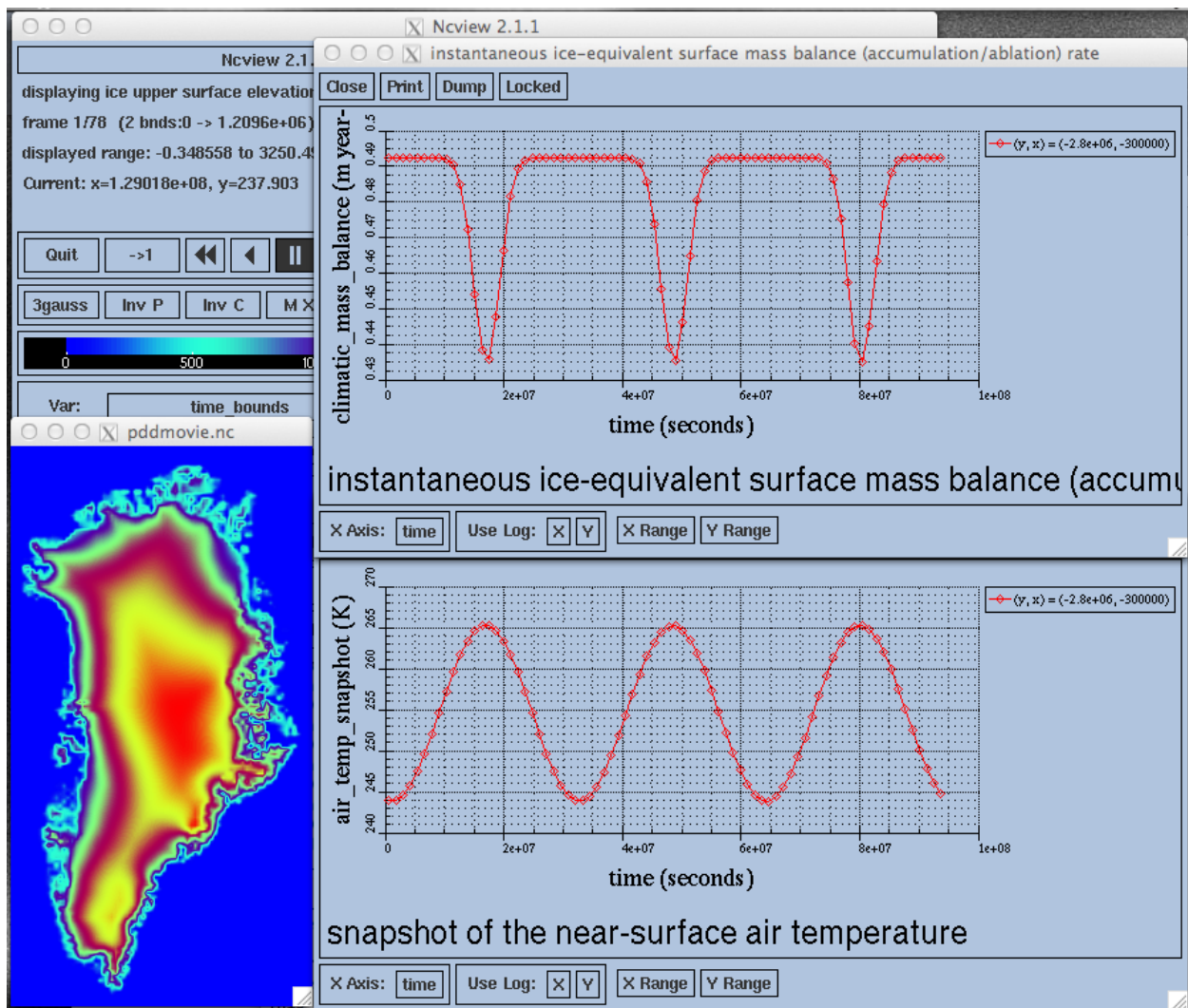


Fig. 4.1: Time series of the surface mass balance rate and near-surface air temperature.

```
-extra_file dT_movie.nc -o_order zyx \  
-test_climate_models -no_mass
```

The output `dT_movie.nc` and `pism_dT.nc` were used to create [Fig. 4.2](#).

This figure shows the GRIP temperature offsets and the time-series of the temperature at the ice surface at a point in southern Greenland (bottom graph), confirming that the temperature offsets are used correctly.

4.4 Surface mass and energy process model components

Contents

- *Surface mass and energy process model components*
 - *The “invisible” model*
 - *Reading top-surface boundary conditions from a file*
 - *Elevation-dependent temperature and mass balance*
 - *Temperature-index scheme*
 - *PIK*
 - *Scalar temperature offsets*
 - *Lapse rate corrections*
 - *Mass flux adjustment*
 - *Using climate data anomalies*
 - *The caching modifier*

4.4.1 The “invisible” model

Options `-surface simple`

Variables `none`

C++ class `pism::surface::Simple`

This is the simplest “surface model” available in PISM, enabled using `-surface simple`. Its job is to re-interpret precipitation as climatic mass balance, and to re-interpret mean annual near-surface (2m) air temperature as the temperature of the ice at the depth at which firn processes cease to change the temperature of the ice. (I.e. the temperature *below* the firn.) This implies that there is no melt. Though primitive, this model component may be desired in cold environments (e.g. East Antarctic ice sheet) in which melt is negligible and heat from firn processes is ignored.

4.4.2 Reading top-surface boundary conditions from a file

Options `-surface given`

Variables `ice_surface_temp, climatic_mass_balance $kg/(m^2s)$`

C++ class `pism::surface::Given`

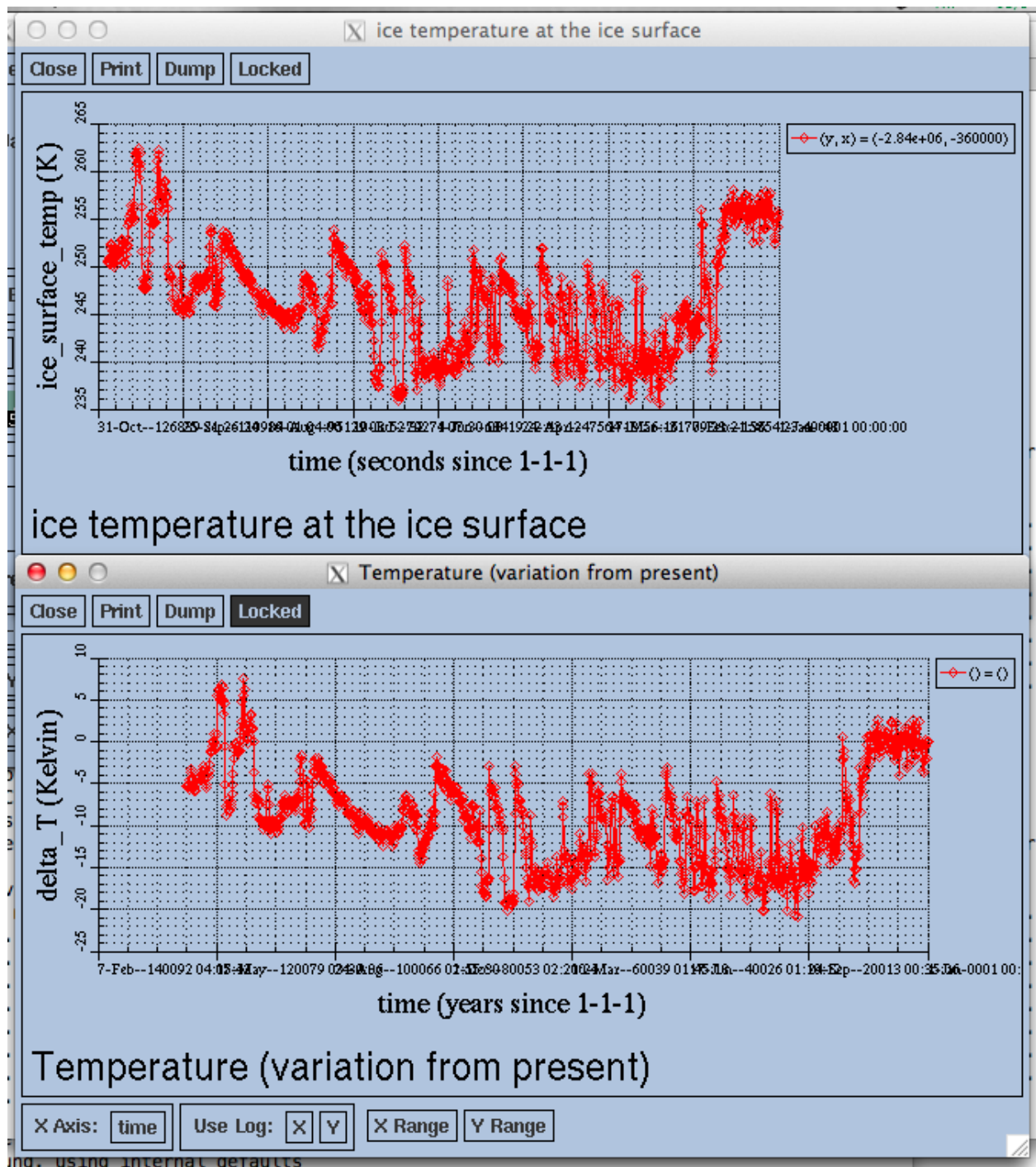


Fig. 4.2: Time series of the surface temperature compared to GRIP temperature offsets

Note: This is the default choice.

This model component was created to force PISM with sampled (possibly periodic) climate data by reading ice upper surface boundary conditions from a file. These fields are provided directly to the ice dynamics code (see [Climate inputs, and their interface with ice dynamics](#) for details).

PISM will stop if variables `ice_surface_temp` (ice temperature at the ice surface but below firm) and `climatic_mass_balance` (top surface mass flux into the ice) are not present in the input file.

Command-line options:

- `-surface_given_file` prescribes an input file
- `-surface_given_period` (*years*) makes PISM interpret data in `-surface_given_file` as periodic. See [Periodic climate data](#).
- `-surface_given_reference_year` sets the reference model year; see [Periodic climate data](#).

A file `foo.nc` used with `-surface given -surface_given_file foo.nc` should contain several records. If this file contains one record (i.e. fields corresponding to one time value only), provided forcing data is interpreted as time-independent. The time variable should describe what model time these records correspond to; see [Managing model time](#) for details.

For example, to use monthly records and period of 1 year, create a file (say, “`foo.nc`”) with 12 records. The time variable may contain 0, 1, 2, 3, ..., 11 and have the units of “month”¹. Then, run

```
pismr -surface given -surface_given_file foo.nc -surface_given_period 1
```

Note:

- This surface model *ignores* the atmosphere model selection made using the option `-atmosphere`.
- PISM can handle files with virtually any number of records: it will read and store in memory at most `climate_forcing.buffer_size` records at any given time (default: 60, or 5 years’ worth of monthly fields).
- when preparing a file for use with this model, it is best to use the `t, y, x` variable storage order: files using this order can be read in faster than ones using the `t, x, y` order, for reasons [explained in the User’s Manual](#).

To change the storage order in a NetCDF file, use `ncpdq`:

```
ncpdq -a t,y,x input.nc output.nc
```

will copy data from `input.nc` into `output.nc`, changing the storage order to `t, y, x` at the same time.

4.4.3 Elevation-dependent temperature and mass balance

Options `-surface elevation`

Variables none

C++ class `pism::surface::Elevation`

This surface model component parameterizes the ice surface temperature $T_h = \text{ice_surface_temp}$ and the mass balance $m = \text{climatic_mass_balance}$ as *piecewise-linear* functions of surface elevation h .

¹ You can use other time units supported by UDUNITS.

The option `-ice_surface_temp` (list of 4 numbers) determines the surface temperature using the 4 parameters T_{\min} , T_{\max} , h_{\min} , h_{\max} . Let

$$\frac{\partial T}{\partial h} = (T_{\max} - T_{\min}) / (h_{\max} - h_{\min})$$

be the temperature gradient. Then

$$T(x, y) = \begin{cases} T_{\min}, & h(x, y) \leq h_{\min}, \\ T_{\min} + \frac{\partial T}{\partial h} (h(x, y) - h_{\min}), & h_{\min} < h(x, y) < h_{\max}, \\ T_{\max}, & h_{\max} \leq h(x, y). \end{cases}$$

The option `-climatic_mass_balance` (list of 5 numbers) determines the surface mass balance using the 5 parameters m_{\min} , m_{\max} , h_{\min} , h_{ELA} , h_{\max} . Let

$$\frac{\partial m_{\text{abl}}}{\partial h} = -m_{\min} / (h_{\max} - h_{\min})$$

and

$$\frac{\partial m_{\text{acl}}}{\partial h} = m_{\max} / (h_{\max} - h_{\min})$$

be the mass balance gradient in the ablation and in the accumulation area, respectively. Then

$$m(x, y) = \begin{cases} m_{\min}, & h(x, y) \leq h_{\min}, \\ \frac{\partial m_{\text{abl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\min} < h(x, y) < h_{\max}, \\ \frac{\partial m_{\text{acl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\min} < h(x, y) < h_{\max}, \\ m_{\max}, & h_{\max} \leq h(x, y). \end{cases}$$

The option `-climatic_mass_balance_limits` (list of 2 numbers) limits the mass balance below h_{\min} to m_{\min}^* and above h_{\max} to m_{\max}^* , thus

$$m(x, y) = \begin{cases} m_{\min}^*, & h(x, y) \leq h_{\min}, \\ \frac{\partial m_{\text{abl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\min} < h(x, y) < h_{\max}, \\ \frac{\partial m_{\text{acl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\min} < h(x, y) < h_{\max}, \\ m_{\max}^*, & h_{\max} \leq h(x, y). \end{cases}$$

Note: this surface model *ignores* the atmosphere model selection made using the `-atmosphere` option.

4.4.4 Temperature-index scheme

Options `-surface pdd`

Variables `air_temp_sd`, `snow_depth`

C++ class `pism::surface::TemperatureIndex`

The default PDD model used by PISM, turned on by option `-surface pdd`, is based on [10] and EISMINT-Greenland intercomparison (see [3]).

Our model computes the solid (snow) precipitation rate using the air temperature threshold with a linear transition. All precipitation during periods with air temperatures above `air_temp_all_precip_as_rain` (default of 2°C) is interpreted as rain; all precipitation during periods with air temperatures below `air_temp_all_precip_as_snow` (default of 0°C) is interpreted as snow.

For long-term simulations, a PDD model generally uses an idealized seasonal temperature cycle. “White noise” is added to this cycle to simulate additional daily variability associated to the vagaries of weather. This additional

random variation is quite significant, as the seasonal cycle may never reach the melting point but that point may be reached with some probability, in the presence of the daily variability, and thus melt may occur. Concretely, a normally-distributed, mean zero random temperature increment is added to the seasonal cycle. There is no assumed spatial correlation of daily variability. The standard deviation of the daily variability is controlled by command-line options:

- `-pdd_sd_file`, which prescribes an input file.
- `-pdd_sd_period (years)`, which interprets its data as periodic; see *Periodic climate data*.
- `-pdd_sd_reference_year`, which sets the reference model year; see *Periodic climate data*.

A file `foo.nc` used with `-surface pdd -pdd_sd_file foo.nc` should contain standard deviation of near-surface air temperature in variable `air_temp_sd`, and the corresponding time coordinate in variable `time`. If `-pdd_sd_file` is not set, PISM uses a constant value for standard deviation, which is set by the `surface.pdd.std_dev` configuration parameter. The default value is 5.0 degrees [3]. However, this approach is not recommended as it induces significant errors in modeled surface mass balance in both ice-covered and ice-free regions [11], [12].

Over ice-covered grid cells, daily variability can also be parameterized as a linear function of near-surface air temperature $\sigma = a \cdot T + b$ using the `surface.pdd.std_dev_use_param` configuration flag, and the corresponding parameters `surface.pdd.std_dev_param_a` and `surface.pdd.std_dev_param_b`. This parametrization replaces prescribed standard deviation values over glacierized grid cells as defined by the mask variable (see `geometry.ice_free_thickness_standard`). Default values for the slope a and intercept b were derived from the ERA-40 reanalysis over the Greenland ice sheet [13].

The number of positive degree days is computed as the magnitude of the temperature excursion above 0°C multiplied by the duration (in days) when it is above zero.

In PISM there are two methods for computing the number of positive degree days. The first computes only the expected value, by the method described in [10]. This is the default when a PDD is chosen (i.e. option `-surface pdd`). The second is a Monte Carlo simulation of the white noise itself, chosen by adding the option `-pdd_rand`. This Monte Carlo simulation adds the same daily variation at every point, though the seasonal cycle is (generally) location dependent. If repeatable randomness is desired use `-pdd_rand_repeatable` instead of `-pdd_rand`.

By default, the computation summarized in Fig. 4.3 is performed every week. (This frequency is controlled by the `surface.pdd.max_evals_per_year` parameter.) To compute mass balance during each week-long time-step, PISM keeps track of the current snow depth (using units of ice-equivalent thickness). This is necessary to determine if melt should be computed using the degree day factor for snow (`surface.pdd.factor_snow`) or the corresponding factor for ice (`surface.pdd.factor_ice`).

A fraction of the melt controlled by the configuration parameter `surface.pdd.refreeze` (θ_{refreeze} in Fig. 4.3, default: 0.6) refreezes. The user can select whether melted ice should be allowed to refreeze using the `surface.pdd.refreeze_ice_melt` configuration flag.

Since PISM does not have a principled firn model, the snow depth is set to zero at the beginning of the balance year. See `surface.pdd.balance_year_start_day`. Default is 274, corresponding to October 1st.

Our PDD implementation is meant to be used with an atmosphere model implementing a cosine yearly cycle such as `searise_greenland` (section *SeaRISE-Greenland*), but it is not restricted to parameterizations like these.

This code also implements latitude- and mean July temperature dependent ice and snow factors using formulas (6) and (7) in [1]; set `-pdd_fausto` to enable. The default standard deviation of the daily variability (`-pdd_std_dev` option) is 2.53 degrees under the `-pdd_fausto` option [1]. See also configuration parameters with the `surface.pdd.fausto` prefix.

Note that when used with periodic climate data (air temperature and precipitation) that is read from a file (see section *Reading boundary conditions from a file*), use of `-timestep_hit_multiplies X` is recommended. (Here X is the length of the climate data period in years.)

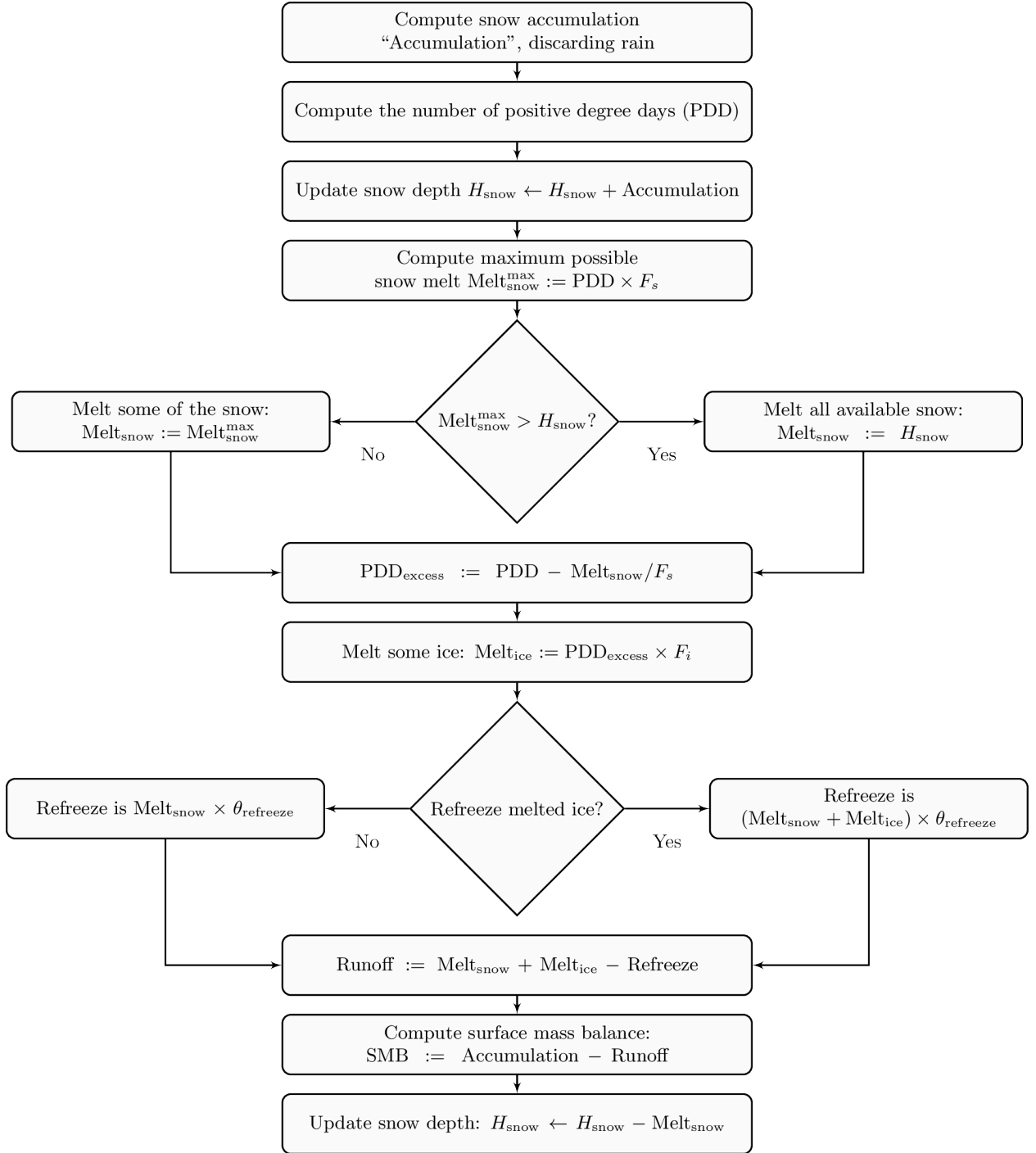


Fig. 4.3: PISM's positive degree day model. F_s and F_i are PDD factors for snow and ice, respectively; θ_{refreeze} is the refreeze fraction.

4.4.5 PIK

Options -surface pik

Variables climatic_mass_balance $kg/(m^2s)$, lat (latitude), (degrees north)

C++ class pism::surface::PIK

This surface model component implements the setup used in [6]. The climatic_mass_balance is read from an input (-i) file; the ice surface temperature is computed as a function of latitude (variable lat) and surface elevation (dynamically updated by PISM). See equation (1) in [6].

4.4.6 Scalar temperature offsets

Options -surface ...,delta_T

Variables delta_T

C++ class pism::surface::Delta_T

Command-line options:

- -surface_delta_T_file sets the name of the file PISM will read delta_T from.
- -surface_delta_T_period (years) sets the period of the forcing data (section *Periodic climate data*)
- -surface_delta_T_reference_year sets the reference year (section *Periodic climate data*).

The time-dependent scalar offsets delta_T are added to ice_surface_temp computed by a surface model.

Please make sure that delta_T has the units of “Kelvin”.

This modifier is identical to the corresponding atmosphere modifier, but applies offsets at a different stage in the computation of top-surface boundary conditions needed by the ice dynamics core.

4.4.7 Lapse rate corrections

Options -surface ...,lapse_rate

Variables surface_altitude (CF standard name),

C++ class pism::surface::LapseRates

The lapse_rate modifier allows correcting ice-surface temperature and surface mass balance using elevation lapse rates. It uses the following options.

- -temp_lapse_rate gives the temperature lapse rate, in K/km . Note that we use the following definition of the temperature lapse rate:

$$\gamma = -\frac{dT}{dz}.$$

- -smb_lapse_rate gives the surface mass balance lapse rate, in $m/year/km$. Here, $\gamma = -\frac{dM}{dz}$.
- -surface_lapse_rate_file specifies the file containing the reference surface elevation field (standard name: surface_altitude). This file can contain several surface elevation records to use lapse rate corrections relative to time-dependent surface. If one record is provided, the reference surface elevation is assumed to be time-independent.
- -surface_lapse_rate_period gives the period, in model years, to use when interpreting data in the file given with -surface_given_file,

- `-surface_lapse_rate_reference_year` takes the time T in model years. The record for t years in `-surface_given_file` is interpreted as corresponding to t years since T .

4.4.8 Mass flux adjustment

Options `-surface ... ,forcing`

Variables `thk` (ice thickness), `ftt_mask` (mask of zeros and ones; 1 where surface mass flux is adjusted and 0 elsewhere)

C++ class `pism::surface::ForceThickness`

The `forcing` modifier implements a surface mass balance adjustment mechanism which forces the thickness of grounded ice to a target thickness distribution at the end of the run. The idea behind this mechanism is that spinup of ice sheet models frequently requires the surface elevation to come close to measured values at the end of a run. A simpler alternative to accomplish this, namely option `-no_mass`, represents an unmodeled, frequently large, violation of the mass continuity equation.

In more detail, let H_{tar} be the target thickness. Let H be the time-dependent model thickness. The surface model component described here produces the term M in the mass continuity equation:

$$\frac{\partial H}{\partial t} = M - S - \nabla \cdot \mathbf{q}.$$

(Other details of this equation do not concern us here.) The `forcing` modifier causes M to be adjusted by a multiple of the difference between the target thickness and the current thickness,

$$\Delta M = \alpha(H_{\text{tar}} - H)$$

where $\alpha > 0$. We are adding mass ($\Delta M > 0$) where $H_{\text{tar}} > H$ and ablating where $H_{\text{tar}} < H$.

Option `-force_to_thickness_file` identifies the file containing the target ice thickness field `thk` and the mask `ftt_mask`. A basic run modifying surface model `given` would look like

```
pismr -i foo.nc -surface given,forcing -force_to_thickness_file bar.nc
```

In this case `foo.nc` contains fields `climatic_mass_balance` and `ice_surface_temp`, as normal for `-surface given`, and `bar.nc` contains fields `thk` which will serve as the target thickness and `ftt_mask` which defines the map plane area where this adjustment is applied. Option `-force_to_thickness_alpha` adjusts the value of α , which has a default value specified in the [Configuration parameters](#).

In addition to this one can specify a multiplicative factor C used in areas where the target thickness field has less than `-force_to_thickness_ice_free_thickness_threshold` meters of ice; $\alpha_{\text{ice free}} = C \times \alpha$. Use the `-force_to_thickness_ice_free_alpha_factor` option to set C .

4.4.9 Using climate data anomalies

Options `-surface ... ,anomaly`

Variables `ice_surface_temp_anomaly`, `climatic_mass_balance_anomaly` $\text{kg}/(\text{m}^2 \text{s})$

C++ class `pism::surface::Anomaly`

This modifier implements a spatially-variable version of `-surface ... ,delta_T` which also applies time-dependent climatic mass balance anomalies.

It takes the following options:

- `-surface_anomaly_file` specifies a file containing variables `ice_surface_temp_anomaly` and `climatic_mass_balance_anomaly`.
- `-surface_anomaly_period` (years) specifies the period of the forcing data, in model years; see *Periodic climate data*
- `-surface_anomaly_reference_year` specifies the reference year; see *Periodic climate data*

See also to `-atmosphere ... , anomaly` (section *Using climate data anomalies*), which is similar, but applies anomalies at the atmosphere level.

4.4.10 The caching modifier

Options `-surface ... ,cache`

C++ class `pism::surface::Cache`

See also *The caching modifier*

This modifier skips surface model updates, so that a surface model is called no more than every `-surface_cache_update_interval` years. A time-step of 1 year is used every time a surface model is updated.

This is useful in cases when inter-annual climate variability is important, but one year differs little from the next. (Coarse-grid paleo-climate runs, for example.)

It takes the following options:

- `-surface_cache_update_interval` (years) Specifies the minimum interval between updates. PISM may take longer time-steps if the adaptive scheme allows it, though.

4.5 Atmosphere model components

Contents

- *Atmosphere model components*
 - *Reading boundary conditions from a file*
 - *Cosine yearly cycle*
 - *SeaRISE-Greenland*
 - *PIK*
 - *One weather station*
 - *Scalar temperature offsets*
 - *Scalar precipitation offsets*
 - *Scalar precipitation scaling*
 - *Precipitation correction using scalar temperature offsets*
 - *Temperature and precipitation lapse rate corrections*
 - *Using climate data anomalies*

4.5.1 Reading boundary conditions from a file

Options -atmosphere given

Variables air_temp, precipitation $kg/(m^2s)$

C++ class pism::atmosphere::Given

See also *Reading top-surface boundary conditions from a file*

Note: This is the default choice.

Command-line options:

- -atmosphere_given_file prescribes an input file
- -atmosphere_given_period (*years*) makes PISM interpret data in -atmosphere_given_file as periodic. See section *Periodic climate data*.
- -atmosphere_given_reference_year sets the reference model year; see section *Periodic climate data*.

A file `foo.nc` used with -atmosphere given -atmosphere_given_file `foo.nc` should contain several records; the time variable should describe what model time these records correspond to.

This model component was created to force PISM with sampled (possibly periodic) climate data, e.g. using monthly records of `air_temp` and `precipitation`.

It can also be used to drive a temperature-index (PDD) climatic mass balance computation (section *Temperature-index scheme*).

4.5.2 Cosine yearly cycle

Options -atmosphere yearly_cycle

Variables air_temp_mean_annual, air_temp_mean_july, precipitation $kg/(m^2s)$
amplitude_scaling

C++ class pism::atmosphere::CosineYearlyCycle

This atmosphere model component computes the near-surface air temperature using the following formula:

$$T(\text{time}) = T_{\text{mean annual}} + A(\text{time}) \cdot (T_{\text{mean July}} - T_{\text{mean annual}}) \cdot \cos(2\pi t),$$

where t is the year fraction “since last July”; the summer peak of the cycle is on `atmosphere.fausto_air_temp.summer_peak_day`, which is set to day 196 by default (approximately July 15).

Here $T_{\text{mean annual}}$ (variable `air_temp_mean_annual`) and $T_{\text{mean July}}$ (variable `air_temp_mean_july`) are read from a file selected using the -atmosphere_yearly_cycle_file command-line option. A time-independent precipitation field (variable `precipitation`) is read from the same file.

Optionally a time-dependent scalar amplitude scaling $A(t)$ can be used. Specify a file to read it from using the -atmosphere_yearly_cycle_scaling_file command-line option. Without this option $A(\text{time}) \equiv 1$.

4.5.3 SeaRISE-Greenland

Options -atmosphere searise_greenland

Variables lon, lat, precipitation $kg/(m^2s)$

C++ class `pism::atmosphere::SeaRISEGreenland`

See also *Precipitation correction using scalar temperature offsets*

This atmosphere model component implements a longitude, latitude, and elevation dependent near-surface air temperature parameterization and a cosine yearly cycle described in [1] and uses a constant in time ice-equivalent precipitation field (in units of thickness per time, variable `precipitation`) that is read from an input (`-i`) file. To read time-independent precipitation from a different file, use the option `-atmosphere_searise_greenland_file`.

The air temperature parameterization is controlled by configuration parameters with the `snow_temp_fausto` prefix.

See also the `-atmosphere ... ,paleo_precip` modifier, section *Precipitation correction using scalar temperature offsets*, for an implementation of the SeaRISE-Greenland formula for paleo-precipitation correction from present; a 7.3% change of precipitation rate for every one degree Celsius of temperature change [2].

4.5.4 PIK

Options `-atmosphere pik`

Variables `lat`, `precipitation`

C++ class `pism::atmosphere::PIK`

This model component reads a time-independent precipitation field from an input (`-i`) file and computes near-surface air temperature using a latitude and surface elevation-dependent formula.

The parameterization is the same as in the `-surface pik` model, section *PIK*.

4.5.5 One weather station

Options `-atmosphere one_station -atmosphere_one_station_file`

Variables `air_temp` [Kelvin], `precipitation` $kg/(m^2s)$

C++ class `pism::atmosphere::WeatherStation`

This model component reads scalar time-series of the near-surface air temperature and precipitation from a file specified using the `-atmosphere_one_station_file` option and uses them at *all* grid points in the domain. In other words, resulting climate fields are constant in space but not necessarily in time.

The `-atmosphere one_station` model should be used with a modifier such as `lapse_rate` (see section *Temperature and precipitation lapse rate corrections*) to create spatial variability.

4.5.6 Scalar temperature offsets

Options `-atmosphere ... ,delta_T`

Variables `delta_T`

C++ class `pism::atmosphere::Delta_T`

This modifier applies scalar time-dependent air temperature offsets to the output of an atmosphere model. It takes the following command-line options.

- `-atmosphere_delta_T_file` sets the name of the file PISM will read `delta_T` from.
- `-atmosphere_delta_T_period` (*years*) sets the period of the forcing data (section *Periodic climate data*).
- `-atmosphere_delta_T_reference_year` sets the reference year (section *Periodic climate data*).

Please make sure that `delta_T` has the units of “Kelvin”.

4.5.7 Scalar precipitation offsets

Options -atmosphere ...,delta_P

Variables delta_P $kg/(m^2s)$

C++ class pism::atmosphere::Delta_P

This modifier applies scalar time-dependent precipitation offsets to the output of an atmosphere model. It takes the following command-line options.

- -atmosphere_delta_P_file sets the name of the file PISM will read delta_P from.
- -atmosphere_delta_P_period (*years*) sets the period of the forcing data (section *Periodic climate data*).
- -atmosphere_delta_P_reference_year sets the reference year (section *Periodic climate data*).

4.5.8 Scalar precipitation scaling

Options -atmosphere ...,frac_P

Variables frac_P [no unit]

C++ class pism::atmosphere::Frac_P

This modifier scales precipitation output of an atmosphere model using a scalar time-dependent precipitation fraction, with a value of one corresponding to no change in precipitation. It takes the following command-line options:

- -atmosphere_frac_P_file sets the name of the file PISM will read frac_P from.
- -atmosphere_frac_P_period (*years*) sets the period of the forcing data (section *Periodic climate data*).
- -atmosphere_frac_P_reference_year sets the reference year (section *Periodic climate data*).

4.5.9 Precipitation correction using scalar temperature offsets

Options -atmosphere ...,paleo_precip

Variables delta_T [degrees Kelvin]

C++ class pism::atmosphere::PaleoPrecip

This modifier implements the SeaRISE-Greenland formula for a precipitation correction from present; a 7.3% change of precipitation rate for every one degree Celsius of air temperature change [2]. See *SeaRISE Greenland model initialization* for details. The input file should contain air temperature offsets in the format used by -atmosphere ...,delta_T modifier, see section *Scalar temperature offsets*.

It takes the following command-line options.

- -atmosphere_paleo_precip_file sets the name of the file PISM will read delta_T from.
- -atmosphere_paleo_precip_period (*years*) sets the period of the forcing data (section *Periodic climate data*).
- -atmosphere_paleo_precip_reference_year sets the reference year (section *Periodic climate data*).

4.5.10 Temperature and precipitation lapse rate corrections

Options -atmosphere ...,lapse_rate

Variables surface_altitude (CF standard name)

C++ class `pism::atmosphere::LapseRates`

The `lapse_rate` modifier allows for correcting air temperature and precipitation using elevation lapse rates. It uses the following options.

- `-temp_lapse_rate` gives the temperature lapse rate, in K/km . Note that we use the following definition of the temperature lapse rate:

$$\gamma = -\frac{dT}{dz}.$$

- `-precip_lapse_rate` gives the precipitation lapse rate, in $(m/year)/km$. Here $\gamma = -\frac{dM}{dz}$.
- `-atmosphere_lapse_rate_file` specifies a file containing the reference surface elevation field (standard name: `surface_altitude`). This file may contain several surface elevation records to use lapse rate corrections relative to a time-dependent surface. If one record is provided, the reference surface elevation is assumed to be time-independent.
- `-atmosphere_lapse_rate_period` gives the period, in model years; see section [Periodic climate data](#).
- `-atmosphere_lapse_rate_reference_year` specifies the reference date; see section [Periodic climate data](#).

4.5.11 Using climate data anomalies

Options `-atmosphere ... ,anomaly`

Variables `air_temp_anomaly, precipitation_anomaly` $kg/(m^2s)$

C++ class `pism::atmosphere::Anomaly`

This modifier implements a spatially-variable version of `-atmosphere ... ,delta_T,delta_P`.

It takes the following options:

- `-atmosphere_anomaly_file` specifies a file containing variables `air_temp_anomaly` and `precipitation_anomaly`.
- `-atmosphere_anomaly_period` (years) specifies the period of the forcing data, in model years; section [Periodic climate data](#).
- `-atmosphere_anomaly_reference_year` specifies the reference year; section [Periodic climate data](#).

See also to `-surface ... ,anomaly` (section [Using climate data anomalies](#)), which is similar, but applies anomalies at the surface level.

4.6 Ocean model components

PISM Ocean model components provide sub-shelf ice temperature (`shelfbtemp`) and sub-shelf mass flux (`shelfbmassflux`) to the ice dynamics core.

The sub-shelf ice temperature is used as a Dirichlet boundary condition in the energy conservation code. The sub-shelf mass flux is used as a source in the mass-continuity (transport) equation. Positive flux corresponds to ice loss; in other words, this sub-shelf mass flux is a “melt rate”.

Contents

- [Ocean model components](#)

- *Constant in time and space*
- *Reading forcing data from a file*
- *PIK*
- *Basal melt rate and temperature from thermodynamics in boundary layer*
- *Scalar sea level offsets*
- *Scalar sub-shelf temperature offsets*
- *Scalar sub-shelf mass flux offsets*
- *Scalar sub-shelf mass flux fraction offsets*
- *Scalar melange back pressure fraction*
- *The caching modifier*

4.6.1 Constant in time and space

Options `-ocean constant`

Variables none

C++ class `pism::ocean::Constant`

Note: This is the default choice.

This ocean model component implements boundary conditions at the ice/ocean interface that are constant *both* in space and time.

The sub-shelf ice temperature is set to pressure melting and the sub-shelf melt rate is assumed to be proportional to the heat flux from the ocean into the ice (configuration parameter `ocean.sub_shelf_heat_flux_into_ice`).

Alternatively, the sub-shelf melt rate in meters per year can be set using the `-shelf_base_melt_rate` command-line option.

4.6.2 Reading forcing data from a file

Options `-ocean given`

Variables `shelfbtemp` Kelvin, `shelfbmassflux` $kg/(m^2s)$

C++ class `pism::ocean::Given`

This ocean model component reads sub-shelf ice temperature `shelfbtemp` and the sub-shelf mass flux `shelfbmassflux` from a file. It takes the following command-line options.

- `-ocean_given_file`: sets the name of the file to read forcing data from. The file may contain several records. If only one record is provided it is interpreted as time-independent.
- `-ocean_given_reference_year` specifies the reference date; see section [Periodic climate data](#).
- `-ocean_given_period` specifies the length of the period of the forcing data, in model years; see section [Periodic climate data](#).

Variables `shelfbtemp` and `shelfbmassflux` may be time-dependent. (The `-ocean given` component is very similar to `-surface given` and `-atmosphere given`.)

4.6.3 PIK

Options -ocean pik

Variables none

C++ class pism::ocean::PIK

This ocean model component implements the ocean forcing setup used in [6]. The sub-shelf ice temperature is set to pressure-melting; the sub-shelf mass flux computation follows [7].

It takes one command-line option:

- -meltfactor_pik: a melt factor F_{melt} in sub-shelf-melting parameterization, see equation (5) in [6].

4.6.4 Basal melt rate and temperature from thermodynamics in boundary layer

Options -ocean th

Variables theta_ocean (absolute potential ocean temperature), [Kelvin], salinity_ocean (salinity of the adjacent ocean), [g/kg]

C++ class pism::ocean::GivenTH

This ocean model component derives basal melt rate and basal temperature from thermodynamics in a boundary layer at the base of the ice shelf. It uses a set of three equations describing

1. the energy flux balance,
2. the salt flux balance,
3. the pressure and salinity dependent freezing point in the boundary layer.

This model is described in [8] and [9].

Inputs are potential temperature (variable theta_ocean) and salinity (variable salinity_ocean) read from a file.

No ocean circulation is modeled, so melt water computed by this model is not fed back into the surrounding ocean.

This implementation uses different approximations of the temperature gradient at the base of an ice shelf column depending on whether there is sub-shelf melt, sub-shelf freeze-on, or neither (see [8] for details).

It takes two command-line options:

- -ocean_th_file: specifies the NetCDF file providing potential temperature and salinity fields.
- -clip_shelf_base_salinity: if this is set (which is the default), the sub-shelf salinity is clipped so that it stays in the [4, 40] psu range. This is done to ensure that we stay in the range of applicability of the melting point temperature parameterization; see [8]. To disable salinity clipping, use the -no_clip_shelf_base_salinity option or set the ocean_three_equation_model_clip_salinity configuration parameter to “no”.

4.6.5 Scalar sea level offsets

Options -ocean ...,delta_SL

Variables delta_SL (meters)

C++ class pism::ocean::Delta_SL

The delta_SL modifier implements sea level forcing using scalar offsets.

It takes the following command-line options:

- `-ocean_delta_SL_file`: specifies the name of the file containing forcing data. This file has to contain the `delta_SL` variable using units “meters” or equivalent.
- `-ocean_delta_SL_period` specifies the length of the period of the forcing data, in model years; see section [Periodic climate data](#).
- `-ocean_delta_SL_reference_year` specifies the reference date; see section [Periodic climate data](#).

4.6.6 Scalar sub-shelf temperature offsets

Options `-ocean ... ,delta_T`

Variables `delta_T` (Kelvin)

C++ class `pism::ocean::Delta_T`

This modifier implements forcing using sub-shelf ice temperature offsets.

It takes the following command-line options:

- `-ocean_delta_T_file`: specifies the name of the file containing forcing data. This file has to contain the `delta_T` variable using units of “Kelvin” or equivalent.
- `-ocean_delta_T_period` specifies the length of the period of the forcing data, in model years; see section [Periodic climate data](#).
- `-ocean_delta_T_reference_year` specifies the reference date; see section [Periodic climate data](#).

4.6.7 Scalar sub-shelf mass flux offsets

Options `-ocean ... ,delta_SMB`

Variables `delta_SMB` $kg/(m^2s)$

C++ class `pism::ocean::Delta_SMB`

This modifier implements forcing using sub-shelf mass flux (melt rate) offsets.

It takes the following command-line options:

- `-ocean_delta_SMB_file`: specifies the name of the file containing forcing data. This file has to contain the `delta_SMB` variable using units $kg/(m^2s)$ or equivalent.
- `-ocean_delta_SMB_period` specifies the length of the period of the forcing data, in model years; see section [Periodic climate data](#).
- `-ocean_delta_SMB_reference_year` specifies the reference date; see section [Periodic climate data](#).

4.6.8 Scalar sub-shelf mass flux fraction offsets

Options `-ocean ... ,frac_SMB`

Variables `frac_SMB` [1]

C++ class `pism::ocean::Frac_SMB`

This modifier implements forcing using sub-shelf mass flux (melt rate) fraction offsets.

It takes the following command-line options:

- `-ocean_frac_SMB_file`: specifies the name of the file containing forcing data. This file has to contain the `frac_SMB` variable.

- `-ocean_frac_SMB_period` specifies the length of the period of the forcing data, in model years; see section *Periodic climate data*.
- `-ocean_frac_SMB_reference_year` specifies the reference date; see section *Periodic climate data*.

4.6.9 Scalar melange back pressure fraction

Options `-ocean ... ,frac_MBP`

Variables `frac_MBP`

C++ class `pism::ocean::Frac_MBP`

This modifier implements forcing using melange back pressure fraction offsets. The variable `frac_MBP` should take on values from 0 to 1; it is understood as the fraction of the maximum melange back pressure possible at a given location. (We assume that melange back pressure cannot exceed the pressure of the ice column at a calving front.)

Please see *Modeling melange back-pressure* for details.

This modifier takes the following command-line options:

- `-ocean_frac_MBP_file`: specifies the name of the file containing forcing data. This file has to contain the `frac_MBP` variable using units of “1” (a dimensionless parameter)
- `-ocean_frac_MBP_period` specifies the length of the period of the forcing data, in model years; see section *Periodic climate data*.
- `-ocean_frac_MBP_reference_year` specifies the reference date; see section *Periodic climate data*.

4.6.10 The caching modifier

Options `-ocean ... ,cache`

C++ class `pism::ocean::Cache`

See also *The caching modifier*

This modifier skips ocean model updates, so that a ocean model is called no more than every `-ocean_cache_update_interval` years. A time-step of 1 year is used every time a ocean model is updated.

This is useful in cases when inter-annual climate variability is important, but one year differs little from the next. (Coarse-grid paleo-climate runs, for example.)

It takes the following options:

- `-ocean_cache_update_interval` (*years*) Specifies the minimum interval between updates. PISM may take longer time-steps if the adaptive scheme allows it, though.

TECHNICAL NOTES

5.1 CF standard names used by PISM

5.1.1 Existing standard names

We start by listing standard names from the CF Standard Name Table. The subset here is a small subset of the [table](#); we list only

- those with “land_ice” in the name and
- those currently used by PISM

The existing names starting with “land_ice” are believed to have all been submitted by Magnus Hagdorn to the CF committee circa 2003. The [SeaRISE assessment process](#) now has a [wiki on CF standard name use](#), which to a significant extent duplicates content regarding proposed names on this page. That wiki is an evolving community standard, and it supercedes this page when it comes to actual evolving standards.

Go to the [CF Conventions](#) page for the list of proposed standard names under consideration.

Because of the use of [UDUNITS](#), PISM input files do not have to have fields already in the canonical units. Rather, the units attribute has to be valid for UDUNITS conversion into the canonical units. Generally within PISM, the canonical units are used internally.

Table 5.1: CF standard names used by PISM

CF standard name	Canonical units (SI)
bedrock_altitude	m
floating_ice_sheet_area_fraction	1
grounded_ice_sheet_area_fraction	1
land_ice_area_fraction	1
land_ice_basal_melt_rate	m s ⁻¹
land_ice_basal_temperature	K
land_ice_basal_upward_velocity	m s ⁻¹
land_ice_basal_x_velocity	m s ⁻¹
land_ice_basal_y_velocity	m s ⁻¹
land_ice_calving_rate	m s ⁻¹
land_ice_lwe_basal_melt_rate	m s ⁻¹
land_ice_lwe_calving_rate	m s ⁻¹
land_ice_lwe_surface_specific_mass_balance	m s ⁻¹
land_ice_sigma_coordinate	1
land_ice_specific_mass_flux_due_to_calving_and_ice_front_melting	kg m ⁻² s ⁻¹
land_ice_surface_specific_mass_balance_flux	kg m ⁻² s ⁻¹

Continued on next page

Table 5.1 – continued from previous page

CF standard name	Canonical units (SI)
land_ice_surface_upward_velocity	m s-1
land_ice_surface_x_velocity	m s-1
land_ice_surface_y_velocity	m s-1
land_ice_temperature	K
land_ice_thickness	m
land_ice_vertical_mean_x_velocity	m s-1
land_ice_vertical_mean_y_velocity	m s-1
land_ice_x_velocity	m s-1
land_ice_y_velocity	m s-1
latitude	degree_north
longitude	degree_east
magnitude_of_land_ice_basal_drag	Pa
projection_x_coordinate	m
projection_y_coordinate	m
surface_altitude	m
temperature_at_ground_level_in_snow_or_firn	K
tendency_of_bedrock_altitude	m s-1
tendency_of_land_ice_mass_due_to_basal_mass_balance	kg s-1
tendency_of_land_ice_thickness	m s-1
upward_geothermal_heat_flux_at_ground_level	W m-2

5.1.2 *Proposed* standard names

These are *unofficially* proposed by Bueler and Aschwanden, for now.

Table 5.2: Desired CF standard names

Proposed name	Canonical units (SI)	Comments
ice_shelf_basal_specific_mass_balance	m s ⁻¹	positive is loss of ice shelf mass (i.e. use outward normal from ice shelf)
ice_shelf_basal_temperature	K	absolute (not pressure-adjusted) temperature
land_ice_age	s	
land_ice_basal_frictional_heating	W m ⁻²	
land_ice_basal_material_yield_stress	Pa	
land_ice_basal_material_friction_angle	degree	majority of standard names with “angle” use canonical units “degree”
land_ice_surface_temperature_below_firn	K	
land_ice_upward_velocity	m s ⁻¹	compare to CF names “upward_air_velocity” and “upward_sea_water_velocity”
lithosphere_temperature	K	
upward_geothermal_flux_in_lithosphere	W m ⁻²	typically applied at depth in lithosphere; compare to “upward_geothermal_heat_flux_at_ground_level”
land_ice_specific_enthalpy	J kg ⁻¹	enthalpy is defined in PISM to be sensible plus latent heat, plus potential energy of pressure; there is a nontrivial issue of the scaling; the enthalpy value for 273.15 K (cold) ice at atmospheric pressure is a possible standard
land_ice_liquid_fraction	1	liquid water fraction in ice, a pure number between 0 and 1; a diagnostic function of enthalpy

5.1.3 Final technical notes

- PISM also uses attributes `grid_mapping = "mapping"` ; and `coordinates = "lat lon"`; on output variables that depend on `y, x`.
- Because PISM uses UDUNITS, it will write some variables in “glaciological units” instead of the SI units listed above, for instance velocities in m year⁻¹ instead of m s⁻¹. This is allowed under CF. When PISM reads such a field from a NetCDF file, the conversion is handled automatically by UDUNITS.

5.2 On the vertical coordinate in PISM, and a critical change of variable

In PISM all fields in the ice, including enthalpy, age, velocity, and so on, evolve within an ice fluid domain of *changing geometry*. See Fig. 5.1. In particular, the upper and lower surfaces of the ice fluid move with respect to the geoid.

Note: FIXME: This figure should show the floating case too.

The (x, y, z) coordinates in Fig. 5.1 are supposed to be from an orthogonal coordinate system with z in the direction anti-parallel to gravity, so this is a flat-earth approximation. In practice, the data inputs to PISM are in some particular projection, of course.

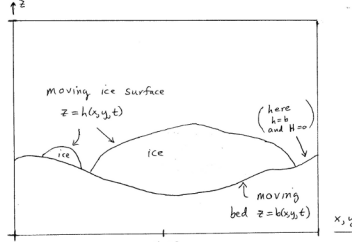


Fig. 5.1: The ice fluid domain evolves, with both the upper and lower surfaces in motion with respect to the geoid.

We make a change of the independent variable z which simplifies how PISM deals with the changing geometry of the ice, especially in the cases of a non-flat or moving bed. We replace the vertical coordinate relative to the geoid with the vertical coordinate relative to the base of the ice. Let

$$s = \begin{cases} z - b(x, y, t), & \text{ice is grounded,} \\ z - \frac{\rho_i}{\rho_o} H(x, y, t), & \text{ice is floating;} \end{cases}$$

where $H = h - b$ is the ice thickness and ρ_i, ρ_o are densities of ice and ocean, respectively.

Now we make the change of variables

$$(x, y, z, t) \mapsto (x, y, s, t)$$

throughout the PISM code. This replaces $z = b$ by $s = 0$ as the equation of the base surface of the ice. The ice fluid domain in the new coordinates only has a free upper surface as shown in Fig. 5.2.

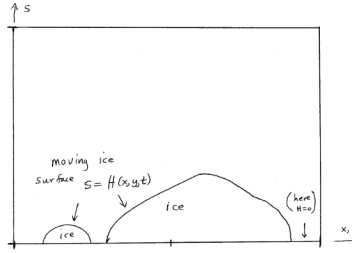


Fig. 5.2: In (x, y, s) space the ice fluid domain only has an upper surface which moves, $s = H(x, y, t)$. Compare to Fig. 5.1.

Note: FIXME: This figure should show the floating case too, and bedrock.”

In PISM the computational domain (region)

$$\mathcal{R} = \{(x, y, s) \mid -L_x \leq x \leq L_x, -L_y \leq y \leq L_y, -Lb_z \leq s \leq L_z\}$$

is divided into a three-dimensional grid. See IceGrid.

The change of variable $z \rightarrow s$ used here is *not* the [133] change of variable $\tilde{s} = (z - b)/H$. That change causes the conservation of energy equation to become singular at the boundaries of the ice sheet. Specifically, the Jenssen change replaces the vertical conduction term by a manifestly-singular term at ice sheet margins where $H \rightarrow 0$, because

$$\frac{\partial^2 E}{\partial z^2} = \frac{1}{H^2} \frac{\partial^2 E}{\partial \tilde{s}^2}.$$

A singular coefficient of this type can be assumed to affect the stability of all time-stepping schemes. The current change $s = z - b$ has no such singularizing effect though the change does result in added advection terms in the

conservation of energy equation, which we now address. See [this page](#) for more general considerations about the conservation of energy equation.

The new coordinates (x, y, s) are not orthogonal.

Recall that if $f = f(x, y, z, t)$ is a function written in the old variables and if $\tilde{f}(x, y, s, t) = f(x, y, z(x, y, s, t), t)$ is the “same” function written in the new variables, equivalently $f(x, y, z, t) = \tilde{f}(x, y, s(x, y, z, t), t)$, then

$$\frac{\partial f}{\partial x} = \frac{\partial \tilde{f}}{\partial x} + \frac{\partial \tilde{f}}{\partial s} \frac{\partial s}{\partial x} = \frac{\partial \tilde{f}}{\partial x} - \frac{\partial \tilde{f}}{\partial s} \frac{\partial b}{\partial x}.$$

Similarly,

$$\frac{\partial f}{\partial y} = \frac{\partial \tilde{f}}{\partial y} - \frac{\partial \tilde{f}}{\partial s} \frac{\partial b}{\partial y},$$

$$\frac{\partial f}{\partial t} = \frac{\partial \tilde{f}}{\partial t} - \frac{\partial \tilde{f}}{\partial s} \frac{\partial b}{\partial t}.$$

On the other hand,

$$\frac{\partial f}{\partial z} = \frac{\partial \tilde{f}}{\partial s}.$$

The following table records some important changes to formulae related to conservation of energy:

old	new
$P = \rho g(h - z)$	$P = \rho g(H - s)$
$\frac{\partial E}{\partial t}$	$\frac{\partial E}{\partial t} - \frac{\partial E}{\partial s} \frac{\partial b}{\partial t}$
∇E	$\nabla E - \frac{\partial E}{\partial s} \nabla b$
$\rho_i \left(\frac{\partial E}{\partial t} + \mathbf{U} \cdot \nabla E + w \frac{\partial E}{\partial z} \right) = \frac{k_i}{c_i} \frac{\partial^2 E}{\partial z^2} + Q$	$\rho_i \left(\frac{\partial E}{\partial t} + \mathbf{U} \cdot \nabla E + \left(w - \frac{\partial b}{\partial t} - \mathbf{U} \cdot \nabla b \right) \frac{\partial E}{\partial s} \right) = \frac{k_i}{c_i} \frac{\partial^2 E}{\partial s^2} + Q$

Note E is the ice enthalpy and T is the ice temperature (which is a function of the enthalpy; see `EnthalpyConverter`), P is the ice pressure (assumed hydrostatic), \mathbf{U} is the depth-dependent horizontal velocity, and Q is the strain-heating term.

Now the vertical velocity is computed by `StressBalance::compute_vertical_velocity(...)`. In the old coordinates (x, y, z, t) it has this formula:

$$w(z) = - \int_b^z \frac{\partial u}{\partial x}(z') + \frac{\partial v}{\partial y}(z') dz' + \frac{\partial b}{\partial t} + \mathbf{U}_b \cdot \nabla b - S.$$

Here S is the basal melt rate, positive when ice is being melted. We have used the basal kinematical equation and integrated the incompressibility statement

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0.$$

In the new coordinates we have

$$w(s) = - \int_0^s \frac{\partial u}{\partial x}(s') + \frac{\partial v}{\partial y}(s') ds' + \mathbf{U}(s) \cdot \nabla b + \frac{\partial b}{\partial t} - S.$$

(Note that the term $\mathbf{U}(s) \cdot \nabla b$ evaluates the horizontal velocity at level s and not at the base.)

Let

$$\tilde{w}(x, y, s, t) = w(s) - \frac{\partial b}{\partial t} - \mathbf{U}(s) \cdot \nabla b.$$

This quantity is the vertical velocity of the ice *relative to the location on the bed immediately below it*. In particular, $\tilde{w} = 0$ for a slab sliding down a non-moving inclined plane at constant horizontal velocity, if there is

no basal melt rate. Also, $\tilde{w}(s = 0)$ is nonzero only if there is basal melting or freeze-on, i.e. when $S \neq 0$. Within PISM, \tilde{w} is written with name `wvel,el` into an input file. Comparing the last two equations, we see how `StressBalance::compute_vertical_velocity(...)` computes \tilde{w} :

$$\tilde{w}(s) = - \int_0^s \frac{\partial u}{\partial x}(s') + \frac{\partial v}{\partial y}(s') ds' - S.$$

The conservation of energy equation is now, in the new coordinate s and newly-defined relative vertical velocity,

$$\rho_i \left(\frac{\partial E}{\partial t} + \mathbf{U} \cdot \nabla E + \tilde{w} \frac{\partial E}{\partial s} \right) = \frac{k_i}{c_i} \frac{\partial^2 E}{\partial s^2} + Q.$$

Thus it looks just like the conservation of energy equation in the original vertical velocity z . This is the form of the equation solved by `EnthalpyModel` using `enthSystemCtx::solve()`.

Under option `-o_size big`, all of these vertical velocity fields are available as fields in the output NetCDF file. The vertical velocity relative to the geoid, as a three-dimensional field, is written as the diagnostic variable `wvel`. This is the “actual” vertical velocity $w = \tilde{w} + \frac{\partial b}{\partial t} + \mathbf{U}(s) \cdot \nabla b$. Its surface value is written as `wvelsurf`, and its basal value as `wvelbase`. The relative vertical velocity \tilde{w} is written to the NetCDF output file as `wvel_rel`.

5.3 Using Schoof’s parameterized bed roughness technique in PISM

Contents

- *Using Schoof’s parameterized bed roughness technique in PISM*
 - *An explanation*
 - *Theory*
 - *Practical application, and Taylor approximation*
 - * *Convexity of P_4*

5.3.1 An explanation

If the bed elevation `c topg` is smoothed by preprocessing then we observe a reduction in the peak values of the SIA diffusivities. From such smoothing there is (generically) also a reduction in the peak magnitudes of horizontal velocities from both the SIA and SSA models. The major consequence of these reductions, through the adaptive time-stepping mechanism, is that PISM can take longer time steps and thus that it can complete model runs in shorter time.

Large peak diffusivities coming from bed roughness are located (generically) at margin locations where the ice is on, or has flowed onto, fjord-like bed topography. At coarser resolutions (e.g. 20km and up), it appears that the effect of increasing bed roughness is not as severe as at finer resolutions (e.g. 10km, 5km and finer). Of course it is true that the shallow models we use, namely the SIA and SSA models, are missing significant stress gradients at the same margin locations which have large bed slopes.

Here we are emphasizing the performance “hit” which the whole model experiences if some small part of the ice sheet is on a rough bed. That part therefore is not well-modeled anyway, compared to the majority of the ice sheet. (Switching to full Stokes or Blatter higher order models without major spatial adaptivity would probably imply a gain in the balanced stress components e and a loss of the ability to model the ice sheet at high resolution. There is a tradeoff between completeness of the continuum model and usable resolution needed to resolve the features of the real ice sheet.)

There exists a theory which addresses exactly this situation for the SIA model, and explains rigorously that one should use a smoothed bed in that model. But with an associated reduction in diffusivity. This theory explains how to improve the SIA model to handle bed roughness more correctly, because it parameterizes the effects of “higher-order” stresses which act on the ice as it flows over bed topography. Specifically it shows the way to a double performance boost for PISM:

- smoothed beds give longer time steps directly, and
- the parameterized effect of the local bed roughness is to further reduce the diffusivity, giving even longer time-steps.

5.3.2 Theory

The theory is in Christian Schoof’s (2003) *The effect of basal topography on ice sheet dynamics* [86]. His mathematical technique is to expand the Stokes equations in two levels of horizontal scales, one for the entire ice sheet (denoted $[L]$) and one for the horizontal scale (wavelength) of bed topography ($[S]$). The “inner” scaling assumes that the typical ice sheet thickness $[D]$ is small compared to $[S]$, while the “outer” scaling assumes that $[S]$ is small compared to $[L]$:

$$\nu = \frac{[D]}{[S]} \ll 1, \quad \delta = \frac{[S]}{[L]} \ll 1.$$

Specifically, there is an “inner” horizontal variable x describing the local topography on scales comparable to $[S]$ or smaller, and an “outer” horizontal variable X describing the large scale bed topography and ice sheet flow on scales larger than $[S]$.

In order to describe the Schoof scheme using PISM notation, we start by recalling the mass continuity equation which is fundamental to any shallow ice theory:

$$\frac{\partial H}{\partial t} = (M - S) - \nabla \cdot \mathbf{q}.$$

Within PISM this equation is handled by GeometryEvolution. Recall that $M - S$ is the mass balance added to the ice column per time. (It plays no further role here.) In the SIA case with zero basal sliding, the horizontal mass flux is

$$\mathbf{q} = -D_{SIA} \nabla h,$$

where $D_{SIA} \geq 0$ is given next. Thus the mass continuity equation is e diffusive. The diffusivity D_{SIA} is a function of the ice geometry and the ice flow law. In the isothermal Glen power law (power = n) case we recall

$$D_{SIA} = \Gamma H^{n+2} |\nabla h|^{n-1} \quad (5.1)$$

where $\Gamma = 2A(\rho g)^n / (n + 2)$ (c.f. details in [21]).

Consider now the “original” bed topography $b_0(x_1, x_2)$, which we assume for the moment is independent of time. (Time-independence is not actually critical, and such a restriction can be removed.) We will use x_1, x_2 to denote the horizontal model coordinates, though they are denoted x, y elsewhere in these PISM docs. Suppose a locally-smoothed bed is computed by averaging b_0 over a rectangular region of sides $2\lambda_1$ by $2\lambda_2$, namely:

$$b_s(x_1, x_2) = \oint b_0(x_1 + \xi_1, x_2 + \xi_2) d\xi_1 d\xi_2$$

where the slashed integral symbol is defined as

$$\oint F(\xi_1, \xi_2) d\xi_1 d\xi_2 = \frac{1}{(2\lambda_1)(2\lambda_2)} \int_{-\lambda_1}^{\lambda_1} \int_{-\lambda_2}^{\lambda_2} F(\xi_1, \xi_2) d\xi_1 d\xi_2.$$

Consider also the “local bed topography”

$$\tilde{b}(x_1, x_2, \xi_1, \xi_2) = b_0(x_1 + \xi_1, x_2 + \xi_2) - b_s(x_1, x_2).$$

As a function of the local coordinates ξ_1, ξ_2 , the local bed topography \tilde{b} is the amount by which the bed deviates from the “local average” $b_s(x_1, x_2)$. Generally we will use $-\lambda_1 \leq \xi_1 \leq \lambda_1$, $-\lambda_2 \leq \xi_2 \leq \lambda_2$ as the smoothing domain, but these specific ranges are not required by the formulas above. Note that the average of the local bed topography is zero by definition:

$$\oint \tilde{b}(x_1, x_2, \xi_1, \xi_2) d\xi_1 d\xi_2 = 0.$$

The result of Schoof’s scaling arguments ([86], equation (49)) is to modify the diffusivity by multiplying by a factor $0 \leq \theta \leq 1$:

$$D = \theta(h(x_1, x_2), x_1, x_2) D_{SIA}.$$

where D_{SIA} is defined by (5.1) earlier, and

$$\theta(h, x_1, x_2) = \left[\oint \left(1 - \frac{\tilde{b}(x_1, x_2, \xi_1, \xi_2)}{H} \right)^{-(n+2)/n} d\xi_1 d\xi_2 \right]^{-n} \quad (5.2)$$

Here the ice thickness and ice surface elevation are related to the smoothed bed topography, so that in PISM notation

$$H(t, x_1, x_2) = h(t, x_1, x_2) - b_s(x_1, x_2).$$

This can be treated as the definition of the ice thickness H in the above formula for θ .

The formula for θ has additional terms if there is basal sliding, but we consider only the non-sliding SIA here.

The very important fact that $0 \leq \theta \leq 1$ is proven in appendix A of [86] by a Jensen’s inequality argument. (See also the convexity argument at the bottom of this page.)

5.3.3 Practical application, and Taylor approximation

The above formulas already reflect the recommendations Schoof gives on how to apply his formulas ([86], subsection 4.2). The rest of this page is devoted to how the class `stressbalance::BedSmoother` implements a practical version of this theory, based on these recommendations plus some additional approximation.

The averages appearing in his scaling arguments are over an infinite domain, e.g.

$$f_s(x) = \lim_{R \rightarrow \infty} \frac{1}{2R} \int_{-R}^R f(\xi, x) d\xi.$$

For practical modeling use, Schoof specifically recommends averaging over some finite length scale which should be “considerably greater than the grid spacing, but much smaller than the size of the ice sheet.” Furthermore he recommends that, because of the typical aspect ratio of ice sheets, “Bed topography on much larger length scales than 10 km should then be resolved explicitly through the smoothed bed height b_s rather than the correction factor θ .” Thus in PISM we use $\lambda_1 = \lambda_2 = 5$ km as the default. Naturally the values are configurable also.

It is, of course, possible to have bed roughness of significant magnitude at essentially any wavelength. We make no claim that PISM results are good models of ice flow over arbitrary geometry; clearly the current models cannot come close to the non-shallow solution (Stokes) in such cases. Rather, the goal right now is to improve on the existing shallow models, the diffusive SIA specifically, while maintaining or increasing high-resolution performance and comprehensive model quality, which necessarily includes many other modeled physical processes like ice thermal state, basal lubrication, and so on. The desirable properties of the Schoof scheme are accepted not because the resulting model is perfect, but because we gain both a physical modeling improvement and a computational performance improvement from its use.

How do we actually compute expression (5.2) quickly? Schoof has this suggestion, which we follow: “To find θ for values of [surface elevation for which θ has not already been computed], some interpolation scheme should then be used. θ is then represented at each grid point by some locally-defined interpolating function [of the surface elevation].”

We need a “locally-defined interpolating function”. As with any approximation scheme, higher accuracy is achieved by doing “more work”, which here is an increase in memory used for storing spatially-dependent coefficients. Pade rational approximations, for example, were considered but are excluded because of the appearance of uncontrolled poles in the domain of approximation. The 4th degree Taylor polynomial is chosen here because it shares the same convexity as the rational function it approximates; this is proven below.

Use of Taylor polynomial $P_4(x)$ only requires the storage of three fields (below), but it has been demonstrated to be reasonably accurate by trying beds of various roughnesses in Matlab/Octave scripts. The derivation of the Taylor polynomial is most easily understood by starting with an abstract rational function like the one which appears in (5.2), as follows.

The fourth-order Taylor polynomial of the function $F(s) = (1 - s)^{-k}$ around $s = 0$ is

$$P_4(s) = 1 + ks + \frac{k(k+1)}{2}s^2 + \dots + \frac{k(k+1)(k+2)(k+3)}{4!}s^4,$$

so $F(s) = P_4(s) + O(s^5)$. Let

$$\omega(f, z) = \oint (1 - f(\xi)z)^{-k} d\xi$$

where f is some function and z a scalar. Then

$$\begin{aligned} \omega(f, z) &= \oint (1 - f(\xi)z)^{-k} d\xi = \oint P_4(f(\xi)z) d\xi + O((\max |f(\xi)|)^5 |z|^5) \\ &\approx 1 + kz \oint f(\xi) d\xi + \frac{k(k+1)}{2} z^2 \oint f(\xi)^2 d\xi + \dots + \frac{k(k+1)(k+2)(k+3)}{4!} z^5 \oint f(\xi)^4 d\xi. \end{aligned}$$

Now, θ can be written

$$\theta(h, x_1, x_2) = \left[\omega(\tilde{b}(x_1, x_2, \cdot, \cdot), H^{-1}) \right]^{-n}.$$

So our strategy should be clear, to approximate $\omega(\tilde{b}(x_1, x_2, \cdot, \cdot), H^{-1})$ by the Taylor polynomial as a function of H^{-1} , whose the coefficients depend on x_1, x_2 . We thereby get a rapidly-computable approximation for θ using stored coefficients which depend on x_1, x_2 . In fact, let $f(\xi) = \tilde{b}(x_1, x_2, \xi_1, \xi_2)$ for fixed x_1, x_2 , and let $z = H^{-1}$. Recall that the mean of this $f(\xi)$ is zero, so that the first-order term drops out in the above expansion of ω . We have the following approximation of θ :

$$\theta(h, x_1, x_2) \approx \left[1 + C_2(x_1, x_2)H^{-2} + C_3(x_1, x_2)H^{-3} + C_4(x_1, x_2)H^{-4} \right]^{-n} \quad (5.3)$$

where

$$C_q(x_1, x_2) = \frac{k(k+1)\dots(k+q-1)}{q!} \oint \tilde{b}(x_1, x_2, \xi_1, \xi_2)^q d\xi_1 d\xi_2$$

for $q = 2, 3, 4$ and $k = (n+2)/n$.

Note that the coefficients C_q depend only on the bed topography, and not on the ice geometry. Thus we will pre-process the original bed elevations b_0 to compute and store the fields b_s, C_2, C_3, C_4 . The computation of equation (5.3) is fast and easily-parallelized if these fields are pre-stored. The computation of the coefficients C_q and the smoothed bed b_s at the pre-processing stage is more involved, especially when done in parallel.

The parameters λ_1, λ_2 must be set, but as noted above we use a default value of 5 km based on Schoof’s recommendation. This physical distance may be less than or more than the grid spacing. In the case that the grid spacing is 1 km, for example, we see that there is a large smoothing domain in terms of the number of grid points. Generally, the ghosting width (in PETSc sense) is unbounded. Therefore move the unsmoothed topography to processor zero and do the smoothing and the coefficient-computing there. The class `pism::stressbalance::BedSmoother` implements these details.

Convexity of P_4

The approximation (5.3) given above relates to the Jensen's inequality argument used by Schoof in Appendix A of [86]. For the nonsliding case, his argument shows that because $F(s) = (1 - s)^{-k}$ is convex on $[-1, 1]$ for $k > 0$, therefore $0 \leq \theta \leq 1$.

Thus it is desirable for the approximation $P_4(z)$ to be convex on the same interval, and this is true. In fact,

$$P_4''(s) = k(k+1) \left[1 + (k+2)s + \frac{(k+2)(k+3)}{2}s^2 \right],$$

and this function turns out to be positive for all s . In fact we will show that the minimum of $P_4''(s)$ is positive. That minimum occurs where $P_4'''(s) = 0$ or $s = s_{min} = -1/(k+3)$. It is a minimum because $P_4^{(4)}(s)$ is a positive constant. And

$$P_4''(s_{min}) = \frac{k(k+1)(k+4)}{2(k+3)} > 0.$$

5.4 PISM coding guidelines

Contents

- *PISM coding guidelines*
 - *File names*
 - *Source and header files*
 - *Inline functions and methods*
 - *Including header files*
 - *Naming*
 - * *Variable*
 - * *Types and classes*
 - * *Functions and class methods*
 - *Namespaces*
 - * *Using directives and declarations*
 - *Formatting*
 - * *Boolean operators should be surrounded by spaces*
 - * *Commas and semicolons should be followed by a space*
 - * *Binary arithmetic operators should be surrounded by spaces*
 - * *Statements*
 - * *Code indentation*
 - * *Return statements*
 - * *Conditionals*
 - * *Loops*

- *Error handling*
 - * *Performing an action every time a PISM exception is thrown*
 - * *Calling C code*
 - * *Use `assert()` for sanity-checks that should not be used in optimized code*
- *Function design*
 - * *Function arguments; constness*
 - * *Method versus a function*
 - * *Virtual methods*
 - * *private versus protected versus public*

5.4.1 File names

C++ source files use the extension `.cc`. Headers use `.hh`. C code uses `.c` and `.h`.

The *basename* of a file containing the source code for a class should match the name of this class, including capitalization. For example, a class `Foo` is declared in `Foo.hh` and implemented in `Foo.cc`.

5.4.2 Source and header files

Each header file must have an include guard. Do *not* use “randomized” names of include guards.

Headers should *not* contain function and class method implementations unless these implementations *should be inlined*; see below.

5.4.3 Inline functions and methods

Implementations of inline methods should be *outside* of class declarations and in a *separate header* called `Foo_inline.hh`. This header will then be included from `Foo.hh`.

5.4.4 Including header files

Include all system headers before PISM headers. Separate system headers from PISM headers with an empty line.

Good:

```
#include <cassert>
#include <cstring>

#include "IceGrid.hh"
```

Bad:

```
#include <cstring>
#include "IceGrid.hh"
#include <cassert>
```

Whenever appropriate add comments explaining why a particular header was included.

```
#include <cstring> // strcmp
```

5.4.5 Naming

Variable

- Variable names should use lower case letters and (if necessary) digits separated by underscores, for example: `ice_thickness`.
- Do not abbreviate names of variables used in more than one scope **unless** this is needed to keep the name under 30 characters. If a function uses a variable so many times typing a long name is a hassle, create a reference with a shorter name that is only visible in this scope. (This alias will be compiled away.)
- Single-letter variable names should only be used in “small” scopes (short functions, etc.)
- If you are going to use a single-letter name, pick a letter that is easy to read (avoid `i`, `l`, and `o`).
- Names of class data members should use the `m_` prefix, for example: `m_name`.
- Names of static data members should use the `sm_` prefix.
- Global variables (which should be avoided in general) use the `g` prefix.

Types and classes

Names of types and classes use `CamelCase`.

Functions and class methods

Names of functions and class methods use the same rules as variable names, with some additions.

- If a method is used to get a property of an object that cannot be reset (example: `IceGrid::Mx()`), omit `get_` from the name.
- If a getter method has a corresponding setter method, their names should be *predictable*: `Foo::get_bar()` and `Foo::set_bar()`. In this case, *do not* omit `get_` from the name of the getter.

5.4.6 Namespaces

Everything in PISM goes into the `pism` namespace. See the source code browser for more namespaces (roughly one per sub-system).

Using directives and declarations

Do *not* import all names from a namespace with `using namespace foo`;

Do import *specific* names with `using ::foo::bar`; in `.cc` files.

5.4.7 Formatting

PISM includes a `.clang-format` file that makes it easy to re-format source to make it conform to these guidelines.

To re-format a file, commit it to the repository, then run

```
clang-format -i filename.cc
```

(Here `-i` tells clang-format to edit files “in place.” Note that editing in place is safe because you added it to the repository.)

Boolean operators should be surrounded by spaces

```
// Good
if (a == b) {
    action();
}

// Bad
if (a==b) {
    action();
}
```

Commas and semicolons should be followed by a space

```
// Good
function(a, b, c);

// Bad
function(a,b,c);
function(a,b ,c);
```

Binary arithmetic operators should be surrounded by spaces

```
// Good
f = x + y / (z * w);

// Bad
f = x+y/(z*w);
```

Statements

One statement per line.

```
// Good
x = 0;
y = x + 1;

// Bad
x = 0; y = x + 1;
```

Code indentation

- Use two spaces per indentation level.
- Do not use tabs.
- Opening braces go with the keyword (“One True Brace Style”).

Examples:

```
int function(int arg) {
    return 64;
}

for (...) {
    something();
}

class Object {
public:
    Object();
};
```

Return statements

Return statements should appear on a line of their own.

Do not surround the return value with parenthesis if you don't have to.

```
// Good
int function(int argument) {
    if (argument != 0) {
        return 64;
    }
}

// Bad
int function(int argument) {
    if (argument != 0) return(64);
}
```

Conditionals

- one space between `if` and the opening parenthesis
- no spaces between `(` and the condition (`(condition)`), not `(condition)`
- all `if` blocks should use braces (`{` and `}`) *unless* it makes the code significantly harder to read
- `if (condition)` should always be on its own line
- the `else` should be on the same line as the closing parenthesis: `} else { ...`

```
// Good
if (condition) {
    action();
}
```

```
// Bad
if (condition) action();

// Sometimes acceptable:
if (condition)
    action();
```

Loops

for, while, do {...} unless loops are formatted similarly to conditional blocks.

```
for (int k = 0; k < N; ++k) {
    action(k);
}

while (condition) {
    action();
}

do {
    action();
} unless (condition);
```

5.4.8 Error handling

First of all, PISM is written in C++, so unless we use a non-throwing new and completely avoid STL, exceptions are something we have to live with. This means that we more or less have to use exceptions to handle errors in PISM. (Mixing two error handling styles is a *bad* idea.)

So, throw an exception to signal an error; PISM has a generic runtime error exception class `pism::RuntimeError`.

To throw an exception with an informative message, use

```
throw RuntimeError::formatted(PISM_ERROR_LOCATION,
                              "format string %s", "data");
```

Error handling in a parallel program is hard. If all ranks in a communicator throw an exception, that's fine. If some do and some don't PISM will hang as soon as one rank performs a locking MPI call. I don't think we can prevent this in general, but we can handle some cases.

Use

```
ParallelSection rank0(communicator);
try {
    if (rank == 0) {
        // something that may throw
    }
} catch (...) {
    rank0.failed();
}
rank0.check();
```

to wrap code that is likely to fail on some (but not all) ranks. `rank0.failed()` prints an error message from the rank that failed and `rank0.check()` calls `MPI_Allreduce(...)` to tell other ranks in a communicator that everybody

needs to throw an exception. (`pism::ParallelSection::failed()` should be called in a `catch (...) {...}` block **only**.)

In general one should not use `catch (...)`. It *should* be used in these three cases, though:

1. With `pism::ParallelSection` (see above).
2. In callback functions passed to C libraries. (A callback is not allowed to throw, so we have to catch everything.)
3. In `main()` to catch all exceptions before terminating.

Performing an action every time a PISM exception is thrown

The class `pism::RuntimeError` allows setting a “hook” that is called by the constructor of `RuntimeError`. See the example below for a way to use it.

```
#include <stdio>

#include "error_handling.hh"

void hook(pism::RuntimeError *exception) {
    printf("throwing exception \"%s\"\n", exception->what());
}

int main(int argc, char **argv) {

    MPI_Init(&argc, &argv);

    pism::RuntimeError::set_hook(hook);

    try {
        throw pism::RuntimeError("oh no!");
    } catch (pism::RuntimeError &e) {
        printf("caught an exception \"%s\"!\n", e.what());
    }

    MPI_Finalize();

    return 0;
}
```

Calling C code

Check the return code of every C call and convert it to an exception if needed. Use macros `PISM_CHK` and `PISM_C_CHK` for this.

When calling several C function in sequence, it may make sense to wrap them in a function. Then we can check its return value and throw an exception if necessary.

```
int call_petsc() {
    // Multiple PETSc calls here, followed by CHKERRQ(ierr).
    // This way we need to convert *one* return code into an exception, not many.
    return 0;
}

// elsewhere:
int err = call_petsc(); PISM_C_CHK(err, 0, "call_petsc");
```

Use `assert()` for sanity-checks that should not be used in optimized code

The `assert` macro should be used to check pre-conditions and post-conditions that can fail *due to programming errors*.

Do not use `assert` to validate user input.

Note that *user input includes function arguments* for all functions and public members of classes accessible using Python wrappers. (Use exceptions instead.)

5.4.9 Function design

Functions are the way to *manage complexity*. They are not just for code reuse: the main benefit of creating a function is that a self-contained piece of code is easier both to **understand** and **test**.

Functions should not have side effects (if at all possible). In particular, do not use and especially do not *modify* “global” objects. If a function needs to modify an existing field “in place”, pass a reference to that field as an argument and document this argument as an “input” or an “input/output” argument.

Function arguments; constness

Pass C++ class instances by const reference *unless* an instance is modified in place. This makes it easier to recognize *input* (read-only) and *output* arguments.

Do **not** use `const` when passing C types: `f()` and `g()` below are equivalent.

```
double f(const double x) {
    return x*x;
}

double g(double x) {
    return x*x;
}
```

Method versus a function

Do **not** implement something as a class method if the same functionality can be implemented as a standalone function. Turn a class method into a standalone function if you notice that it uses the *public* class interface only.

Virtual methods

- Do not make a method virtual unless you have to.
- Public methods should not be virtual (create “non-virtual interfaces”)
- **Never** add `__attribute__((noreturn))` to a virtual class method.

private versus protected versus public

Most data members and class methods should be `private`.

Make it `protected` if it should be accessible from a derived class.

Make it `public` only if it is a part of the interface.

5.5 How do I...?

Contents

- *How do I...?*
 - *Creating and using configuration flags and parameters*
 - *Creating and using additional variables*
 - * *Creating IceModelVec instances*
 - * *Reading data from a file*
 - * *Writing data to a file*
 - * *Creating IceModelVec instances that are data members of a class*
 - * *Reading scalar forcing data*
 - * *Reading 2D forcing fields*
 - * *Adding a new “diagnostic” quantity to an atmosphere model*
 - *Create the class implementing the diagnostic*
 - *“Register” the new diagnostic.*

5.5.1 Creating and using configuration flags and parameters

- Edit `src/pism_config.cdl`. Each flag or parameter is stored as a NetCDF attribute and should have a corresponding “_doc” attribute describing its meaning.

```
pism_config:standard_gravity = 9.81;
pism_config:standard_gravity_doc = "m s-2; acceleration due to gravity on Earth geoid";
```

- One can access these parameters using the `Config` class. `IceModel`¹ has an instance of this class as a data member `m_config`, so no additional code is necessary to initialize the configuration database.

To use a parameter, do

```
double g = m_config->get_double("standard_gravity");
```

To use a flag, do

```
bool compute_age = config->get_boolean("do_age");
```

Note:

- It is a good idea to avoid calling `m_config->get_double()` and `m_config->get_boolean()` from within loops: looking up a parameter by its name is slow.
- Please see [Configuration parameters](#) for a list of flags and parameters currently used in PISM.

¹ And all classes derived from `Component`.

5.5.2 Creating and using additional variables

Creating IceModelVec instances

PISM uses the following classes to manage 2D and 3D fields, their I/O and metadata:

IceModelVec2S	scalar 2D fields
IceModelVec2V	vector 2D fields such as horizontal velocities; corresponds to 2 NetCDF variables
IceModelVec2Int	2D masks, such as the grounded/floating mask
IceModelVec2T	2D time-dependent fields (used to read and store forcing data)
IceModelVec3	scalar 3D fields (within the ice)

Please see the documentation of these classes for more info. The base class `IceModelVec` is a virtual class, so code should use the above derived classes. Only the derived classes have `create()` methods, in particular.

To create a scalar field, for example, one needs to create an instance of one of the classes listed above and then call “create” to allocate it.

```
// land ice thickness
ice_thickness.create(grid, "thk", WITH_GHOSTS, 2);
ice_thickness.set_attrs("model_state", "land ice thickness",
                       "m", "land_ice_thickness");
ice_thickness.metadata().set_double("valid_min", 0.0);
```

Here `grid` is an `IceGrid` instance, `thk` is the name of the NetCDF variable, `WITH_GHOSTS` means that storage for “ghost” (“halo”) points will be allocated, and “2” is the number of ghosts (in other words: needed stencil width).

The `IceModelVec` destructor takes care of undoing all that’s done by the `create()` call. Therefore you don’t need to explicitly de-allocate variables unless you dynamically created the `IceModelVec` (or derived) instance using the C++ “new” operator. (In which case “delete” should be used.)

The `IceModelVec::set_attrs()` call sets commonly used NetCDF variable attributes seen in PISM output files:

<code>pism_intent</code>	the only important case is “model_state”, see below
<code>long_name</code>	the (descriptive) long name used for plotting, etc (a free-form string)
<code>units</code>	units used <i>in the code</i> . Does not have to match units in a file
<code>standard_name</code>	CF standard name, if defined, or an empty string.

The third call above `ice_thickness.metadata()` allows accessing variable metadata and adding arbitrary named attributes. See `VariableMetadata` for details.

The CF convention covers some attribute semantics, including `valid_min` in this example.

PISM will automatically convert units from ones present in an input file into internal units defines by the `set_attrs()` call (see above).

If you want PISM to save data in units other than internal ones, first set these “glaciological” units:

```
ice_thickness.metadata().set_string("glaciological_units", "km");
```

Reading data from a file

There are at least three cases of “reading data from a file”:

- reading a field stored in an input file on a grid matching the one used by the current run (restarting a run) and

- reading a field stored in an input file on a different grid, interpolating onto the current grid (bootstrapping).
- reading a field stored in a file **other** than the input file using interpolation (assuming that grids are compatible but not identical)

FIXME

Writing data to a file

To write a field stored in an `IceModelVec` to an already “prepared” file, just call

```
precip.write(filename);
```

The file referred to by “filename” here has to have the time “time” dimension created, that is, it must be prepared (other dimensions are created automatically, if needed). No action is needed to be able to write to an output (“-o”) file, a snapshot file or the like; `IceModel` has already prepared it.

If you do need to “prepare” a file, do:

```
PIO nc(grid.com, grid.config.get_string("output_format"));

std::string time_name = config.get_string("time_dimension_name");
nc.open(filename, PISM_WRITE); // append == false
nc.def_time(time_name, grid.time->calendar(),
            grid.time->CF_units_string());
nc.append_time(time_name, grid.time->current());
nc.close();
```

When a file is opened with the `PISM_WRITE` mode, PISM checks if this file is present and moves it aside if it is; to append to an existing file, use

```
nc.open(filename, PISM_WRITE, true); // append == true
```

A newly-created file is “empty” and contains no records. The `nc.append_time()` call creates a record corresponding to a particular model year.

Creating `IceModelVec` instances that are data members of a class

To add a new variable to `IceModel`, allocate it in the `createVecs()` method.

If `pism_intent` is set to `model_state` and a variable is added to the “variables” dictionary (see `PISMVars`), `IceModel` will automatically read this variable from a file it is re-starting from and will always write it to an output, snapshot and backup files.

```
variables.add(ice_thickness);
```

Reading scalar forcing data

PISM uses instances of the `Timeseries` class to read scalar forcing data; please see `PA_delta_T` for an example.

The following snippet illustrates creating a `Timeseries` instance and reading data from a file.

```
std::string offset_name = "delta_T";

offset = new Timeseries(&grid, offset_name, config.get_string("time_dimension_name"));
offset->set_string("units", "Kelvin");
```

```

offset->set_dimension_units(grid.time->units_string(), "");

verbPrintf(2, g.com,
           "  reading %s data from forcing file %s...\n",
           offset->short_name.c_str(), filename.c_str());

PIO nc(g.com, "netcdf3", grid.get_unit_system());
nc.open(filename, PISM_NOWRITE);
{
    offset->read(nc, grid.time);
}
nc.close();

// use offset

delete offset; // when done

```

To use `offset`, call

```
double data = (*offset)(time);
```

to get the value corresponding to the time `time` in seconds. The value returned will be computed using linear interpolation.

Reading 2D forcing fields

PISM uses instances of the `IceModelVec2T` class to read 2D forcing fields that vary in time; please see `PSDirectForcing` for an example.

The following snippet from `PSDirectForcing::init()` illustrates creating an `IceModelVec2T` object and reading data from a file.

```

IceModelVec2T temperature;
temperature.set_n_records((unsigned int) config.get_double("climate_forcing_buffer_size"));
temperature.create(grid, "artm", WITHOUT_GHOSTS);
temperature.set_attrs("climate_forcing",
                    "temperature of the ice at the ice surface but below firn processes",
                    "Kelvin", "");
temperature.init(filename);

```

@section using_vars Using fields managed by `IceModel` in a surface model to implement a parameterization

Please see `PA_SeaRISE_Greenland::init()` and `PA_SeaRISE_Greenland::update()` for an example.

@section writing_components Managing I/O in a `PISMComponent` derived class

A PISM component needs to implement the following I/O methods:

- `init()`. It is not an I/O method per se, but most PISM components read their input fields there; see `PA_SeaRISE_Greenland::init()`.
- `add_vars_to_output()`, which adds variable names to the list of fields that need to be written. See `PSTemperatureIndex::add_vars_to_output_impl()` for an example.
- `define_variables()`, which defines variables. (See `PSTemperatureIndex::define_variables()`.)
- `write_variables()`, which writes data; see `PSTemperatureIndex::write_variables()`.

Why are all these methods needed? In PISM we separate defining and writing NetCDF variables because defining all the NetCDF variables before writing data is a lot faster than defining a variable, writing it, defining the second variable, etc. (See [The NetCDF Users' Guide](#) for a technical explanation.)

Within `IceModel` the following steps are done to write 2D and 3D fields to an output file:

- Assemble the list of variables to be written (see `IceModel::set_output_size()`); calls `add_vars_to_output()`
- Create a NetCDF file
- Define all the variables in the file (see `IceModel::write_variables()`); calls `define_variables()`
- Write all the variables to the file (same method); calls `write_variables()`.

Adding a new “diagnostic” quantity to an atmosphere model

To add a new “diagnostic” quantity (i.e. a 2D or 3D field that needs to be saved to an output file but is not permanently stored in memory and is computed on demand), do the following.

Create the class implementing the diagnostic

Assuming that `PA_foo` is the atmosphere model we’re working with and “bar” is the name of the new quantity, we need to add this code

```
class PA_foo_bar : public PISMDiag<PA_foo>
{
public:
    PA_foo_bar(PA_foo *m, IceGrid &g, PISMVars &my_vars);
    virtual IceModelVec::Ptr compute();
};
```

to a header (`.hh`) file and implement it in a `.cc` file.

See `IceModel_diagnostics.cc` for examples. Generally speaking, in every class implementing a “diagnostic” quantity

- the constructor sets metadata
- you have access to a data member “var” of an atmosphere model as “model->var”; you might need to add

```
friend class PA_foo_bar;
```

to the definition of `PA_foo` if you need to access a private data member (see the definition of `IceModel` for one example).

- when working with atmosphere models and other classes derived from `PISMComponent`, you can access the config database as “model->config”.
- the `PISMDiagnostic::compute()` method allocates memory and performs the computation.
- to use a field managed by `IceModel`, use “variables”:

```
const IceModelVec2S *surface = variables.get_2d_scalar("surface_altitude");
```

- the **caller** of the `PISMDiagnostic::compute()` method has to de-allocate the field returned by `PISMDiagnostic::compute()`

Note that in almost every (current) implementation of `PISMDiagnostic::compute()` you see

```
IceModelVec::Ptr ...::compute() {
    const PetscScalar fillval = -0.01;

    <...>

    // 1
    IceModelVec2S::Ptr result(new IceModelVec2S);
    result->create(grid, "hardav", WITHOUT_GHOSTS);
    result->metadata(0) = m_vars[0];

    <...>

    // 2
    return result;
}
```

The block marked “1” allocates a 2D field and copies metadata stored in `m_vars[0]`, while the block marked “2” returns a pointer to a 2D field, which gets cast to a pointer to a “generic” field.

This allows us to have the same interface for both 2D and 3D diagnostics.

“Register” the new diagnostic.

To make the new diagnostic field available (i.e. to be able to use the new `PA_foo_bar` class), implement `PA_foo::diagnostics_impl()` or `PA_foo::ts_diagnostics_impl()`.

```
std::map<std::string, Diagnostic::Ptr> PA_foo_bar::diagnostics_impl() const {
    return {{"name", Diagnostic::Ptr(new PA_foo_bar(this))}};
}
```

Note that if you are implementing this method in a “modifier” of a surface model, you need to remember to call

```
input_model->diagnostics();
```

5.6 Using IceModelVec2T to load space- and time-dependent forcing fields

Contents

- *Using IceModelVec2T to load space- and time-dependent forcing fields*
 - *Rationale*
 - *Setup*
 - * *Allocating storage*
 - * *Initialization*
 - *Actual use*
 - * *Extracting time-series of a field at a particular grid point*
 - *Initializing point-wise access to time-series*

- *Accessing point-wise time-series*
- *Putting it all together to implement an “ocean model”*
 - * *Class declaration*
 - * *Class definition*

These notes document `IceModelVec2T` and implement a minimal `pism::ocean::OceanModel` subclass using `IceModelVec2T` to load and store space-and-time-dependent sub-shelf melt rates.

5.6.1 Rationale

The class `IceModelVec2T` was created to allow using time-dependent climate forcing fields. Unlike three-dimensional fields used by PISM (ice enthalpy, bedrock temperature, velocity components), climate forcing fields

- are not needed “all at once:” only the portion covering the current time step has to be available at a given time, and
- may cover very long time periods, making it impossible to hold the whole data set in RAM.
- may be *periodic* with a given period (whole years)

In addition to this, PDD (temperature index) surface process models use *time-series* of near-surface air temperature and precipitation at each grid point, and `IceModelVec2T` supports this use.

`IceModelVec2T` is derived from `IceModelVec2S` and can be used as a “regular” scalar field in PISM *without* ghosts.

5.6.2 Setup

Allocating storage

`IceModelVec2T` contains a “buffer” storing a number of temporal records of a 2D field. To allocate an instance you need to specify the maximum number of records it can hold. (Climate forcing code in PISM uses the configuration parameter `climate_forcing.buffer_size`.)

```
unsigned int N = 100; // hold at most N records
IceModelVec2T field;
field.set_n_records(N);
field.create(grid, "netcdf_variable_name");
field.set_attrs("intent", "long name", "units", "CF_standard_name");
```

When using periodic forcing data the buffer has to be big enough to contain *all* the records covering the period. (Implemented climate forcing code gets the number of records from an input file.)

The call `field.create(...)` allocates storage.

The call `field.set_attrs()` sets some metadata. PISM will convert data read from a file from the units stored in the NetCDF variable attribute “units” into internal units given in this call. The CF standard name (if present) can be used to find a variable in an input file.

Initialization

To specify the file to read data from, call `field.init(filename)`:

```
field.init("input_file.nc");
```

In the “regular” (non-periodic) case this reads the times stored in the file, preparing to read array data when `update()` is called. In the periodic case, on the other hand, all records are read in right away.

To set an `IceModelVec2T` to a constant, call

```
field.init_constant(value);
```

This call initializes a “fake” time line, turning subsequent `update()` calls into no-ops.

5.6.3 Actual use

To make sure that records covering a time step from t to $t + dt$ are available, call

```
field.update(t, dt);
```

This reads some records from a file while avoiding unnecessary I/O.

If `update(...)` calls go in sequence every records should be read only once, even if some time steps overlap previous ones.

Records stored in an `IceModelVec2T` are interpreted as *piece-wise constant in time*. Depending on the application this may require limiting time steps taken by PISM so that no time step spans more than one of the temporal intervals a field is defined on.

To get the length of such a time step PISM can take at time t , call

```
double max_timestep = field.max_timestep(t);
```

To get a “snapshot” at a given time, call

```
field.interp(t);
```

Sometimes (e.g. for precipitation) it makes sense to average over a time step. In this case, use `average(t, dt)`.

```
field.average(t, dt);
```

This call uses the rectangle rule to approximate the average. The interval $(t, t + dt)$ is split into N sub-intervals, where N depends on the length of the time step. The number of sub-intervals per model year can be set by calling

```
field.set_n_evaluations_per_year(N_per_year);
```

The configuration parameter `climate_forcing.evaluations_per_year` provides the default.

Extracting time-series of a field at a particular grid point

Initializing point-wise access to time-series

Given an array `ts` of times in the interval $(t, t + dt)$, the call

```
field.init_interpolation(ts);
```

will prepare `field` for extracting time-series at grid point (i, j) .

`IceModelVec2T` will use constant extrapolation if some times in the array `ts` are outside the interval given to the last call `update(t, dt)`.

Accessing point-wise time-series

Just like all other classes derived from `IceModelVec`, `IceModelVec2T` requires that `begin_access()` is called before point-wise access and `end_access()` after. (Use `IceModelVec::AccessList` to avoid doing this “by hand.”)

```
IceModelVec::AccessList list(field);

for (Points p(grid); p; p.next()) {
    const int i = p.i(), j = p.j();

    std::vector<double> values;

    field.interp(i, j, values);

    // use time-series "values" at times "ts"
}
```

5.6.4 Putting it all together to implement an “ocean model”

Class declaration

```
#ifndef _PO_EXAMPLE_H_
#define _PO_EXAMPLE_H_

#include "coupler/PISMOcean.hh"
#include "base/util/iceModelVec2T.hh"

namespace pism {
namespace ocean {
    ///! \brief An example ocean model illustrating the use of `IceModelVec2T`.
    class Example : public OceanModel {
    public:
        Example(IceGrid::ConstPtr g);
        virtual ~Example();
    protected:
        virtual MaxTimestep max_timestep_impl(double t) const;
        virtual void update_impl(double my_t, double my_dt);
        virtual void init_impl();
        virtual void sea_level_elevation_impl(double &result) const;
        virtual void shelf_base_temperature_impl(IceModelVec2S &result) const;
        virtual void shelf_base_mass_flux_impl(IceModelVec2S &result) const;
    protected:
        IceModelVec2T m_shelf_melt_rate;
    };

} // end of namespace ocean
} // end of namespace pism

#endif /* _PO_EXAMPLE_H_ */
```

Class definition

```
#include "Example.hh"
```



```

#include "base/util/PISMConfigInterface.hh"
#include "base/util/IceGrid.hh"
#include "base/util/pism_options.hh"
#include "base/util/MaxTimestep.hh"

namespace pism {
namespace ocean {

Example::Example(IceGrid::ConstPtr g)
: OceanModel(g) {

    // assume that climate_forcing.buffer_size is big enough
    m_shelf_melt_rate.set_n_records(m_config->get_double("climate_forcing.buffer_size"));
    m_shelf_melt_rate.create(m_grid, "shelf_base_melt_rate");
    m_shelf_melt_rate.set_attrs("internal", "shelf base melt rate", "m / second", "");
}

Example::~Example() {
    // empty
}

void Example::update_impl(double t, double dt) {
    m_t = t;
    m_dt = dt;

    m_shelf_melt_rate.update(t, dt);

    // Use mid-point of the interval. (We restricted the time step, so
    // the choice of the point within the time step does not matter.)
    m_shelf_melt_rate.interp(t + 0.5 * dt);

    // Alternatively one could call. This does not require a time step restriction.
    // m_shelf_melt_rate.average(t, dt);
}

void Example::init_impl() {
    m_log->message(2, "* Initializing the example ocean model...\n");

    options::String input_file("-ocean_example_file", "Shelf melt rate input file.");

    if (input_file.is_set()) {
        m_log->message(2, " Reading shelf base melt rate from %s...\n",
            input_file->c_str());

        m_shelf_melt_rate.init(input_file, 0.0, 0.0);
    } else {
        m_shelf_melt_rate.init_constant(0.0);
    }
}

MaxTimestep Example::max_timestep_impl(double t) const {
    // Assume that temporal variations in the melt rate have to be resolved.
    return m_shelf_melt_rate.max_timestep(t);

    // Use this to disable the time step restriction
    return MaxTimestep("example ocean model");
}

```

```
void Example::shelf_base_temperature_impl(IceModelVec2S &result) const {
    // PISM uses MKS. This is obviously wrong, but this just an example.
    result.set(273.15);
}

void Example::sea_level_elevation_impl(double &result) const {
    // Also wrong.
    result = 0.0;
}

//! @brief Computes mass flux in [kg m-2 s-1], from assumption that
//! basal heat flux rate converts to mass flux.
void Example::shelf_base_mass_flux_impl(IceModelVec2S &result) const {
    result.copy_from(m_shelf_melt_rate);
}

} // end of namespace ocean
} // end of namespace pism
```

All ocean models need to provide implementatons (`_impl(...)` methods) corresponding to the public API of `pism::ocean::OceanModel`. (See `src/coupler/PISM0cean.hh` and note that some have default implementations.)

AUTHORSHIP

Copyright © 2004 – 2017 the PISM authors.

This file is part of PISM. PISM is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 3 of the License, or (at your option) any later version. PISM is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details. You should have received a copy of the GNU General Public License along with PISM. If not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

PISM is a joint project between developers in the ice sheet modeling group at the University of Alaska (UAF), developers at the Potsdam Institute for Climate Impact Research (PIK), and several additional developers listed here.

Name and affiliation	Area of contribution
Torsten Albrecht (PIK)	ice shelf physics and numerics
Andy Aschwanden (UAF)	scripts, visualization, thermodynamics, SeaRISE-Greenland
Jed Brown (ANL)	source code original author, SSA numerics, PETSc underpinnings
Ed Bueler (UAF)	principal investigator, verification, earth deformation, SIA numerics, thermodynamics, documentation
Dani DellaGiustina (UAF)	regional tools and modeling
Johannes Feldman (PIK)	marine ice sheet processes
Elizabeth Fischer (GFDL)	coupling design
Marijke Habermann (UAF)	inversion
Marianne Haseloff (UBC)	ice streams: physics and numerics
Regine Hock (UAF)	surface mass and energy balance
Constantine Khroulev (UAF)	source code primary author, input/output, software design, climate couplers, parallelization, testing, user support, most documentation, most bug fixes, regional tools, ...
Anders Levermann (PIK)	calving, ice shelf processes
Craig Lingle (UAF)	original SIA model, earth deformation
Maria Martin (PIK)	SeaRISE-Antarctica, Antarctica processes
Mattias Mengel (PIK)	marine ice sheet processes
David Maxwell (UAF)	inversion, SSA finite elements, python bindings
Ward van Pelt (IMAU)	hydrology analysis and design
Julien Seguinot (INK)	bug fixes, temperature index model
Ricarda Winkelmann (PIK)	Antarctica processes, coupling, and modeling
Florian Ziemen (UAF)	bug fixes, sliding

Email the **highlighted** UAF developers at uaf-pism@alaska.edu.

BIBLIOGRAPHY

- [1] R. S. Fausto, A. P. Ahlstrom, D. Van As, C. E. Boggild, and S. J. Johnsen. A new present-day temperature parameterization for Greenland. *J. Glaciol.*, 55(189):95–105, 2009.
- [2] P. Huybrechts. Sea-level changes at the LGM from ice-dynamic reconstructions of the Greenland and Antarctic ice sheets during the glacial cycles. *Quat. Sci. Rev.*, 21:203–231, 2002.
- [3] C. Ritz. EISMINT Intercomparison Experiment: Comparison of existing Greenland models. 1997. URL: <http://homepages.vub.ac.be/~phuybrec/eismint/greenland.html>.
- [4] S. J. Johnsen, D. Dahl-Jensen, W. Dansgaard, and N. Gundestrup. Greenland paleotemperatures derived from GRIP bore hole temperature and ice core isotope profiles. *Tellus*, 47B:624–629, 1995.
- [5] J. Imbrie and eight others. The orbital theory of Pleistocene climate: Support from a revised chronology of the marine delta-O-18 record. In *Milankovitch and Climate: Understanding the Response to Astronomical Forcing*, pages 269–305. D. Reidel, 1984.
- [6] M. A. Martin, R. Winkelmann, M. Haseloff, T. Albrecht, E. Bueler, C. Khroulev, and A. Levermann. The Potsdam Parallel Ice Sheet Model (PISM-PIK) –Part 2: Dynamic equilibrium simulation of the Antarctic ice sheet. *The Cryosphere*, 5:727–740, 2011.
- [7] A Beckmann and H Goosse. A parameterization of ice shelf-ocean interaction for climate models. *Ocean Modelling*, 5(2):157–170, 2003. URL: <http://linkinghub.elsevier.com/retrieve/pii/S1463500302000197>.
- [8] David M Holland and Adrian Jenkins. Modeling thermodynamic ice-ocean interactions at the base of an ice shelf. *Journal of Physical Oceanography*, 29(8):1787–1800, 1999.
- [9] Hartmut H. Hellmer, Stanley S. Jacobs, and Adrian Jenkins. *Oceanic erosion of a floating Antarctic glacier in the Amundsen Sea*. American Geophysical Union, 1998.
- [10] Reinhard Calov and Ralf Greve. Correspondence: A semi-analytical solution for the positive degree-day model with stochastic temperature variations. *J. Glaciol.*, 51(172):173–175, 2005.
- [11] I. Rogozhina and D. Rau. Vital role of daily temperature variability in surface mass balance parameterizations of the greenland ice sheet. *The Cryosphere*, 8:575–585, 2014. doi:10.5194/tc-8-575-2014.
- [12] J. Seguinot. Spatial and seasonal effects of temperature variability in a positive degree day surface melt model. *J. Glaciol.*, 59(218):1202–1204, 2013. doi:10.3189/2013JoG13J081.
- [13] J. Seguinot and I. Rogozhina. Daily temperature variability predetermined by thermal conditions over ice sheet surfaces. *J. Glaciol.*, 2014. doi:10.3189/2014JoG14J036.
- [14] A. Payne and others. Results from the EISMINT model intercomparison: the effects of thermomechanical coupling. *J. Glaciol.*, 153:227–238, 2000.
- [15] R. Hock. Glacier melt: a review of processes and their modelling. *Prog. Phys. Geog.*, 29(3):362–391, 2005.

- [16] G. Cogley and others. Glossary of Mass-Balance and Related Terms. IACS Working Group on Mass-balance Terminology and Methods, Draft 3, 10 July, 2009. URL: http://www.cryosphericsscience.org/mass_balance_glossary/massbalanceglossary.draft3.pdf.
- [17] E. Bueler and J. Brown. Shallow shelf approximation as a “sliding law” in a thermodynamically coupled ice sheet model. *J. Geophys. Res.*, 2009. F03008, doi:10.1029/2008JF001179.
- [18] E. Bueler, J. Brown, and C. Lingle. Exact solutions to the thermomechanically coupled shallow ice approximation: effective tools for verification. *J. Glaciol.*, 53(182):499–516, 2007.
- [19] A. C. Fowler. *Mathematical Models in the Applied Sciences*. Cambridge Univ. Press, 1997.
- [20] I. Joughin, M. Fahnestock, S. Ekholm, and R. Kwok. Balance velocities of the Greenland ice sheet. *Geophysical Research Letters*, 24(23):3045–3048, 1997.
- [21] E. Bueler, C. S. Lingle, J. A. Kallen-Brown, D. N. Covey, and L. N. Bowman. Exact solutions and numerical verification for isothermal ice sheets. *J. Glaciol.*, 51(173):291–306, 2005.
- [22] P. Huybrechts and others. The EISMINT benchmarks for testing ice-sheet models. *Ann. Glaciol.*, 23:1–12, 1996.
- [23] A. Aschwanden, E. Bueler, C. Khroulev, and H. Blatter. An enthalpy formulation for glaciers and ice sheets. *J. Glaciol.*, 58(209):441–457, 2012. doi:10.3189/2012JoG11J088.
- [24] R. Greve. A continuum–mechanical formulation for shallow polythermal ice sheets. *Phil. Trans. Royal Soc. London A*, 355:921–974, 1997.
- [25] R. Winkelmann, M. A. Martin, M. Haseloff, T. Albrecht, E. Bueler, C. Khroulev, and A. Levermann. The Potsdam Parallel Ice Sheet Model (PISM-PIK) Part 1: Model description. *The Cryosphere*, 5:715–726, 2011.
- [26] H. Blatter. Velocity and stress fields in grounded glaciers: a simple algorithm for including deviatoric stress gradients. *J. Glaciol.*, 41(138):333–344, 1995.
- [27] Frank Pattyn. A new three-dimensional higher-order thermomechanical ice sheet model: Basic sensitivity, ice stream development, and ice flow across subglacial lakes. *J. Geophys. Res.*, 2003. doi:10.1029/2002JB002329.
- [28] C. Schoof. Coulomb friction and other sliding laws in a higher order glacier flow model. *Math. Models Methods Appl. Sci. (M3AS)*, 20:157–189, 2010. doi:10.1142/S0218202510004180.
- [29] K. Hutter. *Theoretical Glaciology*. D. Reidel, 1983.
- [30] M. Weis, R. Greve, and K. Hutter. Theory of shallow ice shelves. *Continuum Mech. Thermodyn.*, 11(1):15–50, 1999.
- [31] L. W. Morland. Unconfined ice-shelf flow. In C. J. van der Veen and J. Oerlemans, editors, *Dynamics of the West Antarctic ice sheet*, 99–116. Kluwer Academic Publishers, 1987.
- [32] D. R. MacAyeal. Large-scale ice flow over a viscous basal sediment: theory and application to ice stream B, Antarctica. *J. Geophys. Res.*, 94(B4):4071–4087, 1989.
- [33] C. Schoof. A variational approach to ice stream flow. *J. Fluid Mech.*, 556:227–251, 2006.
- [34] W. S. B. Paterson. *The Physics of Glaciers*. Pergamon, 3rd edition, 1994.
- [35] P. Huybrechts and J. de Wolde. The dynamic response of the Greenland and Antarctic ice sheets to multiple-century climatic warming. *J. Climate*, 12:2169–2188, 1999.
- [36] A. J. Payne and D. J. Baldwin. Analysis of ice–flow instabilities identified in the EISMINT intercomparison exercise. *Ann. Glaciol.*, 30:204–210, 2000.
- [37] Andrew C. Fowler. Modelling the flow of glaciers and ice sheets. In Brian Straughan and others, editors, *Continuum Mechanics and Applications in Geophysics and the Environment*, 201–221. Springer, 2001.
- [38] L. W. Morland and R. Zainuddin. Plane and radial ice-shelf flow with prescribed temperature profile. In C. J. van der Veen and J. Oerlemans, editors, *Dynamics of the West Antarctic ice sheet*, 117–140. Kluwer Academic Publishers, 1987.

- [39] M. Truffer and K. Echelmeyer. Of isbrae and ice streams. *Ann. Glaciol.*, 36(1):66–72, 2003.
- [40] I. Joughin, M. Fahnestock, D. MacAyeal, J. L. Bamber, and P. Gogineni. Observation and analysis of ice flow in the largest Greenland ice stream. *J. Geophys. Res.*, 106(D24):34021–34034, 2001.
- [41] J. L. Bamber, D. G. Vaughan, and I. Joughin. Widespread complex flow in the interior of the Antarctic ice sheet. *Science*, 287:1248–1250, 2000.
- [42] N. Golledge, C. Fogwill, A. Mackintosh, and K. Buckley. Dynamics of the Last Glacial Maximum Antarctic ice-sheet and its response to ocean forcing. *Proc. Nat. Acad. Sci.*, 109(40):16052–16056, 2012. doi:10.1073/pnas.1205385109.
- [43] D. Pollard and R. M. DeConto. A coupled ice-sheet/ice-shelf/sediment model applied to a marine-margin flow-line: Forced and unforced variations. In M. J. Hambrey and others, editors, *Glacial Sedimentary Processes and Products*. Blackwell Publishing Ltd., 2007.
- [44] C. Schoof and R. Hindmarsh. Thin-film flows with wall slip: an asymptotic analysis of higher order glacier flow models. *Quart. J. Mech. Appl. Math.*, 63(1):73–114, 2010. doi:10.1093/qjmam/hbp025.
- [45] R. Greve and H. Blatter. *Dynamics of Ice Sheets and Glaciers*. Advances in Geophysical and Environmental Mechanics and Mathematics. Springer, 2009.
- [46] Jed Brown, Barry Smith, and Aron Ahmadi. Achieving textbook multigrid efficiency for hydrostatic ice sheet flow. *SIAM J. Sci. Comp.*, 35(2):B359–B375, 2013.
- [47] I. Joughin, W. Abdalati, and M. Fahnestock. Large fluctuations in speed on Greenland’s Jakobshavn Isbræ~glacier. *Nature*, 432(23):608–610, 2004.
- [48] D. M. Holland, R. H. Thomas, B. de Young, M. H. Ribergaard, and B. Lyberth. Acceleration of Jakobshavn Isbræ~triggered by warm subsurface ocean waters. *Nature Geoscience*, 1:659–664, 2008. doi:10.1038/ngeo316.
- [49] M. Lüthi, M. Fahnestock, and M. Truffer. Correspondence: calving icebergs indicate a thick layer of temperate ice at the base of Jakobshavn Isbrae, Greenland. *J. Glaciol.*, 55(191):563–566, 2009.
- [50] D. DellaGiustina. Regional modeling of Greenland’s outlet glaciers with the Parallel Ice Sheet Model. Master’s thesis, University of Alaska, Fairbanks, 2011. M.S. Computational Physics.
- [51] R. Bindshadler and twenty-seven others. Ice-sheet model sensitivities to environmental forcing and their use in projecting future sea-level (The SeaRISE Project). *J. Glaciol.*, 59(214):195–224, 2013.
- [52] J. Ettema, M. R. van den Broeke, E. van Meijgaard, W. J. van de Berg, J. L. Bamber, J. E. Box, and R. C. Bales. Higher surface mass balance of the Greenland ice sheet revealed by high-resolution climate modeling. *Geophys. Res. Lett.*, 2009. doi:10.1029/2009GL038110.
- [53] I. Joughin, B. E. Smith, I. M. Howat, T. Scambos, and T. Moon. Greenland flow variability from ice-sheet-wide velocity mapping. *J. Glaciol.*, 56(197):415–430, 2010.
- [54] W. J. J. van Pelt and J. Oerlemans. Numerical simulations of cyclic behaviour in the parallel ice sheet model (pism). *Journal of Glaciology*, 58(208):347–360, 2012. URL: <http://www.igsoc.org/journal/58/208/t11J217.pdf>, doi:10.3189/2012JoG11J217.
- [55] A. Aschwanden, G. Adalgeirsdóttir, and C. Khroulev. Hindcasting to measure ice sheet model sensitivity to initial states. *The Cryosphere*, 7:1083–1093, 2013. doi:10.5194/tc-7-1083-2013.
- [56] M. Habermann, M. Truffer, and D. Maxwell. Changing basal conditions during the speed-up of Jakobshavn Isbrae, Greenland. *The Cryosphere*, 7(6):1679–1692, 2013. doi:10.5194/tc-7-1679-2013.
- [57] D. R. MacAyeal, V. Rommelaere, Ph. Huybrechts, C.L. Hulbe, J. Determann, and C. Ritz. An ice-shelf model test based on the Ross ice shelf. *Ann. Glaciol.*, 23:46–51, 1996.
- [58] S. De La Chapelle, O. Castelnau, V. Lipenkov, and P. Duval. Dynamic recrystallization and texture development in ice as revealed by the study of deep cores in Antarctica and Greenland. *J. Geophys. Res.*, 103(B3):5091–5105, 1998.

- [59] V. Lipenkov, N. I. Barkov, P. Duval, and P. Pimienta. Crystalline texture of the 2083 m ice core at Vostok Station, Antarctica. *J. Glaciol.*, 35(1):392–398, 1989.
- [60] Ralf Greve. Application of a polythermal three-dimensional ice sheet model to the Greenland ice sheet: Response to steady-state and transient climate scenarios. *J. Climate*, 10(5):901–918, 1997.
- [61] A. Aschwanden and H. Blatter. Mathematical modeling and numerical simulation of polythermal glaciers. *J. Geophys. Res.*, 2009. F01027. doi:10.1029/2008JF001028.
- [62] W. S. B. Paterson and W. F. Budd. Flow parameters for ice sheet modeling. *Cold Reg. Sci. Technol.*, 6(2):175–177, 1982.
- [63] D. L. Goldsby and D. L. Kohlstedt. Superplastic deformation of ice: experimental observations. *J. Geophys. Res.*, 106(M6):11017–11030, 2001.
- [64] L. A. Lliboutry and P. Duval. Various isotropic and anisotropic ices found in glaciers and polar ice caps and their corresponding rheologies. *Annales Geophys.*, 3:207–224, 1985.
- [65] E. Bueler and J. Brown. On exact solutions and numerics for cold, shallow, and thermocoupled ice sheets. preprint \texttt{arXiv:physics/0610106}, 2006.
- [66] R. Hooke. Flow law for polycrystalline ice in glaciers: comparison of theoretical predictions, laboratory data, and field measurements. *Rev. Geophys. Space. Phys.*, 19(4):664–672, 1981.
- [67] K. M. Cuffey and W. S. B. Paterson. *The Physics of Glaciers*. Elsevier, 4th edition, 2010.
- [68] M. W. Mahaffy. A three-dimensional numerical model of ice sheets: tests on the Barnes Ice Cap, Northwest Territories. *J. Geophys. Res.*, 81(6):1059–1066, 1976.
- [69] F. Saito, A. Abe-Ouchi, and H. Blatter. An improved numerical scheme to compute horizontal gradients at the ice-sheet margin: its effect on the simulated ice thickness and temperature. *Ann. Glaciol.*, 46:87–96, 2007.
- [70] A. Levermann, T. Albrecht, R. Winkelmann, M. A. Martin, M. Haseloff, and I. Joughin. Kinematic first-order calving law implies potential for abrupt ice-shelf retreat. *The Cryosphere*, 6:273–286, 2012. URL: <http://www.the-cryosphere.net/6/273/2012/tc-6-273-2012.html>.
- [71] M. Morlighem, J. Bondzio, H. Seroussi, E. Rignot, E. Larour, A. Humbert, and S. Rebuffi. Modeling of Store Gletscher’s calving dynamics, West Greenland, in response to ocean thermal forcing. *Geophysical Research Letters*, pages n/a–n/a, 2016. URL: <http://doi.wiley.com/10.1002/2016GL067695>, doi:10.1002/2016GL067695.
- [72] J. M. Amundson, M. Fahnestock, M. Truffer, J. Brown, M. P. Lüthi, and R. J. Motyka. Ice mélange dynamics and implications for terminus stability, Jakobshavn Isbrae, Greenland. *J. Geophys. Res.*, 2010. F01005. doi:10.1029/2009JF001405.
- [73] T. Albrecht, M. Martin, M. Haseloff, R. Winkelmann, and A. Levermann. Parameterization for subgrid-scale motion of ice-shelf calving fronts. *The Cryosphere*, 5:35–44, 2011.
- [74] N. Golledge and twelve others. Glaciology and geological signature of the Last Glacial Maximum Antarctic ice sheet. *Quaternary Sci. Rev.*, 78(0):225–247, 2013. doi:10.1016/j.quascirev.2013.08.011.
- [75] R. Winkelmann, A. Levermann, K. Frieler, and M.A. Martin. Increased future ice discharge from antarctica owing to higher snowfall. *Nature*, 492:239–242, 2012.
- [76] J. Feldmann, T. Albrecht, C. Khroulev, F. Pattyn, and A. Levermann. Resolution-dependent performance of grounding line motion in a shallow model compared to a full-Stokes model according to the MISMIP3d intercomparison. *J. Glaciol.*, 60(220):353–360, 2014. doi:10.3189/2014JoG13J093.
- [77] R. M. Gladstone, A. J. Payne, and S. L. Cornford. Parameterising the grounding line in flow-line ice sheet models. *The Cryosphere*, 4:605–619, 2010. doi:10.5194/tc-4-605-2010.
- [78] G. K. C. Clarke. Subglacial processes. *Annu. Rev. Earth Planet. Sci.*, 33:247–276, 2005. doi:10.1146/annurev.earth.33.092203.122621.

- [79] R. Calov, R. Greve, A. Abe-Ouchi, E. Bueler, P. Huybrechts, J. V. Johnson, F. Pattyn, D. Pollard, C. Ritz, F. Saito, and L. Tarasov. Results from the ice sheet model intercomparison project—Heinrich event intercomparison (IS-MIP HEINO). *J. Glaciol.*, 56(197):371–383, 2010.
- [80] C. Schoof. Variational methods for glacier flow over plastic till. *J. Fluid Mech.*, 555:299–320, 2006.
- [81] S. Tulaczyk, W. B. Kamb, and H. F. Engelhardt. Basal mechanics of Ice Stream B, West Antarctica 1.~Till mechanics. *J. Geophys. Res.*, 105(B1):463–481, 2000.
- [82] E. Bueler and W. van Pelt. Mass-conserving subglacial hydrology in the parallel ice sheet model version 0.6. *Geoscientific Model Development*, 8(6):1613–1635, 2015. doi: 10.5194/gmd-8-1613-2015.
- [83] C. S. Lingle and J. A. Clark. A numerical model of interactions between a marine ice sheet and the solid earth: Application to a West Antarctic ice stream. *J. Geophys. Res.*, 90(C1):1100–1114, 1985.
- [84] E. Bueler, C. S. Lingle, and J. A. Kallen-Brown. Fast computation of a viscoelastic deformable Earth model for ice sheet simulation. *Ann. Glaciol.*, 46:97–105, 2007.
- [85] Ralf Greve. Glacial isostasy: Models for the response of the Earth to varying ice loads. In Brian Straughan and others, editors, *Continuum Mechanics and Applications in Geophysics and the Environment*, 307–325. Springer, 2001.
- [86] C. Schoof. The effect of basal topography on ice sheet dynamics. *Continuum Mech. Thermodyn.*, 15:295–307, 2003. doi:10.1007/s00161-003-0119-3.
- [87] S. Tulaczyk, W. B. Kamb, and H. F. Engelhardt. Basal mechanics of Ice Stream B, West Antarctica 2.~Undrained plastic bed model. *J. Geophys. Res.*, 105(B1):483–494, 2000.
- [88] M. Siegert, A. Le Brocq, and A. Payne. *Hydrological connections between Antarctic subglacial lakes, the flow of water beneath the East Antarctic Ice Sheet and implications for sedimentary processes*, pages 3–10. Wiley-Blackwell, Malden, MA, USA, 2007.
- [89] C. Schoof, I. J. Hewitt, and M. A. Werder. Flotation and free surface flow in a model for subglacial drainage. Part I: Distributed drainage. *J. Fluid Mech.*, 702:126–156, 2012.
- [90] Christina L. Hulbe and Douglas R. MacAyeal. A new numerical model of coupled inland ice sheet, ice stream, and ice shelf flow and its application to the West Antarctic Ice Sheet. *J. Geophys. Res.*, 104(B11):25349–25366, 1999.
- [91] M. Lüthi, M. Funk, A. Iken, S. Gogineni, and M. Truffer. Mechanisms of fast flow in Jakobshavns Isbræ, Greenland; Part III: measurements of ice deformation, temperature and cross-borehole conductivity in boreholes to the bedrock. *J. Glaciol.*, 48(162):369–385, 2002.
- [92] Catherine Ritz, Vincent Rommelaere, and Christophe Dumas. Modeling the evolution of Antarctic ice sheet over the last 420,000 years: Implications for altitude changes in the Vostok region. *J. Geophys. Res.*, 106(D23):31943–31964, 2001.
- [93] A. H. Jarosch, C. G. Schoof, and F. S. Anslow. Restoring mass conservation to shallow ice flow models over complex terrain. *The Cryosphere*, 7(1):229–240, 2013. doi:10.5194/tc-7-229-2013.
- [94] S. Balay and others. PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.5, Argonne National Laboratory, 2014.
- [95] R. C. A. Hindmarsh and A. J. Payne. Time-step limits for stable solutions of the ice-sheet equation. *Ann. Glaciol.*, 23:74–85, 1996.
- [96] A. Payne. EISMINT: Ice sheet model intercomparison exercise phase two. Proposed simplified geometry experiments. 1997. URL: <http://homepages.vub.ac.be/~phuybrec/eismint/thermo-descr.pdf>.
- [97] Ed Bueler. Lessons from the short history of ice sheet model intercomparison. *The Cryosphere Discussions*, 2:399–412, 2008. URL: <http://www.the-cryosphere-discuss.net/2/399/2008/>.
- [98] P. Halfar. On the dynamics of the ice sheets 2. *J. Geophys. Res.*, 88(C10):6043–6051, 1983.

- [99] R. C. A. Hindmarsh. Thermoviscous stability of ice-sheet flows. *J. Fluid Mech.*, 502:17–40, 2004.
- [100] R. C. A. Hindmarsh. Stress gradient damping of thermoviscous ice flow instabilities. *J. Geophys. Res.*, 2006. doi:10.1029/2005JB004019.
- [101] F. Saito, A. Abe-Ouchi, and H. Blatter. European Ice Sheet Modelling Initiative (EISMINT) model intercomparison experiments with first-order mechanics. *J. Geophys. Res.*, 2006. doi:10.1029/2004JF000273.
- [102] A. J. Payne and P. W. Dongelmans. Self-organization in the thermomechanical flow of ice sheets. *J. Geophys. Res.*, 102(B6):12219–12233, 1997.
- [103] F. Pattyn and twenty others. Benchmark experiments for higher-order and full Stokes ice sheet models (ISMIP-HOM). *The Cryosphere*, 2:95–108, 2008.
- [104] O. Gagliardini and T. Zwinger. The ISMIP-HOM benchmark experiments performed using the Finite-Element code Elmer. *The Cryosphere*, 2(1):67–76, 2008. URL: <http://www.the-cryosphere.net/2/67/2008/>.
- [105] Ralf Greve, Ryoji Takahama, and Reinhard Calov. Simulation of large-scale ice-sheet surges: the ISMIP-HEINO experiments. *Polar Meteorol. Glaciol.*, 20:1–15, 2006.
- [106] F. Pattyn, C. Schoof, L. Perichon, and 15 others. Results of the Marine Ice Sheet Model Intercomparison Project, MISIP. *The Cryosphere*, 6:573–588, 2012. doi:10.5194/tc-6-573-2012.
- [107] F. Pattyn, L. Perichon, G. Durand, and 25 others. Grounding-line migration in plan-view marine ice-sheet models: results of the ice2sea MISIP3d intercomparison. *J. Glaciol.*, 59(215):410–422, 2013.
- [108] C. Schoof. Marine ice-sheet dynamics. Part 1. The case of rapid sliding. *J. Fluid Mech.*, 573:27–55, 2007.
- [109] D. Goldberg, D. M. Holland, and C. Schoof. Grounding line movement and ice shelf buttressing in marine ice sheets. *J. Geophys. Res.*, 2009. doi:10.1029/2008JF001227.
- [110] Ian Joughin, Sarah B. Das, Matt A. King, Ben E. Smith, Ian M. Howat, and Twila Moon. Seasonal Speedup Along the Western Flank of the Greenland Ice Sheet. *Science*, 320(5877):781–783, 2008. URL: <http://www.sciencemag.org/cgi/content/abstract/320/5877/781>, doi:10.1126/science.1153288.
- [111] P. Dickens and T. Morey. Increasing the scalability of PISM for high resolution ice sheet models. In *Proceedings of the 14th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, May 2013, Boston*. 2013.
- [112] N. Golledge, A. Mackintosh, and 8 others. Last Glacial Maximum climate in New Zealand inferred from a modelled Southern Alps icefield. *Quaternary Science Reviews*, 46:30–45, 2012. doi:10.1016/j.quascirev.2012.05.004.
- [113] J.L. Bamber, R.L. Layberry, and S.P. Gogenini. A new ice thickness and bed data set for the Greenland ice sheet 1: Measurement, data reduction, and errors. *J. Geophys. Res.*, 106 (D24):33,773–33,780, 2001.
- [114] Ed Bueler, Constantine Khroulev, Andy Aschwanden, Ian Joughin, and Ben E. Smith. Modeled and observed fast flow in the Greenland ice sheet. submitted, 2009.
- [115] E. Larour, H. Seroussi, M. Morlighem, and E. Rignot. Continental scale, high order, high spatial resolution, ice sheet modeling using the Ice Sheet System Model (ISSM). *J. Geophys. Res.*, 2012. doi:10.1029/2011JF002140.
- [116] S. Price, A. Payne, I. Howat, and B. Smith. Committed sea-level rise for the next century from Greenland ice sheet dynamics during the past decade. *Proc. Nat. Acad. Sci.*, 108(22):8978–8983, 2011. doi:10.1073/pnas.1017313108.
- [117] I. Joughin. Ice-sheet velocity mapping: a combined interferometric and speckle-tracking approach. *Ann. Glaciol.*, 34:195–201, 2002.
- [118] W. J. J. van Pelt, J. Oerlemans, C. H. Reijmer, R. Pettersson, V. A. Pohjola, E. Isaksson, and D. Divine. An iterative inverse method to estimate basal topography and initialize ice flow models. *The Cryosphere*, 7(3):987–1006, 2013. doi:10.5194/tc-7-987-2013.
- [119] W. T. Pfeffer, J. T. Harper, and S. O’Neel. Kinematic constraints on glacier contributions to 21st-century sea-level rise. *Science*, 321:1340–1343, 2008.

- [120] R.C. Bales, J.R. McConnell, E. Mosley-Thompson, and G. Lamorey. Accumulation map for the Greenland Ice Sheet: 1971-1990. *Geophys. Res. Lett*, 28(15):2967–2970, 2001. URL: <http://zero.eng.ucmerced.edu/rcbales/PARCA/>.
- [121] P.J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.
- [122] Pieter Wesseling. *Principles of Computational Fluid Dynamics*. Springer-Verlag, 2001.
- [123] R. Sayag and M. G. Worster. Axisymmetric gravity currents of power-law fluids over a rigid horizontal surface. *J. Fluid Mech.*, 2013. doi:10.1017/jfm.2012.545.
- [124] R. Sayag, S. S. Pegler, and M. G. Worster. Floating extensional flows. *Physics of Fluids*, 2012. doi:10.1063/1.4747184.
- [125] C. R. Bentley. Glaciological studies on the Ross Ice Shelf, Antarctica, 1973–1978. *Antarctic Research Series*, 42(2):21–53, 1984.
- [126] V. Rommelaere and D. R. MacAyeal. Large-scale rheology of the Ross Ice Shelf, Antarctica, computed by a control method. *Ann. Glaciol.*, 24:43–48, 1997.
- [127] A. Humbert, R. Greve, and K. Hutter. Parameter sensitivity studies for the ice flow of the Ross Ice Shelf, Antarctica. *J. Geophys. Res.*, 2005. doi:10.1029/2004JF000170.
- [128] A. M. Le Brocq, A. J. Payne, and A. Vieli. An improved Antarctic dataset for high resolution numerical ice sheet models (ALBMAP v1). *Earth System Science Data*, 2(2):247–260, 2010. URL: <http://www.earth-syst-sci-data.net/2/247/2010/>, doi:10.5194/essd-2-247-2010.
- [129] E. Rignot, J. Mouginot, and B. Scheuchl. Ice flow of the Antarctic Ice Sheet. *Science*, 333(6048):1427–1430, 2011. doi:10.1126/science.1208336.
- [130] T. Albrecht and A. Levermann. Fracture field for large-scale ice dynamics. *J. Glaciol.*, 58(207):165–176, 2012. doi:10.3189/2012JoG11J191.
- [131] E. Bueler. An exact solution to the temperature equation in a column of ice and bedrock. preprint \texttt{arXiv:0710.1314}, 2007.
- [132] K. W. Morton and D. F. Mayers. *Numerical Solutions of Partial Differential Equations: An Introduction*. Cambridge University Press, 2nd edition, 2005.
- [133] D. Jenssen. A three-dimensional polar ice-sheet model. *J. Glaciol.*, 18:373–389, 1977.