
PISM, a Parallel Ice Sheet Model

version 1.2

the PISM authors

Nov 24, 2021

CONTENTS

1 Acknowledgements	3
2 Installing PISM	5
2.1 Required tools and libraries	5
2.2 Installing parallel I/O libraries	6
2.2.1 Installing HDF5-based parallel NetCDF	6
2.2.2 Installing PnetCDF	8
2.2.3 Installing NCAR ParallelIO	8
2.3 Installation Cookbook	9
2.3.1 Installing prerequisites on Debian or Ubuntu	9
2.3.2 Installing prerequisites and building PISM on Arch Linux	10
2.3.3 Installing required libraries on macOS	11
2.3.4 Installing PISM using Spack	11
2.3.5 Building PETSc	12
2.3.6 Building PISM	13
2.3.7 Common build problems and solutions	15
2.4 Quick tests of the installation	16
2.5 Next steps	17
2.6 Rebuilding PISM documentation	17
2.6.1 Manual	17
2.6.2 Source Code Browser	18
2.6.3 Re-building documentation without PISM's prerequisites	18
3 PISM User's Manual	19
3.1 Getting started: a Greenland ice sheet example	19
3.1.1 Input data	20
3.1.2 First run	21
3.1.3 Watching the first run	22
3.1.4 Second run: a better ice-dynamics model	24
3.1.5 Third run: higher resolution	26
3.1.6 Fourth run: paleo-climate model spin-up	27
3.1.7 Grid sequencing	30
3.1.8 An ice dynamics parameter study	33
3.2 Ice dynamics, the PISM view	37
3.2.1 Stress balance models: SIA, SSA, and the First Order Approximation	37
3.2.2 A hierarchy of simplifying assumptions for grounded ice flow	38
3.2.3 Evolutionary versus diagnostic modeling	39
3.2.4 Climate inputs, and their interface with ice dynamics	40
3.3 Initialization and bootstrapping	43
3.3.1 Initialization from a saved model state	43

3.3.2	Bootstrapping	44
3.4	Modeling choices	45
3.4.1	Model domain, grid, and time	46
3.4.2	Ice dynamics and thermodynamics	55
3.4.3	The subglacier	71
3.4.4	Marine ice sheet modeling	85
3.4.5	Modeling individual outlet glaciers	94
3.4.6	Disabling sub-models	96
3.4.7	Dealing with more difficult modeling choices	97
3.5	Practical usage	97
3.5.1	Handling NetCDF files	97
3.5.2	Input and output	98
3.5.3	Understanding adaptive time-stepping	101
3.5.4	Scalar diagnostic quantities	104
3.5.5	Spatially-varying diagnostic quantities	106
3.5.6	Snapshots of the model state	107
3.5.7	Run-time diagnostic viewers	109
3.5.8	PISM’s configuration parameters and how to change them	109
3.5.9	Regridding	111
3.5.10	Signals, to control a running PISM model	112
3.5.11	Balancing the books	113
3.5.12	PETSc options for PISM users	116
3.5.13	Utility and test scripts	117
3.5.14	Using PISM for flow-line modeling	117
3.5.15	Managing source code modifications	118
3.5.16	ISMIP6 Greenland	119
3.6	Simplified geometry experiments	121
3.6.1	EISMINT II	122
3.6.2	MISMIP	124
3.6.3	MISMIP3d	125
3.6.4	ISMIP-HOM	128
3.7	Verification	136
3.7.1	Sample convergence plots	139
3.8	Validation case studies	139
3.8.1	An SIA flow model for a table-top laboratory experiment	139
3.8.2	An SSA flow model for the Ross Ice Shelf in Antarctica	144
3.9	Example: A regional model of the Jakobshavn outlet glacier in Greenland	147
3.9.1	Get the drainage basin delineation tool	148
3.9.2	Preprocess the data and get the whole ice sheet model file	148
3.9.3	Identify the drainage basin for the modeled outlet glacier	149
3.9.4	Cut out the computational domain for the regional model	150
3.9.5	Quick start	150
3.9.6	Spinning-up the regional model on a 5 km grid	150
3.9.7	Century run on a 2 km grid	152
3.9.8	Plotting results	152
3.10	Configuration parameters	153
3.11	Diagnostic quantities	221
3.11.1	Spatially-variable fields	222
3.11.2	Scalar time-series	238
4	Climate forcing	243
4.1	Using time-dependent forcing	244
4.1.1	Introduction	244
4.1.2	Scalar time-dependent inputs	245

4.1.3	Spatially-variable time-dependent inputs	245
4.1.4	Periodic forcing	245
4.1.5	Adding time bounds	246
4.2	Examples and corresponding options	247
4.2.1	One way coupling to a climate model	247
4.2.2	Using climate anomalies	247
4.2.3	SeaRISE-Greenland	248
4.2.4	SeaRISE-Greenland paleo-climate run	248
4.2.5	Antarctic paleo-climate runs	248
4.3	Testing if forcing data is used correctly	249
4.3.1	Visualizing climate inputs, without ice dynamics	249
4.3.2	Using low-resolution test runs	250
4.3.3	Visualizing the climate inputs in the Greenland case	250
4.4	Surface mass and energy process model components	252
4.4.1	The “invisible” model	252
4.4.2	Reading top-surface boundary conditions from a file	254
4.4.3	Elevation-dependent temperature and mass balance	255
4.4.4	Temperature-index scheme	256
4.4.5	PIK	260
4.4.6	Scalar temperature offsets	260
4.4.7	Adjustments using modeled change in surface elevation	261
4.4.8	Mass flux adjustment	262
4.4.9	Using climate data anomalies	262
4.4.10	The caching modifier	263
4.4.11	Preventing grounding line retreat	263
4.5	Atmosphere model components	264
4.5.1	Reading boundary conditions from a file	264
4.5.2	Cosine yearly cycle	265
4.5.3	SeaRISE-Greenland	265
4.5.4	PIK	266
4.5.5	One weather station	267
4.5.6	Scalar temperature offsets	267
4.5.7	Scalar precipitation offsets	267
4.5.8	Precipitation scaling	268
4.5.9	Precipitation correction using scalar temperature offsets	268
4.5.10	Adjustments using modeled change in surface elevation	269
4.5.11	Using climate data anomalies	270
4.5.12	Orographic precipitation	270
4.6	Ocean model components	272
4.6.1	Constant in time and space	272
4.6.2	Reading forcing data from a file	273
4.6.3	PIK	273
4.6.4	Basal melt rate and temperature from thermodynamics in boundary layer	274
4.6.5	PICO	275
4.6.6	Scalar sea level offsets	276
4.6.7	Two-dimensional sea level offsets	276
4.6.8	Scalar sub-shelf temperature offsets	277
4.6.9	Scalar sub-shelf mass flux offsets	277
4.6.10	Scalar sub-shelf mass flux fraction offsets	277
4.6.11	Two-dimensional sub-shelf mass flux offsets	278
4.6.12	Scalar melange back pressure offsets	278
4.6.13	Melange back pressure as a fraction of pressure difference	279
4.6.14	The caching modifier	279

5 Technical notes	281
5.1 Release checklist	281
5.2 CF standard names used by PISM	282
5.2.1 Existing standard names	282
5.2.2 <i>Proposed</i> standard names	283
5.2.3 Final technical notes	284
5.3 On the vertical coordinate in PISM, and a critical change of variable	284
5.4 Using Schoof's parameterized bed roughness technique in PISM	286
5.4.1 An explanation	287
5.4.2 Theory	287
5.4.3 Practical application, and Taylor approximation	288
5.5 Calving front stress boundary condition	290
5.5.1 Notation	291
5.5.2 Calving front stress boundary condition	291
5.5.3 Shallow shelf approximation	292
5.5.4 Implementing the calving front boundary condition	292
5.6 Notes about the flow-line SSA	293
5.6.1 An observation	293
5.6.2 Discrete analog of this property	293
5.6.3 Discretization	294
5.7 BOMBPROOF, PISM's numerical scheme for conservation of energy	296
5.7.1 Introduction	296
5.7.2 Conservation of energy in a shallow ice sheet	297
5.7.3 The core BOMBPROOF scheme	298
5.7.4 Stability properties of the BOMBPROOF scheme	300
5.7.5 Temperate basal boundary condition, and computing the basal melt rate	303
5.8 Computing steady-state subglacial water flux	303
5.8.1 The algorithm	305
5.9 Three-equation ocean model (implementation details)	306
5.10 Blatter stress balance solver: technical details	308
5.10.1 Notation	310
5.10.2 Introduction	310
5.10.3 Weak form	311
5.10.4 Boundary conditions	312
5.10.5 Solver implementation	314
5.10.6 Testing and verification	324
5.10.7 Known issues and future work	326
6 Contributing to PISM	327
6.1 Submitting bug reports	328
6.2 PISM coding guidelines	328
6.2.1 File names	329
6.2.2 Source and header files	329
6.2.3 Inline functions and methods	329
6.2.4 Including header files	330
6.2.5 Naming	330
6.2.6 Namespaces	331
6.2.7 Formatting	331
6.2.8 Error handling	334
6.2.9 Function design	335
6.3 Git introduction for PISM developers	336
6.3.1 Recommended Git configuration	336
6.3.2 Working on a new branch	337
6.3.3 Writing better commit messages	338

6.4	Git branches	338
6.5	Development workflow	339
6.5.1	Setting up the environment	340
6.5.2	Editing source code	340
6.5.3	Compiling PISM	341
6.5.4	Debugging	341
6.5.5	Writing tests	343
6.5.6	Editing PISM’s manual	344
6.6	How do I...?	345
6.6.1	Create and use configuration flags and parameters	346
6.6.2	Create and use additional variables	346
6.6.3	Read data from a file	347
6.6.4	Write data to a file	348
6.6.5	Read scalar forcing data	348
6.6.6	Read 2D forcing fields	348
7	Authorship	349
Bibliography		351

This *Manual* was built using revision v1.0-2782-gf20e53faf committed on 2021-11-23 16:44:08 -0900.

CHAPTER**ONE**

ACKNOWLEDGEMENTS

Development of PISM is supported by NSF grants PLR-1644277 and PLR-1914668 and NASA grants NNX17AG65G and 20-CRYO2020-0052.

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 Høyervold Svendsen, Martin Truffer, Shuteng 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 two external libraries we rely on:

- [CalCalcs](#), Copyright © 2010 by David W. Pierce, GPL
- [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	2020	NSF	PLR-1603799	Collaborative Research: Understanding the controls on spatial and temporal variability in ice discharge using a Greenland-wide ice sheet model
2017	2022	NSF	PLR-1644277	Collaborative Research: Feedbacks between Orographic Precipitation and Ice Dynamics
2017	2022	NASA	NNX17AG65G	Using ICESat/OIB elevation and satellite-derived velocity changes to constrain time-varying basal motion
2020	2022	NSF	PLR-1914668	Collaborative Research: Resolving Earth Structure Influence on Ice-Sheet Stability in the Wilkes Subglacial Basin (RESISSt)
2021	2024	NASA	20-CRYO2020-0052	A Reanalysis of the Greenland Ice Sheet (RAGIS-40)
2021	2024	NASA		Calibrated sensitivity of the Greenland Ice Sheet to major climate changes from the next-generation radiotriangulation and ensemble modeling
2022	2027	NSF		iHARP: Institute for Harnessing the Data and Model Revolution in Polar Regions

INSTALLING PISM

The fastest path to a fully functional PISM installation is to use a Linux system with a Debian- or Arch Linux-based package system (e.g. [Ubuntu](#)). In Debian and derivatives, start by following subsection [Installing prerequisites on Debian or Ubuntu](#), to install tools and libraries used by PISM and then [Building PISM](#) to install PISM itself. In Arch Linux and derivatives, follow the instructions in subsection [Installing prerequisites and building PISM on Arch Linux](#).

2.1 Required tools 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 ≥ 4.4
PETSc ²	version ≥ 3.5
UDUNITS	any recent version

Before installing these “by hand”, check sections [Installing prerequisites on Debian or Ubuntu](#) 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.

Note: We recommend un-installing all MPI libraries except one. In most cases there is no reason to have both Open MPI and MPICH, for example.

Optional libraries listed below are needed for certain PISM features, namely computing longitude, latitude coordinates of grid points and parallel I/O. These libraries are recommended, but not strictly required:

Optional Library	Comment
PROJ	version ≥ 6.0 (used to compute longitude-latitude grid coordinates and cell bounds)
PnetCDF	Can be used for faster parallel I/O
ParallelIO	Can be used for faster parallel I/O

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

² “PETSc” is pronounced “pet-see”.

[Python](#) is needed for the PETSc installation process; a number of PISM’s pre- and post-processing scripts also use Python (either 2.7 or 3.x), while [Git](#) is usually needed to download the PISM code.

PISM’s Python bindings support Python 2.7 or 3.3 and later³.

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

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 Installing parallel I/O libraries

High-resolution simulations (e.g. modeling the entire Greenland ice sheet using the 900 m resolution and a grid of 1700 by 3000 points) can produce huge amounts of data and I/O can become a bottleneck.

PISM supports several parallel I/O approaches that take advantage of parallel NetCDF, PnetCDF, and NCAR ParallelIO. The administrators of your HPC system should be able to help you install these libraries, but it may be easier to install them in the “home” directory instead.

This section describes the steps needed to build

- NetCDF with parallel I/O based on HDF5 (needed to use PISM’s option `-o_format netcdf4_parallel`),
- PNetCDF (needed to use PISM’s option `-o_format pnetcdf`),
- ParallelIO (needed to use options `-o_format pio_netcdf4p`, `-o_format pio_netcdf4c`, `-o_format pio_pnetcdf`, `-o_format pio_netcdf`).

Scripts below install libraries in `~/local/library_name`, using `~/local/build/library_name` to build them.

Section [Building PISM with libraries in non-standard locations](#) explains how build PISM with these libraries.

Section [PISM’s I/O performance](#) explains how to use them in PISM.

2.2.1 Installing HDF5-based parallel NetCDF

Installing HDF5

```

1 # Install HDF5 1.12.0 with parallel I/O in ~/local/hdf5,
2 # using ~/local/build/hdf5 as the build directory.
3
4 version=1.12.0
5 prefix=$HOME/local/hdf5
6 build_dir=~/local/build/hdf5
7 hdf5_site=https://support.hdfgroup.org/ftp/HDF5/releases/hdf5-1.12
8 url=${hdf5_site}/hdf5-${version}/src/hdf5-${version}.tar.gz
9
10 mkdir -p ${build_dir}
11 pushd ${build_dir}
12
13 wget -nc ${url}
```

(continues on next page)

³ PISM’s Python bindings are tested using Python 3.6.

(continued from previous page)

```

14 tar xzf hdf5-{version}.tar.gz
15
16 pushd hdf5-{version}
17
18 CC=mpicc CFLAGS=-w ./configure \
19   --disable-static \
20   --enable-parallel \
21   --prefix={prefix} 2>&1 | tee hdf5_configure.log
22
23 make all 2>&1 | tee hdf5_compile.log
24 make install 2>&1 | tee hdf5_install.log
25
26 popd
27 popd

```

To compile parallel HDF5 one should use the MPI compiler wrapper `mpicc` and run `configure` with the option `--enable-parallel`. The flag `-w` is not important: it hides numerous compiler warnings emitted when building HDF5.

Installing NetCDF

```

1 # Install parallel NetCDF using parallel HDF5 in ~/local/hdf5 and
2 # ~/local/build/netcdf as a build directory.
3
4 hdf5=~/local/hdf5
5
6 version=4.7.4
7 prefix=$HOME/local/netcdf
8 build_dir=~/local/build/netcdf
9 url=ftp://ftp.unidata.ucar.edu/pub/netcdf/netcdf-c-{version}.tar.gz
10
11 mkdir -p ${build_dir}
12 pushd ${build_dir}
13
14 wget -nc ${url}
15 tar zxf netcdf-c-{version}.tar.gz
16
17 pushd netcdf-c-{version}
18
19 CC=mpicc CPPFLAGS=-I${hdf5}/include LDFLAGS=-L${hdf5}/lib ./configure \
20   --enable-netcdf4 \
21   --disable-dap \
22   --prefix=${prefix} 2>&1 | tee netcdf_configure.log
23
24 make all 2>&1 | tee netcdf_compile.log
25 make install 2>&1 | tee netcdf_install.log
26
27 popd
28 popd

```

Here we use the same compiler wrapper and set `CPPFLAGS` and `LDFLAGS` to select the parallel HDF5 library installed earlier. The option `--enable-netcdf4` is required for parallel I/O; `--disable-dap` is not required (it disables a NetCDF feature not used by PISM).

2.2.2 Installing PnetCDF

```
1 # Install PnetCDF 1.12.1 in ~/local/pnetcdf,
2 # using ~/local/build/pnetcdf as a build directory.
3
4 version=1.12.1
5 prefix=$HOME/local/pnetcdf
6 build_dir=~/local/build/pnetcdf/
7 url=https://parallel-netcdf.github.io/Release/pnetcdf-${version}.tar.gz
8
9 mkdir -p ${build_dir}
10 pushd ${build_dir}
11
12 wget -nc ${url}
13 tar xzf pnetcdf-${version}.tar.gz
14
15 pushd pnetcdf-${version}
16
17 ./configure \
18   --prefix=${prefix} \
19   --enable-shared \
20   --disable-static \
21   --disable-cxx \
22   --disable-fortran
23
24 make all
25 make install
26
27 popd
28 popd
```

Here we disable PnetCDF's C++ and Fortran APIs and build the shared library.

2.2.3 Installing NCAR ParallelIO

```
1 # Install NCAR ParallelIO in ~/local/parallelio
2 # using parallel NetCDF and PnetCDF installed in ~/local/netcdf
3 # and ~/local/pnetcdf. Uses ~/local/build/parallelio
4 # as a build directory.
5
6 netcdf_prefix=~/local/netcdf
7 pnetcdf_prefix=~/local/pnetcdf
8
9 url=https://github.com/NCAR/ParallelIO.git
10 build=~/local/build/parallelio
11 prefix=$HOME/local/parallelio
12
13 rm -rf ${build}
14 mkdir -p ${build}/build ${build}/sources
15
16 git clone ${url} ${build}/sources
17
18 pushd ${build}/sources
19 git checkout -b 2_5_4 pio2_5_4
20 popd
```

(continues on next page)

(continued from previous page)

```

22 pushd ${build}/build
23
24 CC=mpicc cmake \
25 -DCMAKE_C_FLAGS="-fPIC" \
26 -DCMAKE_INSTALL_PREFIX=${prefix} \
27 -DNetCDF_PATH=${netcdf_prefix} \
28 -DPnetCDF_PATH=${pnetcdf_prefix} \
29 -DPIO_ENABLE_FORTRAN=0 \
30 -DPIO_ENABLE_TIMING=0 \
31 ${build}/sources
32
33 make install
34
35 popd

```

Here we use CMake's variable `CMAKE_FIND_ROOT_PATH` to tell CMake to use libraries in `~/local/netcdf` and `~/local/pnetcdf`, to install in `~/local/parallelio`, and to disable ParallelIO features not used by PISM.

2.3 Installation Cookbook

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

2.3.1 Installing prerequisites on Debian or Ubuntu

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

Name	Comment
cmake	required; used to configure PISM
g++	required; used to build PISM
git	required; used to download PISM's source code
libfftw3-dev	required by PISM
libgsl-dev	required by PISM
libnetcdf-dev	required by PISM
libudunits2-dev	required by PISM
netcdf-bin	required; ncgen is used during the build process
petsc-dev	required by PISM
cdo	optional; used in some pre-processing scripts
cmake-curses-gui	optional; a text-based interface for CMake
libpnetcdf-dev	optional; used for parallel I/O
libproj-dev	optional; used to compute longitude and latitude coordinates of grid points
libx11-dev	optional; used by PISM's run-time viewers
nco	optional; used in many pre-processing scripts
ncview	optional; a nice tool for examining NetCDF datasets
python3-dev	optional; used by PISM's Python bindings
python3-netcdf4	optional; used in most post-processing scripts
python3-nose	optional; used by PISM's regression tests
python3-numpy	optional; used in some pre- and post-processing scripts
python3-pyproj	optional; used in some pre-processing scripts
python3-scipy	optional; used in some pre- and post-processing scripts
swig	optional; used to build PISM's Python bindings

You may be able to install these by running

```
apt-get install -y cmake g++ git libfftw3-dev libgsl-dev libnetcdf-dev libudunits2-dev netcdf-bin  
- petsc-dev cdo cmake-curses-gui libpnetcdf-dev libproj-dev libx11-dev nco ncview python3-dev  
- python3-netcdf4 python3-nose python3-numpy python3-pyproj python3-scipy swig
```

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

The command above takes care of all PISM prerequisites, including PETSc. Set PETSC_DIR=/usr/lib/petsc¹ and follow the steps in *Building PISM* to build PISM itself.

2.3.2 Installing prerequisites and building PISM on Arch Linux

PISM is registered as a package in the Arch Linux User Repository (AUR). This package will build the latest PISM release and its dependencies. PISM is linked to Open MPI and is installed under /usr.

Installation from the AUR requires an AUR helper such as yay. If you do not already have an AUR helper installed, install yay with the following commands:

```
git clone https://aur.archlinux.org/yay.git  
cd git  
makepkg -si
```

You can then install PISM and its dependencies with the following command:

¹ In this case you do not need to set PETSC_ARCH.

```
yay -Sy pism
```

Once installed, the PISM binaries (e.g. `pismr`, `pismv`, various Python tools) are available in the PATH and do not require further intervention to work. It is recommended that the installation is manually verified with the instructions in section [Quick tests of the installation](#).

Note: The AUR package for PISM is maintained by Julien Seguinot; please e-mail him with questions about it. His email is: *first name two first letters + last name three first letters + at + posteo and then dot + eu.*

2.3.3 Installing required libraries on macOS

Follow these steps to install PISM's prerequisites on the macOS 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](#).

2.3.4 Installing PISM using Spack

On supercomputers, Linux, and macOS PISM can be installed using the `Spack` package manager.

Installing PISM using this method is easy: install Spack itself (see [Spack documentation](#)) and then run

```
spack install pism
```

This will install PISM and all its prerequisites, including PETSc. The default PETSc configuration in its Spack package includes many optional features not used by PISM. You may want to disable these; to do this, use this command instead:

```
spack install pism ^petsc~metis~hdf5~hypre~superlu-dist
```

Note: The Spack package for PISM is maintained by Elizabeth Fischer (eafischer2@alaska.edu); please e-mail her with questions about it. General questions about `Spack` should be sent directly to Spack developers.

Note: With default Spack settings this installation method relies on building most (if not all) of PISM's prerequisites from sources.

This may take a very long time.

Please see [Spack documentation](#) (system packages) for a way to avoid re-building tools and libraries available on your system.

2.3.5 Building PETSc

PISM is built on top of [PETSc](#), which is actively developed and an up-to-date PETSc distribution may not be available in package repositories. Download the PETSc source by grabbing the current gzipped tarball at:

<https://petsc.org/release/download/>

Use version 3.5 or newer; see [Required tools and libraries](#) for details.

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.

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:

Listing 2.1: Building PETSc

```
petsc_prefix=$HOME/local/petsc
PETSC_DIR=$PWD
PETSC_ARCH="linux-opt"

./configure \
--prefix=${petsc_prefix} \
--with-cc=mpicc \
--with-cxx=mpicxx \
--with-fc=mpifort \
--with-shared-libraries \
--with-debugging=0 \
--with-petsc4py \
--download-f2cblaslapack

export PYTHONPATH=${petsc_prefix}/lib
make all
make install
make PETSC_DIR=${petsc_prefix} PETSC_ARCH="" check
```

This will install PETSc and its Python bindings in the directory `~/local/petsc`. Remove `--with-petsc4py` if you don't need Python bindings (e.g. if you are not going to use PISM's Python bindings).

- You need to define the environment variables `PETSC_DIR` and `PETSC_ARCH`¹ (one way is shown here) *before* running the configuration script.
- We recommend using MPI's compiler wrappers to specify an MPI library when installing PETSc (see `--with-cc=mpicc`, `--with-cxx=mpicxx`, and `--with-fc=mpifort` above).
- Turning off the inclusion of debugging code and symbols (`--with-debugging=0`) 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.

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

- The option `--download-f2cblaslapack=1` tells PETSc to download BLAS and LAPACK rather than using the system-wide version. This tends to work well for PISM, but see section 3.5.3 of [26] for instructions regarding building PETSc with optimized BLAS and LAPACK libraries.
- If you get messages suggesting that PETSc cannot configure using your existing MPI, you might want to try adding `--download-mpich=1` (or `--download-openmpi=1`).

Note: 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](#).

2.3.6 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 `mpexec` 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-3.10.2/
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.

Note: The `PETSC_ARCH` variable is not needed if PETSc was configured using the `--prefix=` option.

Follow these steps to build PISM:

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

```
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

Note: You can also [download a tarball from GitHub](#).

2. Build PISM:²

```
mkdir -p pism-stable/build
cd pism-stable/build
export CC=mpicc
export CXX=mpicxx
cmake -DCMAKE_INSTALL_PREFIX=~/pism ..
make -j install
```

Here `pism-stable` is the directory containing PISM source code while `~/pism` is the directory PISM will be installed into.

Variables `CC` and `CXX` specify MPI compiler wrappers provided by your MPI installation.

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.

Note: When using MPI's compiler wrappers, make sure that `mpicc` and `mpicxx` you select were used to compile the PETSc library: *PISM and PETSc have to use the same MPI installation.*

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

```
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`.

Temporary 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 and 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 (such as PETSc) 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.

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

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

PISM's build-time configuration

Some of PISM's features (the ones requiring additional libraries, for example) need to be enabled when building PISM. This section lists important build-time options.

Option	Description
CMAKE_BUILD_TYPE	"build type": set to "Debug" for development
BUILD_SHARED_LIBS	build shared (as opposed to static) libraries (this is the default)
Pism_LINK_STATICALLY	set CMake flags to try to ensure that everything is linked statically
Pism_LOOK_FOR_LIBRARIES	specifies whether PISM should look for libraries (disable this on Crays)
Pism_BUILD_EXTRA_EXECS	build additional executables (needed to run <code>make test</code>)
Pism_BUILD_PYTHON_BINDING	build PISM's Python binding; requires <code>petsc4py</code>
Pism_USE_PROJ	use the <code>PROJ</code> library to compute latitudes and longitudes of grid points
Pism_USE_PIO	use the <code>ParallelIO</code> library to write output files
Pism_USE_PARALLEL_NETCDF	use <code>NetCDF</code> for parallel file I/O
Pism_USE_PNETCDF	use <code>PnetCDF</code> for parallel file I/O
Pism_DEBUG	enables extra sanity checks in the code (this makes PISM a lot slower but simplifies development)

To enable PISM's use of `PROJ`, for example, run

```
cmake -DPism_USE_PROJ [other options] ..
```

Building PISM with libraries in non-standard locations

To build PISM with libraries installed in a non-standard location such as `~/local/`, use CMake's variable `CMAKE_FIND_ROOT_PATH`. Set it to a semicolon-separated list of directories.

For example, if `netcdf.h` is located in `~/local/netcdf/include/` and `libnetcdf.so` is in `~/local/netcdf/lib`, add `~/local/netcdf` to `CMAKE_FIND_ROOT_PATH`:

```
cmake -DCMAKE_FIND_ROOT_PATH=~/local/netcdf [other options] ..
```

To build PISM using parallel I/O libraries installed as described in *Installing parallel I/O libraries*, do this:

```
cmake -DCMAKE_FIND_ROOT_PATH="~/local/netcdf;~/local/pnetcdf;~/local/parallelio" \
-DPism_USE_PNETCDF \
-DPism_USE_PARALLEL_NETCDF \
-DPism_USE_PIO \
..
```

2.3.7 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.

- 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" by adding `-DPETSC_EXECUTABLE_RUNS=YES` to `cmake` options.

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 [PISM builds](#) for examples.

2.4 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):

```
pismr -eisII A -y 5000 -view thk,tempabase,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 pismr -eisII A -y 5000 -view thk,tempabase,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 of the output should say

```
100% tests passed, 0 tests failed out of XX
```

or similar.

Feel free to [e-mail us](#) about any test failures you see. Please run

```
ctest --output-on-failure > make-test.log
```

and send us the `make-test.log` that this produces.

2.5 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.

2.6 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 version 3.0 or newer	needed to rebuild this Manual
sphinxcontrib.bibtex	needed to rebuild this Manual
LaTeX	needed to rebuild the PDF version of this Manual
Latexmk	needed to rebuild the PDF version of this Manual
ncgen from NetCDF	needed to rebuild this Manual
netcdf4-python	needed to rebuild this Manual
doxygen	required to rebuild the PISM <i>Source Code Browser</i>
graphviz	required to rebuild the PISM <i>Source Code Browser</i>

On a Debian-based system you may be able to install most of these by running

```
sudo apt-get install \
    graphviz \
    texlive texlive-fonts-extra \
    latexmk

pip3 install sphinx sphinxcontrib-bibtex --user
```

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

2.6.1 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.

2.6.2 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.

2.6.3 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

<https://www.pism.io>

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 or join the [PISM workspace on Slack](#).

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 [66].

We use data assembled by the [Sea-level Response to Ice Sheet Evolution \(SeaRISE\)](#) assessment process [62]. SeaRISE is a community-organized assessment process that provided an upper bound on ice sheet contributions to sea level in the next 100–200 years 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 \sim 500 m grids for whole-Greenland ice sheet modeling) possible by exploiting large-scale parallel processing. See [66], [141], [99], 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 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 [142], and modeled precipitation and surface mass balance rates from RACMO [63], 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 [8] and `pism_dSL.nc` has the SPECMAP sea level record [9].

Any of these NetCDF files can be viewed with `ncview` or other NetCDF visualization tools; see [Table 3.17](#). 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.

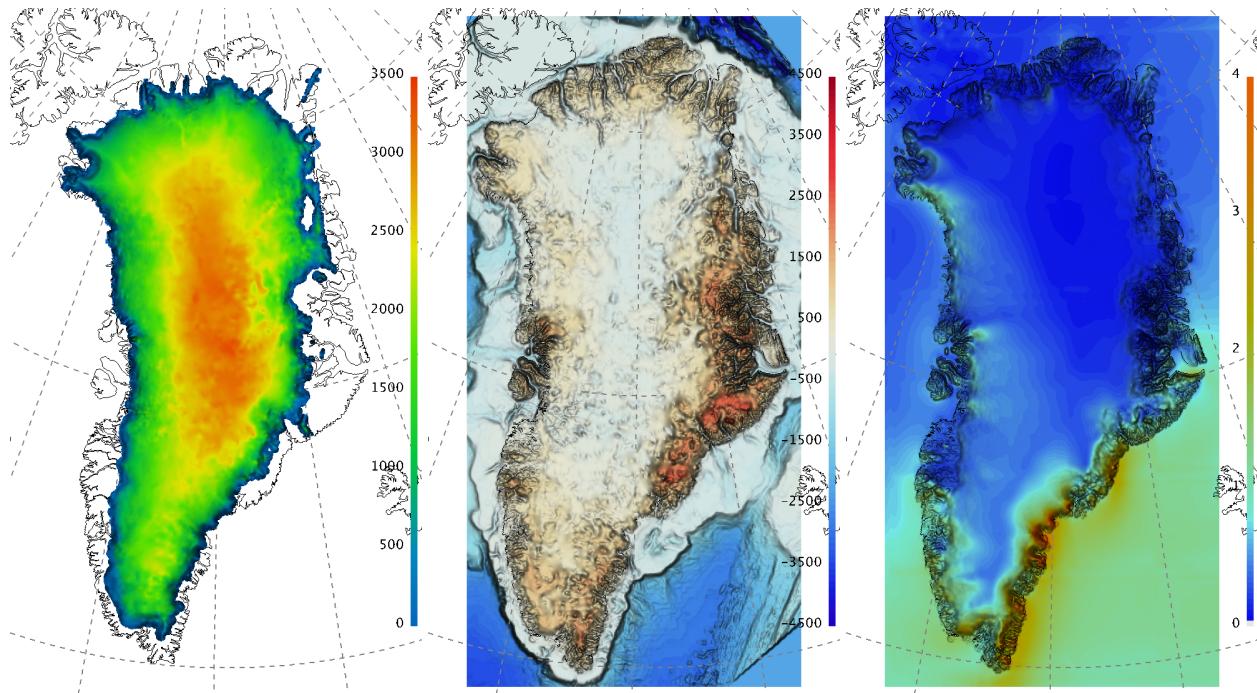


Fig. 3.1: The input file contains present-day ice thickness (left; m), bedrock elevation (center; m), and present-day precipitation (right; m/year ice equivalent) for SeaRISE-Greenland. These are fields `thk`, `topg`, and `precipitation`, respectively, in `pism_Greenland_5km_v1.1.nc`.

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:

```
mpixexec -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.recompute_longitude_and_latitude false -periodicity none -ys -10000 -ye 0 \
-surface given -surface_given_file pism_Greenland_5km_v1.1.nc \
-front_retreat_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,tempabase,tempicethk_basal,bmelt,tillwat,velsurf_mag,mask,thk,topg,
-usurf \
-o g20km_10ka.nc
```

Let’s briefly deconstruct this run.

At the front is “`mpixexec -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 [29], [140]. 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 leaving the area covered by ice (or ice-free with the bed above the sea level) according to the input file (“`-front_retreat_file`”); see section [Calving and front retreat](#) 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 [22], 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 [51], [44], among others). An important aspect of modeling such sliding is the inclusion of membrane or “longitudinal” stresses into the stress balance [29]. The basic stress balance in PISM which involves membrane stresses is the Shallow Shelf Approximation (SSA) [42]. The stress balance used in the previous section was, by contrast, the (thermomechanically-coupled) non-sliding, non-membrane-stress Shallow Ice Approximation (SIA) [30], [22]. 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 [29], [37]. 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 [66], [103], [109], [65], among others). In this regard, PISM uses the concept of a saturated and pressurized subglacial till with a modeled distribution of yield stress [29], [45]. The yield stress arises from the PISM model of the production of subglacial water, which is itself computed through the conservation of energy model [35]. 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 [146], 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 `surfvvelmag` 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 [145] and Figure 8 in [144]. 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 [30], 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 “`mpexec -n 4`” to “`mpexec -n N`” where N is a substantially larger number, up to 100 or so with an expectation of reasonable scaling on this grid [29], [140].

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 [148]). 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 [147] (inversion of DEMs for basal topography) and [67] (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. [149] 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 [23]. 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 [63], 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 [8] for GRIP), along with estimates of past global sea level [9] 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 [23]. 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 $2.9 \times 10^6 \text{ km}^3$ [142], 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 [27].

To see how all this translates into PISM options, run

```
export PARAM_PPQ=0.5
export REGRIDFILE=g20km_10ka_hy.nc
PISM_D0#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,precip_scaling -surface pdd \
-atmosphere_precip_scaling_file pism_dT.nc -atmosphere_delta_T_file pism_dT.nc
```

Then the sea level forcing module providing both a time-dependent sea level to the ice dynamics core, is turned on with `-sea_level_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.

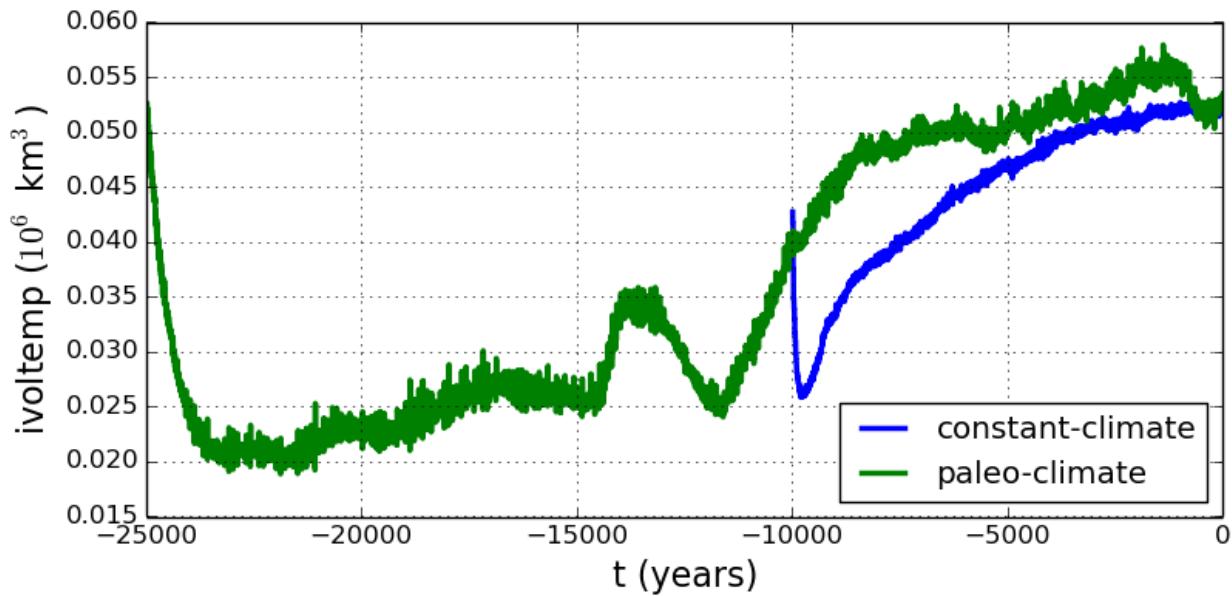


Fig. 3.9: Time series of temperate ice volume `ice_volume_glacierized_temperate` from constant-climate (blue; `ts_g20km_10ka_hy.nc`) and paleo-climate (green; `ts_g20km_25ka_paleo.nc`) spinup runs. The cold of the last ice age affects the fraction of temperate ice. Note different volume scale compared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

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 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 [139], 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](#).) 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 \
    -bootstrap -i pism_Greenland_5km_v1.1.nc \
    -Mx 301 -My 561 \
    -Mz 201 -Lz 4000 -z_spacing equal \
    -Mbz 21 -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 reso-

lution 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 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 km → 10 km → 5 km, with durations of 10000, 2000, and 200 years, 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 km → 10 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).

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.



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. 3.10, 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: 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`.

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 [66]; 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.

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 [140]. 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 five 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 [80].

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

- $q = \text{basal_resistance.pseudo_plastic.q}$: 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
- $e = \text{stress_balance.sia.enhancement_factor}$: values larger than one give flow “enhancement” by making the ice deform more easily in shear than is determined by the standard flow law [77], [76]; 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
```

(continues on next page)

¹ See *Configuration parameters* for the full list.

(continued from previous page)

```

for PPQ in 0.1 0.5 1.0 ; do
    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/paramssubmit.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 [66], or
2. compute the model-observed differences between the histogram of the number of cells with a given surface speed [143].

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 [67], [144], [145], 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 Stress balance models: SIA, SSA, and the First Order Approximation

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 [31]. PISM does not attempt to solve the Stokes equations themselves, however. Instead it can numerically solve, in parallel, three different shallow approximations which are well-suited to ice sheet and ice shelf systems:

- the non-sliding shallow ice approximation (SIA) [41], also called the “lubrication approximation” [31], 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) [42], which describes a membrane-type flow of floating ice [43], or of grounded ice which is sliding over a weak base [44], [45].
- a first order approximation to the Stokes equations due to Blatter ([38], [39]). In the remainder, we refer to is as the “Blatter’s model.”

The SIA equations are easier to solve numerically than the SSA and Blatter’s model, 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 [46]. 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 [31]. 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 [36], [4], [47] additions of ad hoc “sliding laws” into the SIA stress balance, and especially schemes which “switch on” at the pressure-melting temperature [22], have bad continuum [48] and numerical (see [29], 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 [43], [49]. 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” [50]. Such features appear at the margin of, and sometimes well into the interior of, the Greenland [51] and Antarctic [52] 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 [45] 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 [29], [37]. 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 [53], [3], [37]; 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 [54] 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 [44]. 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 [45] model of ice streams, using a plastic (Coulomb) sliding relation. In this model ice streams appear where there is “till failure” [46], 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 [31]. For the corresponding SSA argument, see [42] or the appendices of [45]. Schoof and Hindmarsh [55] have analyzed the connections between these shallowest models and higher-order models, while [56] 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 implemented in PISM one might use the Stokes equations, or a “higher-order” model (e.g. Blatter’s model [38], [39]), 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 [57].

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	• bed elevation
<i>balance velocities</i>	<i>steady state</i> ; ice flows down surface gradient [32]	<i>same as above, plus:</i> • surface mass balance • (initial) ice thickness
isothermal SIA	non-sliding lubrication flow, fixed softness [33], [34]	<i>same as above, but time-dependence is allowed</i>
thermo-coupled SIA	non-sliding lubrication flow, temperature-dependent softness [30], [22]	<i>same as above, plus:</i> • surface temperature • geothermal flux
polythermal SIA	allows liquid water fraction in temperate ice; conserves energy better [35], [36]	<i>same as above</i>
SIA + SSA hybrid	SSA as a sliding law for thermo-coupled SIA [29], [37]; polythermal by default	<i>same as above, plus:</i> • model for subglacial water • model for basal resistance
First Order Approximation	pressure within the ice is hydrostatic; x and y derivatives of the vertical velocity component are small compared to z derivatives of horizontal components [38], [39], [40]	<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 [29], [30].



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.

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 the slowness of the ice, in the sense that inertia can be neglected in the stress balance [31], 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 [28] for terminology and [23] 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)	optional; typically: air temperature, precipitation
top ice surface, but below firn (green)	required: boundary temperature (or enthalpy), mass flux (SMB) into the ice
ice shelf basal surface (blue)	required: mass flux into the ocean, boundary temperature
bottom surface of thermally-modeled bedrock layer (not shown)	required: 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 from a NetCDF file, or
- option `-i -bootstrap` reads an “incomplete” NetCDF file and uses heuristics to fill in needed fields, or
- `pismr -eisII ...` and the `pismv` executable are used to initialize simplified-geometry experiments and 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` 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`”:

```
pismr -eisII A -y 100 -o foo.nc
pismr -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 `pismr -eisII A` 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.

-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:

```
pismr -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 then either

- interpolating some of the missing fields from a separate file, and
- letting essentially heuristic models fill in remaining ones.

So, "bootstrapping" is used whenever some fields are missing *or* interpolation is necessary, for example when going to a finer grid during grid sequencing.

These steps are performed 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.
 1. *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,
 2. *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 input 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. In PISM's regional mode, 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, selected by setting `bootstrapping.temperature_heuristic` to `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.

For example, 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.
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 addition to this, one has to choose the computational grid, the modeling domain, and so on.

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.

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

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 performs 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 measured relative to a reference geoid, is given by

$$z' = z + b(x, y),$$

where $b(x, y)$ is the bed topography. The top surface of the ice $h(x, y)$ is described by $h(x, y) = H(x, y) + b(x, y)$, where $H(x, y)$ is the ice thickness.

In the floating case, the physical vertical coordinate is

$$z' = z + z_{sl} - \frac{\rho_i}{\rho_w} H(x, y) \quad (3.1)$$

where ρ_i is the density of ice, ρ_w the density of sea water, and z_{sl} is the sea level elevation. Again, the physical elevation of the bottom (top) surface of the ice relative to the geoid can be computed by substituting $z = 0$ ($z = H(x, y)$) in (3.1).

Here the *floatation criterion* $z_{sl} - \frac{\rho_i}{\rho_w} H(x, y) > b(x, y)$ 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
<code>-Lx (km)</code>	Half-width of the computational domain (in the x -direction)
<code>-Ly (km)</code>	Half-width of the computational domain (in the y -direction)
<code>-Lz (meters)</code>	Height of the computational domain; must exceed maximum ice thickness
<code>-Lbz (meters)</code>	Depth of the computational domain in the bedrock thermal layer
<code>-x_range A,B (meters)</code>	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 in PISM's regional mode.
<code>-y_range A,B (meters)</code>	Specify the range of y coordinates in PISM's regional mode.

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; set using `grid.ice_vertical_spacing` at bootstrapping. The bedrock thermal layer model always uses equal vertical spacing.

The grid is described by four numbers, namely the number of grid points `grid.Mx` in the x direction, the number `grid.My` in the y direction, the number `grid.Mz` in the z direction within the ice, and the number `grid.Mbz` in the z direction within the bedrock thermal layer. These are specified by options `-Mx`, `-My`, `-Mz`, and `-Mbz`, 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.

The lowest grid point in a grounded column of ice, at $z = 0$, coincides with the highest grid point in the bedrock, so `grid.Mbz` must always be at least one.

Some PISM components (currently: the Blatter stress balance solver) use a geometry-following vertical grid with uniform vertical spacing within each column. See Fig. 3.19.

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 `grid.Mbz` is unrelated to the bed deformation model (glacial isostasy model); see section [Earth deformation models](#).



Fig. 3.18: PISM’s vertical grid with uniform z spacing. See [On the vertical coordinate in PISM, and a critical change of variable](#) for details.

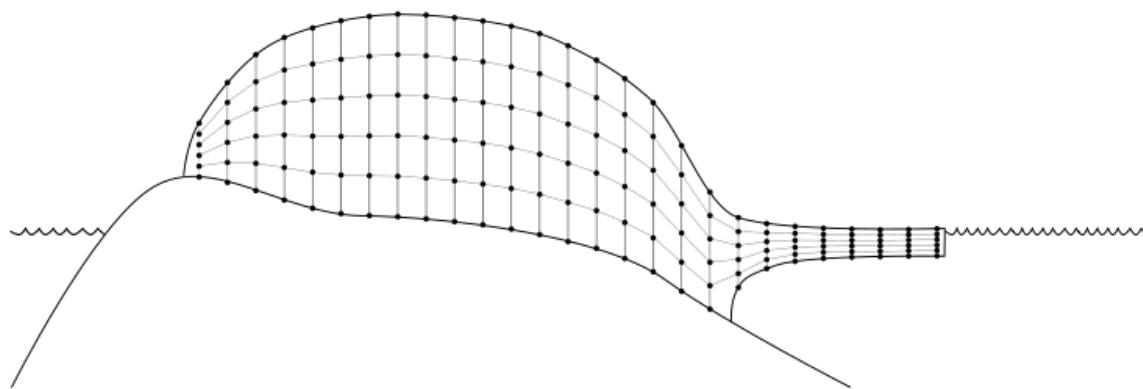


Fig. 3.19: The “sigma” vertical grid used by PISM’s Blatter solver

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 is 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. 3.20.



Fig. 3.20: 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.

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.”

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

$$\Delta x = x_1 - x_0$$

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

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:

```
> pismr -eisII A -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

$$\Delta x = \frac{2L_x}{M_x},$$

$$x_{\min} = x_c - L_x + \frac{1}{2}\Delta x,$$

$$x_{\max} = x_c + L_x - \frac{1}{2}\Delta x,$$
(3.2)

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. 3.20. 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

```
> pismr -eisII A -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

$$\Delta x = \frac{2L_x}{M_x - 1},$$

$$x_{\min} = x_c - L_x,$$

$$x_{\max} = x_c + L_x.$$
(3.3)

See Fig. 3.21 for an illustration.

To switch between (3.2) and (3.3), set the configuration parameter *grid.registration*.



Fig. 3.21: Computational grids using the `corner` grid registration.

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

Grid projections

PISM can use the `PROJ` library (see [Required tools and libraries](#)) and projection information to compute

- 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 and add the global attribute `proj` 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 :proj
:proj = "+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 the code re-computing longitude, latitude grid coordinates using projection information to avoid the dependency on PROJ (look for `-grid.recompute_longitude_and_latitude` 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.recompute_longitude_and_latitude 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 longitude and latitude using projection parameters...
...
... done with run
Writing model state to file `output.nc'...
```

If the `proj` 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 latitude/longitude coordinates.

To simplify post-processing and analysis with CDO PISM adds the PROJ string (if known) to the mapping variable, putting it in the `proj_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:proj_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 `mpiexec -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,

```
mpiexec -n 3 pismr -eisII A -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 *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 of the run is the start time plus the given (`-y`) run length. If both `-ys` and `-ye` are used then the run length is set to the difference. If both `-y` and `-ye` are set PISM will use `-ye` and ignore `-y`.

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

Table 3.4: Command-line options controlling PISM time

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

Times specified by -ys and -ye are interpreted as *years since the reference date in the chosen calendar* (see [time.reference_date](#) and [time.calendar](#)), so by default -ys 0 will correspond to January 1 of year 1.

It is also possible to provide *units*: for example, -ys 1day -ye 2weeks will start the run from January 2, 1 and run for 14 days (until January 15).

In addition to this, -ys and -ye (as well as -extra_times, etc) can take *dates* of the form Y-M-D, so -ys 2000-1-1 -ye 2100-1-1 would tell PISM to run for 100 years from January 1, 2000.

The run length given by -y is interpreted as the number of 365-day years. For more precise control of the run duration, provide units (e.g. -y 1s for a very short run or -y "1000 360day" to run for one thousand 360-day years) or use -ye instead.

Note: The fact that -y is not calendar-aware can lead to some confusion. Note, for example, that -ys 2000-1-1 -y 100 -calendar 360_day will end the run on 2101-05-21: this is the date in the 360-day calendar that is one hundred 365-day years from 2000-1-1.

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.

Note: When preparing input files it is important to use time 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.

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 (see [time.calendar](#)). 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.

¹ Note seconds are part of SI units.

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 addition to setting run start and end times, the calendar setting also affects the interpretation of “monthly” and “yearly” reporting with `-extra_times`, `-ts_times`, and `-save_times` (see *Spatially-varying diagnostic quantities*, *Scalar diagnostic quantities*, and *Snapshots of the model state*).

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
<code>gregorian</code> or <code>standard</code>	Mixed Gregorian/Julian calendar used today.
<code>proleptic_gregorian</code>	Gregorian calendar extended to dates before 1582-10-15.
<code>noleap</code> or <code>365_day</code>	Calendar with fixed-length 365-day years
<code>360_day</code>	Calendar with fixed-length 360-day years divided into 30-day months
<code>julian</code>	Julian calendar

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 1 hours (`-y 10hours`). 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:

```
pismr -eisII A -y 6000 -o foo.nc
pismr -i foo.nc -y 10hours -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 10hours`” 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 [68]. 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 [30]. 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 [31].

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.

In PISM the SSA is also used to describe the sliding of grounded ice and the formation of ice streams [29]. 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 [45], 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 the preferred “sliding law” for the SIA [29], [37]. The SSA should be combined with the SIA, in this way, in preference to classical SIA sliding laws which make the sliding velocity of ice 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 [37]. 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.

In addition to this, PISM includes an implementation of the first order approximation of Stokes equations due to Blatter (`-stress_balance blatter`, [38], [39]).

All stress balance options *except* for the first order approximation correspond to two basic choices:

- modeling basal sliding, and
- modeling of ice velocity within an ice column.

PISM supports the following stress balance choices, controlled using `stress_balance.model` (option `-stress_balance`):

1. **none**: no sliding, ice velocity is constant in each column. This equivalent to disabling ice flow completely.
2. **prescribed_sliding**: Use the constant-in-time prescribed sliding velocity field read from a file set using `stress_balance.prescribed_sliding.file`, variables `ubar` and `vbar`. Horizontal ice velocity is constant throughout ice columns.
3. **ssa**: Use the *Shallow shelf approximation (SSA)* model exclusively. Horizontal ice velocity is constant throughout ice columns.

4. `weertman_sliding`: basal sliding is approximated using the *Weertman-style sliding law*, ice velocity is constant throughout ice columns.
5. `sia (default)`: no sliding; ice velocity within the column is approximated using the *Shallow ice approximation (SIA)*. Floating ice does not flow, so this model is not recommended for marine ice sheets.
6. `prescribed_sliding+sia`: basal ice velocity is read from an input file and held constant, ice velocity within the column is approximated using the *Shallow ice approximation (SIA)*.
7. `ssa+sia`: use *Shallow shelf approximation (SSA)* as a sliding law with a plastic or pseudo-plastic till, combining it with the *Shallow ice approximation (SIA)* according to the combination in [37]; similar to [29]. Floating ice uses SSA only. *This “hybrid” stress balance is the recommended sliding law for the SIA.*
8. `weertman_sliding+sia`: basal sliding is approximated using the *Weertman-style sliding law*, ice velocity within the column is approximated using the *Shallow ice approximation (SIA)*.
9. `blatter`: use *Blatter’s model*.

Please see the following sections for details.

Shallow shelf approximation (SSA)

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 3.6, 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 3.7. 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.

When using SSA as a “sliding law” one also needs to model the yield stress, or a pseudo-yield-stress in the case of power law sliding (section *Controlling basal strength*).

The basal yield stress is normally a function of the amount of water stored in the till and a (generally) spatially-varying till strength. The amount of stored basal water is modeled by the subglacial hydrology model (section *Subglacial hydrology*) based on the basal melt rate which is, primarily, thermodynamically-determined (see *Modeling conservation of energy*).

Table 3.6: 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 [29], while <code>fem</code> uses a Newton method. The <code>fem</code> solver has surface velocity inversion capability [67].
<code>-ssa_eps (10¹³)</code>	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: $\nu H \rightarrow \nu H + \epsilon_{SSA}$, 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 3.7: Controls on the numerical iteration of the `-ssa_method fd` solver

Option	Description
<code>-ssafd_picard_maxi</code> (300)	Set the maximum allowed number of Picard (nonlinear) iterations in solving the shallow shelf approximation.
<code>-ssafd_picard_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 $\nu^{(k)}$ of the vertically-averaged effective viscosity satisfy
	$\ (\nu^{(k)} - \nu^{(k-1)})H\ _1 \leq Z\ \nu^{(k)}H\ _1$
	where $Z = \text{ssafd_picard_rtol}$.
<code>-ssafd_ksp_rtol</code> (10^{-5})	Set the relative change tolerance for the iteration inside the Krylov linear solver used at each Picard iteration.
<code>-ssafd_max_speed</code> (50km/yr)	Limits computed SSA velocities: ice speed is capped at this limit after each Picard iteration of the SSAFD solver. This may allow PISM to take longer time steps by ignoring high velocities at a few troublesome locations.

Parameters

Prefix: `stress_balance.ssa`.

1. `Glen_exponent` (3) Glen exponent in ice flow law for SSA
2. `compute_surface_gradient_inward` (no) If yes then use inward first-order differencing in computing surface gradient in the SSA objects.
3. `dirichlet_bc` (no) apply SSA velocity Dirichlet boundary condition
4. `enhancement_factor` (1) Flow enhancement factor for SSA
5. `epsilon` (1e+13 Pascal second meter) 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 $s^{1/3}$ [68] and a typical strain rate of 0.001 1/year for the Ross ice shelf, giving $\nu = (\bar{B})/(2 \cdot 0.001^{2/3}) = 9.49e+14$ Pa s ~ 30 MPa year, the value in [87], but with a tiny thickness H of about 1 cm.
6. `fd.brutal_sliding` (false) Enhance sliding speed brutally.
7. `fd.brutal_sliding_scale` (1) Brutal SSA Sliding Scale
8. `fd.flow_line_mode` (false) Set ν (the y component of the ice velocity) to zero when assembling the system
9. `fd.lateral_drag.enabled` (false) Set viscosity at ice shelf margin next to ice free bedrock as friction parameterization
10. `fd.lateral_drag.viscosity` (5e+15 Pascal second) Staggered viscosity used as side friction parameterization.
11. `fd.max_iterations` (300) Maximum number of Picard iterations for the ice viscosity computation, in the SSAFD object
12. `fd.max_speed` (300000 km s-1) Upper bound for the ice speed computed by the SSAFD solver.
13. `fd.nuH_iter_failure_underrelaxation` (0.8) In event of “Effective viscosity not converged” failure, use outer iteration rule $\text{nuH} \leftarrow \text{nuH} + f (\text{nuH} - \text{nuH}_{\text{old}})$, where f is this parameter.

14. `fd.relative_convergence` (0.0001) Relative change tolerance for the effective viscosity in the SSAFD object
15. `fd.replace_zero_diagonal_entries` (yes) Replace zero diagonal entries in the SSAFD matrix with :config:'basal_resistance.betta_ice_free_bedrock' to avoid solver failures.
16. `flow_law` (gpbld) The SSA flow law.
17. `method` (fd) Algorithm for computing the SSA solution.
18. `read_initial_guess` (yes) Read the initial guess from the input file when re-starting.
19. `strength_extension.constant_nu` (9.48681e+14 Pascal second) The SSA is made elliptic by use of a constant value for the product of viscosity (nu) and thickness (H). This value for nu comes from hardness (bar B)=1.9e8 Pas^{1/3} [68] 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 Pa s = 30 MPa year in [87].
20. `strength_extension.min_thickness` (50 meters) The SSA is made elliptic by use of a constant value for the product of viscosity (nu) and thickness (H). At ice thicknesses below this value the product nu*H switches from the normal vertical integral to a constant value. The geometry itself is not affected by this value.

Technical remarks

Ice stress balance models ignoring inertia are “diagnostic” models that do not have a “state” evolving through time: the ice velocity is fully determined by ice geometry, basal boundary conditions, and the ice viscosity.

In addition to this, shallow stress balance models (other than the SIA) correspond to *nonlinear* systems of equations, which means that computing an estimate of the ice velocity is an iterative process starting with a particular *initial guess*.

The quality of this guess may affect the number of iterations needed and even whether the solver succeeds at all. It also has an influence on the computed solution: if an equation has a unique solution, estimates produced using different initial guesses should be *close*, but they need not be *identical*. If an equation has multiple solutions, even a “small” change in the initial guess may give a completely different result.

In the context of a evolutionary model we can usually assume that the change in the state of the model from one step to the next is “small” and we use (u, v) estimates from one time step as the initial guess for the next. To ensure that stopping and re-starting a simulation does not affect results we save these to the output file as variables `u_ssa` and `v_ssa` and read them in when re-starting a stopped simulation.

One could say that the continuum SSA model does not have a state, but its implementation does. Set `stress_balance.ssa.read_initial_guess` to “false” to ignore it during initialization and use the zero initial guess instead.

Shallow ice approximation (SIA)

Note: The explicit time stepping of the mass continuity equation in the case of the SIA flow comes with a severe restriction on time step length:

$$\Delta t \leq \frac{2R}{D(1/\Delta x^2 + 1/\Delta y^2)} \quad (3.4)$$

Here D is the maximum diffusivity of the SIA flow and R is `time_stepping.adaptive_ratio`, a tuning parameter that further reduces the maximum allowed time step length.

The maximum diffusivity D may be achieved at an isolated grid point near the ice margin. In this case it might make sense to limit the diffusivity of the SIA flow, sacrificing accuracy at a few grid points to increase time step length and reduce the computational cost. Set `stress_balance.sia.limit_diffusivity` to enable this mechanism.

When `stress_balance.sia.limit_diffusivity` is false PISM stops as soon as the SIA diffusivity at any grid point exceeds `stress_balance.sia.max_diffusivity`. We do this to make it easier to detect problematic model configurations: in many cases it does not make sense to continue a simulation if D is very large.

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.

In the SIA model surface gradients at *staggered* grid locations are computed using one of the following three finite-difference approximations (selected using `stress_balance.sia.surface_gradient_method`):

1. `mahaffy`: This most “standard” way computes the surface slope onto the staggered grid for the SIA [81]. It makes $O(\Delta x^2, \Delta y^2)$ errors.
2. `haseloff`: This is the default method. It only differs from `mahaffy` 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.
3. `eta`: 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:

$$\nabla h = \nabla H + \nabla b = \frac{n}{(2n+2)} \eta^{(-n-2)/(2n+2)} \nabla \eta + \nabla b.$$

Here b is the bed elevation, h is the surface elevation, and n is the Glen exponent. 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 [33] for technical explanation of this transformation and compare [82]. The actual finite difference schemes applied to compute the surface slope are similar to option `mahaffy`.

Note: This method may improve the model performance near *grounded* margins but should not be used in simulations of marine ice sheets.

To the best of our knowledge there is no theoretical advice on the best, most robust mechanism.

Parameterization of bed roughness

Schoof [83] 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 are two associated parameters:

- `stress_balance.sia.bed_smoothen.range` gives the half-width of the square smoothing domain in meters. If zero is given then the mechanism is turned off. The mechanism is on by default using executable `pismr`, with the half-width set to 5 km, giving Schoof’s recommended smoothing size of 10 km [83].
- `stress_balance.sia.bed_smoothen.theta_min` is the minimum value of θ in the parameterization.

This mechanism is turned off by default in `pismv`.

Under the default `output.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 `stress_balance.sia.bed_smoothening.theta_min` (usually zero) and 1, with values significantly below one indicating a reduction in diffusivity, essentially a drag coefficient, from bumpy bed.

Coupling to the age of the ice

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 [84] and [85] (Vostok core data). In PISM, only the Goldsby-Kohlstedt flow law (see *Ice rheology*) uses the grain size.
Set `stress_balance.sia.grain_size_age_coupling` to enable this parameterization.
2. The flow enhancement factor can be coupled to the age of the ice as in [86]: e in (3.12) 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`.

Set `stress_balance.sia.e_age_coupling` to enable this parameterization.

Parameters

Prefix: `stress_balance.sia`.

1. `Glen_exponent` (3) Glen exponent in ice flow law for SIA
2. `bed_smoothening.range` (5000 meters) half-width of smoothing domain in the bed roughness parameterization for SIA [83]; set to zero to disable
3. `bed_smoothening.theta_min` (0) minimum value of θ in the bed roughness parameterization for SIA [83]
4. `e_age_coupling` (no) Couple the SIA enhancement factor to age as in [36].
5. `enhancement_factor` (1) Flow enhancement factor for SIA
6. `enhancement_factor_interglacial` (1) Flow enhancement factor for SIA; used for ice accumulated during interglacial periods.
7. `flow_law` (`gpbl1d`) The SIA flow law.
8. `grain_size_age_coupling` (no) Use age of the ice to compute grain size to use with the Goldsby-Kohlstedt [25] flow law
9. `limit_diffusivity` (no) Limit SIA diffusivity by `stress_balance.sia.max_diffusivity`.
10. `max_diffusivity` (100 $m^2 s^{-1}$) Maximum allowed diffusivity of the SIA flow. PISM stops with an error message if the SIA diffusivity exceeds this limit.
11. `surface_gradient_method` (`haseloff`) method used for surface gradient calculation at staggered grid points

Weertman-style sliding law

Warning: This kind of sliding is, in general, a bad idea. We implement it to simplify comparisons of the “hybrid” model mentioned above to older studies using this parameterization.

The “Weertman-type sliding law” ([56], equations 5.35 and 5.91) has the form

$$\mathbf{u}_s = \begin{cases} \mathbf{0}, & T_b < T_m, \\ -C_b(\rho g H)^{p-q} |\nabla h|^{p-1} \nabla h, & T_b = T_m, \end{cases}$$

T_b is the ice temperature, and T_m is the pressure-melting temperature. The constant C_b and exponents p and q are tuning parameters.

The particular form implemented in PISM comes from equation 5 in [88]:

$$\mathbf{u}_s = -\frac{2A_s \beta_c (\rho g H)^n}{N - P} |\nabla h|^{n-1} \nabla h. \quad (3.5)$$

Table 3.8: Notation used in (3.5)

Variable	Meaning
H	ice thickness
h	ice surface elevation
n	flow law exponent
g	acceleration due to gravity
ρ	ice density
N	ice overburden pressure, $N = \rho g H$
P	basal water pressure
A_s	sliding parameter
β_c	“constriction parameter” capturing the effect of valley walls on the flow; set to 1 in this implementation

We assume that the basal water pressure is a given constant fraction of the overburden pressure: $P = kN$. This simplifies (3.5) to

$$\mathbf{u}_s = -\frac{2A_s}{1-k} (\rho g H |\nabla h|)^{n-1} \nabla h.$$

This parameterization is used for grounded ice *where the base of the ice is temperate*.

To enable, use `-stress_balance weertman_sliding` (this results in constant-in-depth ice velocity) or `-stress_balance weertman_sliding+sia` to use this parameterization as a sliding law with the deformational flow modeled using the SIA model.

Use configuration parameters `stress_balance.weertman_sliding.k` and `stress_balance.weertman_sliding.A` to set k and A_s , respectively. Default values come from [88].

Parameters

Prefix: `stress_balance.weertman_sliding`.

1. `A` (1.8e-16 Pa-3 year-1 m-2) Sliding parameter in the Weertman-style sliding parameterization [88]
2. `k` (0.2) The ratio of the basal water pressure and the ice overburden pressure in the Weertman-style sliding parameterization.

Blatter's model

Unlike the rest of PISM, the Blatter solver uses a geometry-following vertical grid (see Fig. 3.19) to approximate horizontal components of ice velocity. The number of vertical “levels” in this grid is controlled by `stress_balance.blatter.Mz`.

The non-linear system resulting from the discretization of PDEs corresponding to Blatter's stress balance model is much harder to solve than the one corresponding to the SSA system ([57], [69]) and (at this point) experimentation with preconditioner choices seems inevitable. We use PETSc's command-line options to control these choices.

Note: The Blatter solver uses the `-bp_` command-line option prefix.

Run PISM like this

```
pismr -stress_balance blatter \
[other options] -help | grep "-bp_"
```

to see the complete list of PETSc option controlling this solver.

The multigrid (MG) preconditioner using semi-coarsening in the vertical direction followed by further (horizontal) coarsening using algebraic multigrid methods appears to be effective [69]. The option combination

```
-bp_pc_type mg \
-bp_pc_mg_levels N \
-bp_mg_levels_ksp_type gmres \
-bp_mg_coarse_pc_type gamg
```

roughly corresponds to this approach (see *Practical preconditioners choices* for more).

Unlike [69], who used a purely algebraic approach, these options select a combination of geometric and algebraic multigrid preconditioners.

To use a multigrid preconditioner the user has to specify

- the number of MG levels N using `-bp_pc_mg_levels N`,
- the coarsening factor C by setting `stress_balance.blatter.coarsening_factor`, and
- the vertical grid size M_z (`stress_balance.blatter.Mz`).

The values of these parameters have to be compatible. Specifically, M_z has to have the form

$$M_z = A \cdot C^{N-1} + 1 \quad (3.6)$$

for some positive integer A .

Note: PISM stops with an error message if (3.6) is not satisfied.

To set up a multigrid preconditioner PISM needs to build a hierarchy of vertical grids¹ with M_z points on the finest grid.. Starting with this grid, PISM creates the next one by dividing the number of vertical *spaces* by the coarsening factor C . Then the newly-created grid is coarsened and this process is continued, stopping when the desired number N of grids (MG levels) reached.

Overall, the number of points M_z^k in the vertical grid number k in the hierarchy is

$$M_z^0 = M_z, \\ M_z^k = (M_z^{k-1} - 1) / C + 1.$$

This process explains the compatibility condition (3.6): the number of **spaces** in all vertical grids in the hierarchy *except for the coarsest one* has to be divisible by C .

Table 3.9: Some vertical grid hierarchies

Coarsening factor C	Possible sizes of vertical grids in a hierarchy
2	2, 3, 5, 9, 17, 33, 65 , 129, 257, 513, 1025, ...
3	2, 4, 10, 28, 82, 244, 730, ...
4	2, 5, 17, 65 , 257, 1025, ...
5	2, 6, 26, 126, 626, 3126, ...
6	2, 7, 37, 217, 1297, ...
7	2, 8, 50, 344, 2402, ...
8	2, 9, 65 , 513, 4097, ...

By default $C = 2$, but *aggressive coarsening* (i.e. larger values of C , up to around 8) has been observed to work. As highlighted in Table 3.9, sometimes the same number of vertical grid levels can be achieved using more than one combination of the coarsening factor and the number of MG levels.

For example, we can set up a solver using 65 vertical levels and 3 MG levels with the coarsening factor of 8, or 4 MG levels and the factor of 4, or 7 MG levels and the coarsening factor of 2. In general, the computational cost of an MG preconditioner application increases with the number of MG levels, so the first hierarchy (2, 9, 65, $C = 8$) may be the best choice. However, coarsening that is too aggressive may make a less effective preconditioner, requiring more Krylov iterations and increasing the computational cost. Again, one may have to experiment to find settings that work best in a particular setup.

The coarsest grid in a hierarchy should be as small as possible (corresponding to $A = 1$ in (3.6)); two levels is the minimum achievable in the context of the finite element method used to discretize the system (this corresponds to a mesh that is just one element thick).

Surface gradient computation

Some synthetic geometry experiments with grounded margins show “checkerboard” artifacts in computed ice velocity near steep margins. A similar issue and an attempt to address it are described in [70].

This implementation takes a different approach: instead of using an “upwinded” finite difference approximation of the surface gradient we allow using the η transformation described in [Surface gradient method](#). Set `stress_balance.blatter.use_eta_transform` to enable it.

¹ Horizontal coordinates of grid points are the same in all grids in a hierarchy, i.e. each grid is “extruded” from PISM’s 2D grid with uniform spacing in x and y directions.

Adaptive time stepping

PISM's explicit in time mass continuity code is *conditionally stable*. When used with the SSA + SIA hybrid, the maximum allowed time step is computed using a combination of the CFL criterion [71] and the maximum diffusivity of the SIA flow [29]. This time step restriction does not disappear when the same mass continuity code is used with a stress balance model that does not explicitly compute “advection” and “diffusion” parts of the flow. We need a work-around.

Note: Very little is known about stability of explicit time stepping methods of the mass continuity equation coupled to a “generic” stress balance model.

We don't have a rigorous justification for the approach described below.

When this BP solver is coupled to PISM, the vertically-averaged ice velocity is used in place of the “advection” (“sliding”) velocity from the SSA. As a result, the CFL-based time step restriction is applied by existing PISM code.

However, it is almost always the case that the diffusivity-driven time step restriction is more severe and so we need a replacement: CFL alone does not appear to be sufficient for stability.

We compute an estimate of the “SIA-like” maximum diffusivity by observing that for the SIA the vertically-averaged ice flux Q satisfies

$$Q = -D \nabla s.$$

We solve this for the diffusivity D :

$$D = \frac{H |\bar{u}|}{|\nabla s| + \epsilon} \tag{3.7}$$

and use the maximum of this quantity to determine the maximum allowed time step using (3.4).

Note: Other models supporting this stress balance model and using an explicit in time geometry evolution method ([70], [72]) report that the CFL condition appears to be sufficient in practice.

Given the lack of a theory describing the maximum time step necessary for stability it may make sense to experiment with *increasing `time_stepping.adaptive_ratio`*.

Setting it to a very large value would *completely disable* the diffusivity-based time step restriction.

Note: The “time step skipping mechanism” enabled using `time_stepping.skip.enabled` (see *Understanding adaptive time-stepping*) has a different effect when the Blatter stress balance model is used: the full 3D ice velocity is updated during every sub-step and only the energy balance and age models takes the “long” time step.

Since the Blatter solver is likely to dominate the computational cost, setting `time_stepping.skip.enabled` to “true” is not likely to be beneficial.

Practical preconditioners choices

The option combination

```
-bp_pc_type mg \
-bp_pc_mg_levels N \
-bp_mg_levels_ksp_type gmres \
-bp_mg_coarse_pc_type gamg
```

sets up the *kind* is a multigrid preconditioner known to be effective, but it is not the only one, and most likely not the best one.

Our experiments suggest that

```
-bp_pc_type mg \
-bp_pc_mg_levels N \
-bp_snes_ksp_ew \
-bp_snes_ksp_ew_version 3 \
-bp_mg_levels_ksp_type richardson \
-bp_mg_levels_pc_type sor \
-bp_mg_coarse_ksp_type gmres \
-bp_mg_coarse_pc_type hypre \
-bp_mg_coarse_pc_hypre_type boomeramg
```

may work better², but requires PETSc built with [hypre](#).

Here `-bp_snes_ksp_ew -bp_snes_ksp_ew_version 3` enables Luis Chacón's variant of the Eisenstat-Walker [73] method of adjusting linear solver tolerances to avoid oversolving and `-bp_mg_coarse_pc_type hypre -bp_mg_coarse_pc_hypre_type boomeramg` selects the BoomerAMG algebraic MG preconditioner from [hypre](#) for the coarse MG level.

Note: The Eisenstat-Walker adjustment of linear solver tolerances saves time when a low-accuracy estimate of the Newton step is sufficient but may lead to solver failures, especially when the initial guess is of poor quality. In an attempt to reduce computational costs while maintaining robustness PISM disables `-bp_snes_ksp_ew` if the initial guess is zero (beginning of a simulation) or if the solver fails with `-bp_snes_ksp_ew`.

Some simulations may benefit from using a direct solver on the coarse MG level. For example, the following would use [MUMPS](#) on the coarse grid:

```
-bp_pc_type mg \
-bp_pc_mg_levels N \
-bp_snes_ksp_ew \
-bp_snes_ksp_ew_version 3 \
-bp_mg_levels_ksp_type richardson \
-bp_mg_levels_pc_type sor \
-bp_mg_coarse_ksp_type preonly \
-bp_mg_coarse_pc_type lu
```

if PETSc is built with [MUMPS](#).

Note: Parallel direct solvers such as MUMPS really benefit from using optimized BLAS and LAPACK libraries.

Please see section 3.5.3 of [26] for instructions. At the time of writing

² These settings are inspired by [57].

```
--download-f2cblaslapack --download-blis
```

is recommended as a portable high-performance option. However, it makes sense to try other freely-available libraries (Intel MKL, OpenBLAS) as well.

Note, though, that the multigrid preconditioner, even if it is effective in terms of reducing the number of Krylov iterations, may not be the cheapest one [74]: there is a trade off between the number of iterations and the cost of a single iteration. Other preconditioner options may be worth considering as well.

In some cases node ordering and the way the domain is split among processes in a parallel run may affect solver performance (see [57], [74], [69]). These references mention staggering the unknowns so that u and v components at the same node correspond to adjacent equations in the system and using contiguous ordering of unknowns in the same ice column. This allows the solver to capture vertical coupling *locally* using incomplete factorization.

In addition to this, [74] mention that parallel domain distribution partitioning ice columns among multiple processes *sometimes* leads to convergence issues. Following this advice, PISM does not partition the domain in the z direction, but some of our experiments show that if the solver struggles, switching to a *one-dimensional* domain decomposition along the y direction may help (see [Parallel domain distribution](#)).

Run PISM as follows to give this a try:

```
mpiexec -n M pismr -Nx 1 -Ny M ...
```

This forces PISM to split the domain into M parts in the y direction instead of the default (approximately \sqrt{M} in both x and y).

Please see [Blatter stress balance solver: technical details](#) for more.

Parameters

Below is the complete list of configuration parameters controlling this solver (prefix: `stress_balance.blatter.`):

1. [Glen_exponent](#) (3) Glen exponent in ice flow law for the Blatter stress balance solver
2. [Mz](#) (5) Number of vertical grid levels in the ice
3. [coarsening_factor](#) (2) Coarsening factor in the z direction
4. [enhancement_factor](#) (1) Flow enhancement factor for the Blatter stress balance flow law
5. [flow_law](#) (gpbl1d) The flow law used by the Blatter-Pattyn stress balance model
6. [use_eta_transform](#) (no) Use the η transform to improve the accuracy of the surface gradient approximation near grounded margins (see [33] for details).

Ice rheology

Contents

- [Ice rheology](#)
 - [Flow law choices](#)
 - [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 [46]. 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

$$D_{ij} = F(\sigma, T, \omega, P, d) \sigma'_{ij}. \quad (3.8)$$

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 (3.8) of the flow law is used in the SIA, but the “viscosity form” of a flow law, found by inverting the constitutive relation (3.8), is needed for ice shelf and ice stream (SSA) flow and the first-order stress balance approximation [29]:

$$\sigma'_{ij} = 2\nu(D, T, \omega, P, d) D_{ij} \quad (3.9)$$

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,

$$F(\sigma, T) = A(T) \sigma^{n-1} \quad (3.10)$$

is the common temperature-dependent Glen law [76], [30] (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 [25] 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 (3.10).

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 [35], [77], [76], 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 below which do not use the liquid water fraction.

Flow law choices

Configuration parameters

- `stress_balance.sia.flow_law`,
- `stress_balance.ssa.flow_law`, and
- `stress_balance.blatter.flow_law`

choose which flow law is used by the SIA, SSA, and the Blatter stress balances models, respectively. Allowed arguments are listed below.

1. `gpbld`: Glen-Paterson-Budd-Lliboutry-Duval law [77], the enthalpy-based default in PISM [35]. 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 `flow_law.gpbld.water_frac_coeff`, and ω is capped at level `flow_law.gpbld.water_frac_observed_limit`.

Parameters

This flow law uses all the parameters controlling the Paterson-Budd law, plus the ones listed below.

Prefix: `flow_law.gpbld`.

1. `water_frac_coeff` (181.25) coefficient in Glen-Paterson-Budd flow law for extra dependence of softness on liquid water fraction (ω) [56], [77]
2. `water_frac_observed_limit` (0.01) maximum value of liquid water fraction ω for which softness values are parameterized by [77]; used in Glen-Paterson-Budd-Lliboutry-Duval flow law; compare [35]
2. `pb`: 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 $T^* < T_{\text{critical}}$ and

$$A = 1.733 \times 10^3 \text{ s}^{-1} \text{ Pa}^{-3}, \\ Q = 13.9 \times 10^4 \text{ J mol}^{-1}$$

if $T^* > T_{\text{critical}}$. Here T^* is pressure-adjusted temperature [76].

Parameters

Prefix: `flow_law.Paterson_Budd`.

1. `A_cold` (3.61e-13 Pascal-3 / second) Paterson-Budd A_{cold} , see [76]
2. `A_warm` (1730 Pascal-3 / second) Paterson-Budd A_{warm} , see [76]
3. `Q_cold` (60000 Joule / mol) Paterson-Budd Q_{cold} , see [76]
4. `Q_warm` (139000 Joule / mol) Paterson-Budd Q_{warm} , see [76]
5. `T_critical` (263.15 Kelvin) Paterson-Budd critical temperature, see [76]
3. `arr`: *Cold* part of Paterson-Budd. Regardless of temperature, the A and Q values for $T^* < T_{\text{critical}}$ in the Paterson-Budd law apply. This is the flow law used in the thermomechanically-coupled exact solutions run by `pismv -test F` and `pismv -test G` [30], [78].
4. `arrwarm`: *Warm* part of Paterson-Budd. Regardless of temperature, the A and Q values for $T^* > T_{\text{critical}}$ in Paterson-Budd apply.
5. `ooke`: Hooke law with

$$A(T) = A \exp\left(-\frac{Q}{RT^*} + 3C(T_r - T^*)^\kappa\right).$$

Fixed Glen exponent $n = 3$ and constants as in [79], [47].

Parameters

Prefix: `flow_law.Hooke`.

1. `A` (4.42165e-09 Pascal-3 second-1) $A_{\text{Hooke}} = (1/B_0)^n$ where $n=3$ and $B_0 = 1.928 \text{ a}^{1/3} \text{ Pa}$. See [79]
2. `C` (0.16612 Kelvin^k) See [79]
3. `Q` (78800 Joule / mol) Activation energy, see [79]
4. `Tr` (273.39 Kelvin) See [79]
5. `k` (1.17) See [79]
6. `isothermal_glen`: The isothermal Glen flow law.

Here

$$\begin{aligned} F(\sigma) &= A_0 \sigma^{n-1}, \\ v(D) &= \frac{1}{2} B_0 D^{(1-n)/(2n)}, \end{aligned} \tag{3.11}$$

where A_0 is the ice softness and $B_0 = A_0^{-1/n}$ is the ice hardness.

Parameters

Prefix: `flow_law.isothermal_Glen`.

1. `ice_softness` (3.1689e-24 Pascal-3 second-1) ice softness used by the isothermal Glen flow law [34]
7. `gk`: The Goldsby-Kohlstedt flow law. This law has a combination of exponents from $n = 1.8$ to $n = 4$ [25].

Note: The viscosity form (3.9) of this flow law is not known, so it can only be used by the SIA stress balance.

Because it has more than one power, `stress_balance.sia.Glen_exponent` has no effect, though `stress_balance.sia.enhancement_factor` works as expected. This law does not use the liquid water fraction, but only the temperature.

Constants defining this flow law are hard-wired in the implementation. Please see the source code for details.

Enhancement factor and exponent

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

The parameter `stress_balance.sia.enhancement_factor` sets e in

$$D_{ij} = e F(\sigma, T, \omega, P, d) \sigma'_{ij}, \tag{3.12}$$

see (3.8).

Parameters `stress_balance.ssa.enhancement_factor` and `stress_balance.blatter.enhancement_factor` set e in

$$\sigma'_{ij} = e^{-1/n} 2 v(D, T, \omega, P, d) D_{ij}, \tag{3.13}$$

see (3.9).

Parameters `stress_balance.sia.Glen_exponent`, `stress_balance.ssa.Glen_exponent`, `stress_balance.blatter.Glen_exponent` set the exponent when a single-power flow law is used.

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 [80], [76].

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

$$D_0 = E_{old} A \sigma_0^{n_{old}}, \\ D_0 = E_{new} A \sigma_0^{n_{new}}$$

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

$$E_{new} = E_{old} \sigma_0^{n_{old}-n_{new}}. \quad (3.14)$$

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 (3.14).

A corresponding formula applies to changing the enhancement factor for the SSA and Blatter stress balance models.

Note:

1. [66] used $e_{SIA} = 3.0$ for Greenland ice sheet simulations (see the supplement) while [3] used $e_{SIA} = 4.5$ for simulations of the Antarctic ice sheet with PISM-PIK.
 2. [3] used $e_{SSA} = 0.512$ for simulations of the Antarctic ice sheet with PISM-PIK.
-

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 [35] 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 [75].

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 [35]. 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 [29] and verified in [30] 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. Set `age.enabled` to turn it on. A 3D variable `age` will appear in output files. It is read at input if `age.enabled` is set and otherwise it is ignored even if present in the input file. If `age.enabled` is set and the variable `age` is absent in the input file then the initial age is set to `age.initial_value`.

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*) 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

Contents

- *Controlling basal strength*
 - *Choosing the sliding law*
 - * *Lateral drag*
 - *Determining the yield stress*
 - * *Till friction angle heuristic*
 - * *Till friction angle optimization*
 - *Determining the effective pressure*
 - * *Controlling minimum effective pressure*

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 [103]. 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 $\text{tauc} < \text{taud_mag}$ 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 [104] and in EISMINT II experiment H [22], 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 [29], [48]. 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,

$$|\boldsymbol{\tau}_b| \leq \tau_c \quad \text{and} \quad \boldsymbol{\tau}_b = -\tau_c \frac{\mathbf{u}}{|\mathbf{u}|} \quad \text{if and only if} \quad |\mathbf{u}| > 0. \quad (3.15)$$

Equation (3.15) says that the (vector) basal shear stress $\boldsymbol{\tau}_b$ 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

$$\boldsymbol{\tau}_b = -\tau_c \frac{\mathbf{u}}{u_{\text{threshold}}^q |\mathbf{u}|^{1-q}}, \quad (3.16)$$

where $u_{\text{threshold}}$ is a parameter with units of velocity (see below). Condition (3.15) is studied in [45] and [105] in particular, while power laws for sliding are common across the glaciological literature (e.g. see [80], [56]). Notice that the coefficient τ_c in (3.16) has units of stress, regardless of the power q .

In both of the above equations (3.15) and (3.16) we call τ_c the *yield stress*. It corresponds to the variable `tauc` in PISM output files. We call the power law (3.16) a “pseudo-plastic” law with power q and threshold velocity $u_{\text{threshold}}$. At the threshold velocity the basal shear stress $\boldsymbol{\tau}_b$ has exact magnitude τ_c . In equation (3.16), 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 (3.15) is the $q = 0$ case of (3.16).

See [Table 3.10](#) 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 3.10: Sliding law command-line options

Option	Description
-pseudo_plastic	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.
-plastic_reg (m/a)	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.
-pseudo_plastic_q	Set the exponent q in (3.16). The default is 0.25.
-pseudo_plastic_uthreshold (m/a)	Set $u_{\text{threshold}}$ in (3.16). The default is 100 m/a.

Equation (3.16) 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

$$\tau_b = -\beta \mathbf{u} \quad (3.17)$$

(The “ β ” coefficient is also called β^2 in some sources (see [44], 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 1m/s \
-yield_stress constant -tauc beta
```

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 1m/s \
-yield_stress constant \
-tauc C
```

Another alternative is the slip law by [106] that combines processes of hard-bedded sliding (Coulomb behavior) and viscous bed deformation without required knowledge of the bed type. It is defined by

$$\tau_b = -\tau_c \frac{\mathbf{u}}{(|\mathbf{u}| + u_{\text{threshold}})^q |\mathbf{u}|^{1-q}}, \quad (3.18)$$

Set `-regularized_coulomb` to select this sliding law.

The original equation (3) in [106] uses the exponent $q = 1/p$. Otherwise, same configuration parameters can be used as in the pseudo-plastic case, but they should have different values, namely $q=0.2$, $u_{\text{threshold}}=40-80$ m/year⁻¹:

```
-regularized_coulomb \
-pseudo_plastic_q 0.2 \
-pseudo_plastic_uthreshold 50.0
```

The model’s performance should be close to the pseudo-plastic implementation (Eq. (3.16)), although there ought to be slightly more fast sliding and a less diffuse onset of sliding.

Lateral drag

PISM prescribes lateral drag at ice margins next to ground with elevation above the ice. (This is relevant in outlet glaciers flowing through fjords, valley glaciers, and next to nunataks.) Set `basal_resistance`.
`beta_lateral_margin` to control the amount of additional drag at these margins.

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 3.11, 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 [45]. 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 (W_{till} , NetCDF variable `tillwat`; see [Subglacial hydrology](#)), which then determines the effective pressure N_{till} (see below). The value of τ_c is also determined in part by a material property field ϕ , the “till friction angle” (NetCDF variable `tillphi`). These quantities are related by the Mohr-Coulomb criterion [80]:

$$\tau_c = c_0 + (\tan \phi) N_{till}. \quad (3.19)$$

Here c_0 is called the “till cohesion”, whose default value in PISM is zero (see [45], 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 3.11: Command-line options controlling how yield stress is determined

Option	Description
<code>-yield_stress mohr_coulomb</code>	The default. Use equation (3.19) 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 (3.20).
<code>-yield_stress tillphi_opt</code>	Compute the till friction angle ϕ in (3.19) iteratively in an equilibrium simulation using equation (3.21).
<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.

Till friction angle heuristic

We find that an effective, though heuristic, way to determine ϕ in (3.19) is to make it a function of bed elevation [66], [3], [37]. This heuristic is motivated by hypothesis that basal material with a marine history should be weak [4]. PISM has a mechanism setting ϕ to be a *piece-wise 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

$$\phi(x, y) = \begin{cases} \phi_{\min}, & b(x, y) \leq b_{\min}, \\ \phi_{\min} + (b(x, y) - b_{\min}) M, & b_{\min} < b(x, y) < b_{\max}, \\ \phi_{\max}, & b_{\max} \leq b(x, y). \end{cases} \quad (3.20)$$

It is worth noting that an earth deformation model (see section [Earth deformation models](#)) changes $b(x, y)$ (NetCDF variable `topg`) used in (3.20), 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 `-topg_to_phi` in the second run would make PISM continue with the same `tillphi` field which was set in the first run.

Parameters

Prefix: `basal_yield_stress.mohr_coulomb.topg_to_phi`.

1. `enabled` (no) 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.
2. `phi_max` (15 degrees) upper value of the till friction angle; see the implementation of `MohrCoulombYieldStress`
3. `phi_min` (5 degrees) lower value of the till friction angle; see the implementation of `MohrCoulombYieldStress`
4. `topg_max` (1000 meters) the elevation at which the upper value of the till friction angle is used; see the implementation of `MohrCoulombYieldStress`
5. `topg_min` (-1000 meters) the elevation at which the lower value of the till friction angle is used; see the implementation of `MohrCoulombYieldStress`

Till friction angle optimization

Warning: This is a work in progress. Use at your own risk.

In *grounded* areas the distribution of till friction angle ϕ (see (3.19)) can be iteratively optimized in a forward equilibrium simulation.¹

The iteration starts from a ϕ distribution set using `-plastic_phi`, `-topg_to_phi`, or read from an input file (variable `tillphi`).

During each step, an adjustment $\Delta\phi$ is added to the previous value of ϕ and the result is clipped to ensure $\phi \in [\phi_{\min}, \phi_{\max}]$:

$$\phi_{n+1} = \min(\max(\phi_{\min}, \phi_n + \Delta\phi), \phi_{\max}), \quad (3.21)$$

The value of $\Delta\phi$ is proportional to the difference between the target (usually observed present day, e.g. [107]) and modeled surface elevations. It is similarly clipped to ensure $\Delta\phi \in [\Delta\phi_{\min}, \Delta\phi_{\max}]$:

$$\begin{aligned} \Delta\phi &= \min(\max(\Delta\phi_{\min}, \Delta\tilde{\phi}), \Delta\phi_{\max}), \\ \Delta\tilde{\phi} &= C(h_{\text{observed}} - h_{\text{modeled}}), \\ \Delta\phi_{\min} &= -2\Delta\phi_{\max}. \end{aligned} \quad (3.22)$$

Here C is the (positive) scaling factor (units: $^{\circ}/\text{m}$) set using `basal_yield_stress.mohr_coulomb.tillphi_opt.dphi_scale`.

Note: The adjustment $\Delta\phi$ is *positive* if the modeled surface elevation is below the reference value, and *negative* otherwise.

¹ This is similar to the simple inversion method described in [111] which optimizes the basal sliding coefficient τ_c instead.

In other words, the basal resistance is *increased* if the ice thickness is too low and *decreased* otherwise.

The lower bound $\phi_{\min} = \phi_0$ is a piece-wise linear function of the bed topography b :

$$\phi_0(b) = \begin{cases} \phi_{0,\min}, & b \leq b_{\min}, \\ \phi_{0,\min} + (\phi_{0,\max} - \phi_{0,\min}) \frac{b - b_{\min}}{b_{\max} - b_{\min}}, & b_{\min} < b \leq b_{\max}, \\ \phi_{0,\max} & b_{\max} < b. \end{cases} \quad (3.23)$$

Similarly to the till friction angle heuristic (3.20), we assume that “marine” sediments (below b_{\min}) can be much weaker than rather “continental” bedrock material (above b_{\max}). In sensitivity experiments we found a strong sensitivity of the Antarctic Ice Sheet’s ice volume in particular to the choice of ϕ_{\min} (see [108]).

To allow ice geometry to respond to changes in the till friction angle the simulation goes on for Δt_ϕ years between iterations.

Iterations at a particular location are considered “done” when the rate of change of the surface elevation mismatch $\Delta h = h_{\text{observed}} - h_{\text{modeled}}$ approximated using subsequent steps falls below a threshold D set using `basal_yield_stress.mohr_coulomb.tillphi_opt.dhdt_min`:

$$\frac{\Delta h(x, y, T + \Delta t_\phi) - \Delta h(x, y, T)}{\Delta t_\phi} \leq D \quad (3.24)$$

The `diff_mask` diagnostic variable is set to 0 to indicate that ϕ “converged” at this location.

Parameters

Prefix: `basal_yield_stress.mohr_coulomb.tillphi_opt`.

1. `dhdt_min` (0.001 *meters year-1*) rate of change in the surface elevation mismatch D used as a convergence criterion
2. `dphi_max` (1 *degrees*) maximum till friction angle adjustment $\Delta\phi_{\max}$
3. `dphi_scale` (0.002 *degree / meters*) scaling factor C used to compute the $\Delta\phi$ adjustment, C degrees per meter of surface elevation mismatch
4. `dt` (250 365*days*) time step Δt_ϕ for optimization of till friction angle
5. `file` Name of the file containing the time-independent variable `usurf` used as target surface elevation
6. `phi0_max` (5 *degrees*) maximum value of the lower bound of the till friction angle, $\phi_{0,\max}$
7. `phi0_min` (2 *degrees*) minimum value of the lower bound of the till friction angle, $\phi_{0,\min}$
8. `phi_max` (70 *degrees*) upper bound of the till friction angle ϕ_{\max}
9. `topg_max` (700 *meters*) the bed elevation b_{\max} above which `basal_yield_stress.mohr_coulomb.tillphi_opt.phi0_max` is used
10. `topg_min` (-300 *meters*) the bed elevation b_{\min} below which `basal_yield_stress.mohr_coulomb.tillphi_opt.phi0_min` is used

When the domain contains a grounding line the mismatch between modeled and observed surface elevation is meaningful only if the ice is grounded in *both* data sets. A retreat of the grounding line would make it impossible to optimize the till friction angle in areas that are observed to contain grounded ice but are covered by water in a simulation.

To avoid this issue, we

- disable the influence of the basal melt rate on geometry evolution by setting `geometry.update.use_basal_melt_rate` to “false”,

- modify the surface mass balance in grounded areas to disallow grounding line retreat by adding `no_gl_retreat` to the command-line option selecting a surface model (see [Preventing grounding line retreat](#)), and
- fix ice thickness where the bed elevation is below sea level and the ice (if present) is floating.

To fix ice thickness, we create a mask with ones where ice is floating or there is no ice and the bed is below sea level:

```
ncap2 -O \
-s "where(topg < 0 && thk*(910.0/1028.0) < 0 - topg) thk_bc_mask=1;" \
input.nc input-with-mask.nc
```

Here 910 is the ice density (see `constants.ice.density`), 1028 is the sea water density (see `constants.sea_water.density`), and 0 is the sea level elevation.

Reported diagnostic quantities

1. `usurf_difference` reports the mismatch between modeled and reference surface elevation fields
2. `diff_mask` reports the area where the till friction angle is iteratively adjusted or where the convergence criterion is met.
3. `usurf_target` reports the reference (target) ice surface elevation in use.

Determining the effective pressure

When using the default option `-yield_stress mohr_coulomb`, the effective pressure on the till N_{till} 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 (3.19) sets the value of τ_c proportionately to N_{till} . The amount of water in the till is, however, a nontrivial output of the hydrology (see [Subglacial hydrology](#)) and conservation-of-energy (see [Modeling conservation of energy](#)) submodels in PISM.

Following [109], based on laboratory experiments with till extracted from an ice stream in Antarctica, [110] propose the following parameterization which is used in PISM. It is based on the ratio $s = W_{till}/W_{till}^{max}$ where W_{till} is the effective thickness of water in the till and W_{till}^{max} ([hydrology.tillwat_max](#)) is the maximum amount of water in the till (see section [Subglacial hydrology](#)):

$$N_{till} = \min \left\{ P_o, N_0 \left(\frac{\delta P_o}{N_0} \right)^s 10^{(e_0/C_c)(1-s)} \right\} \quad (3.25)$$

Here P_o is the ice overburden pressure, which is determined entirely by the ice thickness and density, and the remaining parameters are listed below

Parameters

Prefix: `basal_yield_stress.mohr_coulomb`.

1. `delta.file` Name of the file containing space- and time-dependent variable `mohr_coulomb_delta` to use instead of `basal_yield_stress.mohr_coulomb.till_effective_fraction_overburden`.
2. `delta.periodic` (no) If true, interpret forcing data as periodic in time
3. `tauc_to_phi.file` File containing the basal yield stress field that should be used to recover the till friction angle distribution.
4. `till_cohesion` (0 Pascal) 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...”

5. `till_compressibility_coefficient` (0.12) coefficient of compressibility of till; value from [109]
6. `till_effective_fraction_overburden` (0.02) 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 [109]; default value from Greenland and Antarctic model runs
7. `till_log_factor_transportable_water` (0.1 meters) If `basal_yield_stress.add_transportable_water` is set then the water amount in the transport system is added to `tillwat` in determining `tauc`. Normally only the water in the till is used. This factor multiplies the logarithm in that formula.
8. `till_phi_default` (30 degrees) fill value for till friction angle
9. `till_reference_effective_pressure` (1000 Pascal) reference effective pressure N_0 ; value from [109]
10. `till_reference_void_ratio` (0.69) void ratio at reference effective pressure N_0 ; value from [109]

Note: While there is experimental support for the default values of C_c , e_0 , and N_0 , the value of δ should be regarded as uncertain, important, and subject to parameter studies to assess its effect.

Controlling minimum effective pressure

The effective pressure N_{till} above satisfies (see equation 20 in [110])

$$\delta P_o \leq N_{till} \leq P_o. \quad (3.26)$$

In other words, δ controls the lower bound of the effective pressure. In addition to setting it using a configuration parameter one can use a space- and time-dependent field. Set `basal_yield_stress.mohr_coulomb.delta.file` to the name of the file containing the variable `mohr_coulomb_delta` (dimensionless, i.e. units of “1”).

Note: PISM restricts the time step to capture changes in δ . For example, if you provide monthly records of δ PISM will make sure no time step spans more than one month.

PISM uses piece-wise linear interpolation in time for model times between records of δ .

Subglacial hydrology

At the present time, two simple subglacial hydrology models are implemented *and documented* in PISM, namely `-hydrology null` (and its modification `steady`) and `-hydrology routing`; see Table 3.12 and [110]. 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 *Controlling basal strength*). The pressure of the transportable water `bwat` in the `routing` model does not relate directly to the effective pressure on the till.

Note: Both `null` and `routing` described here provide all diagnostic quantities needed for mass accounting, even though `null` is not mass-conserving. See *Mass accounting in subglacial hydrology models* for details.

Table 3.12: Command-line options to choose the hydrology model

Option	Description
-hydrology null	The default model with only a layer of water stored in till. Not mass conserving in the map-plane but much faster than -hydrology routing. Based on “undrained plastic bed” model of [114]. The only state variable is <code>tillwat</code> .
-hydrology steady	A version of the null model that computes an approximation of the steady state subglacial water flux that can be used as an input for a frontal melt parameterization such as the one described in <i>Frontal melt parameterization</i> .
-hydrology routing	A mass-conserving horizontal transport model in which the pressure of translatable 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 3.13 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).

To use water runoff computed by a PISM surface model, set `hydrology.surface_input_from_runoff`. (The *Temperature-index scheme* computes runoff using computed melt and the assumption that a fraction of this melt refreezes, all other models assume that all melt turns into runoff.)

Table 3.13: Subglacial hydrology command-line options which apply to all hydrology models

Option	Description
<code>-hydrology.surface_input.file</code>	Specifies a NetCDF file which contains a time-dependent field <code>water_input_rate</code> which has units of water thickness per time. This rate is <i>added to</i> the basal melt rate computed by the energy conservation code.
<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; the configuration parameter `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 [114]; see also [29].

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

$$\frac{\partial W_{till}}{\partial t} = \frac{m}{\rho_w} - C \quad (3.27)$$

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 $0 \leq W_{till} \leq W_{till}^{\max}$ are satisfied.

This `-hydrology null` model has been extensively tested in combination with the Mohr-Coulomb till (section [Controlling basal strength](#)) for modelling ice streaming (see [66] and [29], among others).

“Steady state flow” model

This model (`-hydrology steady`) adds an approximation of the subglacial water flux to the `null` model. It *does not* model the steady state distribution of the subglacial water thickness.

Here we assume that the water input from the surface read from the file specified using `hydrology.surface_input-file` instantaneously percolates to the base of the ice and enters the subglacial water system.

We also assume that the subglacial drainage system instantaneously reaches its steady state. Under this assumption the water flux can be approximated by using an auxiliary problem that is computationally cheaper to solve, compared to using the mass conserving routing model (at least at high grid resolutions).

We solve

$$\frac{\partial u}{\partial t} = -\nabla \cdot (\mathbf{V}u) \quad (3.28)$$

on the grounded part of the domain with the initial state $u_0 = \tau M$, where τ is the scaling of the input rate (`hydrology.steady.input_rate_scaling`) and M is the input rate read from an input file.

The velocity field \mathbf{V} approximates the steady state flow direction. In this implementation \mathbf{V} is the normalized gradient of

$$\psi = \rho_i g H + \rho_w g b + \Delta\psi.$$

Here H is the ice thickness, b is the bed elevation, g is the acceleration due to gravity and ρ_i, ρ_w are densities of ice and water, respectively.

Note: Note that ψ ignores subglacial water thickness, producing flow patterns that are more concentrated than would be seen in a model that includes it. This effect is grid-dependent.

This implies that this code cannot model the distribution of the flow along the grounding line of a particular outlet glacier but it *can* tell us how surface runoff is distributed among the outlets.

The term $\Delta\psi$ is the adjustment needed to remove internal minima from the “raw” potential, filling any “lakes” it might have. This modification of ψ is performed using an iterative process; set `hydrology.steady.potential_n_iterations` to control the maximum number of iterations and `hydrology.steady.potential_delta` to affect the amount it is adjusted by at each iteration.

The equation (3.28) is advanced forward in time until $\int_{\Omega} u < \epsilon \int_{\Omega} u_0$, where ϵ (`hydrology.steady.volume_ratio`) is a fraction controlling the accuracy of the approximation: lower values of ϵ would result in a better approximation and a higher computational cost (more iterations). Set `hydrology.steady.n_iterations` to control the maximum number of these iterations.

This model restricts the time step length in order to capture the temporal variability of the forcing: the flux is updated at least once for each time interval in the forcing file.

In simulations with a slowly-varying surface input rate the flux has to be updated every once in a while to reflect changes in the flow pattern coming from changing geometry. Use `hydrology.steady.flux_update_interval` (years) to set the update frequency.

See [Computing steady-state subglacial water flux](#) for technical details.

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 [115]. 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 (3.27), but several additional parameters are used in determining how the transportable water `bwat` flows in the model; see Table 3.14. Specifically, the horizontal subglacial water flux is determined by a generalized Darcy flux relation [103], [116]

$$\mathbf{q} = -k W^\alpha |\nabla \psi|^{\beta-2} \nabla \psi \quad (3.29)$$

where \mathbf{q} is the lateral water flux, $W = bwat$ is the effective thickness of the layer of transportable water, ψ is the hydraulic potential, and k, α, β are controllable parameters (Table 3.14).

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

$$\psi = P_o + \rho_w g(b + W) \quad (3.30)$$

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. We also recommend 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 3.14: Command-line options specific to hydrology model `routing`

Option	Description
<code>-hydrology_hydraulic_conductivity k</code>	= k in formula (3.29).
<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>	= β in formula (3.29).
<code>-hydrology_thickness_power_in_flux α</code>	= α in formula (3.29).

Earth deformation models

The option `-bed_def [iso, lc, given]` (flag `bed_deformation.model`) turns one of the three available bed deformation models.

Point-wise isostasy

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 the values of `bed_deformation.mantle_density` and `constants.ice.density` in the configuration file. For an example and verification, see Test H in [Verification](#).

Lingle-Clark

The second model `-bed_def lc` is much more physical. It is based on papers by Lingle and Clark [112] and Bueler and others [27]. It generalizes and improves the most widely-used earth deformation model in ice sheet modeling, the flat earth Elastic Lithosphere Relaxing Asthenosphere (ELRA) model [113]. It imposes essentially no computational burden because the Fast Fourier Transform is used to solve the linear differential equation [27]. 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.

All parameters (except for `constants.ice.density`) controlling the Lingle-Clark model are listed below (they all have the prefix `bed.deformation.`).

Parameters

Prefix: `bed_deformation`.

1. `lc.elastic_model` (yes) Use the elastic part of the Lingle-Clark bed deformation model.
2. `lc.grid_size_factor` (4) 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 [112], [27].
3. `lc.update_interval` (10 365days) Interval between updates of the Lingle-Clark model
4. `lithosphere_flexural_rigidity` (5e+24 Newton meter) lithosphere flexural rigidity used by the bed deformation model. See [112], [27]
5. `mantle_density` (3300 kg meter-3) half-space (mantle) density used by the bed deformation model. See [112], [27]
6. `mantle_viscosity` (1e+21 Pascal second) half-space (mantle) viscosity used by the bed deformation model. See [112], [27]

Here are minimal example runs to compare these models:

```
mpiexec -n 4 pismr -eisII A -y 8000 -o eisIIA_nobd.nc
mpiexec -n 4 pismr -eisII A -bed_def iso -y 8000 -o eisIIA_bdiso.nc
mpiexec -n 4 pismr -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 [27].

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

Use the option `-topg_delta_file` (parameter `bed_deformation.bed_topography_delta_file`) 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:

$$b = b_0 + \Delta b. \quad (3.31)$$

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 bed elevations that are closer to observed values.

Warning: The variable `viscous_bed_displacement` does not correspond to any measured physical quantity. Do not even attempt to analyze it without a careful reading of [27].

Trying to provide a “hand-crafted” `viscous_bed_displacement` field to PISM is not a good idea.

Keep in mind that zero `viscous_bed_displacement` does *not* mean that the bed deformation model is in equilibrium.

Given bed deformation history

The last option `-bed_def given` can be used if a bed deformation history (i.e. bed elevation changes relative to a reference topography) is known from an external solid-Earth model¹. This can be useful when running simulations using offline coupling to such a model.

The bed topography b is set to

$$b(t, x, y) = b_{\text{ref}}(x, y) + \Delta b(t, x, y),$$

which is a time-dependent version of (3.31).

This class uses two input files:

1. Reference topography $b_{\text{ref}}(x, y)$ (variable `topg`, in meters).
2. Time-dependent history of bed elevation changes $\Delta b(t, x, y)$ relative to the reference topography (variable `topg_delta`, in meters).

Use the following configuration parameters (prefix: `bed_deformation.given.`) to set them.

1. `file` Name of the file containing time-dependent `topg_delta`.
2. `reference_file` Name of the file containing the reference bed topography `topg`.

Note: It is possible to combine high-resolution reference bed topography with low-spatial-frequency bed elevation changes: both files have to use the same grid projection and cover the modeling domain but they **do not** have to use the same grid.

¹ E.g. a relative sea level model.

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 [98], [89], [37] 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 [29]. These improvements make PISM an effective Antarctic model, as demonstrated by [99], [3], [100], 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 3.15](#). 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 3.15: 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
-cfbc	apply the stress boundary condition along the ice shelf calving front [37]
-kill_icebergs	identify and eliminate free-floating icebergs, which cause well-posedness problems for the SSA stress balance solver [37]
-part_grid	allow the ice shelf front to advance by a part of a grid cell, avoiding the development of unphysically-thinned ice shelves [98]
-subgl	apply interpolation to compute basal shear stress and basal melt near the grounding line [101]
-no_subgl_basal_melt	don't apply interpolation to compute basal melt near the grounding line if -subgl is set [101]
-pik	equivalent to option combination -cfbc -kill_icebergs -part_grid -subgl

Note: When in doubt, PISM users should set option `-pik` to turn on all of mechanisms in [Table 3.15](#). The user should also choose a calving model from [Calving and front retreat](#). 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 [43] as

$$\int_{z_s - \frac{\rho}{\rho_w} H}^{z_s + (1 - \frac{\rho}{\rho_w}) H} \sigma \cdot \mathbf{n} \, dz = \int_{z_s - \frac{\rho}{\rho_w} H}^{z_s} \rho_w g(z - z_s) \mathbf{n} \, dz. \quad (3.32)$$

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 (3.32) 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 (3.32) 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 (3.32), so as to apply the correct stresses [98], [37].

Partially-filled cells at the boundaries of ice shelves

Albrecht et al [98] 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 and front retreat](#)) 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 [45].) 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 [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 [102] and [101]. 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 [MISIP3d](#) 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” [37] 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 3.16](#). 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 3.16: The PISM mask, in combination with user options, determines the dynamical model.

Mask value	Meaning
0 = MASK_ICE_FREE_BEDROCK	ice free bedrock
2 = MASK_GROUNDED	ice is grounded
3 = MASK_FLOATING	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 = MASK_ICE_FREE_OCEAN	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 and front retreat

Overview

All mechanisms described below fall into two categories:

- mechanisms computing a *retreat rate* due to calving and using it to update ice geometry (*Eigen calving*, *von Mises stress calving*, *Hayhurst calving*), and
- mechanisms removing ice at a grid point according to a certain criterion (*Calving of thin floating ice*, *Calving of all floating ice*, *Prescribed front retreat*).

To select several calving mechanisms, use a comma-separated list of corresponding keywords. For example,

```
-calving eigen_calving,thickness_calving
```

selects *Eigen calving* and *Calving of thin floating ice*.

If more than one retreat-rate-based mechanism is selected, the corresponded rates are *added* and then used to update ice extent. (In other words: selected calving mechanisms are applied *together* instead of applying their effects *in sequence*.)

The partially-filled grid cell formulation (section *Partially-filled cells at the boundaries of ice shelves*) provides a framework suitable to relate calving rates 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 combined calving rate (velocity) can be used to limit the overall timestep of PISM (thus slowing down all of PISM) by using `geometry.front_retreat.use_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.

Setting the flag `geometry.front_retreat.wrap_around` to `true` allows the front retreat to “wrap around” the computational domain. (This is appropriate in some regional synthetic geometry setups.)

Scaling calving rates

Set `calving.rate_scaling.file` to scale the *total* (combined) calving rate from all selected rate-based mechanisms, e.g. to introduce calving variability corresponding to seasonal changes in ice melange. The file used with this option should contain the scalar time-dependent variable `frac_calving_rate` (units: 1).

Parameters

Prefix: `calving.rate_scaling`.

1. `file` File containing the scaling factor applied to calving rates from `eigen_calving`, `vonmises_calving`, and `hayhurst_calving` (variable name: `frac_calving_rate`)
2. `periodic` (no) If true, interpret forcing data as periodic in time

Eigen calving

PISM-PIK introduced a physically-based 2D-calving parameterization [89]. 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

$$c = K \dot{\epsilon}_+ \dot{\epsilon}_-, \quad (3.33)$$

$$\dot{\epsilon}_{\pm} > 0.$$

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 `calving.eigen_calving.K`.

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.

Parameters

Prefix: `calving.eigen_calving`.

1. `K` (0 meter second) Set proportionality constant to determine calving rate from strain rates. Note references [89], [3] use K in range 10^9 to 3×10^{11} m a, that is, 3×10^{16} to 10^{19} m s.

von Mises stress calving

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, [90] propose a calving law where the calving rate c is functionally related to tensile stresses:

$$c = |\mathbf{u}| \frac{\tilde{\sigma}}{\sigma_{max}}, \quad (3.34)$$

where $\tilde{\sigma}$ is the tensile von Mises stress and σ_{max} is a threshold that has units Pa (see `calving.vonmises_calving.sigma_max`). As the tensile fracture strength is much smaller than the compressive fracture strength, the effective tensile strain rate is defined as

$$\tilde{\epsilon}_e = \left(\frac{1}{2} \left(\max(0, \dot{\epsilon}_+)^2 + \max(0, \dot{\epsilon}_-)^2 \right) \right)^{1/2}. \quad (3.35)$$

Following [90], $\tilde{\sigma}$ is given by

$$\tilde{\sigma} = \sqrt{3} B \tilde{\epsilon}_e^{1/n}, \quad (3.36)$$

where B is the ice hardness.

Parameters

Prefix: `calving.vonmises_calving`.

1. `Glen_exponent` (3) Glen exponent in ice flow law for von Mises calving
2. `flow_law` (gpbld) The custom flow law for the von Mises stress computation
3. `sigma_max` (1e+06 Pa) Set maximum tensile stress. Note references [90] use 1.0e6 Pa.
4. `threshold_file` Name of the file containing the spatially-variable `vonmises_calving_threshold`.
5. `use_custom_flow_law` (no) Use custom flow law in the von Mises stress computation

Hayhurst calving

The option `-calving hayhurst_calving` implements the parameterization described in [91] (equation 22).

Note: FIXME: not documented.

Parameters

Prefix: `calving.hayhurst_calving`.

1. `B_tilde` (65 (MPa)^{r/year}) Effective damage rate [91]
2. `exponent_r` (0.43) Damage law exponent [91]
3. `sigma_threshold` (0.17 MPa) Damage threshold stress [91]

Calving of thin floating ice

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 configuration parameter `calving.thickness_calving.threshold`.

To set a time-and-space dependent ice thickness threshold, set the parameter `calving.thickness_calving.file`. This file should contain the variable `thickness_calving_threshold` in meters.

Parameters

Prefix: `calving.thickness_calving`.

1. `file` Name of the file containing the spatially-variable thickness calving threshold.
2. `periodic` (no) If true, interpret forcing data as periodic in time
3. `threshold` (50 meters) When terminal ice thickness of floating ice shelf is less than this threshold, it will be calved off.

Calving of all floating ice

The 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.

Set `calving.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.

Parameters

Prefix: `calving.float_kill`.

1. `calve_near_grounding_line` (yes) Calve floating ice near the grounding line.
2. `margin_only` (no) Apply float_kill at ice margin cells only.

Prescribed front retreat

Option `-front_retreat_file` allows prescribing retreat of the ice front. The forcing file specified using this option should contain `land_ice_area_fraction_retreat` — a 2D field, possibly time-dependent, that contains ones in areas that may be covered by ice and zeros in areas that have to be ice-free. Values between 0 and 1 allow for a “partial” retreat on coarser grids.

More precisely, `land_ice_area_fraction_retreat` is a mask prescribing the *maximum ice extent* at a given time throughout a simulation; a certain rate of retreat can be prescribed by creating a field with an appropriately decreasing maximum extent.

Changes in ice mass resulting from using this mechanism are reported as a part of the `discharge` (`tendency_of_ice_mass_due_to_discharge`).

Note: This replaces the `ocean_kill` mechanism available in previous PISM versions.

Parameters

Prefix: `geometry.front_retreat.prescribed`.

1. `file` Name of the file containing the maximum ice extent mask `land_ice_area_fraction_retreat`
2. `periodic` (no) If true, interpret forcing data as periodic in time

Modeling melange back-pressure

Equation (3.32) above, describing the stress boundary condition for ice shelves, can be written in terms of velocity components:

$$\begin{aligned} 2\nu H(2u_x + u_y)\mathbf{n}_x + 2\nu H(u_y + v_x)\mathbf{n}_y &= \int_b^h (p_{\text{ice}}(z) - p_{\text{water}}(z))dz \mathbf{n}_x, \\ 2\nu H(u_y + v_x)\mathbf{n}_x + 2\nu H(2v_y + u_x)\mathbf{n}_y &= \int_b^h (p_{\text{ice}}(z) - p_{\text{water}}(z))dz \mathbf{n}_y. \end{aligned} \quad (3.37)$$

Here ν is the vertically-averaged ice viscosity, H is the ice thickness, b is the elevation of the bottom and h of the top ice surface, p_{water} and p_{ice} are pressures of the column of ice and water, respectively:

$$\begin{aligned} p_{\text{ice}} &= \rho_{\text{ice}} g(h - z), \\ p_{\text{water}} &= \rho_{\text{water}} g \max(z_{\text{sea level}} - z, 0). \end{aligned} \quad (3.38)$$

We call the integral on the right hand side of (3.37) the *pressure difference term*.

It can be re-written as

$$\begin{aligned} \int_b^h p_{\text{ice}}(z) - p_{\text{water}}(z)dz &= H(\bar{p}_{\text{ice}} - \bar{p}_{\text{water}}), \text{ where} \\ \bar{p}_{\text{ice}} &= \frac{1}{2} \rho_{\text{ice}} g H, \\ \bar{p}_{\text{water}} &= \frac{1}{2} \rho_{\text{water}} g \frac{\max(z_s - b, 0)^2}{H}. \end{aligned} \quad (3.39)$$

PISM's ocean model components provide \bar{p}_{water} , the vertically-averaged pressure of the water column adjacent to an ice margin.

To model the effect of melange [97] on the stress boundary condition we modify the pressure difference term in (3.37), adding \bar{p}_{melange} , the vertically-averaged melange back pressure:

$$\int_b^h (p_{\text{ice}} - (p_{\text{water}} + \bar{p}_{\text{melange}})) dz. \quad (3.40)$$

By default, \bar{p}_{melange} is zero, but PISM implements two ocean model components to support scalar time-dependent melange pressure forcing. Please see the [Climate Forcing Manual](#) for details.

Fracture density modeling

The fracture density approach in PISM is based on [92] and assumes a macroscopic measure for the abundance of (partly microscale) crevasses and rifts that form in ice (shelves) and that can be transported with the ice flow as represented in a continuum ice-flow model. This approach is similar to the Continuum Damage Mechanics (CDM) (e.g. [93] and [94]) introducing a damage state variable (ϕ or D) that equals zero for fully intact ice and one for fully fractured ice, that can be interpreted as a loss of all load bearing capacity.

The feedback of damage to the ice flow (creep) works within the existing constitutive framework by introducing a linear mapping between the actual physical (damaged) state of the material and an effective state that is compatible with a homogeneous, continuum representation of the creep law (Eq. 6 in [95]).

Fractures form above a critical stress threshold σ_{cr} in the ice (e.g. von Mises criterion, maximum stress criterion or fracture toughness from Linear Elastic Fracture Mechanics) with a fracture growth rate proportional to γ (Eq. 2 in [95]), that is related to the strain rate (longitudinal spreading or effective strain rate; Eq. 9 in [92]). Fracture healing is assumed to occur with a defined healing rate below a strain rate threshold (scaled with the difference to the threshold or constant; Eq. 11 in [92]).

The fracture growth constant γ (`fracture_density.gamma`) is ignored if `fracture_density.borstad_limit` is set.

To enable this model, set `fracture_density.enabled`.

Parameters

Prefix: `fracture_density`.

1. `borstad_limit` (no) Model fracture growth according to the constitutive law in [96] (Eq. 4), ignoring `fracture_density.gamma`.
2. `constant_fd` (no) Keep fracture density fields constant in time but include its softening effect.
3. `constant_healing` (no) Use a constant healing rate $-\gamma_h \dot{\epsilon}_h$ independent of the local strain rate.
4. `fd2d_scheme` (no) Use an alternative transport scheme to reduce numerical diffusion (Eq. 10 in [95])
5. `fracture_weighted_healing` (no) Multiply the healing rate by $1 - D$, i.e. assume that highly damaged ice heals slower. This mechanism can be combined with `fracture_density.constant_healing`.
6. `gamma` (1) fracture growth constant γ
7. `gamma_h` (0) fracture healing constant γ_h
8. `healing_threshold` (2e-10 1/s) fracture healing strain rate threshold $\dot{\epsilon}_h$
9. `include_grounded_ice` (no) Model fracture density in grounded areas (e.g. along ice stream shear zones) in addition to ice shelves
10. `initiation_threshold` (70000 Pa) fracture initiation stress threshold σ_{cr}
11. `lefm` (no) Use the mixed-mode fracture toughness stress criterion based on Linear Elastic Fracture Mechanics, Eqs. 8-9 in [95]
12. `max_shear_stress` (no) Use the maximum shear stress criterion for fracture formation (Tresca or Guest criterion in literature), which is more stringent than the default von Mises criterion, see Eq. 7 in [95]
13. `phi0` (0) Fracture density value used at grid points where ice velocity is prescribed. This assumes that all ice entering a shelf at `bc_mask` locations has the same fracture density.
14. `softening_lower_limit` (1) Parameter controlling the strength of the feedback of damage on the ice flow. If 1: no feedback, if 0: full feedback (ϵ in Eq. 6 in [95])

Testing

See the scripts in `example/ross/fracture` for a way to test different damage options and parameter values. Build a setup for the Ross Ice Shelf and let the damage field evolve, with fracture bands reaching all the way from the inlets to the calving front.

Contents

- *Modeling individual outlet glaciers*
 - *Energy*
 - *Stress balance*
 - *Mass continuity*
 - * *Mass balance adjustment*
 - * *Calving*

3.4.5 Modeling individual outlet glaciers

PISM was created to model ice sheets in entirety; in this context we can assume that ice does not extend to the edge of the computational domain and there is no need to provide lateral boundary conditions.

However, in some interesting cases the ice *does* extend to the edge of the domain, but we can assume that changes near the boundary do not affect the behavior in the region we want to model. Two examples come to mind.

1. Modeling an individual alpine glacier: there may be ice near a domain boundary, but we can select the domain so that corresponding ice masses are not connected to the glacier we're modeling. Note that it is not always possible to "remove" glaciers we don't care about because they may re-appear due to the mass balance forcing.
2. Modeling an outlet glacier in an ice sheet: we cut out a region from an ice sheet, so ice *will* extend to the edge of the domain, but we can select the domain so that it contains the whole *drainage basin* of the outlet glacier of interest.

In some ways this case is harder to model: the shape and size of drainage basins of an ice sheet changes with its geometry, so we have to assume that our simulation is short enough to ensure that the drainage basin we're modeling remains roughly the same in shape and extent.

See [Example: A regional model of the Jakobshavn outlet glacier in Greenland](#) for an example.

Use

```
pismr -regional ...
```

to enable PISM's "regional mode."

To run a regional mode simulation using a subset of the grid in an input file, use

```
pismr -regional -bootstrap -i input.NYC \
-x_range x_min,x_max \
-y_range y_min,y_max
```

where x_{\min} , x_{\max} , etc (in meters) define a bounding box for a sub-domain.

To use the same subset of the domain, but at a higher resolution, add `-refinement_factor N` for a grid N times finer than the one in `input.nc`:

```
pismr -regional -bootstrap -i input.NYC \
-x_range x_min,x_max \
-y_range y_min,y_max \
-refinement_factor N
```

Ideally, modeling a region containing an ice mass extending to the edge of the domain would use the following time-dependent lateral boundary conditions

- energy balance: enthalpy flux across the boundary,
- mass continuity: mass flux across the boundary,
- stress balance: sliding speed at the boundary.

PISM's regional mode uses a special mask `no_model_mask` (zeros in the interior of the modeling domain, ones at the edge of the domain or in other areas that are *not modeled*) to implement modifications at domain boundaries. This mask is saved to output files and read back in when the model is re-started. Set `regional.no_model_strip` during bootstrapping to create a “non-modeled” strip of a given width along the domain boundary.

Energy

PISM assumes that ice enthalpy and the basal melt rate (i.e. parts of the model state that capture the energy state) near the boundary of the domain *remain constant*: at the end of each time step updated enthalpy and basal melt rate are re-set to values read from an input file or computed during bootstrapping at all grid points where `no_model_mask` is 1.

Stress balance

When prescribing the sliding velocity, the `no_model_mask` overrides the basal sliding B.C. mask: all `no_model_mask` locations are *also* the Dirichlet B.C. locations for the sliding velocity. This makes it possible to prescribe the sliding velocity of the ice across the domain boundary. Set `stress_balance.ssa.dirichlet_bc` to `true` to enable this feature.

In many cases it makes sense to *disable* sliding at the boundary. When the sliding velocity near the boundary is not prescribed, PISM sets the basal yield stress to a high value (see `regional.no_model_yield_stress`).

The domain in PISM “wraps around”, which means that we can not accurately compute gradients near the boundary in the non-periodic case.

Note, though, that updating the velocity field requires computing the gravitational driving stress, which depends on gradients of the ice thickness and surface elevation.

To avoid using finite differences across the domain boundary when computing these gradients, PISM stores ice thickness and surface elevation near the edge of the domain and uses them to modify surface elevation and thickness gradients.

Note: In the SIA stress balance model, prescribing ice thickness and surface elevation near the edge of the domain is equivalent to prescribing the *flux* across the domain boundary.

To use *zero* surface elevation and thickness gradients, set `regional.zero_gradient`. (This disables SIA flow across the boundary.)

Warning: High surface elevation and ice thickness gradients near the domain boundary *will* affect time-stepping even if they do not now affect model evolution.

The resulting high SIA diffusivity will force PISM to take unreasonably short time steps, wasting computational time.

Consider setting `regional.zero_gradient` if you see high SIA diffusivities near domain boundaries (see `diffusivity_staggered` to check).

Mass continuity

PISM uses the SSA Dirichlet B.C. mask as the ice thickness Dirichlet B.C. mask, i.e. ice thickness is fixed wherever the sliding velocity is fixed. (In other words, PISM allows prescribing the *ice flux* at a given location.)

This means that the *ice thickness does not evolve* in the `no_model_mask` area.

Mass balance adjustment

Prescribing the ice thickness near the boundary when the ice in the interior of the domain thins would lead to high thickness and surface elevation gradients at the inner boundary of the “non-modeled” strip. Use *Mass flux adjustment* to keep the ice geometry from deviating from the target *without* sharp transitions at the boundary from fixed to evolving ice thickness.

Calving

Set `geometry.front_retreat.wrap_around` to `true` to allow calving front retreat due to calving to “wrap around” the computational domain. This may be necessary in some regional synthetic-geometry setups.

3.4.6 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.

3.4.7 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 3.17](#).

Table 3.17: 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 `nc2cd0.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 Table 3.18 summarizes command-line options controlling the most basic ways to input and output NetCDF files when starting and ending PISM runs.

Table 3.18: 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 Initialization and bootstrapping .
<code>-bootstrap</code>	Bootstrap from the file set using <code>-i</code> using heuristics to “fill in” missing fields. See section Initialization and bootstrapping .
<code>-ssa_read_initial_guess false</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 Choosing the stress balance).
<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 3.19 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 3.19: 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 grep to get usefully-filtered information on options, for example <code>pismr -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’s I/O performance

When working with fine grids (resolutions of 2km and higher on the whole-Greenland scale, for example), the time PISM spends writing output files, spatially-varying diagnostic files, or backup files can become significant.

For fast file I/O the order of dimensions of a NetCDF variable in an output file has to match the order used by PISM in memory, so we use the `time,y,x,z` storage order instead of the more convenient (e.g. for NetCDF tools) order `time,z,y,x`.

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

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

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

PISM also supports parallel I/O using parallel NetCDF, PnetCDF, or ParallelIO, which can give better performance in high-resolution runs.

Use the command-line option `-o_format` (parameter `output.format`) to choose the approach to use when writing to output files (see [Table 3.20](#)). The `netcdf4_parallel` requires parallel NetCDF, `pnetcdf` requires PnetCDF, and `pio_...` require ParallelIO build with parallel NetCDF and PnetCDF. Section [PISM’s build-time configuration](#)) explains how to select these libraries when building PISM.

Note: When built with parallel NetCDF or PnetCDF (or both) PISM attempts to choose the best way to *read* from

input files and this logic appears to work well. This is why there is no `-i_format`.

Table 3.20: Methods of writing to output files

<code>-o_format</code> argument	Description
<code>netcdf3</code>	(default); serialized I/O from rank 0 (NetCDF-3 file)
<code>netcdf4_parallel</code>	parallel I/O using NetCDF (HDF5-based NetCDF-4 file)
<code>pnetcdf</code>	parallel I/O using PnetCDF (CDF5 file)
<code>pio_pnetcdf</code>	parallel I/O using ParallelIO (CDF5 file)
<code>pio_netcdf4p</code>	parallel I/O using ParallelIO (HDF5-based NetCDF-4 file)
<code>pio_netcdf4c</code>	serial I/O using ParallelIO (<i>compressed</i> HDF5-based NetCDF-4 file)
<code>pio_ncdf</code>	serial I/O using ParallelIO (using data aggregation in ParallelIO)

The ParallelIO library can aggregate data in a subset of processes used by PISM. To choose a subset, set

- `output.pio.n_writers` number of “writers”
- `output.pio.base` the index of the first writer
- `output.pio.stride` interval between writers

Note: The CDF5 file format is a large-variable extension of the NetCDF-3 file format developed by the authors of PnetCDF. This format is supported by NetCDF since version 4.4.

We recommend performing a number of test runs to determine the best choice for your simulations.

In our test runs on 120 cores (whole Greenland setup on a 900m grid) `pio_pnetcdf` with `output.pio.n_writers` set to the number of cores used by PISM (120) gave the best performance.

Note: It is important to make sure that PISM’s output files are written to a parallel file system and this file system is configured to achieve optimal performance.

On Lustre (a common parallel file systems) the theoretical throughput when writing to a file depends on the number of *object storage targets* used to store it: if a target can write 500 MiB/s, a file spread over 2 could be written at 1000 MiB/s assuming that we are writing to both of them at the same time, and so on.

For maximum speed we want to distribute an output file over all available targets.

To do this:

1. Create a directory that will contain PISM output files (`output_directory` below).
2. Run

```
lfs setstripe -c -1 output_directory
```

This sets the “stripe count” to -1, which means “all”.

Now all files in `output_directory` and all its sub-directories can use all available targets.

3.5.3 Understanding adaptive time-stepping

It is helpful to keep in mind this fundamental fact:

length of time steps taken by a model affects results of a simulation.

This applies to all evolutionary models and PISM is no different.

We expect model results to converge to the solution of the continuum problem corresponding to a model as Δt goes to zero. Also, results using different Δt should be “close” to this solution and to each other, *but they need not be the same*.

One important consequence is that changes in PISM settings that may not seem to be related to physical choices may affect results *if* they affect time stepping.

Here is a possibly-incomplete list of PISM components and settings that may affect time step length.

1. Numerical stability criteria.

1. Diffusivity-based time step restriction for the mass continuity (mass transport) step using SIA diffusivity (or its estimate when the Blatter solver is used).
2. The value of `time_stepping.adaptive_ratio` adjusting the diffusivity-based time step restriction (see (3.4)).
3. CFL time step restriction for the mass continuity step using sliding velocity, (or vertically-averaged horizontal velocity with the Blatter solver).
4. CFL time step restriction for horizontal advection within the ice volume within energy balance and age models. Uses horizontal (u, v) components of the ice velocity within the 3D volume of the ice.

2. Reporting.

1. If `time_stepping.hit_ts_times` is set, PISM will adjust time step lengths to “hit” times requested with `output.timeseries.times`.
2. If `time_stepping.hit_extra_times` is set (the default), PISM will adjust time step lengths to “hit” times requested with `output.extra.times`.
3. If `time_stepping.hit_save_times` is set, PISM will adjust time step lengths to “hit” times requested with `output.snapshot.times`.
3. Time-step “skipping” to reduce computational costs: `time_stepping.skip.enabled` and `time_stepping.skip.max`.

This mechanism enables PISM to take a number of “cheap” mass-balance steps (including SIA diffusivity updates) before more expensive temperature, age, and SSA stress balance computations are done.

Time step “skipping” roughly corresponds to asynchronous coupling between

- ice flow by shear in planes parallel to the geoid and
- membrane-type ice flow and advection of energy and tracers (such as age).

This is only effective if the time step is being limited by the diffusivity time step restriction associated to mass continuity using the SIA.

PISM computes time step restrictions $\{\Delta t_0, \Delta t_1, \dots, \Delta t_n\}$ from *all* of PISM’s sub-modules and sorts them from smallest to largest. Then the maximum allowed time step is

$$\Delta t_{\max} = \Delta t_0.$$

If `time_stepping.skip.enabled` is set *and* the most severe restriction comes from the SIA-diffusivity-based stability criterion for mass continuity, it skips

$$N = \min \left(\left\lfloor 0.95 \frac{\Delta t_1}{\Delta t_0} \right\rfloor, N_{\max} \right) \quad (3.41)$$

energy, age, and 3D velocity updates, where N_{\max} is set using `time_stepping.skip.max`.

Warning: The effects of this mechanism are not well understood. Please use with caution.

The maximum recommended value for `time_stepping.skip.max` depends on the context. The temperature field should be updated when the surface changes significantly, and likewise the basal sliding velocity if it comes from the SSA calculation.

4. Atmosphere, surface process, ocean, and sea level forcing components.
5. The Lingle-Clark bed deformation model (see [Lingle-Clark](#) and [bed_deformation.lc.update_interval](#)).
6. If `geometry.front_retreat.use_cfl` is set, PISM adjusts time step lengths to satisfy the CFL condition that uses the total front retreat rate coming from calving and frontal melt models.
7. The time step length never exceeds `time_stepping.maximum_time_step`.
8. If `time_stepping.hit_multiples` is set to a positive number, PISM will “hit” multiples of the number of model years specified. For example, if stability criteria require a time-step of 11 years and the `-timestep_hit_multiples 3` 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.
9. If the value R set using `time_stepping.resolution` is positive PISM reduces the time step length so that

$$\Delta t = N \cdot R \quad (3.42)$$

for some integer N .

The default R (1 second) allows PISM to represent model time more accurately, reducing the influence of rounding errors.

Note: This is an intermediate-term solution for an issue reported by Thomas Kleiner: some simulations produced different results with identical inputs but *different* start years.

We tracked it down to the fact that these simulations ended up using slightly different time step lengths. This, in turn, was caused by differences in the *absolute* precision of the C++ type `double` for numbers of different magnitudes.

10. The time step length never exceeds the total length of the run.

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 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 [30]. The first word we see here, namely “`diffusivity`”, is the adaptive-timestepping “reason”. See Table 3.21. 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 parameter `time_stepping.skip.enabled` enables this mechanism, while `time_stepping.skip.max` sets the maximum number of such substeps. The adaptive mechanism may choose to take fewer substeps than `time_stepping.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_sliding_vel
U	years	10^6 km^3	10^6 km^2	$\text{m}^2 \text{ s}^{-1}$	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 sliding velocity”. Specifically, the last number is $\max(\max(|u|), \max(|v|))$, i.e. the number that appears in the CFL time step restriction for the “advection” part of the flow in the mass continuity equation.

Note: `max_sliding_vel` reported here is not the same as the maximum sliding speed, $\max(\sqrt{u^2 + v^2})$.

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 [29].

Table 3.21: Meaning of some adaptive time-stepping “reasons” in the standard output line

PISM output	Active adaptive constraint or PISM sub-system that limited time-step size
2D CFL	CFL condition for the “advection” part of the mass continuity equation. Uses (u, v) components of the vertically-averaged horizontal ice velocity with the Blatter stress balance model. Uses sliding velocity with all other stress balance choices [29].
<code>diffusivity</code>	SIA-diffusivity-based time step restriction for the mass continuity equation [30], [123]
<code>energy, age model</code>	CFL condition using horizontal (u, v) components of the ice velocity within the ice volume. Restricts the time step of enthalpy, temperature, or age advection [30]. (This CFL condition does not use the vertical (w) component of ice velocity: PISM’s 3D advection scheme is explicit in x and y and implicit in z .)
<code>end_of_the_run</code>	end of prescribed run time
<code>max</code>	maximum allowed Δt set with <code>time_stepping.maximum_time_step</code>
<code>-ts_... reporting</code>	the <code>-ts_times</code> option and the configuration parameter <code>time_stepping.hit_ts_times</code> ; see section <i>Scalar diagnostic quantities</i>
<code>-extra_... reporting</code>	the <code>-extra_times</code> option and the configuration parameter <code>time_stepping.hit_extra_times</code> ; see section <i>Spatially-varying diagnostic quantities</i>
<code>surface</code>	a surface or an atmosphere model
<code>ocean</code>	an ocean model
<code>hydrology</code>	a hydrology model stability criterion, see section <i>Subglacial hydrology</i>
<code>front_retreat</code>	CFL condition using the 2D horizontal retreat rate such as the eigen-calving model, see section <i>Calving and front retreat</i>

Parameters

Prefix: `time_stepping`.

1. `adaptive_ratio` (0.12) Adaptive time stepping ratio for the explicit scheme for the mass balance equation; [30], inequality (25)
2. `count_steps` (no) 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.
3. `hit_extra_times` (yes) Modify the time-stepping mechanism to hit times requested using `output.extra.times`.
4. `hit_multiples` (0 years) Hit every X years, where X is specified using this parameter. Use 0 to disable.
5. `hit_save_times` (no) Modify the time-stepping mechanism to hit times requested using `output.snapshot.times`.
6. `hit_ts_times` (no) Modify the time-stepping mechanism to hit times requested using `output.timeseries.times`.
7. `maximum_time_step` (60 365days) Maximum allowed time step length
8. `resolution` (1 seconds) Time steps are rounded down to be a multiple of this number (set to zero to allow arbitrary time step lengths)
9. `skip.enabled` (no) Use the temperature, age, and SSA stress balance computation skipping mechanism.
10. `skip.max` (10) Number of mass-balance steps, including SIA diffusivity updates, to perform before a the temperature, age, and SSA stress balance computations are done

3.5.4 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 [Parameters](#) below for the list of options and [Scalar time-series](#) for the full list of supported time-series.

Note: Similarly to the snapshot-saving code (section [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` 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.)

Parameters

1. `append` (false) If true, append to the scalar time series output file.
2. `buffer_size` (10000) 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.
3. `filename` Name of the file to save scalar time series to. Leave empty to disable reporting scalar time-series.
4. `times` List or range of times defining reporting time intervals.
5. `variables` Requested scalar (time-series) diagnostics. Leave empty to save all available diagnostics.

Comments

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`. Corresponding quantities without the suffix *do* include areas with a thin, “seasonal” ice cover.
- Ice volume and area are computed and then split among floating and grounded portions:

```
ice_volume_glacierized ↪(ice_volume_glacierized_shelf,
                           ice_volume_glacierized_grounded),
```

while

```
ice_area_glacierized ↪(ice_area_glacierized_shelf,
                           ice_area_glacierized_grounded)
```

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 ice. Thus there is another splitting:

```
ice_volume_glacierized ↪(ice_volume_glacierized_cold,
                           ice_volume_glacierized_temperate)
```

and

```
ice_area_glacierized ↪(ice_area_glacierized_cold_base,
                           ice_area_glacierized_temperate_base).
```

- The sea level rise potential `sea_level_rise_potential` is the increase in sea level (in meters) that would result from melting all the grounded ice not displacing sea water and distributing the corresponding *fresh water* volume uniformly over the entire global ocean ($362.5 \cdot 10^6 \text{ km}^2$, see [118] and `constants.global_ocean_area`). This follows the definition used in the SeaRISE project [62].
- 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.

3.5.5 Spatially-varying diagnostic quantities

Sometimes it is useful to have PISM save a handful of diagnostic *maps* at some interval, for example every 10 years or even every month. One can use snapshots (section [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.

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 [Parameters](#) below for all the parameters controlling this feature. The section [Spatially-variable fields](#) list all the variable choices.

Note: Some diagnostics are only available if the simulation uses a sub-model that provides them. PISM will stop with an error message if a diagnostic is requested but not available. To print a warning and continue instead of stopping, set `output.extra.stop_missing` to “false”.

Note that options `-extra_times`, `-save_times`, `-ts_times` take *dates* if a non-trivial calendar is selected. Here are some examples.

```
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 $3.14\ldots \times 10^7$ ” 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 $1, \dots, 10$ 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 [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.

Parameters

Prefix: `output.extra`.

1. `append` (no) Append to an existing output file. No effect if file does not yet exist, and no effect if `output.extra.split` is set.
2. `file` Name of the file that will contain spatially-variable diagnostics. Should be different from `output.file_name`.
3. `split` (no) Save spatially-variable diagnostics to separate files (one per time record).
4. `stop_missing` (yes) Stop if requested variable is not available instead of warning.
5. `times` List or a range of times defining reporting intervals for spatially-variable diagnostics.
6. `vars` Comma-separated list of spatially-variable diagnostics.

3.5.6 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`”.

Parameters

1. `file` Snapshot (output) file name (or prefix, if saving to individual files).
2. `size` (small) The “size” of a snapshot file. See parameters `output.sizes.medium`, `output.sizes.big_2d`, `output.sizes.big`
3. `split` (no) Save model state snapshots to separate files (one per time record).
4. `times` List or a range of times to save model state snapshots at.

3.5.7 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 3.22. 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 3.23.

Table 3.22: Options controlling run-time diagnostic viewers

Option	Description
<code>-view</code>	Turns on map-plane views of one or several variables, see <i>Spatially-variable fields</i> .
<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 3.23: 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>-ssafd_ksp_monitor_draw</code>	Iteration monitor for the Krylov subspace routines (KSP) in PETSc. Residual norm versus iteration number.

3.5.8 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
```

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

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 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 = "number" ;
@@ -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 (snapshots, 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.

3.5.9 Regridding

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:

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

By specifying regredded 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 3.24 for the regreddable variables using `-regrid_file`. Only model state variables are regreddable, while climate and boundary data generally are not explicitly regreddable. (Bootstrapping, however, allows the same general interpolation as this explicit regrid.)

Table 3.24: Regreddable variables. Use `-regrid_vars` with these names.

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

3.5.10 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 3.25](#) 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 3.25: Signalling running PISM processes. “*PIDs*” stands for list of all identifiers of the PISM processes.

Command	Signal	PISM behavior
<code>kill -KILL <i>PIDs</i></code>	SIGKILL	Terminate with extreme prejudice. PISM cannot catch it and no state is saved.
<code>kill -TERM <i>PIDs</i></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 <code>history</code> string in the output file will contain an “EARLY EXIT caused by signal SIGTERM” indication.
<code>kill -USR1 <i>PIDs</i></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 <i>PIDs</i></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.

3.5.11 Balancing the books

2D diagnostics

PISM provides a number of 2D diagnostics to keep track of mass conservation.¹

All of them are computed as time-averaged fluxes over requested reporting intervals. Positive values correspond to mass gain.

For ice mass, at every grid point we have

```
ice_mass_accounting_error = 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_flux +
tendency_of_ice_mass_due_to_basal_mass_flux +
tendency_of_ice_mass_due_to_discharge);

ice_mass_accounting_error@long_name = "ice mass accounting error";
```

All names on the right-hand side correspond to valid PISM diagnostic quantities.

To check that all changes in mass are accounted for, download the script above and run²

```
ncap2 -v -S ice_mass_accounting_error.txt \
pism_output.nc mass_accounting_error.nc
```

The variable `ice_mass_accounting_error` in `mass_accounting_error.nc` will contain ice mass accounting errors at each point. All values of this variable should be close to or equal to zero. They are not zero (in general) due to rounding errors, though.

Use a shortcut

¹ See [Diagnostic quantities](#) for the full list of diagnostics.

² `ncap2` is a part of `NCO`.

```
pismr -extra_file ex.nc -extra_times N -extra_vars mass_fluxes,...
```

to save all fluxes needed to “balance the books” in terms of ice mass.

Alternatively, use fluxes in terms of “ice amount” (mass per unit area):

```
ice_amount_accounting_error = 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_flux +
   tendency_of_ice_amount_due_to_basal_mass_flux +
   tendency_of_ice_amount_due_to_discharge);

ice_amount_accounting_error@long_name = "ice amount accounting error";
```

To save these, use the shortcut

```
pismr -extra_file ex.nc -extra_times N -extra_vars amount_fluxes,...
```

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 “balance the books”. This includes changes needed to preserve non-negativity of ice thickness.⁴
- `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 scalars, integrated over the whole computational domain. The “integrated” mass accounting error can be computed using the `ncap2` script below.

```
ice_mass_accounting_error = 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_basal_mass_flux +
   tendency_of_ice_mass_due_to_surface_mass_flux +
   tendency_of_ice_mass_due_to_discharge);

ice_mass_accounting_error@long_name = "ice mass accounting error";
```

⁴ While PISM’s mass transport scheme is not proven to be mass-conserving *in every case* (see [122]), in most simulations the field is uniformly zero.

Comments

- `tendency_of_ice_mass_due_to_flow` 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

Mass accounting in subglacial hydrology models

PISM's hydrology models provide all the diagnostic fields needed to keep track of changes in subglacial water thickness.³

At every grid point we have

```
water_mass_accounting_error = tendency_of_subglacial_water_mass -
  (tendency_of_subglacial_water_mass_due_to_input +
   tendency_of_subglacial_water_mass_due_to_flow +
   tendency_of_subglacial_water_mass_due_to_conservation_error +
   tendency_of_subglacial_water_mass_at_grounded_margins +
   tendency_of_subglacial_water_mass_at_grounding_line +
   tendency_of_subglacial_water_mass_at_domain_boundary);

water_mass_accounting_error@long_name = "subglacial water mass accounting error";
```

All names on the right-hand side correspond to valid PISM diagnostic quantities.

Use a shortcut

```
pismr -extra_file ex.nc -extra_times N -extra_vars hydrology_fluxes,...
```

to save all diagnostics mentioned above.

See [Subglacial hydrology](#) for more information about hydrology models.

Mass accounting in the PDD model

PISM's PDD model provides diagnostics needed to compare computed accumulation, melt, and runoff to the effective mass balance. Use diagnostic quantities `surface_accumulation_flux`, `surface_melt_flux`, and `surface_runoff_flux` (units of mass per area per time) and `surface_accumulation_rate`, `surface_melt_rate`, `surface_runoff_rate` (units of mass per time).

To save all these, use `-extra_vars` shortcuts `pdd_fluxes` and `pdd_rates`.

³ We keep track of $W_{\text{till}} + W$, i.e. the sum of the effective thickness of subglacial water stored in till *and* the effective thickness of subglacial water in the transport layer (if applicable).

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 [26] 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 `.petscrc` 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 `.petscrc`) 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. `-ssafd_picard_rtol 1e-7 -ssafd_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 `-ssafd_pc_type none`.

For the full list of PETSc options controlling the SSAFD solver, run

```
ssa_testi -ssa_method fd -help | grep ssafd_ | 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 3.26](#). The python scripts are all documented at the *Packages* tab on the [PISM Source Code Browser](#). The Python scripts all take option `--help`.

Table 3.26: Some scripts which help in using PISM

Script	Function
<code>test/vfnow.py</code>	Organizes the process of verifying PISM. Specifies standard refinement paths for each of the tests (section Verification).
<code>test/vnreport.py</code>	Automates the creation of convergence graphs like figures Fig. 3.34 – Fig. 3.38 .
<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 Using PISM for flow-line modeling .
<code>util/flowlineslab.py</code>	See section Using PISM for flow-line modeling .
<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 `grid.Mx` and `grid.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 `grid.Mx` or `grid.My` to 1.

There is a way around this, though: by using `grid.periodicity` 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.

In this case `grid.My` can be any number; we want to avoid unnecessary computations, though, so `grid.My` of 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 `examples/preprocessing/flowlineslab.py` to create a minimal, and obviously unrealistic, dataset. The file `slab.nc` created by this script contains all the required information but is not ready to use with PISM. Proceed as follows:

```
examples/preprocessing/flowlineslab.py # creates slab.nc with only an x-direction
flowline.py -o slab-in.nc --expand -d y slab.nc
```

This 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`:

```
mpixexec -n 2 pismr \
    -surface given \
    -bootstrap -i slab-in.nc \
    -Mx 201 -Lx 1000 \
    -My 3 -Ly 4 -periodicity y \
    -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 using [NCO](#):

```
ncks -O -d y,1 pism-out.nc slab-out.nc
ncwa -O -a time,y slab-out.nc slab-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 <https://www.pism.io> 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.

- 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.

¹ This script requires the `numpy` and `netCDF4` Python modules. Run `flowline.py --help` for a full list of options.

¹ This automates running verification tests described in section [Verification](#), for example.

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.

3.5.16 ISMIP6 Greenland

Running [ISMIP6-Greenland](#) projections required implementing some additional sub-models as well as several modifications needed to follow ISMIP6 conventions. This section describes these modifications and explains how to use PISM to run ISMIP6 projections.

Top surface mass balance and temperature

Use the `ismip6` surface model to implement ISMIP6 surface mass balance forcing.

```
pismr -surface ismip6 \
    -surface_ismip6_file climate_forcing.nc \
    -surface_ismip6_reference_file climate_forcing_reference.nc
```

Here `climate_forcing.nc` should contain time-dependent variables

- `climatic_mass_balance_anomaly` (units: $kg/(m^2 s)$) and
- `ice_surface_temp_anomaly` (units: *Kelvin*).

The file `climate_forcing_reference.nc` should contain time-independent (2D) variables

- `climatic_mass_balance_reference` (units: $kg/(m^2 s)$),
- `climatic_mass_balance_gradient` (units: $(kg/(m^2 s))/m$),
- `ice_surface_temp_reference` (units: *Kelvin*),
- `ice_surface_temp_gradient` (units: *Kelvin/m*),
- `surface_elevation` (units: *m*)

The surface mass balance is computed using the following formula:

```
SMB(x,y,t) = SMB_ref(x,y) + aSMB(x,y,t) + dSMBdz(x,y) * [h(x,y,t) - h_ref(x,y)]
```

Frontal melt parameterization

Use the `discharge_given` frontal melt model to implement the ISMIP6 frontal melt parameterization.

```
pismr -frontal_melt discharge_given \
    -frontal_melt_discharge_given_file forcing.nc ...
```

The file `forcing.nc` has to contain variables `theta_ocean` (potential temperature of adjacent ocean, *degrees Celsius*) and `subglacial_discharge` (water flux per unit area of submerged ice front, $kg/(m^2 s)$).

These inputs are used in the frontal melt parameterization described in [121]:

$$q_m = (A h q_{sg}^\alpha + B) \theta^\beta$$

Here q_m is the frontal melt rate in *m/day*, h is the water depth at an ice front, θ in the *thermal forcing* and A, B, α, β are model parameters.

Parameterized front retreat

To use the parameterized front retreat mechanism use the *Prescribed front retreat* mechanism.

```
pismr -front_retreat_file retreat_forcing.nc ...
```

The file `retreat_forcing.nc` should contain the variable `land_ice_area_fraction_retreat` which defines the maximum ice extent at a given time.

Mass losses resulting from applying this mechanism are reported as a part of `tendency_of_ice_amount_due_to_discharge` and related diagnostics (i.e. they are *not* attributed to calving or frontal melt).

Output variables

See Table 3.27 for a list of variables requested by ISMIP6. Note that they have names different from the ones listed in *Spatially-variable fields* and use MKS units. To reduce the amount of post-processing output files require PISM can follow these conventions.

Setting `output.ISMIP6` makes PISM save diagnostics using MKS units and recognize ISMIP6 variable names.

To save *all* the diagnostics requested by ISMIP6 use the short-cut

```
pismr -extra_vars ismip6 ...
```

The list of variables is stored in the configuration parameter `output.ISMIP6_extra_variables` and contains variables Greenland projections are required to provide. (Add `base`, `ligrundf` to this list for Antarctic projections.)

To save all the time series supported by PISM, omit the `-ts_vars` option:

```
pismr -ts_times TIMES -ts_file ts.nc
```

To save all variables requested by ISMIP6, use `-ts_vars ismip6`:

```
pismr -ts_times TIMES -ts_file ts.nc -ts_vars ismip6
```

Table 3.27: ISMIP6 variables

Variable	Units	Description
<code>lithk(x,y,t)</code>	m	Ice thickness
<code>orog(x,y,t)</code>	m	Surface elevation
<code>base(x,y,t)</code>	m	Base elevation
<code>topg(x,y,t)</code>	m	Bedrock elevation
<code>hfgeoubed(x,y)</code>	W m-2	Geothermal heat flux
<code>acabf(x,y,t)</code>	kg m-2 s-1	Surface mass balance flux
<code>libmassbfgr(x,y,t)</code>	kg m-2 s-1	Basal mass balance flux beneath grounded ice
<code>libmassbffl(x,y,t)</code>	kg m-2 s-1	Basal mass balance flux beneath floating ice
<code>dlithkdt(x,y,t)</code>	m s-1	Ice thickness imbalance
<code>xvelsurf(x,y,t)</code>	m s-1	Surface velocity in x
<code>yvelsurf(x,y,t)</code>	m s-1	Surface velocity in y
<code>zvelsurf(x,y,t)</code>	m s-1	Surface velocity in z
<code>xvelbase(x,y,t)</code>	m s-1	Basal velocity in x
<code>yvelbase(x,y,t)</code>	m s-1	Basal velocity in y

continues on next page

Table 3.27 – continued from previous page

Variable	Units	Description
<code>zvelbase(x,y,t)</code>	m s^{-1}	Basal velocity in z
<code>xvelmean(x,y,t)</code>	m s^{-1}	Mean velocity in x
<code>yvelmean(x,y,t)</code>	m s^{-1}	Mean velocity in y
<code>litemptop(x,y,t)</code>	K	Surface temperature
<code>litempbottgr(x,y,t)</code>	K	Basal temperature beneath grounded ice sheet
<code>litempbottfl(x,y,t)</code>	K	Basal temperature beneath floating ice shelf
<code>strbasemag(x,y,t)</code>	Pa	Basal drag
<code>licalvf(x,y,t)</code>	$\text{kg m}^{-2} \text{s}^{-1}$	Calving flux
<code>lifmassbf(x,y,t)</code>	$\text{kg m}^{-2} \text{s}^{-1}$	Ice front melt and calving flux
<code>ligroundf(x,y,t)</code>	$\text{kg m}^{-2} \text{s}^{-1}$	Grounding line flux
<code>sftgif(x,y,t)</code>	1	Land ice area fraction
<code>sftgrf(x,y,t)</code>	1	Grounded ice sheet area fraction
<code>sftflf(x,y,t)</code>	1	Floating ice sheet area fraction
<code>lim(t)</code>	kg	Total ice mass
<code>limnsw(t)</code>	kg	Mass above floatation
<code>iareagr(t)</code>	m^2	Grounded ice area
<code>iareafl(t)</code>	m^2	Floating ice area
<code>tendacabf(t)</code>	kg s^{-1}	Total SMB flux
<code>tendlibmassbf(t)</code>	kg s^{-1}	Total BMB flux
<code>tendlibmassbfff(t)</code>	kg s^{-1}	Total BMB flux beneath floating ice
<code>tendlcalvf(t)</code>	kg s^{-1}	Total calving flux
<code>tendlifmassbf(t)</code>	kg s^{-1}	Total calving and ice front melting flux
<code>tendligroundf(t)</code>	kg s^{-1}	Total grounding line flux

3.6 Simplified geometry experiments

There have been three stages of ice sheet model intercomparisons based on simplified geometry experiments since the early 1990s [125].

EISMINT I (European Ice Sheet Modeling INiTiative) [34]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 superceded, however, by verification using the full variety of known exact solutions to the isothermal SIA [33]. The “rediscovery”, since EISMINT I, of the Halfar similarity solution with zero accumulation [126], 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 [22] pointed out interesting and surprising properties of the thermocoupled SIA. References [30], [127], [128], [47], [129], [29] 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 [130], [47], [30] adequately explains the “spokes” in EISMINT II experiment F as a genuine fluid instability, while [48] and Appendix B of [29] 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,

¹ See <http://homepages.vub.ac.be/~phuybrec/eismint.html>

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 [131], [132], HEINO = Heinrich Event INtercOmparison [133], [104], MISMIP (below), and MISMIP3d (also below).

PISM did not participate in ISMIP-HOM but does support most of prescribed experiments (see [ISMIP-HOM](#)).

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 ([29], 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 [29], 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](#)).

A third and fourth ISMIP parts are the two parts of the Marine Ice Sheet Model Intercomparison Project, MISMIP [134] and MISMIP3D [135]. These experiments are supported in PISM, as described in subsections [MISMIP](#) and [MISMIP3d](#) below.

3.6.1 EISMINT II

There are seven experiments described in the published EISMINT II writeup [22]. 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 [22]; 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 ([29], 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 [22], there were an additional five experiments described in the EISMINT II intercomparison description [124], 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 3.28](#) for how to run supported EISMINT II experiments in PISM. Experiments E – L are only documented in [124].

Note: Experiments G and H are not supported.

² See <http://homepages.vub.ac.be/~phuybrec/ismip.html>

Table 3.28: Running the EISMINT II experiments in PISM. Use `-skip` `-skip_max 5`, on the 61×61 default grid, for significant speedup.

Command: “ <code>pismr +</code> ”	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 [30]
<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 3.28). 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 3.28 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` `-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 3.29 lists some optional settings which are particular to the EISMINT II experiments.

Table 3.29: Changing the default settings for EISMINT II

Option	Default values [experiments]	Units	Meaning
<code>-eisII</code>	A		Choose single character name of EISMINT II [22] simplified geometry experiment. See Table 3.28.
<code>-Mmax</code>	0.5 [ABDEFIK], 0.25 [CJL]	$m/year$	max value of accumulation rate
<code>-Rel</code>	450 [ABEFIK], 425 [CDJL]	km	radial distance to equilibrium line
<code>-Sb</code>	10^{-2} [<i>all</i>]	$(m/year)/km$	radial gradient of accumulation rate
<code>-ST</code>	1.67×10^{-2} [<i>all</i>]	K/km	radial gradient of surface temperature
<code>-Tmin</code>	238.15 [ACDEIJKL], 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 [134]; 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 [137] 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 [138]. 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 MISMIP 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 MISMIP experiments with either of two applicable ice dynamics models. Model 1 is a pure SSA model; “category 2” in the MISMIP classification. Model 2 combines SIA and SSA velocities as described in [37]; “category 3” because it resolves “vertical” shear (i.e. using SIA flow).

There are many runs for a complete MISMIP 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 MISMIP 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 MISMIP 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/mismip/mismip2d/README.md`

for usage of the scripts that run MISMIP 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. 3.22 (left).

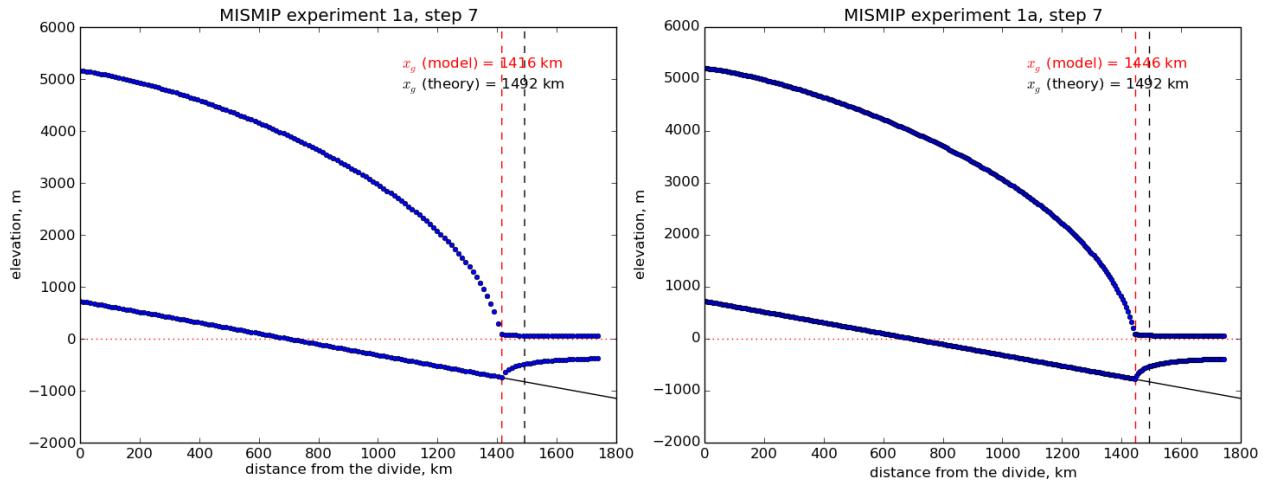


Fig. 3.22: A marine ice sheet profile in the MISMIP intercomparison; PISM model 1, experiment 1a, at step 7. Left: grid mode 1 (12 km grid). Right: grid mode 3 (6 km grid).

The script `MISMIP.py` in `examples/mismip/mismip2d` has the ability to compute the profile from the Schoof's [137] 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. 3.23. By default `run.py` uses the asymptotic-matching thickness result from the [137] theory to initialize the initial ice thickness, as allowed by the MISMIP specification.

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. 3.22. The corresponding results from grid mode 2, with 1.2 km spacing, are in Figure Fig. 3.24. 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 [135].

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

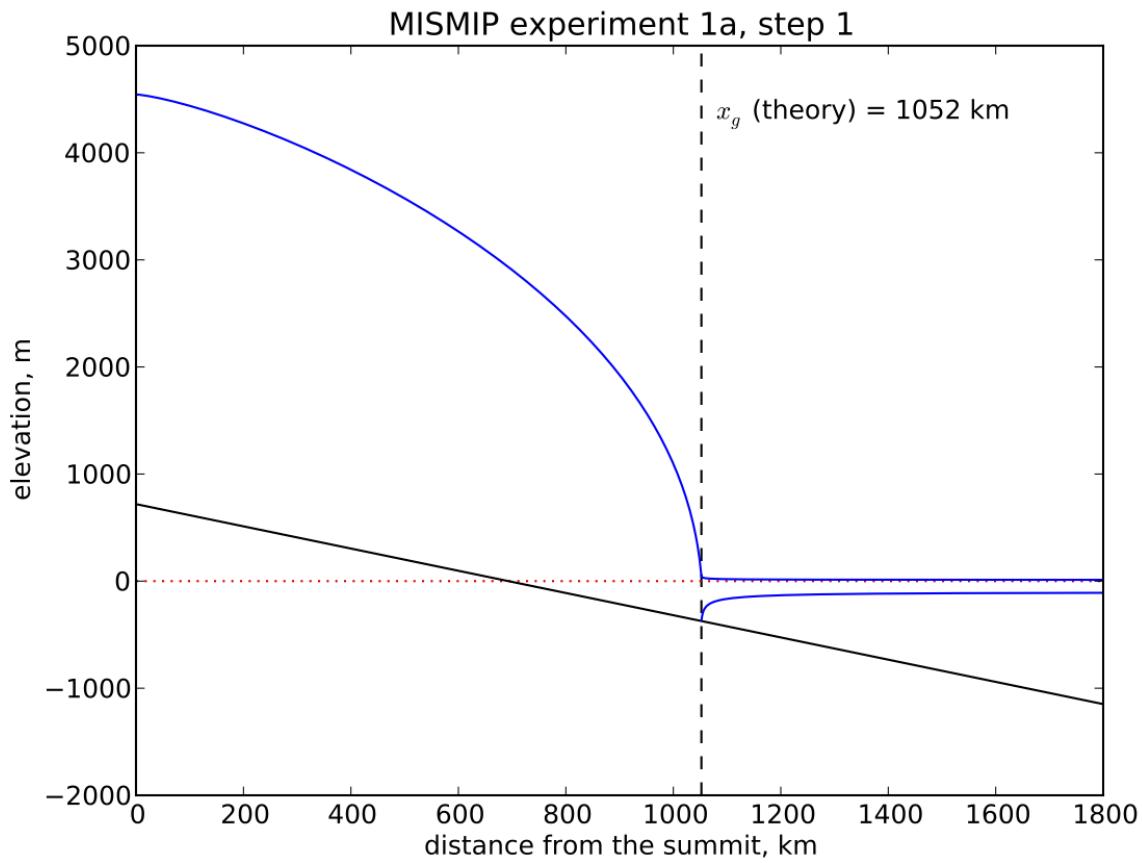


Fig. 3.23: Analytical profile for steady state of experiment 1a, step 1, from theory in [137]. This is a boundary layer asymptotic matching result, but not the exact solution to the equations.

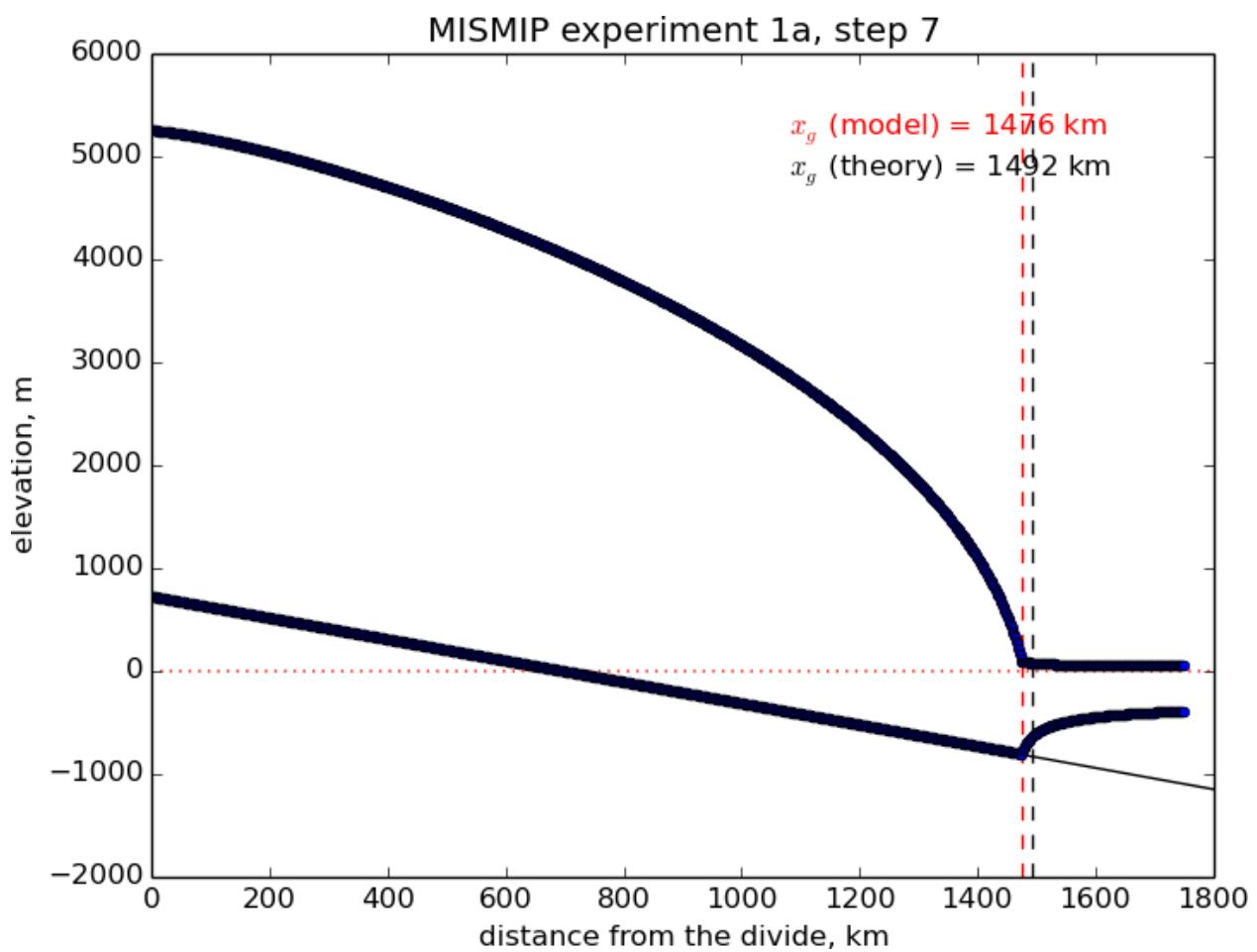


Fig. 3.24: Results from MISMIP grid mode 2, with 1.2 km spacing, for steady state of experiment 1a: profile at step 7 (compare Fig. 3.22).

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 [135] 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 [101].

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. 3.25. 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. 3.26 for a comparison of SSA and SIA velocities in the MISMIP3D geometry. Note that both Figures Fig. 3.25 and Fig. 3.26 were generated with resolution of $\Delta x = \Delta y = 1$ km.

3.6.4 ISMIP-HOM

The ISMIP-HOM intercomparison project ([136], [131]) consists of 8 experiments:

- A: ice flow over a rippled bed,
- B: flow over a rippled bed along a flowline (similar to A, but the basal topography does not depend on y),
- C: ice stream flow over a flat sloping bed with a spatially-variable basal friction coefficient β ,
- D: ice stream flow along a flowline (similar to C, but β does not depend on y)
- E1: simulation of the flow along the central flowline of Haut Glacier d’Arolla with a no-slip boundary condition,
- E2: same as E1 but with a 300m zone of zero traction.
- F1: prognostic experiment modeling the relaxation of the free surface towards a steady state with zero surface mass balance,
- F2: same as F1 but with a different slip ratio.

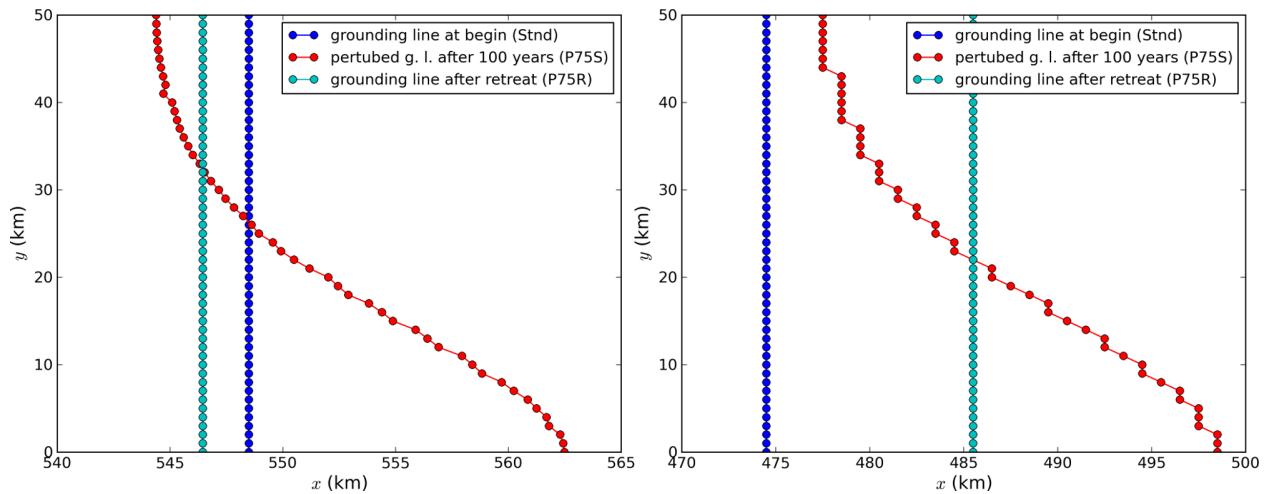


Fig. 3.25: Comparison between the grounding lines of the higher-amplitude (“P75”) MISMIP3d experiments performed with PISM when using the subgrid grounding line interpolation method (left) or not using it (right). In both cases the SIA+SSA hybrid is used.

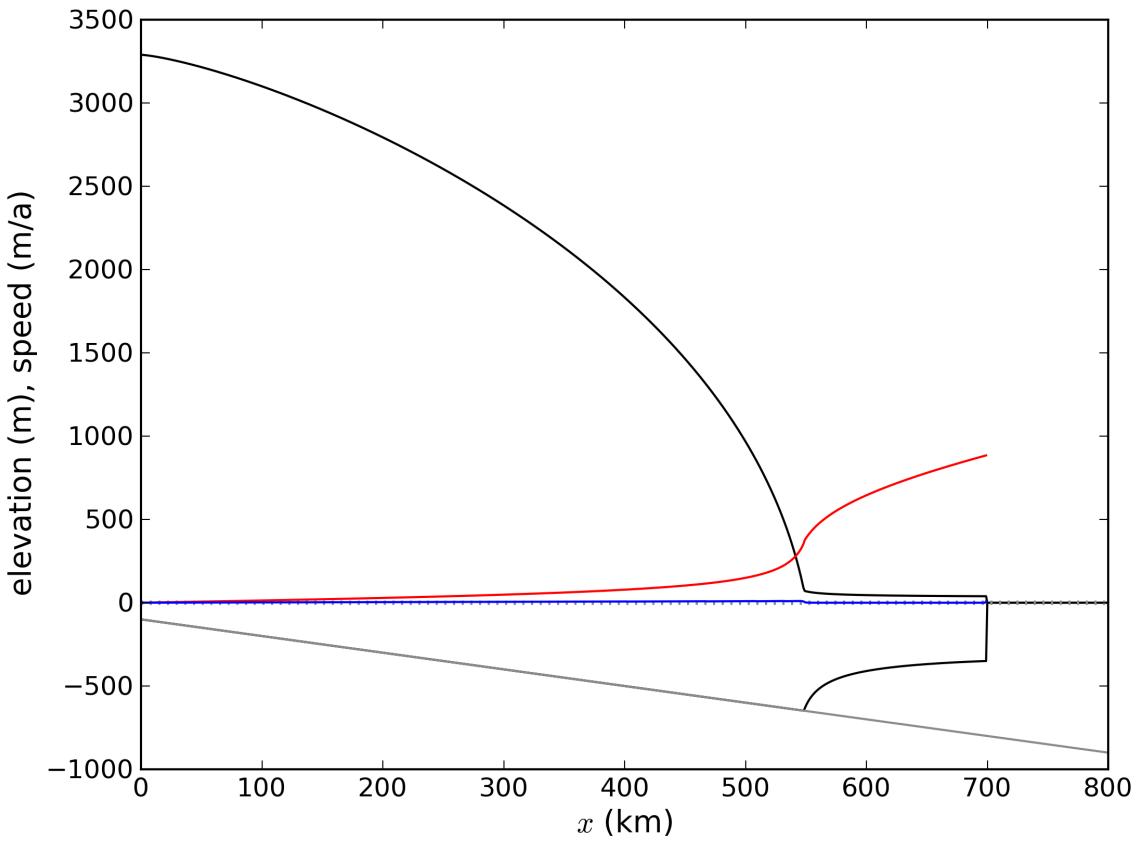


Fig. 3.26: The SIA velocities are negligible in the MISMIP3d standard experiment (“Stnd”). The steady state ice geometry is plotted (black) together with the computed SSA velocity (red) and SIA velocity (blue). The SIA velocity reaches its maximum value of about 10 m/a at the grounding line, about two orders of magnitude less than the maximum of the SSA velocity.

All experiments use the isothermal Glen flow law and all sliding flow uses the linear sliding law.

We implement ISMIP-HOM experiments A, B, C, D, E1, E2 and report computed ice velocities.

Experiments A, B, C, D

Experiments A through D require additional code to implement periodic input geometry; see `examples/ismip-hom/abcd` and PISM's source code for details.

Figures Fig. 3.27, Fig. 3.28, Fig. 3.29, and Fig. 3.30 compare PISM's results to some "LMLa" (Blatter-Pattyn) and "FS" (Stokes) models from [131] using data provided in the supplement.

Note: We exclude 1 model (mbr1) from Fig. 3.29 and three models (rhi1, rhi2, rhi3) from Fig. 3.30. Either corresponding data in the supplement are wrong or these results were excluded from figures 8 and 9 in [131] as well.

In all of the figures below results from individual models are plotted using faint green (Blatter-Pattyn) and orange (Stokes) lines.

In Fig. 3.29 for the length scale L of 5km PISM's results coincide with the whole set of curves corresponding to Stokes models: an outlier among Blatter-Pattyn models distorts the vertical scale of the plot.

PISM results below use $101 \times 101 \times 5$ grid points for experiments A and C and 101×5 for B and D. All 24 diagnostic computations complete in under 2 minutes on a 2020 laptop.

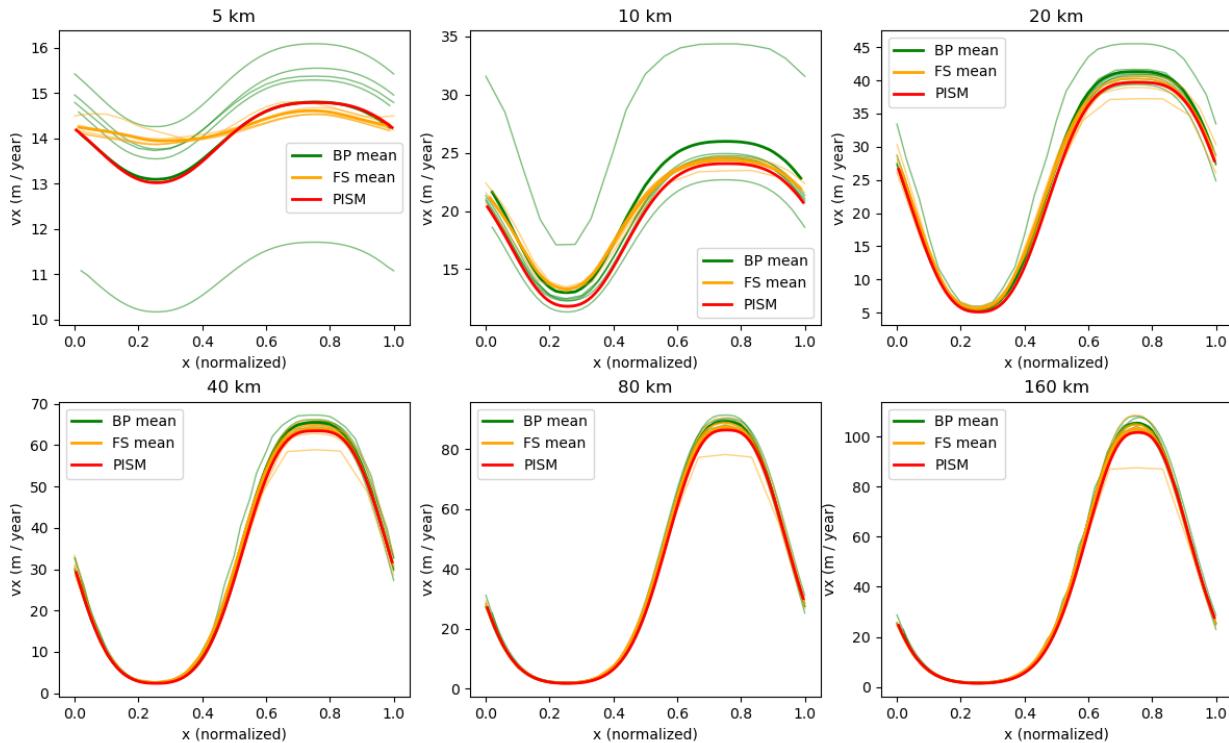


Fig. 3.27: Surface velocity at $y = 0.25L$ for the Experiment A

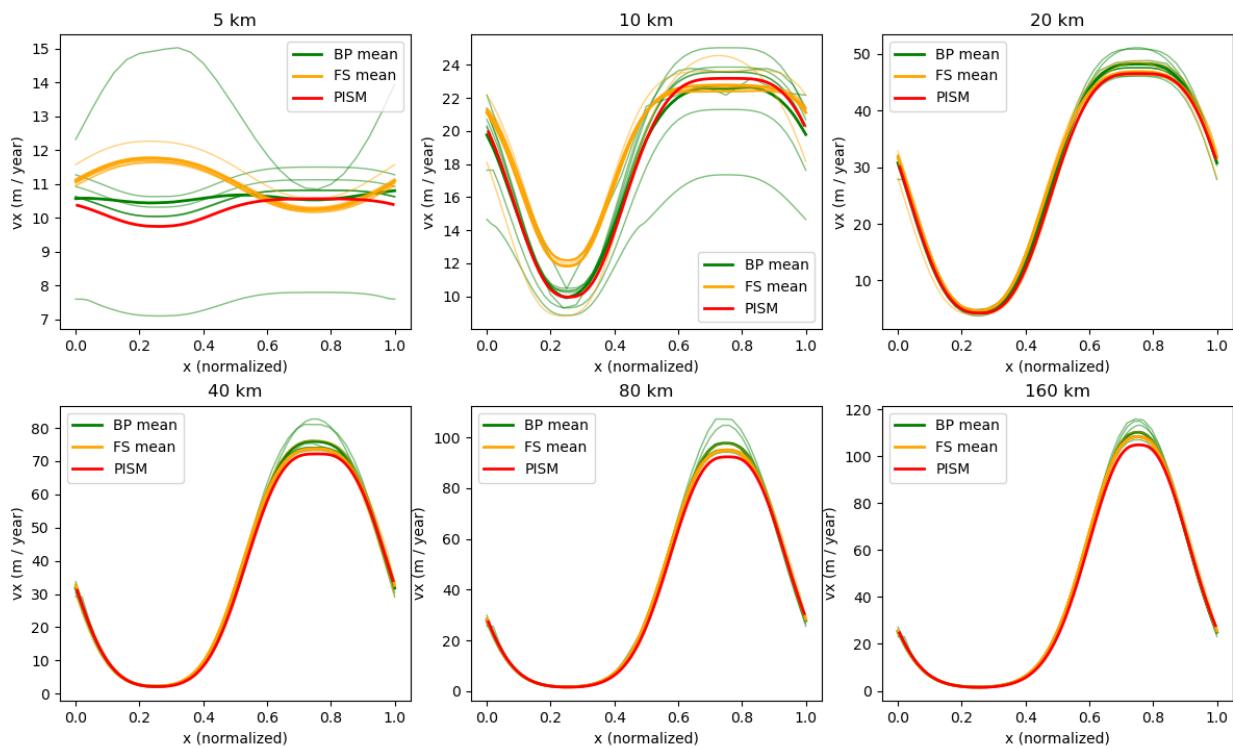


Fig. 3.28: Surface velocity for the Experiment B

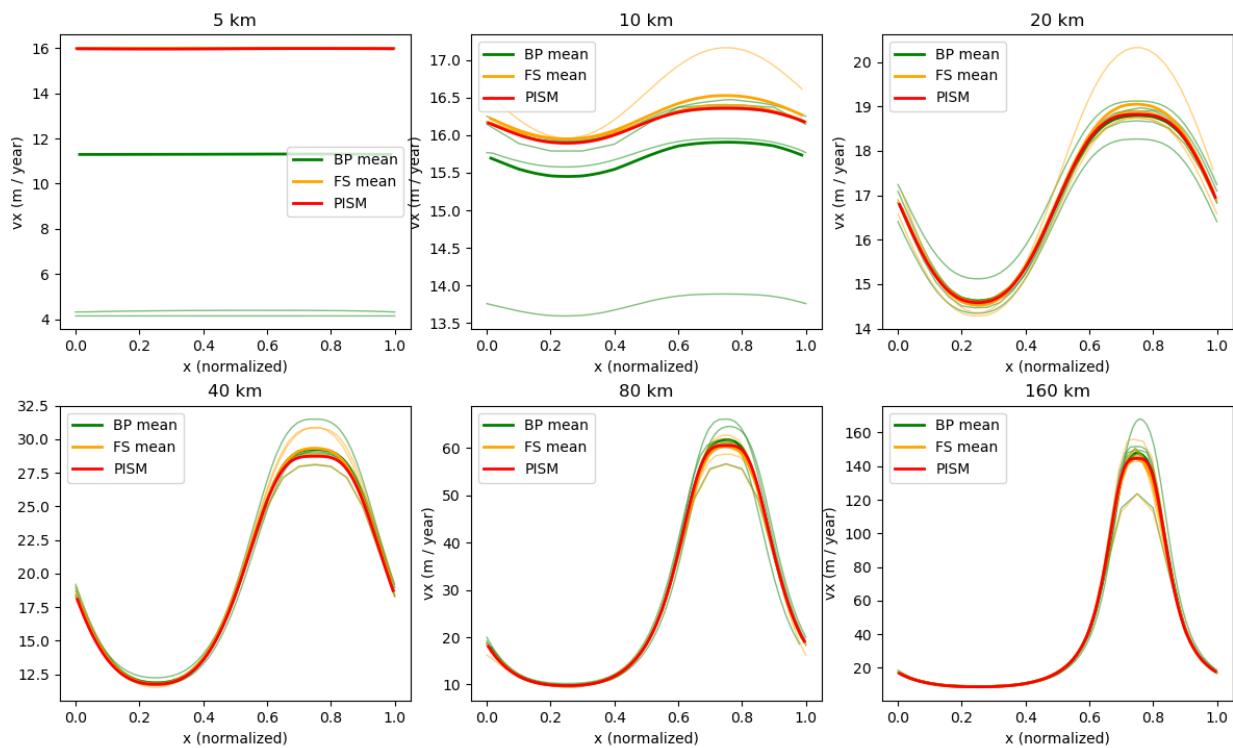


Fig. 3.29: Surface velocity at $y = 0.25L$ for the Experiment C

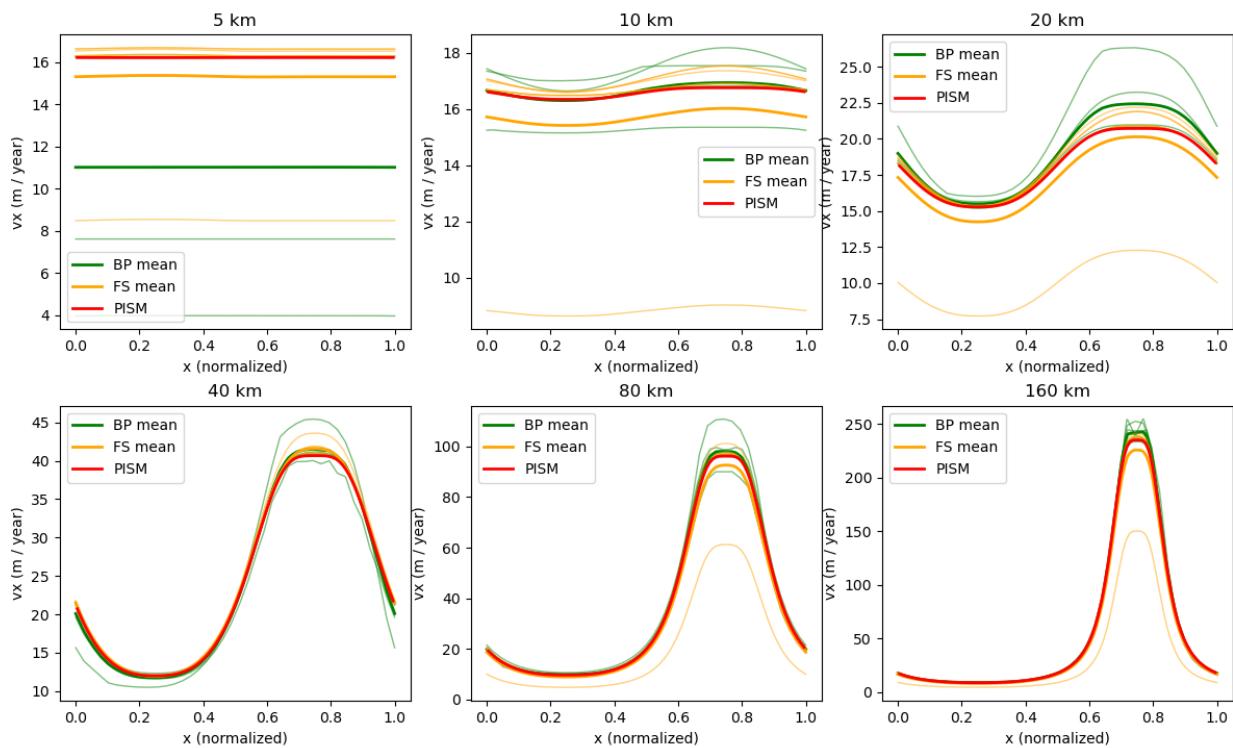


Fig. 3.30: Surface velocity for the Experiment D

Experiment E

Unlike simplified-geometry experiments A–D, the diagnostic simulation of the flow along the central flowline of Haut Glacier d’Arolla does not require any code modifications and uses the `pismr` executable. Please see `examples/ismip-hom/e-arolla` for details.

The complete command used to produce Fig. 3.31, Fig. 3.32, and Fig. 3.33 is below:

```
# set ice softness
A=$(echo "scale=50; 10^(-16) / (365.2524 * 86400.0)" | bc -l)

pismr -i ${input} -bootstrap \
-Mx 401 \
-grid.registration corner \
-grid.periodicity y \
-stress_balance.model blatter \
-stress_balance.blatter.flow_law isothermal_glen \
-flow_law.isothermal_Glen.ice_softness ${A} \
-stress_balance.blatter.coarsening_factor 7 \
-blatter_Mz 50 \
-bp_snes_monitor_ratio \
-bp_ksp_type gmres \
-bp_pc_type mg \
-bp_pc_mg_levels 3 \
-bp_mg_levels_ksp_type richardson \
-bp_mg_levels_pc_type sor \
-bp_mg_coarse_ksp_type preonly \
-bp_mg_coarse_pc_type lu \
-basal_resistance.pseudo_plastic.enabled \
-basal_resistance.pseudo_plastic.q 1.0 \
-basal_resistance.pseudo_plastic.u_threshold 3.1556926e7 \
-basal_yield_stress.model constant \
-energy none \
-geometry.update.enabled false \
-atmosphere uniform \
-atmosphere.uniform.precipitation 0 \
-surface simple \
-y 1e-16 \
-o ${output}
```

This run uses the 12.5 m grid resolution along the flowline and 50 vertical levels, which corresponds to the vertical resolution of under 5 meters where the ice is thickest.

This grid is small enough to perform the diagnostic computation serially, avoiding dependence on parallel direct solvers (e.g. MUMPS) that may be needed in parallel.

We use 3 multigrid levels with a *very* aggressive coarsening factor (7).

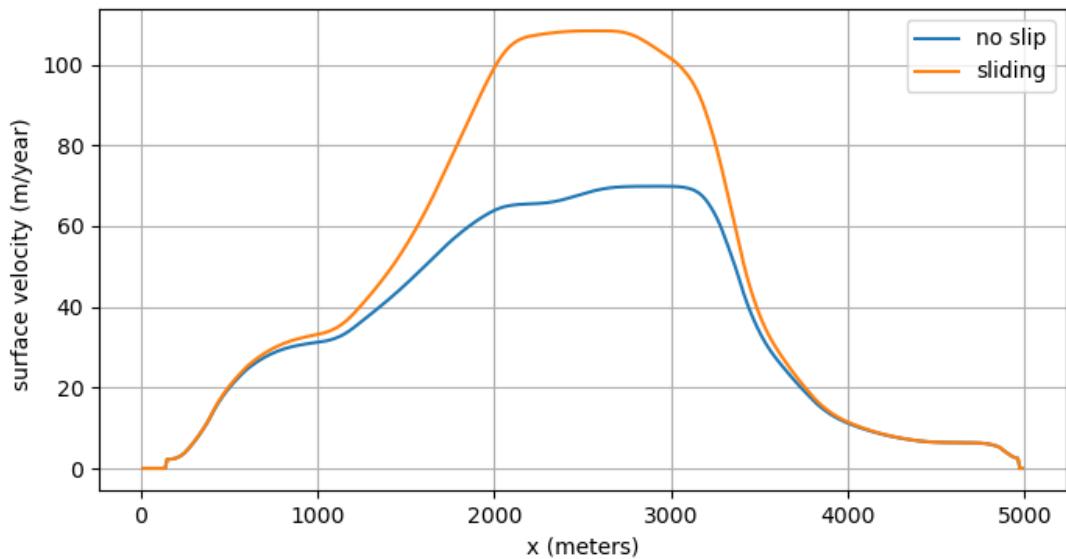


Fig. 3.31: Surface ice velocity for the Experiment E

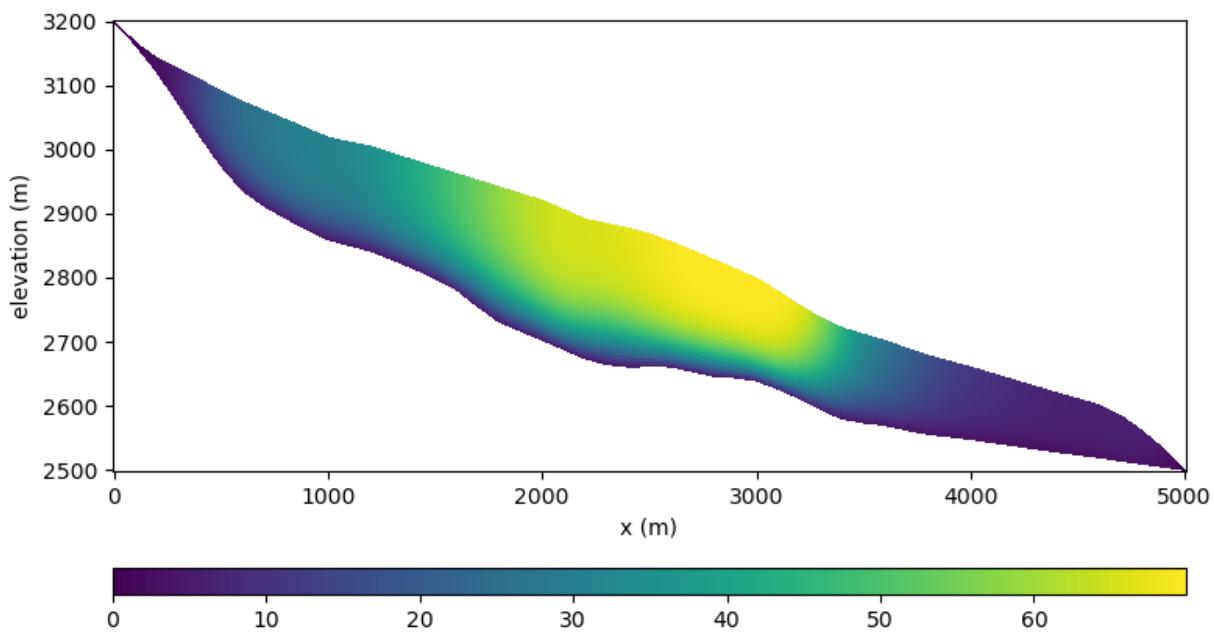


Fig. 3.32: Ice velocity for the Experiment E1 (no slip)

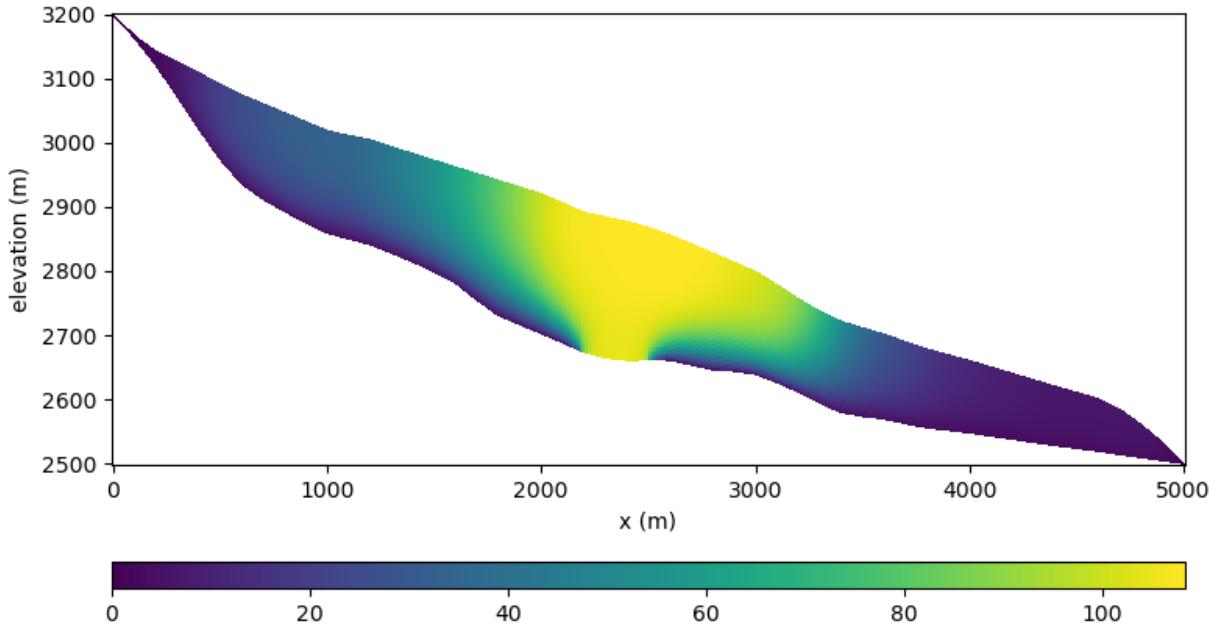


Fig. 3.33: Ice velocity for the Experiment E2 (zone of zero traction)

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 [151]

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 [150] gives a broad discussion of verification and validation in computational fluid dynamics. See [33] and [30] for discussion of verification issues for the isothermal and thermomechanically coupled shallow ice approximation (SIA), respectively, and for exact solutions to these models, and [29], [45] 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 3.30: Exact solutions for verification

Test	Continuum model tested	Reference	Comments
A	isothermal SIA, steady, flat bed, constant accumulation	[33]	
B	isothermal SIA, flat bed, zero accumulation	[33], [126]	similarity solution
C	isothermal SIA, flat bed, growing accumulation	[33]	similarity solution
D	isothermal SIA, flat bed, oscillating accumulation	[33]	uses compensatory accumulation
E	isothermal SIA; as A, but with sliding in a sector	[33]	uses compensatory accumulation
F	thermomechanically coupled SIA (mass and energy conservation), steady, flat bed	[78], [30]	uses compensatory accumulation and heating
G	thermomechanically coupled SIA; as F but with oscillating accumulation	[78], [30]	ditto
H	bed deformation coupled with isothermal SIA	[27]	joined similarity solution
I	stream velocity computation using SSA (plastic till)	[45], [29]	
J	shelf velocity computation using SSA	(source code)	
K	pure conduction in ice and bedrock	[159]	
L	isothermal SIA, steady, non-flat bed	(source code)	numerical ODE solution

Table 3.31: Canonical PISM verification runs using the exact solutions listed in Table 3.30.

Test	Example invocation
A	pismv -test A -Mx 61 -My 61 -Mz 11 -y 25000
B	pismv -test B -Mx 61 -My 61 -Mz 11 -ys 422.45 -y 25000
C	pismv -test C -Mx 61 -My 61 -Mz 11 -y 15208.0
D	pismv -test D -Mx 61 -My 61 -Mz 11 -y 25000
E	pismv -test E -Mx 61 -My 61 -Mz 11 -y 25000
F	pismv -test F -Mx 61 -My 61 -Mz 61 -y 25000
G	pismv -test G -Mx 61 -My 61 -Mz 61 -y 25000
H	pismv -test H -Mx 61 -My 61 -Mz 11 -y 40034 -bed_def iso
I	ssa_testi -ssa_method fd -Mx 5 -My 500 -ssafdfpicard_rtol 1e-6 -ssafdksp_rtol 1e-11
J	ssa_testj -ssa_method fd -Mx 60 -My 60 -ssafdfksp_rtol 1e-12
K	pismv -test K -Mx 6 -My 6 -Mz 401 -Mbz 101 -y 130000
L	pismv -test L -Mx 61 -My 61 -Mz 31 -y 25000

Table 3.32: pismv command-line options

Option	Description
-test	Choose verification test by single character name; see Table 3.30.
-no_report	Do not report errors at the end of a verification run.
-eo	Only evaluate the exact solution; no numerical approximation at all.
-report_file	Save error report to a netCDF file.
-append	Append to a report file.

Table 3.30 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 3.31 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 3.32 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 [150]. The refinement path is defined by a sequence of spatial grid cell sizes which decrease toward the refinement limit of zero size [71]. 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 [30]:

```
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 (`mpiexec -n NN pismv`) and of the `-skip` modification to adaptive timestepping accelerates such fine-grid runs; see section [Understanding adaptive time-stepping](#).

Figures Fig. 3.34 through Fig. 3.38 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/vfnow.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 [33]. For the errors in test I, the exact continuum solution is not very smooth at the free boundary [45].

3.7.1 Sample convergence plots

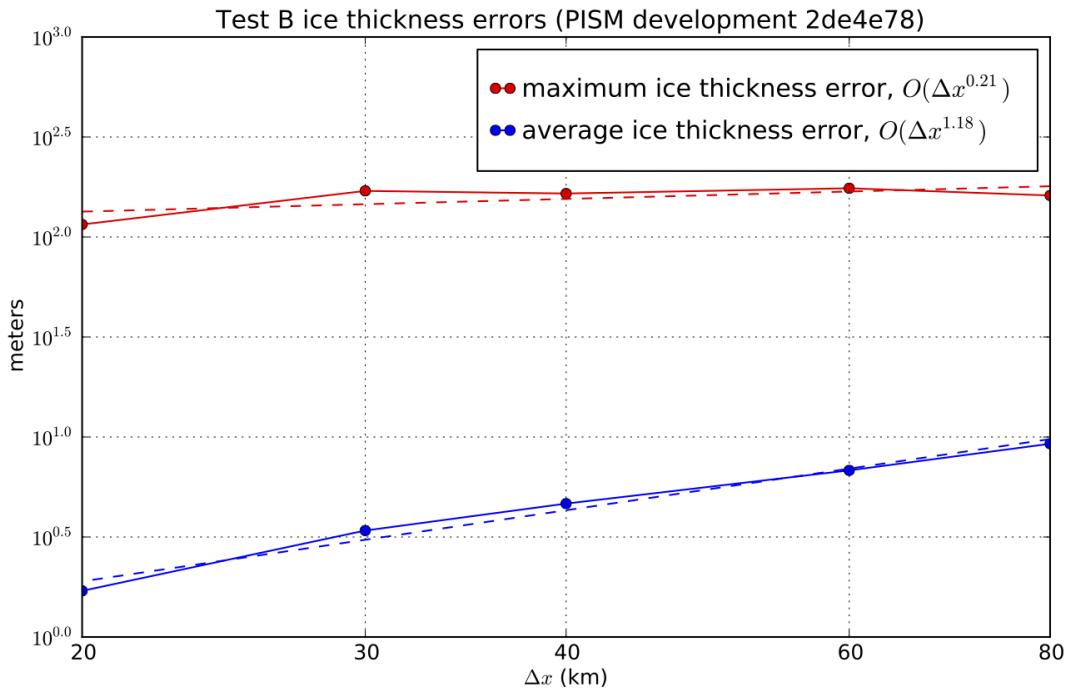


Fig. 3.34: Numerical thickness errors in test B. See [33] for discussion.

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 [150], [151]. 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 [152], [153]. 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.

¹ Which explains the rise of “simplified geometry intercomparisons”; see section [Simplified geometry experiments](#).

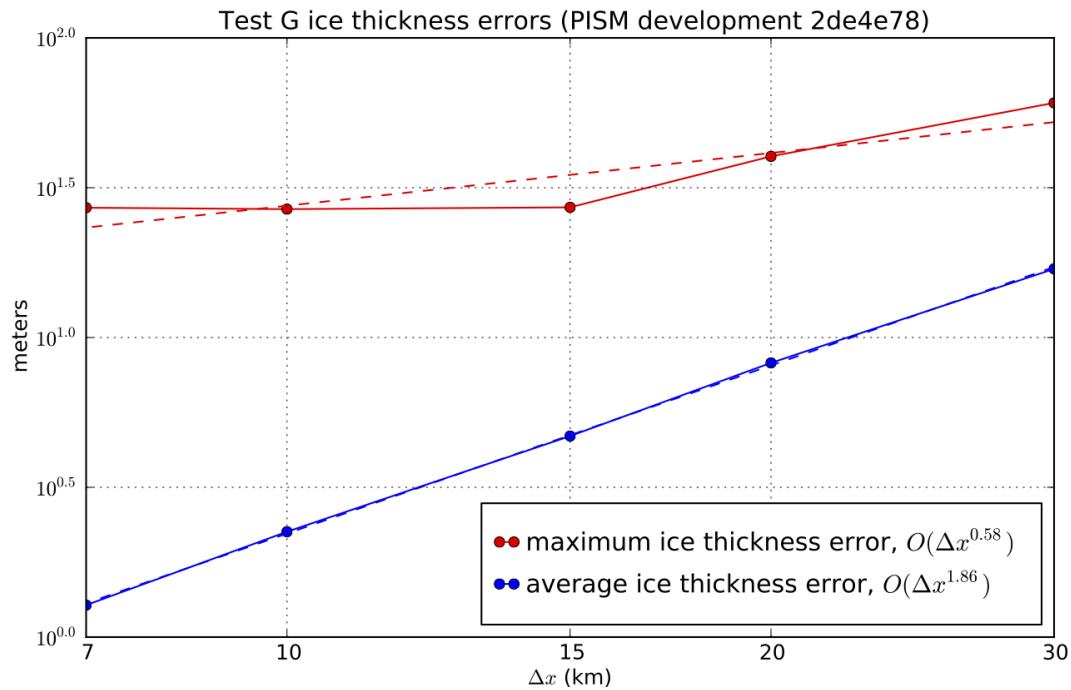


Fig. 3.35: Numerical thickness errors in test G. See [30] and [33].

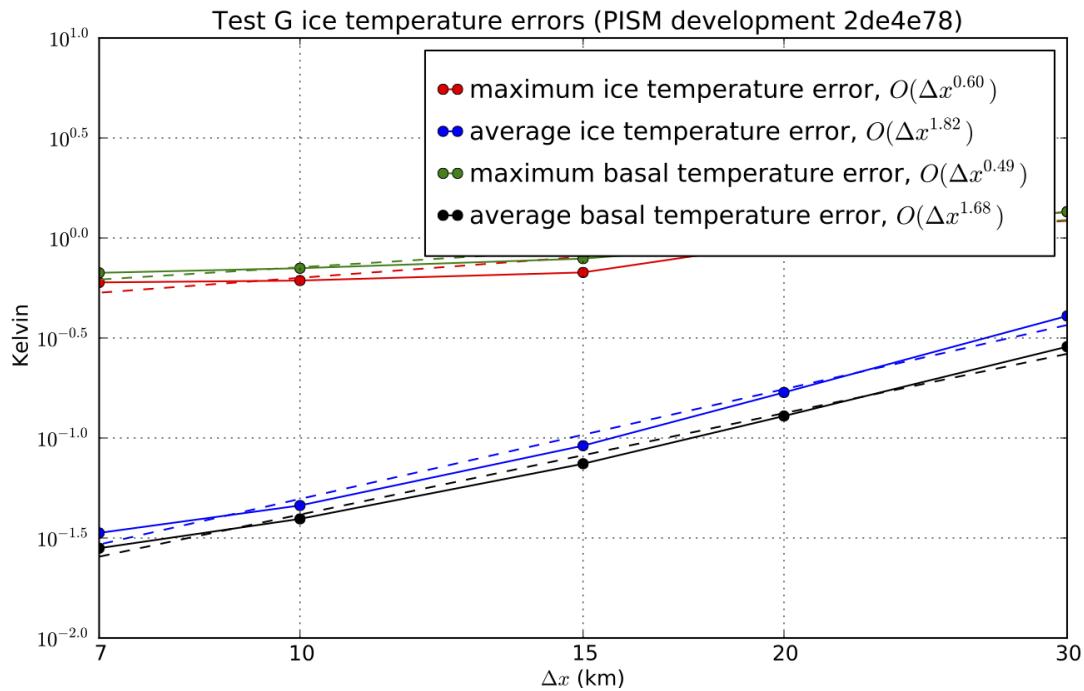


Fig. 3.36: Numerical temperature errors in test G. See [30].

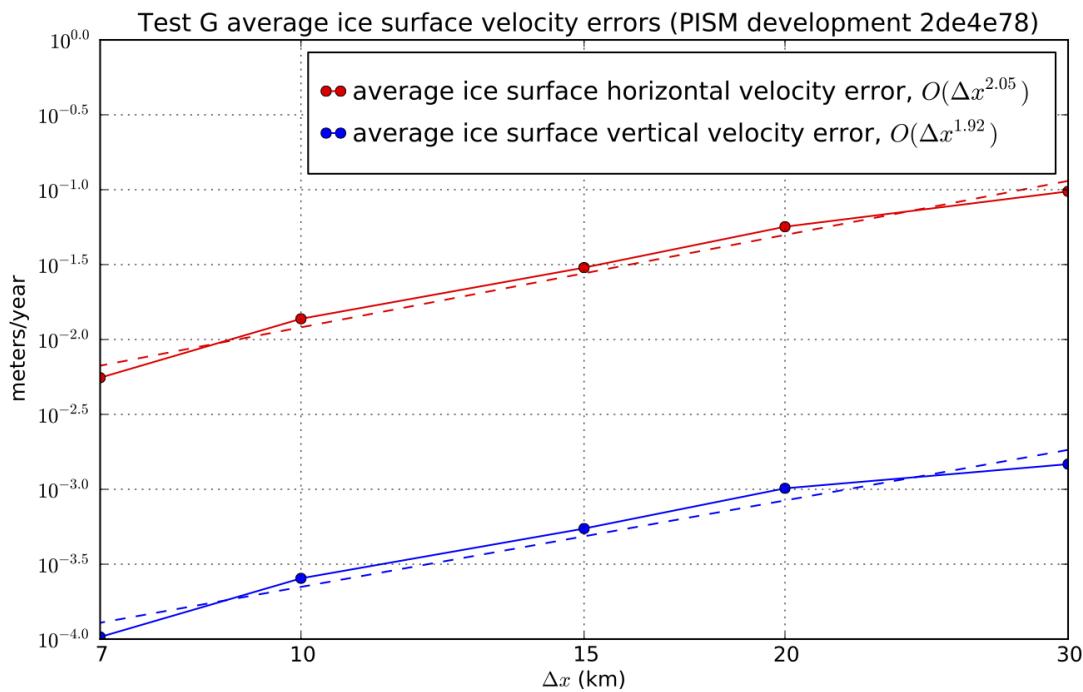


Fig. 3.37: Numerical errors in computed surface velocities in test G.

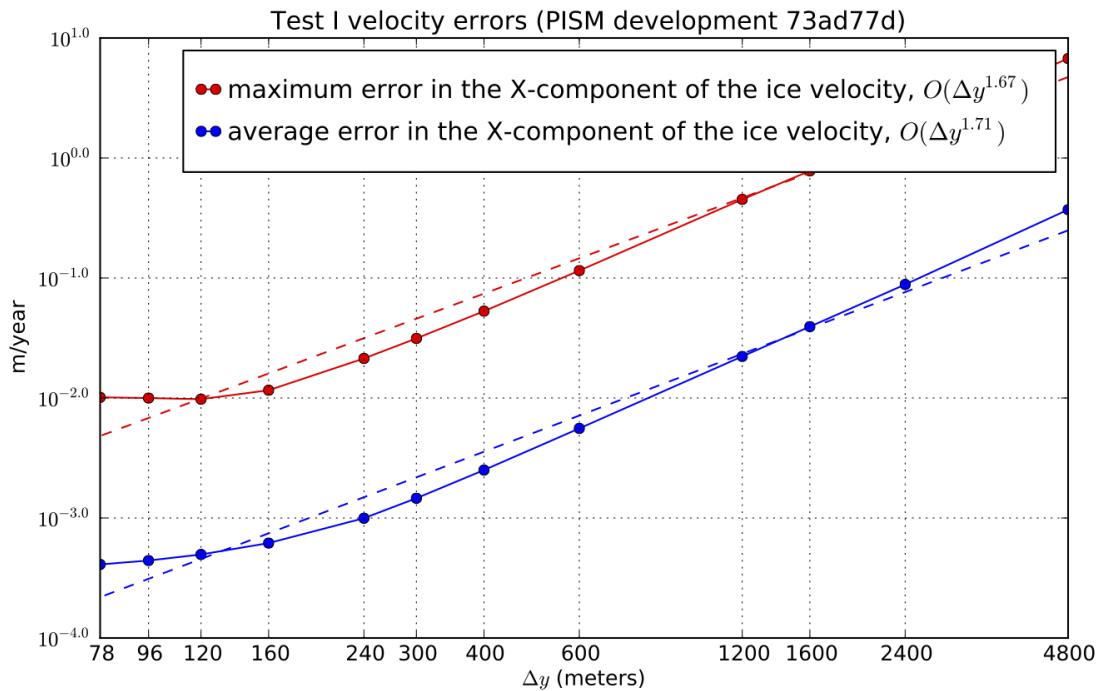


Fig. 3.38: Numerical errors in horizontal velocities in test I, an ice stream. See [45], [29].

We compare our numerical results to the “constant-flux” experiment from [152]. Fig. 3.39 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. 3.40.

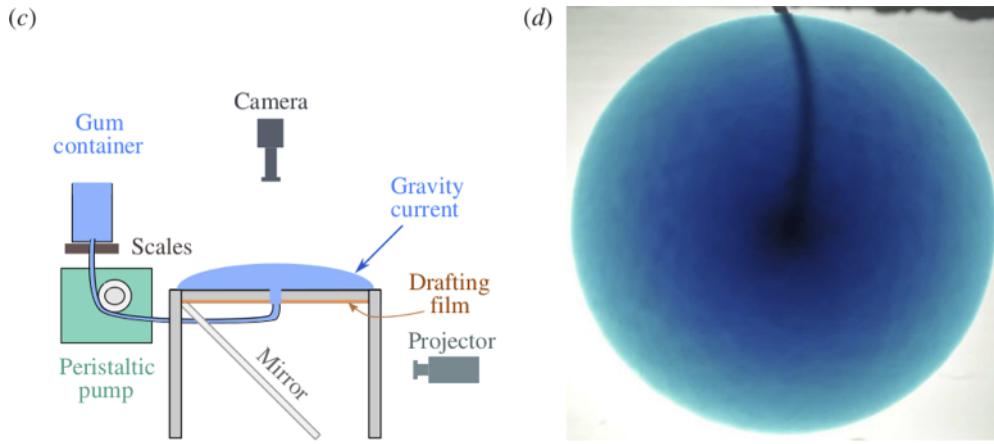


Fig. 3.39: Reproduction of Figures 2(c) and 2(d) from [152]. Left: experimental apparatus used for “constant-flux release” experiment. Right: snapshot of constant-flux experiment (plan view), showing an axisymmetric front.

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.

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 [152] on the 10 mm grid on a 520 mm \times 520 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. 3.40:

```

./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

```

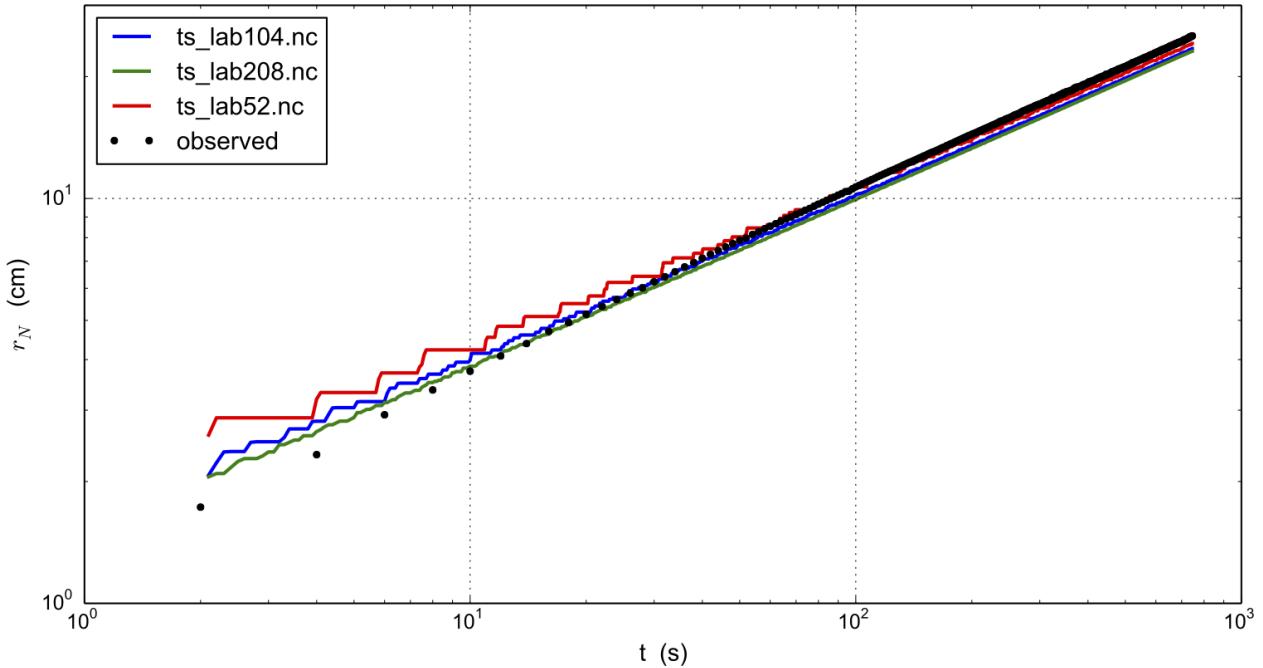


Fig. 3.40: Radius $r_N(t)$ for runs with 10 mm (`ts_lab52.nc`), 5 mm (`ts_lab104.nc`), and 2.5 mm (`ts_lab208.nc`) grids, compared to observations from Sayag & Worster's [152] table-top "ice cap" (gravity current) made from a 1% Xanthan gum suspension, as shown in Figure Fig. 3.39.

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 [152], 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. 3.40 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 [68] successfully validated early-1990s ice shelf numerical models using velocity data for the Ross ice shelf. The data were from the RIGGS survey [154], 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 [155] and parameter-sensitivity studies [156]. 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 [157] ice sheet geometry, bedrock, and climate data set, and the radar-derived (InSAR) MEaSUREs Antarctica Velocity Map [158], 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 [89] 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. 3.41.

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_bc` and `v_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 `vel_bc_mask` specifies these locations. Finally the variables `u_bc`, `v_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_bc`, `v_bc`, `vel_bc_mask` described above. The field `vel_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.

¹ High accumulation rates, cold firn with minimal compression, and basal freeze-on of marine ice may all generate significant variation in shelf density.

The script `plot.py` takes PISM output such as `diag_Mx211.nc` to produce Fig. 3.41. 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
```

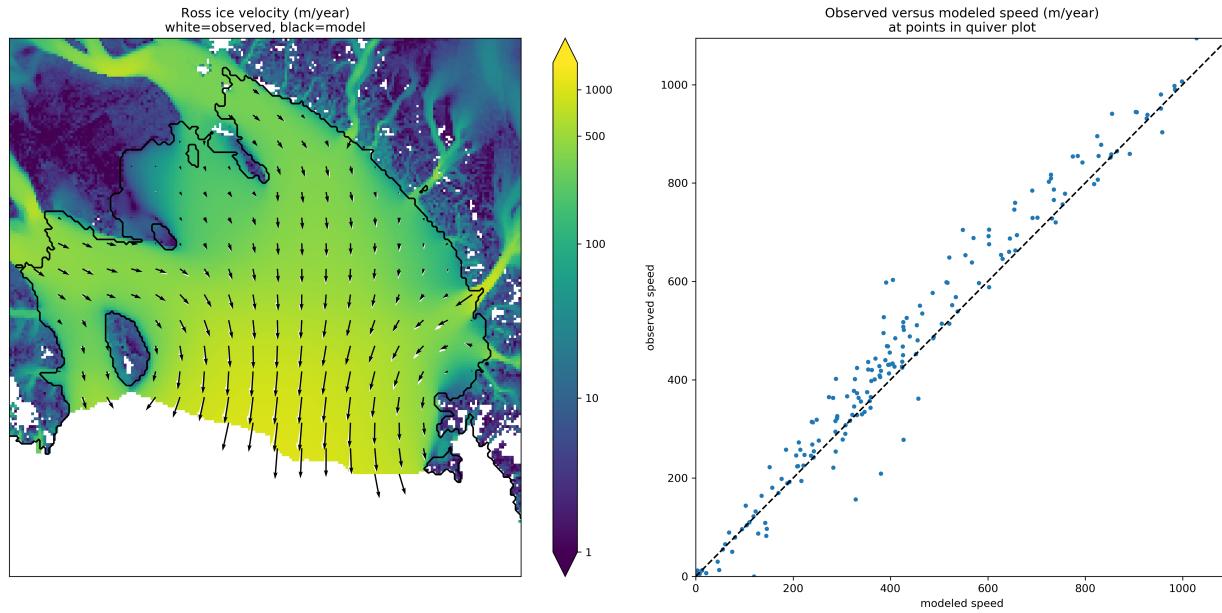


Fig. 3.41: *Left:* Color is speed in m/a. Arrows are observed (white) and modeled (black) velocities. *Right:* Comparison between modeled and observed speeds at points plotted on the left.

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 [92], 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 [89].

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 \
-front_retreat_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 -subg1`, 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 [98]. The option `-kill_icebergs` is essential to maintain well-posedness of the SSA velocity problem at each time step [37]. See section [PIK options for marine ice sheets](#).

Option combination

```
-calving eigen_calving,thickness_calving -eigen_calving_K 1e17 \
-front_retreat_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 [89], namely `eigen_calving` with the given constant K , or if the ice thickness goes below the given threshold of 50 meters. See section [Calving and front retreat](#).

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 [58], an event which seems to have been driven by warmer ocean temperatures [59]. Because it is thick, has a steep surface slope, has a deep trough in its bedrock topography (Figure Fig. 3.42), and has a thick layer of low-viscosity temperate ice at its base [60], this ice flow is different from the ice streams in West Antarctica or Northeast Greenland [50].

This section describes how to build a PISM regional model of this outlet glacier [61] 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 [62] 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 climatic mass balance from the Greenland-region climate model RACMO [63].

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 [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 script from [Getting started: a Greenland ice sheet example](#).

¹ PISM can also do 1 km runs for the whole Greenland ice sheet; see this [news item](#).

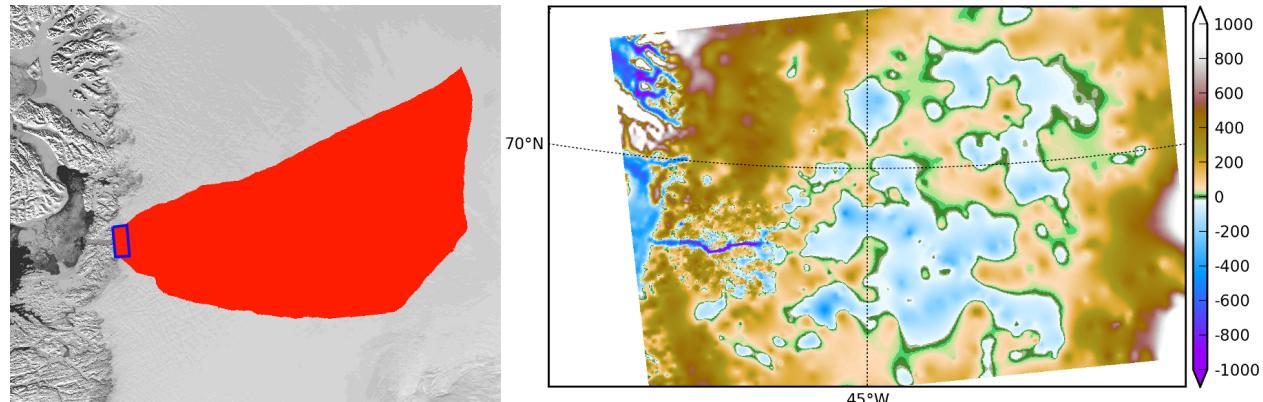


Fig. 3.42: A `regional-tools` script computes a drainage basin mask from the surface DEM (left; Modis background) and from a user-identified terminus rectangle (blue). The regional model can exploit high-resolution bedrock elevations inland from Jakobshavn fjord (right; meters asl).

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/pismRegional.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 [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,

```
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
```

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.

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_bc`, `v_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 `pismRegional.py` computes the drainage basin mask based on a user choice of a “terminus rectangle”; see Figure Fig. 3.42. There are two ways to use this script:

- To use the graphical user interface (GUI) mode.

Run

```
python pismRegional.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 `pismRegional.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:

```
python pismRegional.py -i gr1km.nc -o jakomask.nc -x 360,382 -y 1135,1176 -b 50
```

This call generates the red region in Fig. 3.42. 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 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.

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 `pismRegional.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 `fft_mask` identifies the drainage basin, outside of which we will use simplified time-independent boundary conditions. Specifically, outside of the `fft_mask` area, but within the computational domain defined by the extent of `jako.nc`, we will essentially keep the initial thickness. Inside the `fft_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 `pismRegional.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_bc
```

- Dirichlet boundary conditions `u_bc`, `v_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 command-line options:

```
-front_retreat_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 (see [Prescribed front retreat](#)). 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 prescribing the maximum ice extent, 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.

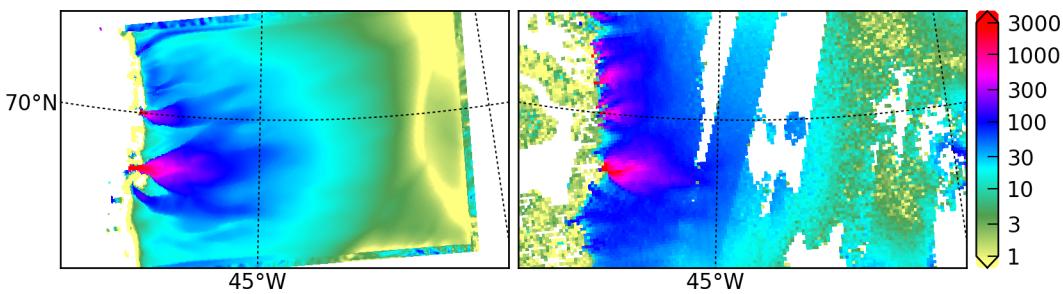


Fig. 3.43: Left: modeled surface speed at the end of a 2 km grid, 100 model year, steady present-day climate run. Right: observed surface speed, an average of four winter velocity maps (2000, 2006–2008) derived from RADARSAT data, as included in the SeaRISE 5 km data set [64], for the same region. Scales 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. 3.43, 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 [65].

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 [66]) or a study of their relation to inverse modeling results (e.g. [67]).

3.9.8 Plotting results

Fig. 3.43 was generated using `pypismtools`, `NCO` and `CDO`. Do

```
ncpdq -a time,z,y,x spunjako_0.nc jako5km.nc
nc2cd0.py jako5km.nc
cdo remapbil,jako5km.nc Greenland_5km_v1.1.nc Greenland_5km_v1.1_jako.nc # FIXME: if fails, u
-ceed?
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 quantity, usually with units),
3. Integers (an integer number, usually corresponding to a count),
4. Flags (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`

Solve age equation (advection equation for ice age).

Type flag

Default value no

Option `-age`

2. `age.initial_value`

Initial age of ice

Type number

Default value 0 (*years*)

3. `atmosphere.anomaly.file`

Name of the file containing climate forcing fields.

Type string

Default value *empty*

Option `-atmosphere_anomaly_file`

4. `atmosphere.anomaly.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

5. `atmosphere.delta_P.file`

Name of the file containing scalar precipitation offsets.

Type string

Default value *empty*

Option `-atmosphere_delta_P_file`

6. `atmosphere.delta_P.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

7. `atmosphere.delta_T.file`

Name of the file containing temperature offsets.

Type string

Default value *empty*

Option `-atmosphere_delta_T_file`

8. `atmosphere.delta_T.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

9. `atmosphere.elevation_change.file`

Name of the file containing the reference surface elevation field (variable usurf).

Type string

Default value *empty*

Option `-atmosphere_lapse_rate_file`

10. `atmosphere.elevation_change.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

11. `atmosphere.elevation_change.precipitation.lapse_rate`

Elevation lapse rate for the precipitation

Type number

Default value 0 ($kg\ m^{-2}/year)/km$)

Option -precip_lapse_rate

12. `atmosphere.elevation_change.precipitation.method`

Choose the precipitation adjustment method. `scale`: use temperature-change-dependent scaling factor. `shift`: use the precipitation lapse rate.

Type keyword

Default value shift

Choices scale, shift

Option -precip_adjustment

13. `atmosphere.elevation_change.precipitation.temp_lapse_rate`

Elevation lapse rate for the surface temperature used to compute ΔT in the precipitation scaling factor $\exp(C \cdot \Delta T)$

Type number

Default value 0 (Kelvin/km)

Option -precip_temp_lapse_rate

14. `atmosphere.elevation_change.temperature_lapse_rate`

Elevation lapse rate for the surface temperature

Type number

Default value 0 (Kelvin/km)

Option -temp_lapse_rate

15. `atmosphere.fausto_air_temp.c_ma`

latitude-dependence coefficient for formula (1) in [1]

Type number

Default value -0.7189 (Kelvin/degree_north)

16. `atmosphere.fausto_air_temp.c_mj`

latitude-dependence coefficient for formula (2) in [1]

Type number

Default value -0.1585 (Kelvin/degree_north)

17. `atmosphere.fausto_air_temp.d_ma`

41.83+273.15; base temperature for formula (1) in [1]

Type number

Default value 314.98 (Kelvin)

18. `atmosphere.fausto_air_temp.d_mj`

= 14.70+273.15; base temperature for formula (2) in [1]

Type number

Default value 287.85 (Kelvin)

19. `atmosphere.fausto_air_temp.gamma_ma`
= $-6.309 / 1\text{km}$; mean slope lapse rate for formula (1) in [1]

Type number

Default value -0.006309 (*Kelvin / meter*)

20. `atmosphere.fausto_air_temp.gamma_mj`
= $-5.426 / 1\text{km}$; mean slope lapse rate for formula (2) in [1]

Type number

Default value -0.005426 (*Kelvin / meter*)

21. `atmosphere.fausto_air_temp.kappa_ma`
longitude-dependence coefficient for formula (1) in [1]

Type number

Default value 0.0672 (*Kelvin / degree_west*)

22. `atmosphere.fausto_air_temp.kappa_mj`
longitude-dependence coefficient for formula (2) in [1]

Type number

Default value 0.0518 (*Kelvin / degree_west*)

23. `atmosphere.fausto_air_temp.summer_peak_day`
day of year for July 15; used in corrected formula (4) in [1]

Type integer

Default value 196 (*ordinal day number*)

24. `atmosphere.frac_P.file`

Name of the file containing scalar precipitation scaling.

Type string

Default value *empty*

Option `-atmosphere_fra_P_file`

25. `atmosphere.fra_P.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

26. `atmosphere.given.file`

Name of the file containing climate forcing fields.

Type string

Default value *empty*

Option `-atmosphere_given_file`

27. `atmosphere.given.periodic`

If true, interpret forcing data as periodic in time

- Type** flag
Default value no
28. `atmosphere.models`
Comma-separated list of atmosphere melt models and modifiers.
Type string
Default value given
Option `-atmosphere`
29. `atmosphere.one_station.file`
Specifies a file containing scalar time-series `precipitation` and `air_temp`.
Type string
Default value *empty*
Option `-atmosphere_one_station_file`
30. `atmosphere.orographic_precipitation.background_precip_post`
Background precipitation P_{post} added after the truncation.
Type number
Default value 0 (*mm/hr*)
Option `-background_precip_post`
31. `atmosphere.orographic_precipitation.background_precip_pre`
Background precipitation P_{pre} added before the truncation.
Type number
Default value 0 (*mm/hr*)
Option `-background_precip_pre`
32. `atmosphere.orographic_precipitation.conversion_time`
Cloud conversion time τ_c
Type number
Default value 1000 (*s*)
Option `-conversion_time`
33. `atmosphere.orographic_precipitation.coriolis_latitude`
Latitude used to compute Coriolis force
Type number
Default value 0 (*degrees_N*)
Option `-coriolis_latitude`
34. `atmosphere.orographic_precipitation.fallout_time`
Fallout time τ_f
Type number
Default value 1000 (*s*)

Option `-fallout_time`

35. `atmosphere.orographic_precipitation.grid_size_factor`

The size of the extended grid is $(Z^*(\text{grid.Mx} - 1) + 1, Z^*(\text{grid.My} - 1) + 1)$ where Z is given by this parameter.

Type integer

Default value 2 (*count*)

36. `atmosphere.orographic_precipitation.lapse_rate`

Lapse rate γ

Type number

Default value -5.8 (*K/km*)

Option `-lapse_rate`

37. `atmosphere.orographic_precipitation.moist_adiabatic_lapse_rate`

Moist adiabatic lapse rate Γ_m

Type number

Default value -6.5 (*K/km*)

Option `-moist_adiabatic_lapse_rate`

38. `atmosphere.orographic_precipitation.moist_stability_frequency`

Moist stability frequency N_m

Type number

Default value 0.05 (*1/s*)

Option `-moist_stability_frequency`

39. `atmosphere.orographic_precipitation.reference_density`

Reference density $\rho_{S_{\text{ref}}}$

Type number

Default value 0.0074 (*kg m-3*)

Option `-reference_density`

40. `atmosphere.orographic_precipitation.scale_factor`

Precipitation scaling factor S .

Type number

Default value 1 (*1*)

Option `-scale_factor`

41. `atmosphere.orographic_precipitation.smoothing_standard_deviation`

Standard deviation of the Gaussian filter used to smooth surface elevation or zero if disabled

Type number

Default value 0 (*m*)

42. atmosphere.orographic_precipitation.truncate

Truncate precipitation at 0, disallowing negative precipitation values.

Type flag

Default value true

Option -truncate

43. atmosphere.orographic_precipitation.water_vapor_scale_height

Water vapor scale height H_w

Type number

Default value 2500 (*m*)

Option -water_vapor_scale_height

44. atmosphere.orographic_precipitation.wind_direction

The direction the wind is coming from

Type number

Default value 270 (*degrees*)

Option -wind_direction

45. atmosphere.orographic_precipitation.wind_speed

The wind speed.

Type number

Default value 10 (*m/s*)

Option -wind_speed

46. atmosphere.pik.file

Name of the file containing the reference surface elevation field (variable usurf).

Type string

Default value empty

Option -atmosphere_pik_file

47. atmosphere.pik.parameterization

Selects parameterizations of mean annual and mean summer near-surface air temperatures.

Type keyword

Default value martin

Choices martin, huybrechts_dewolde, martin_huybrechts_dewolde, era_interim, era_interim_sin, era_interim_lon

Option -atmosphere_pik

48. atmosphere.precip_exponential_factor_for_temperature

= 0.169/2.4; in SeaRISE-Greenland formula for precipitation correction using air temperature offsets relative to present; a 7.3% change of precipitation rate for every one degC of temperature change [2]

Type number

Default value 0.0704167 (*Kelvin-1*)

49. `atmosphere.precip_scaling.file`

Name of the file containing temperature offsets to use for a precipitation correction.

Type string

Default value *empty*

Option `-atmosphere_precip_scaling_file`

50. `atmosphere.precip_scaling.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

51. `atmosphere.searise_greenland.file`

Name of the file containing a precipitation field.

Type string

Default value *empty*

Option `-atmosphere_searise_greenland_file`

52. `atmosphere.uniform.precipitation`

Precipitation used by the `uniform` atmosphere model.

Type number

Default value 1000 (*kg m⁻² year⁻¹*)

53. `atmosphere.uniform.temperature`

Air temperature used by the `uniform` atmosphere model.

Type number

Default value 273.15 (*Kelvin*)

54. `atmosphere.yearly_cycle.file`

Name of the file containing mean annual and mean July temperatures (`air_temp_mean_annual` and `air_temp_mean_summer`) and the precipitation field.

Type string

Default value *empty*

Option `-atmosphere_yearly_cycle_file`

55. `atmosphere.yearly_cycle.scaling.file`

Name of the file containing amplitude scaling (`amplitude_scaling`) for the near-surface air temperature.

Type string

Default value *empty*

Option `-atmosphere_yearly_cycle_scaling_file`

56. `atmosphere.yearly_cycle.scaling.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

57. `basal_resistance.betta_ice_free_bedrock`

value is for ice stream E from [117]; 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

Type number

Default value 1.8e+09 (*Pascal second meter-1*)

58. `basal_resistance.betta_lateral_margin`

high value of β used to simulate drag at lateral ice margins (fjord walls, etc); the default value is chosen to disable flow in the direction along a margin

Type number

Default value 1e+19 (*Pascal second meter-1*)

59. `basal_resistance.plastic.regularization`

Set the value of ϵ regularization of plastic till; this is the second ϵ in formula (4.1) in [45]

Type number

Default value 0.01 (*meter / year*)

Option -plastic_reg

60. `basal_resistance.pseudo_plastic.enabled`

Use the pseudo-plastic till model (basal sliding law).

Type flag

Default value no

Option -pseudo_plastic

61. `basal_resistance.pseudo_plastic.q`

The exponent of the pseudo-plastic basal resistance model

Type number

Default value 0.25 (*pure number*)

Option -pseudo_plastic_q

62. `basal_resistance.pseudo_plastic.sliding_scale_factor`

divides pseudo-plastic tauc (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

Type number

Default value -1 (1)

Option -sliding_scale_factor_reduces_tauc

63. `basal_resistance.pseudo_plastic.u_threshold`

threshold velocity of the pseudo-plastic sliding law

Type number

Default value 100 (*meter / year*)

Option -pseudo_plastic_uthreshold

64. `basal_resistance.regularized_coulomb.enabled`

Use the regularized Coulomb till friction model (basal sliding law) as in equation (3) in [106]

Type flag

Default value no

Option `-regularized_coulomb`

65. `basal_yield_stress.add_transportable_water`

If “yes” then the water amount in the transport system is added to tillwat in determining tauc (in the Mohr-Coulomb relation). Normally only the water in the till is used.

Type flag

Default value no

Option `-tauc_add_transportable_water`

66. `basal_yield_stress.constant.value`

fill value for yield stress for basal till (plastic or pseudo-plastic model); note 2×10^5 Pa = 2 bar is quite strong and little sliding should occur

Type number

Default value 200000 (*Pascal*)

Option `-tauc`

67. `basal_yield_stress.ice_free_bedrock`

the “high” yield stress value used in grounded ice-free areas.

Type number

Default value 1e+06 (*Pascal*)

Option `-high_tauc`

68. `basal_yield_stress.model`

The basal yield stress model to use when sliding is active.

Type keyword

Default value `mohr_coulomb`

Choices `constant`, `mohr_coulomb`, `tillphi_opt`

Option `-yield_stress`

69. `basal_yield_stress.mohr_coulomb.delta.file`

Name of the file containing space- and time-dependent variable `mohr_coulomb_delta` to use instead of `basal_yield_stress.mohr_coulomb.till_effective_fraction_overburden`.

Type string

Default value `empty`

Option `-mohr_coulomb_delta_file`

70. `basal_yield_stress.mohr_coulomb.delta.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

71. `basal_yield_stress.mohr_coulomb.tauc_to_phi.file`

File containing the basal yield stress field that should be used to recover the till friction angle distribution.

Type string

Default value empty

Option -tauc_to_phi

72. `basal_yield_stress.mohr_coulomb.till_cohesion`

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...”

Type number

Default value 0 (*Pascal*)

Option -till_cohesion

73. `basal_yield_stress.mohr_coulomb.till_compressibility_coefficient`

coefficient of compressiblity of till; value from [109]

Type number

Default value 0.12 (*pure number*)

Option -till_compressibility_coefficient

74. `basal_yield_stress.mohr_coulomb.till_effective_fraction_overburden`

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 [109]; default value from Greenland and Antarctic model runs

Type number

Default value 0.02 (*pure number*)

Option -till_effective_fraction_overburden

75. `basal_yield_stress.mohr_coulomb.till_log_factor_transportable_water`

If `basal_yield_stress.add_transportable_water` is set then the water amount in the transport system is added to `tillwat` in determining `tauc`. Normally only the water in the till is used. This factor multiplies the logarithm in that formula.

Type number

Default value 0.1 (*meters*)

Option -till_log_factor_transportable_water

76. `basal_yield_stress.mohr_coulomb.till_phi_default`

fill value for till friction angle

Type number

Default value 30 (*degrees*)

Option -plastic_phi

77. `basal_yield_stress.mohr_coulomb.till_reference_effective_pressure`
reference effective pressure N_0 ; value from [109]

Type number

Default value 1000 (*Pascal*)

78. `basal_yield_stress.mohr_coulomb.till_reference_void_ratio`
void ratio at reference effective pressure N_0 ; value from [109]

Type number

Default value 0.69 (*pure number*)

Option `-till_reference_void_ratio`

79. `basal_yield_stress.mohr_coulomb.tillphi_opt.dhdt_min`

rate of change in the surface elevation mismatch D used as a convergence criterion

Type number

Default value 0.001 (*meters year-1*)

Option `-tillphi_opt_dhdt_min`

80. `basal_yield_stress.mohr_coulomb.tillphi_opt.dphi_max`

maximum till friction angle adjustment $\Delta\phi_{\max}$

Type number

Default value 1 (*degrees*)

Option `-tillphi_opt_dphi_max`

81. `basal_yield_stress.mohr_coulomb.tillphi_opt.dphi_scale`

scaling factor C used to compute the $\Delta\phi$ adjustment, C degrees per meter of surface elevation mismatch

Type number

Default value 0.002 (*degree / meters*)

Option `-tillphi_opt_dphi_scale`

82. `basal_yield_stress.mohr_coulomb.tillphi_opt.dt`

time step Δt_ϕ for optimization of till friction angle

Type number

Default value 250 (*365days*)

Option `-tillphi_opt_dt`

83. `basal_yield_stress.mohr_coulomb.tillphi_opt.file`

Name of the file containing the time-independent variable `usurf` used as target surface elevation

Type string

Default value *empty*

Option `-tillphi_opt_file`

84. `basal_yield_stress.mohr_coulomb.tillphi_opt.phi0_max`

maximum value of the lower bound of the till friction angle, $\phi_{0,\max}$

- Type** number
Default value 5 (*degrees*)
Option `-tillphi_opt_phi0_max`
85. `basal_yield_stress.mohr_coulomb.tillphi_opt.phi0_min`
minimum value of the lower bound of the till friction angle, $\phi_{0,\min}$
Type number
Default value 2 (*degrees*)
Option `-tillphi_opt_phi0_min`
86. `basal_yield_stress.mohr_coulomb.tillphi_opt.phi_max`
upper bound of the till friction angle ϕ_{\max}
Type number
Default value 70 (*degrees*)
Option `-tillphi_opt_phi_max`
87. `basal_yield_stress.mohr_coulomb.tillphi_opt.topg_max`
the bed elevation b_{\max} above which `basal_yield_stress.mohr_coulomb.tillphi_opt.phi0_max` is used
Type number
Default value 700 (*meters*)
Option `-tillphi_opt_topg_max`
88. `basal_yield_stress.mohr_coulomb.tillphi_opt.topg_min`
the bed elevation b_{\min} below which `basal_yield_stress.mohr_coulomb.tillphi_opt.phi0_min` is used
Type number
Default value -300 (*meters*)
Option `-tillphi_opt_topg_min`
89. `basal_yield_stress.mohr_coulomb.topg_to_phi.enabled`
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.
Type flag
Default value no
90. `basal_yield_stress.mohr_coulomb.topg_to_phi.phi_max`
upper value of the till friction angle; see the implementation of MohrCoulombYieldStress
Type number
Default value 15 (*degrees*)
91. `basal_yield_stress.mohr_coulomb.topg_to_phi.phi_min`
lower value of the till friction angle; see the implementation of MohrCoulombYieldStress
Type number
Default value 5 (*degrees*)

92. `basal_yield_stress.mohr_coulomb.topg_to_phi.topg_max`

the elevation at which the upper value of the till friction angle is used; see the implementation of MohrCoulombYieldStress

Type number

Default value 1000 (*meters*)

93. `basal_yield_stress.mohr_coulomb.topg_to_phi.topg_min`

the elevation at which the lower value of the till friction angle is used; see the implementation of MohrCoulombYieldStress

Type number

Default value -1000 (*meters*)

94. `basal_yield_stress.slippery_grounding_lines`

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.

Type flag

Default value no

Option `-tauc_slippery_grounding_lines`

95. `bed_deformation.bed_topography_delta_file`

The name of the file to read the `topg_delta` from. This field is added to the bed topography during initialization.

Type string

Default value *empty*

Option `-topg_delta_file`

96. `bed_deformation.bed_uplift_file`

The name of the file to read the uplift (dbdt) from. Leave empty to read it from an input file or a regridding file.

Type string

Default value *empty*

Option `-uplift_file`

97. `bed_deformation.given.file`

Name of the file containing time-dependent `topg_delta`.

Type string

Default value *empty*

Option `-topg_delta_file`

98. `bed_deformation.given.reference_file`

Name of the file containing the reference bed topography `topg`.

Type string

- Default value** *empty*
- Option** `-topg_reference_file`
99. `bed_deformation.lc.elastic_model`
- Use the elastic part of the Lingle-Clark bed deformation model.
- Type** flag
- Default value** yes
- Option** `-bed_def_lc_elastic_model`
100. `bed_deformation.lc.grid_size_factor`
- 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 [112], [27].
- Type** integer
- Default value** 4 (*count*)
101. `bed_deformation.lc.update_interval`
- Interval between updates of the Lingle-Clark model
- Type** number
- Default value** 10 (*365days*)
102. `bed_deformation.lithosphere_flexural_rigidity`
- lithosphere flexural rigidity used by the bed deformation model. See [112], [27]
- Type** number
- Default value** 5e+24 (*Newton meter*)
103. `bed_deformation.mantle_density`
- half-space (mantle) density used by the bed deformation model. See [112], [27]
- Type** number
- Default value** 3300 (*kg meter-3*)
104. `bed_deformation.mantle_viscosity`
- half-space (mantle) viscosity used by the bed deformation model. See [112], [27]
- Type** number
- Default value** 1e+21 (*Pascal second*)
105. `bed_deformation.model`
- Selects a bed deformation model to use. `iso` is point-wise isostasy, `lc` is the Lingle-Clark model (see [112], requires `FFTW`), `given` uses prescribed bed topography changes read from a file.
- Type** keyword
- Default value** none
- Choices** `none`, `iso`, `lc`, `given`
- Option** `-bed_def`
106. `bootstrapping.defaults.bed`
- bed elevation value to use if `topg` (`bedrock_altitude`) variable is absent in bootstrapping file

Type number

Default value 1 (*meters*)

107. bootstrapping.defaults.bmelt

basal melt rate value to use if variable `bmelt` is absent in bootstrapping file

Type number

Default value 0 (*meter / second*)

108. bootstrapping.defaults.bwat

till water thickness value to use if variable `tillwat` is absent in bootstrapping file

Type number

Default value 0 (*meters*)

109. bootstrapping.defaults.bwp

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

Type number

Default value 0 (*Pascal*)

110. bootstrapping.defaults.enwat

effective englacial water thickness value to use if variable `enwat` is absent in bootstrapping file

Type number

Default value 0 (*meters*)

111. bootstrapping.defaults.geothermal_flux

geothermal flux value to use if `bheatflx` variable is absent in bootstrapping file

Type number

Default value 0.042 (*Watt meter-2*)

112. bootstrapping.defaults.ice_thickness

thickness value to use if `thk` (`land_ice_thickness`) variable is absent in bootstrapping file

Type number

Default value 0 (*meters*)

113. bootstrapping.defaults.tillwat

till water thickness value to use if variable `tillwat` is absent in bootstrapping file

Type number

Default value 0 (*meters*)

114. bootstrapping.defaults.uplift

uplift value to use if `dbdt` variable is absent in bootstrapping file

Type number

Default value 0 (*meter / second*)

115. bootstrapping.temperature_heuristic

The heuristic to use to initialize ice temperature during bootstrapping: `smb` uses the surface mass balance, surface temperature, and the geothermal flux, `quartic_guess` uses the surface temperature and the geothermal flux.

Type keyword

Default value `smb`

Choices `smb`, `quartic_guess`

Option `-boot_temperature_heuristic`

116. calving.eigen_calving.K

Set proportionality constant to determine calving rate from strain rates. Note references [89], [3] use K in range 10^9 to 3×10^{11} m a, that is, 3×10^{16} to 10^{19} m s.

Type number

Default value 0 (*meter second*)

Option `-eigen_calving_K`

117. calving.float_kill.calve_near_grounding_line

Calve floating ice near the grounding line.

Type flag

Default value yes

Option `-float_kill_calve_near_grounding_line`

118. calving.float_kill.margin_only

Apply float_kill at ice margin cells only.

Type flag

Default value no

Option `-float_kill_margin_only`

119. calving.hayhurst_calving.B_tilde

Effective damage rate [91]

Type number

Default value 65 ((MPa)^r/year)

120. calving.hayhurst_calving.exponent_r

Damage law exponent [91]

Type number

Default value 0.43 (I)

121. calving.hayhurst_calving.sigma_threshold

Damage threshold stress [91]

Type number

Default value 0.17 (MPa)

122. `calving.methods`

comma-separated list of calving methods; one or more of `eigen_calving`, `float_kill`, `thickness_calving`

Type string

Default value *empty*

Option `-calving`

123. `calving.rate_scaling.file`

File containing the scaling factor applied to calving rates from `eigen_calving`, `vonmises_calving`, and `hayhurst_calving` (variable name: `frac_calving_rate`)

Type string

Default value *empty*

124. `calving.rate_scaling.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

125. `calving.thickness_calving.file`

Name of the file containing the spatially-variable thickness calving threshold.

Type string

Default value *empty*

Option `-thickness_calving_file`

126. `calving.thickness_calving.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

127. `calving.thickness_calving.threshold`

When terminal ice thickness of floating ice shelf is less than this threshold, it will be calved off.

Type number

Default value 50 (*meters*)

Option `-thickness_calving_threshold`

128. `calving.vonmises_calving.Glen_exponent`

Glen exponent in ice flow law for von Mises calving

Type number

Default value 3 (*pure number*)

Option `-vonmises_calving_n`

129. `calving.vonmises_calving.flow_law`

The custom flow law for the von Mises stress computation

Type keyword

Default value gpbld
Choices arr, arrwarm, gpbld, hooke, isothermal_glen, pb
130. calving.vonmises_calving.sigma_max

Set maximum tensile stress. Note references [90] use 1.0e6 Pa.

Type number
Default value 1e+06 (*Pa*)
Option -vonmises_calving_calving_sigma_max

131. calving.vonmises_calving.threshold_file

Name of the file containing the spatially-variable vonmises_calving_threshold.

Type string
Default value *empty*
Option -vonmises_calving_threshold_file

132. calving.vonmises_calving.use_custom_flow_law

Use custom flow law in the von Mises stress computation

Type flag
Default value no
Option -vonmises_calving_use_custom_flow_law

133. constants.fresh_water.density

density of fresh water

Type number
Default value 1000 (*kg meter-3*)

134. constants.fresh_water.latent_heat_of_fusion

latent heat of fusion for water [75]

Type number
Default value 334000 (*Joule/kg*)

135. constants.fresh_water.melting_point_temperature

melting point of pure water

Type number
Default value 273.15 (*Kelvin*)

136. constants.fresh_water.specific_heat_capacity

at melting point T_0 [75]

Type number
Default value 4170 (*Joule/(kg Kelvin)*)

137. constants.global_ocean_area

area of the global ocean [118]

Type number

- Default value** 3.625e+14 (*meter*²)
138. `constants.ice.beta_Clausius_Clapeyron`
- Clausius-Clapeyron constant relating melting temperature and pressure: $\beta = dT/dP$ [24]
- Type** number
- Default value** 7.9e-08 (*Kelvin / Pascal*)
139. `constants.ice.density`
- ρ_i ; density of ice in ice sheet
- Type** number
- Default value** 910 (*kg meter*⁻³)
140. `constants.ice.grain_size`
- Default constant ice grain size to use with the Goldsby-Kohlstedt [25] flow law
- Type** number
- Default value** 1 (*mm*)
- Option** `-ice_grain_size`
141. `constants.ice.specific_heat_capacity`
- specific heat capacity of pure ice at melting point T_0
- Type** number
- Default value** 2009 (*Joule / (kg Kelvin)*)
142. `constants.ice.thermal_conductivity`
- = W m⁻¹ K⁻¹; thermal conductivity of pure ice
- Type** number
- Default value** 2.1 (*Joule / (meter Kelvin second)*)
143. `constants.ideal_gas_constant`
- ideal gas constant
- Type** number
- Default value** 8.31441 (*Joule / (mol Kelvin)*)
144. `constants.sea_water.density`
- density of sea water
- Type** number
- Default value** 1028 (*kg meter*⁻³)
145. `constants.sea_water.salinity`
- salinity of sea water
- Type** number
- Default value** 35 (*g/kg*)
146. `constants.sea_water.specific_heat_capacity`
- at 35 psu, value taken from [Kaye and Laby](#), section 2.7.9

- Type** number
Default value 3985 (*Joule/(kg Kelvin)*)
147. `constants.standard_gravity`
acceleration due to gravity on Earth geoid
Type number
Default value 9.81 (*meter second-2*)
148. `energy.allow_temperature_above_melting`
If set to “yes”, allow temperatures above the pressure-melting point in the cold mode temperature code. Used by some verification tests.
Type flag
Default value no
149. `energy.basal_melt.use_grounded_cell_fraction`
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.
Type flag
Default value true
Option `-subgl_basal_melt`
150. `energy.bedrock_thermal.conductivity`
= W m-1 K-1; for bedrock used in thermal model [7]
Type number
Default value 3 (*Joule/(meter Kelvin second)*)
151. `energy.bedrock_thermal.density`
for bedrock used in thermal model
Type number
Default value 3300 (*kg meter-3*)
152. `energy.bedrock_thermal.file`
Name of the file containing the geothermal flux field `bheatflx`. Leave empty to read it from the `input.file`.
Type string
Default value *empty*
153. `energy.bedrock_thermal.specific_heat_capacity`
for bedrock used in thermal model [7]
Type number
Default value 1000 (*Joule/(kg Kelvin)*)
154. `energy.ch_warming.average_channel_spacing`
Average spacing between elements of the cryo-hydrologic system (controls the rate of heat transfer from the CH system into the ice).
Type number

Default value 20 (*meters*)

155. `energy.ch_warming.enabled`

Enable the cryo-hydrologic warming model

Type flag

Default value no

156. `energy.ch_warming.residual_water_fraction`

residual water fraction in the cryo-hydrologic system at the end of a melt season

Type number

Default value 0.005 (*pure number*)

157. `energy.ch_warming.temperate_ice_thermal_conductivity_ratio`

ratio of thermal conductivities of temperate and cold ice in the cryo-hydrologic system

Type number

Default value 1 (*pure number*)

158. `energy.drainage_maximum_rate`

0.05 year-1; maximum rate at which liquid water fraction in temperate ice could possibly drain; see [35]

Type number

Default value 1.58444e-09 (*second-1*)

159. `energy.drainage_target_water_fraction`

liquid water fraction (omega) above which drainage occurs, but below which there is no drainage; see [35]

Type number

Default value 0.01 (*I*)

160. `energy.enabled`

Solve energy conservation equations.

Type flag

Default value yes

161. `energy.enthalpy.cold_bulge_max`

= (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

Type number

Default value 60270 (*Joule / kg*)

162. `energy.enthalpy.temperate_ice_thermal_conductivity_ratio`

K in cold ice is multiplied by this fraction to give K0 in [35]

Type number

Default value 0.1 (*pure number*)

163. `energy.margin_exclude_horizontal_advection`

Exclude horizontal advection of energy at grid points near ice margins. See `energy.margin_ice_thickness_limit`.

- Type** flag
Default value yes
164. `energy.margin_exclude_strain_heating`
Exclude strain heating at grid points near ice margins. See `energy.margin_ice_thickness_limit`.
Type flag
Default value yes
165. `energy.margin_exclude_vertical_advection`
Exclude vertical advection of energy at grid points near ice margins. See `energy.margin_ice_thickness_limit`.
Type flag
Default value yes
166. `energy.margin_ice_thickness_limit`
use special margin treatment at grid points with a neighbor with the thickness below this limit.
Type number
Default value 100 (*meters*)
167. `energy.max_low_temperature_count`
Maximum number of grid points with ice temperature below `energy.minimum_allowed_temperature`.
Type integer
Default value 10 (*count*)
Option `-max_low_temps`
168. `energy.minimum_allowed_temperature`
Minimum allowed ice temperature
Type number
Default value 200 (*Kelvin*)
Option `-low_temp`
169. `energy.temperature_based`
Use cold ice (i.e. not polythermal) methods.
Type flag
Default value no
170. `energy.temperature_dependent_thermal_conductivity`
If yes, use varkenthSystemCtx class in the energy step. It is base on formula (4.37) in [56]. Otherwise use enthSystemCtx, which has temperature-independent thermal conductivity set by constant `ice.thermal_conductivity`.
Type flag
Default value no
Option `-vark`
171. `enthalpy_converter.T_reference`
= `T_0` in enthalpy formulas in [35]

Type number

Default value 223.15 (*Kelvin*)

172. `enthalpy_converter.relaxed_is_temperate_tolerance`

Tolerance within which ice is treated as temperate (cold-ice mode and diagnostics).

Type number

Default value 0.001 (*Kelvin*)

173. `flow_law.Hooke.A`

$A_{\text{Hooke}} = (1/B_0)^n$ where $n=3$ and $B_0 = 1.928 \text{ a}^{1/3} \text{ Pa}$. See [79]

Type number

Default value 4.42165e-09 (*Pascal-3 second-1*)

174. `flow_law.Hooke.C`

See [79]

Type number

Default value 0.16612 (*Kelvin^k*)

175. `flow_law.Hooke.Q`

Activation energy, see [79]

Type number

Default value 78800 (*Joule / mol*)

176. `flow_law.Hooke.Tr`

See [79]

Type number

Default value 273.39 (*Kelvin*)

177. `flow_law.Hooke.k`

See [79]

Type number

Default value 1.17 (*pure number*)

178. `flow_law.Paterson_Budd.A_cold`

Paterson-Budd A_{cold} , see [76]

Type number

Default value 3.61e-13 (*Pascal-3 / second*)

179. `flow_law.Paterson_Budd.A_warm`

Paterson-Budd A_{warm} , see [76]

Type number

Default value 1730 (*Pascal-3 / second*)

180. `flow_law.Paterson_Budd.Q_cold`

Paterson-Budd Q_{cold} , see [76]

- Type** number
Default value 60000 (*Joule/mol*)
181. `flow_law.Paterson_Budd.Q_warm`
 Paterson-Budd Q_{warm} , see [76]
Type number
Default value 139000 (*Joule/mol*)
182. `flow_law.Paterson_Budd.T_critical`
 Paterson-Budd critical temperature, see [76]
Type number
Default value 263.15 (*Kelvin*)
183. `flow_law.Schoof_regularizing_length`
 Regularizing length (Schoof definition)
Type number
Default value 1000 (*km*)
184. `flow_law.Schoof_regularizing_velocity`
 Regularizing velocity (Schoof definition)
Type number
Default value 1 (*meter/year*)
185. `flow_law.gpbld.water_frac_coeff`
 coefficient in Glen-Paterson-Budd flow law for extra dependence of softness on liquid water fraction (omega) [56], [77]
Type number
Default value 181.25 (*pure number*)
186. `flow_law.gpbld.water_frac_observed_limit`
 maximum value of liquid water fraction omega for which softness values are parameterized by [77]; used in Glen-Paterson-Budd-Lliboutry-Duval flow law; compare [35]
Type number
Default value 0.01 (*I*)
187. `flow_law.isothermal_Glen.ice_softness`
 ice softness used by the isothermal Glen flow law [34]
Type number
Default value 3.1689e-24 (*Pascal-3 second-I*)
188. `fracture_density.borstad_limit`
 Model fracture growth according to the constitutive law in [96] (Eq. 4), ignoring `fracture_density.gamma`.
Type flag
Default value no
Option `-constitutive_stress_limit`

189. `fracture_density.constant_fd`

Keep fracture density fields constant in time but include its softening effect.

Type flag

Default value no

Option `-constant_fd`

190. `fracture_density.constant_healing`

Use a constant healing rate $-\gamma_h \dot{\epsilon}_h$ independent of the local strain rate.

Type flag

Default value no

Option `-constant_healing`

191. `fracture_density.enabled`

Model the fracture density using computed to stresses and strain rates.

Type flag

Default value no

Option `-fractures`

192. `fracture_density.fd2d_scheme`

Use an alternative transport scheme to reduce numerical diffusion (Eq. 10 in [95])

Type flag

Default value no

Option `-scheme_fd2d`

193. `fracture_density.fracture_weighted_healing`

Multiply the healing rate by $1 - D$, i.e. assume that highly damaged ice heals slower. This mechanism can be combined with `fracture_density.constant_healing`.

Type flag

Default value no

Option `-fracture_weighted_healing`

194. `fracture_density.gamma`

fracture growth constant γ

Type number

Default value 1 (1)

Option `-fracture_gamma`

195. `fracture_density.gamma_h`

fracture healing constant γ_h

Type number

Default value 0 (1)

Option `-fracture_gamma_h`

196. `fracture_density.healing_threshold`

fracture healing strain rate threshold $\dot{\epsilon}_h$

Type number

Default value 2e-10 ($1/s$)

Option `-fracture_healing_threshold`

197. `fracture_density.include_grounded_ice`

Model fracture density in grounded areas (e.g. along ice stream shear zones) in addition to ice shelves

Type flag

Default value no

Option `-do_frac_on_grounded`

198. `fracture_density.initiation_threshold`

fracture initiation stress threshold σ_{cr}

Type number

Default value 70000 (Pa)

Option `-fracture_initiation_threshold`

199. `fracture_density.lefm`

Use the mixed-mode fracture toughness stress criterion based on Linear Elastic Fracture Mechanics, Eqs. 8-9 in [95]

Type flag

Default value no

Option `-lefm`

200. `fracture_density.max_shear_stress`

Use the maximum shear stress criterion for fracture formation (Tresca or Guest criterion in literature), which is more stringent than the default von Mises criterion, see Eq. 7 in [95]

Type flag

Default value no

Option `-max_shear`

201. `fracture_density.phi0`

Fracture density value used at grid points where ice velocity is prescribed. This assumes that all ice entering a shelf at `bc_mask` locations has the same fracture density.

Type number

Default value 0 (1)

Option `-phi0`

202. `fracture_density.softening_lower_limit`

Parameter controlling the strength of the feedback of damage on the ice flow. If 1: no feedback, if 0: full feedback (ϵ in Eq. 6 in [95])

Type number

Default value 1 (*I*)

Option -fracture_softening

203. `frontal_melt.constant.melt_rate`

default melt rate used by the `constant` frontal_melt model

Type number

Default value 1 (*m/day*)

Option -frontal_melt_rate

204. `frontal_melt.discharge_given.file`

Name of the file containing climate forcing fields.

Type string

Default value *empty*

Option -frontal_melt_discharge_given_file

205. `frontal_melt.discharge_given.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

206. `frontal_melt.given.file`

Name of the file containing climate forcing fields.

Type string

Default value *empty*

Option -frontal_melt_given_file

207. `frontal_melt.given.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

208. `frontal_melt.include_floating_ice`

Apply frontal melt to all grid icy cells next to ocean cells

Type flag

Default value no

209. `frontal_melt.models`

Comma-separated list of frontal melt models and modifiers. (Leave empty to disable.)

Type string

Default value *empty*

Option -frontal_melt

210. `frontal_melt.routing.file`

Name of the file containing climate forcing fields.

- Type** string
Default value *empty*
Option `-frontal_melt_routing_file`
211. `frontal_melt.routing.parameter_a`
parameter A in eqn. 1 in [119]
Type number
Default value $0.0003 \text{ (m-alpha day}^{\alpha-1} \text{ Celsius-beta)}$
212. `frontal_melt.routing.parameter_b`
parameter B in eqn. 1 in [119]
Type number
Default value $0.15 \text{ (m day}^{\alpha-1} \text{ Celsius-beta)}$
213. `frontal_melt.routing.periodic`
If true, interpret forcing data as periodic in time
Type flag
Default value no
214. `frontal_melt.routing.power_alpha`
exponent α in eqn. 1 in [119]
Type number
Default value 0.39 (1)
215. `frontal_melt.routing.power_beta`
exponent β in eqn. 1 in [119]
Type number
Default value 1.18 (1)
216. `geometry.front_retreat.prescribed.file`
Name of the file containing the maximum ice extent mask `land_ice_area_fraction_retreat`
Type string
Default value *empty*
Option `-front_retreat_file`
217. `geometry.front_retreat.prescribed.periodic`
If true, interpret forcing data as periodic in time
Type flag
Default value no
218. `geometry.front_retreat.use_cfl`
apply CFL criterion for eigen-calving rate front retreat
Type flag
Default value false

Option `-front_retreat_cfl`

219. `geometry.front_retreat.wrap_around`

If true, wrap around domain boundaries. This may be needed in some regional synthetic geometry setups.

Type flag

Default value false

Option `-front_retreat_wrap_around`

220. `geometry.grounded_cell_fraction`

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.

Type flag

Default value false

Option `-subgl`

221. `geometry.ice_free_thickness_standard`

If ice is thinner than this standard then the mask is set to MASK_ICE_FREE_BEDROCK or MASK_ICE_FREE_OCEAN.

Type number

Default value 0.01 (*meters*)

222. `geometry.part_grid.enabled`

apply partially filled grid cell scheme

Type flag

Default value no

Option `-part_grid`

223. `geometry.part_grid.max_iterations`

maximum number of residual redistribution iterations

Type integer

Default value 10 (*count*)

224. `geometry.remove_icebergs`

identify and kill detached ice-shelf areas

Type flag

Default value no

Option `-kill_icebergs`

225. `geometry.update.enabled`

Solve the mass conservation equation

Type flag

Default value yes

Option `-mass`

226. `geometry.update.use_basal_melt_rate`

Include basal melt rate in the continuity equation

Type flag

Default value yes

Option `-bmr_in_cont`

227. `grid.Lbz`

Thickness of the thermal bedrock layer. (Inactive if `grid.Mbz < 2`)

Type number

Default value 1000 (*meters*)

Option `-Lbz`

228. `grid.Lx`

Default computational box is 3000 km x 3000 km (= 2 Lx x 2 Ly) in horizontal.

Type number

Default value 1.5e+06 (*meters*)

229. `grid.Ly`

Default computational box is 3000 km x 3000 km (= 2 Lx x 2 Ly) in horizontal.

Type number

Default value 1.5e+06 (*meters*)

230. `grid.Lz`

Height of the computational domain.

Type number

Default value 4000 (*meters*)

Option `-Lz`

231. `grid.Mbz`

Number of thermal bedrock layers; 1 level corresponds to no bedrock.

Type integer

Default value 1 (*count*)

Option `-Mbz`

232. `grid.Mx`

Number of grid points in the x direction.

Type integer

Default value 61 (*count*)

Option `-Mx`

233. `grid.My`

Number of grid points in the y direction.

Type integer

Default value 61 (*count*)

Option -My

234. `grid.Mz`

Number of vertical grid levels in the ice.

Type integer

Default value 31 (*count*)

Option -Mz

235. `grid.allow_extrapolation`

Allow extrapolation during regridding.

Type flag

Default value no

Option -allow_extrapolation

236. `grid.ice_vertical_spacing`

vertical spacing in the ice

Type keyword

Default value quadratic

Choices quadratic, equal

Option -z_spacing

237. `grid.lambda`

Vertical grid spacing parameter. Roughly equal to the factor by which the grid is coarser at an end away from the ice-bedrock interface.

Type number

Default value 4 (*pure number*)

238. `grid.max_stencil_width`

Maximum width of the finite-difference stencil used in PISM.

Type integer

Default value 2 (*count*)

239. `grid.periodicity`

horizontal grid periodicity

Type keyword

Default value xy

Choices none, x, y, xy

Option -periodicity

240. `grid.recompute_longitude_and_latitude`

Re-compute longitude and latitude using grid information and provided projection parameters. Requires PROJ.

Type flag

- Default value** yes
241. `grid.registration`
 horizontal grid registration
Type keyword
Default value center
Choices center, corner
242. `hydrology.add_water_input_to_till_storage`
 Add surface input to water stored in till. If no it will be added to the transportable water.
Type flag
Default value yes
243. `hydrology.cavitation_opening_coefficient`
 c_1 in notes; coefficient of cavitation opening term in evolution of layer thickness in hydrology::Distributed
Type number
Default value 0.5 (*meter-1*)
Option -hydrology_cavitation_opening_coefficient
244. `hydrology.creep_closure_coefficient`
 c_2 in notes; coefficient of creep closure term in evolution of layer thickness in hydrology::Distributed
Type number
Default value 0.04 (*pure number*)
Option -hydrology_creep_closure_coefficient
245. `hydrology.distributed.init_p_from_steady`
 if “yes”, initialize subglacial water pressure from P(W) formula that applies in steady state
Type flag
Default value no
Option -hydrology_init_p_form_steady
246. `hydrology.distributed.sliding_speed_file`
 name of the file containing velbase_mag, the basal sliding speed to use with hydrology.distributed.
`init_p_from_steady`
Type string
Default value empty
Option -hydrology_sliding_speed_file
247. `hydrology.gradient_power_in_flux`
 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
Type number
Default value 1.5 (*pure number*)
Option -hydrology_gradient_power_in_flux

248. hydrology.hydraulic_conductivity

= 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

Type number

Default value $0.001 \left(m^{2\beta} - \alpha \right) s^{2\beta - 3} kg^{1-\beta}$

Option -hydrology_hydraulic_conductivity

249. hydrology.maximum_time_step

maximum allowed time step length used by hydrology::Routing and hydrology::Distributed

Type number

Default value 1 (*365days*)

250. hydrology.model

Basal hydrology sub-model.

Type keyword

Default value null

Choices null, routing, steady, distributed

Option -hydrology

251. hydrology.null_diffuse_till_water

Diffuse stored till water laterally. See equation (11) of [29]

Type flag

Default value no

252. hydrology.null_diffusion_distance

diffusion distance for till water thickness; see equation (11) in [29]; only active if hydrology.null_diffuse_till_water is set

Type number

Default value 20000 (*meters*)

253. hydrology.null_diffusion_time

diffusion time for till water thickness; see equation (11) in [29]; only active if hydrology.null_diffuse_till_water is set

Type number

Default value 1000 (*365days*)

254. hydrology.null_strip_width

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

Type number

Default value -1 (*meters*)

255. hydrology.regularizing_porosity

phi_0 in notes; regularizes pressure equation by multiplying time derivative term

Type number

Default value 0.01 (*pure number*)

Option -hydrology_regularizing_porosity

256. hydrology.roughness_scale

W_r in notes; roughness scale determining maximum amount of cavitation opening in hydrology::Distributed

Type number

Default value 0.1 (*meters*)

Option -hydrology_roughness_scale

257. hydrology.routing.include_floating_ice

Route subglacial water under ice shelves. This may be appropriate if a shelf is close to floatation. Note that this has no effect on ice flow.

Type flag

Default value no

258. hydrology.steady.flux_update_interval

interval between updates of the steady state flux

Type number

Default value 1 (*365days*)

259. hydrology.steady.input_rate_scaling

input rate scaling

Type number

Default value 1e+07 (*seconds*)

260. hydrology.steady.n_iterations

maximum number of iterations to use in while estimating steady-state water flux

Type integer

Default value 7500 (*count*)

261. hydrology.steady.potential_delta

potential adjustment used to fill sinks (smaller values require more iterations but produce fewer artifacts)

Type number

Default value 10000 (*Pa*)

262. hydrology.steady.potential_n_iterations

maximum number of iterations to take while pre-processing hydraulic potential

Type integer

Default value 1000 (*count*)

263. hydrology.steady.volume_ratio

water volume ratio used as the stopping criterion

Type number

Default value 0.1 (1)

264. `hydrology.surface_input.file`

Name of the file containing `water_input_rate`, the rate at which water from the ice surface is added to the subglacial hydrology system

Type string

Default value empty

265. `hydrology.surface_input.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

266. `hydrology.surface_input.from_runoff`

Use surface runoff as surface input.

Type flag

Default value no

267. `hydrology.thickness_power_in_flux`

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`

Type number

Default value 1.25 (1)

Option -hydrology_thickness_power_in_flux

268. `hydrology.tillwat_decay_rate`

rate at which tillwat is reduced to zero, in absence of other effects like input

Type number

Default value 1 (mm/year)

Option -hydrology_tillwat_decay_rate

269. `hydrology.tillwat_max`

maximum effective thickness of the water stored in till

Type number

Default value 2 (meters)

Option -hydrology_tillwat_max

270. `hydrology.use_const_bmelt`

if "yes", subglacial hydrology model sees basal melt rate which is constant and given by `hydrology.const_bmelt`

Type flag

Default value no

Option -hydrology_use_const_bmelt

271. `input.bootstrap`

It true, use bootstrapping heuristics when initializing PISM.

- Type** flag
Default value no
Option -bootstrap
272. **input.file**
Input file name
Type string
Default value *empty*
Option -i
273. **input.forcing.buffer_size**
number of 2D climate forcing records to keep in memory; = 5 years of monthly records
Type integer
Default value 60 (*count*)
274. **input.forcing.time_extrapolation**
If ‘true’, time-dependent forcing inputs are extrapolated in time
Type flag
Default value false
275. **input.regrid.file**
Regridding (input) file name
Type string
Default value *empty*
Option -regrid_file
276. **input.regrid.vars**
Comma-separated list of variables to regrid. Leave empty to regrid all model state variables.
Type string
Default value *empty*
Option -regrid_vars
277. **inverse.design.cH1**
weight of derivative part of an H1 norm for inversion design variables
Type number
Default value 0 (1)
Option -inv_design_cH1
278. **inverse.design.cL2**
weight of derivative-free part of an H1 norm for inversion design variables
Type number
Default value 1 (1)
Option -inv_design_cL2

279. `inverse.design.func`

functional used for inversion design variables

Type keyword

Default value sobolevH1

Choices sobolevH1, tv

Option -inv_design_func

280. `inverse.design.param`

parameterization of design variables used during inversion

Type keyword

Default value exp

Choices ident, trunc, square, exp

Option -inv_design_param

281. `inverse.design.param_hardav_eps`

tiny vertically-averaged hardness used as a substitute for 0 in some tauc parameterizations

Type number

Default value 10000 (*Pascal second^(1/3)*)

282. `inverse.design.param_hardav_scale`

typical size of ice hardness

Type number

Default value 1e+08 (*Pascal second^(1/3)*)

283. `inverse.design.param_tauc_eps`

tiny yield stress used as a substitute for 0 in some tauc parameterizations

Type number

Default value 100 (*Pascal*)

284. `inverse.design.param_tauc_scale`

typical size of yield stresses

Type number

Default value 100000 (*Pascal*)

285. `inverse.design.param_trunc_hardav0`

transition point of change to linear behaviour for design variable parameterization type trunc

Type number

Default value 1e+06 (*Pascal second^(1/3)*)

286. `inverse.design.param_trunc_tauc0`

transition point of change to linear behaviour for design variable parameterization type trunc

Type number

Default value 1000 (*Pascal*)

287. `inverse.log_ratio_scale`

Reference scale for log-ratio functionals

Type number

Default value 10 (*pure number*)

Option `-inv_log_ratio_scale`

288. `inverse.max_iterations`

maximum iteration count

Type integer

Default value 1000 (*count*)

Option `-inv_max_it`

289. `inverse.ssa.hardav_max`

Maximum allowed value of hardav for inversions with bound constraints

Type number

Default value 1e+10 (*Pascal second^(1/3)*)

290. `inverse.ssa.hardav_min`

Minimum allowed value of hardav for inversions with bound constraints

Type number

Default value 0 (*Pascal second^(1/3)*)

291. `inverse.ssa.length_scale`

typical length scale for rescaling derivative norms

Type number

Default value 50000 (*meters*)

292. `inverse.ssa.method`

algorithm to use for SSA inversions

Type keyword

Default value `tikhonov_lmvm`

Choices `sd`, `nlcg`, `ign`, `tikhonov_lmvm`, `tikhonov_cg`, `tikhonov_blmvm`,
`tikhonov_lcl`, `tikhonov_gn`

Option `-inv_method`

293. `inverse.ssa.tauc_max`

Maximum allowed value of tauc for inversions with bound constraints

Type number

Default value 5e+07 (*Pascal*)

294. `inverse.ssa.tauc_min`

Minimum allowed value of tauc for inversions with bound constraints

Type number

Default value 0 (*Pascal*)

295. `inverse.ssa.tv_exponent`

Lebesgue exponent for pseudo-TV norm

Type number

Default value 1.2 (*pure number*)

Option `-inv_ssa_tv_exponent`

296. `inverse.ssa.velocity_eps`

tiny size of ice velocities during inversion

Type number

Default value 0.1 (*meter/year*)

297. `inverse.ssa.velocity_scale`

typical size of ice velocities expected during inversion

Type number

Default value 100 (*meter/year*)

298. `inverse.state_func`

functional used for inversion design variables

Type keyword

Default value meansquare

Choices meansquare, log_ratio, log_relative

Option `-inv_state_func`

299. `inverse.target_misfit`

desired root misfit for SSA inversions

Type number

Default value 100 (*meter/year*)

Option `-inv_target_misfit`

300. `inverse.tikhonov.atol`

absolute threshold for Tikhonov stopping criterion

Type number

Default value 1e-10 (*meter/year*)

Option `-tikhonov_atol`

301. `inverse.tikhonov.penalty_weight`

penalty parameter for Tikhonov inversion

Type number

Default value 1 (*I*)

Option `-tikhonov_penalty`

302. `inverse.tikhonov.ptol`

threshold for reaching desired misfit for adaptive Tikhonov algorithms

Type number

Default value 0.1 (*pure number*)

Option `-tikhonov_ptol`

303. `inverse.tikhonov.rtol`

relative threshold for Tikhonov stopping criterion

Type number

Default value 0.05 (*I*)

Option `-tikhonov_rtol`

304. `inverse.use_design_prior`

Use prior from inverse data file as initial guess.

Type flag

Default value yes

Option `-inv_use_design_prior`

305. `inverse.use_zeta_fixed_mask`

Enforce locations where the parameterized design variable should be fixed. (Automatically determined if not provided)

Type flag

Default value yes

Option `-inv_use_zeta_fixed_mask`

306. `ocean.anomaly.file`

Name of the file containing shelf basal mass flux offset fields.

Type string

Default value *empty*

Option `-ocean_anomaly_file`

307. `ocean.anomaly.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

308. `ocean.cache.update_interval`

update interval of the cache ocean modifier

Type integer

Default value 10 (*365days*)

Option `-ocean_cache_update_interval`

309. `ocean.constant.melt_rate`

default melt rate used by the constant ocean model (computed as $Q/(L\rho_i)$)

Type number

Default value 0.0519142 (*m/year*)

Option `-shelf_base_melt_rate`

310. `ocean.delta_MBP.file`

Name of the file containing melange back-pressure offsets

Type string

Default value *empty*

311. `ocean.delta_MBP.melange_thickness`

Melange thickness (assumed to be constant in space and time)

Type number

Default value 100 (*meters*)

Option `-melange_thickness`

312. `ocean.delta_MBP.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

313. `ocean.delta_T.file`

Name of the file containing temperature offsets.

Type string

Default value *empty*

Option `-ocean_delta_T_file`

314. `ocean.delta_T.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

315. `ocean.delta_mass_flux.file`

Name of the file containing sub-shelf mass flux offsets.

Type string

Default value *empty*

Option `-ocean_delta_mass_flux_file`

316. `ocean.delta_mass_flux.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

317. ocean.delta_sl.file

Name of the file containing sea level offsets.

Type string

Default value *empty*

Option -ocean_delta_sl_file

318. ocean.delta_sl.periodic

If true, interpret forcing data as periodic in time

Type flag

Default value no

319. ocean.delta_sl_2d.file

Name of the file containing climate forcing fields.

Type string

Default value *empty*

320. ocean.delta_sl_2d.periodic

If true, interpret forcing data as periodic in time

Type flag

Default value no

321. ocean.frac_MBP.file

Name of the file containing melange back-pressure scaling.

Type string

Default value *empty*

Option -ocean_frc_MBP_file

322. ocean.frc_MBP.periodic

If true, interpret forcing data as periodic in time

Type flag

Default value no

323. ocean.frc_mass_flux.file

Name of the file containing sub-shelf mass flux scaling.

Type string

Default value *empty*

Option -ocean_frc_mass_flux_file

324. ocean.frc_mass_flux.periodic

If true, interpret forcing data as periodic in time

Type flag

Default value no

325. `ocean.given.file`

Name of the file containing climate forcing fields.

Type string

Default value *empty*

Option `-ocean_given_file`

326. `ocean.given.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

327. `ocean.models`

Comma-separated list of ocean models and modifiers.

Type string

Default value constant

Option `-ocean`

328. `ocean.pico.continental_shelf_depth`

Specifies the depth up to which oceanic input temperatures and salinities are averaged over the continental shelf areas in front of the ice shelf cavities.

Type number

Default value -800 (*meters*)

Option `-continental_shelf_depth`

329. `ocean.pico.exclude_ice_rises`

If set to true, grounding lines of ice rises are excluded in the geometrical routines that determine the ocean boxes; using this option is recommended.

Type flag

Default value yes

Option `-exclude_icerises`

330. `ocean.pico.file`

Specifies the NetCDF file containing potential temperature (`theta_ocean`), salinity (`salinity_ocean`) and ocean basins (`basins`).

Type string

Default value *empty*

Option `-ocean_pico_file`

331. `ocean.pico.heat_exchange_coefficient`

Sets the coefficient for turbulent heat exchange from the ambient ocean across the boundary layer beneath the ice shelf base.

Type number

Default value 2e-05 (*meters second-1*)

- Option** `-gamma_T`
332. `ocean.pico.maximum_ice_rise_area`
 Specifies an area threshold that separates ice rises from continental regions.
Type number
Default value 100000 (*km*²)
333. `ocean.pico.number_of_boxes`
 For each ice shelf the number of ocean boxes is determined by interpolating between 1 and `number_of_boxes` depending on its size and geometry such that larger ice shelves are resolved with more boxes; a value of 5 is suitable for the Antarctic setup.
Type integer
Default value 5 (*pure number*)
Option `-number_of_boxes`
334. `ocean.pico.overturning_coefficient`
 Sets the overturning strength coefficient.
Type number
Default value 1e+06 (*meters*⁶ *seconds*⁻¹ *kg*⁻¹)
Option `-overturning_coeff`
335. `ocean.pico.periodic`
 If true, interpret forcing data as periodic in time
Type flag
Default value no
336. `ocean.pik_melt_factor`
 dimensionless tuning parameter in the `-ocean_pik` ocean heat flux parameterization; see [3]
Type number
Default value 0.005 (*I*)
Option `-meltfactor_pik`
337. `ocean.runoff_to_ocean_melt.b`
 parameter B in eqn. 1 in [120]
Type number
Default value 0.15 (*I*)
338. `ocean.runoff_to_ocean_melt.file`
 Name of the file containing the `delta_T` variable used to scale ocean melt
Type string
Default value *empty*
Option `-ocean_runoff_smb`
339. `ocean.runoff_to_ocean_melt.power_alpha`
 exponent α in eqn. 1 in [119]

Type number

Default value 0.54 (*I*)

340. ocean.runoff_to_ocean_melt.power_beta

exponent β in eqn. 1 in [119]

Type number

Default value 1.17 (*I*)

341. ocean.sub_shelf_heat_flux_into_ice

= J meter-2 second-1; naively chosen default value for heat from ocean; see comments in pism::ocean::Constant::shelf_base_mass_flux().

Type number

Default value 0.5 (*W meter-2*)

342. ocean.th.clip_salinity

Clip shelf base salinity so that it is in the range [4, 40] k/kg. See [11].

Type flag

Default value yes

Option -clip_shelf_base_salinity

343. ocean.th.file

Name of the file containing climate forcing fields.

Type string

Default value empty

Option -ocean_th_file

344. ocean.th.gamma_S

Turbulent salt transfer coefficient. See [11].

Type number

Default value 5.05e-07 (*m s-1*)

Option -gamma_T

345. ocean.th.gamma_T

Turbulent heat transfer coefficient. See [11].

Type number

Default value 0.0001 (*m s-1*)

Option -gamma_T

346. ocean.th.periodic

If true, interpret forcing data as periodic in time

Type flag

Default value no

347. output.ISMIP6

Follow ISMIP6 conventions (units, variable names, “standard names”) when writing output variables.

Type flag

Default value false

348. output.ISMIP6_extra_variables

Comma-separated list of fields reported by models participating in ISMIP6 simulations.

Type string

Default value lithk, orog, topg, hfgeoubed, acabf, libmassbfgr, libmassbffd, dlithkdt, velsurf, zvelsurf, velbase, zvelbase, velmean, litemptop, litemppbotgr, litemppbotfl, strbasemag, licalvf, lifmassbf, sftgif, sftgrf, sftflf

349. output.ISMIP6_ts_variables

Comma-separated list of scalar variables (time series) reported by models participating in ISMIP6 simulations.

Type string

Default value lim, limnsw, iareagr, iareafl, tendacabf, tendlibmassbf, tendlibmassbffd, tendlicalvf, tendlifmassbf

350. output.backup_interval

wall-clock time between automatic backups

Type number

Default value 1 (*hours*)

Option -backup_interval

351. output.backup_size

The “size” of a backup file. See parameters `output.sizes.medium`, `output.sizes.big_2d`, `output.sizes.big`

Type keyword

Default value small

Choices none, small, medium, big_2d, big

Option -backup_size

352. output.compression_level

Compression level for 2D and 3D output variables (if supported by `output.format`)

Type integer

Default value 0 (*count*)

353. output.extra.append

Append to an existing output file. No effect if file does not yet exist, and no effect if `output.extra.split` is set.

Type flag

Default value no

Option -extra_append

354. `output.extra.file`

Name of the file that will contain spatially-variable diagnostics. Should be different from `output.file_name`.

Type string

Default value *empty*

Option `-extra_file`

355. `output.extra.split`

Save spatially-variable diagnostics to separate files (one per time record).

Type flag

Default value no

Option `-extra_split`

356. `output.extra.stop_missing`

Stop if requested variable is not available instead of warning.

Type flag

Default value yes

Option `-extra_stop_missing`

357. `output.extra.times`

List or a range of times defining reporting intervals for spatially-variable diagnostics.

Type string

Default value *empty*

Option `-extra_times`

358. `output.extra.vars`

Comma-separated list of spatially-variable diagnostics.

Type string

Default value *empty*

Option `-extra_vars`

359. `output.file_name`

The file to save final model results to.

Type string

Default value `unnamed.nc`

Option `-o`

360. `output.fill_value`

_FillValue used when saving diagnostic quantities

Type number

Default value -2e+09 (*none*)

361. output.format

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.

Type keyword

Default value netcdf3

Choices netcdf3, netcdf4_serial, netcdf4_parallel, pnetcdf, pio_pnetcdf, pio_netcdf4p, pio_netcdf4c, pio_netcdf

Option -o_format

362. output.ice_free_thickness_standard

If ice is thinner than this standard then a grid cell is considered ice-free for purposes of reporting glacierized area, volume, etc.

Type number

Default value 10 (*meters*)

363. output.pio.base

Rank of the first I/O task

Type integer

Default value 0 (*count*)

364. output.pio.n_writers

Number of I/O tasks to use

Type integer

Default value 1 (*count*)

365. output.pio.stride

Offset between I/O tasks

Type integer

Default value 1 (*count*)

366. output.runtime.area_scale_factor_log10

an integer; log base 10 of scale factor to use for area (in km²) in summary line to stdout

Type integer

Default value 6 (*pure number*)

Option -summary_area_scale_factor_log10

367. output.runtime.time_unit_name

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.)

Type string

Default value year

368. output.runtime.time_use_calendar

Whether to use the current calendar when printing model time in summary to stdout.

Type flag

Default value yes

369. `output.runtime.viewer.size`

default diagnostic viewer size (number of pixels of the longer side)

Type integer

Default value 320 (*count*)

Option `-view_size`

370. `output.runtime.viewer.variables`

comma-separated list of map-plane diagnostic quantities to view at runtime

Type string

Default value *empty*

Option `-view`

371. `output.runtime.volume_scale_factor_log10`

an integer; log base 10 of scale factor to use for volume (in km³) in summary line to stdout

Type integer

Default value 6 (*pure number*)

Option `-summary_vol_scale_factor_log10`

372. `output.size`

The “size” of an output file. See parameters `output.sizes.medium`, `output.sizes.big_2d`, `output.sizes.big`

Type keyword

Default value medium

Choices none, small, medium, big_2d, big

Option `-o_size`

373. `output.sizes.big`

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.

Type string

Default value cts, liqfrac, temp, temp_pa, uvel, vvel, wvel, wvel_rel

374. `output.sizes.big_2d`

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.

Type string

Default value age, bfrikt, bheatflx, bmelt, bwp, bwprel, dbdt, effbwp,
enthalpybase, enthalpsurf, flux_divergence, hardav, hydroinput, lat,
litho_temp, lon, nuH, rank, tempbase, tempicethk, tempicethk_basal,

temp_pabase, tempsurf, thk, thksmooth, tillphi, topg, velbar, velbase,
wallmelt, wvelbase

375. `output.sizes.medium`

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.

Type string

Default 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

376. `output.snapshot.file`

Snapshot (output) file name (or prefix, if saving to individual files).

Type string

Default value *empty*

Option `-save_file`

377. `output.snapshot.size`

The “size” of a snapshot file. See parameters `output.sizes.medium`, `output.sizes.big_2d`, `output.sizes.big`

Type keyword

Default value small

Choices none, small, medium, big_2d, big

Option `-save_size`

378. `output.snapshot.split`

Save model state snapshots to separate files (one per time record).

Type flag

Default value no

Option `-save_split`

379. `output.snapshot.times`

List or a range of times to save model state snapshots at.

Type string

Default value *empty*

Option `-save_times`

380. `output.timeseries.append`

If true, append to the scalar time series output file.

Type flag

Default value false

Option `-ts_append`

381. `output.timeseries.buffer_size`

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.

Type integer

Default value 10000 (*count*)

382. `output.timeseries.filename`

Name of the file to save scalar time series to. Leave empty to disable reporting scalar time-series.

Type string

Default value *empty*

Option `-ts_file`

383. `output.timeseries.times`

List or range of times defining reporting time intervals.

Type string

Default value *empty*

Option `-ts_times`

384. `output.timeseries.variables`

Requested scalar (time-series) diagnostics. Leave empty to save all available diagnostics.

Type string

Default value *empty*

Option `-ts_vars`

385. `output.use_MKS`

Use MKS units in output files.

Type flag

Default value false

386. `regional.no_model_strip`

Default width of the “no model strip” in regional setups.

Type number

Default value 5 (*km*)

Option `-no_model_strip`

387. `regional.no_model_yield_stress`

High yield stress used in the `no_model_mask` area in the regional mode.

Type number

Default value 1000 (*kPa*)

388. `regional.zero_gradient`

Use zero ice thickness and ice surface gradient in the `no_model_mask` area.

Type flag

- Default value** false
Option -zero_grad_where_no_model
389. `run_info.institution`
Institution name. This string is written to output files as the `institution` global attribute.
Type string
Default value *empty*
Option -institution
390. `run_info.title`
Free-form string containing a concise description of the current run. This string is written to output files as the `title` global attribute.
Type string
Default value *empty*
Option -title
391. `sea_level.constant.value`
Sea level elevation used by the constant sea level model
Type number
Default value 0 (*meters*)
392. `sea_level.models`
Comma-separated list of sea level models and modifiers.
Type string
Default value constant
Option -sea_level
393. `stress_balance.blatter.Glen_exponent`
Glen exponent in ice flow law for the Blatter stress balance solver
Type number
Default value 3 (*pure number*)
394. `stress_balance.blatter.Mz`
Number of vertical grid levels in the ice
Type integer
Default value 5 (*count*)
Option -blatter_Mz
395. `stress_balance.blatter.coarsening_factor`
Coarsening factor in the *z* direction
Type integer
Default value 2 (*count*)
Option -blatter_coarsening_factor

396. `stress_balance.blatter.enhancement_factor`

Flow enhancement factor for the Blatter stress balance flow law

Type number

Default value 1 (*I*)

Option `-blatter_e`

397. `stress_balance.blatter.flow_law`

The flow law used by the Blatter-Pattyn stress balance model

Type keyword

Default value `gpbld`

Choices `arr`, `arrwarm`, `gpbld`, `hooke`, `isothermal_glen`, `pb`

398. `stress_balance.blatter.use_eta_transform`

Use the η transform to improve the accuracy of the surface gradient approximation near grounded margins (see [33] for details).

Type flag

Default value no

399. `stress_balance.calving_front_stress_bc`

Apply CFBC condition as in [98], [37]. 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 [29].

Type flag

Default value no

Option `-cfbc`

400. `stress_balance.ice_free_thickness_standard`

If ice is thinner than this standard then a cell is considered ice-free for purposes of computing ice velocity distribution.

Type number

Default value 10 (*meters*)

401. `stress_balance.model`

Stress balance model

Type keyword

Default value sia

Choices `none`, `prescribed_sliding`, `weertman_sliding`, `sia`, `ssa`,
`prescribed_sliding+sia`, `weertman_sliding+sia`, `ssa+sia`, `blatter`

Option `-stress_balance`

402. `stress_balance.prescribed_sliding.file`

The name of the file containing prescribed sliding velocity (variable names: `ubar`, `vbar`).

Type string

Default value *empty*

403. `stress_balance.sia.Glen_exponent`

Glen exponent in ice flow law for SIA

Type number

Default value 3 (*pure number*)

Option `-sia_n`

404. `stress_balance.sia.bed_smoothening.range`

half-width of smoothing domain in the bed roughness parameterization for SIA [83]; set to zero to disable

Type number

Default value 5000 (*meters*)

Option `-bed_smoothening_range`

405. `stress_balance.sia.bed_smoothening.theta_min`

minimum value of θ in the bed roughness parameterization for SIA [83]

Type number

Default value 0 (*I*)

406. `stress_balance.sia.e_age_coupling`

Couple the SIA enhancement factor to age as in [36].

Type flag

Default value no

Option `-e_age_coupling`

407. `stress_balance.sia.enhancement_factor`

Flow enhancement factor for SIA

Type number

Default value 1 (*I*)

Option `-sia_e`

408. `stress_balance.sia.enhancement_factor_interglacial`

Flow enhancement factor for SIA; used for ice accumulated during interglacial periods.

Type number

Default value 1 (*I*)

Option `-sia_e_interglacial`

409. `stress_balance.sia.flow_law`

The SIA flow law.

Type keyword

Default value `gpbld`

Choices `arr`, `arrwarm`, `gk`, `gpbld`, `hooke`, `isothermal_glen`, `pb`

Option `-sia_flow_law`

410. `stress_balance.sia.grain_size_age_coupling`

Use age of the ice to compute grain size to use with the Goldsby-Kohlstedt [25] flow law

Type flag

Default value no

Option `-grain_size_age_coupling`

411. `stress_balance.sia.limit_diffusivity`

Limit SIA diffusivity by `stress_balance.sia.max_diffusivity`.

Type flag

Default value no

Option `-limit_sia_diffusivity`

412. `stress_balance.sia.max_diffusivity`

Maximum allowed diffusivity of the SIA flow. PISM stops with an error message if the SIA diffusivity exceeds this limit.

Type number

Default value 100 ($m^2 s^{-1}$)

413. `stress_balance.sia.surface_gradient_method`

method used for surface gradient calculation at staggered grid points

Type keyword

Default value haseloff

Choices eta, haseloff, mahaffy

Option `-gradient`

414. `stress_balance.ssa.Glen_exponent`

Glen exponent in ice flow law for SSA

Type number

Default value 3 (*pure number*)

Option `-ssa_n`

415. `stress_balance.ssa.compute_surface_gradient_inward`

If yes then use inward first-order differencing in computing surface gradient in the SSA objects.

Type flag

Default value no

416. `stress_balance.ssa.dirichlet_bc`

apply SSA velocity Dirichlet boundary condition

Type flag

Default value no

Option `-ssa_dirichlet_bc`

417. `stress_balance.ssa.enhancement_factor`

Flow enhancement factor for SSA

Type number

Default value 1 (*I*)

Option `-ssa_e`

418. `stress_balance.ssa.epsilon`

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 ($\bar{B} = 1.9\text{e}8 \text{ Pa s}^{1/3}$ [68] and a typical strain rate of 0.001 1/year for the Ross ice shelf, giving $\nu = (\bar{B})/(2 \cdot 0.001^{2/3}) = 9.49\text{e}+14 \text{ Pa s} \sim 30 \text{ MPa year}$, the value in [87], but with a tiny thickness H of about 1 cm.

Type number

Default value 1e+13 (*Pascal second meter*)

Option `-ssa_eps`

419. `stress_balance.ssa.fd.brutal_sliding`

Enhance sliding speed brutally.

Type flag

Default value false

Option `-brutal_sliding`

420. `stress_balance.ssa.fd.brutal_sliding_scale`

Brutal SSA Sliding Scale

Type number

Default value 1 (*I*)

Option `-brutal_sliding_scale`

421. `stress_balance.ssa.fd.flow_line_mode`

Set v (the y component of the ice velocity) to zero when assembling the system

Type flag

Default value false

422. `stress_balance.ssa.fd.lateral_drag.enabled`

Set viscosity at ice shelf margin next to ice free bedrock as friction parameterization

Type flag

Default value false

423. `stress_balance.ssa.fd.lateral_drag.viscosity`

Staggered viscosity used as side friction parameterization.

Type number

Default value 5e+15 (*Pascal second*)

Option `-nu_bedrock`

424. `stress_balance.ssa.fd.max_iterations`

Maximum number of Picard iterations for the ice viscosity computation, in the SSAFD object

Type integer

Default value 300 (*count*)

Option `-ssafd_picard_maxi`

425. `stress_balance.ssa.fd.max_speed`

Upper bound for the ice speed computed by the SSAFD solver.

Type number

Default value 300000 (*km s-1*)

Option `-ssafd_max_speed`

426. `stress_balance.ssa.fd.nuH_iter_failure_underrelaxation`

In event of “Effective viscosity not converged” failure, use outer iteration rule $\text{nuH} \leftarrow \text{nuH} + f (\text{nuH} - \text{nuH}_{\text{old}})$, where f is this parameter.

Type number

Default value 0.8 (*pure number*)

Option `-ssafd_nuH_iter_failure_underrelaxation`

427. `stress_balance.ssa.fd.relative_convergence`

Relative change tolerance for the effective viscosity in the SSAFD object

Type number

Default value 0.0001 (*I*)

Option `-ssafd_picard_rtol`

428. `stress_balance.ssa.fd.replace_zero_diagonal_entries`

Replace zero diagonal entries in the SSAFD matrix with :config:’basal_resistance.beta_ice_free_bedrock’ to avoid solver failures.

Type flag

Default value yes

429. `stress_balance.ssa.flow_law`

The SSA flow law.

Type keyword

Default value gpbld

Choices arr, arrwarm, gpbld, hooke, isothermal_glen, pb

Option `-ssa_flow_law`

430. `stress_balance.ssa.method`

Algorithm for computing the SSA solution.

Type keyword

Default value fd

Choices fd, fem

Option -ssa_method431. `stress_balance.ssa.read_initial_guess`

Read the initial guess from the input file when re-starting.

Type flag**Default value** yes**Option -ssa_read_initial_guess**432. `stress_balance.ssa.strength_extension.constant_nu`

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)= $1.9e8 \text{ Pas}^{1/3}$ [68] 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 [87].

Type number**Default value** $9.48681e+14$ (*Pascal second*)433. `stress_balance.ssa.strength_extension.min_thickness`

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.

Type number**Default value** 50 (*meters*)434. `stress_balance.vertical_velocity_approximation`

Vertical velocity FD approximation. “Upstream” uses first-order finite difference to compute u_x and v_y . Uses basal velocity to make decisions.

Type keyword**Default value** centered**Choices** centered, upstream**Option -vertical_velocity_approximation**435. `stress_balance.weertman_sliding.A`

Sliding parameter in the Weertman-style sliding parameterization [88]

Type number**Default value** $1.8e-16$ (*Pa $^{-3}$ year $^{-1}$ m $^{-2}$*)436. `stress_balance.weertman_sliding.k`

The ratio of the basal water pressure and the ice overburden pressure in the Weertman-style sliding parameterization.

Type number**Default value** 0.2 (*l*)437. `surface.anomaly.file`

Name of the file containing climate forcing fields.

Type string**Default value** *empty*

Option `-surface_anomaly_file`

438. `surface.anomaly.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

439. `surface.cache.update_interval`

Update interval (in 365-day years) for the `-surface cache` modifier.

Type integer

Default value 10 (*365days*)

440. `surface.delta_T.file`

Name of the file containing temperature offsets.

Type string

Default value *empty*

Option `-surface_delta_T_file`

441. `surface.delta_T.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

442. `surface.elevation_change.file`

Name of the file containing the reference surface elevation field (variable `usurf`).

Type string

Default value *empty*

Option `-surface_elevation_change_file`

443. `surface.elevation_change.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

444. `surface.elevation_change.smb.exp_factor`

Exponential for the surface mass balance.

Type number

Default value 0 (*Kelvin-1*)

Option `-smb_exp_factor`

445. `surface.elevation_change.smb.lapse_rate`

Lapse rate for the surface mass balance.

Type number

Default value 0 ((*m/year*)/*km*)

Option `-smb_lapse_rate`446. `surface.elevation_change.smb.method`

Choose the SMB adjustment method. `scale`: use temperature-change-dependent scaling factor. `shift`: use the SMB lapse rate.

Type keyword

Default value `shift`

Choices `scale`, `shift`

Option `-smb_adjustment`

447. `surface.elevation_change.temperature_lapse_rate`

Lapse rate for the temperature at the top of the ice.

Type number

Default value 0 (K/km)

Option `-temp_lapse_rate`

448. `surface.force_to_thickness.alpha`

exponential coefficient in force-to-thickness mechanism

Type number

Default value 0.01 ($year^{-1}$)

Option `-force_to_thickness_alpha`

449. `surface.force_to_thickness.ice_free_alpha_factor`

`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`

Type number

Default value 1 (I)

Option `-force_to_thickness_ice_free_alpha_factor`

450. `surface.force_to_thickness.ice_free_thickness_threshold`

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`.

Type number

Default value 1 ($meters$)

Option `-force_to_thickness_ice_free_thickness_threshold`

451. `surface.force_to_thickness.start_time`

Starting time for the “force to thickness” modifier; the default is “start from the creation of the Earth.”

Type number

Default value -4.54e+09 ($years$)

452. `surface.force_to_thickness_file`

The name of the file to read the target ice thickness from.

Type string

Default value *empty*

Option `-force_to_thickness_file`

453. `surface.given.file`

Name of the file containing climate forcing fields.

Type string

Default value *empty*

Option `-surface_given_file`

454. `surface.given.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

455. `surface.given.smb_max`

Maximum climatic mass balance value (used to check input data). Corresponds to 100 m/year ice equivalent.

Type number

Default value 91000 (*kg m⁻² year⁻¹*)

456. `surface.ismip6.file`

Name of the file containing climate forcing anomaly fields.

Type string

Default value *empty*

Option `-surface_ismip6_file`

457. `surface.ismip6.periodic`

If true, interpret forcing data as periodic in time

Type flag

Default value no

458. `surface.ismip6.reference_file`

Name of the file containing reference climate forcing fields.

Type string

Default value *empty*

Option `-surface_ismip6_reference_file`

459. `surface.models`

Comma-separated list of surface models and modifiers.

Type string

Default value given

Option `-surface`

460. `surface.pdd.air_temp_all_precip_as_rain`

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

Type number

Default value 275.15 (*Kelvin*)

461. `surface.pdd.air_temp_all_precip_as_snow`

threshold temperature below which all precipitation is snow

Type number

Default value 273.15 (*Kelvin*)

462. `surface.pdd.balance_year_start_day`

day of year for October 1st, beginning of the balance year in northern latitudes.

Type integer

Default value 274 (*ordinal day number*)

463. `surface.pdd.factor_ice`

EISMINT-Greenland value [7]; = (8 mm liquid-water-equivalent) / (pos degree day)

Type number

Default value 0.00879121 (*meter / (Kelvin day)*)

464. `surface.pdd.factor_snow`

EISMINT-Greenland value [7]; = (3 mm liquid-water-equivalent) / (pos degree day)

Type number

Default value 0.0032967 (*meter / (Kelvin day)*)

465. `surface.pdd.fausto.T_c`

= -1 + 273.15; for formula (6) in [1]

Type number

Default value 272.15 (*Kelvin*)

466. `surface.pdd.fausto.T_w`

= 10 + 273.15; for formula (6) in [1]

Type number

Default value 283.15 (*Kelvin*)

467. `surface.pdd.fausto.betta_ice_c`

water-equivalent thickness; for formula (6) in [1]

Type number

Default value 0.015 (*meter / (Kelvin day)*)

468. `surface.pdd.fausto.betta_ice_w`

water-equivalent thickness; for formula (6) in [1]

Type number

Default value 0.007 (*meter / (Kelvin day)*)
469. `surface.pdd.fausto.beta_snow_c`

water-equivalent thickness; for formula (6) in [1]

Type number

Default value 0.003 (*meter / (Kelvin day)*)

470. `surface.pdd.fausto.beta_snow_w`
water-equivalent thickness; for formula (6) in [1]

Type number

Default value 0.003 (*meter / (Kelvin day)*)

471. `surface.pdd.fausto.enabled`
Set PDD parameters using formulas (6) and (7) in [1]

Type flag

Default value false

Option `-pdd_fausto`

472. `surface.pdd.fausto.latitude_beta_w`
latitude below which to use warm case, in formula (6) in [1]

Type number

Default value 72 (*degree_north*)

473. `surface.pdd.firn_compaction_to_accumulation_ratio`
How much firn as a fraction of accumulation is turned into ice

Type number

Default value 0.75 (1)

474. `surface.pdd.firn_depth_file`

The name of the file to read the firn_depth from.

Type string

Default value *empty*

Option `-pdd_firn_depth_file`

475. `surface.pdd.interpret_precip_as_snow`

Interpret precipitation as snow fall.

Type flag

Default value no

476. `surface.pdd.max_evals_per_year`

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.value`

Type integer

Default value 52 (*count*)

477. `surface.pdd.method`

PDD implementation method

Type keyword

Default value `expectation_integral`

Choices `expectation_integral`, `repeatable_random_process`, `random_process`

Option `-pdd_method`

478. `surface.pdd.positive_threshold_temp`

temperature used to determine meaning of “positive” degree day

Type number

Default value 273.15 (*Kelvin*)

479. `surface.pdd.refreeze`

EISMINT-Greenland value [7]

Type number

Default value 0.6 (*I*)

480. `surface.pdd.refreeze_ice_melt`

If set to “yes”, refreeze `surface.pdd.refreeze` fraction of melted ice, otherwise all of the melted ice runs off.

Type flag

Default value yes

481. `surface.pdd.std_dev.file`

The name of the file to read `air_temp_sd` (standard deviation of air temperature) from.

Type string

Default value `empty`

Option `-pdd_sd_file`

482. `surface.pdd.std_dev.lapse_lat_base`

standard deviation is a function of latitude, with value `surface.pdd.std_dev.value` at this latitude; this value is only active if `surface.pdd.std_dev.lapse_lat_rate` is nonzero

Type number

Default value 72 (*degree_north*)

483. `surface.pdd.std_dev.lapse_lat_rate`

standard deviation is a function of latitude, with rate of change with respect to latitude given by this constant

Type number

Default value 0 (*Kelvin / degree_north*)

484. `surface.pdd.std_dev.param_a`

Parameter a in $\Sigma = aT + b$, with T in degrees C. Used only if `surface.pdd.std_dev.use_param` is set to yes.

Type number

Default value -0.15 (*pure number*)

485. `surface.pdd.std_dev.param_b`

Parameter b in $\Sigma = aT + b$, with T in degrees C. Used only if `surface.pdd.std_dev.use_param` is set to yes.

Type number

Default value 0.66 (*Kelvin*)

486. `surface.pdd.std_dev.periodic`

If true, interpret `air_temp_sd` read from `surface.pdd.std_dev.file` as periodic in time

Type flag

Default value no

487. `surface.pdd.std_dev.use_param`

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`.

Type flag

Default value no

488. `surface.pdd.std_dev.value`

standard deviation of daily temp variation; = EISMINT-Greenland value [7]

Type number

Default value 5 (*Kelvin*)

489. `surface.pressure`

atmospheric pressure; = pressure at ice surface

Type number

Default value 0 (*Pascal*)

490. `surface.temp_to_runoff_a`

a in runoff=a * temp + b

Type number

Default value 0.5 (*K-l*)

491. `time.calendar`

The calendar to use.

Type keyword

Default value 365_day

Choices standard, gregorian, proleptic_gregorian, noleap, 365_day, 360_day, julian

Option -calendar

492. `time.dimension_name`

The name of the time dimension in PISM output files.

Type string

Default value `time`493. `time.eemian_end`

End of the Eemian interglacial period. See [86].

Type number**Default value** -114500 (*years*)494. `time.eemian_start`

Start of the Eemian interglacial period. See [86].

Type number**Default value** -132000 (*years*)495. `time.end`End time, interpreted relative to `time.reference_date`.**Type** string**Default value** *empty***Option** -ye496. `time.file`Name of the file used to set `time.start`, `time.end`, `time.calendar`, and `time.reference_date`**Type** string**Default value** *empty***Option** -time_file497. `time.file.continue`If true, don't set `time.start` using `time.file` (helpful when continuing an interrupted simulation)**Type** flag**Default value** false498. `time.holocene_start`

Start of the Holocene interglacial period. See [86].

Type number**Default value** -11000 (*years*)499. `time.reference_date`

year-month-day; reference date used for calendar computations and in PISM output files

Type string**Default value** 1-1-1500. `time.run_length`

Run length

Type number**Default value** 1000 (*365days*)**Option** -y

501. `time.start`

Start time, interpreted relative to `time.reference_date`.

Type string

Default value *empty*

Option `-ys`

502. `time_stepping.adaptive_ratio`

Adaptive time stepping ratio for the explicit scheme for the mass balance equation; [30], inequality (25)

Type number

Default value 0.12 (1)

Option `-adapt_ratio`

503. `time_stepping.count_steps`

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.

Type flag

Default value no

Option `-count_steps`

504. `time_stepping.hit_extra_times`

Modify the time-stepping mechanism to hit times requested using `output.extra.times`.

Type flag

Default value yes

Option `-extra_force_output_times`

505. `time_stepping.hit_multiples`

Hit every X years, where X is specified using this parameter. Use 0 to disable.

Type integer

Default value 0 (years)

Option `-timestep_hit_multiples`

506. `time_stepping.hit_save_times`

Modify the time-stepping mechanism to hit times requested using `output.snapshot.times`.

Type flag

Default value no

Option `-save_force_output_times`

507. `time_stepping.hit_ts_times`

Modify the time-stepping mechanism to hit times requested using `output.timeseries.times`.

Type flag

Default value no

508. `time_stepping.maximum_time_step`

Maximum allowed time step length

Type number

Default value 60 (*365days*)

Option `-max_dt`

509. `time_stepping.resolution`

Time steps are rounded down to be a multiple of this number (set to zero to allow arbitrary time step lengths)

Type number

Default value 1 (*seconds*)

510. `time_stepping.skip.enabled`

Use the temperature, age, and SSA stress balance computation skipping mechanism.

Type flag

Default value no

Option `-skip`

511. `time_stepping.skip.max`

Number of mass-balance steps, including SIA diffusivity updates, to perform before a the temperature, age, and SSA stress balance computations are done

Type integer

Default value 10 (*count*)

Option `-skip_max`

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_summer

Units Kelvin

Description mean summer 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. bfrcit

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 subglacial water layer

13. `bwatvel`

- `bwatvel[0]`

Units m s-1

Description velocity of water in subglacial layer, i-offset

- `bwatvel[1]`

Units m s-1

Description velocity of water in subglacial layer, j-offset

14. `bpw`

Units Pa

Description pressure of transportable water in subglacial layer

15. `bwpref`

Units —

Description pressure of transportable water in subglacial layer as fraction of the overburden pressure

16. `cell_grounded_fraction`

Units —

Description fractional grounded/floating mask (floating=0, grounded=1)

17. `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

18. `cts`

Units 1

Description cts = E/E_s(p), so cold-temperate transition surface is at cts = 1

19. `dHdt`

Units m year-1

Description ice thickness rate of change

Standard name tendency_of_land_ice_thickness

20. dbdt

Units mm year-1

Description bedrock uplift rate

Standard name tendency_of_bedrock_altitude

21. 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

22. diffusivity

Units m² s⁻¹

Description diffusivity of SIA mass continuity equation

23. 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)

24. effbwp

Units Pa

Description effective pressure of transportable water in subglacial layer (overburden pressure minus water pressure)

25. effective_air_temp

Units Kelvin

Description effective mean-annual near-surface air temperature

26. effective_precipitation

Units kg m⁻² year⁻¹

Description effective precipitation rate

Standard name precipitation_flux

27. effective_viscosity

- Units** kPascal second
Description effective viscosity of ice
28. `effective_water_velocity`
- `u_effective_water_velocity`
Units m day⁻¹
Description x-component of the effective water velocity in the steady-state hydrology model
 - `v_effective_water_velocity`
Units m day⁻¹
Description y-component of the effective water velocity in the steady-state hydrology model
29. `eigen_calving_rate`
Units m year⁻¹
Description horizontal calving rate due to eigen-calving
30. `elastic_bed_displacement`
Units meters
Description elastic part of the displacement in the Lingle-Clark bed deformation model; see [27]
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. `enthalpsurf`
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 day⁻¹

Description frontal melt rate

40. `frontal_melt_retreat_rate`

Units m year⁻¹

Description retreat rate due to frontal melt

Comment takes into account what part of the front is submerged

41. `grounding_line_flux`

Units kg m⁻² year⁻¹

Description grounding line flux

Comment Positive flux corresponds to mass moving from the ocean to an icy grounded area. This convention makes it easier to compare grounding line flux to the total discharge into the ocean

42. `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

43. `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

44. `hardav`

Units Pa s0.333333

Description vertical average of ice hardness

45. hardness

Units Pa s0.333333

Description ice hardness computed using the SIA flow law

46. heat_flux_from_bedrock

Units mW m⁻²

Description upward geothermal flux at the top bedrock surface

Standard name upward_geothermal_heat_flux_at_ground_level_in_land_ice

Comment positive values correspond to an upward flux

47. height_above_flotation

Units m

Description ice thickness in excess of the maximum floating ice thickness

Comment shows how close to floatation the ice is at a given location

48. hydraulic_potential

Units Pa

Description hydraulic potential in the subglacial hydrology system

49. hydraulic_potential_adjustment

Units Pa

Description potential adjustment needed to fill sinks when computing an estimate of the steady-state hydraulic potential

50. hydraulic_sinks

Units —

Description map of sinks in the domain (for debugging)

51. 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

52. ice_base_elevation

Units m

Description ice bottom surface elevation

53. ice_margin_pressure_difference

Units Pa

Description vertically-averaged pressure difference at ice margins (including calving fronts)

54. ice_mass

Units kg

Description mass per cell

55. ice_surface_liquid_water_fraction

Units 1

Description ice liquid water fraction at the ice surface

56. `ice_surface_temp`

Units K

Description ice temperature at the top ice surface

Standard name `temperature_at_top_of_ice_sheet_model`

57. `lat`

Units degree_north

Description latitude

Standard name `latitude`

58. `liqfrac`

Units 1

Description liquid water fraction in ice (between 0 and 1)

59. `lon`

Units degree_east

Description longitude

Standard name `longitude`

60. `mask`

Units —

Description ice-type (ice-free/grounded/floating/ocean) integer mask

61. `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

62. `pressure`

Units Pa

Description pressure in ice (hydrostatic)

63. `rank`

Units 1

Description processor rank

64. `remaining_water_thickness`

Units m

Description scaled water thickness in the steady state hydrology model (has no physical meaning)

65. schoofs_theta

Units 1

Description multiplier ‘theta’ in Schoof’s (2003) theory of bed roughness in SIA

66. sea_level

Units meters

Description sea level elevation, relative to the geoid

67. sftflf

Units 1

Description fraction of a grid cell covered by floating ice

Standard name floating_ice_shelf_area_fraction

68. sftgif

Units 1

Description fraction of a grid cell covered by ice (grounded or floating)

Standard name land_ice_area_fraction

69. sftgrf

Units 1

Description fraction of a grid cell covered by grounded ice

Standard name grounded_ice_sheet_area_fraction

70. shelfbmassflux

Units kg m-2 s-1

Description mass flux at the basal surface of ice shelves

71. shelfbtemp

Units Kelvin

Description ice temperature at the basal surface of ice shelves

72. snow_depth

Units m

Description snow cover depth (set to zero once a year)

73. steady_state_hydraulic_potential

Units Pa

Description estimate of the steady state hydraulic potential in the steady hydrology model

74. 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
75. `strainheat`
- Units** mW m⁻³
- Description** rate of strain heating in ice (dissipation heating)
76. `subglacial_water_flux_mag`
- Units** m² year⁻¹
- Description** magnitude of the subglacial water flux
77. `subglacial_water_input_rate`
- Units** m year⁻¹
- Description** water input rate from the ice surface into the subglacial water system
- Comment** positive flux corresponds to water gain
78. `surface_accumulation_flux`
- Units** kg m⁻² year⁻¹
- Description** accumulation (precipitation minus rain), averaged over the reporting interval
79. `surface_accumulation_rate`
- Units** Gt year⁻¹
- Description** accumulation (precipitation minus rain), averaged over the reporting interval
80. `surface_layer_mass`
- Units** kg
- Description** mass of the surface layer (snow and firn)
81. `surface_layer_thickness`
- Units** meters
- Description** thickness of the surface layer (snow and firn)
82. `surface_melt_flux`
- Units** kg m⁻² year⁻¹
- Description** surface melt, averaged over the reporting interval
- Standard name** `surface_snow_and_ice_melt_flux`
83. `surface_melt_rate`
- Units** Gt year⁻¹
- Description** surface melt, averaged over the reporting interval
84. `surface_runoff_flux`
- Units** kg m⁻² year⁻¹
- Description** surface runoff, averaged over the reporting interval
- Standard name** `surface_runoff_flux`
85. `surface_runoff_rate`
- Units** Gt year⁻¹

Description surface runoff, averaged over the reporting interval

86. 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)

87. taub_mag

Units Pa

Description magnitude of the basal shear stress at the base of ice

Standard name land_ice_basal_drag

Comment this field is purely diagnostic (not used by the model)

88. tau_c

Units Pa

Description yield stress for basal till (plastic or pseudo-plastic model)

89. tau_d

- tau_d_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

- tau_d_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

90. tau_d_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

91. tau_xz

Units Pa

Description shear stress xz component (in shallow ice approximation SIA)

92. tau_yz

Units Pa

Description shear stress yz component (in shallow ice approximation SIA)

93. `temp`

Units K

Description ice temperature

Standard name land_ice_temperature

94. `temp_pa`

Units deg_C

Description pressure-adjusted ice temperature (degrees above pressure-melting point)

95. `tempbase`

Units K

Description ice temperature at the base of ice

Standard name land_ice_basal_temperature

96. `tempicethk`

Units m

Description temperate ice thickness (total column content)

97. `tempicethk_basal`

Units m

Description thickness of the basal layer of temperate ice

98. `temppabase`

Units Celsius

Description pressure-adjusted ice temperature at the base of ice

99. `tempsurf`

Units K

Description ice temperature at 1m below the ice surface

Standard name temperature_at_ground_level_in_snow_or_firn

100. `tendency_of_ice_amount`

Units kg m⁻² year⁻¹

Description rate of change of the ice amount

101. `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

102. `tendency_of_ice_amount_due_to_calving`

Units kg m⁻² year⁻¹

Description calving flux

Standard name land_ice_specific_mass_flux_due_to_calving

- Comment** positive flux corresponds to ice gain
103. `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
104. `tendency_of_ice_amount_due_to_discharge`
- Units** kg m⁻² year⁻¹
- Description** discharge flux (calving, frontal melt, forced retreat)
- Standard name** land_ice_specific_mass_flux_due_to_calving_and_ice_front_melting
- Comment** positive flux corresponds to ice gain
105. `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
106. `tendency_of_ice_amount_due_to_surface_mass_flux`
- Units** kg m⁻² year⁻¹
- Description** average surface mass flux over reporting interval
- Standard name** land_ice_surface_specific_mass_balance_flux
- Comment** positive flux corresponds to ice gain
107. `tendency_of_ice_mass`
- Units** Gt year⁻¹
- Description** rate of change of the ice mass
108. `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
109. `tendency_of_ice_mass_due_to_calving`
- Units** Gt year⁻¹
- Description** calving flux
- Comment** positive flux corresponds to ice gain
110. `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
111. `tendency_of_ice_mass_due_to_discharge`

Units Gt year-1

Description discharge flux (calving, frontal melt, forced retreat)

Comment positive flux corresponds to ice gain

112. `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

113. `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

114. `tendency_of_subglacial_water_mass`

Units Gt year-1

Description rate of change of the total mass of subglacial water

Comment positive flux corresponds to water gain

115. `tendency_of_subglacial_water_mass_at_domain_boundary`

Units Gt year-1

Description subglacial water flux at domain boundary (in regional model configurations)

Comment positive flux corresponds to water gain

116. `tendency_of_subglacial_water_mass_at_grounded_margins`

Units Gt year-1

Description subglacial water flux at grounded ice margins

Comment positive flux corresponds to water gain

117. `tendency_of_subglacial_water_mass_at_grounding_line`

Units Gt year-1

Description subglacial water flux at grounding lines

Comment positive flux corresponds to water gain

118. `tendency_of_subglacial_water_mass_due_to_conservation_error`

Units Gt year-1

Description subglacial water flux due to conservation error (mass added to preserve non-negativity)

Comment positive flux corresponds to water gain

119. `tendency_of_subglacial_water_mass_due_to_flow`

Units Gt year-1

Description rate of change subglacial water mass due to lateral flow

Comment positive flux corresponds to water gain

120. `tendency_of_subglacial_water_mass_due_to_input`

- Units** Gt year-1
Description subglacial water flux due to input
Comment positive flux corresponds to water gain
121. thickness_calving_threshold
- Units** m
Description threshold used by the ‘calving at threshold’ calving method
122. thk
- Units** m
Description land ice thickness
Standard name land_ice_thickness
123. thk_bc_mask
- Units** —
Description Mask specifying locations where ice thickness is held constant
124. thksmooth
- Units** m
Description thickness relative to smoothed bed elevation in Schoof’s (2003) theory of bed roughness in SIA
125. tillphi
- Units** degrees
Description friction angle for till under grounded ice sheet
126. tillwat
- Units** m
Description effective thickness of subglacial water stored in till
127. topg
- Units** m
Description bedrock surface elevation
Standard name bedrock_altitude
128. topg_sl_adjusted
- Units** meters
Description sea-level adjusted bed topography (zero is at sea level)
129. topgsmooth
- Units** m
Description smoothed bed elevation in Schoof’s (2003) theory of bed roughness in SIA
130. usurf
- Units** m
Description ice top surface elevation

Standard name surface_altitude

131. uvel

Units m year-1

Description horizontal velocity of ice in the X direction

Standard name land_ice_x_velocity

132. vel_bc_mask

Units —

Description Mask prescribing Dirichlet boundary locations for the sliding velocity

133. vel_bc_values

- u_bc

Units m year-1

Description X-component of the SSA velocity boundary conditions

- v_bc

Units m year-1

Description Y-component of the SSA velocity boundary conditions

134. 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

135. velbar_mag

Units m year-1

Description magnitude of vertically-integrated horizontal velocity of ice

136. 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

137. `velbase_mag`

Units m year-1

Description magnitude of horizontal velocity of ice at base of ice

138. `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

139. `velsurf_mag`

Units m year-1

Description magnitude of horizontal velocity of ice at ice surface

140. `viscous_bed_displacement`

Units meters

Description bed displacement in the viscous half-space bed deformation model; see BuelerLingle-Brown

141. `vonmises_calving_rate`

Units m year-1

Description horizontal calving rate due to von Mises calving

142. `vonmises_calving_threshold`

Units Pa

Description threshold used by the ‘von Mises’ calving method

143. `vonmises_stress`

Units Pascal

Description tensile von Mises stress

144. `vvel`

Units m year-1

Description horizontal velocity of ice in the Y direction

Standard name land_ice_y_velocity

145. `wallmelt`

Units m year-1

Description wall melt into subglacial hydrology layer from (turbulent) dissipation of energy in trans-portable water

146. `wvel`

Units m year-1

Description vertical velocity of ice, relative to geoid

147. wvel_rel

Units m year-1

Description vertical velocity of ice, relative to base of ice directly below

148. 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

149. 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 Gt year-1

Description total sub-shelf ice flux

Standard name tendency_of_land_ice_mass_due_to_basal_mass_balance

Comment positive means ice gain

2. basal_mass_flux_grounded

Units Gt year-1

Description total over grounded ice domain of basal mass flux

Standard name tendency_of_land_ice_mass_due_to_basal_mass_balance

Comment positive means ice gain

3. dt

Units year

Description mass continuity time step

4. grounding_line_flux

Units Gt year-1

Description total ice flux across the grounding line

Comment negative flux corresponds to ice loss into the ocean

5. ice_area_glacierized

Units m²

Description glacierized area

6. ice_area_glacierized_cold_base

- Units** m²
- Description** glacierized area where basal ice is cold
7. `ice_area_glacierized_floating`
- Units** m²
- Description** area of ice shelves in glacierized areas
- Standard name** floating_ice_shelf_area
8. `ice_area_glacierized_grounded`
- Units** m²
- Description** area of grounded ice in glacierized areas
- Standard name** grounded_ice_sheet_area
9. `ice_area_glacierized_temperate_base`
- Units** m²
- Description** glacierized area where basal ice is temperate
10. `ice_enthalpy`
- Units** J
- Description** enthalpy of the ice, including seasonal cover
11. `ice_enthalpy_glacierized`
- Units** J
- Description** enthalpy of the ice in glacierized areas
12. `ice_mass`
- Units** kg
- Description** mass of the ice, including seasonal cover
- Standard name** land_ice_mass
13. `ice_mass_glacierized`
- Units** kg
- Description** mass of the ice in glacierized areas
14. `ice_volume`
- Units** m³
- Description** volume of the ice, including seasonal cover
15. `ice_volume_cold`
- Units** m³
- Description** volume of cold ice, including seasonal cover
16. `ice_volume_glacierized`
- Units** m³
- Description** volume of the ice in glacierized areas
17. `ice_volume_glacierized_cold`

Units m³

Description volume of cold ice in glacierized areas

18. `ice_volume_glacierized_floating`

Units m³

Description volume of ice shelves in glacierized areas

19. `ice_volume_glacierized_grounded`

Units m³

Description volume of grounded ice in glacierized areas

20. `ice_volume_glacierized_temperate`

Units m³

Description volume of temperate ice in glacierized areas

21. `ice_volume_temperate`

Units m³

Description volume of temperate ice, including seasonal cover

22. `limnsw`

Units kg

Description mass of the ice not displacing sea water

Standard name land_ice_mass_not_displacing_sea_water

23. `liquefied_ice_flux`

Units m³ / year

Description rate of ice loss due to liquefaction, averaged over the reporting interval

Comment positive means ice loss

24. `max_diffusivity`

Units m² s⁻¹

Description maximum diffusivity

25. `max_sliding_vel`

Units m year⁻¹

Description max(max(abs(u)), max(abs(v))) for the sliding velocity of ice over grid in last time step
during time-series reporting interval

26. `sea_level_rise_potential`

Units m

Description the sea level rise that would result if all the ice were melted

27. `surface_accumulation_rate`

Units kg year⁻¹

Description surface accumulation rate (PDD model)

28. `surface_melt_rate`

- Units** kg year-1
Description surface melt rate (PDD model)
29. `surface_runoff_rate`
Units kg year-1
Description surface runoff rate (PDD model)
30. `tendency_of_ice_mass`
Units Gt year-1
Description rate of change of the mass of ice, including seasonal cover
31. `tendency_of_ice_mass_due_to_basal_mass_flux`
Units Gt year-1
Description total over ice domain of bottom surface ice mass flux
Standard name `tendency_of_land_ice_mass_due_to_basal_mass_balance`
Comment positive means ice gain
32. `tendency_of_ice_mass_due_to_calving`
Units Gt year-1
Description calving flux
Standard name `tendency_of_land_ice_mass_due_to_calving`
Comment positive means ice gain
33. `tendency_of_ice_mass_due_to_conservation_error`
Units Gt year-1
Description total numerical flux needed to preserve non-negativity of ice thickness
Comment positive means ice gain
34. `tendency_of_ice_mass_due_to_discharge`
Units Gt year-1
Description discharge flux (frontal melt, calving, forced retreat)
Standard name `tendency_of_land_ice_mass_due_to_calving_and_ice_front_melting`
Comment positive means ice gain
35. `tendency_of_ice_mass_due_to_flow`
Units Gt year-1
Description rate of change of the mass of ice due to flow (i.e. prescribed ice thickness)
36. `tendency_of_ice_mass_due_to_surface_mass_flux`
Units Gt year-1
Description total over ice domain of top surface ice mass flux
Standard name `tendency_of_land_ice_mass_due_to_surface_mass_balance`
Comment positive means ice gain
37. `tendency_of_ice_mass_glacierized`

Units Gt year-1

Description rate of change of the ice mass in glacierized areas

38. `tendency_of_ice_volume`

Units m3 year-1

Description rate of change of the ice volume, including seasonal cover

39. `tendency_of_ice_volume_glacierized`

Units m3 year-1

Description rate of change of the ice volume in glacierized areas

CHAPTER
FOUR

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 or 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 “adjustments” of various kinds, such as lapse rate corrections (sections *Adjustments using modeled change in surface elevation* and *Adjustments using modeled change in surface elevation*) 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_T -ocean_delta_T_file delta_T.nc
```

combines the component providing constant (both in space and time) ocean boundary conditions with a modifier that applies scalar temperature 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 interprets time-dependent inputs (Section *Using time-dependent forcing*).

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; arrows in this figure indicate the data flow.

4.1 Using time-dependent forcing

Contents

- *Using time-dependent forcing*
 - *Introduction*
 - *Scalar time-dependent inputs*
 - *Spatially-variable time-dependent inputs*
 - *Periodic forcing*
 - *Adding time bounds*
 - * *Times as mid-points of intervals*
 - * *Times as left end points of intervals*
 - * *Times as right end points of intervals*

4.1.1 Introduction

PISM can use time-dependent *scalar* and *spatially-variable* forcing inputs.

The `ncdump` output for a typical forcing file would look similar to *this example*.

Listing 4.1: Example time-dependent forcing

```
netcdf delta_T {  
dimensions:  
    nv = 2 ;  
    time = UNLIMITED ; // (4 currently)  
variables:  
    double nv(nv) ;  
    double time(time) ;  
    time:units = "365 days since 1-1-1" ;  
    time:axis = "T" ;  
    time:bounds = "time_bounds" ;  
    time:calendar = "365_day" ;  
    time:long_name = "time" ;  
    double time_bounds(time, nv) ;  
    double delta_T(time) ;  
    delta_T:units = "Kelvin" ;  
    delta_T:long_name = "temperature offsets" ;  
data:  
  
    time      = 0, 100, 900, 1000;  
    time_bounds = 0, 100, 100, 500, 500, 900, 900, 1000;  
    delta_T    = 0, -30, -30, 0;  
}
```

A data set like this one could be used to model a scenario in which the temperature at the top surface of the ice drops by 30 degrees over the course of 100 years, remains constant for 800 years, then increases by 30 degrees over 100 years. See [Scalar temperature offsets](#).

In addition to *times*, all forcing files have to contain *time bounds* defining time intervals corresponding to individual records.

Note: PISM will check if the modeled time interval (set using `time.start` and `time.end` or `time.run_length`) is a subset of the time covered by a forcing file and **stop** if it is not.

Set `input.forcing.time_extrapolation` to `true` to tell PISM to use constant extrapolation instead.

4.1.2 Scalar time-dependent inputs

All scalar time-dependent inputs are interpreted as *piecewise-linear*.

Time bounds are used to compute period length for periodic forcing and the time interval covered by provided data otherwise. Only the left end point of the first interval and the right end point of the last interval are used.

4.1.3 Spatially-variable time-dependent inputs

To make *balancing the books* possible PISM interprets *fluxes* such as the top surface mass balance, precipitation, and the sub shelf mass flux as *piecewise-constant in time* over intervals defined by time bounds.

In this case *times* corresponding to individual forcing records are irrelevant (and are ignored) and only time bounds are used.

Other 2D input fields (examples: ice surface temperature, near-surface air temperature, sea level elevation) are interpreted as *piecewise-linear in time*.

In this case times are read from the file and time bounds are used to compute period length for periodic forcing and the time interval covered by provided data otherwise.

4.1.4 Periodic forcing

A PISM forcing file with a periodic forcing **has to contain exactly one period**.

The left end point of the first time interval defines the *start of the period*.

The total duration of forcing in the file defines the length of the period. Specifically, the length of the period is the difference of the right end point of the last interval and the left end point of the first interval.

When used as periodic forcing, *Example time-dependent forcing* would be interpreted as having the period of one thousand 365-day years, with the period starting on January 1 of year 1.

Note:

- A real life (Gregorian, Julian, etc) calendar does not usually make sense in simulations using periodic forcing.
 - It is usually a good idea to use time units that are an integer multiple of one second, for example “day” or “365 days” as in the example above. This makes forcing files easier to interpret. (The units “years” corresponds to the mean tropical year. This is appropriate when converting from m/s to m/year, for example, but not for keeping track of time.)
-

4.1.5 Adding time bounds

NCO's `ncap2` makes it easy to add time bounds to a data set.

Save one of the scripts below to `add_time_bounds.txt` and then run

```
ncap2 -O -S add_time_bounds.txt forcing.nc forcing-with-bounds.nc
```

to add time bounds to `forcing.nc`.

Times as mid-points of intervals

Use this script to interpret each time as a mid-point of the corresponding interval.

```
defdim("nv",2);
time_bnds=make_bounds(time,$nv,"time_bnds");
```

or just use this one command:

Listing 4.2: Adding time bounds, interpreting times as mid-points of intervals

```
ncap2 -O -s 'defdim("nv",2);time_bnds=make_bounds(time,$nv,"time_bnds");' \
forcing.nc forcing-with-bounds.nc
```

Times as left end points of intervals

Use this script to interpret each time as a the left end point of the corresponding interval.

Listing 4.3: Adding time bounds, interpreting times as left end points of intervals

```
time@bounds="time_bnds";
defdim("nv",2);
time_bnds=array(0,0,/time,$nv);
time_bnds(:,0)=time;
time_bnds(:-2,1)=time(1:);
time_bnds(-1,1)=2*time(-1)-time(-2);
```

Times as right end points of intervals

Use this script to interpret each time as a the right end point of the corresponding interval.

Listing 4.4: Adding time bounds, interpreting times as right end points of intervals

```
time@bounds="time_bnds";
defdim("nv",2);
time_bnds=array(0,0,/time,$nv);
time_bnds(:,1)=time;
time_bnds(1:,0)=time(:-2);
time_bnds(0,0)=2*time(0)-time(1);
```

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 [7].

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 [8], precipitation adjustments [2], and sea level offsets from SPECMAP [9].

Variables `precipitation, delta_T, delta_SL, lat, lon`

Options `-atmosphere searise_greenland, delta_T -atmosphere_delta_T_file delta_T.nc -surface pdd -sea_level 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 -sea_level 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:

```
mpixexec -n 2 pismr -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 `pismr -eisII A` merely generates demonstration climate data, using EISMINT II choices [22]. 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 firm* and it does not include seasonal variations [23].

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,precip_scaling \
    -surface pdd -atmosphere_precip_scaling_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 [18].

We can also test the surface temperature forcing code with the following command.



Fig. 4.1: Time series of the surface mass balance rate and near-surface air temperature.

```
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 \
    -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*
 - *Adjustments using modeled change in surface elevation*
 - *Mass flux adjustment*
 - *Using climate data anomalies*
 - *The caching modifier*
 - *Preventing grounding line retreat*

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.

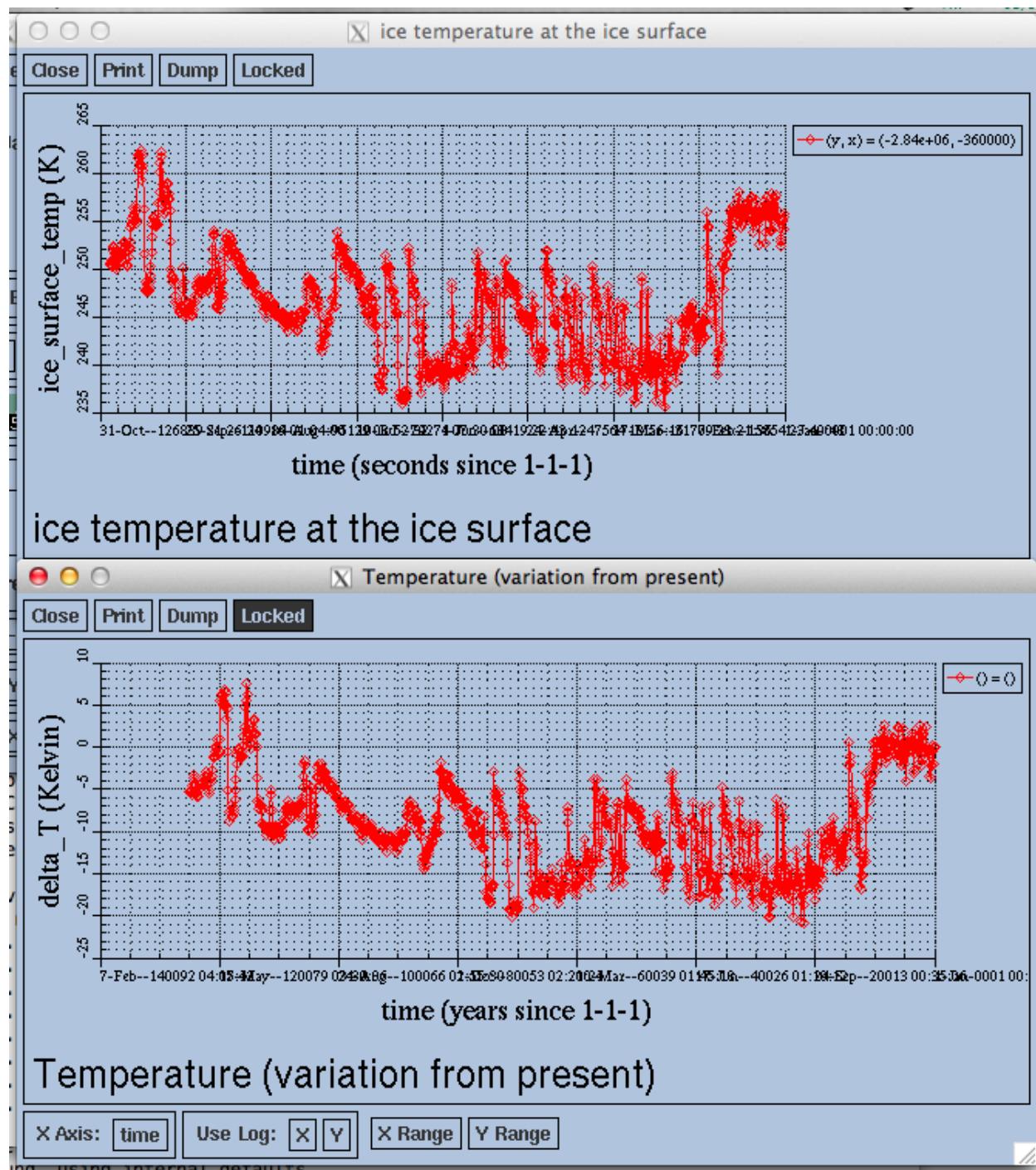


Fig. 4.2: Time series of the surface temperature compared to GRIP temperature offsets

4.4.2 Reading top-surface boundary conditions from a file

Options -surface given

Variables ice_surface_temp, climatic_mass_balance $kg/(m^2 s)$

C++ class pism::surface::Given

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 firn) and `climatic_mass_balance` (top surface mass flux into the ice) are not present in the input file.

A file `foo.nc` used with `-surface given -surface_given_file foo.nc` may 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. Variables `time` and `time_bounds` should specify model times corresponding to individual records.

For example, to use monthly periodic forcing with a period of 1 year starting at the beginning of 1980 (let's use the 360-day calendar for simplicity), create a file (say, "`foo.nc`") with 12 records. The `time` variable may contain 15, 45, 75, ..., 345 (mid-month for all 12 months) and have the units of "days since 1980-1-1". (It is best to avoid units of "months" and "years" because their meanings depend on the calendar.) Next, add the `time_bounds` variable for the time dimension with the values 0, 30, 30, 60, ... specifying times corresponding to beginnings and ends of records for each month and set the `time:bounds` attribute accordingly. Now run

```
pismr -surface given -surface_given_file foo.nc -surface.given.periodic
```

See [Using time-dependent forcing](#) for more information.

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 `input_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.

Parameters

Prefix: `surface.given.`

1. `file` Name of the file containing climate forcing fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time
3. `smb_max` (91000 kg m-2 year-1) Maximum climatic mass balance value (used to check input data). Corresponds to 100 m/year ice equivalent.

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 .

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

$$\frac{\partial m_{\text{abl}}}{\partial h} = -m_{\min}/(h_{\text{ELA}} - h_{\min})$$

and

$$\frac{\partial m_{\text{acl}}}{\partial h} = m_{\max}/(h_{\max} - h_{\text{ELA}})$$

be the mass balance gradient in the ablation and in the accumulation area, respectively. Then

$$m(x, y) = \begin{cases} m_{\min}, & h(x, y) \leq h_{\min}, \\ \frac{\partial m_{\text{abl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\min} < h(x, y) \leq h_{\text{ELA}}, \\ \frac{\partial m_{\text{acl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\text{ELA}} < h(x, y) \leq h_{\max}, \\ m_{\max}, & h_{\max} < h(x, y). \end{cases}$$

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

$$m(x, y) = \begin{cases} m^*_{\min}, & h(x, y) \leq h_{\min}, \\ \frac{\partial m_{\text{abl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\min} < h(x, y) \leq h_{\text{ELA}}, \\ \frac{\partial m_{\text{acl}}}{\partial h} (h(x, y) - h_{\text{ELA}}), & h_{\text{ELA}} < h(x, y) \leq h_{\max}, \\ m^*_{\max}, & h_{\max} < h(x, y). \end{cases}$$

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 [18] and EISMINT-Greenland intercomparison (see [7]).

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 `surface.pdd.air_temp_all_precip_as_rain` (default of 2°C) is interpreted as rain; all precipitation during periods with air temperatures below `surface.pdd.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 configuration parameters with the prefix `surface.pdd.std_dev.:`:

1. `file` The name of the file to read `air_temp_sd` (standard deviation of air temperature) from.
2. `lapse_lat_base` (72 *degree_north*) standard deviation is a function of latitude, with value `surface.pdd.std_dev.value` at this latitude; this value is only active if `surface.pdd.std_dev.lapse_lat_rate` is nonzero
3. `lapse_lat_rate` (0 *Kelvin/degree_north*) standard deviation is a function of latitude, with rate of change with respect to latitude given by this constant
4. `param_a` (-0.15) Parameter *a* in $\Sigma = aT + b$, with *T* in degrees C. Used only if `surface.pdd.std_dev.use_param` is set to yes.
5. `param_b` (0.66 *Kelvin*) Parameter *b* in $\Sigma = aT + b$, with *T* in degrees C. Used only if `surface.pdd.std_dev.use_param` is set to yes.
6. `periodic` (no) If true, interpret `air_temp_sd` read from `surface.pdd.std_dev.file` as periodic in time
7. `use_param` (no) 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`.
8. `value` (5 *Kelvin*) standard deviation of daily temp variation; = EISMINT-Greenland value [7]

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 configuration parameter `surface.pdd.std_dev.value`. The default value is 5.0 degrees [7]. However, this approach is not recommended as it induces significant errors in modeled surface mass balance in both ice-covered and ice-free regions [19], [20].

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 [21].

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 [18]. 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_method random_process`. 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_method repeatable_random_process` instead.

By default, the computation summarized in Fig. 4.3 is performed every week. (This frequency is controlled by the parameter `surface.pdd.max_evals_per_year`.) 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 configuration flag `surface.pdd.refreeze_ice_melt`.

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 (option `-pdd_std_dev`) is 2.53 degrees when `-pdd_fausto` is set [1]. See also configuration parameters with the prefix `surface.pdd.fausto.:`

1. `T_c` (272.15 Kelvin) = $-1 + 273.15$; for formula (6) in [1]
2. `T_w` (283.15 Kelvin) = $10 + 273.15$; for formula (6) in [1]
3. `beta_ice_c` (0.015 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
4. `beta_ice_w` (0.007 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
5. `beta_snow_c` (0.003 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
6. `beta_snow_w` (0.003 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
7. `enabled` (false) Set PDD parameters using formulas (6) and (7) in [1]
8. `latitude_beta_w` (72 degree_north) latitude below which to use warm case, in formula (6) in [1]

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 `time_stepping.hit_multiples` is recommended: set it to the length of the climate data period in years.

This model provides the following scalar:

- `surface_accumulation_rate`
- `surface_melt_rate`
- `surface_runoff_rate`

and these 2D diagnostic quantities (averaged over reporting intervals; positive flux corresponds to ice gain):

- `surface_accumulation_flux`
- `surface_melt_flux`
- `surface_runoff_flux`

This makes it easy to compare the surface mass balance computed by the model to its individual components:

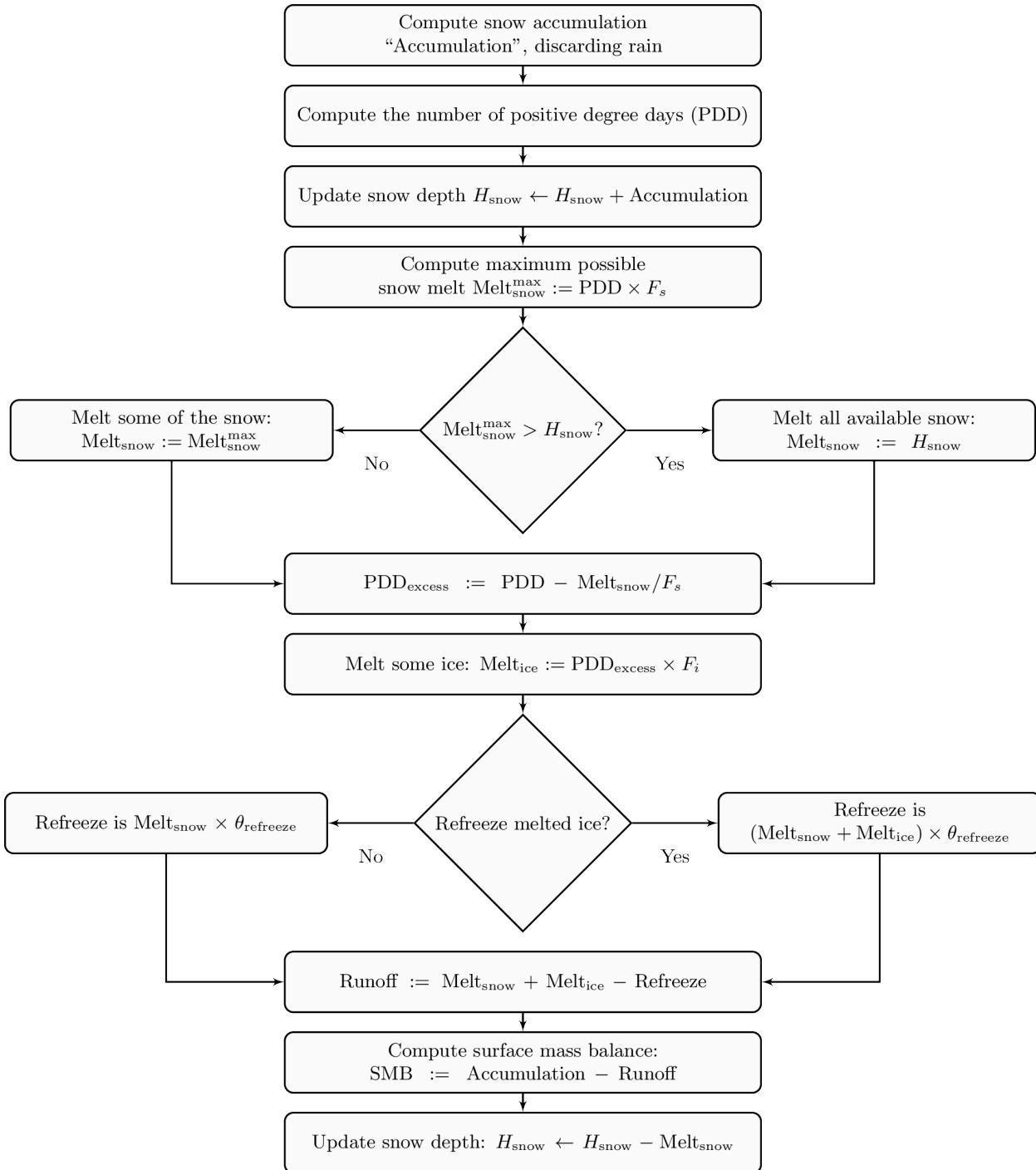


Fig. 4.3: PISM's positive degree day model. F_s and F_i are PDD factors for snow and ice, respectively; θ_{refreeze} is the refreeze fraction.

```
SMB = surface_accumulation_flux - surface_runoff_flux
```

Parameters

Prefix: `surface.pdd..`

1. `air_temp_all_precip_as_rain` (275.15 Kelvin) 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
2. `air_temp_all_precip_as_snow` (273.15 Kelvin) threshold temperature below which all precipitation is snow
3. `balance_year_start_day` (274 ordinal day number) day of year for October 1st, beginning of the balance year in northern latitudes.
4. `factor_ice` (0.00879121 meter / (Kelvin day)) EISMINT-Greenland value [7]; = (8 mm liquid-water-equivalent) / (pos degree day)
5. `factor_snow` (0.0032967 meter / (Kelvin day)) EISMINT-Greenland value [7]; = (3 mm liquid-water-equivalent) / (pos degree day)
6. `fausto.T_c` (272.15 Kelvin) = -1 + 273.15; for formula (6) in [1]
7. `fausto.T_w` (283.15 Kelvin) = 10 + 273.15; for formula (6) in [1]
8. `fausto.bet_ice_c` (0.015 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
9. `fausto.bet_ice_w` (0.007 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
10. `fausto.bet_snow_c` (0.003 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
11. `fausto.bet_snow_w` (0.003 meter / (Kelvin day)) water-equivalent thickness; for formula (6) in [1]
12. `fausto.enabled` (false) Set PDD parameters using formulas (6) and (7) in [1]
13. `fausto.latitude_beta_w` (72 degree_north) latitude below which to use warm case, in formula (6) in [1]
14. `firn_compaction_to_accumulation_ratio` (0.75) How much firn as a fraction of accumulation is turned into ice
15. `firn_depth_file` The name of the file to read the firn_depth from.
16. `interpret_precip_as_snow` (no) Interpret precipitation as snow fall.
17. `max_evals_per_year` (52) 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.value`
18. `method` (expectation_integral) PDD implementation method
19. `positive_threshold_temp` (273.15 Kelvin) temperature used to determine meaning of “positive” degree day
20. `refreeze` (0.6) EISMINT-Greenland value [7]
21. `refreeze_ice_melt` (yes) If set to “yes”, refreeze `surface.pdd.refreeze` fraction of melted ice, otherwise all of the melted ice runs off.
22. `std_dev.file` The name of the file to read `air_temp_sd` (standard deviation of air temperature) from.
23. `std_dev.lapse_lat_base` (72 degree_north) standard deviation is a function of latitude, with value `surface.pdd.std_dev.value` at this latitude; this value is only active if `surface.pdd.std_dev.lapse_lat_rate` is nonzero

24. `std_dev.lapse_lat_rate` (*0 Kelvin / degree_north*) standard deviation is a function of latitude, with rate of change with respect to latitude given by this constant
25. `std_dev.param_a` (-0.15) Parameter a in $\Sigma = aT + b$, with T in degrees C. Used only if `surface.pdd.std_dev.use_param` is set to yes.
26. `std_dev.param_b` (0.66 *Kelvin*) Parameter b in $\Sigma = aT + b$, with T in degrees C. Used only if `surface.pdd.std_dev.use_param` is set to yes.
27. `std_dev.periodic` (no) If true, interpret `air_temp_sd` read from `surface.pdd.std_dev.file` as periodic in time
28. `std_dev.use_param` (no) 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`.
29. `std_dev.value` (5 *Kelvin*) standard deviation of daily temp variation; = EISMINT-Greenland value [7]

4.4.5 PIK

Options `-surface pik`

Variables `climatic_mass_balance kg/(m2s)`, `lat` (latitude), (degrees north)

C++ class `pism::surface::PIK`

This surface model component implements the setup used in [3]. 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 [3].

4.4.6 Scalar temperature offsets

Options `-surface ...,delta_T`

Variables `delta_T`

C++ class `pism::surface::Delta_T`

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.

Parameters

Prefix: `surface.delta_T`.

1. `file` Name of the file containing temperature offsets.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.4.7 Adjustments using modeled change in surface elevation

Options `-surface ... ,elevation_change`

Variables `surface_altitude` (CF standard name),

C++ class `pism::surface::LapseRates`

The `elevation_change` modifier adjusts ice-surface temperature and surface mass balance using modeled changes in surface elevation relative to a reference elevation read from a file.

The surface temperature is modified using an elevation lapse rate $\gamma_T = \text{surface.elevation_change.temperature_lapse_rate}$. Here

$$\gamma_T = -\frac{dT}{dz}.$$

Two methods of adjusting the SMB are available:

- Scaling using an exponential factor

$$\text{SMB} = \text{SMB}_{\text{input}} \cdot \exp(C \cdot \Delta T),$$

where $C = \text{surface.elevation_change.smb.exp_factor}$ and ΔT is the temperature difference produced by applying `surface.elevation_change.temperature_lapse_rate`.

This mechanism increases the SMB by $100(\exp(C) - 1)$ percent for each degree of temperature increase.

To use this method, set `-smb_adjustment scale`.

- Elevation lapse rate for the SMB

$$\text{SMB} = \text{SMB}_{\text{input}} - \Delta h \cdot \gamma_M,$$

where $\gamma_M = \text{surface.elevation_change.smb.lapse_rate}$ and Δh is the difference between modeled and reference surface elevations.

To use this method, set `-smb_adjustment shift`.

Parameters

Prefix: `surface.elevation_change..`

1. `file` Name of the file containing the reference surface elevation field (variable `usurf`).
2. `periodic` (no) If true, interpret forcing data as periodic in time
3. `smb.exp_factor` (0 Kelvin-1) Exponential for the surface mass balance.
4. `smb.lapse_rate` (0 (m/year)/km) Lapse rate for the surface mass balance.
5. `smb.method` (shift) Choose the SMB adjustment method. `scale`: use temperature-change-dependent scaling factor. `shift`: use the SMB lapse rate.
6. `temperature_lapse_rate` (0 K/km) Lapse rate for the temperature at the top of the ice.

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.

See also `-atmosphere ... ,anomaly` (section [Using climate data anomalies](#)), which is similar but applies anomalies at the atmosphere level.

Parameters

Prefix: `surface.anomaly`.

1. `file` Name of the file containing climate forcing fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time

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` 365-day “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.)

Parameters

Prefix: `surface.cache`.

1. `update_interval` (10 365days) Update interval (in 365-day years) for the `-surface cache` modifier.

4.4.11 Preventing grounding line retreat

Options `-surface ... ,no_gl_retreat`

C++ class `pism::surface::NoGLRetreat`

This modifier adjust the surface mass balance to prevent the retreat of the grounding line. See [Till friction angle optimization](#) for an application.

Note:

- This modifier *adds mass* in violation of mass conservation. Save the diagnostic `no_gl_retreat_smb_adjustment` to get an idea about the amount added. Note, though, that this is an imperfect measure: it includes mass added to maintain non-negativity of ice thickness.
- We assume that the sea level and the bed elevation remain constant throughout the simulation.
- This does *not* prevent grounding line retreat caused by the thinning of the ice due to the melt at the base. Set `geometry.update.use_basal_melt_rate` to “false” to ensure that basal melt has no effect on the position of the grounding line

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*
 - *Precipitation scaling*
 - *Precipitation correction using scalar temperature offsets*
 - *Adjustments using modeled change in surface elevation*
 - *Using climate data anomalies*
 - *Orographic precipitation*

4.5.1 Reading boundary conditions from a file

Options `-atmosphere given`

Variables `air_temp, precipitation kg/(m2s)`

C++ class `pism::atmosphere::Given`

See also *Reading top-surface boundary conditions from a file*

Note: This is the default choice.

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*).

Parameters

Prefix: `atmosphere.given.`

1. `file` Name of the file containing climate forcing fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.5.2 Cosine yearly cycle

Options `-atmosphere yearly_cycle`

Variables `air_temp_mean_annual`, `air_temp_mean_july`, `precipitation` $kg/(m^2 s)$
`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 command-line option `-atmosphere_yearly_cycle_file`. 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 $A(\text{time}) \equiv 1$.

Parameters

Prefix: `atmosphere.yearly_cycle.`

1. `file` Name of the file containing mean annual and mean July temperatures (`air_temp_mean_annual` and `air_temp_mean_summer`) and the `precipitation` field.
2. `scaling.file` Name of the file containing amplitude scaling (`amplitude_scaling`) for the near-surface air temperature.
3. `scaling.periodic` (no) If true, interpret forcing data as periodic in time

4.5.3 SeaRISE-Greenland

Options `-atmosphere searise_greenland`

Variables `lon, lat, precipitation` $kg/(m^2 s)$

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 prefix `atmosphere.fausto_air_temp`:

1. c_{ma} (-0.7189 Kelvin / degree_north) latitude-dependence coefficient for formula (1) in [1]
2. c_{mj} (-0.1585 Kelvin / degree_north) latitude-dependence coefficient for formula (2) in [1]
3. d_{ma} (314.98 Kelvin) 41.83+273.15; base temperature for formula (1) in [1]
4. d_{mj} (287.85 Kelvin) = 14.70+273.15; base temperature for formula (2) in [1]
5. γ_{ma} (-0.006309 Kelvin / meter) = -6.309 / 1km; mean slope lapse rate for formula (1) in [1]
6. γ_{mj} (-0.005426 Kelvin / meter) = -5.426 / 1km; mean slope lapse rate for formula (2) in [1]
7. κ_{ma} (0.0672 Kelvin / degree_west) longitude-dependence coefficient for formula (1) in [1]
8. κ_{mj} (0.0518 Kelvin / degree_west) longitude-dependence coefficient for formula (2) in [1]
9. $summer_peak_day$ (196 ordinal day number) day of year for July 15; used in corrected formula (4) in [1]

See [Precipitation correction using scalar temperature offsets](#) for an implementation of the SeaRISE-Greenland formula for precipitation adjustment using air temperature offsets relative to 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, lon, precipitation

C++ class pism::atmosphere::PIK

This model component reads a time-independent precipitation field from an input file specified by `atmosphere.pik.file` and computes near-surface air temperature using a parameterization selected using `atmosphere.pik.parameterization`.

Note:

- Parameterizations implemented in this model are appropriate for the **Antarctic** ice sheet.
 - All parameterizations except for the first one implement a cosine annual cycle of the near-surface air temperature.
-

Table 4.1: Near-surface air temperature parameterizations

Keyword	Description
martin (default)	Uses equation (1) from [3] to parameterize mean annual temperature with <i>no seasonal variation in temperature</i> .
huybrechts_dewolde	Mean annual and mean summer temperatures are computed using parameterizations for the Antarctic ice sheet described in [4] (equations C1 and C2).
martin_huybrechts_dewolde	Hybrid of the two above: mean annual temperature as in [3] with the amplitude of the annual cycle from [4].
era_interim	Mean annual and mean summer temperatures use parameterizations based on multiple regression analysis of ERA INTERIM data.
era_interim_sin	Mean annual and mean summer temperatures use parameterizations based on multiple regression analysis of ERA INTERIM data with a $\sin(\text{latitude})$ dependence.
era_interim_lon	Mean annual and mean summer temperatures use parameterizations based on multiple regression analysis of ERA INTERIM data with a $\cos(\text{longitude})$ dependence.

See [PIK](#) for a surface model that implements the `martin` choice as a parameterization of the ice temperature at its top surface.

4.5.5 One weather station

Options `-atmosphere one_station -atmosphere_one_station_file`

Variables `air_temp [Kelvin], precipitation kg/(m2s)`

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 `atmosphere.one_station.file` 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 `elevation_change` (see section *Adjustments using modeled change in surface elevation*) 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.

Please make sure that `delta_T` has the units of “Kelvin”.

Parameters

Prefix: `atmosphere.delta_T`.

1. `file` Name of the file containing temperature offsets.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.5.7 Scalar precipitation offsets

Options `-atmosphere ...,delta_P`

Variables `delta_P kg/(m2s)`

C++ class `pism::atmosphere::Delta_P`

This modifier applies scalar time-dependent precipitation offsets to the output of an atmosphere model.

Parameters

Prefix: `atmosphere.delta_P`.

1. `file` Name of the file containing scalar precipitation offsets.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.5.8 Precipitation scaling

Options `-atmosphere ... ,frac_P`

Variables `frac_P [1]`

C++ class `pism::atmosphere::Frac_P`

This modifier scales precipitation output of an atmosphere model using a time-dependent precipitation fraction, with a value of *one* corresponding to no change in precipitation. It supports both 1D (scalar) and 2D (spatially-variable) factors.

If the variable `frac_P` in the input file (see `atmosphere.frac_P.file`) depends on time only it is used as a time-dependent constant-in-space scaling factor.

If the variable `frac_P` has more than one dimension PISM tries to use it as a time-and-space-dependent scaling factor.

Parameters

Prefix: `atmosphere.frac_P`.

1. `file` Name of the file containing scalar precipitation scaling.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.5.9 Precipitation correction using scalar temperature offsets

Options `-atmosphere ... ,precip_scaling`

Variables `delta_T [degrees Kelvin]`

C++ class `pism::atmosphere::PrecipitationScaling`

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](#).

This mechanism increases precipitation by $100(\exp(C) - 1)$ percent for each degree of temperature increase, where $C = \text{atmosphere.precip_exponential_factor_for_temperature}$.

Parameters

Prefix: `atmosphere.precip_scaling`.

1. `file` Name of the file containing temperature offsets to use for a precipitation correction.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.5.10 Adjustments using modeled change in surface elevation

Options `-atmosphere ... ,elevation_change`

Variables `surface_altitude` (CF standard name)

C++ class `pism::atmosphere::ElevationChange`

The `elevation_change` modifier adjusts air temperature and precipitation using modeled changes in surface elevation relative to a reference elevation read from a file.

The near-surface air temperature is modified using an elevation lapse rate $\gamma_T = \text{atmosphere.elevation_change.temperature_lapse_rate}$. Here

$$\gamma_T = -\frac{dT}{dz}.$$

Warning: Some atmosphere models (`PIK`, for example) use elevation-dependent near-surface air temperature parameterizations that include an elevation lapse rate.

In most cases one should not combine such a temperature parameterization with an additional elevation lapse rate for temperature.

Two methods of adjusting precipitation are available:

- Scaling using an exponential factor

$$P = P_{\text{input}} \cdot \exp(C \cdot \Delta T),$$

where $C = \text{atmosphere.precip_exponential_factor_for_temperature}$ and ΔT is the temperature difference produced by applying the lapse rate `atmosphere.elevation_change.precipitation.temp_lapse_rate`.

This mechanism increases precipitation by $100(\exp(C) - 1)$ percent for each degree of temperature increase.

To use this method, set `-precip_adjustment scale`.

- Elevation lapse rate for precipitation

$$P = P_{\text{input}} - \Delta h \cdot \gamma_P,$$

where $\gamma_P = \text{atmosphere.elevation_change.precipitation.lapse_rate}$ and Δh is the difference between modeled and reference surface elevations.

To use this method, set `-smb_adjustment shift`.

Parameters

Prefix: `atmosphere.elevation_change`.

1. `file` Name of the file containing the reference surface elevation field (variable `usurf`).
2. `periodic` (no) If true, interpret forcing data as periodic in time
3. `precipitation.lapse_rate` (0 ($kg\ m^{-2}/year$) / km) Elevation lapse rate for the precipitation
4. `precipitation.method` (shift) Choose the precipitation adjustment method. `scale`: use temperature-change-dependent scaling factor. `shift`: use the precipitation lapse rate.
5. `precipitation.temp_lapse_rate` (0 Kelvin / km) Elevation lapse rate for the surface temperature used to compute ΔT in the precipitation scaling factor $\exp(C \cdot \Delta T)$
6. `temperature_lapse_rate` (0 Kelvin / km) Elevation lapse rate for the surface temperature

The file specified using `atmosphere.elevation_change.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.

4.5.11 Using climate data anomalies

Options `-atmosphere ... ,anomaly`
Variables `air_temp_anomaly, precipitation_anomaly kg/(m2s)`
C++ class `pism::atmosphere::Anomaly`

This modifier implements a spatially-variable version of `-atmosphere ... ,delta_T,delta_P`.

Parameters

Prefix: `atmosphere.anomaly`.

1. `file` Name of the file containing climate forcing fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time

See also to `-surface ... ,anomaly` (section *Using climate data anomalies*), which is similar but applies anomalies at the surface level.

4.5.12 Orographic precipitation

Options `-atmosphere ... ,orographic_precipitation`
Variables None
C++ class `pism::atmosphere::OrographicPrecipitation`

This modifier implements the linear orographic precipitation model described in [5] with a modification incorporating the influence of the Coriolis force from [6].

We compute the Fourier transform of the precipitation field using the formula below (see equation 49 in [5] or equation 3 in [6]).

$$\hat{P}_{LT}(k, l) = \frac{C_w i\sigma \hat{h}(k, l)}{(1 - imH_w)(1 + i\sigma\tau_c)(1 + i\sigma\tau_f)}, \quad (4.1)$$

where h is the surface elevation, $C_w = \rho_{S_{\text{ref}}} \Gamma_m / \gamma$ relates the condensation rate to vertical motion (see the appendix of [5]), m is the vertical wavenumber (see equation 6 in [6]), and σ is the intrinsic frequency. The rest of the constants are defined below.

The spatial pattern of precipitation is recovered using an inverse Fourier transform followed by post-processing:

$$P = \max(P_{\text{pre}} + P_{\text{LT}}, 0) \cdot S + P_{\text{post}}. \quad (4.2)$$

Note:

- Discontinuities in the surface gradient (e.g. at ice margins) may cause oscillations in the computed precipitation field (probably due to the Gibbs phenomenon). To address this our implementation includes the ability to smooth the surface topography using a Gaussian filter. Set `atmosphere.orographic_precipitation.smoothing_standard_deviation` to a positive number to enable smoothing. Values around Δx appear to be effective.
- The spectral method used to implement this model requires that the input (i.e. surface elevation h) is periodic in x and y . To simulate periodic h the implementation uses an extended grid (see `atmosphere.orographic_precipitation.grid_size_factor`) and pads modeled surface elevation with zeros.

It is worth noting that the resulting precipitation field P_{LT} is also periodic in x and y on the *extended* grid. The appropriate size of this extended grid may differ depending on the spatial scale of the domain and values of model parameters.

It is implemented as a “modifier” that overrides the precipitation field provided by an input model. Use it with a model providing air temperatures to get a complete model. For example, `-atmosphere yearly_cycle orographic_precipitation ...` would use the annual temperature cycle from `yearly_cycle` combined with precipitation computed using this model.

The only spatially-variable input of this model is the surface elevation (h above) modeled by PISM.

Parameters

Prefix: `atmosphere.orographic_precipitation`.

1. `background_precip_post` (0 mm/hr) Background precipitation P_{post} added after the truncation.
2. `background_precip_pre` (0 mm/hr) Background precipitation P_{pre} added before the truncation.
3. `conversion_time` (1000 s) Cloud conversion time τ_c
4. `coriolis_latitude` (0 degrees_N) Latitude used to compute Coriolis force
5. `fallout_time` (1000 s) Fallout time τ_f
6. `grid_size_factor` (2) The size of the extended grid is $(Z^*(\text{grid.Mx} - 1) + 1, Z^*(\text{grid.My} - 1) + 1)$ where Z is given by this parameter.
7. `lapse_rate` (-5.8 K/km) Lapse rate γ
8. `moist_adiabatic_lapse_rate` (-6.5 K/km) Moist adiabatic lapse rate Γ_m
9. `moist_stability_frequency` (0.05 1/s) Moist stability frequency N_m
10. `reference_density` (0.0074 kg m-3) Reference density $\rho_{S_{\text{ref}}}$
11. `scale_factor` (1) Precipitation scaling factor S .
12. `smoothing_standard_deviation` (0 m) Standard deviation of the Gaussian filter used to smooth surface elevation or zero if disabled

13. `truncate` (true) Truncate precipitation at 0, disallowing negative precipitation values.
14. `water_vapor_scale_height` (2500 m) Water vapor scale height H_w
15. `wind_direction` (270 degrees) The direction the wind is coming from
16. `wind_speed` (10 m/s) The wind speed.

4.6 Ocean model components

PISM's 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*
 - *PICO*
 - *Scalar sea level offsets*
 - *Two-dimensional sea level offsets*
 - *Scalar sub-shelf temperature offsets*
 - *Scalar sub-shelf mass flux offsets*
 - *Scalar sub-shelf mass flux fraction offsets*
 - *Two-dimensional sub-shelf mass flux offsets*
 - *Scalar melange back pressure offsets*
 - *Melange back pressure as a fraction of pressure difference*
 - *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 controlled by `ocean.constant.melt_rate`.

4.6.2 Reading forcing data from a file

Options `-ocean given`

Variables `shelfbtemp Kelvin, shelfbmassflux kg/(m2s)`

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.

Variables `shelfbtemp` and `shelfbmassflux` may be time-dependent. (The `-ocean given` component is very similar to `-surface given` and `-atmosphere given`.)

Parameters

Prefix: `ocean.given.`

1. `file` Name of the file containing climate forcing fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time

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 [3]. The sub-shelf ice temperature is set to pressure-melting; the sub-shelf mass flux computation follows [10].

Parameters

Prefix: `ocean.pik_`

1. `melt_factor` (0.005) dimensionless tuning parameter in the `-ocean pik` ocean heat flux parameterization; see [3]

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 [11] and [12].

Inputs are potential temperature (variable `theta_ocean`) and salinity (variable `salinity_ocean`) read from a file. A constant salinity (see `constants.sea_water.salinity`) is used if the input file does not contain `salinity_ocean`.

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 [11] for details).

Parameters

Prefix: `ocean.th`.

1. `clip_salinity` (yes) Clip shelf base salinity so that it is in the range [4, 40] k/kg. See [11].
2. `file` Name of the file containing climate forcing fields.
3. `gamma_S` (5.05e-07 m s⁻¹) Turbulent salt transfer coefficient. See [11].
4. `gamma_T` (0.0001 m s⁻¹) Turbulent heat transfer coefficient. See [11].
5. `periodic` (no) If true, interpret forcing data as periodic in time

Note: If `ocean.th.clip_salinity` is set (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 [11].

Set `ocean.th.clip_salinity` to `false` if restricting salinity is not appropriate.

See *Three-equation ocean model (implementation details)* for implementation details.

4.6.5 PICO

Options `-ocean pico`

Variables `theta_ocean` (potential ocean temperature), [Kelvin],

`salinity_ocean` (salinity of the adjacent ocean), [g/kg],

`basins` (mask of large-scale ocean basins that ocean input is averaged over), [integer]

C++ class `pism::ocean::Pico`

The PICO model provides sub-shelf melt rates and temperatures consistent with the vertical overturning circulation in ice shelf cavities that drives the exchange with open ocean water masses. It is based on the ocean box model of [13] and includes a geometric approach which makes it applicable to ice shelves that evolve in two horizontal dimensions. For each ice shelf, PICO solves the box model equations describing the transport between coarse ocean boxes. It applies a boundary layer melt formulation [14], [11]. The overturning circulation is driven by the ice-pump [15]: melting at the ice-shelf base reduces the density of ambient water masses. Buoyant water rising along the shelf base draws in ocean water at depth, which flows across the continental shelf towards the deep grounding lines. The model captures this circulation by defining consecutive boxes following the flow within the ice shelf cavity, with the first box adjacent to the grounding line. The extents of the ocean boxes are computed adjusting to the evolving grounding lines and calving fronts. Open ocean properties in front of the shelf as well as the geometry of the shelf determine basal melt rate and basal temperature at each grid point.

The main equations reflect the

1. heat and salt balance for each ocean box in contact with the ice shelf base,
2. overturning flux driven by the density difference between open-ocean and grounding-line box,
3. boundary layer melt formulation.

The PICO model is described in detail in [16].

Inputs are potential temperature (variable `theta_ocean`), salinity (variable `salinity_ocean`) and ocean basin mask (variable `basins`). Variables `theta_ocean` and `salinity_ocean` may be time-dependent.

Forcing ocean temperature and salinity are taken from the water masses that occupy the sea floor in front of the ice shelves, which extends down to a specified continental shelf depth (see `ocean.pico.continental_shelf_depth`). These water masses are transported by the overturning circulation into the ice shelf cavity and towards the grounding line. The basin mask defines regions of similar, large-scale ocean conditions; each region is marked with a distinct positive integer. In PICO, ocean input temperature and salinity are averaged on the continental shelf within each basins. For each ice shelf, the input values of the overturning circulation are calculated as an area-weighted average over all basins that intersect the ice shelf. Only those basins are considered in the average, in which the ice shelf has in fact a connection to the ocean. Large ice shelves, that cover across two basins, that do not share an ocean boundary, are considered as two separate ice shelves with individual ocean inputs. If ocean input parameters cannot be identified, standard values are used (**Warning:** this could strongly influence melt rates computed by PICO). In regions where the PICO geometry cannot be identified, [10] is applied.

Parameters

Prefix: `ocean.pico`.

1. `continental_shelf_depth` (-800 meters) Specifies the depth up to which oceanic input temperatures and salinities are averaged over the continental shelf areas in front of the ice shelf cavities.
2. `exclude_ice_rises` (yes) If set to true, grounding lines of ice rises are excluded in the geometrical routines that determine the ocean boxes; using this option is recommended.
3. `file` Specifies the NetCDF file containing potential temperature (`theta_ocean`), salinity (`salinity_ocean`) and ocean basins (`basins`).

4. `heat_exchange_coefficient` ($2e-05 \text{ meters second}^{-1}$) Sets the coefficient for turbulent heat exchange from the ambient ocean across the boundary layer beneath the ice shelf base.
5. `maximum_ice_rise_area` (100000 km^2) Specifies an area threshold that separates ice rises from continental regions.
6. `number_of_boxes` (5) For each ice shelf the number of ocean boxes is determined by interpolating between 1 and `number_of_boxes` depending on its size and geometry such that larger ice shelves are resolved with more boxes; a value of 5 is suitable for the Antarctic setup.
7. `overturning_coefficient` ($1e+06 \text{ meters}^6 \text{ seconds}^{-1} \text{ kg}^{-1}$) Sets the overturning strength coefficient.
8. `periodic` (no) If true, interpret forcing data as periodic in time

4.6.6 Scalar sea level offsets

Options `-sea_level ... ,delta_sl`

Variables `delta_SL` (meters)

C++ class `pism::ocean::sea_level::Delta_SL`

The `delta_sl` modifier implements sea level forcing using scalar offsets.

Parameters

Prefix: `ocean.delta_sl`.

1. `file` Name of the file containing sea level offsets.
2. `periodic` (no) If true, interpret forcing data as periodic in time
3. `2d.file` Name of the file containing climate forcing fields.
4. `2d.periodic` (no) If true, interpret forcing data as periodic in time

4.6.7 Two-dimensional sea level offsets

Options `-sea_level ... ,delta_sl_2d`

Variables `delta_SL` (meters)

C++ class `pism::ocean::sea_level::Delta_SL_2D`

The `delta_sl` modifier implements sea level forcing using time-dependent and spatially-variable offsets.

Parameters

Prefix: `ocean.delta_sl_2d`.

1. `file` Name of the file containing climate forcing fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.6.8 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.

Parameters

Prefix: ocean.delta_T.

1. *file* Name of the file containing temperature offsets.
2. *periodic* (no) If true, interpret forcing data as periodic in time

4.6.9 Scalar sub-shelf mass flux offsets

Options -ocean ... ,delta_SMB
Variables delta_SMB kg/(m²s)
C++ class pism::ocean::Delta_SMB

This modifier implements forcing using sub-shelf mass flux (melt rate) offsets.

Parameters

Prefix: ocean.delta_mass_flux.

1. *file* Name of the file containing sub-shelf mass flux offsets.
2. *periodic* (no) If true, interpret forcing data as periodic in time

4.6.10 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.

Parameters

Prefix: ocean.fractional_mass_flux.

1. *file* Name of the file containing sub-shelf mass flux scaling.
2. *periodic* (no) If true, interpret forcing data as periodic in time

4.6.11 Two-dimensional sub-shelf mass flux offsets

Options `-ocean ... ,anomaly`

Variables `shelf_base_mass_flux_anomaly kg/(m2s)`

C++ class `pism::ocean::Anomaly`

This modifier implements a spatially-variable version of `-ocean ... ,delta_SMB` which applies time-dependent shelf base mass flux anomalies, as used for initMIP or LARMIP model intercomparisons.

See also to `-atmosphere ... ,anomaly` or `-surface ... ,anomaly` (section [Using climate data anomalies](#)) which is similar, but applies anomalies at the atmosphere or surface level, respectively.

Parameters

Prefix: `ocean.anomaly`.

1. `file` Name of the file containing shelf basal mass flux offset fields.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.6.12 Scalar melange back pressure offsets

Options `-ocean ... ,delta_MBP`

Variables `delta_MBP [Pascal]`

C++ class `pism::ocean::Delta_MBP`

The scalar time-dependent variable `delta_MBP` (units: Pascal) has the meaning of the melange back pressure σ_b in [17]. It is assumed that σ_b is applied over the thickness of melange h specified using `ocean.delta_MBP.melange_thickness`.

To convert to the average pressure over the ice front thickness, we compute

$$\bar{p}_{\text{melange}} = \frac{\sigma_b \cdot h}{H}, \quad (4.3)$$

where H is ice thickness.

See [Modeling melange back-pressure](#) for details.

Parameters

Prefix: `ocean.delta_MBP`.

1. `file` Name of the file containing melange back-pressure offsets
2. `melange_thickness` (100 meters) Melange thickness (assumed to be constant in space and time)
3. `periodic` (no) If true, interpret forcing data as periodic in time

4.6.13 Melange back pressure as a fraction of pressure difference

Options `-ocean ... ,frac_MBP`

Variables `frac_MBP`

C++ class `pism::ocean::Frac_MBP`

This modifier implements forcing using melange back pressure fraction (scaling).

Here we assume that the total vertically-averaged back pressure at an ice margin cannot exceed the vertically-averaged ice pressure at the same location:

$$\bar{p}_{\text{water}} + \bar{p}_{\text{melange}} \leq \bar{p}_{\text{ice}}, \text{ or}$$

$$\bar{p}_{\text{melange}} \leq \bar{p}_{\text{ice}} - \bar{p}_{\text{water}}.$$

We introduce $\lambda \in [0, 1]$ such that

$$\bar{p}_{\text{melange}} = \lambda(\bar{p}_{\text{ice}} - \bar{p}_{\text{water}}).$$

The scalar time-dependent variable `frac_MBP` should take on values between 0 and 1 and has the meaning of λ above.

Please see [Modeling melange back-pressure](#) for details.

Parameters

Prefix: `ocean.fraction_MBP`.

1. `file` Name of the file containing melange back-pressure scaling.
2. `periodic` (no) If true, interpret forcing data as periodic in time

4.6.14 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` 365-day “years”. A time-step of 1 year (respecting the chosen calendar) 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.)

Parameters

Prefix: `ocean.cache`.

1. `update_interval` (10 365days) update interval of the `cache` ocean modifier

TECHNICAL NOTES

5.1 Release checklist

1. Run `make manual_linkcheck` and fix any broken links in the manual.
2. Run `make` in the `doc/sphinx` directory to update lists of diagnostics and configuration parameters.
3. Run `make` in the `doc` directory to update funding sources.
4. Create a “pre-release” branch starting from the “`dev`” branch and remove code that should not be a part of the release.
5. Set `PISM_BRANCH` in `CMakeLists.txt` to “`stable`”.
6. Update `version`, `release`, and `copyright` in `doc/sphinx/conf.py`.
7. Update `CHANGES.rst`.
8. Tag.

```
git tag -a v1.X -m "The v1.X release. See CHANGES.rst for the list of changes since v1.X-1."
```

9. Push.

```
git push -u origin HEAD
```

10. Push tags.

```
git push --tags
```

11. Re-build docs.

```
make manual_html manual_pdf browser.tgz pismpython_docs
```

12. Upload these docs.

13. Write a news item for `pism.github.io`.

14. Update the current PISM version on `pism.github.io`.

15. Send an e-mail to CRYOLIST.

16. Tell more people, if desired.

17. Create a new “release” on <https://github.com/pism/pism/releases>

5.2 CF standard names used by PISM

5.2.1 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)
<code>bedrock_altitude</code>	m
<code>floating_ice_sheet_area_fraction</code>	1
<code>grounded_ice_sheet_area_fraction</code>	1
<code>land_ice_area_fraction</code>	1
<code>land_ice_basal_melt_rate</code>	m s^{-1}
<code>land_ice_basal_temperature</code>	K
<code>land_ice_basal_upward_velocity</code>	m s^{-1}
<code>land_ice_basal_x_velocity</code>	m s^{-1}
<code>land_ice_basal_y_velocity</code>	m s^{-1}
<code>land_ice_calving_rate</code>	m s^{-1}
<code>land_ice_lwe_basal_melt_rate</code>	m s^{-1}
<code>land_ice_lwe_calving_rate</code>	m s^{-1}
<code>land_ice_lwe_surface_specific_mass_balance</code>	m s^{-1}
<code>land_ice_sigma_coordinate</code>	1
<code>land_ice_specific_mass_flux_due_to_calving_and_ice_front_melting</code>	$\text{kg m}^{-2} \text{s}^{-1}$
<code>land_ice_surface_specific_mass_balance_flux</code>	$\text{kg m}^{-2} \text{s}^{-1}$
<code>land_ice_surface_upward_velocity</code>	m s^{-1}
<code>land_ice_surface_x_velocity</code>	m s^{-1}
<code>land_ice_surface_y_velocity</code>	m s^{-1}
<code>land_ice_temperature</code>	K
<code>land_ice_thickness</code>	m
<code>land_ice_vertical_mean_x_velocity</code>	m s^{-1}
<code>land_ice_vertical_mean_y_velocity</code>	m s^{-1}
<code>land_ice_x_velocity</code>	m s^{-1}
<code>land_ice_y_velocity</code>	m s^{-1}
<code>latitude</code>	degree_north
<code>longitude</code>	degree_east
<code>magnitude_of_land_ice_basal_drag</code>	Pa
<code>projection_x_coordinate</code>	m

continues on next page

Table 5.1 – continued from previous page

CF standard name	Canonical units (SI)
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

5.2.2 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

5.2.3 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`.

5.3 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.

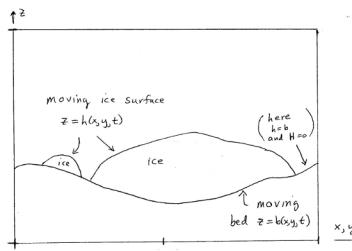


Fig. 5.1: The ice fluid domain evolves, with both the upper and lower surfaces in motion with respect to the geoid.

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.

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.

Note: FIXME: This figure should show the floating case too, and bedrock.”

In PISM the computational domain (region)

$$\mathcal{R} = \{(x, y, s) \mid -L_x \leq x \leq L_x, -L_y \leq y \leq L_y, -Lb_z \leq s \leq L_z\}$$

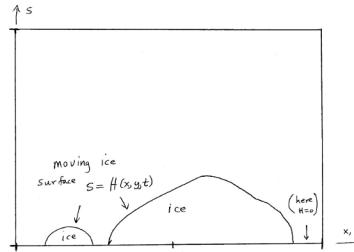


Fig. 5.2: In (x, y, s) space the ice fluid domain only has an upper surface which moves, $s = H(x, y, t)$. Compare to Fig. 5.1.

is divided into a three-dimensional grid. See [IceGrid](#).

The change of variable $z \rightarrow s$ used here *is not* the [169] 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 [BOMBPROOF, PISM's numerical scheme for conservation of energy](#) 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 `welsurf`, and its basal value as `welbase`. The relative vertical velocity \tilde{w} is written to the NetCDF output file as `wvel_rel`.

5.4 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*

5.4.1 An explanation

If the bed elevation `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 *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.

5.4.2 Theory

The theory is in Christian Schoof’s (2003) *The effect of basal topography on ice sheet dynamics* [83]. 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 *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

$$D_{SIA} = \Gamma H^{n+2} |\nabla h|^{n-1} \tag{5.1}$$

where $\Gamma = 2A(\rho g)^n/(n+2)$ (c.f. details in [33]).

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:

$$b_s(x_1, x_2) = \int b_0(x_1 + \xi_1, x_2 + \xi_2) d\xi_1 d\xi_2$$

where the slashed integral symbol is defined as

$$\int F(\xi_1, \xi_2) d\xi_1 d\xi_2 = \frac{1}{(2\lambda_1)(2\lambda_2)} \int_{-\lambda_1}^{\lambda_1} \int_{-\lambda_2}^{\lambda_2} F(\xi_1, \xi_2) d\xi_1 d\xi_2.$$

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:

$$\int \tilde{b}(x_1, x_2, \xi_1, \xi_2) d\xi_1 d\xi_2 = 0.$$

The result of Schoof’s scaling arguments ([83], 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

$$\theta(h, x_1, x_2) = \left[\int \left(1 - \frac{\tilde{b}(x_1, x_2, \xi_1, \xi_2)}{H} \right)^{-(n+2)/n} d\xi_1 d\xi_2 \right]^{-n} \quad (5.2)$$

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 [83] by a Jensen’s inequality argument. (See also the convexity argument at the bottom of this page.)

5.4.3 Practical application, and Taylor approximation

The above formulas already reflect the recommendations Schoof gives on how to apply his formulas ([83], subsection 4.2). The rest of this page is devoted to how the class `stressbalance::BedSmoothen` 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 (set `stress_balance.sia.bed_smoothen.range` to change this value).

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. Padé 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

$$\omega(f, z) = \int (1 - f(\xi)z)^{-k} d\xi$$

where f is some function and z a scalar. Then

$$\begin{aligned} \omega(f, z) &= \int (1 - f(\xi)z)^{-k} d\xi = \int P_4(f(\xi)z) d\xi + O((\max |f(\xi)|^5 |z|^5)) \\ &\approx 1 + kz \int f(\xi) d\xi + \frac{k(k+1)}{2}z^2 \int f(\xi)^2 d\xi + \dots + \frac{k(k+1)(k+2)(k+3)}{4!}z^5 \int f(\xi)^4 d\xi. \end{aligned}$$

Now, θ can be written

$$\theta(h, x_1, x_2) = [\omega(\tilde{b}(x_1, x_2, \cdot, \cdot), H^{-1})]^{-n}.$$

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 :

$$\theta(h, x_1, x_2) \approx [1 + C_2(x_1, x_2)H^{-2} + C_3(x_1, x_2)H^{-3} + C_4(x_1, x_2)H^{-4}]^{-n} \quad (5.3)$$

where

$$C_q(x_1, x_2) = \frac{k(k+1)\dots(k+q-1)}{q!} \int \tilde{b}(x_1, x_2, \xi_1, \xi_2)^q d\xi_1 d\xi_2$$

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. We therefore move the unsmoothed topography to processor zero and do the smoothing and the coefficient-computing there. The class `stressbalance::BedSmoothen` 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 [83]. 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

$$P_4''(s_{min}) = \frac{k(k+1)(k+4)}{2(k+3)} > 0.$$

5.5 Calving front stress boundary condition

Contents

- *Calving front stress boundary condition*
 - *Notation*
 - *Calving front stress boundary condition*
 - *Shallow shelf approximation*
 - *Implementing the calving front boundary condition*

5.5.1 Notation

Variable	Meaning
h	ice top surface elevation
b	ice bottom surface elevation
$H = h - b$	ice thickness
g	acceleration due to gravity
ν	vertically-averaged viscosity of ice
\mathbf{n}	normal vector
$B(T)$	ice hardness
D	strain rate tensor
d_e	effective strain rate
t	Cauchy stress tensor
t^D	deviatoric stress tensor; note $t_{ij}^D = t_{ij} + p\delta_{ij}$

5.5.2 Calving front stress boundary condition

In the 3D case the calving front stress boundary condition ([56], equation (6.19)) reads

$$t|_{\text{cf}} \cdot \mathbf{n} = -p_{\text{water}} \mathbf{n}.$$

Expanded in component form, and evaluating the left-hand side at the calving front and assuming that the calving front face is vertical ($\mathbf{n}_z = 0$), this gives

$$\begin{aligned} (t_{xx}^D - p)\mathbf{n}_x + t_{xy}^D \mathbf{n}_y &= -p_{\text{water}} \mathbf{n}_x, \\ t_{xy}^D \mathbf{n}_x + (t_{yy}^D - p)\mathbf{n}_y &= -p_{\text{water}} \mathbf{n}_y, \\ t_{xz}^D \mathbf{n}_x + t_{yz}^D \mathbf{n}_y &= 0. \end{aligned}$$

Because we seek boundary conditions for the SSA stress balance, in which the vertically-integrated forms of the stresses $t_{xx}^D, t_{xy}^D, t_{yy}^D$ are balanced separately from the shear stresses t_{xz}^D, t_{yz}^D , the third of the above equations can be omitted from the remaining analysis.

Let $p_{\text{ice}} = \rho_{\text{ice}} g(h - z)$. In the hydrostatic approximation, $t_{zz} = -p_{\text{ice}}$ ([56], equation (5.59)). Next, since t^D has trace zero,

$$\begin{aligned} p &= p - t_{xx}^D - t_{yy}^D - t_{zz}^D \\ &= -t_{zz} - t_{xx}^D - t_{yy}^D \\ &= p_{\text{ice}} - t_{xx}^D - t_{yy}^D. \end{aligned}$$

Thus

$$\begin{aligned} (2t_{xx}^D + t_{yy}^D)\mathbf{n}_x + t_{xy}^D \mathbf{n}_y &= (p_{\text{ice}} - p_{\text{water}})\mathbf{n}_x, \\ t_{xy}^D \mathbf{n}_x + (2t_{yy}^D + t_{xx}^D)\mathbf{n}_y &= (p_{\text{ice}} - p_{\text{water}})\mathbf{n}_y. \end{aligned} \quad (5.4)$$

Now, using the “viscosity form” of the flow law

$$\begin{aligned} t^D &= 2\nu D, \\ \nu &= \frac{1}{2}B(T)d_e^{1/n-1} \end{aligned}$$

and the fact that in the shallow shelf approximation u and v are depth-independent, we can define the membrane stress N as the vertically-integrated deviatoric stress

$$N_{ij} = \int_b^h t_{ij}^D dz = 2\bar{v} H D_{ij}.$$

Here \bar{v} is the vertically-averaged effective viscosity.

Integrating (5.4) with respect to z , we get

$$\begin{aligned} (2N_{xx} + N_{yy})\mathbf{n}_x &+ N_{xy}\mathbf{n}_y = \int_b^h (p_{\text{ice}} - p_{\text{water}})dz \mathbf{n}_x, \\ N_{xy}\mathbf{n}_x &+ (2N_{yy} + N_{xx})\mathbf{n}_y = \int_b^h (p_{\text{ice}} - p_{\text{water}})dz \mathbf{n}_y. \end{aligned} \quad (5.5)$$

5.5.3 Shallow shelf approximation

The shallow shelf approximation written in terms of N_{ij} is

$$\begin{aligned} \frac{\partial}{\partial x}(2N_{xx} + N_{yy}) &+ \frac{\partial N_{xy}}{\partial y} = \rho g H \frac{\partial h}{\partial x}, \\ \frac{\partial N_{xy}}{\partial x} &+ \frac{\partial}{\partial y}(2N_{yy} + N_{xx}) = \rho g H \frac{\partial h}{\partial y}. \end{aligned} \quad (5.6)$$

5.5.4 Implementing the calving front boundary condition

We use centered finite difference approximations in the discretization of the SSA (5.6). Consider the first equation:

$$\frac{\partial}{\partial x}(2N_{xx} + N_{yy}) + \frac{\partial N_{xy}}{\partial y} = \rho g H \frac{\partial h}{\partial x}. \quad (5.7)$$

Let \tilde{F} be an approximation of F using a finite-difference scheme. Then the first SSA equation is approximated by

$$\frac{1}{\Delta x} \left((2\tilde{N}_{xx} + \tilde{N}_{yy})_{i+\frac{1}{2},j} - (2\tilde{N}_{xx} + \tilde{N}_{yy})_{i-\frac{1}{2},j} \right) + \frac{1}{\Delta y} \left((\tilde{N}_{xy})_{i,j+\frac{1}{2}} - (\tilde{N}_{xy})_{i,j-\frac{1}{2}} \right) = \rho g H \frac{h_{i+\frac{1}{2},j} - h_{i-\frac{1}{2},j}}{\Delta x}.$$

Now, assume that the cell boundary (face) at $i + \frac{1}{2}, j$ is at the calving front. Then $\mathbf{n} = (1, 0)$ and from (5.5) we have

$$2N_{xx} + N_{yy} = \int_b^h (p_{\text{ice}} - p_{\text{water}})dz. \quad (5.8)$$

We call the right-hand side of (5.8) the ‘‘pressure difference term.’’

In forming the matrix approximation of the SSA [29], [37], instead of assembling a matrix row corresponding to the generic equation (5.7) we use

$$\frac{1}{\Delta x} \left(\left[\int_b^h (p_{\text{ice}} - p_{\text{water}})dz \right]_{i+\frac{1}{2},j} - (2\tilde{N}_{xx} + \tilde{N}_{yy})_{i-\frac{1}{2},j} \right) + \frac{1}{\Delta y} \left((\tilde{N}_{xy})_{i,j+\frac{1}{2}} - (\tilde{N}_{xy})_{i,j-\frac{1}{2}} \right) = \rho g H \frac{h_{i+\frac{1}{2},j} - h_{i-\frac{1}{2},j}}{\Delta x}.$$

Moving terms that do not depend on the velocity field to the right-hand side, we get

$$\frac{1}{\Delta x} \left(-(2\tilde{N}_{xx} + \tilde{N}_{yy})_{i-\frac{1}{2},j} \right) + \frac{1}{\Delta y} \left((\tilde{N}_{xy})_{i,j+\frac{1}{2}} - (\tilde{N}_{xy})_{i,j-\frac{1}{2}} \right) = \rho g H \frac{h_{i+\frac{1}{2},j} - h_{i-\frac{1}{2},j}}{\Delta x} + \left[\frac{\int_b^h (p_{\text{water}} - p_{\text{ice}})dz}{\Delta x} \right]_{i+\frac{1}{2},j}.$$

The second equation and other cases ($\mathbf{n} = (-1, 0)$, etc) are treated similarly.

5.6 Notes about the flow-line SSA

Using the same notation as in the rest of the manual, the flow-line case the shallow shelf approximation reads

$$(4vHu')' = \rho_i g H h'. \quad (5.9)$$

Here u is the ice velocity, $v = v(u')$ is the ice viscosity, H is the ice thickness, and h is the ice surface elevation.

Let $\alpha = (1 - \rho_{\text{ice}}/\rho_{\text{water}})$ and note that $h = \alpha H$ whenever ice is floating, so

$$(4\bar{v}Hu')' = \frac{1}{2}\alpha\rho_i g (H^2)'. \quad (5.10)$$

We can easily integrate this, getting

$$4\bar{v}Hu' = \frac{1}{2}\alpha\rho_i g H^2. \quad (5.11)$$

Note: This is a non-linear *first-order* ODE for the ice velocity u .

5.6.1 An observation

Consider a boundary-value problem for the velocity u of a floating ice shelf on an interval $[0, L]$:

$$\begin{aligned} (4vHu')' &= \rho_i g H h', \\ u(0) &= u_0, \\ (4vHu')|_{x=L} &= \tau_{\text{stat}}(L), \\ \tau_{\text{stat}}(x) &= \frac{1}{2}\rho_{\text{ice}} g \alpha H(x)^2. \end{aligned} \quad (5.12)$$

Now consider a similar BVP on $[0, a]$ for some positive $a < L$:

$$\begin{aligned} (4vHv')' &= \rho_i g H h', \\ v(0) &= u_0, \\ (4v(v')Hv')|_{x=a} &= \tau_{\text{stat}}(a). \end{aligned} \quad (5.13)$$

Because (5.12) is a first-order ODE, the solutions u of (5.12) and v of (5.13) *coincide* on $[0, a]$, i.e. $u(x) = v(x)$ for all $x \in [0, a]$.

Note: This implies that the velocity $u(x)$ of a floating flow-line ice shelf modeled by (5.12) is not sensitive to ice geometry perturbations at locations downstream from x .

5.6.2 Discrete analog of this property

Let $N > 2$ be an integer and define the N -point grid $x_j = 0 + j\Delta x$, $\Delta x = L/(N - 1)$.

Discretizing (5.12) on this grid results in a non-linear system with $N - 1$ unknowns (call this system S_L).

Let $a = L - \Delta x$ and write down a system of equations by discretizing (5.13) on the grid consisting of the first $N - 1$ of x_j . This system (call it S_a) has $N - 2$ unknowns.

The property we would like our discretization to have is this:

If u_1, \dots, u_{N-1} is the solution of S_L and v_1, \dots, v_{N-2} solves S_a , then $u_k = v_k$ for all $k = 1, \dots, N - 2$.

Note that the first $N - 3$ equations in S_L and S_a are the same.

5.6.3 Discretization

Let $[X]_j$ be an approximation of X at a grid location j .

The standard approach *within* the domain is to use centered finite differences and linear interpolation to approximate staggered-grid values, i.e. averaging values at immediate regular grid point neighbors:

$$\begin{aligned} [f]_{i+\frac{1}{2}} &= \frac{1}{2}(f_i + f_{i+1}), \\ [f']_i &= \frac{1}{\Delta x}([f]_{i+\frac{1}{2}} + [f]_{i-\frac{1}{2}}). \end{aligned} \quad (5.14)$$

Interior

$$\frac{1}{\Delta x} ([4\nu(u')Hu']_{i+\frac{1}{2}} - [4\nu(u')Hu']_{i-\frac{1}{2}}) = [\rho_{\text{ice}} g \alpha H H']_i. \quad (5.15)$$

Ice front

Let n be the last grid point with non-zero ice thickness, i.e. assume that $H_k = 0$ for $k > n$.

The implementation of the stress boundary condition at the ice front amounts to adding one more equation (see (5.8)):

$$[4\nu(u')Hu']_{n+\frac{1}{2}} = [\tau_{\text{stat}}]_{n+\frac{1}{2}}. \quad (5.16)$$

We can then combine (5.16) with (5.15) (with i replaced by n) to get the discretization at the ice front:

$$\begin{aligned} \frac{1}{\Delta x} ([\tau_{\text{stat}}]_{n+\frac{1}{2}} - [4\nu(u')Hu']_{n-\frac{1}{2}}) &= \rho_{\text{ice}} g \alpha [H H']_n, \\ -[4\nu(u')Hu']_{n-\frac{1}{2}} &= \Delta x \rho_{\text{ice}} g \alpha [H H']_n - [\tau_{\text{stat}}]_{n+\frac{1}{2}}. \end{aligned} \quad (5.17)$$

Choosing FD approximations

Assuming that the ice front is at n

If we assume that the ice front is at n , the last equation in the system looks like (5.17):

$$-[4\nu(u')Hu']_{n-\frac{1}{2}} = \Delta x \rho_{\text{ice}} g \alpha [H H']_n - [\tau_{\text{stat}}]_{n+\frac{1}{2}}. \quad (5.18)$$

Assuming that the ice front is at $n + 1$

If we assume that the ice front is at $n + 1$, the n -th equation looks like a “generic” interior equation (5.15) and we have one more equation ((5.18)) shifted by 1:

$$\begin{aligned} \frac{1}{\Delta x} ([4\nu(u')Hu']_{n+\frac{1}{2}} - [4\nu(u')Hu']_{n-\frac{1}{2}}) &= [\rho_{\text{ice}} g \alpha H H']_n, \\ -[4\nu(u')Hu']_{(n+1)-\frac{1}{2}} &= \Delta x \rho_{\text{ice}} g \alpha [H H']_{n+1} - [\tau_{\text{stat}}]_{n+1+\frac{1}{2}}. \end{aligned} \quad (5.19)$$

Note that the second equation in (5.19) is the same as (5.18), but with the index shifted by 1. Both correspond to locations *at the ice front*.

The goal

We want to choose FD approximations $[\tau_{\text{stat}}]_*$ and $[H H']_*$ in a way that would make it possible to obtain (5.18) by transforming equations (5.19).

We propose using *constant extrapolation* to approximate $H_{n+\frac{1}{2}}$

$$[H]_{i+\frac{1}{2}} = \begin{cases} H_i, & i \text{ is at the ice front} \\ \frac{1}{2}(H_i + H_{i+1}), & \text{otherwise.} \end{cases} \quad (5.20)$$

This gives us the following approximation of derivatives:

$$[H']_i = \begin{cases} \frac{1}{\Delta x}(H_i - [H]_{i-\frac{1}{2}}), & i \text{ is at the ice front} \\ \frac{1}{\Delta x}([H]_{i+\frac{1}{2}} - [H]_{i-\frac{1}{2}}), & \text{otherwise.} \end{cases}$$

After substituting (5.14) this becomes

$$[H']_i = \begin{cases} \frac{1}{2\Delta x}(H_i - H_{i-1}), & i \text{ is at the ice front} \\ \frac{1}{2\Delta x}(H_{i+1} - H_{i-1}), & \text{otherwise.} \end{cases} \quad (5.21)$$

Note: The ice front case in (5.21) is the **one half** of the standard one-sided finite-difference approximation of H' .

Checking if (5.20) is the right choice

Consider the first equation in (5.19) and note that it corresponds to the case in which n is **not** at the ice front.

$$\frac{1}{\Delta x} ([4\nu(u')Hu']_{n+\frac{1}{2}} - [4\nu(u')Hu']_{n-\frac{1}{2}}) = [\rho_{\text{ice}} g \alpha H H']_n,$$

Multiplying by Δx and moving one of the terms to the right hand side, we get

$$\begin{aligned} -[4\nu(u')Hu']_{n-\frac{1}{2}} &= \Delta x [\rho_{\text{ice}} g \alpha H H']_n - [4\nu(u')Hu']_{n+\frac{1}{2}}. \\ -[4\nu(u')Hu']_{n-\frac{1}{2}} &= \Delta x \rho_{\text{ice}} g \alpha H_n \frac{H_{n+1} - H_{n-1}}{2\Delta x} - [4\nu(u')Hu']_{n+\frac{1}{2}}. \\ &= \frac{1}{2} \rho_{\text{ice}} g \alpha H_n (H_{n+1} - H_{n-1}) - [4\nu(u')Hu']_{n+\frac{1}{2}}. \end{aligned} \quad (5.22)$$

Now consider the second equation in (5.19). Note that here $n+1$ **is** at the ice front.

$$\begin{aligned} -[4\nu(u')Hu']_{(n+1)-\frac{1}{2}} &= \Delta x \rho_{\text{ice}} g \alpha [H H']_{n+1} - [\tau_{\text{stat}}]_{n+1+\frac{1}{2}}. \\ &= \Delta x \rho_{\text{ice}} g \alpha H_{n+1} [H']_{n+1} - \frac{1}{2} \rho_{\text{ice}} g \alpha [H]_{n+1+\frac{1}{2}}^2. \\ &= \Delta x \rho_{\text{ice}} g \alpha H_{n+1} \frac{H_{n+1} - H_n}{2\Delta x} - \frac{1}{2} \rho_{\text{ice}} g \alpha H_{n+1}^2 \\ &= \frac{1}{2} \rho_{\text{ice}} g \alpha (H_{n+1}(H_{n+1} - H_n) - H_{n+1}^2) \\ &= \frac{1}{2} \rho_{\text{ice}} g \alpha (H_{n+1}^2 - H_{n+1}H_n - H_{n+1}^2) \\ &= -\frac{1}{2} \rho_{\text{ice}} g \alpha H_{n+1}H_n. \end{aligned} \quad (5.23)$$

Put together, (5.22) and (5.23) read as follows:

$$\begin{aligned} -[4\nu(u')Hu']_{n-\frac{1}{2}} &= \frac{1}{2} \rho_{\text{ice}} g \alpha H_n (H_{n+1} - H_{n-1}) - [4\nu(u')Hu']_{n+\frac{1}{2}}, \\ -[4\nu(u')Hu']_{n+\frac{1}{2}} &= -\frac{1}{2} \rho_{\text{ice}} g \alpha H_{n+1}H_n. \end{aligned} \quad (5.24)$$

Substituting the second equation into the first produces

$$\begin{aligned}
-[4\nu(u')Hu']_{n-\frac{1}{2}} &= \frac{1}{2} \rho_{\text{ice}} g \alpha H_n (H_{n+1} - H_{n-1}) - \frac{1}{2} \rho_{\text{ice}} g \alpha H_{n+1} H_n \\
&= \frac{1}{2} \rho_{\text{ice}} g \alpha (H_n (H_{n+1} - H_{n-1}) - H_{n+1} H_n) \\
&= -\frac{1}{2} \rho_{\text{ice}} g \alpha H_n H_{n-1}
\end{aligned} \tag{5.25}$$

Compare (5.25) to (5.23). Note that *they are the same*, except for the index shift. In other words, (5.25) is the same as (5.18), as desired.

This confirms that finite difference approximations (5.20) and (5.21) result in a discretization with the property we seek:

modeled ice velocity at a given location x along a flow-line ice shelf is not sensitive to geometry perturbations downstream from x .

5.7 BOMPROOF, PISM's numerical scheme for conservation of energy

5.7.1 Introduction

One of the essential goals for any thermomechanically-coupled numerical ice sheet model is a completely bombproof numerical scheme for the advection-conduction-reaction problem for the conservation of energy within the ice. “Bombproof” means being as stable as possible in as many realistic modeling contexts as possible. PISM’s scheme is observed to be highly robust in practice, but it is also provably stable in a significant range of circumstances. The scheme is special to shallow ice sheets because it makes specific tradeoff choices with respect to vertical velocity. It is generic and low order in how it treats horizontal velocity. In this page we state the scheme, prove its stability properties, and address the basal boundary condition.

The scheme is conditionally-stable. The length of the time step is limited only by the maximum magnitude of the horizontal velocities within the ice, i.e. horizontal CFL. This condition for stability is included in the PISM adaptive time-stepping technique.

Accuracy is necessarily a second goal. Our shallow scheme has truncation error $O(\Delta z^2)$ in many circumstances, though it reverts to lower order when it “detects trouble” in the form of large vertical velocities. Overall the scheme has order $O(\Delta t, \Delta x, \Delta y, \Delta z^2)$ in circumstances where the vertical ice flow velocity is small enough, relative to conductivity, and otherwise reverts to $O(\Delta t, \Delta x, \Delta y, \Delta z^1)$.

The conservation of energy problem for the ice is in terms of an enthalpy field [35]. The current scheme supercedes the cold-ice, temperature-based scheme described in the appendices of [30] and in [29]. Compared to a cold-ice scheme, the enthalpy formulation does a better job of conserving energy, has a more-physical model for basal melt rate and drainage, and can model polythermal ice with any CTS topology [35]. The finite difference implementation of the enthalpy method is robust and avoids the CFL condition on vertical advection which was present in the older, cold-ice scheme.

The bedrock thermal problem is solved by splitting the timestep into an update of the bedrock temperature field, assuming the ice base is at constant temperature, and an update of the ice enthalpy field, by the BOMPROOF scheme here, assuming the upward heat flux from the bedrock layer is constant during the timestep. For more on the implementation, see the `BedThermalUnit` class.

The region in which the conservation of energy equation needs to be solved changes over time. This is an essential complicating factor in ice sheet modeling. Also relevant is that the velocity field has a complicated provenance as it comes from different stress balance equations chosen at runtime. These stress balances, especially with transitions in flow type, for instance at grounding lines, are incompletely understood when thermomechanically-coupled. (The `ShallowStressBalance` instance owned by `IceModel1` could be the SIA, the SSA, a hybrid of these, or other stress

balances in future PISM versions.) We will therefore not make, in proving stability, assumptions about the regularity of the velocity field in space or time other than boundedness.

Nor do we want the numerical scheme for advection to need any information about the velocity except its value at the beginning of the time step. Thus the conservation of energy timestep is assumed to be split from the mass continuity time step and its associated stress balance solve. We have not considered implementing a scheme which requires the Jacobian of the velocity field with respect to changes in enthalpy, for example. At very least such a fully-implicit scheme would require blind iteration (e.g. with no guarantee of convergence of the iteration). The scheme we propose involves no such iteration.

5.7.2 Conservation of energy in a shallow ice sheet

In an enthalpy formulation [35] (and references therein), the ice sheet is regarded as a mixture of two phases, solid and liquid, so that both cold and temperate ice with liquid ice matrix can be modeled. The specific enthalpy field of the ice mixture is denoted $E(x, y, z, t)$ and has units J/kg . (Within the PISM documentation the symbol H is used for ice thickness so we use E for enthalpy here and in the PISM source code versus “H” in [35].) The conservation of energy equation is

$$\rho \frac{dE}{dt} = -\nabla \cdot \mathbf{q} + Q, \quad (5.26)$$

where ρ is the mixture density. The mixture density is assumed to be the same as ice density even if there is a nonzero liquid fraction, and the mixture is assumed to be incompressible [35]. The left and right sides of equation (5.26), and thus the quantity Q , have units $J s^{-1} m^{-3} = W m^{-3}$.

Neglecting the dependence of conductivity and heat capacity on temperature [35], the heat flux in cold ice and temperate ice is

$$\mathbf{q} = \begin{cases} -\frac{k_i}{c_i} \nabla E, & \text{cold ice,} \\ -K_0 \nabla E, & \text{temperate ice,} \end{cases} \quad (5.27)$$

where k_i, c_i, K_0 are constant [35]. The nonzero flux in the temperate ice case, may be conceptualized as a regularization of the “real” equation, or as a flux of latent heat carried by liquid water. Also, dE/dt stands for the material derivative of the enthalpy field, so the expanded form of (5.26) is

$$\rho \left(\frac{\partial E}{\partial t} + \mathbf{U} \cdot \nabla E \right) = \nabla \cdot \left(\left\{ \frac{k_i}{c_i} \right\}_{K_0} \nabla E \right) + Q,$$

where \mathbf{U} is the three dimensional velocity, thus advection is included.

The additive quantity Q is the dissipation (strain-rate) heating,

$$Q = \sum_{i,j=1}^3 D_{ij} \tau_{ij}$$

where D_{ij} is the strain rate tensor and τ_{ij} is the deviatoric stress tensor. Reference [29] addresses how this term is computed in PISM, according to the shallow stress balance approximations; see `StressBalance::compute_volumetric_strain_heating()`. (Q is called Σ in [30], [29] and in many places in the source code.)

Friction from sliding also is a source of heating. It has units of $W/m^2 = J/(m^2 s)$, that is, the same units as the heat flux \mathbf{q} above. In formulas we write

$$F_b = -\tau_b \cdot \mathbf{u}_b,$$

where τ_b is the basal shear stress and \mathbf{u}_b is the basal sliding velocity; the basal shear stress is oppositely-directed to the basal velocity. For example, in the plastic case $\tau_b = -\tau_c \mathbf{u}_b / |\mathbf{u}_b|$ where τ_c is a positive scalar, the yield stress.

See method `StressBalance::basal_frictional_heating()`. The friction heating is concentrated at $z = 0$, and it enters into the basal boundary condition and melt rate calculation, addressed in section [Temperate basal boundary condition, and computing the basal melt rate](#) below.

We use a shallow approximation of equation (5.26) which lacks horizontal conduction terms [31]. For the initial analysis of the core BOMPROOF scheme, we specialize to cold ice. Within cold ice, the coefficient in the heat flux is constant, so

$$\nabla \cdot \mathbf{q} = -\frac{k_i}{c_i} \frac{\partial^2 E}{\partial z^2}.$$

Therefore the equation we initially analyze is

$$\rho_i \left(\frac{\partial E}{\partial t} + \mathbf{U} \cdot \nabla E \right) = \frac{k_i}{c_i} \frac{\partial^2 E}{\partial z^2} + Q, \quad (5.28)$$

We focus the analysis on the direction in which the enthalpy has largest derivative, namely with respect to the vertical coordinate z . Rewriting equation (5.28) to emphasize the vertical terms we have

$$\rho_i \left(\frac{\partial E}{\partial t} + w \frac{\partial E}{\partial z} \right) = \frac{k_i}{c_i} \frac{\partial^2 E}{\partial z^2} + \Phi \quad (5.29)$$

where

$$\Phi = Q - \rho_i \left(u \frac{\partial E}{\partial x} + v \frac{\partial E}{\partial y} \right)$$

We assume that the surface enthalpy $E_s(t, x, y)$ (K) and the geothermal flux $G(t, x, y)$ (W/m^2) at $z = 0$ are given. (The latter is the output of the `energy::BedThermalUnit` object, and it may come from an evolving temperature field within the upper crust, the bedrock layer. If a surface temperature is given then it will be converted to enthalpy by the `EnthalpyConverter` class.) The boundary conditions to problem (5.29) are, therefore,

$$\begin{aligned} E(t, x, y, z = H) &= E_s(t, x, y), \\ -\frac{k_i}{c_i} \frac{\partial E}{\partial z} \Big|_{z=0} &= G. \end{aligned} \quad (5.30)$$

For a temperate ice base, including any ice base below which there is liquid water, the lower boundary condition is more interesting. It is addressed below in section [Temperate basal boundary condition, and computing the basal melt rate](#).

5.7.3 The core BOMPROOF scheme

For the discussion of the numerical scheme below, let E_{ijk}^n be our approximation to the exact enthalpy E at the grid point with coordinates (x_i, y_j, z_k) at time t_n . When i, j are uninteresting we suppress them and write E_k^n , and we will use similar notation for numerical approximations to the other quantities. We put the horizontal advection terms in the source term Φ because we treat them explicitly, evaluating at time t_n . (Implicit or semi-implicit treatment of horizontal advection would require a coupled system distributed across processors, a difficulty which is currently avoided.)

The scheme we use for horizontal advection is explicit first-order upwinding. There is a CFL condition for the scheme to be stable, in the absence of conduction, based on the magnitude of the horizontal velocity components. To state the upwind scheme itself, let

$$\text{Up}(f_\bullet | \alpha) = \begin{cases} f_i - f_{i-1}, & \alpha \geq 0, \\ f_{i+1} - f_i, & \alpha < 0. \end{cases}$$

The approximate horizontal advection terms, and thus the approximation to the whole term Φ , are

$$\Phi_{ijk}^n = \Sigma_{ijk}^n - \rho_i \left(u_{ijk}^n \frac{\text{Up}(E_{\bullet,jk}^n | u_{ijk}^n)}{\Delta x} + v_{ijk}^n \frac{\text{Up}(E_{i,k}^n | v_{ijk}^n)}{\Delta y} \right).$$

The CFL stability condition for this part of the scheme is

$$\Delta t \left(\left| \frac{u_{ijk}^n}{\Delta x} \right| + \left| \frac{v_{ijk}^n}{\Delta y} \right| \right) \leq 1. \quad (5.31)$$

The routine `max_timestep_cfl_3d()` computes the maximum of velocity magnitudes. This produces a time step restriction based on the above CFL condition. Then `IceModel::max_timestep()` implements adaptive time-stepping based on this and other stability criteria.

In the analysis below we assume an equally-spaced grid z_0, \dots, z_{M_z} with $\Delta z = z_{k+1} - z_k$. In fact PISM has a remapping scheme in each column, wherein the enthalpy in a column of ice is stored on an unequally-spaced vertical grid, but is mapped to a fine, equally-spaced grid for the conservation of energy computation described here. (Similar structure applies to the age computation. See classes `EnthalpyModel` and `AgeModel`.)

The z derivative terms in (5.29) will be approximated implicitly. Let λ be in the interval $0 \leq \lambda \leq 1$. Suppressing indices i, j , the approximation to (5.29) is

$$\begin{aligned} & \rho_i \left(\frac{E_k^{n+1} - E_k^n}{\Delta t} + \lambda w_k^n \frac{E_{k+1}^{n+1} - E_{k-1}^{n+1}}{2\Delta z} + (1 - \lambda) w_k^n \frac{\text{Up}(E_\bullet^{n+1} | w_k^n)}{\Delta z} \right) \\ &= \frac{k_i}{c_i} \frac{E_{k+1}^{n+1} - 2E_k^{n+1} + E_{k-1}^{n+1}}{\Delta z^2} + \Phi_k^n. \end{aligned} \quad (5.32)$$

Equation (5.32), along with a determination of λ by (5.36) below, is the scheme BOMBPROOF. It includes two approximations of vertical advection, implicit centered difference ($\lambda = 1$) and implicit first-order upwinding ($\lambda = 0$). They are combined using nonnegative coefficients which sum to one, a convex combination. The centered formula has higher accuracy,

$$w_k^n \frac{E_{k+1}^{n+1} - E_{k-1}^{n+1}}{2\Delta z} = w \frac{\partial E}{\partial z} + O(\Delta t, \Delta z^2),$$

while the first order upwind formula has lower accuracy,

$$w_k^n \frac{\text{Up}(E_\bullet^{n+1} | w_k^n)}{\Delta z} = w \frac{\partial E}{\partial z} + O(\Delta t, \Delta z).$$

Thus we prefer to use the centered formula when possible, but we apply (implicit) upwinding when it is needed for its added stability benefits.

We now rewrite (5.32) for computational purposes as one of a system of equations for the unknowns $\{E_k^{n+1}\}$. In this system the coefficients will be scaled so that the diagonal entries of the matrix have limit one as $\Delta t \rightarrow 0$. Let

$$\begin{aligned} v &= \frac{\Delta t}{\Delta z}, \\ R &= \frac{k_i \Delta t}{\rho_i c_i \Delta z^2}. \end{aligned}$$

Now multiply equation (5.32) by Δt , divide it by ρ_i , and rearrange:

$$\begin{aligned} & \left(-R - vw_k^n \begin{Bmatrix} 1 - \lambda/2 \\ \lambda/2 \end{Bmatrix} \right) E_{k-1}^{n+1} \\ &+ \left(1 + 2R + vw_k^n (1 - \lambda) \begin{Bmatrix} +1 \\ -1 \end{Bmatrix} \right) E_k^{n+1} \\ &+ \left(-R + vw_k^n \begin{Bmatrix} \lambda/2 \\ 1 - \lambda/2 \end{Bmatrix} \right) E_{k+1}^{n+1} \\ &= E_k^n + \Delta t \rho_i^{-1} \Phi_k^n \end{aligned} \quad (5.33)$$

Here $\begin{cases} a \\ b \end{cases} = a$ when $w_k^n \geq 0$ and $\begin{cases} a \\ b \end{cases} = b$ when $w_k^n < 0$.

Equation (5.33) has coefficients which are scaled to have no units. It is ready to be put in the system managed by `enthSystemCtx`.

One way of stating the stability of first-order upwinding is to say it satisfies a “maximum principle” [71]. An example of a maximum principle for this kind of finite difference scheme is that if $U_{k-1}^n, U_k^n, U_{k+1}^n$ are adjacent gridded values of some abstract quantity at time step t_n , and if the next value satisfies the scheme

$$U_k^{n+1} = C_{-1}U_{k-1}^n + C_0U_k^n + C_{+1}U_{k+1}^n \quad (5.34)$$

for *nonnegative* coefficients C_i summing to one, $C_{-1} + C_0 + C_{+1} = 1$, then it follows by the triangle inequality that

$$\min\{|U_{k-1}^n|, |U_k^n|, |U_{k+1}^n|\} \leq |U_k^{n+1}| \leq \max\{|U_{k-1}^n|, |U_k^n|, |U_{k+1}^n|\}.$$

Thus a “wiggle” cannot appear in $\{U_k^{n+1}\}$ if previous values $\{U_k^n\}$ were smoother. The proof below shows the corresponding “wiggle-free” property for scheme (5.33).

However, the pure implicit centered difference scheme ($\lambda = 1$), namely

$$\begin{aligned} & (-R - \nu w_k^n / 2) E_{k-1}^{n+1} + (1 + 2R) E_k^{n+1} \\ & + (-R + \nu w_k^n / 2) E_{k+1}^{n+1} = E_k^n + \Delta t \rho_i^{-1} \Phi_k^n \end{aligned} \quad (5.35)$$

is *less stable* than implicit first-order upwinding. It is less stable in the same sense that Crank-Nicolson is a less stable scheme than backwards Euler for the simplest heat equation $u_t = u_{xx}$ [71]. In fact, although oscillatory modes cannot grow exponentially under equation (5.35), those modes *can* appear when none are present already, even in the homogeneous case $\Phi_k^n = 0$.

5.7.4 Stability properties of the BOMBPROOF scheme

We want to be precise about the phrase “unconditionally stable” for BOMBPROOF. To do so we consider somewhat simplified cases which are amenable to analysis, and we prove two stability properties. These stability properties identify the precise advantages of BOMBPROOF.

Theorem (stating the stability properties).

Assume, for the precise but limited assertion of this theorem, that the surface temperature T_s and the geothermal flux G are constant in time. Assume also that the entire source function Φ is identically zero (but see comments below). Fix an equally-spaced vertical grid $z_0 = 0 < z_1 < \dots < z_N = H$, so that the upper grid point coincides with the surface of the ice. With these assumptions, if

$$\lambda = \min \left\{ 1, \min_{k=0,\dots,N} \left\{ \frac{2k_i}{|w_k^n| \rho_i c_i \Delta z} \right\} \right\}, \quad (5.36)$$

reset at each time step n , then scheme (5.32), (5.33) is unconditionally-stable in the following two senses:

1. A maximum principle applies without further assumptions.
2. Suppose we freeze the coefficients of the problem to have constant values in time and space. (Concretely, we assume that λ is chosen independently of the time step n , and that Δt is the same for each time step. We assume constant vertical velocity $w_k^n = w_0$. We also consider a spatially-periodic or unbounded version of our problem, with no boundary conditions.) Then a von Neumann analysis of the constant coefficient problem yields a growth factor less than one for all modes on the grid.

Remarks

The phrases *maximum principle* and *von Neumann analysis* will be precisely illustrated in the following proof. Both approaches are in [71]. There is additional information on the von Neumann analysis of implicit finite difference methods for advection in [166].

These statements also apply in case $k_i = 0$, in which case (5.36) implies $\lambda = 0$, and the method reduces to implicit first-order upwinding. (Implicit first-order upwinding has properties 1 and 2 [166].) The case $k_i = 0$ is relevant because it applies to the least-transport model of temperate ice in which there is zero enthalpy conduction. (One reasonable model for temperate ice is to assume no transport of the liquid fraction, whether diffusive transport or otherwise, and to ignore conduction along the temperature gradient, because the gradient is only from pressure-melting temperature differences.)

Proof of 1

In the case considered for the maximum principle, with $\Phi_k^n = 0$, we can rewrite (5.33) as

$$\begin{aligned} & \left(1 + 2R + \nu w_k^n (1 - \lambda) \begin{Bmatrix} +1 \\ -1 \end{Bmatrix}\right) E_k^{n+1} \\ &= E_k^n + \left(R + \nu w_k^n \begin{Bmatrix} 1 - \lambda/2 \\ \lambda/2 \end{Bmatrix}\right) E_{k-1}^{n+1} + \left(R - \nu w_k^n \begin{Bmatrix} \lambda/2 \\ 1 - \lambda/2 \end{Bmatrix}\right) E_{k+1}^{n+1}. \end{aligned} \quad (5.37)$$

We claim that with choice (5.36) for $0 \leq \lambda \leq 1$, all coefficients in (5.37) are nonnegative. At one extreme, in the upwinding case ($\lambda = 0$), all the coefficients are nonnegative. Otherwise, note that $\nu w_k^n (1 - \lambda) \begin{Bmatrix} +1 \\ -1 \end{Bmatrix}$ is nonnegative for any valid value of λ and for any value of w_k^n , noting the meaning of the $\begin{Bmatrix} +1 \\ -1 \end{Bmatrix}$ symbol. Thus the coefficient on the left is always nonnegative. The coefficient of E_{k-1}^{n+1} is clearly nonnegative for any valid value of λ if $w_k^n \geq 0$. The coefficient of E_{k+1}^{n+1} is clearly nonnegative for any valid value of λ if $w_k^n \leq 0$.

Therefore the only concerns are for the coefficient of E_{k-1}^{n+1} when $w_k^n \leq 0$ and the coefficient of E_{k+1}^{n+1} when $w_k^n \geq 0$. But if λ is smaller than $2k_i/(|w_k^n|\rho_i c_i \Delta z)$ then

$$R - \nu |w_k^n|(\lambda/2) = \frac{k_i \Delta t}{\rho_i c_i \Delta z^2} - \frac{\Delta t |w_k^n| \lambda}{\Delta z} \frac{1}{2} \geq \frac{k_i \Delta t}{\rho_i c_i \Delta z^2} - \frac{\Delta t |w_k^n|}{\Delta z} \frac{k_i}{|w_k^n| \rho_i c_i \Delta z} = 0.$$

Thus all the coefficients in (5.37) are nonnegative. On the other hand, in equation (5.37), all coefficients on the right side sum to

$$1 + 2R + \nu w_k^n \begin{Bmatrix} 1 - \lambda \\ -1 + \lambda \end{Bmatrix} = 1 + 2R + \nu w_k^n (1 - \lambda) \begin{Bmatrix} +1 \\ -1 \end{Bmatrix},$$

which is exactly the coefficient on the left side of (5.37). It follows that

$$E_k^{n+1} = a_k E_k^n + b_k E_{k-1}^{n+1} + c_k E_{k+1}^{n+1}$$

where a_k, b_k, c_k are positive and $a_k + b_k + c_k = 1$. Thus a maximum principle applies [71]. **END OF PROOF OF 1.**

Proof of 2

As a von Neumann analysis is much more restrictive than the analysis above, we will be brief. Let's assume the velocity is downward, $w_0 < 0$; the other case is similar. Equation (5.33) becomes

$$\begin{aligned} & (-R - \nu w_0(\lambda/2)) E_{k-1}^{n+1} + (1 + 2R - \nu w_0(1 - \lambda)) E_k^{n+1} \\ & + (-R + \nu w_0(1 - \lambda/2)) E_{k+1}^{n+1} = E_k^n. \end{aligned} \quad (5.38)$$

The heart of the von Neumann analysis is the substitution of a growing or decaying (in time index n) oscillatory mode on the grid of spatial wave number μ :

$$E_k^n = \sigma^n e^{i\mu(k\Delta z)}.$$

Here $k\Delta z = z_k$ is a grid point. Such a mode is a solution to (5.38) if and only if

$$\begin{aligned} & \sigma [(-R - \nu w_0(\lambda/2)) e^{-i\mu\Delta z} + (1 + 2R - \nu w_0(1 - \lambda)) \\ & + (-R + \nu w_0(\lambda/2)) e^{+i\mu\Delta z} + \nu w_0(1 - \lambda) e^{+i\mu\Delta z}] = 1. \end{aligned}$$

This equation reduces by standard manipulations to

$$\sigma = \frac{1}{1 + (4R - 2\nu w_0(1 - \lambda)) \cos^2(\mu\Delta z/2) + i\nu w_0(1 - \lambda/2) \sin(\mu\Delta z)}.$$

Note $4R - 2\nu w_0(1 - \lambda) \geq 0$ without restrictions on numerical parameters Δt , Δz , because $w_0 < 0$ in the case under consideration. Therefore

$$|\sigma|^2 = \frac{1}{[1 + (4R - 2\nu w_0(1 - \lambda)) \cos^2(\mu\Delta z/2)]^2 + [\nu w_0(1 - \lambda/2) \sin(\mu\Delta z)]^2}.$$

This positive number is less than one, so $|\sigma| < 1$. It follows that all modes decay exponentially. **END OF PROOF OF 2.**

Remark about our von Neumann stability analysis

The constant λ is carefully chosen in (5.36) so that the maximum principle 1 applies. On the other hand, both the implicit first-order upwind and the implicit centered difference formulas have unconditional stability in the von Neumann sense. The proof of case 2 above is thus a formality, merely showing that a convex combination of unconditionally stable (von Neumann sense) schemes is still unconditionally stable in the same sense.

Convergence: a consequence of the maximum principle

If we define the pointwise numerical error $e_k^n = E_k^n - E(t_n, x_i, y_j, z_k)$, where $E(\dots)$ is the unknown exact solution (exact enthalpy field) [71], then (5.37) implies an equality of the form

$$Ae_k^{n+1} = e_k^n + B_- e_{k-1}^{n+1} + B_+ e_{k+1}^{n+1} + \Delta t \tau_k^n$$

where τ_k^n is the truncation error of the scheme and A, B_\pm are nonnegative coefficients, which need no detail for now other than to note that $1 + B_- + B_+ = A$. Letting $\bar{e}^n = \max_k |e_k^n|$ we have, because of the positivity of coefficients,

$$A|e_k^{n+1}| \leq \bar{e}^n + (B_- + B_+) \bar{e}^{n+1} + \Delta t \bar{\tau}^n \quad (5.39)$$

for all k , where $\bar{\tau}^n = \max_k |\tau_k^n|$. Now let k be the index for which $|e_k^{n+1}| = \bar{e}^{n+1}$. For that k we can replace $|e_k^{n+1}|$ in equation (5.39) with \bar{e}^{n+1} . Subtracting the same quantity from each side of the resulting inequality gives

$$\bar{e}^{n+1} \leq \bar{e}^n + \Delta t \bar{\tau}^n,$$

It follows that $\bar{e}^n \leq C\Delta t$, for some finite C , if $\bar{e}^0 = 0$ [71]. Thus a maximum principle for BOMBPROOF implies convergence in the standard way [71]. This convergence proof has the same assumptions as case 1 in the theorem, and thus it only suggests convergence in any broad range of glaciologically-interesting cases.

Remark on nonzero source term

Now recall we assumed in Theorem 1 that the entire “source” Φ_k^n was identically zero. Of course this is not realistic. What we understand is provable, however, is that if a numerical scheme for a linear advection/conduction equation

$$u_t + Au_x = Bu_{xx}$$

is stable in the general sense of numerical schemes for partial differential equations (e.g. as defined in subsection 5.5 of [71]) then the same scheme is stable in the same general sense when applied to the equation with (linear) lower order terms:

$$u_t + Au_x = Bu_{xx} + Cu + D.$$

A precise statement of this general fact is hard to find in the literature, to put it mildly, but theorem 2.2.3 of [166] is one interesting case ($B = 0$ and $D = 0$). But even the form we state with linear term ($Cu + D$) is not adequate to the job because of the strongly-nonlinear dependence of Φ on the temperature T [30].

Nonetheless the maximum principle is a highly-desirable form of stability because we can exclude “wiggles” from the finite difference approximations of the conductive and advective terms, even if the complete physics, with strain heating in particular, is not yet shown to be non-explosive. Because the complete physics includes the appearance of the famous “spokes” of EISMINT II, for example, a maximum principle cannot apply too literally. Indeed there is an underlying fluid instability [30], one that means the solution of the continuum equations can include growing “wiggles” which are fluid features (though not at the grid-based spatial frequency of the usual numerical wiggles). Recall that, because we use first-order upwinding on the horizontal advection terms, we can expect maximum principle-type stability behavior of the whole three-dimensional scheme.

5.7.5 Temperate basal boundary condition, and computing the basal melt rate

At the bottom of grounded ice, a certain amount of heat comes out of the earth and either enters the ice through conduction or melts the base of the ice. On the one hand, see the documentation for `BedThermalUnit` for the model of how much comes out of the earth. On the other hand, [35] includes a careful analysis of the subglacial layer equation and the corresponding boundary conditions and basal melt rate calculation, and the reader should consult that reference.

Regarding the floating case

The shelf base temperature T_{sb} and the melt rate M are supplied by `OceanModel`. Note that we make the possibly-peculiar physical choice that the shelf base temperature is used as the temperature at the *top of the bedrock*, which is actually the bottom of the ocean. This choice means that there should be no abrupt changes in top-of-bedrock heat flux as the grounding line moves. This choice also means that the conservation of energy code does not need to know about the bedrock topography or the elevation of sea level. (In the future `OceanModel` could have a `subshelf_bed_temperature()` method.)

5.8 Computing steady-state subglacial water flux

The “routing” subglacial hydrology model is described by equations

$$\begin{aligned} \frac{\partial W}{\partial t} + \frac{\partial W_{\text{till}}}{\partial t} + \nabla \cdot \mathbf{q} &= \frac{m}{\rho_w} \\ \frac{\partial W_{\text{till}}}{\partial t} &= \frac{m}{\rho_w} - C_d \\ \mathbf{q} &= -kW^\alpha |\nabla \psi|^{\beta-2} \nabla \psi \\ \psi &= P_o + \rho_w g(b + W) \end{aligned} \tag{5.40}$$

on a an ice covered area Ω . We assume zero flux boundary conditions on the inflow part of the boundary and no boundary condition on the outflow boundary. See [110] (equations 1, 2, 6, 16, 26) for details and the notation. Here we also assume that $m \geq 0$.

Our goal is to estimate $Q = \mathbf{q} \cdot \mathbf{n}$, the flux through the outflow part of the boundary of Ω corresponding to the steady state of (5.40) using a method that is computationally cheaper than using the explicit in time approximation of (5.40) described by [110].

Pick a contiguous section ω of $\partial\Omega$ (the terminus of an outlet glacier, for example). Let B be the union of all the trajectories of the vector field \mathbf{q} in Ω that pass through ω . The area B is the “drainage basin” corresponding to ω .

Let $\gamma = \partial B \setminus \omega$. Note that if a point P is in γ then one of the following conditions is satisfied.

1. $|\mathbf{q}| = 0$ (it is the origin of a trajectory that starts within Ω) or
2. $P \in \partial\Omega$ (specifically, P is a part of the inflow part of the boundary of Ω)
3. $\mathbf{q} \cdot \mathbf{n} = 0$ (P is not at the end of a trajectory, and so the normal to the boundary is orthogonal to \mathbf{q}).

Therefore $\mathbf{q} \cdot \mathbf{n} = 0$ on γ and

$$\begin{aligned} \oint_{\partial B} \mathbf{q} \cdot \mathbf{n} \, ds &= \int_{\omega} \mathbf{q} \cdot \mathbf{n} \, ds + \int_{\gamma} \mathbf{q} \cdot \mathbf{n} \, ds \\ &= \int_{\omega} \mathbf{q} \cdot \mathbf{n} \, ds. \end{aligned}$$

Assuming the steady state (and setting time derivatives in (5.40) to zero), integrating over B , and applying the divergence theorem gives

$$\int_{\omega} \mathbf{q} \cdot \mathbf{n} \, ds = \int_B \frac{m}{\rho_w}, \quad (5.41)$$

i.e. *in a steady state the flux through a terminus is equal to the total rate at which water is added to the corresponding drainage basin due to the source term.*

Next, consider a related initial boundary value problem

$$\frac{\partial u}{\partial t} = -\nabla \cdot (\mathbf{V}u) \quad (5.42)$$

on B with $u(x, y, 0) = u_0(x, y)$ ($u_0 \geq 0$), $\mathbf{V} = -k(x, y)\nabla\psi$, zero flux on the inflow boundary, and no boundary condition on the outflow boundary.

Here ψ is the hydraulic potential corresponding to the steady state of (5.40) and $k(x, y)$ is a strictly positive but otherwise arbitrary conductivity function.

Note that since ψ is a steady state hydraulic potential all trajectories of the vector field \mathbf{V} leave B and for $\epsilon > 0$ there is a time $T > 0$ such that

$$\int_B u(T) = \epsilon \int_B u_0.$$

Integrating over time from 0 to T , we get

$$\begin{aligned} \int_0^T \frac{\partial u}{\partial t} \, dt &= - \int_0^T \nabla \cdot (\mathbf{V}u) \, dt, \text{ or} \\ u_0 &= u(T) + \int_0^T \nabla \cdot (\mathbf{V}u) \, dt. \end{aligned}$$

Integrating over B and using the divergence theorem gives

$$\begin{aligned} \int_B u_0 &= \int_B u(T) + \int_B \int_0^T \nabla \cdot (\mathbf{V}u) \\ &= \epsilon \int_B u_0 + \int_0^T \int_B \nabla \cdot (\mathbf{V}u) \\ &= \epsilon \int_B u_0 + \int_0^T \oint_{\partial B} (\mathbf{V}u) \cdot \mathbf{n} \\ &= \epsilon \int_B u_0 + \int_0^T \int_{\omega} (\mathbf{V}u) \cdot \mathbf{n}. \end{aligned}$$

Finally,

$$\int_B u_0 = \frac{1}{1-\epsilon} \int_0^T \int_{\omega} (\mathbf{V}u) \cdot \mathbf{n} \quad (5.43)$$

Combining (5.41) and (5.43) and choosing $u_0 = \tau m / \rho_w$ for some $\tau > 0$ ¹ gives us a way to estimate the flux through the outflow boundary if we know the direction of the steady state flux:

$$\int_{\omega} \mathbf{q} \cdot \mathbf{n} ds = \frac{1}{\tau(1-\epsilon)} \int_0^T \int_{\omega} (\mathbf{V}u) \cdot \mathbf{n} ds. \quad (5.44)$$

Here the right hand side of (5.44) can be estimated by advancing an explicit-in-time approximation of (5.42) until $\int_B u$ drops below a chosen threshold.

However, the direction of the steady state flux \mathbf{q} depends on steady state distributions of W and W_{till} and these quantities are expensive to compute.

To avoid this issue we note that $W \ll H$ and so ψ is well approximated by $\psi_0 = P_o + \rho_w g b$ everywhere except the vicinity of subglacial lakes. Moreover, if $|\nabla W|$ is small then $\nabla \psi_0$ is a reasonable approximation of $\nabla \psi$.

We approximate ψ by $\tilde{\psi} = P_o + \rho_w g b + \delta$ where $\delta > 0$ is an adjustment needed to ensure that $\tilde{\psi}$ has no local minima in the interior of Ω and $|\nabla \tilde{\psi}| > 0$ everywhere on Ω except possibly on a set of measure zero (no “plateaus”).

The approximation of $\tilde{\psi}$ on a computational grid is computed as follows.

1. Set $k = 0$, $\tilde{\psi}_0 = \psi$.
2. Iterate over all grid points. If a grid point (i, j) is at a local minimum, set $\tilde{\psi}_{k+1}(i, j)$ to the average of neighboring values of $\tilde{\psi}_k$ plus a small increment $\Delta\psi$, otherwise set $\tilde{\psi}_{k+1}(i, j)$ to $\tilde{\psi}_k(i, j)$.
3. If step 2 found no local minima, stop. Otherwise increment k and proceed to step 2.

Next, note that it is not necessary to identify the drainage basin B for a terminus ω : it is defined by ψ and therefore an approximation of (5.42) will automatically distribute water inputs from the ice surface (or melting) along the ice margin.

5.8.1 The algorithm

Using an explicit time stepping approximation of (5.42) we can estimate $\int_{\omega} \mathbf{q} \cdot \mathbf{n} ds$ as follows.

1. Given ice thickness H and bed elevation b compute $\tilde{\psi}$ by filling “dips” as described above.
2. Choose the stopping criterion $\epsilon > 0$ and the scaling for the source term $\tau > 0$.

¹ The constant τ is needed to get appropriate units, but its value is irrelevant.

3. Set

$$\begin{aligned} u &\leftarrow \frac{\tau m}{\rho_w}, \\ t &\leftarrow 0, \\ Q &\leftarrow (0, 0). \end{aligned}$$

4. Compute the CFL time step Δt using u and \mathbf{V} .

5. Perform an explicit step from t to $t + \Delta t$, updating u .

6. Accumulate this step's contribution to Q :

$$Q \leftarrow Q + \Delta t \cdot \mathbf{V}u.$$

7. Set $t \leftarrow t + \Delta t$

8. If $\int_{\Omega} u \, dx \, dy > \epsilon$, go to 4.

9. Set

$$Q \leftarrow \frac{1}{t(1 - \epsilon^*)} Q,$$

where

$$\epsilon^* = \frac{\int_{\Omega} u}{\int_{\Omega} u_0}.$$

5.9 Three-equation ocean model (implementation details)

This model is based on [14] and [11].

We use equations for the heat and salt flux balance at the base of the shelf to compute the temperature at the base of the shelf and the sub-shelf melt rate.

Following [14], let Q_T be the total heat flux crossing the interface between the shelf base and the ocean, Q_T^B be the amount of heat lost by the ocean due to melting of glacial ice, and Q_T^I be the conductive flux into the ice column.

Q_T is parameterized by (see [14], equation 10):

$$Q_T = \rho_W c_{PW} \gamma_T (T^B - T^W),$$

where ρ_W is the sea water density, c_{PW} is the heat capacity of sea water, and γ_T is a turbulent heat exchange coefficient.

We assume that the difference between the basal temperature and adjacent ocean temperature $T^B - T^W$ is well approximated by $\Theta_B - \Theta_W$, where Θ is the corresponding potential temperature.

Q_T^B is (see [14], equation 11):

$$Q_T^B = \rho_I L \frac{\partial h}{\partial t},$$

where ρ_I is the ice density, L is the latent heat of fusion, and $\partial h / \partial t$ is the ice thickening rate (equal to minus the melt rate).

The conductive flux into the ice column is ([12], equation 7):

$$Q_T^I = \rho_I c_{PI} \kappa T_{\text{grad}},$$

where ρ_I is the ice density, c_{pI} is the heat capacity of ice, κ is the ice thermal diffusivity, and T_{grad} is the vertical temperature gradient at the base of a column of ice.

Now, the heat flux balance implies

$$Q_T = Q_T^B + Q_T^I.$$

For the salt flux balance, we have

$$Q_S = Q_S^B + Q_S^I,$$

where Q_S is the total salt flux across the interface, Q_S^B is the basal salt flux (negative for melting), $Q_S^I = 0$ is the salt flux due to molecular diffusion of salt through ice.

Q_S is parameterized by ([12], equation 13)

$$Q_S = \rho_W \gamma_S (S^B - S^W),$$

where γ_S is a turbulent salt exchange coefficient, S^B is salinity at the shelf base, and S^W is the salinity of adjacent ocean.

The basal salt flux Q_S^B is ([12], equation 10)

$$Q_S^B = \rho_I S^B \frac{\partial h}{\partial t}.$$

To avoid converting shelf base temperature to shelf base potential temperature and back, we use two linearizations of the freezing point equation for sea water for in-situ and for potential temperature, respectively:

$$\begin{aligned} T^B(S, h) &= a_0 \cdot S + a_1 + a_2 \cdot h, \\ \Theta^B(S, h) &= b_0 \cdot S + b_1 + b_2 \cdot h, \end{aligned}$$

where S is salinity and h is ice shelf thickness.

Note: The linearized equation for the freezing point of seawater as a function of salinity and pressure (ice thickness) is only valid for salinity ranges from 4 to 40 psu (see [11]).

The linearization coefficients for the basal temperature $T^B(S, h)$ are taken from [12], going back to [167].

Given $T^B(S, h)$ and a function $\Theta_T^B(T)$ one can define $\Theta_*^B(S, h) = \Theta_T^B(T^B(S, h))$.

The parameterization $\Theta^B(S, h)$ used here was produced by linearizing $\Theta^B(S, h)$ near the melting point. (The definition of $\Theta_T^B(T)$, converting in situ temperature into potential temperature, was adopted from FESOM [168]).

Treating ice thickness, sea water salinity, and sea water potential temperature as “known” and choosing an approximation of the temperature gradient at the base T_{grad} (see below) we can write down a system of equations

$$\begin{aligned} Q_T &= Q_T^B + Q_T^I, \\ Q_S &= Q_S^B + Q_S^I, \\ T^B(S, h) &= a_0 \cdot S + a_1 + a_2 \cdot h, \\ \Theta^B(S, h) &= b_0 \cdot S + b_1 + b_2 \cdot h \end{aligned}$$

and simplify it to produce a quadratic equation for the salinity at the shelf base, S^B :

$$A \cdot (S^B)^2 + B \cdot S^B + C = 0.$$

The coefficients A , B , and C depend on the basal temperature gradient approximation for the sub-shelf melt, sub-shelf freeze-on, and diffusion-only cases.

- Melt at the base:

$$T_{\text{grad}} = -\Delta T \frac{\partial h / \partial t}{\kappa}.$$

See equation 13 in [12].

- Freeze on at the base: we assume that

$$T_{\text{grad}} = 0.$$

- No melt and no freeze on:

$$T_{\text{grad}} = \frac{\Delta T}{h}.$$

See [11], equation 21.

One remaining problem is that we cannot compute the basal melt rate without making an assumption about whether there is basal melt or not, and cannot pick one of the three cases without computing the basal melt rate first.

Our implementation tries to compute basal salinity that is consistent with the corresponding basal melt rate. See the code for details.

Once S_B is found by solving this quadratic equation, we can compute the basal temperature using the parameterization for $T^B(S, h)$.

To find the basal melt rate, we solve the salt flux balance equation for $\frac{\partial h}{\partial t}$, obtaining

$$w_b = -\frac{\partial h}{\partial t} = \frac{\gamma_S \rho_W (S^W - S^B)}{\rho_I S^B}.$$

See [Basal melt rate and temperature from thermodynamics in boundary layer](#) for the user's documentation of this model.

5.10 Blatter stress balance solver: technical details

Contents

- *Blatter stress balance solver: technical details*
 - *Notation*
 - *Introduction*
 - *Weak form*
 - *Boundary conditions*
 - * *Basal boundary*
 - * *Marine margins*
 - *Solver implementation*
 - * *Discretization*
 - * *Residual evaluation*
 - *Main residual contribution*

- *Driving stress contribution*
- *Basal contribution*
- *Marine boundary contribution*
- *Residual at Dirichlet BC locations*
- * *Jacobian evaluation*
 - *Basal contribution to the Jacobian*
 - *Jacobian at Dirichlet locations*
- * *Iceberg elimination*
- * *Preconditioning*
 - *Vertical grid sizes compatible with coarsening*
 - *Controlling using PETSc options*
 - *Additional code needed to support geometric multigrid*
- * *Parameter continuation as a recovery mechanism*
- * *Model domain and mesh structure*
 - *Supporting evolving ice extent*
- * *Solver inputs and outputs*
- * *Steps performed by the solver*
- * *Integration with the rest of PISM*
 - *Conservation of energy*
- *Testing and verification*
 - * *Verification test XY*
 - * *Verification test XZ*
 - * *Verification test XZ-CFBC*
 - * *Verification test XZ-VV (van der Veen profile)*
- *Known issues and future work*

5.10.1 Notation

u	x -component of ice velocity
v	y -component of ice velocity
\mathbf{u}	2D vector (u, v)
\mathbf{n}	outward-pointing normal vector at domain boundaries
η	ice viscosity (see (5.46))
ρ	ice density
ρ_w	water (usually sea water) density
g	gravitational acceleration
$z_{\text{sea level}}$	sea level elevation
H	ice thickness
s	ice surface elevation
B	ice hardness
A	ice softness
ε_0	regularization parameter
β	basal resistance coefficient
n	Glen exponent

5.10.2 Introduction

This implementation is based on the PETSc example `snes/tutorials/ex48.c` (see [57]) and is inspired by [160], [70], [72], [69], and [161].

Define

$$\begin{aligned}\dot{\boldsymbol{\epsilon}}_1 &= \left(2u_x + v_y, \quad \frac{1}{2}(u_y + v_x), \quad \frac{1}{2}u_z \right), \\ \dot{\boldsymbol{\epsilon}}_2 &= \left(\frac{1}{2}(u_y + v_x), \quad u_x + 2v_y, \quad \frac{1}{2}v_z \right), \\ \dot{\boldsymbol{\epsilon}} &= (\dot{\boldsymbol{\epsilon}}_1, \dot{\boldsymbol{\epsilon}}_2).\end{aligned}$$

Using this notation, the Blatter-Pattyn (BP) stress balance equations are

$$\begin{aligned}-\nabla \cdot (2\eta \dot{\boldsymbol{\epsilon}}_1) + \rho g s_x &= 0, \\ -\nabla \cdot (2\eta \dot{\boldsymbol{\epsilon}}_2) + \rho g s_y &= 0.\end{aligned}\tag{5.45}$$

Here “ $\nabla \cdot$ ” is the three-dimensional divergence operator and the regularized ice viscosity η is defined by

$$\begin{aligned}\eta &= \frac{B}{2} \left(\gamma_{\text{BP}} + \frac{\varepsilon_0}{2} \right)^{\frac{1-n}{2n}}, \\ \gamma_{\text{BP}} &= u_x^2 + v_y^2 + u_x v_y + \frac{1}{4}(u_y + v_x)^2 + \frac{1}{4}u_z^2 + \frac{1}{4}v_z^2,\end{aligned}\tag{5.46}$$

where γ_{BP} is the Blatter-Pattyn approximation of the second invariant of the strain rate tensor:

$$\begin{aligned}\dot{\epsilon}_{ij} &= \frac{1}{2} \left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} \right) \\ \gamma &= \frac{1}{2} \left(\text{trace}(\dot{\boldsymbol{\epsilon}}^2) - \text{trace}(\dot{\boldsymbol{\epsilon}})^2 \right).\end{aligned}\tag{5.47}$$

It is also assumed that

- ice is incompressible and $\text{trace}(\dot{\boldsymbol{\epsilon}}) = 0$, and

- $\frac{\partial w}{\partial x} \ll \frac{\partial u}{\partial x}$ and $\frac{\partial w}{\partial z} \ll \frac{\partial u}{\partial y}$.

The BP approximation of the second invariant γ_{BP} is obtained by omitting w_x and w_y and expressing w_z using incompressibility.

Note: There are at least three equivalent expressions for ice viscosity in the literature: the form in (5.46) appears in [57] while [160] and [70] define the effective strain rate $\dot{\epsilon}_e$:

$$\dot{\epsilon}_e^2 = \dot{\epsilon}_{xx}^2 + \dot{\epsilon}_{yy}^2 + \dot{\epsilon}_{xx}\dot{\epsilon}_{yy} + \dot{\epsilon}_{xy}^2 + \dot{\epsilon}_{xz}^2 + \dot{\epsilon}_{yz}^2.$$

Meanwhile [162] have

$$\begin{aligned} \dot{\epsilon}_{\text{BP}}^2 &= \left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 + \left(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \right)^2 \\ &\quad + \frac{1}{2} \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right)^2 + \frac{1}{2} \left(\frac{\partial u}{\partial z} \right)^2 + \frac{1}{2} \left(\frac{\partial v}{\partial z} \right)^2. \end{aligned}$$

5.10.3 Weak form

Note: Recall the product rule

$$\begin{aligned} \nabla \cdot (fX) &= \nabla f \cdot X + f \nabla \cdot X, \text{ or} \\ f \nabla \cdot X &= \nabla \cdot (fX) - \nabla f \cdot X \end{aligned}$$

and the divergence theorem:

$$\int_{\Omega} \nabla \cdot (fX) = \int_{\partial\Omega} (fX) \cdot \mathbf{n} ds.$$

We omit discussions of function spaces; see [160], [161] and other references mentioned above for details.

To obtain the weak form, we multiply both equations in (5.45) by a *scalar* test function ψ and integrate over the domain Ω . For the first equation, we get

$$\begin{aligned} \int_{\Omega} \psi (-\nabla \cdot (2\eta \dot{\epsilon}_1) + \rho g s_x) &= 0, \\ - \int_{\Omega} \psi \nabla \cdot (2\eta \dot{\epsilon}_1) + \int_{\Omega} \psi \rho g s_x &= 0, \\ - \int_{\Omega} (\nabla \cdot (\psi 2\eta \dot{\epsilon}_1) - \nabla \psi \cdot 2\eta \dot{\epsilon}_1) + \int_{\Omega} \psi \rho g s_x &= 0, \\ - \int_{\Omega} \nabla \cdot (\psi 2\eta \dot{\epsilon}_1) + \int_{\Omega} \nabla \psi \cdot 2\eta \dot{\epsilon}_1 + \int_{\Omega} \psi \rho g s_x &= 0, \\ - \int_{\partial\Omega} (\psi 2\eta \dot{\epsilon}_1) \cdot \mathbf{n} ds + \int_{\Omega} \nabla \psi \cdot 2\eta \dot{\epsilon}_1 + \int_{\Omega} \psi \rho g s_x &= 0. \end{aligned} \tag{5.48}$$

Similarly, multiplying the second equation by ψ and integrating by parts yields

$$- \int_{\partial\Omega} (\psi 2\eta \dot{\epsilon}_2) \cdot \mathbf{n} ds + \int_{\Omega} \nabla \psi \cdot 2\eta \dot{\epsilon}_2 + \int_{\Omega} \psi \rho g s_y = 0. \tag{5.49}$$

We combine these and say that the weak form of (5.45) is

$$- \int_{\partial\Omega} (\psi 2\eta \dot{\epsilon}) \cdot \mathbf{n} ds + \int_{\Omega} \nabla \psi \cdot 2\eta \dot{\epsilon} + \int_{\Omega} \psi \rho g \nabla s = 0, \tag{5.50}$$

where $\nabla s = (s_x, s_y)$.

The first term corresponds to *Neumann and Robin boundary conditions*; it vanishes for “natural” BC $(2\eta\dot{\epsilon}) \cdot \mathbf{n} = 0$. In the basal and lateral cases this stress is nonzero. The third one corresponds to the gravitational driving stress and is replaced by a compensatory term in *verification tests* that use manufactured solutions.

5.10.4 Boundary conditions

The domain boundary consists of three disjoint parts:

1. The interface between ice and the underlying bed or (if the ice is floating) water.

At this interface PISM uses Robin BC implementing basal sliding.

2. The interface between ice and the air¹ above it.

The integral over this part of the boundary is *zero* because we assume that natural (“no stress”) boundary conditions apply. (We ignore atmospheric pressure.)

Vertical “cliffs” at grounded margins are interpreted as approximations of the very steep, but not vertical, upper surface. Following this interpretation we use natural boundary conditions at grounded lateral margins.

3. The *vertical* interface between ice and air (above sea level) or water (below sea level) at marine ice margins.

At this interface PISM uses Neumann BC corresponding to the difference between the cryostatic pressure of the ice on one side and the hydrostatic pressure of water on the other side of the interface.

In addition to this we support Dirichlet boundary conditions for *verification* and to *de-couple unknowns at ice-free locations*.

Note: Our implementation supports Dirichlet boundary conditions, but this feature is not exposed to the rest of PISM.

Unlike [57] we *do not* support “no-slip” BC at the base. This allows us to avoid Jacobian scaling tricks they needed to achieve good multigrid performance.

For each node that belongs to the Dirichlet boundary we assemble “trivial” equations

$$\begin{aligned} u &= u_0, \\ v &= v_0, \end{aligned}$$

where (u_0, v_0) are given.

The implementation avoids adding contributions from adjacent elements to residual and Jacobian entries corresponding to Dirichlet locations. Prescribed velocity values are substituted into equations that depend on them (see section 3.2 of [57] for details).

Basal boundary

The boundary condition corresponding to basal sliding is

$$2\eta\dot{\epsilon} \cdot \mathbf{n} = -\beta\mathbf{u}. \quad (5.51)$$

In the weak form (5.50) this corresponds to replacing the first term:

$$-\int_{\text{base}} (\psi 2\eta\dot{\epsilon}) \cdot \mathbf{n} ds = \int_{\text{base}} \beta\mathbf{u} ds. \quad (5.52)$$

¹ Strictly speaking the top surface of the ice may be in contact with firm or snow as well as air, but these details are not relevant here.

Here β has the same meaning as in (3.17).

Where ice is grounded β is determined as described in *Controlling basal strength*. It is assumed to be zero (corresponding to no drag) elsewhere.

The grounding line (if present) divides bottom faces of some elements into grounded parts that experience drag and floating parts that do not. This implementation uses a low-order quadrature with *many* equally-spaced points (a Newton-Cotes quadrature) to integrate over the part of the basal boundary containing the grounding line. Here β is computed at each quadrature point, depending on whether the ice is grounded or floating at its location. This is similar to the SEP3 parameterization described in [163].

Note:

- It may be a good idea to implement scaling of the β coefficient according to the fraction of an element face that is grounded, similar to SEP1 (equation 7 in [163]). An approximation of the grounded fraction for an element column can be computed using existing code in PISM.
 - In general the bottom face of an element is *not* planar and *not* parallel to the plane $z = 0$. This means that integrals over the basal boundary should be evaluated using parameterizations of element faces (i.e. as surface integrals) and *not* using 2D FEM machinery.
-

Marine margins

We assume that marine ice margins consist of *vertical* cliffs, i.e. the outward-pointing normal vector has the form $\mathbf{n} = (\cdot, \cdot, 0)$.

The Neumann boundary condition at marine margins is

$$\begin{aligned} 2\eta\dot{\epsilon}\cdot\mathbf{n} &= p_{\text{ice}} - p_{\text{water}}, \text{ where} \\ p_{\text{ice}} &= \rho g(s - z), \\ p_{\text{water}} &= \rho_w g \max(z_{\text{sea level}} - z, 0). \end{aligned} \tag{5.53}$$

In other words, this boundary condition corresponds to the difference between the cryostatic pressure of the ice on one side of the interface and the hydrostatic pressure of water on the other. The atmospheric pressure is ignored. Equation (5.53) is a generalization of equation 18 in [160].

Just like in the implementation of the basal boundary condition near grounding lines, we use a low order quadrature with many equally-spaced points to approximate integrals over element faces intersected by the sea level. This *should* improve the quality of the approximation of this boundary condition (note that the right hand side of (5.53) is continuous but not continuously differentiable).

Note: We need to evaluate the importance of the quadrature choice described above. Does using depth-dependent BC matter? (We could simplify the code and use depth-averaged BC if it does not.) Should we use lots of quadrature points?

Note: CISM 2.1 [70] uses a depth-averaged lateral boundary condition *without justification*.

The lateral BC described in [160] is equivalent to the one described here, but the implementation in Albany/FELIX (and therefore in MALI) matches the one in CISM.

Implementations in CISM and MALI *do not* use this boundary condition at grounded margins because doing so appears to produce over-estimates of the ice speed near grounded ice margins. Our experiments show the same behavior.

5.10.5 Solver implementation

Discretization

To create a non-linear algebraic system of equations approximating (5.50), we create a hexahedral mesh on the domain Ω and use Q_1 Galerkin finite elements.

Let ϕ_j be the scalar trial function associated with the node j , then the FE approximation of the solution \mathbf{u} has the form

$$\begin{aligned} \mathbf{u} &= \sum_j \phi_j \mathbf{u}_j, \\ \mathbf{v} &= \sum_j \phi_j \mathbf{v}_j. \end{aligned} \tag{5.54}$$

Then the problem is

Find $\mathbf{u}_j, \mathbf{v}_j$ ($j = 1, \dots, N$) so that

$$-\int_{\partial\Omega} (\psi_i 2 \eta \dot{\epsilon}) \cdot \mathbf{n} ds + \int_{\Omega} \nabla \psi_i \cdot 2 \eta \dot{\epsilon} + \int_{\Omega} \psi_i \rho g \nabla s = 0$$

holds for all $i = 1, \dots, N$, where N is the number of nodes in the mesh, subject to the *boundary conditions*.

As in section 3 of [57], we write the discretization of (5.50) as an algebraic system $F(U) = 0$ with Jacobian $J(U)$ and solve this nonlinear system using Newton iterations requiring approximations of δU in

$$J(U) \delta U = -F(U), \tag{5.55}$$

where

$$\begin{aligned} F(U) &= F^1(U) + F^2(U) + F^3(U), \\ F^1 &= -\int_{\partial\Omega} (\psi 2 \eta \dot{\epsilon}) \cdot \mathbf{n} ds, \\ F^2 &= \int_{\Omega} \nabla \psi \cdot 2 \eta \dot{\epsilon}, \\ F^3 &= \int_{\Omega} \psi \rho g \nabla s. \end{aligned} \tag{5.56}$$

(compare to (5.50)) and $J(U)$ is a square sparse matrix containing one row per node in the mesh and at most 54 non-zero entries per row (there are 2 unknowns per node and each node belongs to at most 8 elements forming a $3 \cdot 3 \cdot 3 = 27$ node “neighborhood”).

Residual evaluation

The residual evaluation is performed in the usual manner, by iterating over all the elements containing ice (see *Model domain and mesh structure*) and adding element contributions to the “global” residual vector.

The residual itself can be broken up into the following parts:

1. The *basal boundary* term implementing the basal drag (part of F^1)
2. The “main” part (F^2)
3. The *source term* corresponding to the driving stress (F^3)
4. The *top boundary* part (zero in actual simulations because we use the natural BC at the top boundary; can be non-zero in verification tests, part of F^1)

5. The *marine boundary* part implementing stress (Neumann) BC at the calving front (part of F^1).
6. Residual at *Dirichlet nodes*.

This decomposition makes it possible to use source terms dictated by the choice of a manufactured solution while keeping (and testing) the rest of the code.

Note: We integrate over the whole domain (Ω) below (see (5.57)) for simplicity. In actuality each integral is over the *intersection of supports of test and trial functions* appearing in the integrand.

Main residual contribution

$$\begin{aligned}
 F_{i,u}^2 &= \int_{\Omega} \nabla \psi_i \cdot 2 \eta \dot{\epsilon}_1 \\
 &= \int_{\Omega} \eta \left(\frac{\partial \psi_i}{\partial x} (4u_x + 2v_y) + \frac{\partial \psi_i}{\partial y} (u_y + v_x) + \frac{\partial \psi_i}{\partial z} u_z \right), \\
 F_{i,v}^2 &= \int_{\Omega} \nabla \psi_i \cdot 2 \eta \dot{\epsilon}_2 \\
 &= \int_{\Omega} \eta \left(\frac{\partial \psi_i}{\partial y} (2u_x + 4v_y) + \frac{\partial \psi_i}{\partial x} (u_y + v_x) + \frac{\partial \psi_i}{\partial z} v_z \right).
 \end{aligned} \tag{5.57}$$

Here $F_{i,u}$ if the contribution to the u -component of the residual at the i -th node, etc.

Driving stress contribution

$$\begin{aligned}
 F_{i,u}^3 &= \int_{\Omega} \psi_i \rho g s_x, \\
 F_{i,v}^3 &= \int_{\Omega} \psi_i \rho g s_y.
 \end{aligned} \tag{5.58}$$

Basal contribution

$$\begin{aligned}
 F_{i,u}^1 &= \int_{\text{base}} \psi_i \beta u, \\
 F_{i,v}^1 &= \int_{\text{base}} \psi_i \beta v.
 \end{aligned} \tag{5.59}$$

Marine boundary contribution

$$\begin{aligned} F_{i,u}^1 &= - \int_{\partial\Omega} \psi_i(p_{\text{ice}} - p_{\text{water}}) \mathbf{n}_x, \\ F_{i,v}^1 &= - \int_{\partial\Omega} \psi_i(p_{\text{ice}} - p_{\text{water}}) \mathbf{n}_y, \end{aligned} \quad (5.60)$$

where

$$\begin{aligned} p_{\text{ice}} &= \rho g (s - z), \\ p_{\text{water}} &= \rho_w g \max(z_{\text{sea level}} - z, 0). \end{aligned}$$

Residual at Dirichlet BC locations

$$\begin{aligned} F_{i,u} &= u - u_0, \\ F_{i,v} &= v - v_0, \end{aligned} \quad (5.61)$$

where (u_0, v_0) is the prescribed velocity.

Jacobian evaluation

We use an analytical (as opposed to approximated using finite differences or automatic differentiation) Jacobian computed using formulas listed below.

Here we focus on the derivation of the Jacobian contribution corresponding to the “main” part of the residual (F^2 , see (5.56) and (5.57)). The only other non-trivial contribution comes from the basal boundary condition.

This Jacobian contribution has four parts:

$$\begin{aligned} K_{i,j,u,u} &= \frac{\partial F_{i,u}^2}{\partial u_j}, \\ K_{i,j,u,v} &= \frac{\partial F_{i,u}^2}{\partial v_j}, \\ K_{i,j,v,u} &= \frac{\partial F_{i,v}^2}{\partial u_j}, \\ K_{i,j,v,v} &= \frac{\partial F_{i,v}^2}{\partial v_j}, \end{aligned}$$

with $F_{\cdot,\cdot}^2$ defined by (5.57).

Let $G_{i,k} = \nabla \psi_i \cdot 2 \dot{\epsilon}_k$, then

$$F_{i,u}^2 = \int_{\Omega} \eta G_{i,1}$$

and (using the product rule) we get

$$\begin{aligned} K_{i,j,u,u} &= \int_{\Omega} \eta \frac{\partial G_{i,1}}{\partial u_j} + \frac{\partial \eta}{\partial u_j} G_{i,1}, \\ K_{i,j,u,v} &= \int_{\Omega} \eta \frac{\partial G_{i,1}}{\partial v_j} + \frac{\partial \eta}{\partial v_j} G_{i,1}, \\ K_{i,j,v,u} &= \int_{\Omega} \eta \frac{\partial G_{i,2}}{\partial u_j} + \frac{\partial \eta}{\partial u_j} G_{i,2}, \\ K_{i,j,v,v} &= \int_{\Omega} \eta \frac{\partial G_{i,2}}{\partial v_j} + \frac{\partial \eta}{\partial v_j} G_{i,2}. \end{aligned} \quad (5.62)$$

The derivatives of η are computed using the chain rule:

$$\begin{aligned} \frac{\partial \eta}{\partial u_j} &= \frac{\partial \eta}{\partial \gamma} \frac{\partial \gamma}{\partial u_j}, \\ \frac{\partial \eta}{\partial v_j} &= \frac{\partial \eta}{\partial \gamma} \frac{\partial \gamma}{\partial v_j}. \end{aligned}$$

Taking derivatives of η and γ (5.46) gives

$$\begin{aligned} \frac{\partial \eta}{\partial \gamma} &= \frac{B}{2} \cdot \frac{1-n}{2n} \cdot \left(\gamma + \frac{\varepsilon_0}{2} \right)^{\frac{1-n}{2n}-1} \\ &= \eta \cdot \frac{1-n}{2n} \cdot \left(\gamma + \frac{\varepsilon_0}{2} \right)^{-1}. \end{aligned} \quad (5.63)$$

and

$$\begin{aligned} \frac{\partial \gamma}{\partial u_j} &= 2u_x \frac{\partial u_x}{\partial u_j} + 2v_y \frac{\partial v_y}{\partial u_j} + \frac{\partial u_x}{\partial u_j} v_y + u_x \frac{\partial v_y}{\partial u_j} + \frac{1}{4} \cdot 2(u_y + v_x) \left(\frac{\partial u_y}{\partial u_j} + \frac{\partial v_x}{\partial u_j} \right) \\ &\quad + \frac{1}{4} \cdot 2u_z \frac{\partial u_z}{\partial u_j} + \frac{1}{4} \cdot 2v_z \frac{\partial v_z}{\partial u_j} \\ &= 2u_x \frac{\partial \varphi_j}{\partial x} + v_y \frac{\partial \varphi_j}{\partial x} + \frac{1}{2} \frac{\partial \varphi_j}{\partial y} (u_y + v_x) + \frac{1}{2} u_z \frac{\partial \varphi_j}{\partial z}, \\ \frac{\partial \gamma}{\partial v_j} &= 2u_x \frac{\partial u_x}{\partial v_j} + 2v_y \frac{\partial v_y}{\partial v_j} + \frac{\partial u_x}{\partial v_j} v_y + u_x \frac{\partial v_y}{\partial v_j} + \frac{1}{4} \cdot 2(u_y + v_x) \left(\frac{\partial u_y}{\partial v_j} + \frac{\partial v_x}{\partial v_j} \right) \\ &\quad + \frac{1}{4} \cdot 2u_z \frac{\partial u_z}{\partial v_j} + \frac{1}{4} \cdot 2v_z \frac{\partial v_z}{\partial v_j} \\ &= 2v_y \frac{\partial \varphi_j}{\partial y} + u_x \frac{\partial \varphi_j}{\partial y} + \frac{1}{2} \frac{\partial \varphi_j}{\partial x} (u_y + v_x) + \frac{1}{2} v_z \frac{\partial \varphi_j}{\partial z}. \end{aligned} \quad (5.64)$$

The derivatives of $G_{\cdot,\cdot}$ are

$$\begin{aligned} \frac{\partial G_{i,1}}{\partial u_j} &= 4 \frac{\partial \psi_i}{\partial x} \frac{\partial \varphi_j}{\partial x} + \frac{\partial \psi_i}{\partial y} \frac{\partial \varphi_j}{\partial y} + \frac{\partial \psi_i}{\partial z} \frac{\partial \varphi_j}{\partial z}, \\ \frac{\partial G_{i,1}}{\partial v_j} &= 2 \frac{\partial \psi_i}{\partial x} \frac{\partial \varphi_j}{\partial y} + \frac{\partial \psi_i}{\partial y} \frac{\partial \varphi_j}{\partial x}, \\ \frac{\partial G_{i,2}}{\partial u_j} &= 2 \frac{\partial \psi_i}{\partial y} \frac{\partial \varphi_j}{\partial x} + \frac{\partial \psi_i}{\partial x} \frac{\partial \varphi_j}{\partial y}, \\ \frac{\partial G_{i,2}}{\partial v_j} &= 4 \frac{\partial \psi_i}{\partial y} \frac{\partial \varphi_j}{\partial y} + \frac{\partial \psi_i}{\partial x} \frac{\partial \varphi_j}{\partial x} + \frac{\partial \psi_i}{\partial z} \frac{\partial \varphi_j}{\partial z}. \end{aligned} \quad (5.65)$$

To compute $\frac{\partial \gamma}{\partial u_j}$ (5.64) and $\frac{\partial G_{\cdot \cdot}}{\partial \cdot}$ (5.65) we use FE basis expansions of u and v (5.54), which imply:

$$\begin{aligned}\frac{\partial u}{\partial u_j} &= \phi_j, \\ \frac{\partial u_x}{\partial u_j} &= \frac{\partial \phi_j}{\partial x},\end{aligned}$$

and so on.

Basal contribution to the Jacobian

$$\begin{aligned}K_{i,j,u,u} &= \int_{\text{base}} \psi_i \phi_j \left(\beta + \frac{\partial \beta}{\partial \alpha} u^2 \right), \\ K_{i,j,u,v} &= \int_{\text{base}} \psi_i \phi_j \frac{\partial \beta}{\partial \alpha} u v, \\ K_{i,j,v,u} &= \int_{\text{base}} \psi_i \phi_j \frac{\partial \beta}{\partial \alpha} v u, \\ K_{i,j,v,v} &= \int_{\text{base}} \psi_i \phi_j \left(\beta + \frac{\partial \beta}{\partial \alpha} v^2 \right).\end{aligned}\tag{5.66}$$

Here $\frac{\partial \beta}{\partial \alpha}$ is the derivative of β with respect to $\alpha = \frac{1}{2}|\mathbf{u}|^2 = \frac{1}{2}(u^2 + v^2)$ (one of the outputs of PISM's basal resistance parameterizations).

Note that

$$\begin{aligned}\frac{\partial \alpha}{\partial u_j} &= \frac{1}{2} \cdot (2u) \cdot \frac{\partial u}{\partial u_j} \\ &= u \phi_j, \\ \frac{\partial \alpha}{\partial v_j} &= v \phi_j.\end{aligned}$$

Recall (see (5.59)) that the basal sliding contribution to the residual is equal to

$$\begin{aligned}F_u^1 &= \int_{\text{base}} \psi_i \beta u, \\ F_v^1 &= \int_{\text{base}} \psi_i \beta v,\end{aligned}$$

and so the basal contribution to the Jacobian consists of partial derivatives of these with respect to u_j and v_j .

For example,

$$\begin{aligned}
K_{i,j,u,u} &= \frac{\partial \left(\int_{\text{base}} \psi_i \beta u \right)}{\partial u_j} \\
&= \int_{\text{base}} \psi_i \frac{\partial (\beta u)}{\partial u_j} \\
&= \int_{\text{base}} \psi_i \left(\beta \frac{\partial u}{\partial u_j} + \frac{\partial \beta}{\partial u_j} u \right) \\
&= \int_{\text{base}} \psi_i \left(\beta \phi_j + \frac{\partial \beta}{\partial \alpha} \cdot \frac{\partial \alpha}{\partial u_j} u \right) \\
&= \int_{\text{base}} \psi_i \left(\beta \phi_j + \frac{\partial \beta}{\partial \alpha} u^2 \phi_j \right) \\
&= \int_{\text{base}} \psi_i \phi_j \left(\beta + \frac{\partial \beta}{\partial \alpha} u^2 \right).
\end{aligned}$$

Jacobian at Dirichlet locations

The Jacobian at Dirichlet locations is set to 1. Together with the *residual at Dirichlet locations* this completes the assembly of trivial equations at these locations.

More specifically (due to the interleaved ordering of the unknowns: $u_j, v_j, u_{j+1}, v_{v+1}, \dots$) this requires setting the corresponding block of the Jacobian to the 2×2 identity matrix.

Note: It may be interesting to see if varying the scaling of Jacobian entries at Dirichlet nodes affects the condition number of the Jacobian matrix. See the variable **scaling** in the code and set

-bp_ksp_view_singularvalues

to see if it has an effect.

Iceberg elimination

As described in [69], isolated patches of ice with low basal resistance *and* patches connected only via a single node (a “hinge”) are problematic because the system (5.45) determines ice velocity up to rigid rotations and translations.

This is not a new issue: both FD and FEM solvers of the SSA stress balance require some form of iceberg elimination. We use a connected component labeling algorithm to identify patches of *floating* ice. In the FEM context this requires inspecting *elements*: two elements are considered connected if they share a boundary.

Note: We could improve this mechanism by implementing a version of the method described in [69]: instead of removing *floating ice* not connected to grounded ice (PISM’s approach) they

1. Identify all the connected patches of ice.
2. Remove patches of ice which have no mesh *nodes* with

$$\beta > \beta_{\text{threshold}}.$$

Preconditioning

Back in 2013 Brown et al [57] showed that multigrid can act as an effective preconditioner for BP stress balance solvers. However, all test cases considered in that work use periodic boundary conditions and therefore avoid all the issues related to the moving ice margin present in complete ice sheet models. Our tests suggests that a naive implementation assembling trivial equations at “ice free” nodes combined with standard geometric multigrid (coarsening in all 3 directions) is not likely to succeed and we need a different approach.

So, we use semi-coarsening in the vertical direction *even though Brown et al state that “semi-coarsening is unattractive”*. One of the arguments against semi-coarsening is the larger number of multigrid levels needed: semi-coarsening gives a smaller reduction in the number of unknowns from one level to the next (factor of 2 instead of 8 in the full multigrid approach). Our tests show that “aggressive” semi-coarsening (i.e. using coarsening factors larger than 2 and as high as 8) appears to be effective, allowing one to achieve similar reductions in the number of unknowns from one level to the next in a multigrid hierarchy.

The second argument against semi-coarsening is deeper: spatial variations in the sliding parameter β may lead to the kind of anisotropy that cannot be addressed by coarsening in the vertical direction (see chapter 7 in [164] for a discussion). Still, we are encouraged by results published by Tuminaro et al [69], who used a similar mesh structure and an approach equivalent to using geometric multigrid with semi-coarsening in the vertical direction for the finer part of the hierarchy and algebraic multigrid for coarser levels.²

Inspired by [69], we use *geometric multigrid* to build a mesh hierarchy with the coarsest level containing a small number (2 or 3) of vertical levels combined with *algebraic multigrid* as a preconditioner for the solver on the coarsest level. (Semi-coarsening in the vertical direction cannot reduce the number of unknowns in the x and y directions and the coarsest problem is likely to be too large for the redundant direct solver, which is PETSc’s default.)

In addition to this, we follow [57] in ordering unknowns so that columns are contiguous (and u and v are interleaved), allowing ILU factorization to compute a good approximation of ice velocities in areas where SIA applicable.

Vertical grid sizes compatible with coarsening

Ideally, the coarsest mesh in the hierarchy should have 2 nodes in the z direction, i.e. be *one element thick*. If N is the coarsening factor, the total number of vertical levels (M_z) has to have the form

$$M_z = A \cdot N^k + 1 \quad (5.67)$$

for some positive integer A (ideally $A = 1$), so that the mesh hierarchy containing k levels will include levels with

$$A + 1, A \cdot N + 1, A \cdot N^2 + 1, \dots, A \cdot N^k + 1$$

nodes in the z direction.

This means that for a given `stress_balance.blatter.coarsening_factor` and number of multigrid levels `-bp_pc_mg_levels k` the value of M_z cannot be chosen at random.

² The code developed by [69] uses the algebraic multigrid framework *throughout*, i.e. even for the part of the hierarchy where the mesh structure allows one to use “geometric” coarsening.

Controlling using PETSc options

The PETSc SNES object solving the BP system uses the `bp_` prefix for all command-line options.

To choose a preconditioner, set

```
-bp_pc_type XXX
```

where `XXX` is the name of a preconditioner.

Run

```
pismr -stress_balance blatter [other options] -help | grep "-bp"
```

to get the list of all PETSc options controlling this solver.

To use a geometric multigrid preconditioner with N levels, set

```
-bp_pc_type mg -bp_pc_mg_levels N
```

An “aggressive” (i.e. greater than 2) coarsening factor may work well. Use `stress_balance.blatter.coarsening_factor` to set it.

See [Vertical grid sizes compatible with coarsening](#) for the discussion of the relationship between the number of vertical levels, number of multigrid levels, and the coarsening factor.

Set

```
-bp_mg_coarse_pc_type gamg
```

to use PETSc’s GAMG on the coarsest multigrid level.

Note: It would be interesting to compare different preconditioning options on the coarsest MG level (GAMG, Hypre BoomerAMG, ...).

Additional code needed to support geometric multigrid

To support the multigrid preconditioner we need to be able re-discretize the system on the mesh provided by *PETSc* in our residual and Jacobian evaluators. In general, this requires

- re-computing grid-related constants (Δx , etc) using the grid (avoid using hard-wired constants, e.g. computed using the fine grid), and
- additional code to restrict gridded inputs from the fine grid mesh to coarser meshes.

This solver does not support coarsening in horizontal directions, so gridded two-dimensional inputs can be used on all multigrid levels. The grid spacing ($\Delta x, \Delta y$) remains the same as well.

To transfer the one three-dimensional gridded input field (ice hardness), we create interpolation matrices mapping from a coarse level to the next (finer) level in the hierarchy. The transpose of this matrix is used as a restriction operator.

Parameter continuation as a recovery mechanism

As in [160], we can start with a large ε_0 , find an approximate solution, then use it as an initial guess for the next solve with a reduced ε_0 .

Note: Not implemented yet.

Model domain and mesh structure

The domain is

$$\begin{aligned}x &\in [x_{\min}, x_{\max}], \\y &\in [y_{\min}, y_{\max}], \\z &\in [z_{\min}, z_{\min} + H],\end{aligned}$$

where $[x_{\min}, x_{\max}] \times [y_{\min}, y_{\max}]$ is the “map plane” domain corresponding to the maximum ice extent, z_{\min} is the bottom ice surface elevation (equal to bed elevation where ice is grounded and determined using sea level, ice thickness, and the floatation criterion where floating) and H is the ice thickness.

Coordinates of the mesh nodes have the form

$$\begin{aligned}x_i &= x_{\min} + i \cdot \Delta x, \\y_j &= y_{\min} + j \cdot \Delta y, \\z_k &= z_{\min}(x_i, y_j) + k \cdot \frac{H(x_i, y_j)}{M_z - 1}.\end{aligned}\tag{5.68}$$

Each element’s projection onto the plane $z = 0$ is a rectangle with sides Δx and Δy , but the spacing between nodes in the vertical direction is *not* constant: each vertical column of nodes contains M_z nodes with the spacing of $H/(M_z - 1)$. This mesh structure is *exactly the same* as the one used in [57] and CISM 2.1 [70].

Supporting evolving ice extent

The ice extent changes as the model runs and the solver implementation has to allow for these changes.

To simplify the logic used to identify the interior of the ice volume and its lateral boundaries we compute the “type” of each node in the mesh. Given a threshold H_{\min} (see `stress_balance.ice_free_thickness_standard`), we say that

- an element contains ice if ice thickness at all its nodes equals to or exceeds H_{\min} ,
- a node is *interior* (within the ice) if all the elements it belongs to contain ice,
- a node is *exterior* (outside the ice volume) if no element it belongs to contains ice,
- a node that is neither interior nor exterior is a *boundary* node.

Only elements containing ice are included in the residual and Jacobian evaluation.

We prescribe zero $(0, 0)$ velocity at exterior nodes by marking them as Dirichlet BC locations.

In addition to this, we need to identify vertical faces of elements at *lateral boundaries*.

An element face is a part of the lateral boundary if all four of its nodes are *boundary* nodes.

Solver inputs and outputs

The BP solver uses the following inputs:

- basal yield stress (τ_c),
- ice thickness,
- bed elevation,
- sea level elevation,
- ice enthalpy (used to compute ice hardness)

And provides the following outputs:

- u and v components of ice velocity at the *nodes of the mesh* (saved to output files to be used as an initial guess later)
- vertically-averaged u and v (used in the mass continuity step, i.e. to update ice geometry)
- basal frictional heating (used to couple stress balance to energy conservation)

Steps performed by the solver

1. Compute ice bottom elevation.
2. Compute floatation function f ($f \leq 0$ if ice is grounded, $f > 0$ if floating).
3. Compute *node type*.
4. Compute ice hardness at the *nodes of the mesh*.
5. Call PETSc's `SNESolve()`.
6. Extract basal velocity and compute basal frictional heating.
7. Compute vertically-averaged ice velocity.

Integration with the rest of PISM

Conservation of energy

Coupling to PISM's energy balance models requires

- all 3 components of ice velocity on PISM's grid, and
- strain heating.

These are computed by using piecewise-linear interpolation in the vertical direction to put u , v on PISM's grid, after which vertical velocity w and strain heating are computed using existing code.

5.10.6 Testing and verification

Verification test XY

Exact solution

$$u = \exp(x) \sin(2\pi y)$$

$$v = \exp(x) \cos(2\pi y)$$

Domain $x \in [0, 1]$, $y \in [0, 1]$

Boundary conditions Dirichlet BC corresponding to the exact solution on all lateral boundaries ($x = 0$, $x = 1$, $y = 0$, $y = 1$). Natural BC at the top and bottom boundaries.

Comments This test uses a constant ice hardness, $n = 3$, and has no variation in the z direction. It is similar to the $x - y$ MMS verification test in [160], section 4.1 (we use Dirichlet boundary conditions along the whole lateral boundary instead of Robin conditions derived from the chosen exact solution).

The compensatory term is computed using SymPy; please see the code in `src/stressbalance/blatter/verification`.

Verification test XZ

Exact solution

$$u = \frac{2A(g\rho)^n \left((s-z)^{n+1} - H^{n+1}\right) |s_x|^{n-1} s_x}{n+1} - \frac{Hg\rho s_x}{\beta}$$

$$v = 0$$

Domain

$$x \in [-L, L],$$

$$z \in [s(x) - H, s(x)],$$

$$s(x) = s_0 - \alpha x^2.$$

Boundary conditions

- Dirichlet BC corresponding to the exact solution at $x = -L$ and $x = L$.
- Basal ($z = s(x) - H$) BC is a combination (i.e. sum) of the BC in *Basal boundary* and a compensatory term derived using the exact solution.
- Top surface ($z = s(x)$) BC is derived using the exact solution.

Comments This test

- uses a constant ice hardness,
- Glen exponent $n = 3$,
- has no variation (and is periodic) in the y direction,
- uses a constant basal resistance coefficient β .

It is similar to the $x - z$ MMS verification test [160], section 4.2 (again, we use Dirichlet BC at lateral boundaries instead of Robin conditions stated in the paper).

See [160] and the code in `src/stressbalance/blatter/verification` for details.

Verification test XZ-CFBC

This setup tests the “calving front boundary condition” (see [Marine margins](#)).

Exact solution

$$u = \frac{(\rho - \rho_w)gL}{2B\pi} \sin\left(\frac{\pi x}{L}\right)z,$$

$$v = 0$$

Domain $x \in [0, L]$, $z \in [-H, 0]$

Boundary conditions

- Dirichlet BC at $x = 0$.
- Uses the BC described in [Marine margins](#) at $x = L$.

Comments This test uses the Glen exponent of 1 (constant viscosity) and has no variation in the y direction.

The sea level is set to 0, overriding the floatation criterion to ensure that *all* the ice is submerged.

Verification test XZ-VV (van der Veen profile)

This setup tests the implementation of the basal sliding boundary condition (see [Basal boundary](#)) using the van der Veen shelf profile [165]:

$$H(x) = \left[\frac{4Cx}{Q_0} + H_0^{-4} \right]^{-\frac{1}{4}},$$

$$u(x) = \frac{Q_0}{H(x)},$$

$$v(x) = 0,$$

$$C = \left(\frac{\alpha g \rho_i}{2B} \right)^3,$$
(5.69)

where Q_0 is the flux at the left boundary and H_0 is the corresponding ice thickness.

The surface elevation s and bed elevation b are defined by

$$s(x) = \alpha H(x),$$

$$b(x) = (\alpha - 1)H(x)$$

for some positive constant α . (A free-floating shelf corresponds to $\alpha = 1 - \rho_i/\rho_w$).

Note: Functions (u, v) in (5.69) solve (5.45) *exactly* in the interior of the domain

$$0 \leq x_{\min} \leq x \leq x_{\max},$$

$$b(x) \leq z \leq s(x)$$

if the Glen exponent $n = 3$. No compensatory source term is needed.

We use a Dirichlet BC at the left boundary:

$$u(x_{\min}) = \frac{Q_0}{H_0},$$

$$v(x_{\min}) = 0$$

and a stress BC at the right boundary:

$$2\eta \dot{\epsilon} \cdot \mathbf{n}_{\text{right}} = (\alpha g \rho_i H(x), 0) \quad (5.70)$$

The stress BC at the top surface is

$$2\eta \dot{\epsilon} \cdot \mathbf{n}_{\text{top}} = \left(\frac{2BC^{\frac{4}{3}}\alpha H^6(x)}{\sqrt{C^2\alpha^2 H^{10}(x) + Q_0^2}}, 0 \right) \quad (5.71)$$

The boundary condition at the bottom surface has the form

$$2\eta \dot{\epsilon} \cdot \mathbf{n}_{\text{bottom}} = -\beta \mathbf{u},$$

$$\beta = \frac{2BC^{\frac{4}{3}}(\alpha - 1)H^7(x)}{Q_0 \sqrt{C^2(\alpha - 1)^2 H^{10}(x) + Q_0^2}}$$

5.10.7 Known issues and future work

- Eliminate “wiggles” near areas with steep surface slopes.
- Implement drag at lateral boundaries in fjords and alpine valleys.
- Implement parameter continuation.
- Couple to melange back pressure parameterizations by replacing p_{water} in *Marine margins*.

CONTRIBUTING TO PISM

Bug reports, contributions of code, documentation, and tests are always appreciated.

You will need a [GitHub](#) account and some familiarity with [Git](#)¹.

Please see [Submitting bug reports](#) for bug reporting guidelines.

Note: By submitting code, the contributor gives irrevocable consent to the redistribution and modification of the contributed source code as described in the PISM's open source license.

Contributions are preferred via pull requests to <https://github.com/pism/pism>.

1. [Fork PISM's repository](#).
2. Create a branch that will contain your changes.
3. Implement proposed changes.
 - a. Make changes to the code or documentation (or both).
 - b. Test your changes.
 - c. Add verification or regression tests (optional but **strongly encouraged**).
 - d. Update documentation, if necessary.
 - e. Update the change log `CHANGES.rst`. If your contribution contains a bug fix, please describe the bug and its effects.
4. [Create a pull request](#) and make sure to allow edits from maintainers.

Every time you push your code to [GitHub CircleCI](#) will

- build it with pedantic compiler settings, treating all compiler warnings as errors
- run `make test` in the build directory.

Please make sure all tests pass (you will get an e-mail if there was a failure).

If you are planning a large contribution we encourage you to open an issue at <https://github.com/pism/pism/issues> or e-mail us at uaf-pism@alaska.edu and interact with us frequently to ensure that your effort is well-directed.

When working on a large contribution it is essential to stay in touch with PISM's maintainers.

We urge you to use a version control system and give PISM maintainers access to a repository containing your code. We realize that your group's policy may make it impossible to put the code in a public repository. However, it is very easy to set up a private repository instead.

¹ Please see [Git introduction for PISM developers](#) for a brief introduction and [Git documentation](#) for more.

Warning: Working in isolation will lead to a waste of many person-hours of effort once your work is ready to be merged into PISM.

See sections listed below for various technical details.

6.1 Submitting bug reports

Please see the <https://github.com/pism/pism/issues> to check if someone already found a similar bug. You can post an issue there, and label it as a bug, if it is new. Alternatively, send a report by e-mail to uaf-pism@alaska.edu.

Please include the following information in **all** bug-reports and questions about particular PISM's behavior:

- the PISM version (the output of `pismr -version`),
- the **full** command necessary to reproduce the bug,
- the input files used by the run reproducing the bug,
- a description of what PISM does wrong.

When sending a PISM standard output snippet or a compilation log, please attach it as a text file instead of pasting it into the message body.

It is great if you can

- remove irrelevant command-line options from the command exhibiting the bug,
- find a quick run that exhibits the bug,
- reduce the size of files it requires, and
- provide a definitive way of checking if the bug in question is fixed.

6.2 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*
 - * Logical 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

6.2.1 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.

6.2.2 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.

6.2.3 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.

6.2.4 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
```

6.2.5 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.

6.2.6 Namespaces

Everything in PISM goes into the `pism` namespace. See the source code browser for more namespaces (roughly one per sub-system).

Notable namespaces include:

- `atmosphere`
- `bed`
- `calving`
- `energy`
- `frontalmelt`
- `hydrology`
- `ocean`
- `rheology`
- `sea_level`
- `stressbalance`
- `surface`

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.

6.2.7 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.)

Logical 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);
```

6.2.8 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 <cstdio>

#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);
```

(continues on next page)

(continued from previous page)

```

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.)

6.2.9 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.

6.3 Git introduction for PISM developers

6.3.1 Recommended Git configuration

Set name and e-mail address:

```
git config --global user.name "John Doe"
git config --global user.email johndoe@example.com
```

Do not push local branches nonexistent on upstream by default:

```
git config --global push.default simple
```

Set an editor to use when writing commit messages. For example, run

```
git config --global core.editor vi
```

to use vi.

6.3.2 Working on a new branch

This section summarizes Git commands used in a typical development workflow. A good Git GUI can save you from having to type these commands yourself but one should still know them.

- When starting work on a new feature make sure to start from the `dev` branch:

```
git checkout dev
```

When working on a bug fix for the current (released) PISM version, start from `main` instead:

```
git checkout main
```

See [Git branches](#) for more.

- Next, create and switch to a new branch:

```
git checkout -b <user-name>/<short-description>
```

where `<user-name>` is your GitHub user name and `<short-description>` is a short (two or three words) description of the topic you intend to work on.

For example:

```
git checkout -b ckhroulev/energy-balance
```

- Work on the code, documentation, etc.

- Inspect changes:

```
git status
```

- Commit changes:

- To commit all changes to files that are already in the repository:

```
git commit -a
```

- To commit particular files

```
git commit file1 file2 ...
```

- To add new files that are to be committed:

```
git add file1 file2 ...
git commit
```

- Push changes to GitHub:

```
git push -u origin ckhroulev/energy-balance
```

Push changes to your own branch or other branches dedicated to a topic you may be sharing with your collaborators.

Note: Do *not* push to `dev` or `main` directly unless you know what you are doing.

- If you started your branch from `dev` and need to use a feature that was added to `dev` since then you can run

```
git merge dev
```

to “merge” the dev branch into your branch.

Note: Please do not merge dev into your branch unless you need to: doing it too often makes the development history (`git log`) more confusing and less useful.

6.3.3 Writing better commit messages

Every commit should be accompanied by a meaningful message.

A commit message consists of a short one-line summary followed by a blank line and a free-form body of the message.

The summary should be capitalized, use the imperative mood, and should not end in a period.

The body of a commit message should explain what changes were made and why (but not *how*).

If a commit addresses a GitHub issue, please include the issue title and number in the body. Summarize the issue if its title is not descriptive enough.

6.4 Git branches

PISM development loosely follows the Git branching model described in [A successful Git branching model](#) by Vincent Driessen.¹

PISM’s repository contains two long-lived branches: `main` and `dev`.

The default branch `main` contains released code. This way users can clone <https://github.com/pism/pism> and get the latest PISM release.

The `dev` branch contains code that is ready to be included in the next release.

Current development is done in *topic branches* started from `dev`. Once a new feature or improvement is finished, tested, and documented, the `topic` branch is merged into `dev` and deleted. Each topic branch should contain changes related to one particular topic.²

If one person is responsible for working on a topic branch it works well to keep it up to date with `dev` by rebasing it on top of `dev`, effectively applying all the changes contained in it to the current state of `dev`. If rebasing is impractical one could merge `dev` into a topic branch to get access to some features that were not available when the branch was started. However, merging `dev` into a topic branch “just to stay up to date” is not a good idea since it confuses commit history.

When a released version of the code needs a fix, a “bug-fix” branch is created from the `main` branch. When the implementation of a fix is complete, the bug-fix branch is merged into `main` (and `main` is tagged to mark the new bug-fix release) and into `dev` so that the fix is included in the next major release.

In <https://github.com/pism/pism> branches are named using the name of the person responsible for the branch as a prefix. For example, `ckhroulev/pnetcdf` is the name of Constantine Khroulev’s branch containing improvements of the I/O code using `PnetCDF`.

¹ This model may not be perfect but works well for a project of PISM’s size.

² See [Fun with merges and purposes of branches](#) by Junio C Hamano for more about “topic branches.”

Note: Please commit all your changes to “topic” branches. The `main` and `dev` branches are managed by PISM developers at UAF.

6.5 Development workflow

The recommended development workflow is:

1. When starting a new project, create a topic branch starting from `dev`. If you are fixing a bug in a released version of PISM, create a topic branch starting from `main`.
2. Make changes to the code or documentation (or both).
 - a) Compile.
 - b) Fix any compilation errors and warnings. Repeat until your code compiles without warnings.
 - c) Run `make test` and fix any test failures.
 - d) Push your code to GitHub for review or to get help with b) and c).
3. Add verification or regression tests.
4. Test your code and repeat 2a–2c until all tests pass.
5. Update documentation (if necessary).
6. Update the change log `CHANGES.rst`.¹
7. Merge new features into `dev` and fixes into `main` and `dev` (or submit a pull request).

This document covers the tools and approaches we found useful for the steps listed above.

Contents

- *Development workflow*
 - *Setting up the environment*
 - *Editing source code*
 - *Compiling PISM*
 - *Debugging*
 - * *Floating point exceptions*
 - * *Issues visible in parallel runs only*
 - * *Issues caught by automatic tests*
 - * *Using clang-tidy*
 - *Writing tests*
 - * *Running tests*
 - *Editing PISM’s manual*
 - * *Listing configuration parameters*

¹ See [Keep a change log for inspiration](#).

* Listing diagnostic quantities

6.5.1 Setting up the environment

The majority of interesting PISM runs are performed on supercomputers, but we **do not** recommend using supercomputers for development.

Use a desktop (or a laptop) computer running Linux or macOS.

While you can use SSH to connect to a remote system to write, compile, and test your code, doing so will reduce your productivity when compared to using a computer you have physical access to.

Any MPI implementation would work, but we prefer to use [MPICH](#) for PISM development. This MPI implementation

- has headers that compile without warnings,
- provides type checking for pointer arguments in MPI calls, and
- does not produce “false positives” when debugging memory access with [Valgrind](#).

When working on a fix for a stubborn bug it *may* be helpful to use PETSc compiled with debugging enabled (option `--with-debugging=1`), but in our experience this is rarely needed. Optimized PETSc builds (using `--with-debugging=0`) are faster and this helps with overall productivity.

Configure PISM with debugging symbols enabled.

```
export PETSC_DIR=~/local/petsc/petsc-3.11.3/
export PETSC_ARCH=opt

CC=mpicc CXX=mpicxx cmake \
-DCMAKE_BUILD_TYPE=Debug \
-DPism_BUILD_EXTRA_EXECS=YES \
-DPism_BUILD_PYTHON_BINDINGS=YES \
-DPism_DEBUG=YES \
${pism_source_dir}
```

Table 6.1: PISM’s configuration flags for development

Flag	Meaning
<code>-DCMAKE_BUILD_TYPE=Debug</code>	Enables pedantic compiler warnings
<code>-DPism_BUILD_EXTRA_EXECS=YES</code>	Build extra testing executables (needed by some of regression test)
<code>-DPism_BUILD_PYTHON_BINDINGS=YES</code>	Build PISM’s Python bindings (used by many regression tests)
<code>-DPism_DEBUG=YES</code>	Enables extra sanity checks in PISM

6.5.2 Editing source code

Any text editor supporting C++, Python, and reStructuredText will work, but we recommend [Emacs](#).

Your editor needs to provide the ability to jump from a compiler’s error message to the relevant part of the code. In Emacs, use `M-x compile` to start a compilation and `M-x recompile` to re-run it.

An editor that can help you navigate the code and find function definitions, etc is also helpful; try an IDE such as [KDevelop](#), for example.

6.5.3 Compiling PISM

If the computer you use for development has multiple CPU cores you should tell `make` to use all of them. Run `make -j4` on a four-core laptop, for example; this will significantly speed up compilation.

To further speed up re-compiling PISM, install `ccache` and configure PISM as follows:

```
CC="ccache mpicc" CXX="ccache mpicxx" cmake ...
```

It may be helpful to use `LLD` to link PISM during development since it is a lot faster than GNU `ld`. Add the following `CMake` options to give this a try.

```
-DCMAKE_EXE_LINKER_FLAGS="-fuse-ld=lld" \
-DCMAKE_SHARED_LINKER_FLAGS="-fuse-ld=lld" \
-DCMAKE_MODULE_LINKER_FLAGS="-fuse-ld=lld"
```

6.5.4 Debugging

The first step in debugging an issue is *always* this:

find the shortest simulation using the smallest possible grid that exhibits the problematic behavior.

It does not have to be *the* shortest simulation, but it should complete (or stop because of a failure) within seconds when running on the machine used for development.

A debugger such as `GDB` or `LLDB` can be very useful.² There are many online tutorials for both.

You will need to know how to

- start a program,
- interrupt execution,
- set and remove a breakpoint,
- continue execution after stopping at a breakpoint,
- continue execution to the next line of the code,
- continue execution to the end of the current function call,
- step into a function call,
- print the value of a variable,
- print the stack trace.

This basic set of debugging skills is often sufficient.

Sometimes a failure happens in a loop that iterates over grid points and stepping through the code in a debugger is impractical. A *conditional breakpoint* would help (i.e. stop only if a condition is true), but this debugger feature is not always well supported and often significantly slows down execution.

Here's a different way to stop the code when a condition is met: add `#include <cassert>` to the top of the file (if it is not there), then add `assert(!condition);` to the place in the code where you would like to stop if `condition` is met.

For example,

```
assert(!(i == 228 and j == 146));
```

² In most cases a serial debugger is sufficient.

will stop execution at the grid point where `i == 228` and `j == 146`.

Some of the more troublesome bugs involve memory access errors (*segmentation fault* errors are often caused by these). Consider using [Valgrind](#) to detect them.

Note: Your code will run much, much slower when using Valgrind, so it is important to find a small test case reproducing the error.

Floating point exceptions

Run PISM like this

```
pismr -fp_trap -on_error_attach_debugger [other options]
```

to catch floating point exceptions (division by zero, operations with a not-a-number or infinity, square root of a negative number, etc).

Issues visible in parallel runs only

Every once in a while a bug shows up in a parallel run but *not* in an equivalent serial one. These bugs tend to be hard to fix and there is no definitive technique (or tool) that helps with this. Here are some tips, though.

- Reduce the number of processes as much as possible. Most of the time the number of processes can be reduced all the way down to 2 (the smallest truly parallel case).
- Run PISM with the option `-start_in_debugger`. This will produce a number of terminal windows with [GDB](#). You will need to *continue execution* (GDB's command `c`) in all of the windows. If PISM freezes, interrupting execution and printing the stack trace would tell you where it got stuck.

Executing commands in all the windows with GDB is tedious and error-prone. To execute a number of commands in all of them at the beginning of the run, create a file called `.gdbinit` (in the current directory) and put GDB commands there (one per line).

For example,

```
break pism::RuntimeError::RuntimeError()
continue
```

will set a breakpoint at `pism::RuntimeError::RuntimeError()` and continue execution.

- A parallel debugger such as TotalView may be helpful but requires a license. We don't have experience with it and cannot give any advice.

Issues caught by automatic tests

Every time somebody pushes changes to PISM's repository on GitHub the continuous integration system attempts to build PISM and (if it was built successfully) run a suite of tests.

It is often helpful to be able to run the same tests locally. To do this, install [Docker](#) and [CircleCI CLI](#) (command-line interface), then run

```
circleci local execute --job={job}
# where job is one of
# build-gcc build-clang
# build-clang-minimal build-gcc-minimal build-manual
```

in PISM’s source code directory.

Using clang-tidy

Clang’s tool `clang-tidy` can help one find possible portability, readability, and performance issues, resulting in cleaner, easier to maintain code.

To use it, add `-DCMAKE_EXPORT_COMPILE_COMMANDS=ON` to PISM’s CMake options. This will tell CMake to save information needed by `clang-tidy`.

Then save the script below to `tidy.sh`.

Listing 6.1: Running `clang-tidy`

```
#!/bin/bash

set -e
set -u

# PISM's build directory
build_dir=~/local/build/pism

# Extract compiler options specifying MPI's include directories
mpi=$(mpicc -show | grep -E -o -e "-I[^ ]+")

# Requested checks
checks="bugprone*,clang-analyzer*,mpi*,performance*,portability*,readability*,-readability-isolate-
-declaration"

# Run clang-tidy
clang-tidy --extra-arg=${mpi} -p ${build_dir} --checks=${checks} $@
```

This script assumes that PISM’s build directory is in `~/local/build/pism`. You may need to adjust this to suit your setup.

Now, to get the report about a source file `foo.cc`, run

```
tidy.sh foo.cc
```

The output is formatted in a way similar to compiler error messages, which makes it easy to navigate using Emacs or a similar editor.

6.5.5 Writing tests

All contributions containing new features should contain tests for the new code.³

A contribution fixing a bug should (ideally) contain a test that will ensure that it is fixed.

Add verification tests (tests comparing results to an analytical solution) whenever possible. If a verification test is not an option, consider adding a *regression* test that compares computed results to a stored output from a trusted version of the code. This will make it easier to detect a regression, i.e. an undesirable change in model results.

Here are some test writing tips:

- Make sure that a verification test uses a grid that is not square (different number of grid points in x and y directions).

³ Contributions of tests for existing code are also welcome.

- If possible, write a test that performs a number of computations along a *refinement path* and compare the computed convergence rate to the theoretical one.
- Try to include tests ensuring that x and y directions are interchangeable: in most cases flow from left to right should behave the same as from bottom towards the top, etc.

Here are two ways to do this:

- Repeat the test twice, the second time using transposed inputs, then transpose results and compare.
 - Repeat the test twice, once using a refinement path along x and the second time along y; make sure that you see the same convergence rate.
- It is good to check if the implementation preserves symmetries if the setup has any.
 - If a test uses a temporary file, make sure that it will not clash with names of files used by other tests. One easy way to do this is by generating a unique file name using `mktemp` (in Bash scripts) or `str(uuid.uuid4())` (in Python).

Python bindings make it possible to test many PISM's components in isolation from the rest of the code. See tests in `test/regression` for some examples.

Note: This manual should cover PISM's Python bindings. If you see this, please e-mail uaf-pism@alaska.edu and remind us to document them.

Running tests

Run `make test` in parallel by adding

```
export CTEST_PARALLEL_LEVEL=N
```

to your `.bashrc`. This will tell `ctest` to run N at the same time. Or run `ctest -j N` instead of `make test`.

6.5.6 Editing PISM's manual

PISM's manual is written using the `reStructuredText` markup and `Sphinx`.

See [Rebuilding PISM documentation](#) for a list of tools needed to build PISM's documentation.

When working on major edits, `sphinx-autobuild` can save you a lot of time. Run

```
make manual_autobuild
```

in the build directory to get a browser window containing PISM's manual that will stay up to date with your edits.

To generate the HTML version of the manual, run `make manual_html`; the output is saved to `doc/sphinx/html/` in your build directory.

Edit `doc/sphinx/math-definitions.tex` to add custom LaTeX commands used in formulas (one per line).

Listing configuration parameters

The list in *Configuration parameters* is generated from `src/pism_config.cdl`. Edit this file to update the list.

When documenting a sub-model, use the `:config:` role to mention a parameter. This will create a hyperlink to the complete list of parameters and ensure that all parameter names are spelled correctly.

To create a list of parameters controlling a sub-model, use the `pism-parameters` directive. For example, to list all parameters with the prefix `constants.ice.`, add this:

```
.. pism-parameters::  
:prefix: constants.ice.
```

This is the resulting list:

1. `beta_Clausius_Clapeyron` (7.9e-08 *Kelvin / Pascal*) Clausius-Clapeyron constant relating melting temperature and pressure: $\text{beta} = \frac{dT}{dP}$ [24]
2. `density` (910 *kg meter-3*) ρ_i ; density of ice in ice sheet
3. `grain_size` (1 *mm*) Default constant ice grain size to use with the Goldsby-Kohlstedt [25] flow law
4. `specific_heat_capacity` (2009 *Joule / (kg Kelvin)*) specific heat capacity of pure ice at melting point T_0
5. `thermal_conductivity` (2.1 *Joule / (meter Kelvin second)*) = $W \text{ m-1 K-1}$; thermal conductivity of pure ice

For this to work, configuration parameters should be documented in `src/pism_config.cdl` (see the `..._doc` attribute for each configuration parameter).

Listing diagnostic quantities

The list of diagnostics reported by PISM is generated by running the PISM code itself. To make sure that it is up to date, run `make` in `doc/sphinx` and commit the changes.

6.6 How do I... ?

Contents

- *How do I... ?*
 - *Create and use configuration flags and parameters*
 - *Create and use additional variables*
 - * *Creating IceModelVec instances*
 - *Read data from a file*
 - *Write data to a file*
 - *Read scalar forcing data*
 - *Read 2D forcing fields*

6.6.1 Create and use 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 and the “`_type`” defining the type. All scalar parameters have to have “`_units`” set as well.

```
pism_config:constants.standard_gravity = 9.81;
pism_config:constants.standard_gravity_doc = "acceleration due to gravity on Earth geoid";
pism_config:constants.standard_gravity_type = "scalar";
pism_config:constants.standard_gravity_units = "meter second-2";
```

- One can access these parameters using the `Config` class. `IceModel` and all classes derived from `Component` have a pointer to 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 scalar parameter, do

```
double g = m_config->get_number("constants.standard_gravity");
```

To use a flag, do

```
bool compute_age = config->get_flag("age.enabled");
```

Note:

- It is best to avoid calling `m_config->get_...()` 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.
-

6.6.2 Create and use additional variables

Creating `IceModelVec` instances

PISM uses the following classes to manage 2D and 3D fields, their I/O and metadata:

<code>IceModelVec2S</code>	scalar 2D fields
<code>IceModelVec2V</code>	vector 2D fields such as horizontal velocities; corresponds to 2 NetCDF variables
<code>IceModelVec2Int</code>	2D masks, such as the grounded/floating mask
<code>IceModelVec2T</code>	2D time-dependent fields (used to read and store forcing data)
<code>IceModelVec3</code>	scalar 3D fields (usually 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.

To create a scalar field, for example, one needs to create an instance of `IceModelVec2S` and then set its metadata:

```
// land ice thickness
IceModelVec2S ice_thickness(grid, "thk", WITH_GHOSTS, 2);
ice_thickness.set_attrs("model_state", "land ice thickness",
                       "m", "land_ice_thickness");
ice_thickness.metadata().set_number("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: required stencil width).

The `IceModelVec::setAttrs()` call sets commonly used NetCDF variable attributes seen in PISM output files:

<code>pism_intent</code>	variables that are a part of the model state of a sub-model should have <code>pism_intent</code> set to “model_state”
<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 `ice_thickness.metadata()` call above allows accessing variable metadata and adding arbitrary 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 `setAttrs()` call 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");
```

6.6.3 Read data from a file

There are at least three cases of “reading data from a file”:

1. reading a field stored in an input file on a grid matching the one used by the current run (restarting a run),
2. reading a field stored in an input file on a different grid, interpolating onto the current grid (bootstrapping), and
3. reading a field stored in a file **other** than the input file using interpolation (assuming that grids are compatible but not identical)

```
// case 1, using a file name
unsigned int time = 0;
std::string filename = "filename.nc";
ice_thickness.read(filename, time);

// case 1, using an existing File instance (file is already open)
File file(communicator, "guess_mode", filename, PISM_READONLY);
ice_thickness.read(file, time);

RegriddingFlag flag = OPTIONAL;
double default_value = 0.0;
// cases 2 and 3 (interpolation)
ice_thickness.regrid(filename, flag, default_value);

// cases 2 and 3 (interpolation) using an existing File instance
ice_thickness.regrid(file, flag, default_value);
```

When interpolating (“regridding”) a field, the `flag` specifies whether a variable is required (`flag` is `CRITICAL` or `CRITICAL_FILL_MISSING`) or optional (`flag` is `OPTIONAL` or `OPTIONAL_FILL_MISSING`). PISM will stop with an error message if a required variable is not found in an input file.

Flag values of `CRITICAL_FILL_MISSING` and `OPTIONAL_FILL_MISSING` replaces “missing” values matching the `_FillValue` attribute by the default value.

If `flag` is `OPTIONAL` or `OPTIONAL_FILL_MISSING` PISM will fill the variable with `default_value` if it was not found in the file.

6.6.4 Write data to a file

`IceModelVec::define()` will define all spatial dimensions used by a variable. However, we usually need to “prepare” a file by defining the time dimension and appending a value to the time variable.

```
File file(m_grid->com, m_config->get_string("output.format"),
          filename, PISM_READWRITE_CLOBBER, m_grid->ctx()->pio_iosys_id());

io::define_time(file, *m_grid->ctx());
io::append_time(file, *m_grid->ctx()->config(), current_time);
```

When a file is opened with the mode `PISM_READWRITE_CLOBBER`, PISM checks if this file is present overwrites if it is; to append to an existing file, use `PISM_READWRITE`. To move the file aside (appending “`~`” to the file name), use `PISM_READWRITE_MOVE`.

A newly-created file is “empty” and contains no records. The `io::append_time()` call creates a record corresponding to the `current_time` (in seconds).

To write a field to an already “prepared” file, call

```
precip.write("filename.nc");
// or, if the file is already open and a File instance is available:

precip.write.(file);
```

6.6.5 Read scalar forcing data

PISM uses instances of the `ScalarForcing` class to read scalar forcing data; please see `pism::surface::Delta_T` for an example.

6.6.6 Read 2D forcing fields

PISM uses instances of the `IceModelVec2T` class to read 2D forcing fields that vary in time; please see `pism::surface::Given` for an example.

CHAPTER
SEVEN

AUTHORSHIP

Copyright © 2004 – 2021 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
Ronja Reese (PIK)	sub-shelf mass balance, ice shelf numerics, bug fixes
Julien Seguinot (ETH)	bug fixes, temperature index model
Ricarda Winkelmann (PIK)	Antarctica processes, coupling, and modeling
Florian Ziemen (MPI)	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] 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.
- [4] 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.
- [5] Ronald B Smith and Idar Barstad. A linear theory of orographic precipitation. *Journal of the Atmospheric Sciences*, 61(12):1377–1391, 2004.
- [6] Ronald B Smith, Idar Barstad, and Laurent Bonneau. Orographic precipitation and oregon’s climate transition. *Journal of the Atmospheric Sciences*, 62(1):177–191, 2005.
- [7] C. Ritz. EISMINT Intercomparison Experiment: Comparison of existing Greenland models. 1997. URL: <http://homepages.vub.ac.be/~phuybrec/eismint/greenland.html>.
- [8] 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.
- [9] 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.
- [10] A Beckmann and H Goosse. A parameterization of ice shelf-ocean interaction for climate models. *Ocean Modelling*, 5(2):157–170, 2003. URL: <https://linkinghub.elsevier.com/retrieve/pii/S1463500302000197>.
- [11] 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.
- [12] Hartmut H. Hellmer, Stanley S. Jacobs, and Adrian Jenkins. *Oceanic erosion of a floating Antarctic glacier in the Amundsen Sea*. American Geophysical Union, 1998.
- [13] Dirk Olbers and Hartmut Hellmer. A box model of circulation and melting in ice shelf caverns. *Ocean Dynamics*, 60(1):141–153, 2010.
- [14] HH Hellmer and DJ Olbers. A two-dimensional model for the thermohaline circulation under an ice shelf. *Antarctic Science*, 1(04):325–336, 1989.
- [15] E. L. Lewis and R. G. Perkin. Ice pumps and their rates. *Journal of Geophysical Research: Oceans*, 91(C10):11756–11762, 1986. doi:10.1029/JC091iC10p11756.

- [16] Ronja Reese, Torsten Albrecht, Matthias Mengel, Xylar Asay-Davis, and Ricarda Winkelmann. Antarctic sub-shelf melt rates via pico. *The Cryosphere*, 12(6):1969–1985, 2018. URL: <https://www.the-cryosphere.net/12/1969/2018/>, doi:10.5194/tc-12-1969-2018.
- [17] J. Krug, G. Durand, O. Gagliardini, and J. Weiss. Modelling the impact of submarine frontal melting and ice mélange on glacier dynamics. *The Cryosphere*, 9(3):989–1003, may 2015. doi:10.5194/tc-9-989-2015.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] J. Seguinot and I. Rogozhina. Daily temperature variability predetermined by thermal conditions over ice sheet surfaces. *J. Glaciol.*, 2014. doi:10.3189/2014JoG14J036.
- [22] A. Payne and others. Results from the EISMINT model intercomparison: the effects of thermomechanical coupling. *J. Glaciol.*, 153:227–238, 2000.
- [23] R. Hock. Glacier melt: a review of processes and their modelling. *Prog. Phys. Geog.*, 29(3):362–391, 2005.
- [24] M. Lüthi, M. Funk, A. Iken, S. Gogineni, and M. Truffer. Mechanisms of fast flow in Jakobshavn 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.
- [25] D. L. Goldsby and D. L. Kohlstedt. Superplastic deformation of ice: experimental observations. *J. Geophys. Res.*, 106(M6):11017–11030, 2001.
- [26] S. Balay and others. PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.15, Argonne National Laboratory, 2021.
- [27] 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.
- [28] 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://unesdoc.unesco.org/images/0019/001925/192525e.pdf>.
- [29] 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.
- [30] 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.
- [31] A. C. Fowler. *Mathematical Models in the Applied Sciences*. Cambridge Univ. Press, 1997.
- [32] I. Joughin, M. Fahnestock, S. Ekholm, and R. Kwok. Balance velocities of the Greenland ice sheet. *Geophysical Research Letters*, 24(23):3045–3048, 1997.
- [33] E. Bueler, C. S. Lingle, J. A. Kallen-Brown, D. N. Covey, and L. N. Bowman. Exact solutions and verification of numerical models for isothermal ice sheets. *J. Glaciol.*, 51(173):291–306, 2005. doi:10.3189/172756505781829449.
- [34] P. Huybrechts and others. The EISMINT benchmarks for testing ice-sheet models. *Ann. Glaciol.*, 23:1–12, 1996.
- [35] 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.
- [36] R. Greve. A continuum–mechanical formulation for shallow polythermal ice sheets. *Phil. Trans. Royal Soc. London A*, 355:921–974, 1997.

- [37] 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.
- [38] H. Blatter. Velocity and stress fields in grounded glaciers: a simple algorithm for including deviatoric stress gradients. *J. Glaciol.*, 41(138):333–344, 1995.
- [39] 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](https://doi.org/10.1029/2002JB002329).
- [40] 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](https://doi.org/10.1142/S0218202510004180).
- [41] K. Hutter. *Theoretical Glaciology*. D. Reidel, 1983.
- [42] M. Weis, R. Greve, and K. Hutter. Theory of shallow ice shelves. *Continuum Mech. Thermodyn.*, 11(1):15–50, 1999.
- [43] 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.
- [44] 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.
- [45] C. Schoof. A variational approach to ice stream flow. *J. Fluid Mech.*, 556:227–251, 2006.
- [46] W. S. B. Paterson. *The Physics of Glaciers*. Pergamon, 3rd edition, 1994.
- [47] A. J. Payne and D. J. Baldwin. Analysis of ice-flow instabilities identified in the EISMINT intercomparison exercise. *Ann. Glaciol.*, 30:204–210, 2000.
- [48] 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.
- [49] 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.
- [50] M. Truffer and K. Echelmeyer. Of isbrae and ice streams. *Ann. Glaciol.*, 36(1):66–72, 2003.
- [51] 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.
- [52] 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.
- [53] 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](https://doi.org/10.1073/pnas.1205385109).
- [54] 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.
- [55] 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](https://doi.org/10.1093/qjmam/hbp025).
- [56] R. Greve and H. Blatter. *Dynamics of Ice Sheets and Glaciers*. Advances in Geophysical and Environmental Mechanics and Mathematics. Springer, 2009.
- [57] Jed Brown, Barry Smith, and Aron Ahmadia. Achieving textbook multigrid efficiency for hydrostatic ice sheet flow. *SIAM J. Sci. Comp.*, 35(2):B359–B375, 2013.
- [58] I. Joughin, W. Abdalati, and M. Fahnestock. Large fluctuations in speed on Greenland’s Jakobshavn Isbrae glacier. *Nature*, 432(23):608–610, 2004.

- [59] D. M. Holland, R. H. Thomas, B. de Young, M. H. Rignot, and B. Lyberth. Acceleration of Jakobshavn Isbrae triggered by warm subsurface ocean waters. *Nature Geoscience*, 1:659–664, 2008. doi:10.1038/ngeo316.
- [60] 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.
- [61] 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.
- [62] R. Bindschadler 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.
- [63] 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.
- [64] 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.
- [65] 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: <https://www.igsoc.org/journal/58/208/t11J217.pdf>, doi:10.3189/2012JoG11J217.
- [66] 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.
- [67] 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.
- [68] D. R. MacAyeal, V. Rommelaeere, 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.
- [69] R. Tuminaro, Mauro Perego, I. Tezaur, A. Salinger, and Stephen Price. A matrix dependent/algebraic multigrid approach for extruded meshes with applications to ice sheet modeling. *SIAM Journal on Scientific Computing*, 38(5):C504–C532, 2016. doi:10.1137/15M1040839.
- [70] William H. Lipscomb, Stephen F. Price, Matthew J. Hoffman, Gunter R. Leguy, Andrew R. Bennett, Sarah L. Bradley, Katherine J. Evans, Jeremy G. Fyke, Joseph H. Kennedy, Mauro Perego, and others. Description and evaluation of the Community Ice Sheet Model (CISM) v2.1. *Geoscientific Model Development*, 2019. doi:10.5194/gmd-12-387-2019.
- [71] K. W. Morton and D. F. Mayers. *Numerical Solutions of Partial Differential Equations: An Introduction*. Cambridge University Press, 2nd edition, 2005.
- [72] Matthew J. Hoffman, Mauro Perego, Stephen F. Price, William H. Lipscomb, Tong Zhang, Douglas Jacobsen, Irina Tezaur, Andrew G. Salinger, Raymond Tuminaro, and Luca Bertagna. MPAS-Albany land ice (MALI): a variable-resolution ice sheet model for Earth system modeling using Voronoi grids. *Geoscientific Model Development*, 2018. doi:10.5194/gmd-11-3747-2018.
- [73] Stanley C. Eisenstat and Homer F. Walker. Choosing the forcing terms in an inexact newton method. *SIAM Journal on Scientific Computing*, 17(1):16–32, jan 1996. doi:10.1137/0917003.
- [74] Irina K. Tezaur, Raymond S. Tuminaro, Mauro Perego, Andrew G. Salinger, and Stephen F. Price. On the scalability of the albany/FELIX first-order stokes approximation ice sheet solver for large-scale simulations of the Greenland and antarctic ice sheets. *Procedia Computer Science*, 51:2026–2035, 2015. doi:10.1016/j.procs.2015.05.467.
- [75] A. Aschwanden and H. Blatter. Mathematical modeling and numerical simulation of polythermal glaciers. *J. Geophys. Res.*, 2009. F01027. doi:10.1029/2008JF001028.
- [76] W. S. B. Paterson and W. F. Budd. Flow parameters for ice sheet modeling. *Cold Reg. Sci. Technol.*, 6(2):175–177, 1982.

- [77] 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.
- [78] E. Bueler and J. Brown. On exact solutions and numerics for cold, shallow, and thermocoupled ice sheets. preprint \texttt{arXiv:physics/0610106}, 2006.
- [79] 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.
- [80] K. M. Cuffey and W. S. B. Paterson. *The Physics of Glaciers*. Elsevier, 4th edition, 2010.
- [81] 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.
- [82] 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.
- [83] 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.
- [84] 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.
- [85] 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.
- [86] 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.
- [87] 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.
- [88] Jonathan H. Tomkin. Coupling glacial erosion and tectonics at active orogens: a numerical modeling study. *Journal of Geophysical Research: Earth Surface*, 112(F2):, 2007. URL: <https://agupubs.onlinelibrary.wiley.com/doi/full/10.1029/2005JF000332>, doi:10.1029/2005JF000332.
- [89] 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: <https://www.the-cryosphere.net/6/273/2012/>.
- [90] 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: <https://agupubs.onlinelibrary.wiley.com/doi/full/10.1002/2016GL067695>, doi:10.1002/2016GL067695.
- [91] Rémy Mercenier, Martin P. Lüthi, and Andreas Vieli. Calving relation for tidewater glaciers based on detailed stress field analysis. *The Cryosphere*, 12(2):721–739, feb 2018. doi:10.5194/tc-12-721-2018.
- [92] T. Albrecht and A. Levermann. Fracture field for large-scale ice dynamics. *J. Glaciol.*, 58(207):165–176, 2012. doi:10.3189/2012JoG11J191.
- [93] Jean Lemaitre. Phenomenological aspects of damage. In *A course on damage mechanics*, pages 1–37. Springer, 1996.
- [94] CP Borstad, E Rignot, J Mouginot, and MP Schodlok. Creep deformation and buttressing capacity of damaged ice shelves: theory and application to larsen c ice shelf. *The Cryosphere*, 7(6):1931–1947, 2013.
- [95] T. Albrecht and A. Levermann. Fracture-induced softening for large-scale ice dynamics. *The Cryosphere*, 8(2):587–605, 2014. URL: <https://www.the-cryosphere.net/8/587/2014/>, doi:10.5194/tc-8-587-2014.

- [96] Chris Borstad, Ala Khazendar, Bernd Scheuchl, Mathieu Morlighem, Eric Larour, and Eric Rignot. A constitutive framework for predicting weakening and reduced buttressing of ice shelves based on observations of the progressive deterioration of the remnant Larsen B ice shelf. *Geophysical Research Letters*, 43(5):2027–2035, 2016.
- [97] 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](https://doi.org/10.1029/2009JF001405).
- [98] 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.
- [99] 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](https://doi.org/10.1016/j.quascirev.2013.08.011).
- [100] 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.
- [101] 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](https://doi.org/10.3189/2014JoG13J093).
- [102] 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](https://doi.org/10.5194/tc-4-605-2010).
- [103] G. K. C. Clarke. Subglacial processes. *Annu. Rev. Earth Planet. Sci.*, 33:247–276, 2005. doi:[10.1146/annurev.earth.33.092203.122621](https://doi.org/10.1146/annurev.earth.33.092203.122621).
- [104] 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 (ISMIP HEINO). *J. Glaciol.*, 56(197):371–383, 2010.
- [105] C. Schoof. Variational methods for glacier flow over plastic till. *J. Fluid Mech.*, 555:299–320, 2006.
- [106] Lucas K Zoet and Neal R Iverson. A slip law for glaciers on deformable beds. *Science*, 368(6486):76–78, 2020.
- [107] P Fretwell, Hamish D Pritchard, David G Vaughan, JL Bamber, NE Barrand, R Bell, C Bianchi, RG Birmingham, DD Blankenship, G Casassa, and others. Bedmap2: improved ice bed, surface and thickness datasets for Antarctica. *The Cryosphere*, 7:375–393, 2013.
- [108] Torsten Albrecht, Ricarda Winkelmann, and Anders Levermann. Glacial-cycle simulations of the Antarctic ice sheet with the parallel ice sheet model (pism)—part 1: boundary conditions and climatic forcing. *Cryosphere*, 14(2):599–632, 2020.
- [109] 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.
- [110] 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](https://doi.org/10.5194/gmd-8-1613-2015).
- [111] D Pollard and RM DeConto. A simple inverse method for the distribution of basal sliding coefficients under ice sheets, applied to Antarctica. *The Cryosphere*, 6(5):953, 2012.
- [112] 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.
- [113] 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.
- [114] 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.

- [115] 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.
- [116] 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.
- [117] 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.
- [118] JG Cogley, R Hock, LA Rasmussen, AA Arendt, A Bauder, RJ Braithwaite, P Jansson, G Kaser, M Möller, L Nicholson, and others. Glossary of glacier mass balance and related terms. *IHP-VII technical documents in hydrology*, 86:965, 2011.
- [119] Yun Xu, Eric Rignot, Ian Fenty, Dimitris Menemenlis, and M. Mar Flexas. Subaqueous melting of Store Glacier, west Greenland from three-dimensional, high-resolution numerical modeling and ocean observations. *Geophysical Research Letters*, 40(17):4648–4653, 2013. doi:10.1002/grl.50825.
- [120] Andy Aschwanden, Mark A Fahnestock, Martin Truffer, Douglas J Brinkerhoff, Regine Hock, Constantine Khroulev, Ruth Mottram, and S Abbas Khan. Contribution of the greenland ice sheet to sea level over the next millennium. *Science Advances*, 5(6):eaav9396, 2019.
- [121] E. Rignot, Y. Xu, D. Menemenlis, J. Mouginot, B. Scheuchl, X. Li, M. Morlighem, H. Seroussi, M. van den Broeke, I. Fenty, C. Cai, L. An, and B. de Fleurian. Modeling of ocean-induced ice melt rates of five west greenland glaciers over the past two decades. *Geophysical Research Letters*, 43(12):6374–6382, 2016. doi:10.1002/2016GL068784.
- [122] 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.
- [123] 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.
- [124] 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>.
- [125] Ed Bueler. Lessons from the short history of ice sheet model intercomparison. *The Cryosphere Discussions*, 2:399–412, 2008. URL: <https://www.the-cryosphere-discuss.net/2/399/2008/>.
- [126] P. Halfar. On the dynamics of the ice sheets 2. *J. Geophys. Res.*, 88(C10):6043–6051, 1983.
- [127] R. C. A. Hindmarsh. Thermoviscous stability of ice-sheet flows. *J. Fluid Mech.*, 502:17–40, 2004.
- [128] R. C. A. Hindmarsh. Stress gradient damping of thermoviscous ice flow instabilities. *J. Geophys. Res.*, 2006. doi:10.1029/2005JB004019.
- [129] 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.
- [130] A. J. Payne and P. W. Dongelmans. Self-organization in the thermomechanical flow of ice sheets. *J. Geophys. Res.*, 102(B6):12219–12233, 1997.
- [131] F. Pattyn and twenty others. Benchmark experiments for higher-order and full Stokes ice sheet models (ISMIP-HOM). *The Cryosphere*, 2:95–108, 2008.
- [132] 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: <https://www.the-cryosphere.net/2/67/2008/>.
- [133] 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.

- [134] F. Pattyn, C. Schoof, L. Perichon, and 15 others. Results of the Marine Ice Sheet Model Intercomparison Project, MISMIP. *The Cryosphere*, 6:573–588, 2012. doi:[10.5194/tc-6-573-2012](https://doi.org/10.5194/tc-6-573-2012).
- [135] F. Pattyn, L. Perichon, G. Durand, and 25 others. Grounding-line migration in plan-view marine ice-sheet models: results of the ice2sea MISMIP3d intercomparison. *J. Glaciol.*, 59(215):410–422, 2013.
- [136] Frank Pattyn and Tony Payne. ISMIP-HOM Ice Sheet Model Intercomparison Project: benchmark experiments for Higher-Order ice sheet Models. 2008. URL: <http://homepages.ulb.ac.be/~fpattyn/ismip/>.
- [137] C. Schoof. Marine ice-sheet dynamics. Part 1. The case of rapid sliding. *J. Fluid Mech.*, 573:27–55, 2007.
- [138] 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](https://doi.org/10.1029/2008JF001227).
- [139] 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: <https://science.sciencemag.org/content/320/5877/781>, doi:[10.1126/science.1153288](https://doi.org/10.1126/science.1153288).
- [140] 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.
- [141] 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](https://doi.org/10.1016/j.quascirev.2012.05.004).
- [142] 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.
- [143] Ed Bueler, Constantine Khroulev, Andy Aschwanden, Ian Joughin, and Ben E. Smith. Modeled and observed fast flow in the Greenland ice sheet. submitted, 2009.
- [144] 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](https://doi.org/10.1029/2011JF002140).
- [145] 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](https://doi.org/10.1073/pnas.1017313108).
- [146] I. Joughin. Ice-sheet velocity mapping: a combined interferometric and speckle-tracking approach. *Ann. Glaciol.*, 34:195–201, 2002.
- [147] 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](https://doi.org/10.5194/tc-7-987-2013).
- [148] 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.
- [149] 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. doi:[10.1029/2000GL012052](https://doi.org/10.1029/2000GL012052).
- [150] P.J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, New Mexico, 1998.
- [151] Pieter Wesseling. *Principles of Computational Fluid Dynamics*. Springer-Verlag, 2001.
- [152] 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](https://doi.org/10.1017/jfm.2012.545).
- [153] R. Sayag, S. S. Pegler, and M. G. Worster. Floating extensional flows. *Physics of Fluids*, 2012. doi:[10.1063/1.4747184](https://doi.org/10.1063/1.4747184).

- [154] C. R. Bentley. Glaciological studies on the Ross Ice Shelf, Antarctica, 1973–1978. *Antarctic Research Series*, 42(2):21–53, 1984.
- [155] 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.
- [156] 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.
- [157] 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: <https://www.earth-syst-sci-data.net/2/247/2010/>, doi:10.5194/essd-2-247-2010.
- [158] 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.
- [159] E. Bueler. An exact solution to the temperature equation in a column of ice and bedrock. preprint \textttt arXiv:0710.1314, 2007.
- [160] Irina K. Tezaur, Mauro Perego, Andrew G. Salinger, Raymond S. Tuminaro, and Stephen F. Price. Al-bany/FELIX: a parallel, scalable and robust, finite element, first-order Stokes approximation ice sheet solver built for advanced analysis. *Geoscientific Model Development*, 2015. doi:10.5194/gmd-8-1197-2015.
- [161] Mauro Perego, Max Gunzburger, and John Burkardt. Parallel finite-element implementation for higher-order ice-sheet models. *Journal of Glaciology*, 58(207):76–88, 2012. doi:10.3189/2012JoG11J063.
- [162] John K. Dukowicz, Stephen F. Price, and William H. Lipscomb. Consistent approximations and boundary conditions for ice-sheet dynamics from a principle of least action. *Journal of Glaciology*, 56(197):480–496, 2010. doi:10.3189/002214310792447851.
- [163] H. Seroussi, M. Morlighem, E. Larour, E. Rignot, and A. Khazendar. Hydrostatic grounding line parameterization in ice sheet models. *The Cryosphere*, 8(6):2075–2087, 2014. doi:10.5194/tc-8-2075-2014.
- [164] William L. Briggs, Van Emden Henson, and Steve F. McCormick. *A multigrid tutorial*. SIAM, 2000.
- [165] C. J. van der Veen. *Fundamentals of Glacier Dynamics*. CRC Press, 2nd edition, 2013.
- [166] John C. Strikwerda. *Finite Difference Schemes and Partial Differential Equations*. Wadsworth, Pacific Grove, California, 1989.
- [167] Arne Foldvik and Thor Kvinge. Conditional instability of sea water at the freezing point. In *Deep Sea Research and Oceanographic Abstracts*, volume 21, 169–174. Elsevier, 1974.
- [168] Q. Wang, S. Danilov, D. Sidorenko, R. Timmermann, C. Wekerle, X. Wang, T. Jung, and J. Schröter. The Finite Element Sea ice-Ocean Model (FESOM): formulation of an unstructured-mesh ocean general circulation model. *Geoscientific Model Development Discussions*, 6:3893–3976, July 2013. doi:10.5194/gmdd-6-3893-2013.
- [169] D. Jenssen. A three-dimensional polar ice-sheet model. *J. Glaciol.*, 18:373–389, 1977.