CISM 2.0.5 Documentation

Stephen Price[1], William Lipscomb[1], Matthew Hoffman[1],
Magnus Hagdorn[2], Ian Rutt[3], Tony Payne[4], Felix Hebeler[5], Joseph H. Kennedy [6]

August 28, 2015

[1]Group T-3, Fluid Dynamics and Solid Mechanics, Los Alamos National Laboratory
[2]School of GeoSciences, University of Edinburgh
[3]Department of Geography, Swansea University
[4]School of Geographical Sciences, University of Bristol
[5]Department of Geography, University of Zurich (original affiliation while working on project)
[6]Oak Ridge National Laboratory

CISM2 Code Developers

Erin Barker (Los Alamos National Laboratory*), Tim Bocek (University of Montana, Missoula*), Josh Campbell (University of Montana, Missoula), Katherine J. Evans (Oak Ridge National Laboratory), Jeremy Fyke (Los Alamos National Laboratory*), Glen Granzow (University of Montana, Missoula), Magnus Hagdorn (School of GeoSciences, University of Edinburgh), Brian Hand (University of Montana, Missoula*), Felix Hebeler (University of Zurich*), Matthew Hoffman (Los Alamos National Laboratory), Jesse Johnson (University of Montana, Missoula), Irina Kalashnikova (Sandia National Laboratories), Joseph H. Kennedy (Oak Ridge National Laboratory), Jean-Francois Lemieux (New York University*), William Lipscomb (Los Alamos National Laboratory), Daniel Martin (Lawrence Berkeley National Laboratory), Jeffrey A. Nichols (Oak Ridge National Laboratory), Ryan Nong (Sandia National Laboratories*), Matthew R. Norman (Oak Ridge National Laboratory), Tony Payne (University of Bristol), Stephen Price (Los Alamos National Laboratory), Doug Ranken (Los Alamos National Laboratory), Ian Rutt (Department of Geography, Swansea University), William Sacks (National Center for Atmospheric Research), Andrew Salinger (Sandia National Laboratories), James B. White III (Oak Ridge National Laboratory*), Jon Wolfe (National Center for Atmospheric Research*), Patrick Worley (Oak Ridge National Laboratory), Timothy Wylie (University of Montana, Missoula*),

** author's original affiliation while working on project*

Funding

The development of CISM has been supported by the U.K. National Environmental Research Council (through the Centre for Polar and Ocean Modelling), the U.S. National Science Foundation (through the Office of Polar Programs), and the U.S. Department of Energy (through the Office of Science, Biological and Environmental Research and Advanced Scientific Computing Research Programs).

Contents

Part I

User Documentation

Chapter 1

Introduction and Overview

1.1 Introduction

CISM, the Community Ice Sheet Model, originates from the Glimmer and Glimmer–CISM projects (Rutt et al., 2009)[1]. The current name reflects the project's evolution from a stand-alone ice sheet model to a fully supported, coupled component of the Community Earth System Model, or CESM. CISM is a numerical model—a collection of software libraries, utilities and drivers— used to simulate ice sheet evolution. CISM is modular in design and coded almost entirely in standards-compliant Fortran 95. It currently supports two different dynamical cores ("dycores"), which solve the equations for conservation of mass, energy, and momentum. As with previous versions of Glimmer and Glimmer-CISM, the current version of CISM supports a serial, shallow-ice representation of ice dynamics. New with CISM2 are support for "higher-order" ice dynamics, scalable parallelism, and software links for coupling to modern, robust, C++ based, third-party solver libraries.

1.2 Overview

CISM consists of several components:

- **cism_driver:** the high-level driver (i.e., the executable) that is used to run the ice sheet model. Unlike the drivers in previous versions of Glimmer and Glimmer-CISM, cism_driver is used to run the code in all model configurations (e.g., for idealized test cases with simplified climate forcing and for model runs based on realistic geometries and climate forcing data).

- **Glide:** the dynamical core based on shallow-ice dynamics. This component is responsible for solving the governing conservation equations and determining ice velocities, internal ice temperature, and ice geometry evolution (see Chapter 4). Apart from minor changes, this is the same shallow-ice dynamical core used by Glimmer and Glimmer-CISM.

- **Glissade:** the dynamical core based on a first-order-accurate approximation of the Stokes equations for ice flow. This dycore, like Glide, solves the governing conservation equations. Unlike Glide, Glissade is fully parallel in order to take advantage of modern, multi-processor, high-performance architectures (see Chapter 6).

- **Glint:** the climate model interface. Glint allows the core ice sheet model to be coupled to a variety of global climate models, or indeed any source of time-varying climate data on a lat-lon grid.

[1] *Glimmer* was originally an acronym for GENIE Land Ice Model with Multiply Enabled Regions, reflecting the project's origin within the GENIE (Grid ENabled Integrated Earth-system) model.

- **Test cases:** idealized test cases for the Glide and Glissade dynamical cores and for the Glint climate interface. These are used to (1) confirm that the model is working as expected and (2) provide a range of simple model configurations from which new users can learn about model options and create their own configurations (see Chapter 8).

- **Shared code:** a number of modules shared by different parts of the code. Examples include modules for defining derived types, physical constants, and model parameters, and modules that handle parsing of the configuration file and data input/output (I/O), as discussed below.

Each component is configured using a configuration file (*.config) similar to Windows `.ini` files. At run-time, the model configuration is written to a log file.

1D, 2D, and 3D data are written and read to and from netCDF files using the CF (Climate-Forecast) metadata convention[2]. NetCDF is a scientific data format for storing multidimensional data in a platform- and language-independent binary format. The CF conventions specify the metadata used to describe the file contents. Many programs (e.g., Python, Matlab, OpenDX, Ferret, and IDL) can process and visualize netCDF data.

[2]http://cfconventions.org/

Chapter 2

Installing CISM

2.1 Getting and Installing CISM

CISM[1] is a relatively complex system of libraries and programs which build on other libraries. This section documents how to download and install CISM and its prerequisites. Many common problems and questions can be addressed using the user discussion boards[2] . A CISM users mailing list is also available and can be subscribed to by sending an email to `cism-users@googlegroups.com`[3]. Please report unresolved problems using the bug reporting facility at the CISM Github website[4].

CISM is distributed as source code, and a reasonably complete build environment is therefore required to compile the model. For UNIX and LINUX based systems (including Mac), the CMake build system is used to build the model. Sample build scripts for a number of standard architectures are included, as are working build scripts for a number of large-scale, high-performance-computing architectures (e.g., *Yellowstone* (CISL), *Titan* (OLCF), and *Hopper* (NERSC)).

There are two ways to get the source code:

1. Download[5] a *released* version of the code as an archive (.zip or .tar.gz file).

2. Clone the code from the CISM Github repository[6] using the following command: `git clone https://github.com/CISM/cism.git` (It is also possible to clone the repository using the SSH protocol if you have an SSH keypair generated on your computer and attached to your GitHub account. See the CISM Github repository webpage and Github's help pages for more information.)

For beginners, downloading a zip archive of the latest release tag is recommended. More experienced users may want to download directly from the repository, as it will make updating the code easier in the future.

In either case, a Fortran90 compiler is required. Other software dependencies include the netCDF library (used for data I/O) and a Python distribution (used to analyze dependencies and to automatically generate parts of the code) with a number of specific Python modules. Users who want to run the code in parallel will need to install MPI, and users who want access to the *Trilinos* solver library will need to download and build *Trilinos*, and link to it when

[1]`http://oceans11.lanl.gov/cism/`

[2]`http://forum.cgd.ucar.edu/forums/ice-sheet-modeling-cism`

[3]In order to subscribe from a non-Google email address, you should first make sure to completely log out from any Google sites (e.g., Gmail) before sending your request. If you do not, it will automatically try to associate your request with your Gmail account instead.

[4]`https://github.com/CISM/cism/issues`

[5]`https://github.com/CISM/cism/releases`

[6]`https://github.com/CISM/cism`

building CISM. Finally, you will need CMake and Gnu Make to compile the code and link to the various third-party libraries.

If you have not done so already, clone a tagged version of CISM or download an archive of the code, as noted above. Store the code or the unzipped/untarred archive in a location of your choice. More detailed build instructions, including instructions for the installation of supporting software, are given below.

2.2 Installing Supporting Software for Basic (Serial) CISM

Because the build process can be fairly complicated, we describe it in detail below, relying on the use of a package manager to handle many of the standard software dependencies. For each step we give specific instructions for both Mac OS X using MacPorts (in red boxes) and Linux (in blue boxes). For the latter, we have verified that the instructions work for Ubuntu 12.10 and 14.04 LTS. For different but related systems, hopefully these instructions can be used as a guide.

CISM can be installed in either a serial or parallel configuration. The parallel mode allows the model to be run on multiple processors which can greatly speed up execution. This is a common configuration to use on supercomputing clusters, but can also be convenient on modern desktops and laptops which often have four or more cores available. However, the parallel build requires additional supporting software, so we first detail how to build serial CISM. For new users, it is recommended to first build and successfully run serial CISM before moving on to the parallel build.

Note: Glide, the shallow-ice dycore, can only run on a single processor, even when the code is built with full parallel support. This is also true of the SLAP solver routines.

The instructions below assume the user has administrative privileges for installing new software (note the extensive use of `sudo`). If you are working on a shared machine without administrative privileges, you might proceed by assuming all needed packages are present and continue to the CISM installation section. If you encounter problems, you can refer back to this section to determine which packages might be missing or problematic before contacting your system administrator.

Mac OS X

As mentioned above, we will take advantage of MacPorts, a software package manager for Macs. This will allow us to install most of the base level software libraries needed by CISM with few complications.

Go to http://www.macports.org/install.php, where you will find a range of ".pkg" installs available, including those for Mountain Lion, Lion, and Mavericks versions of Mac OS X.

Installing MacPorts requires installing the Xcode developer toolset provided by Apple. Details of how to obtain Xcode vary by version of OS X. See MacPorts installation instructions and this link for details. Once Xcode is installed, you may need to additionally download the "command line tool" from the Preferences / Downloads menu of Xcode.

Depending on computer security settings at your institution (firewalls, etc.), you may need to add proxy information so that Macports can communicate and download software from the outside world. All Macports software will be installed under `/opt/local/` by default. To add proxy information, after installing Macports, edit the configuration file at `/opt/local/etc/macports/macports.conf`. (Note this is probably a read-only file that requires superuser permission to edit, so you will need to edit the file with something like: `sudo vim /opt/local/etc/macports/macports.conf`). By searching for the text string "proxy", you will find the lines like `proxy_http hostname:12345` near the bottom of the file. Enter your proxy information here as appropriate (e.g., `hostname:your_host_info_here`).

If you have previously installed Macports but not updated it recently, it's generally a good idea to do so. Ideally, this should be done with admin or root privileges (you will be prompted to enter your password) using:

```
   sudo port selfupdate
```

You will then be prompted to update any installed ports that are outdated, which you can do using:

```
   sudo port upgrade outdated
```

To search for available software in Macports, type:

```
   port search software-name
```

Software is installed through Macports using the command:

```
   sudo port install software-name
```

Additional Macports tips will follow inline below. Extensive documentation for Macports can be found at the Macports website.

Ubuntu 12.10

This Ubuntu instructions describe setting up supporting software and CISM in a Linux environment. These instructions were written using a fresh installation of Ubuntu 12.10 but steps should be very similar in other versions of Ubuntu or other distributions of Linux. Instructions make use of the command line tool for installing packages that comes with Ubuntu, `apt-get`. Other package management tools (e.g., Software Center) could also be used.

It's generally a good idea to synchronize your local package index files before installing new software using `apt-get`:

```
   sudo apt-get update
```

To search for available packages, type:

```
   apt-cache search software-name
```

And to see detailed information about a package, type:

```
   apt-cache show software-name
```

Packages are installed through apt-get using the command:

```
   sudo apt-get install software-name
```

Some users have reported that BLAS and LAPACK libraries need to installed explicitly, for example when starting from a "clean" machine. To do this, use the following two commands:

```
   sudo apt-get install libblas-dev
```

and

```
   sudo apt-get install liblapack-dev
```

Additional apt-get tips will follow inline below. Extensive documentation for apt-get can be found at the Ubuntu website and through man pages (`man apt-get`).

2.2.1 Install git version control software

If you intend to download the CISM code as a git repository, you will need the `git` package installed. If you prefer to download a zipped archive of the code, this step can be skipped.

Mac OS X

Install git with:

```
   sudo port install git
```

Ubuntu 12.10

Install git with:

```
sudo apt-get install git
```

2.2.2 Install the GCC compiler suite

The GCC compiler suite contains compilers for C, C++, and, optionally, Fortran. Fortran and C compilers are required for serial CISM, and a C++ compiler is also needed for parallel CISM. CISM is known to work with GNU gfortran compilers, Intel ifort, and PGI. In these instructions we will use GNU compilers because they have been extensively tested with CISM and are freely available. Advanced users are welcome to use other compilers of their choosing.

CISM has been tested extensively with `gfortran` versions 4.5 and 4.6. Newer (or older) versions may also work, although version 4.8 introduces new features that may uncover issues.

Mac OS X

Searching for gcc with `port search gcc` will return:

```
gcc44 @4.4.7 (lang)
    The GNU compiler collection
...
```

in addition to a lot of other information on available Macports installs related to the GCC (Gnu) compiler suite.

Where possible, we want to make sure that all other software we build and install with Macports uses the version of GCC we choose to install. To date, we've had success with GCC 4.6.3 (others may work as well but have not been tested). To install GCC 4.6.3 type:

```
sudo port install gcc46
```

You will see some verbose output telling you what is happening (downloading packages, expanding them, building, installing, checking, etc.). When the install is complete, you can type:

```
port installed
```

to see what packages you currently have installed. You should see something like `gcc46 4.6.3_3 (active)`. (The minor version numbers after the "4.6" may differ as MacPorts makes updates to the port.) You will likely also see other packages that have been installed (software dependencies for GCC that were automatically installed by MacPorts and/or other ports you have manually installed).

The "(active)" description identifies which version of a particular package Macports currently thinks you want to use (e.g., you could also have another older GCC suite installed). To make sure the newly installed version is active, you would type:

```
port select gcc
```

which will return something like:

```
Available versions for gcc:
   gcc40
   gcc42
   mp-gcc46 (active)
   none
```

This confirms that GCC 4.6 is active (the `mp` indicates a Macports version). It is possible that gcc46 will be listed as active when you type `port installed`, but that mp-gcc46 will not be listed as active when you type `port select gcc`. If mp-gcc46 is not active as shown above, then you will need to select it using:

```
sudo port select --set gcc mp-gcc46
```

This will ensure that any generic call to gcc, gfortran, g++, will point to the libraries just installed.

Ubuntu 12.10

GNU compilers may have come with your Linux distribution. If not, they need to installed. Ubuntu 12.10 comes with `gcc` installed but not `gfortran`.

Install `gfortran` with:

```
sudo apt-get install gfortran
```

2.2.3 Install build tools

Additional tools are needed for managing the build process. `make` (specifically, GNU's gmake) usually comes with Mac and Linux distributions, but if not it should be installed. Additionally, CISM uses the `CMake` build utility (a cross-platform, open-source build system).

Mac OS X

While you probably already have a version of `make` on your system, it may be out of date or conflict with other Macports installed software. The required versions for CISM can be installed through Macports with this command:

```
sudo port install gmake cake
```

In addition to the software installed above, you should now see something like the following when you type `port installed`:

```
  gmake @3.82_0 (active)
  cmake @2.8.10_1 (active)
```

Ubuntu 12.10

On Ubuntu (and other Debian systems) there is usually a package called `build-essential` that includes a large collection of tools and libraries that are typically necessary for compiling code. Install these tools and `CMake` with:

```
sudo apt-get install build-essential cmake cmake-curses-gui
```

2.2.4 Install netCDF

NetCDF stands for “network Common Data Form” libraries, which are a machine-independent format for representing scientific data. This is required by CISM for performing input/output. The netCDF package you install must include Fortran libraries for CISM to compile (in some package managers, the Fortran libraries are in a separate package). There are substantial differences between versions 3.x and 4.x of netCDF, but both version series should work with CISM. It is also possible to download and compile netCDF libraries manually, which may be preferred by advanced users wanting to use a specific version.

It is also recommended that you install optional tools for working with netCDF datafiles. `ncview` is a convenient tools for viewing netCDF files. (Some alternatives are to write Python or

Matlab scripts or to use another tool like Paraview or `Ferret`.) `NCO` ("netCDF Operators") is a toolkit of command line tools for manipulating and analyzing data stored in netCDF-accessible formats.

Mac OS X

To install NetCDF, use `sudo port install netcdf-fortran +gcc46`. Note that there are other versions of NetCDF available to install. It is important to choose the one with the "Fortran" extension. The "gcc46" syntax specifies a port "variant". This tell Macports that, if there is a version of the selected software to install that is consistent with the GCC 4.6 compiler suite, then it should choose that one. Typing `port installed` should now include:

```
netcdf @4.3.2_0+dap+gcc46+netcdf4 (active)
netcdf-fortran @4.2_12+gcc46 (active)
```

The "dap+gcc46+netcdf4" comes along automatically.

Optional but recommended: Tools for working with netCDF data files.

```
sudo port install ncview nco
```

If you encounter an 'unable to open display' error when running `ncview`, you may need to install a newer version of the X Window System than the one provided by Apple. We have had success using the latest version of XQuartz: http://xquartz.macosforge.org

Ubuntu 12.10

Install netCDF libraries with:

```
sudo apt-get install libnetcdf-dev
```

Optional but recommended: Tools for working with netCDF data files.

```
sudo apt-get install netcdf-bin ncview nco
```

2.2.5 Install Python and related modules

Python is used by CISM to autogenerate I/O code during compilation, and is also used by most test case scripts to set up initial conditions and analyze and plot results. Only Python 2.7 has been tested. Python 3 may work for some uses but is likely to generate errors due to extensive changes between versions 2 and 3. Also, CISM uses a number of python modules:

- `numpy` - required for generating many test case initial conditions
- `matplotlib` - used by some plotting scripts. Not strictly necessary but required for those scripts to work properly.
- a python netCDF I/O module. Options are `netCDF4`, `Scientific.IO.NetCDF`, or `PyCDF`. `netCDF4` is the ideal choice, but it is often not available through Linux package managers and must be installed through a python package manager like pip, or manually. `PyCDF` is the least recommended option here because it is not entirely compatible with the others. `Scientific.IO.NetCDF` is usually available through Linux package managers.

Mac OS X

While Mac OS X already comes with a working Python distribution, we will need additional modules that can sometimes be tricky to get working together correctly. We have successfully used both the Enthought Python distribution (which is free for people associated with a university) and a version installed using Macports. To obtain and install

Enthought, click on the link above and follow their directions. To install version 2.7 using Macports, along with the necessary additional modules, do the following:

```
sudo port install python27 py27-numpy py27-matplotlib py27-scientific
 py27-netcdf4
```

The existence of two versions of python on your system can lead to confusion. It is important that you leave the version of python that came supplied by Apple so that your system has access to it. However, you will want to be sure that CISM has access to the new, more modern version of python you have installed. In our experience, this can be one of the most problematic parts of the installation process. You can use `port select`:

```
sudo port select python python27
```

You can check that Macports python is used by default by typing:

```
which python
```

and you should see: `/opt/local/bin/python`. If you instead see `/usr/bin/python` then the default Apple python is still the version that is being used on the command line. If this happens, or if you encounter errors with this setup, an alternative approach is to modify the `PATH` variable in your `.bashrc` or similar environment settings script to make sure that `/opt/local/bin` is before `/usr/bin` in your path.

Ubuntu 12.10

Python generally comes with most Linux distributions. If it is not present, it must be installed. Often, there is an additional python development package that is necessary when working with compiled code (tpyically called `python-dev` on Ubuntu).

Install python modules with:

```
sudo apt-get install python-dev python-numpy python-matplotlib
 python-scientific
```

Optional: Installing `netCDF4` python module.

Install pip (a tool for installing and managing Python packages):

```
sudo apt-get install pip
```
[a]

Next, install `HDF5` using pip:

```
sudo apt-get install libhdf5-dev hdf5-tools hdf5-helpers flex
```

Finally, install `netCDF4` using pip:

```
sudo -E pip install netcdf4
```

[a]For Ubuntu 14.04 there is a known issue with pip and handling freetypes. This can be fixed using `sudo apt-get install libfreetype6-dev libxft-dev`.

2.3 Building Serial CISM

At this point we are ready to build a *serial* version of CISM and its linked libraries. While we ultimately want to build a version of the code that also runs in parallel, it is often useful to stop at this step to make sure everything is working. Then, if problems occur during the parallel build process (as they sometimes do), we know those problems have occurred only during the last step of the process.

If you have not already done so, obtain the source code following the instructions above in section 2.1. Below, all paths starting with `./` indicate the root level of the source code directory. E.g., if you expanded your tar.gz archive into a main source code directory with the path `/usr/JohnDoe/CISM`, the `./` refers to the path `/usr/JohnDoe/CISM/`.

Unlike previous versions of the code, the build system is now entirely based on CMake (Autotools is no longer used).

Build scripts are provided that should work for most standard Mac and Linux setups, as well as some supercomputing platforms on which CISM is commonly run. All build scripts are located in the `./builds` directory from the root level of the code. In general, change to the subdirectory that most closely matches your system and intended build.

If you encounter an error when using the included scripts, you may need to modify some details, such as the location of your NetCDF libraries or your compiler names. Other errors you might encounter may indicate that some of the supporting software (above) is missing.

Mac OS X

On a Mac, you should be able to build the code by doing the following:

1. Change to the `./builds/mac-gnu` directory from the root level of the code.
2. Configure the build using `source mac-gnu-serial-cmake`
3. Build the code using `make -j X`, where the "X" refers to the number of processors available for use in the build (or just `make` if you have only one processor).

Ubuntu 12.10

On a Linux platform, you should be able to build the code by doing the following:

1. Change to the `./builds/linux-gnu` directory from the root level of the code.
2. Configure the build using `source linux-gnu-serial-cmake`
3. Build the code using `make -j X`, where the "X" refers to the number of processors available for use in the build (or just `make` if you have only one processor).

When the build completes, you can check for the executable driver by typing `ls cism_driver` from within the current build directory (here, from within the `./builds/mac-gnu` or `./builds/linux-gnu/` directory). The file `cism_driver` is the executable you will link to when running the model, which is generally done using a symbolic link. For example, from the `./tests/higher-order/shelf/` directory, one would link to this executable using,

```
ln -s ../../../builds/mac-gnu/cism_driver/cism_driver ./
```

Chapter 8 discusses running the executable for standard test cases.

Advanced users may want more control over the build scripts. There are a number of build options used by CMake to customize the build. You can manually modify the build scripts included with the code, or use the tool `ccmake` to interactively adjust build options (type `ccmake ../../` from any build directory after having run the configure script once). The available options are listed in Table 2.1. Many of these options pertain to the parallel build which is discussed in more detail below.

Also, there are standard CMake options that can be set (e.g., `CMAKE_C_COMPILER`, `CMAKE_Fortran_COMPILER`, etc.). Many of these are explained in the CMake documentation.

2.4 Installing Supporting Software for Parallel CISM

To build parallel CISM, MPI compilers and libraries are required. Only the higher-order dycore (Glissade) can run in parallel. (There is also a higher-order dycore called Glam that can be run in parallel, but it is used for development and testing and is not supported for scientific applications.)

In addition, you may choose to include the Trilinos package of external solver libraries. These are not required, but for some problems Trilinos may provide better performance and

`CISM_BUILD_CISM_DRIVER`	Toggle to build cism_driver, on by default
`CISM_BUILD_EXTRA_EXECUTABLES`	Toggle to build other executables, off by default
`CISM_COUPLED`	Toggle to build CISM for use with CESM, off by default
`CISM_ENABLE_BISICLES`	Toggle to build a BISICLES-capable cism_driver, off by default
`CISM_FORCE_FORTRAN_LINKER`	Toggle to force using a Fortran linker for building executables, off by default
`CISM_GNU`	Toggle to set compilation flags needed for the gnu compiler, off by default
`CISM_INCLUDE_IMPLICIT_LINK_LIB`	Toggle to explicitly include the CMAKE_Fortran_IMPLICIT_LINK_LIBRARIES on the link line, on by default
`CISM_MPI_MODE`	Toggle to configure with MPI, on by default
`CISM_NETCDF_LIBS`	netCDF library name(s)
`CISM_NO_EXECUTABLE`	Set to ON to just build libraries, off by default
`CISM_SERIAL_MODE`	Toggle to configure in serial mode: off by default
`CISM_SOURCEMOD_DIR`	Path to SourceMod directory of F90 files to replace CISM files
`CISM_STATIC_LINKING`	Toggle to set static linking for executables, off by default
`CISM_USE_DEFAULT_IO`	Toggle to use default i/o files rather than running python script, off by default
`CISM_USE_GPTL_INSTRUMENTATION`	Toggle to use GPTL instrumentation, on by default
`CISM_USE_MPI_WITH_SLAP`	Toggle to use mpi when using SLAP solver, only relevant if CISM_SERIAL_MODE=ON, off by default
`CISM_USE_TRILINOS`	Toggle to use Trilinos external solver libraries, on by default
`CMAKE_VERBOSE_CONFIGURE`	Verbose CMake configuration, on by default

Table 2.1: Available `CMake` settings for configuring the CISM build process

stability than the native solvers. Trilinos can also technically be used with a serial build, but this configuration is not supported or recommended.

2.4.1 Install MPI

MPI (Message Passing Interface) libraries and compilers are necessary for compiling parallel CISM. These libraries are used for handling parallel communications when running the code on multiple processors. A more complete description of parallel model configurations is given in Chapter 7. (For example, some test cases and configurations when running the shallow-ice dycore are not fully supported in parallel). OpenMPI and MPICH are two common MPI implementations.

Mac OS X

It is likely that you already have versions of MPI installed on your system, but they may be out of date or not compatible with the other libraries we have and will be installing. Using Macports, the MPICH version of MPI is known to work when building CISM. (OpenMPI may also work, but we've seen more consistent success on Macs with MPICH.)

First, check Macports for available versions of MPICH using `port search mpich*`. We want the version that is compatible with our GCC compiler suite, so we type:

```
sudo port install mpich-devel-gcc46 +fortran
```

To make sure this is active, type

```
port installed mpi*
```

which should return

```
     mpich-devel-gcc46 @3.2a1_0+fortran (active)
```

As when installing the GCC compilers, we want to make sure any generic call to MPI points to MPICH. This can be done with the following command:

```
sudo port select --set mpi mpich-devel-gcc46-fortran
```

Ubuntu 12.10

Either OpenMPI or MPICH are likely to work with CISM on Linux machines. On Linux machines, we have tested OpenMPI more thoroughly. Install OpenMPI with:

```
   sudo apt-get install openmpi-bin
```

2.4.2 Install Trilinos solver libraries

Trilinos is a modern, open source, C++ based library of parallel nonlinear and linear solvers, preconditioning and mesh-partitioning tools, and much more. It can be downloaded here[7]. (The software is free, but you are required to enter your email address to download it.) The documentation below assumes that you are working with version 11.10.* and was specifically tested using version 11.10.2.

Building Trilinos requires CMake version 2.8 or later, which ideally you have already installed as discussed above. Trilinos is not needed to run the default parallel, higher-order dycore

[7]http://trilinos.org/download/

(Glissade), but it may be useful for more difficult problems or for debugging in cases where the native Fortran solvers fail to converge.

The build instructions for Trilinos on Mac and Linux are very similar, so users of both systems can follow the primary instructions below, except where noted.

Trilinos requires both (1) an "out-of-source build" and (2) an "out-of-build installation". This means that you cannot build the code in the same directory where the source code lives, and you cannot install the libraries in the same directory where you build the code. (Older versions of Trilinos required an out-of-source build but not an out-of-build installation.) The easiest way to satisfy this requirement is to have separate "source", "build" and "install" directories in the location where you want to install the code. For example, in `/usr/local/`, you could set up the following three directories:

```
trilinos-11.10.2-Source/
trilinos-11.10.2-Build/
trilinos-11.10.2-Install/
```

The "source" directory will be created on its own when you uncompress the tar.gz archive that you download. You do not have to keep the source code where you build and install the Trilinos libraries, but you will need to remember the path to where that source code lives on your computer.

To configure the Trilinos build, you will need to execute a CMake configure script. Sample configure scripts for a number of standard platforms are included in the "sampleScripts" directory under the root level of the Trilinos source code. Also, the CISM code includes examples of Trilinos configure scripts ("do-configure") for use with CISM for both Linux and Mac platforms in the `./utils/trilinos_config_scripts_examples` directory. We recommend starting with one of those scripts and modifying it as necessary to work on your system[8].

The paths to both the "source" and "install" directories are specified within the "do-configure" scripts. In these instructions, those directories are both assumed to live within `/usr/local/`, but other locations are fine to use too (e.g., in your home/User directory).

Mac OS X

Also note the explicit path in the MPI lines, e.g.,
`-D MPI_EXEC="/opt/local/mpiexec" \`

Since some Macs may come with their own pre-installed OpenMPI libraries, it is important here to specify the path to the version we previously installed using Macports.

Find the example script most appropriate for your system, copy it to the `trilinos-11.10.2-Build` directory, and modify it if necessary (e.g., adjust paths, compiler locations, etc.). Execute it with:

`source ./do-cmake`[9]

from within your `trilinos-11.10.2-Build` directory. Depending on where you are building and installing the code, you may need to have administrative privileges (in which case you would type `sudo source ./do-cmake`). If the configure step was successful, you should see the following displayed on your screen:

```
...
Processing enabled package: [PACKAGE NAME]
```

[8]If you are following the above installation instructions for Mac exactly, then the configure script `./utils/trilinos_config_scripts_examples/do-configure-Trilinos-11.10.2-for-Mac-10.9.4` should work with few modifications.

[9]Here we have assumed that the name of the configure script is "do-cmake". The script name may differ depending on what you have called it or if you copied and modified one of the scripts from `./utils/trilinos_config_scripts_examples`.

```
...

Exporting library dependencies ...

Finished configuring Trilinos!

-- Configuring done
-- Generating done
-- Build files have been written to: /usr/local/trilinos-11.10.2-Build
```

It is a good idea to scan the output while the "do-cmake" script is executing, for example to ensure the configure process is picking up the compilers you specified (e.g., it is using the Macports versions as opposed to some Mac default versions that might also be on your system). Once the code is configured successfully, build the libraries from within the `trilinos-11.10.2-Build` directory by typing:

`make` (or `sudo make` if necessary)

For multiprocessor machines, the build process can be sped up significantly using the "-j" command as described above for building serial CISM:

`make -j X`

where "X" is the number of cores available on your machine (e.g., `make -j 4` for a 2-processor, dual-core machine).

Building Trilinos can take a long time (e.g., an hour or more), depending on your machine, the number of processors used for the build, and the number and type of libraries you are installing. Once you have built the code, we highly recommend testing it using:

`make test`

(The `Trilinos_ENABLE_TESTS:BOOL` variable in the do-cmake script can be set to "OFF" to disable building of the tests.) Screen output will tell you if and how many tests failed. We have seen a few tests fail while still having a perfectly good and working Trilinos library. In general, if the number of tests passed is above 90%, the library will likely work fine with CISM. Query the CISM users or developers lists[10] if you have questions about specific Trilinos tests failing.

Mac OS X

On a Mac, MPI tests have been known to trigger a dialog box from the firewall. With more than 300 tests, these messages popping up continually can make it impossible to use your computer until the tests complete. To keep them from appearing, you can temporarily turn off your firewall under "System Preferences" (Security > Firewall > Stop). Be sure to turn the firewall back on when the tests are complete!

After running the tests, you will need to install Trilinos using:

`make install`

This will build the actual Trilinos libraries in the path specified in the

```
-D CMAKE_INSTALL_PREFIX:PATH=/path
```

line of your "do-cmake" script (above). For this example, those libraries will be installed in: `/usr/local/trilinos-11.10.2-Install`

After successfully building Trilinos, create an environment variable called `CISM_TRILINOS_DIR` so the CISM build process can find the Trilinos installation. For example, if you are using the bash shell and your current directory is the Trilinos install directory, you can do:

```
export CISM_TRILINOS_DIR=$PWD
```

[10] cism-users+subscribe@googlegroups.com -or- cism-devel+subscribe@googlegroups.com

You may prefer to modify your .bashrc or .bash_profile (or similar) to set this environment variable on every login.

Alternatively, you can modify the CISM parallel build script (below) so that the line:

```
-D CISM_TRILINOS_DIR=$CISM_TRILINOS_DIR \
```

is set to the Trilinos installation directory.

2.5 Building Parallel CISM

The procedure for building parallel CISM is nearly identical to the serial build (above). The build script for parallel CISM for a Mac is located at `builds/mac-gnu/mac-gnu-cmake`, while the build script for Linux is located at `builds/linux-gnu/linux-gnu-cmake`. From the appropriate directory, run:

`source mac-gnu-cmake` or `source linux-gnu-cmake`

Once the configuration step completes successfully, you can compile the code as before with:

```
make
```

or

```
make -j 4
```

if you have 4 processors available (or as many processors as you would like to use). See Section 2.3 for details about customizing the build process.

Building a parallel version of CISM that includes Trilinos requires setting the

```
-D CISM_USE_TRILINOS
```

flag to `ON` in the `builds/mac-gnu/mac-gnu-cmake` script.

2.6 Next Steps

If you make any changes to the source code, you only need to re-run `make` from your build directory to generate an updated executable. One exception is that if you edit the lists of input/output netCDF variables in `*_vars.def` files (see Appendices A and B), you need to first re-source the configuration script (e.g., `source mac-gnu-cmake`) before re-running `make`.

Now that you have successfully built the code, you can proceed to Chapters 3 and 5 to learn more detailed information about ice sheet modeling, to Chapters 4 and 6 to learn more about the various model approximations available through CISM, or you can proceed to Chapter 8 to learn how to run and examine some standard model test cases.

Chapter 3

Introduction to Ice Sheet Modeling: Derivation of Field Equations

In this chapter we give an introduction to ice dynamics and the other conservation equations that must be accounted for when simulating glacier and ice sheet evolution.

Ice sheets are key components of Earth's climate system. They contain nearly all of the planet's fresh water; changes in their volume have an immediate effect on sea level; changes in their area and surface characteristics affect global albedo; and they play a role in the circulation of both the atmosphere and the ocean. Through the latter half of the Quaternary Period, ice sheets have modulated the planetary response to orbitally-driven insolation cycles. Looking forward, the Greenland and Antarctic ice sheets have the potential to play important roles in climate change.

A numerical model is a discrete approximation of a continuous process. The approximation is discrete, due to both the finite nature of a computer's precision and the finite problem size that can be tackled using a computer. The underlying process is continuous because it is commonly formulated in terms of ordinary or partial differential equations (ODEs and PDEs, respectively). Numerical models cannot be "solved" until the boundary and initial conditions are specified. Such models may be very simple, such as a harmonic oscillator, or very complex, as in a complete Earth System Model (ESM). In general, ESMs are composed of a number of component models, of which land ice (encompassing glaciers, ice caps, and large ice sheets) is but one component (with atmosphere, ocean, sea ice, and land surface models being the other primary components).

3.1 Conservation Equations

For the majority of the physical systems encountered in Earth science problems, the first step in modeling is a mathematical description of the conservation of energy, momentum, and mass. Only after that description is laid out do we turn to the question of approximating those equations in a form that can be solved on a computer.

3.1.1 Integral form

The mathematical formulation of conservation can be arrived at by considering the change in a quantity ϕ that is known within a **control** volume V. The control volume is enclosed by a surface S, with the outward positive unit vector $\hat{n}$, normal to S.

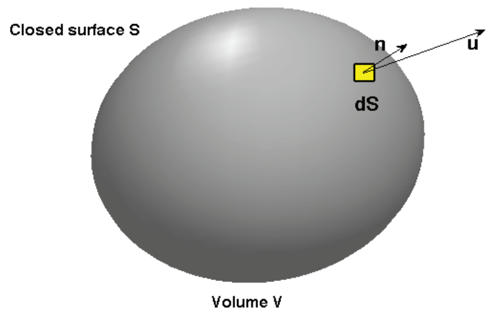


Figure 3.1: Diagram of control volume and associated quantities.

The value of ϕ within V may change over time if:

1. There is a flux of ϕ through S. The flux is partitioned into two parts, one due to diffusion and another due to advection.
2. ϕ is created or destroyed within V.

Formally, the time rate of change in ϕ within V is written:

$$\frac{\partial}{\partial t}\int_V \phi dV = -\int_S \mathbf{F}\cdot\hat{n}\, dS - \int_S \phi\mathbf{u}\cdot\hat{n}\, dS + \int_V R dV \tag{3.1}$$

where $\mathbf{F}$ represents the flux due to diffusion ($\mathbf{F} \propto \nabla\phi$), $\phi\mathbf{u}$ represents a velocity field *advecting* ϕ, and R represents a source (or sink) of ϕ. Vector quantities are represented in **boldface**. The negative signs in front of the first two terms on the right-hand side indicate that an outward flux results in a decrease in ϕ within the volume enclosed by S.

This statement of conservation of ϕ in the unit volume V is always true, independent of the size of V and even if the fields enclosed by S are not continuous (this is the case because we integrate over V). It is important to note that that any information on spatial scales smaller than V is lost in the process of integration.

3.1.2 Derivative form

Numerical models are often easier to formulate from the derivative form of the conservation equation, and this is more often the form in which conservation equations are written. This requires the derivatives of ϕ to exist within V, which in turn allows the integral form of the conservation equation to be written as partial differential equations, which are upheld within the control volume.

Begin with the terms describing diffusive and advective fluxes into or out of the control volume. The divergence theorem states that

$$\int_S \mathbf{F}\cdot\hat{n}\, dS = \int_V \nabla\cdot\mathbf{F}\, dV. \tag{3.2}$$

Here and below, we will alternate between standard vector notation (as above) and index notation, where a single subscript indicates a component of a vector, two different subscripts indicate a tensor quantity, and two repeated subscripts indicate summation. Thus, an identical way of writing equation (3.2) is

$$\int_S \mathbf{F}_j\, n_j\, dS \;=\; \int_V \frac{\partial \mathbf{F}_j}{\partial x_j}\, dV. \tag{3.3}$$

Using the divergence theorem, the surface integrals over fluxes in (3.1) may be replaced by

$$-\int_S \mathbf{F}\cdot\hat{n} dS \;-\; \int_S \phi\mathbf{u}\cdot\hat{n} dS \;=\; -\int_V \nabla\cdot(\mathbf{F} \;+\; \phi\; \mathbf{u})\, dV. \tag{3.4}$$

Assuming that the coordinate system is stationary with respect to the velocity field $\mathbf{u}$ (i.e, assuming an Eularian reference frame), it is possible to write

$$\frac{\partial}{\partial t}\int_V \phi\; dV \;=\; \int_V \frac{\partial \phi}{\partial t}\, dV \tag{3.5}$$

The integral form of the conservation equation can now be written as

$$\int_V \left\{\frac{\partial \phi}{\partial t} \;+\; \nabla\cdot(\mathbf{F} \;+\; \phi\mathbf{u}) \;-\; R\right\} dV \;=\; 0 \tag{3.6}$$

Because V is an arbitrary volume, this equation can be true only if the term in brackets is zero for the volume. Hence, for any volume having continuously differentiable ϕ,

$$\frac{\partial \phi}{\partial t} \;+\; \nabla\cdot(\mathbf{F} \;+\; \phi\mathbf{u}) \;-\; R \;=\; 0. \tag{3.7}$$

This is the general form for all conservation laws in continuum mechanics. Below, we apply this equation to the three quantities conserved in an ice sheet (and ideally, in an ice sheet model): mass, energy, and momentum.

3.2 Applications of the General Conservation Equation

3.2.1 Conservation of momentum

Starting from Newton's second law of motion, conservation of momentum is expressed as

$$\frac{d}{dt}\int\limits_V \rho u_i\; dV \;=\; \int\limits_V \frac{\partial \sigma_{ij}}{\partial x_j}\, dV + \int\limits_V \rho g_i\; dV \tag{3.8}$$

where t is time, ρ is density, u is velocity, σ_{ij} is the stress tensor, g is the acceleration due to gravity, V is the volume of an arbitrary fluid element, and $(i, j) = \{x, y, z\}$ in a Cartesian coordinate system. Equation (3.8) tells us that a fluid element of arbitrary size experiences a "body force" $\rho g_i \delta V$ due to gravity, which is balanced by stress divergence $\frac{\partial \sigma_{ij}}{\partial x_j}\delta V$ and acceleration of the fluid in the volume δV.

Making the assumptions that we have continuous fields and that ice is incompressible (i.e., its density ρ does not change under conditions of interest), we can write

$$\rho\frac{Du_i}{Dt} \;=\; \frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \tag{3.9}$$

in which D is a material derivative. Because the Froude number for ice flow is extremely small, the acceleration term (the first term on the left-hand side) can be neglected, leaving the steady-state form,

$$\frac{\partial \sigma_{ij}}{\partial x_j} + \rho g_i \;=\; 0. \tag{3.10}$$

Equation (3.10) states that the body force (the gravitational driving force) is balanced by forces resulting from gradients in the stress tensor σ_{ij}. All models of ice-flow dynamics are based on solving this equation in some form. Chapters 4 and 6 provide additional details on the approximations to this equation that are solved by CISM.

The stress tensor σ_{ij} has nine components in a three-dimensional, Cartesian coordinate system,

$$\sigma = \begin{vmatrix} \sigma_{xx} & \sigma_{xy} & \sigma_{xz} \\ \sigma_{yx} & \sigma_{yy} & \sigma_{yz} \\ \sigma_{zx} & \sigma_{zy} & \sigma_{zz}. \end{vmatrix} \tag{3.11}$$

Since σ_{ij} is symmetric, only six of these components are independent. The components along the diagonal are called normal stresses, and the off-diagonal components are called shear stresses. Deformation results not from the full stress but from the deviatoric stress,

$$\tau_{ij} \; = \; \sigma_{ij} \; - \; \frac{1}{3}\sigma_{kk}\delta_{ij}, \tag{3.12}$$

in which δ_{ij} is the Kroneker delta (or the identity tensor). For shear stresses, (3.12) indicates that the full and deviatoric stresses are identical.

Constitutive relationship

To relate the stress tensor to fluid motion, we introduce the strain rate tensor,

$$\dot{\epsilon}_{ij} \; = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \quad i,j = x,y,z, \tag{3.13}$$

where u_i are the velocity vector components. The strain rate tensor $\dot{\epsilon}_{ij}$, and hence gradients in the velocity field, are related to the stress tensor τ_{ij} by a constitutive relation. For a Newtonian fluid, this can be expressed as

$$\tau_{ij} \; = \eta\dot{\epsilon}_{ij}, \tag{3.14}$$

which states that the strain rate is proportional to the stress, with the ice viscosity η serving as the constant of proportionality. Ice does not behave as a Newtonian fluid, and instead exhibits a power-law rheology, so that it becomes more fluid (less viscous) the faster it deforms. This relationship can be expressed through Nye's generalization of Glen's flow law,

$$\tau_{ij} \; = \; A(T^*)^{\frac{-1}{n}}\dot{\epsilon}_e^{\frac{1-n}{n}}\dot{\epsilon}_{ij} \tag{3.15}$$

in which T^* is the absolute temperature corrected for the pressure dependence of the melt temperature, $\dot{\epsilon}_e$ is the second invariant (a norm) of the stress tensor, and the power-law exponent n is commonly taken as 3. A comparison of (3.14) and (3.15) indicates that one can define an "effective" ice viscosity for (3.14) as

$$\eta_e \; = \; A(T^*)^{\frac{-1}{n}}\dot{\epsilon}_e^{\frac{1-n}{n}}. \tag{3.16}$$

The temperature-dependent rate factor A follows the Arrhenius relationship

$$A\left(T^*\right) \; = \; EA_o e^{-Q/RT^*}, \tag{3.17}$$

in which A_o is a constant, Q is the activation energy for crystal creep, R is the gas constant, and E is a tuning parameter, which can be used to account for the effects of impurities and anisotropic ice fabrics. The homologous temperature is

$$T^* = T + \rho g H \Phi, \tag{3.18}$$

in which Φ is 9.8 $\times 10^{-8}$ K Pa^{-1}, or about 8.7 $\times 10^{-4}$ K m^{-1} in ice. The pressure-dependent melt temperature is simply the triple point temperature minus the product $\rho g H \Phi$.

3.2.2 Conservation of energy

The first law of thermodynamics is used to make a basic statement of conservation of energy in a volume of ice V enclosed within a surface S:

$$\frac{d}{dt}\int_V E\; dV \;=\; -\int_S \mathbf{F}\cdot\hat{n}\; dS \;-\; \int_S E\mathbf{u}\cdot\hat{n}\; dS \;+\; \int_V W dV \tag{3.19}$$

in which E is the total energy within the volume, F_i is the energy flux due to diffusion, and W represents any sources or sinks of energy within the volume. The term Eu_i is an energy flux through S due to advection. Following the steps laid out earlier, we use the divergence theorem and the assumptions of continuous fields and incompressibility to obtain

$$\frac{dE}{dt} \;+\; \nabla\cdot(F_i + Eu_i) \;-\; W \;=\; 0 \tag{3.20}$$

Our goal is to use the first law of thermodynamics to compute the temperature of the ice and any changes in that temperature over time.

The energy E is the product of density and the specific internal energy of the ice e, which is itself the product of the specific heat capacity c_p and temperature T (because there is no transfer between internal energy and pressure for an incompressible fluid). Thus,

$$\begin{aligned}\frac{dE}{dt} &= \frac{d(\rho e)}{dt}\\ &= \rho\frac{de}{dt} + e\frac{d\rho}{dt}\\ &= \rho c_p\frac{dT}{dt}\end{aligned} \tag{3.21}$$

The heat flux due to diffusion follows Fourier's "law" for heat conduction,

$$\begin{aligned}\nabla\cdot F_i &= \nabla\cdot(-k\;\nabla T)\\ &= -k\;\nabla^2 T,\end{aligned} \tag{3.22}$$

in which k is the thermal conductivity of ice and we assume gradients in its magnitude to be negligible.

Using progress made above, we can write the advection term

$$\nabla\cdot(Eu_i) \;=\; \rho c_p\; u_i\cdot\nabla T. \tag{3.23}$$

In the expansion of the terms on the left-hand side of (3.23) (using the product rule), we have implicitly ignored the term involving $\nabla\cdot u_i$ because it is small with respect to the other terms retained on the right-hand side.

Two energy sources must be considered: the work done on the system by internal deformation and the latent heat associated with phase changes. The former is the product of the strain rate and deviatoric stress, $\dot{\epsilon}_{ij}\tau_{ij}$. The latter is the product of the latent heat of fusion and the amount of material (ice) subject to melting (or freezing) per unit volume, per unit time, $L_f M_f$.

At last, we can write equation (3.20) in terms of temperature:

$$\frac{\partial T}{\partial t} \;=\; \frac{k}{\rho c_p}\nabla^2 T \;-\; u_i\cdot\nabla T \;+\; \frac{1}{\rho c_p}\dot{\epsilon}_{ij}\tau_{ij} \;+\; \frac{1}{\rho c_p}L_f M_f. \tag{3.24}$$

It is often the case that horizontal terms $\frac{\partial^2 T}{\partial x^2}$ and $\frac{\partial^2 T}{\partial y^2}$ are small enough to be ignored.

3.2.3 Conservation of mass

In this case, ϕ from our general conservation equation represents the mass M, or more conveniently $M = \int_V \rho dV$, the integral of the density over the volume. Assuming that there are no sources or sinks of mass in the volume (R=0), the mass conservation equation is written

$$\int_V \frac{\partial \rho}{\partial t}\, dV \;+\; \int_V \nabla \cdot \rho \mathbf{u} dV \;=\; 0 \tag{3.25}$$

Ice is incompressible (the density does not change in time), which provides the equation for local mass continuity,

$$\nabla \cdot \mathbf{u} \;=\; 0. \tag{3.26}$$

Equation (3.26) says that the velocity field is divergence-free. Applying the ∇ operator in Cartesian coordinates gives

$$\frac{\partial u_x}{\partial x} \;+\; \frac{\partial u_y}{\partial y} \;+\; \frac{\partial u_z}{\partial z} \;=\; 0. \tag{3.27}$$

To make use of this statement, we need to integrate from the base b to the upper surface s of the ice mass,

$$\int_b^s \left(\frac{\partial u_x}{\partial x} \;+\; \frac{\partial u_y}{\partial y} \;+\; \frac{\partial u_z}{\partial z} \right) dz \;=\; 0. \tag{3.28}$$

The integral of $\frac{\partial u_z}{\partial z}$ is simply the difference between the vertical component of the velocity at the upper and lower surfaces, so

$$u_z\,(s) - u_z\,(b) \;=\; -\int_b^s \frac{\partial u_x}{\partial x} dz \;-\; \int_b^s \frac{\partial u_y}{\partial y} dz. \tag{3.29}$$

Changing the order of integration and differentiation using Leibniz's rule, we obtain

$$\begin{aligned} u_z\,(s) - u_z\,(b) \;=\; & -\tfrac{\partial}{\partial x} \int_b^s u_x dz \;+\; u_x(s)\tfrac{\partial s}{\partial x} \;-\; u_x(b)\tfrac{\partial b}{\partial x} \\ & -\tfrac{\partial}{\partial y} \int_b^s u_y dz \;+\; u_y(s)\tfrac{\partial s}{\partial y} \;-\; u_y(b)\tfrac{\partial b}{\partial x}. \end{aligned} \tag{3.30}$$

The vertical velocity at the upper surface $u_z(s)$ is the result of motion parallel to the surface slope, the rate of new ice accumulation B_s, and any time change in the surface height,

$$u_z\,(s) \;=\; \frac{\partial s}{\partial t} \;+\; u_x(s)\frac{\partial s}{\partial x} \;+\; u_y(s)\frac{\partial s}{\partial y} \;-\; B_s, \tag{3.31}$$

recognizing that a negative accumulation rate indicates ablation. Similarly, the vertical velocity at the lower surface is

$$u_z\,(b) \;=\; \frac{\partial b}{\partial t} \;+\; u_x(b)\frac{\partial b}{\partial x} \;+\; u_y(b)\frac{\partial b}{\partial y} \;-\; M_b \tag{3.32}$$

in which M_b is the basal melt rate ($M_b < 0$ for freeze-on). Substituting (3.31) and (3.32) into (3.30), we find that many terms cancel:

$$\frac{\partial s}{\partial t} \;-\; B_s \;-\; \frac{\partial b}{\partial t} \;+\; M_b \;=\; -\frac{\partial}{\partial x}\int_b^s u_x dz \;-\; \frac{\partial}{\partial y}\int_b^s u_y dz. \tag{3.33}$$

Finally, making the substitution that the ice thickness $H = s - b$, we obtain

$$\frac{\partial H}{\partial t} \;=\; -\frac{\partial}{\partial x}\int_b^s u_x dz \;-\; \frac{\partial}{\partial y}\int_b^s u_y dz \;+\; B_s \;-\; M_b. \tag{3.34}$$

Integrating in the vertical gives

$$\frac{\partial H}{\partial t} \;=\; -\,\nabla \cdot (U_i H) +\; B_s \;-\; M_b, \tag{3.35}$$

in which U_i is the vertically averaged velocity, i.e., $U_i = \frac{1}{H}\int_b^s u_i dz$. Equation (3.35) is prognostic; we can use the current velocity and geometry of the ice to compute the rate of change in the geometry.

Chapter 4

Shallow Ice Dynamics: Glide Dynamical Core

This chapter describes the numerical implementation of mass, momentum, and energy conservation in the Glide dynamical core, which uses a Shallow Ice Approximation for ice flow. For a model governed by shallow-ice dynamics, the solutions for the conservation of mass and momentum are intimately linked.

4.1 Ice Thickness Evolution

The evolution of the ice thickness, H, stems from the continuity equation and can be expressed as

$$\frac{\partial H}{\partial t} = -\boldsymbol{\nabla} \cdot (\overline{\boldsymbol{u}}H) + B, \tag{4.1}$$

where $\overline{\boldsymbol{u}}$ is the vertically averaged ice velocity, B is the surface mass balance and $\boldsymbol{\nabla}$ is the horizontal gradient operator (Payne and Dongelmans, 1997).

For some regions of large-scale ice sheets, such as the slow-moving interior, or for simulations run at coarse spatial resolution, a model governed by the *shallow ice approximation* (SIA) may be appropriate. Further, for very long time integrations, such as those required in paleoclimate studies, a model governed by the SIA may be the only computationally practical approach.

Based largely on the assumption that bedrock and ice surface slopes are sufficiently small (Hutter, 1983), the SIA neglects all stress components other than those associated with vertical shearing in the horizontal directions. These stresses, τ_{xz} and τ_{yz}, are approximated by

$$\begin{aligned} \tau_{xz}(z) &= -\rho g(s-z)\frac{\partial s}{\partial x}, \\ \tau_{yz}(z) &= -\rho g(s-z)\frac{\partial s}{\partial y}, \end{aligned} \tag{4.2}$$

where ρ is the density of ice, g the acceleration due to gravity and $s = H + b$ the ice surface.

Strain rates $\dot{\epsilon}_{ij}$ of polycrystalline ice are related to the stress tensor by the nonlinear flow law:

$$\dot{\epsilon}_{iz} = \frac{1}{2}\left(\frac{\partial u_i}{\partial z} + \frac{\partial u_z}{\partial i}\right) = A(T^*)\tau_*^{(n-1)}\tau_{iz} \qquad i = x, y, \tag{4.3}$$

where τ_* is the effective shear stress defined by the second invariant of the stress tensor, n the flow law exponent and A the temperature–dependent flow law coefficient. T^* is the absolute temperature corrected for the dependence of the melting point on pressure ($T^* = T + 8.7 \cdot 10^{-4}(s-z)$, T in Kelvin, Huybrechts, 1986). The parameters n and A are determined experimentally; n

is usually taken to be 3 and A depends primarily on temperature and secondarily on factors such as crystal size and orientation and ice impurities. Experiments suggest that A follows the Arrhenius relationship:

$$A(T^*) = fae^{-Q/RT^*}, \tag{4.4}$$

where a is a temperature–independent material constant, Q is the activation energy for creep and R is the universal gas constant (Paterson, 1994). The tuning parameter f may be used to speed up the ice flow, accounting for the effects of ice impurities and the development of anisotropic ice fabrics (Payne, 1999; Tarasov and Peltier, 1999, 2000; Peltier et al., 2000).

Integrating (4.4) with respect to z gives the vertical profile of the horizontal velocity in each column:

$$\boldsymbol{u}(z) - \boldsymbol{u}(b) = -2(\rho g)^n |\boldsymbol{\nabla} s|^{n-1} \boldsymbol{\nabla} s \int_b^z A(s-z)^n dz, \tag{4.5}$$

where $\boldsymbol{u}(b)$ is the basal sliding velocity. Integrating (4.5) again with respect to z gives an expression for the vertically averaged ice velocity:

$$\overline{\boldsymbol{u}}H = -2(\rho g)^n |\boldsymbol{\nabla} s|^{n-1} \boldsymbol{\nabla} s \int_b^s \int_b^z A(s-z)^n dz dz'. \tag{4.6}$$

The vertical ice velocity can be derived from the conservation of mass for an incompressible material:

$$\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} = 0. \tag{4.7}$$

Integrating (4.7) with respect to z gives the vertical profile of the vertical velocity in each column:

$$w(z) = -\int_h^z \boldsymbol{\nabla} \cdot \boldsymbol{u}(z) dz + w(b), \tag{4.8}$$

with lower kinematic boundary condition

$$w(b) = \frac{\partial b}{\partial t} + \boldsymbol{u}(b) \cdot \boldsymbol{\nabla} b - M_b, \tag{4.9}$$

where M_b is the basal melt rate given by (4.55). The upper kinematic boundary is given by the surface mass balance and must satisfy

$$w(s) = \frac{\partial s}{\partial t} + \boldsymbol{u}(s) \cdot \boldsymbol{\nabla} s - B_s. \tag{4.10}$$

4.1.1 Numerical grid

The continuous equations describing ice physics have to be discretized in order to be solved by a computer (which is inherently finite). This section describes the finite–difference grids used by the model.

Horizontal grid

The modelled region ($x \in [0, L_x]$, $y \in [0, L_y]$) is discretized using a regular grid so that $x_i = (i-1)\Delta x$ for $i \in [1, N]$ (and similarly for y_j). The model uses two staggered horizontal grids in order to improve stability. Both grids use the same grid spacing, Δx and Δy, but are offset by half a grid cell (see Fig. 4.1). Quantities calculated on the staggered (r, s)–grid are denoted with

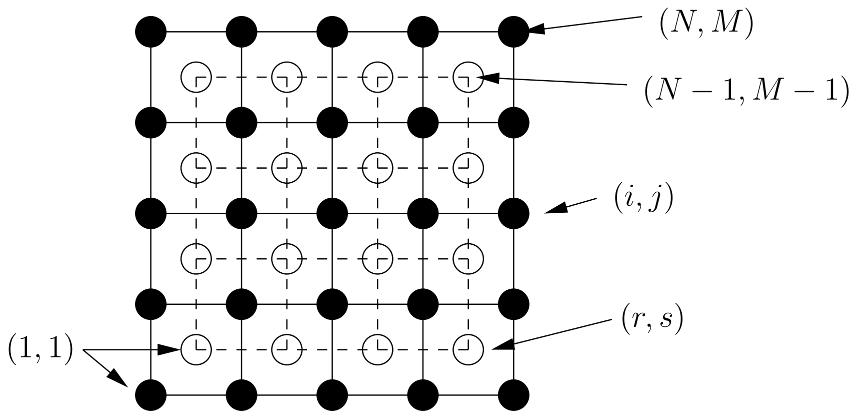


Figure 4.1: Horizontal Grid.

a tilde, i.e., $\tilde{F}$. Quantities are transformed between grids by averaging over the surrounding nodes; i.e., a quantity in the (i, j)–grid becomes in the (r, s)–grid:

$$\tilde{F}_{r,s} = \tilde{F}_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{4}(F_{i,j} + F_{i+1,j} + F_{i+1,j+1} + F_{i,j+1}), \tag{4.11a}$$

and similarly for the reverse transformation:

$$F_{i,j} = F_{r-\frac{1}{2},s-\frac{1}{2}} = \frac{1}{4}(\tilde{F}_{r-1,s-1} + \tilde{F}_{r,s-1} + \tilde{F}_{r,s} + \tilde{F}_{r-1,s}). \tag{4.11b}$$

In general, horizontal velocities and associated quantities like the diffusivity are calculated on the (r, s)–grid. Ice thickness, temperatures and vertical velocities are calculated on the (i, j)–grid.

Horizontal gradients are calculated on the (r, s)–grid; i.e., surface gradients are

$$\left(\frac{\partial s}{\partial x}\right)_{r,s} = \tilde{s}^x_{r,s} = \frac{s_{i+1,j} - s_{i,j} + s_{i+1,j+1} - s_{i,j+1}}{2\Delta x}, \tag{4.12a}$$

$$\left(\frac{\partial s}{\partial y}\right)_{r,s} = \tilde{s}^y_{r,s} = \frac{s_{i,j+1} - s_{i,j} + s_{i+1,j+1} - s_{i+1,j}}{2\Delta y}. \tag{4.12b}$$

Ice thickness gradients, $\tilde{H}^x_{r,s}$ and $\tilde{H}^y_{r,s}$, are formed analogously. Gradients in the (r, s)–grid are formed in a similar way:

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} = u^x_{i,j} = \frac{\tilde{u}_{r,s-1} - \tilde{u}_{r-1,s-1} + \tilde{u}_{r,s} - \tilde{u}_{r-1,s}}{2\Delta x}. \tag{4.13}$$

Periodic boundary conditions

The model can be run with horizontal periodic boundary conditions, i.e. with the western edge of the modelled region joined to the eastern edge. Figure 4.2 illustrates the numeric grid when the model is run in torus mode.

These boundary conditions are enforced by exchanging points for the temperature and vertical velocity calculations. The ice thicknesses are calculated explicitly at the ghostpoints.

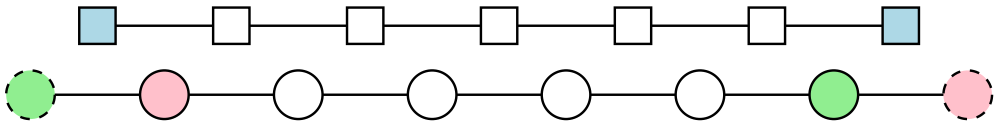


Figure 4.2: A row of the numeric grid when the model is used in torus mode. Circles indicate points in (i,j)–grid and squares indicate points in the (r,s)–grid. Points with the same color are logically the same.

σ–coordinate system

The vertical coordinate, z, is scaled by the ice thickness analogous to the s–coordinate in numerical weather simulations (e.g., Holton, 1992). A new vertical coordinate, σ, is introduced so that the ice surface is at $\sigma = 0$ and the ice base at $\sigma = 1$ (see Fig. 4.3), i.e.

$$\sigma = \frac{s-z}{H}. \tag{4.14}$$

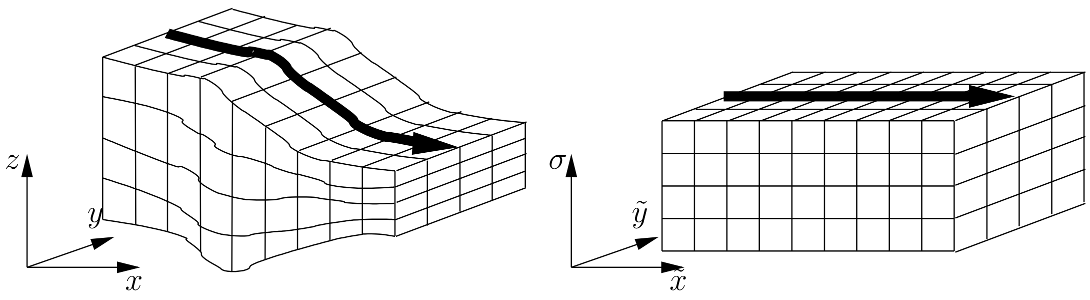


Figure 4.3: Vertical scaling of the ice sheet model. The vertical axis is scaled to unity. The horizontal coordinates are not changed.

The derivatives of a function f in (x, y, z, t) become in the new $(\tilde{x}, \tilde{y}, \sigma, \tilde{t})$ system:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial \tilde{x}} + \frac{1}{H}\Delta_{\tilde{x}}\frac{\partial f}{\partial \sigma}, \tag{4.15a}$$
$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial \tilde{y}} + \frac{1}{H}\Delta_{\tilde{y}}\frac{\partial f}{\partial \sigma}, \tag{4.15b}$$
$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial \tilde{t}} + \frac{1}{H}\Delta_{\tilde{t}}\frac{\partial f}{\partial \sigma}, \tag{4.15c}$$
$$\frac{\partial f}{\partial z} = -\frac{1}{H}\frac{\partial f}{\partial \sigma}, \tag{4.15d}$$

where the geometric factors, $\Delta_{\tilde{x}}$, $\Delta_{\tilde{y}}$ and $\Delta_{\tilde{t}}$, are defined by

$$\Delta_{\tilde{x}} = \left(\frac{\partial s}{\partial \tilde{x}} - \sigma\frac{\partial H}{\partial \tilde{x}}\right), \tag{4.16a}$$
$$\Delta_{\tilde{y}} = \left(\frac{\partial s}{\partial \tilde{y}} - \sigma\frac{\partial H}{\partial \tilde{y}}\right), \tag{4.16b}$$
$$\Delta_{\tilde{t}} = \left(\frac{\partial s}{\partial \tilde{t}} - \sigma\frac{\partial H}{\partial \tilde{t}}\right). \tag{4.16c}$$

The integral of z becomes in the σ–coordinate system:

$$\int_b^z f dz = -H \int_1^\sigma f d\sigma. \tag{4.17}$$

The vertical coordinate can be discretized using an irregular grid spacing to reflect the fact that ice flow is more variable at the bottom of the ice column. In the vertical the index k is used.

4.1.2 Ice sheet equations in σ–coordinates

The horizontal velocity, (4.5), becomes in the σ–coordinate system

$$\boldsymbol{u}(\sigma) = -2(\rho g)^n H^{n+1} |\boldsymbol{\nabla} s|^{n-1} \boldsymbol{\nabla} s \int_1^\sigma A\sigma^n d\sigma + \boldsymbol{u}(1). \tag{4.18}$$

The vertically averaged velocity is

$$\overline{\boldsymbol{u}} H = H \int_0^1 \boldsymbol{u} d\sigma + \boldsymbol{u}(1)H. \tag{4.19}$$

The vertical velocity (4.8) becomes

$$w(\sigma) = -\int_1^\sigma \left(\frac{\partial \boldsymbol{u}}{\partial \sigma} \cdot (\boldsymbol{\nabla} s - \sigma \boldsymbol{\nabla} H) + H \boldsymbol{\nabla} \cdot \boldsymbol{u} \right) d\sigma + w(1), \tag{4.20}$$

with lower boundary condition

$$w(1) = \frac{\partial b}{\partial t} + \boldsymbol{u}(1) \cdot \boldsymbol{\nabla} b - M_b. \tag{4.21}$$

4.1.3 Calculating the horizontal velocity and diffusivity

Horizontal velocity and diffusivity calculations are split up into two parts:

$$\boldsymbol{u}(\sigma) = c\boldsymbol{\nabla} s + \boldsymbol{u}(1), \tag{4.22a}$$

$$D = H \int_0^1 c d\sigma, \tag{4.22b}$$

$$\boldsymbol{q} = D\boldsymbol{\nabla} s + H\boldsymbol{u}(1), \tag{4.22c}$$

with

$$c(\sigma) = -2(\rho g)^n H^{n+1} |\boldsymbol{\nabla} s|^{n-1} \int_1^\sigma A\sigma^n d\sigma. \tag{4.22d}$$

Quantities $\boldsymbol{u}$ and D are found on the velocity grid. Integrating from the ice base ($k = N-1$), the discretized quantities become

$$\tilde{c}_{r,s,N} = 0, \tag{4.23a}$$

$$\tilde{c}_{r,s,k} = -2(\rho g)^n H_{r,s}^{n+1}\left((\tilde{s}^x_{r,s})^2 + (\tilde{s}^y_{r,s})^2\right)^{\frac{n-1}{2}}$$
$$\sum_{\kappa=N-1}^{k} \frac{A_{r,s,\kappa} + A_{r,s,\kappa+1}}{2}\left(\frac{\sigma_{\kappa+1}+\sigma_\kappa}{2}\right)^n (\sigma_{\kappa+1}-\sigma_\kappa), \qquad (4.23b)$$

$$\tilde{D}_{r,s} = H_{r,s}\sum_{k=0}^{N-1}\frac{\tilde{c}_{r,s,k}+\tilde{c}_{r,s,k+1}}{2}(\sigma_{k+1}-\sigma_k). \qquad (4.23c)$$

Expressions for $\boldsymbol{u}_{i,j,k}$ and $\boldsymbol{q}_{i,j}$ are straightforward.

4.1.4 Solving the ice thickness evolution equation

Equation (4.1) can be rewritten as a diffusion equation, with nonlinear diffusion coefficient D:

$$\frac{\partial H}{\partial t} = -\boldsymbol{\nabla}\cdot D\boldsymbol{\nabla}s + B = -\boldsymbol{\nabla}\cdot\boldsymbol{q} + B, \qquad (4.24)$$

where $B = B_s - M_b$ is the total mass balance. This nonlinear partial differential equation can be linearized by using the diffusion coefficient from the previous time step. The diffusion coefficient is calculated on the (r,s)–grid, i.e. staggered in both x and y directions. Figure 4.4 illustrates the staggered grid. Using finite differences, the fluxes in the x direction, q^x, become

$$q^x_{i+\frac{1}{2},j} = -\frac{1}{2}(\tilde{D}_{r,s}+\tilde{D}_{r,s-1})\frac{s_{i+1,j}-s_{i,j}}{\Delta x}, \qquad (4.25a)$$
$$q^x_{i-\frac{1}{2},j} = -\frac{1}{2}(\tilde{D}_{r-1,s}+\tilde{D}_{r-1,s-1})\frac{s_{i,j}-s_{i-1,j}}{\Delta x}, \qquad (4.25b)$$

and the fluxes in the y direction are

$$q^y_{i,j+\frac{1}{2}} = -\frac{1}{2}(\tilde{D}_{r,s}+\tilde{D}_{r-1,s})\frac{s_{i,j+1}-s_{i,j}}{\Delta y}, \qquad (4.25c)$$
$$q^y_{i,j-\frac{1}{2}} = -\frac{1}{2}(\tilde{D}_{r,s-1}+\tilde{D}_{r-1,s-1})\frac{s_{i,j}-s_{i,j-1}}{\Delta y}. \qquad (4.25d)$$

ADI Scheme

The alternating–direction implicit (ADI) method uses the concept of operator splitting where (4.24) is solved first in the x–direction and then in the y–direction (Press et al., 1992). The time step Δt is divided into two partial steps $\Delta t/2$. The discretized version of Equation (4.24) becomes (Huybrechts, 1986):

$$2\frac{H^{t+\frac{1}{2}}_{i,j}-H^t_{i,j}}{\Delta t} = -\frac{q^{x,t+\frac{1}{2}}_{i+\frac{1}{2},j}-q^{x,t+\frac{1}{2}}_{i-\frac{1}{2},j}}{\Delta x} - \frac{q^{y,t}_{i,j+\frac{1}{2}}-q^{y,t}_{i,j-\frac{1}{2}}}{\Delta y} + B_{i,j}, \qquad (4.26a)$$
$$2\frac{H^{t+1}_{i,j}-H^{t+\frac{1}{2}}_{i,j}}{\Delta t} = -\frac{q^{x,t+\frac{1}{2}}_{i+\frac{1}{2},j}-q^{x,t+\frac{1}{2}}_{i-\frac{1}{2},j}}{\Delta x} - \frac{q^{y,t+1}_{i,j+\frac{1}{2}}-q^{y,t+1}_{i,j-\frac{1}{2}}}{\Delta y} + B_{i,j}. \qquad (4.26b)$$

Gathering all $t+\frac{1}{2}$ terms on the left side,(4.26a) can be expressed as a tridiagonal set of equations for each row j:

$$-\alpha_{i,j}H^{t+\frac{1}{2}}_{i-1,j} + (1-\beta_{i,j})H^{t+\frac{1}{2}}_{i,j} - \gamma_{i,j}H^{t+\frac{1}{2}}_{i+1,j} = \delta_{i,j}, \qquad (4.27)$$

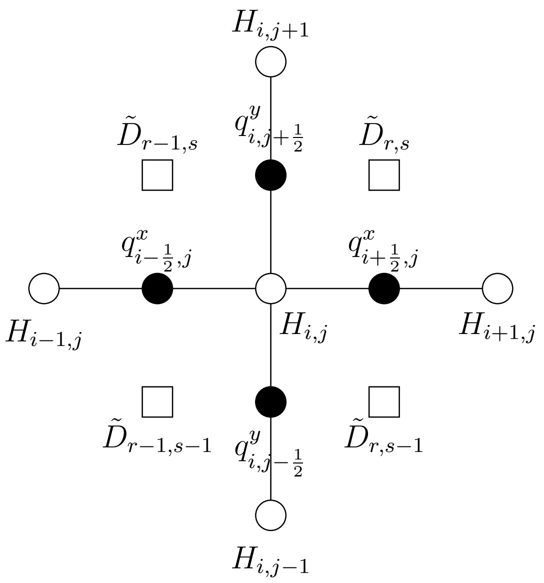


Figure 4.4: Illustration of the staggered grid used to calculate ice thicknesses, diffusivities and mass fluxes.

with

$$\alpha_{i,j} = \frac{\tilde{D}_{r-1,s} + \tilde{D}_{r-1,s-1}}{4\Delta x^2}\Delta t, \tag{4.28a}$$

$$\beta_{i,j} = -\frac{\tilde{D}_{r,s} + 2\tilde{D}_{r-1,s} + \tilde{D}_{r-1,s-1}}{4\Delta x^2}\Delta t = -(\alpha_{i,j} + \gamma_{i,j}), \tag{4.28b}$$

$$\gamma_{i,j} = \frac{\tilde{D}_{r,s} + \tilde{D}_{r,s-1}}{4\Delta x^2}\Delta t. \tag{4.28c}$$

The RHS is given by

$$\delta_{i,j} = H^t_{i,j} - \frac{\Delta t}{2\Delta y}\left(q^{y,t}_{i,j+\frac{1}{2}} - q^{y,t}_{i,j-\frac{1}{2}}\right) + \frac{\Delta t}{2}B_{i,j} + \alpha_{i,j}h_{i-1,j} - \beta_{i,j}h_{i,j} + \gamma_{i,j}h_{i+1,j}. \tag{4.28d}$$

A similar tridiagonal system is found for each column i of (4.26b).

Linearised semi–implicit scheme

Using the Crank–Nicolson scheme, the semi–implicit temporal discretisation of (4.24) is

$$\frac{H^{t+1}_{i,j} - H^t_{i,j}}{\Delta t} = \frac{q^{x,t+1}_{i+\frac{1}{2},j} - q^{x,t+1}_{i-\frac{1}{2},j}}{2\Delta x} + \frac{q^{y,t+1}_{i,j+\frac{1}{2}} - q^{y,t+1}_{i,j-\frac{1}{2}}}{2\Delta y} + \frac{q^{x,t}_{i+\frac{1}{2},j} - q^{x,t}_{i-\frac{1}{2},j}}{2\Delta x} + \frac{q^{y,t}_{i,j+\frac{1}{2}} - q^{y,t}_{i,j-\frac{1}{2}}}{2\Delta y} + B_{i,j}. \tag{4.29}$$

The superscripts ^{t} and $^{t+1}$ indicate at what time the ice thickness H is evaluated. Collecting all H^{t+1} terms of (4.29) on the LHS and moving all other terms to the RHS, we can rewrite (4.29) as

$$-\alpha_{i,j}H^{t+1}_{i-1,j} - \beta_{i,j}H^{t+1}_{i+1,j} - \gamma_{i,j}H^{t+1}_{i,j-1} - \delta_{i,j}H^{t+1}_{i,j+1} + (1 - \epsilon_{i,j})H^{t+1}_{i,j} = \zeta_{i,j}, \tag{4.30}$$

with the RHS

$$\begin{aligned}\zeta_{i,j} = \alpha_{i,j}H^t_{i-1,j} + \beta_{i,j}H^t_{i+1,j} + \gamma_{i,j}H^t_{i,j-1} + \delta_{i,j}H^t_{i,j+1} + (1+\epsilon_{i,j})H^t_{i,j} \\ + 2(\alpha_{i,j}h_{i-1,j} + \beta_{i,j}h_{i+1,j} + \gamma_{i,j}h_{i,j-1} + \delta_{i,j}h_{i,j+1} + \epsilon_{i,j}h_{i,j}) + B_{i,j}\Delta t. \end{aligned} \tag{4.31}$$

Here the elements of the sparse matrix are

$$\alpha_{i,j} = \frac{\tilde{D}_{r-1,s} + \tilde{D}_{r-1,s-1}}{4\Delta x^2}\Delta t, \tag{4.32a}$$

$$\beta_{i,j} = \frac{\tilde{D}_{r,s} + \tilde{D}_{r,s-1}}{4\Delta x^2}\Delta t, \tag{4.32b}$$

$$\gamma_{i,j} = \frac{\tilde{D}_{r,s-1} + \tilde{D}_{r-1,s-1}}{4\Delta y^2}\Delta t, \tag{4.32c}$$

$$\delta_{i,j} = \frac{\tilde{D}_{r,s} + \tilde{D}_{r-1,s}}{4\Delta y^2}\Delta t, \tag{4.32d}$$

$$\epsilon_{i,j} = -(\alpha_{i,j} + \beta_{i,j} + \gamma_{i,j} + \delta_{i,j}). \tag{4.32e}$$

This matrix equation is solved using an iterative solver for non-symmetric sparse matrices. The solver used here is the bi–conjugate gradient method with incomplete LU decomposition preconditioning provided by the SLAP package.

Nonlinear scheme

The nonlinearity of (4.24) arises from the dependance of D on s. A nonlinear scheme for (4.24) can be formulated using Picard iteration, which consists of two iterations: an outer, nonlinear and an inner, linear equation. The scheme starts with the diffusivity from the previous time step:

$$D^{(0),t+1} = D^t. \tag{4.33a}$$

Equation (4.30) becomes

$$\begin{aligned} - \alpha^{(\xi),t+1}_{i,j}H^{t+1}_{i-1,j} - \beta^{(\xi),t+1}_{i,j}H^{(\xi+1),t+1}_{i+1,j} - \gamma^{(\xi),t+1}_{i,j}H^{(\xi+1),t+1}_{i,j-1} \\ - \delta^{(\xi),t+1}_{i,j}H^{(\xi+1),t+1}_{i,j+1} + (1-\epsilon^{(\xi),t+1}_{i,j})H^{(\xi+1),t+1}_{i,j} = \zeta^{(0),t}_{i,j}. \end{aligned} \tag{4.33b}$$

Equation (4.33b) is iterated over ξ until the maximum ice thickness residual is smaller than some threshold:

$$\max\left(\left|H^{(\xi+1),t+1} - H^{(\xi),t+1}\right|\right) < H_{\text{res}}. \tag{4.34}$$

4.1.5 Calculating vertical velocities

Grid velocity

The vertical grid moves as a result of using a σ–coordinate system. The grid velocity is

$$w^{\text{grid}}(\sigma) = \frac{\partial s}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}s - \sigma\left(\frac{\partial H}{\partial t} + \boldsymbol{u}\cdot\boldsymbol{\nabla}H\right). \tag{4.35}$$

The numerical implementation of (4.35) is straightforward.

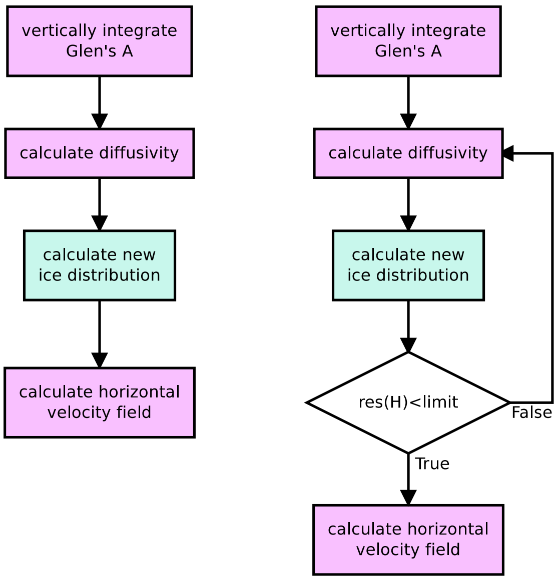

vertically integrate Glen's A
calculate diffusivity
calculate new ice distribution
calculate horizontal velocity field
vertically integrate Glen's A
calculate diffusivity
calculate new ice distribution
res(H)<limit
False
True
calculate horizontal velocity field

Figure 4.5: Flow diagram showing how the linearised solver (on the left) and the non–linear solver work. The inner, linear iteration is contained within the box labeled "calculate new ice distribution".

Vertical velocity

The discretized version of the vertical velocity equation (4.20) is slightly more compilicated because the horizontal velocities are calculated on the (r, s) grid. The vertical velocity at the ice base is $w_{i,j,N} = w^{\text{grid}}_{i,j,N} - M_b i, j$, where $M_b i, j$ is the basal melt rate. Integrating from the bottom, the vertical velocity is then

$$\begin{aligned} w_{i,j,k} = -\sum_{\tilde{k}=N-1}^{1} \Bigg\{ \mathcal{H}_{i,j} \left(\frac{u^x_{i,j,k} + u^x_{i,j,k+1}}{2} + \frac{v^y_{i,j,k} + v^y_{i,j,k+1}}{2} \right) (\sigma_{k+1} - \sigma_k) \\ + (\tilde{u}_{i,j,k+1} - \tilde{u}_{i,j,k}) \left(\tilde{s}^x_{i,j} - \frac{1}{2}(\sigma_{k+1} + \sigma_k) \tilde{H}^x_{i,j} \right) \\ + (\tilde{v}_{i,j,k+1} - \tilde{v}_{i,j,k}) \left(\tilde{s}^y_{i,j} - \frac{1}{2}(\sigma_{k+1} + \sigma_k) \tilde{H}^y_{i,j} \right) \Bigg\} + w_{i,j,N}, \end{aligned} \tag{4.36}$$

with the weighted ice thickness

$$\begin{aligned} \mathcal{H}_{i,j} = & \frac{4H_{i,j} + 2(H_{i-1,j} + H_{i+1,j} + H_{i,j-1} + H_{i,j+1})}{16} \\ & + \frac{H_{i-1,j-1} + H_{i+1,j-1} + H_{i+1,j+1} + H_{i-1,j+1}}{16}. \end{aligned}$$

This scheme produces vertical velocities at the ice divide which are too small. The vertical velocities on the ice surface are given by the upper kinematic boundary condition (4.10). Equation (4.36) can be corrected with

$$w^*_{i,j,k} = w_{i,j,k} - (1 - \sigma_k)(w_{i,j,k} - w_{s\,i,j}), \tag{4.37}$$

where $w_{s\,i,j}$ is the vertical velocity at the ice surface given by (4.10). Figure 4.6 shows the different vertical velocities at the ice surface. The difference between the vertical velocities calculated by the model and the vertical velocities given by (4.10) at the ice margin are due

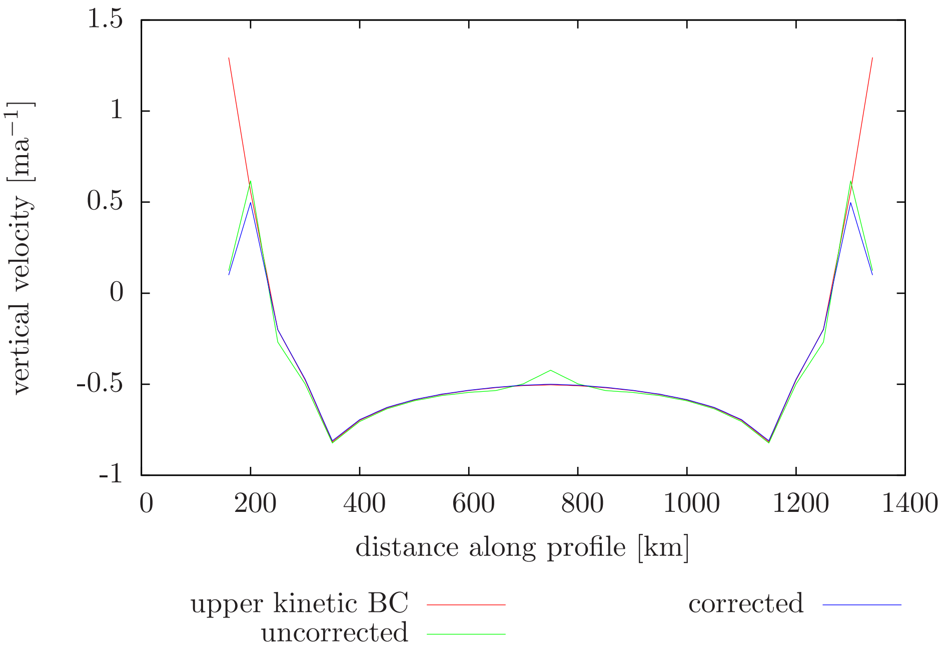


Figure 4.6: Vertical ice surface velocities of the EISMINT-1 moving margin experiment.

to the fact that temperatures and velocities are only calculated when the ice is thicker than a certain threshold value which is not met at the ice margin.

Figure 4.7 shows vertical profiles of the vertical velocity at the ice divide and a point halfway between the divide and the domain margin. A corresponding temperature profile is also shown since the vertical velocity determines the vertical temperature advection (see Section 4.2.4).

4.2 Temperature Solver

The flow law coefficient A in (4.3) depends on the ice temperature. Thus it is necessary to determine how the distribution of ice temperatures changes with a changing ice sheet configuration. The thermal evolution of the ice sheet is described by

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c}\nabla^2 T - \boldsymbol{u}\cdot\boldsymbol{\nabla}T + \frac{\Phi}{\rho c} - w\frac{\partial T}{\partial z}, \tag{4.38}$$

where T is the absolute temperature, k is the thermal conductivity of ice, c is the specific heat capacity, and Φ is the heat generated by internal friction. In the σ–coordinate system, (4.38) becomes

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c H^2}\frac{\partial^2 T}{\partial \sigma^2} - \boldsymbol{u}\cdot\boldsymbol{\nabla}T + \frac{\sigma g}{c}\frac{\partial \boldsymbol{u}}{\partial \sigma}\cdot\boldsymbol{\nabla}s + \frac{1}{H}\frac{\partial T}{\partial \sigma}\left(w - w_{\text{grid}}\right). \tag{4.39}$$

The terms on the RHS represent (1) vertical diffusion, (2) horizontal advection, (3) internal heat generation due to friction, and (4) vertical advection and a correction due to the σ–coordinate system. We rewrite (4.39) as

$$\frac{\partial T}{\partial t} = a\frac{\partial^2 T}{\partial \sigma^2} + b(\sigma) + \Phi(\sigma) + c(\sigma)\frac{\partial T}{\partial \sigma}, \tag{4.40}$$

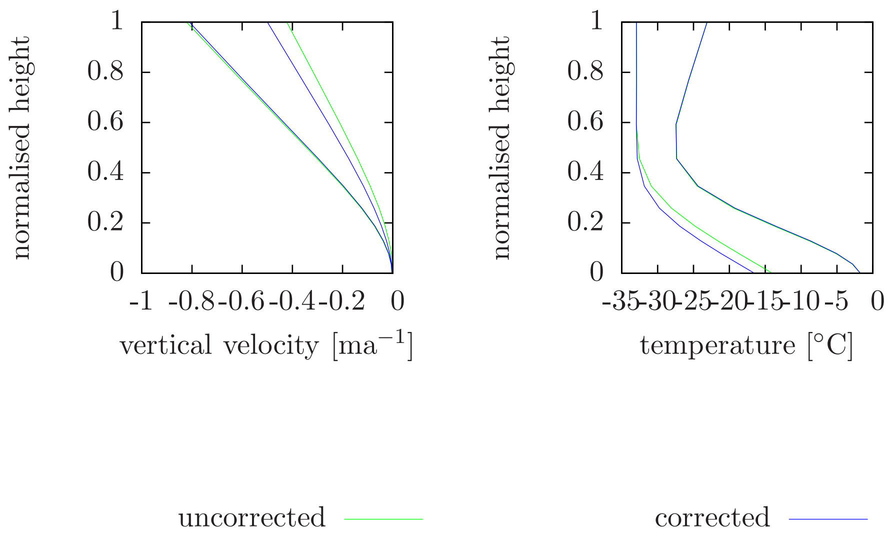


Figure 4.7: Vertical velocity and temperature distribution for columns at the ice divide and a point halfway between the divide and the domain margin.

where

$$a = \frac{k}{\rho c H^2} \tag{4.41a}$$

$$b(\sigma) = -\boldsymbol{u} \cdot \boldsymbol{\nabla} T \tag{4.41b}$$

$$\Phi(\sigma) = \frac{\sigma g}{c} \frac{\partial \boldsymbol{u}}{\partial \sigma} \cdot \boldsymbol{\nabla} s \tag{4.41c}$$

$$c(\sigma) = \frac{1}{H} (w - w_{\mathrm{grid}}) \tag{4.41d}$$

(Note the different meanings for b and c here relative to usage elsewhere in this document.)

4.2.1 Vertical diffusion

Discretization of $\partial^2 T/\partial\sigma^2$ is slightly complicated because the vertical grid is irregular. Using Taylor series the central difference formulas are

$$\left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_{k-1/2}} = \frac{T_k - T_{k-1}}{\sigma_k - \sigma_{k-1}} \tag{4.42a}$$

and

$$\left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_{k+1/2}} = \frac{T_{k+1} - T_k}{\sigma_{k+1} - \sigma_k}. \tag{4.42b}$$

The second partial derivative is then, also using central differences:

$$\left.\frac{\partial^2 T}{\partial \sigma^2}\right|_{\sigma_k} = \frac{\partial T/\partial\sigma|_{\sigma_{k+1/2}} - \partial T/\partial\sigma|_{\sigma_{k-1/2}}}{1/2\,(\sigma_{k+1} - \sigma_{k-1})}. \tag{4.42c}$$

Inserting (4.42a) and (4.42b) into (4.42c), we obtain

$$\left.\frac{\partial^2 T}{\partial \sigma^2}\right|_{\sigma_k} = \frac{2(T_{k+1}-T_k)}{(\sigma_{k+1}-\sigma_k)(\sigma_{k+1}-\sigma_{k-1})} - \frac{2(T_k-T_{k-1})}{(\sigma_k-\sigma_{k-1})(\sigma_{k+1}-\sigma_{k-1})}. \tag{4.42d}$$

Finally, the terms of equation (4.42d) are rearranged:

$$\left.\frac{\partial^2 T}{\partial \sigma^2}\right|_{\sigma_k} = \frac{2T_{k-1}}{(\sigma_k-\sigma_{k-1})(\sigma_{k+1}-\sigma_{k-1})} - \frac{2T_k}{(\sigma_{k+1}-\sigma_k)(\sigma_k-\sigma_{k-1})} + \frac{2T_{k+1}}{(\sigma_{k+1}-\sigma_k)(\sigma_{k+1}-\sigma_{k-1})}. \tag{4.43}$$

4.2.2 Horizontal advection

The horizontal advection term, $-\boldsymbol{u}\cdot\boldsymbol{\nabla}T$, is found using an upwinding scheme. We start with the 1D case. (The method can readily be extended to 2D.) As always, the temperature function is expressed as a Taylor series:

$$T(x+\Delta x) = T(x) + \Delta x T'(x) + \frac{\Delta x^2}{2}T''(x) + \dots \tag{4.44a}$$

If we replace Δx with $2\Delta x$, (4.44a) becomes

$$T(x+2\Delta x) = T(x) + 2\Delta x T'(x) + 2\Delta x^2 T''(x) + \dots \tag{4.44b}$$

From (4.44a) and (4.44b) we can construct a difference formula where the $\mathcal{O}(\Delta x^2)$ error is cancelled, by multiplying (4.44a) with 4 and substracting the result from (4.44b):

$$T'_+(x) = \frac{4T(x+\Delta x) - T(x+2\Delta x) - 3T(x)}{2\Delta x}, \tag{4.45a}$$

and similarly for the backward difference:

$$T'_-(x) = -\frac{4T(x-\Delta x) - T(x-2\Delta x) - 3T(x)}{2\Delta x}. \tag{4.45b}$$

So the horizontal advection term in 1D becomes

$$b_x = -u_x\frac{\partial T}{\partial x} = \frac{-u_x}{2\Delta x}\begin{cases} -(4T_{i-1} - T_{i-2} - 3T_i) & \text{when } u_x > 0 \\ 4T_{i+1} - T_{i+2} - 3T_i & \text{when } u_x < 0 \end{cases} \tag{4.46}$$

A similar expression is found for b_y by substituting y for x. The combined horizontal advection term is

$$b = -\boldsymbol{u}\cdot\boldsymbol{\nabla}T = -\left(u_x\frac{\partial T}{\partial x} + u_y\frac{\partial T}{\partial y}\right) = b_x + b_y = b_1 + b_2 T_i. \tag{4.47}$$

4.2.3 Heat generation

Taking the derivative of (4.18) with respect to σ, we get

$$\frac{\partial u_x}{\partial \sigma} = -2(\rho g)^n H^{n+1}|\boldsymbol{\nabla}s|^{n-1}\frac{\partial s}{\partial x}A(T^*)\sigma^n. \tag{4.48}$$

Thus,

$$\begin{aligned}\Phi(\sigma) &= \frac{\sigma g}{c}\frac{\partial \boldsymbol{u}}{\partial \sigma}\cdot \boldsymbol{\nabla} s = \frac{\sigma g}{c}\left(\frac{\partial u_x}{\partial \sigma}\frac{\partial s}{\partial x} + \frac{\partial u_y}{\partial \sigma}\frac{\partial s}{\partial y}\right) \\ &= -2(\rho g)^n H^{n+1}|\boldsymbol{\nabla} s|^{n-1}\frac{\sigma g}{c}A(T^*)\sigma^n\left(\left(\frac{\partial s}{\partial x}\right)^2 + \left(\frac{\partial s}{\partial y}\right)^2\right) \\ &= -\frac{2}{c\rho}(g\sigma\rho)^{n+1}\left(H|\boldsymbol{\nabla} s|\right)^{n+1} A(T^*).\end{aligned} \tag{4.49}$$

The constant factor $\frac{2}{c\rho}(g\sigma\rho)^{n+1}$ is calculated during initialization in the subroutine `glide_init_temp`. This factor is assigned to array `c1(1:upn)`. `c1` also includes various scaling factors and the factor 1/16 to normalise $\mathcal{A}$.

The factor $(H|\boldsymbol{\nabla} s|)^{n+1}$ is calculated in subroutine `glide_finddisp`:

$$c_{2i,j} = \left(\tilde{H}_{i,j}\sqrt{\tilde{S}_{x\,i,j}^{\,2} + \tilde{S}_{y\,i,j}^{\,2}}\right)^{n+1}. \tag{4.50}$$

The final factor is found by averaging over the neighboring nodes:

$$\mathcal{A}_{i,j} = 4A_{i,j} + 2(A_{i-1,j} + A_{i+1,j} + A_{i,j-1} + A_{i,j+1}) + (A_{i-1,j-1} + A_{i+1,j-1} + A_{i+1,j+1} + A_{i-1,j+1}). \tag{4.51}$$

4.2.4 Vertical advection

The vertical advection term $\partial T/\partial\sigma$ is computed from the central difference formula for unevenly spaced nodes:

$$\frac{\partial T}{\partial \sigma} = \frac{T_{k+1} - T_{k-1}}{\sigma_{k+1} - \sigma_{k-1}}. \tag{4.52}$$

4.2.5 Boundary conditions

At the upper boundary, ice temperatures are set to the surface temperature, T_{surf}. The ice at the base is heated by the geothermal heat flux and sliding friction:

$$\left.\frac{\partial T}{\partial \sigma}\right|_{\sigma=1} = -\frac{GH}{k} - \frac{H\boldsymbol{\tau}_b \cdot \boldsymbol{u}(1)}{k}, \tag{4.53}$$

where $\boldsymbol{\tau}_b = -\rho g H \boldsymbol{\nabla} s$ is the basal shear stress and $\boldsymbol{u}(1)$ is the basal ice velocity. Ice temperatures are held constant if they reach the pressure melting point of ice, i.e.

$$T^* = T_{\text{pmp}} \quad \text{if } T \geq T_{\text{pmp}}. \tag{4.54}$$

Excess heat is then used to formulate a melt rate, M_b:

$$M_b = \frac{k}{\rho L}\left(\frac{\partial T^*}{\partial z} - \frac{\partial T}{\partial z}\right), \tag{4.55}$$

where L is the latent heat of fusion. Finally, basal temperatures are held constant if the ice is floating:

$$\frac{\partial T(1)}{\partial t} = 0. \tag{4.56}$$

4.2.6 Putting it all together

Equation (4.39) is solved for each ice column. The horizontal dependency of the horizontal advection term, (4.41b), is resolved by iterating the vertical solution. Combining the individual terms using a fully explicit finite-difference scheme, (4.40) becomes

$$\frac{T_{k,t+1}-T_{k,t}}{\Delta t} = \left(\frac{2aT_{k-1,t}}{(\sigma_k-\sigma_{k-1})(\sigma_{k+1}-\sigma_{k-1})} - \frac{2aT_{k,t}}{(\sigma_{k+1}-\sigma_k)(\sigma_k-\sigma_{k-1,t})} + \frac{2aT_{k+1,t}}{(\sigma_{k+1}-\sigma_k)(\sigma_{k+1}-\sigma_{k-1})}\right) + b_{1\,k,t} + b_{2\,k}T_{k,t} + \Phi_k + c_k\frac{T_{k+1,t}-T_{k-1,t}}{\sigma_{k+1}-\sigma_{k-1}}. \tag{4.57a}$$

Similarly, the fully implicit scheme is

$$\frac{T_{k,t+1}-T_{k,t}}{\Delta t} = \left(\frac{2aT_{k-1,t+1}}{(\sigma_k-\sigma_{k-1})(\sigma_{k+1}-\sigma_{k-1})} - \frac{2aT_{k,t+1}}{(\sigma_{k+1}-\sigma_k)(\sigma_k-\sigma_{k-1,t+1})} + \frac{2aT_{k+1,t+1}}{(\sigma_{k+1}-\sigma_k)(\sigma_{k+1}-\sigma_{k-1})}\right) + b_{1\,k,t+1} + b_{2\,k}T_{k,t+1} + \Phi_k + c_k\frac{T_{k+1,t+1}-T_{k-1,t+1}}{\sigma_{k+1}-\sigma_{k-1}}. \tag{4.57b}$$

Taking the average of (4.57a) and (4.57b) gives the *Crank–Nicolson scheme*. The resulting equation is then rearranged, with terms of $T_{k-1,t+1}$, $T_{k,t+1}$ and $T_{k+1,t+1}$ combined to give the tridiagonal system

$$\alpha_k T_{k-1,t+1} + \beta_k T_{k,t+1} + \gamma_k T_{k+1,t+1} = \delta_k, \tag{4.58}$$

where for $k=2$ to $N-1$ we have

$$\alpha_k = -\frac{1}{2}\frac{2a\Delta t}{(\sigma_k-\sigma_{k-1})(\sigma_{k+1}-\sigma_{k-1})} + \frac{1}{2}\frac{c_k\Delta t}{\sigma_{k+1}-\sigma_{k-1}}, \tag{4.59a}$$

$$\beta_k = 1 + \frac{1}{2}\frac{2a\Delta t}{(\sigma_{k+1}-\sigma_k)(\sigma_k-\sigma_{k-1})} - \frac{1}{2}b_{2\,k}\Delta t = 1 - \alpha_k - \gamma_k - \frac{1}{2}b_{2\,k}\Delta t, \tag{4.59b}$$

$$\gamma_k = -\frac{1}{2}\frac{2a\Delta t}{(\sigma_{k+1}-\sigma_k)(\sigma_{k+1}-\sigma_{k-1})} - \frac{1}{2}\frac{c_k\Delta t}{\sigma_{k+1}-\sigma_{k-1}}, \tag{4.59c}$$

$$\delta_k = -\alpha_k T_{k-1,t} + (2-\beta_k)T_{k,t} - \gamma_k T_{k+1,t} + \frac{1}{2}(b_{1\,k,t}+b_{1\,k,t+1})\Delta t + \Phi_k\Delta t. \tag{4.59d}$$

Boundary conditions

At the upper boundary we have

$$\alpha_1 = 0, \quad \beta_1 = 1, \quad \gamma_1 = 0, \quad \delta_1 = T_{\text{surf}}. \tag{4.59e}$$

The lower boundary condition is more complicated. Here we consider only the case when the temperature is below the pressure melting point of ice. The BCs for floating ice and temperatures at the pressure melting point of ice are trivial. The geothermal heat flux is applied at the lower boundary; i.e., (4.42b) becomes

$$\left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_{k+1/2}} = -\frac{GH}{k}. \tag{4.60}$$

Assuming that $\sigma_k - \sigma_{k-1} = \sigma_{k+1} - \sigma_k = \Delta\sigma$ and inserting (4.42a) and (4.60) into (4.42c), the second partial derivative becomes

$$\left.\frac{\partial^2 T}{\partial \sigma^2}\right|_{\sigma_N} = \left(-\frac{GH}{k} - \frac{T_N - T_{N-1}}{\Delta\sigma}\right)/\Delta\sigma = -\frac{GH}{k\Delta\sigma} - \frac{T_N - T_{N-1}}{\Delta\sigma^2}. \tag{4.61}$$

Inserting the new conduction term and replacing the derivative of the vertical advection term with the Neumann boundary condition, (4.57a) becomes

$$\frac{T_{N,t+1}-T_{N,t}}{\Delta t} = -a\left(\frac{GH}{k\Delta\sigma} + \frac{T_{N,t}-T_{N-1,t}}{\Delta\sigma^2}\right) + b_{1\,N,t} + b_{2\,N}T_{N,t} + \Phi_N - c_N\frac{GH}{k}, \tag{4.62a}$$

and similarly for (4.57b):

$$\frac{T_{N,t+1}-T_{N,t}}{\Delta t} = -a\left(\frac{GH}{k\Delta\sigma}+\frac{T_{N,t+1}-T_{N-1,t+1}}{\Delta\sigma^2}\right)+b_{1N,t+1}+b_{2N}T_{N,t+1} + \Phi_N - c_N\frac{GH}{k}. \tag{4.62b}$$

The elements of the tridiagonal system at the lower boundary are then

$$\alpha_N = -\frac{a\Delta t}{2(\sigma_N-\sigma_{N-1})^2}, \tag{4.63a}$$

$$\beta_N = 1-\alpha_N+\frac{1}{2}b_{2N}\Delta t, \tag{4.63b}$$

$$\gamma_N = 0, \tag{4.63c}$$

$$\begin{aligned}\delta_N = &-\alpha_N T_{N-1,t}+(2-\beta_N)T_{N,t}-a\frac{GH\Delta t}{k(\sigma_N-\sigma_{N-1})}\\ &+\frac{1}{2}(b_{1N,t}+b_{1N,t+1})\Delta t+\Phi_N\Delta t-c_N\frac{GH\Delta t}{k}.\end{aligned} \tag{4.63d}$$

4.3 Basal Boundary Condition

This section describes the formulation of the basal boundary condition. An interface for the upper boundary condition (atmospheric BC) is easily defined by the surface temperature and mass balance. Similarly, the basal boundary consists of mechanical and thermal boundary conditions. The complications arise because the thermal and mechanical boundary conditions depend on each other. The interface of the basal boundary can be described with the following fields (see also Fig. 4.8):

1. **basal traction:** This field specifies a parameter which is used to allow basal sliding.

2. **basal heat flux:** This is the heat flux entering the ice sheet from below.

3. **basal water depth:** The presence of basal melt water affects the basal ice temperature.

Also, the ice sheet model calculates a melt/freeze rate based on the temperature gradient and basal water depth. This is handled by Glide.

4.3.1 Mechanical boundary conditions

If the ice is not frozen to the bed, basal décollement may occur. This can be parameterized by a traction factor, t_b. Within the ice sheet model, t_b can be used to calculate basal sliding velocities $\boldsymbol{u}_b$ in the case of zero–order physics. That is,

$$\boldsymbol{u}_b = t_b\boldsymbol{\tau}_b, \tag{4.64}$$

where τ_b is the basal shear stress. Alternatively, t_b can be used as part of the stress–balance calculations when the model is used with higher order physics. In simple models t_b may be uniform or prescribed as a spatial variable. More complex models may wish to make t_b dependent on other variables, such as basal melt rate. Typically t_b will depend on the presence of basal water.

The second mechanical boundary condition, basal melting/freeze–on M_b, is handled within the ice sheet model. The details are described in Section 4.3.2.

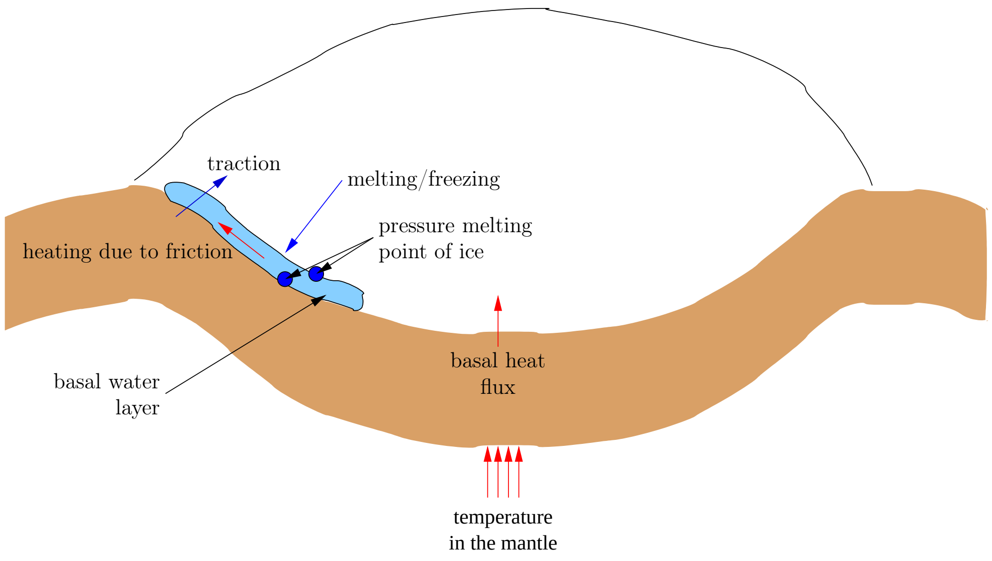


Figure 4.8: Basal boundary condition.

4.3.2 Thermal boundary conditions

The thermal boundary condition at the ice base is more complicated than the mechanical BC. The ice is heated from below by the geothermal heat flux. Heat is generated by friction with the bed. Furthermore, the ice temperature is constrained to be less than or equal to the pressure melting point of ice. The thermal boundary condition is a flux condition if there is no water present. If there is water, the basal temperature is set to the pressure melting temperature. (If it were lower, there would be no water, and if it were higher, there would be no ice.)

Basal melting and freezing

At the ice base, $z = b$, we can define outgoing and incoming heat fluxes, H_o and H_i:

$$H_o = -k_{\text{ice}} \left. \frac{\partial T}{\partial z} \right|_{z=b^+} \tag{4.65a}$$

and

$$H_i = -k_{\text{rock}} \left. \frac{\partial T}{\partial z} \right|_{z=b^-} + \boldsymbol{u}_b \cdot \boldsymbol{\tau}_b + \begin{cases} \rho_{\text{ice}} M_b / L & \text{when } M_b < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.65b}$$

where k_{ice} and k_{rock} are the thermal conductivities of ice and rock, $\boldsymbol{u}_b \cdot \boldsymbol{\tau}_b$ is the heat generated by friction with the bed, and L is the latent heat of fusion. The basal melt/freeze–on rate M_b can then be calculated from the difference between the incoming and outgoing heat fluxes:

$$M_b = \frac{H_i - H_o}{\rho_{\text{ice}} L} \tag{4.66}$$

There is freeze–on if $M_b < 0$, and basal melting if $B_b > 0$.

Geothermal heat flux

The heat flux accross the basal boundary depends on past temperature variations since temperature perturbations penetrate the bedrock if the ice is frozen to the ground (Ritz, 1987). The heat equation for the bedrock layer is given by the diffusion equation

$$\frac{\partial T}{\partial t} = \frac{k_{\text{rock}}}{\rho_{\text{rock}} c_{\text{rock}}} \nabla^2 T = \frac{k_{\text{rock}}}{\rho_{\text{rock}} c_{\text{rock}}} \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} + \frac{\partial^2 T}{\partial z^2} \right), \tag{4.67}$$

where k_{rock} is the thermal conductivity, ρ_{rock} the density, and c_{rock} the specific heat capacity of the bedrock layer.

Initial conditions for the temperature field T are found by applying the geothermal heat flux G to an arbitrary surface temperature T_0:

$$T(x, y, z) = T_0 + \frac{G}{k_{\text{rock}}} z. \tag{4.68}$$

This ensures that initially the geothermal heat flux experienced by the ice sheet is equal to the regional heat flux. The basal boundary condition of the bedrock layer is kept constant, i.e.

$$T(x, y, H_{\text{rock}}) = T_0 + \frac{G}{k_{\text{rock}}} H_{\text{rock}}. \tag{4.69}$$

Lateral boundary conditions are given by

$$\left.\frac{\partial T}{\partial x}\right|_{x=0} = \left.\frac{\partial T}{\partial x}\right|_{x=L_x} = \left.\frac{\partial T}{\partial y}\right|_{y=0} = \left.\frac{\partial T}{\partial y}\right|_{y=L_y} = 0. \tag{4.70}$$

At the upper boundary, the heat flux of the rock layer has to be matched with the heat flux in the basal ice layer when the ice is frozen to the bed, i.e.

$$k_{\text{rock}} \left.\frac{\partial T}{\partial z}\right|_{z=-0} = k_{\text{ice}} \left.\frac{\partial T}{\partial z}\right|_{z=+0}. \tag{4.71}$$

Otherwise the temperature of the top bedrock layer is set to the surface temperature (if the cell has been occupied by ice, but there is no ice present) or the basal ice temperature (if there is ice). Equation (4.71) is automatically fulfilled if we set the top bedrock temperature to the basal ice temperature *everywhere* and then calculate the geothermal heat flux to be used as boundary condition for (4.38).

4.3.3 Numerical solution

The horizontal grid of the bedrock layer is described in Section 4.1.1. The vertical bedrock grid is irregular like the vertical grid of the ice sheet. However, it is not scaled. Also for now, we ignore topography or isostatic adjustment, i.e. the bedrock layer is assumed to be flat and constant.

The horizontal second derivative in (4.67) becomes using finite–differences

$$\left.\frac{\partial^2 T}{\partial x^2}\right|_{x_i, y_i, z_i} = T_{xx,i,j,k} = \frac{T_{i+1,j,k} - 2T_{i,j,k} + T_{i-1,j,k}}{\Delta x}, \tag{4.72}$$

and similarly for $\partial^2 T / \partial y^2$. The vertical second derivative $\partial^2 T / \partial z^2$ is similar to (4.43):

$$\left.\frac{\partial^2 T}{\partial z^2}\right|_{x_i, y_i, z_i} = T_{zz,i,j,k} = \frac{2T_{i,j,k-1}}{(z_k - z_{k-1})(z_{k+1} - z_{k-1})} - \frac{2T_{i,j,k}}{(z_{k+1} - z_k)(z_k - z_{k-1})} + \frac{2T_{i,j,k+1}}{(z_{k+1} - z_k)(z_{k+1} - z_{k-1})}. \tag{4.73}$$

Using the Crank-Nicholson scheme, (4.67) becomes

$$\frac{T^{t+1}_{i,j,k} - T^t_{i,j,k}}{\Delta t} = D \left\{ \frac{T^{t+1}_{xx,i,j,k} + T^t_{xx,i,j,k}}{2} + \frac{T^{t+1}_{yy,i,j,k} + T^t_{yy,i,j,k}}{2} + \frac{T^{t+1}_{zz,i,j,k} + T^t_{zz,i,j,k}}{2} \right\}, \tag{4.74}$$

with $D = k_{\text{rock}}/(\rho_{\text{rock}} c_{\text{rock}})$. Equation (4.74) is solved by gathering all T^{t+1} terms on the LHS and all other terms on the RHS. The index (i, j, k) is linearized using $\iota = i+(j-1)N+(k-1)NM$. The resulting matrix system is solved using the same bi–conjugate gradient solver as for the ice thickness evolution.

4.3.4 Basal hydrology

It is clear from the discussion above that the presence of basal water plays a crucial role in specifying both the mechanical and thermal boundary conditions. However, the treatment of basal water can vary greatly. Basal water is, therefore, left as an unspecified interface. Glide does provide a simple local water balance model which can be run in the absence of more complex models.

4.3.5 Putting it all together

The basal boundary consists of the individual components described in the previous sections. All components are tightly linked with each other. Figure 4.9 illustrates how the modules are linked and in what order they are resolved. The order of executions is then:

1. Find the basal heat flux by either solving the equation describing the thermal evolution of the lithosphere, (4.67), or by using the geothermal heat flux directly. The upper boundary condition of (4.67) is the same as the lower boundary condition of the thermal evolution of the ice sheet.

2. Either (1) the lower boundary condition for the thermal evolution of the ice sheet is given by the basal heat flux from *Step 1*; or (2) if melt water is present, the basal temperature is set to the pressure melting point of ice.

3. Calculate the temperature distribution within the ice sheet given the boundary condition found during *Step 2* and the atmospheric BC.

4. Calculate a melt/freeze–on rate using Equation (4.66) given the outgoing heat flux calculated during *Step 3*, friction with the bed (calculated during the previous *Step 7*) and the incoming heat flux from *Step 1*. Freezing occurs only when there is basal water.

5. Track basal water. This is a user supplied module which can take any complexity. Inputs will typically be the melt/freeze–on rate determined during *Step 4*.

6. Calculate the basal traction parameter. Again, this is a user supplied module which typically will involve the presence of basal water (calculated during *Step 5*).

7. Solve the mechanical ice equations given basal traction parameter from *Step 6*.

Clearly, this scheme has the problem that heat is lost if the basal heat flux is such that more water could be frozen than is available. This might be avoided by iterating the process. On the other hand, the heat loss may be negligible if time steps are fairly small.

4.4 Isostatic Adjustment

The ice sheet model includes simple approximations for calculating isostatic adjustment. These approximations depend on how the lithosphere and the mantle are treated. For each subsystem there are two models. The lithosphere can be described as a

local lithosphere: the flexural rigidity of the lithosphere is ignored, i.e. this is equivalent to ice floating directly on the asthenosphere; or

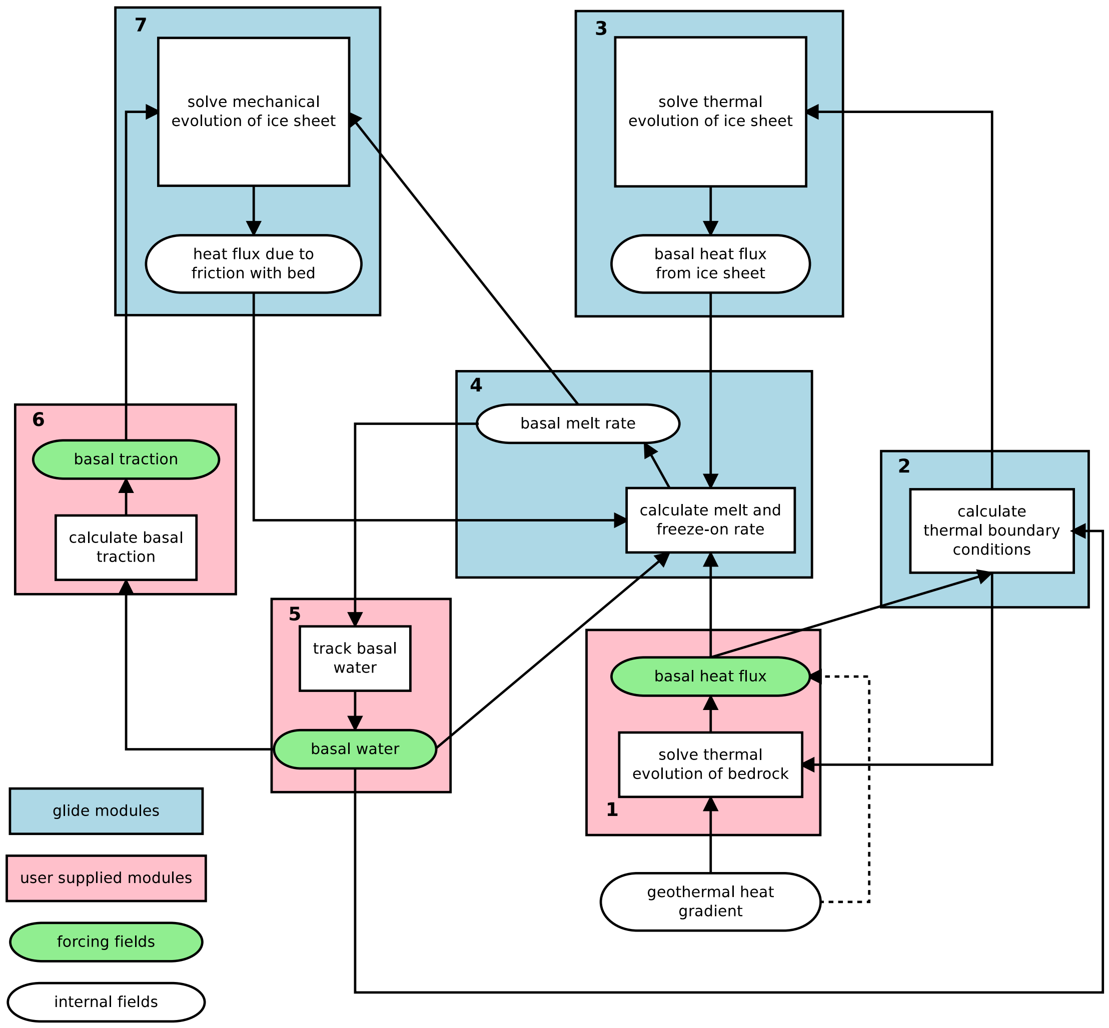

7
solve mechanical evolution of ice sheet
heat flux due to friction with bed
3
solve thermal evolution of ice sheet
basal heat flux from ice sheet
6
basal traction
calculate basal traction
4
basal melt rate
calculate melt and freeze-on rate
2
calculate thermal boundary conditions
5
track basal water
basal water
basal heat flux
solve thermal evolution of bedrock
1
geothermal heat gradient
glide modules
user supplied modules
forcing fields
internal fields

Figure 4.9: Flow diagram illustrating how the various modules communicate with each other by exchanging data fields.

elastic lithosphere: the flexural rigidity is taken into account;

while the mantle is treated as a

fluid mantle: the mantle behaves like a non-viscous fluid, with isostatic equilibrium reached instantaneously; or

relaxing mantle: the flow within the mantle is approximated by an exponentially decaying hydrostatic response function, i.e. the mantle is treated as a viscous half space.

4.4.1 Calculation of ice-water load

At each isostasy time step, the load of ice and water is calculated as an equivalent mantle-depth (L). If the basal elevation is above sea level, the load is simply due to the ice:

$$L = \frac{\rho_i}{\rho_m} H, \tag{4.75}$$

where H is the ice thickness, with ρ_i and ρ_m being the densities of the ice and mantle respectively. In the case where the bedrock is below sea level, the load is calculated is that due to a change in sea level and/or the presence of non-floating ice. When the ice is floating ($\rho_i H < \rho_o(z_0 - h)$), the load is due only to sea-level changes:

$$L = \frac{\rho_o}{\rho_m} z_0, \tag{4.76}$$

whereas when the ice is grounded, it displaces the water and adds an additional load:

$$L = \frac{\rho_i H + \rho_o b_r}{\rho_m}. \tag{4.77}$$

Here, ρ_o is the density of sea water, z_0 is the change in sea-level relative to a reference level, and b_r is the bedrock elevation relative to the same reference level. The value of b_r will be negative for submerged bedrock, hence the plus sign in (4.77).

4.4.2 Elastic lithosphere model

The elastic model is selected by setting `lithosphere = 1` in the configuration file. By simulating the deformation of the lithosphere, the deformation seen by the asthenosphere beneath is calculated. In the absence of this model, the deformation is that due to Archimedes' Principle, as though the load were floating on the asthenosphere.

The elastic lithosphere model is based on work by Lambeck and Nakiboglu (1980), and its implementation is fully described in Hagdorn (2003). The lithosphere model affects only the geometry of the deformation; the timescale for isostatic adjustment is controlled by the asthenosphere model.

The load due to a single (rectangular) grid point is approximated as being applied to a disc of the same area. The deformation due to a disc of ice of radius A and thickness H is given by the following expressions. For $r < A$:

$$w(r) = \frac{\rho_i H}{\rho_m}\left[1 + C_1 \operatorname{Ber}\left(\frac{r}{L_r}\right) + C_2 \operatorname{Bei}\left(\frac{r}{L_r}\right)\right], \tag{4.78}$$

and for $r \geq A$:

$$w(r) = \frac{\rho_i H}{\rho_m}\left[D_1 \operatorname{Ber}\left(\frac{r}{L_r}\right) + D_2 \operatorname{Bei}\left(\frac{r}{L_r}\right) + D_3 \operatorname{Ker}\left(\frac{r}{L_r}\right) + D_4 \operatorname{Kei}\left(\frac{r}{L_r}\right)\right], \tag{4.79}$$

where Ber(x), Bei(x), Ker(x) and Kei(x) are Kelvin functions of zero order, $L_r = (D/\rho_m g))^{1/4}$ is the radius of relative stiffness, and D is the flexural rigidity. The constants C_i and D_i are given by

$$\begin{aligned} C_1 &= a\,\mathrm{Ker}'(a), \\ C_2 &= -a\,\mathrm{Ker}'(a), \\ D_1 &= 0, \\ D_2 &= 0, \\ D_3 &= a\,\mathrm{Ber}'(a), \\ D_4 &= -a\,\mathrm{Ber}'(a). \end{aligned} \tag{4.80}$$

Here, the prime indicates the first spatial derivative of the Kelvin functions.

4.4.3 Relaxing asthenosphere model

If a fluid mantle is selected, it adjusts instantly to changes in lithospheric loading. However, a relaxing mantle is also available.

4.5 Time Step Ordering

Relative to Glimmer-CISM 1.x, the order of operations on each time step has been somewhat reorganized in CISM2.0. This was done for consistency with the ordering of operations used in Glissade and to eliminate discrepancies between time levels applied in the model and the time levels written in output files. **These changes have only a minor impact on model results, but do result in output that will not be an exact match between the two model versions for the same configuration and initial conditions.**

While unlikely to be of much interest to the average user, particularly if you are not migrating from Glimmer-CISM 1.x, the details of these changes follow. Specifically, in Glimmer-CISM 1.x, for a given time step, H is advanced in time relative to T and $\boldsymbol{v}$, and, because a complete solve occurs even at the initial time, the output for the initial time is different than the input for the prognostic variables H and T, even though no time step has occurred at that point.

In Glimmer-CISM 1.x, the time-stepping loop is organized as follows:

```
glide_initialise(): init T, H, v
do while (t < tend)
   glide_tstep_p1(): solve T
   glide_tstep_p2(): solve v, H
   glide_tstep_p3(): calculate some diagnostic variables, write output for time t
   advance time
end do
```

This results in the following relation between the state of model variables and the time level at which they are output:

Output time:	0	1	2
T	1	2	3
H	1	2	3
v	0	1	2

In CISM 2.0, the time-stepping loop is now organized as follows:

```
initialize modules
initial diagnostic solve of v (and any other diagnostic variables like upper surface)
write output for time 0
do while (t <= tend)
   advance time
   solve column physics: T = f(H,T)
   solve prognostic variables: advection of H=f(H,v), T=f(H,v)
   solve diagnostic variables: v=f(H,T)
   write output for time t
end do
```

This results in the following relation between the state of model variables and the time level at which they are output:

Output time:	0	1	2
T	0	1	2
H	0	1	2
v	0	1	2

Chapter 5

Introduction to Higher-Order Ice Dynamics

5.1 Higher-Order Dynamics in CISM

5.1.1 Basics

The main distinction between so-called higher-order models and 0-order (or "shallow ice") models is that higher-order models attempt a closer approximation to solving the nonlinear Stokes equations. This usually means incorporating some approximation of horizontal-stress gradients (along-flow stretching or compression and across-flow shearing) in addition to the vertical stress gradients that are accounted for in shallow ice models (Figure 5.1). This is important for several reasons:

- For the ice-sheet regions of greatest interest – e.g. ice streams, ice shelves, and other regions of fast flow – horizontal-stress gradients are at least as important as vertical stress gradients. To model the flow accurately in these regions, higher-order models are required.
- The shallow-ice approximation, applied to situations in which there is basal sliding, gives rise to a singularity in the vertical velocity. Models compute the vertical velocity by integrating the incompressibility equation:

  $$\frac{\partial w}{\partial z} = -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}. \tag{5.1}$$

  If there is a jump from no sliding in one grid cell to sliding in an adjacent cell, the horizontal velocity gradients at the bed will depend on the grid spacing; the horizontal gradients (and through incompressibility, the vertical velocity gradient and thus the vertical velocity) will become increasingly large as the grid spacing decreases. Obviously, this should be avoided.
- Near the grounding line (the boundary between grounded and floating ice), horizontal rather than vertical stresses often control the flow. Incomplete knowledge of the stresses makes it unlikely that shallow ice models will ever be able to accurately simulate grounding-line advance and retreat.
- In some regions of very slow flow, like ice divides, horizontal-stress gradients are important or dominant. Ice cores are often recovered at ice divides, and flow modeling is important for interpreting ice core records and using information (such as layer thickness) to infer the past flow history in the region. In order to model that flow correctly, one must include horizontal stresses. (At an ice divide the surface slope is ∼0, in which case vertical stress

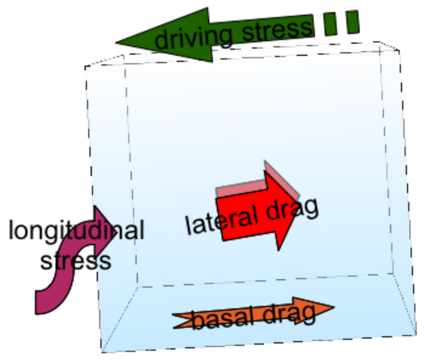


Figure 5.1: The gravitational stress available to move the ice is the driving stress, indicated in green. Because the ice is assumed to be in equilibrium, the sum of the other stresses is equal to (i.e., must balance) the gravitational driving stress. In the 0-order model (shallow-ice approximation) the driving stress is assumed to be balanced by basal drag alone. In higher-order models, this restriction is relaxed and the balance of stresses now includes lateral and/or longitudinal stresses. Because these stresses must be computed based on conditions outside the local ice column, the model becomes significantly more complex.

gradients that drive deformation in 0-order models are also ~0. In reality, deformation is not 0 at ice divides, but is controlled by horizontal stretching rather than vertical shearing).

The term "higher-order" comes from scaling analyses of the Stokes equations for which a scaling parameter $\lambda = H/L$ (the ratio of the thickness to the horizontal length scale of interest) is used to assign importance to the various terms. Shallow ice models retain only terms of order 0 while higher-order models also retain terms of order 1 (and possibly more) (Figure 5.2).

5.1.2 Available schemes

- The most fundamental higher-order scheme is the full set of nonlinear Stokes equations. Because of the computational burden, many large-scale 3D models solve lower-order approximations of the Stokes equations (Figure 5.3). However, several groups have made significant advances in Stokes modeling (e.g. the ELMER-Ice effort; Gagliardini et al. (2013)). Current DOE-funded efforts include implementation of a nonlinear Stokes model on unstructured grids (Leng et al., 2012).

- Probably the most long-lived higher-order approximation in glaciology is the shallow-shelf approximation (or SSA) describing flow within an ice shelf. While not truly "higher-order" when only applied to ice shelf flow, it can provide nearly equivalent results to "true" higher-order models in cases where basal drag is small but non-zero (e.g., for modeling the flow of ice streams). The SSA was developed and made popular by Douglas MacAyeal (e.g., Macayeal (1989)). Its main disadvantage is that it is not fully 3D, as it assumes uniform velocity throughout the ice thickness driven only by horizontal stress gradients. It is adequate for describing fast flow in many parts of ice sheets, such as ice shelves and some ice streams. In these regions, not resolving vertical gradients is a computational advantage.

- The SSA equations are actually a depth-averaged form of a more general higher-order model, commonly referred to as the "Blatter-Pattyn model" (Blatter (1995); Pattyn

$$\boldsymbol{\tau} - P\boldsymbol{I} = 2\eta \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u}{\partial z} + \boldsymbol{\frac{\partial w}{\partial x}}\right) \\ \frac{1}{2}\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right) & \frac{\partial v}{\partial y} & \frac{1}{2}\left(\frac{\partial v}{\partial z} + \boldsymbol{\frac{\partial w}{\partial y}}\right) \\ \frac{1}{2}\left(\boldsymbol{\frac{\partial w}{\partial x}} + \frac{\partial u}{\partial z}\right) & \frac{1}{2}\left(\boldsymbol{\frac{\partial w}{\partial y}} + \frac{\partial v}{\partial z}\right) & \frac{\partial w}{\partial z} \end{pmatrix}$$

$$\boldsymbol{\tau} - P\boldsymbol{I} = 2\eta \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) & \frac{1}{2}\left(\frac{\partial u}{\partial z}\right) \\ \frac{1}{2}\left(\frac{\partial v}{\partial x} + \frac{\partial u}{\partial y}\right) & \frac{\partial v}{\partial y} & \frac{1}{2}\left(\frac{\partial v}{\partial z}\right) \\ \frac{1}{2}\left(\frac{\partial u}{\partial z}\right) & \frac{1}{2}\left(\frac{\partial v}{\partial z}\right) & -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y} \end{pmatrix}$$

Figure 5.2: Stokes-flow (top) and first-order (bottom) constitutive equations, which relate stresses to strain rates (and in turn to velocity gradients). τ is the deviatoric stress tensor, $\boldsymbol{I}$ is the identity tensor, P is the pressure, and η is the effective ice viscosity. The strain rate tensor is given by the term in parentheses on the right-hand side of the equation. For the first-order approximation (bottom), the bold terms from the Stokes equations are assumed negligible.

(2003)). Blatter-Pattyn dynamics, synonymous with and more formally described as the "first-order accurate Stokes approximation"[1], are implemented in CISM's default higher-order dynamical core, Glissade.

- Another formally first-order accurate Stokes approximation, which might be considered "quasi" three-dimensional, is the so-called "L1L2" approximation (Schoof and Hindmarsh, 2010). This model combines a two-dimensional SSA solve and a one-dimensional (column) SIA solve in a mathematically rigorous way. The model solution, from a combination of SSA (for approximating membrane stresses) and SIA (for vertical shear stresses) solutions, mimics a fully 3D, higher-order model solution but at a fraction of the computational cost.

- Several "hybrid" schemes exist that are also computationally cheaper than the Blatter-Pattyn model. These are similar to the L1L2 model in spirit, but combine solutions to the SSA and SIA models in a heuristic fashion. Bueler and Brown (2009) and Pollard and Deconto (2009) describe large-scale models using distinct hybrid approaches.

Review papers by Schoof and Hewitt (2013) and Kirchner et al. (2011) go into a fair amount of detail about the various ice flow modeling approximations, their derivations, and their applicability.

5.1.3 Shallow-ice vs. higher-order models: practical differences

There are large differences between shallow-ice and higher-order models in **(1)** how the momentum equations (the dynamics, or stress balance) are solved and **(2)** how the information derived from that solution (the kinematics, or velocity fields) is used to evolve the ice sheet geometry in time. There are two main reasons for these differences:

[1]See Schoof and Hindmarsh (2010) and Dukowicz et al. (2010) for a more complete scaling analysis and derivation of the first-order Stokes approximation.

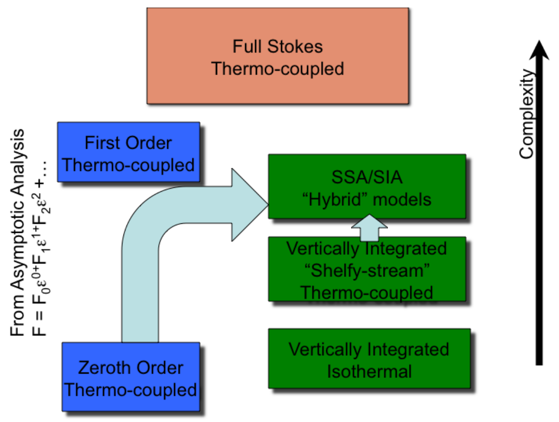

Full Stokes Thermo-coupled
First Order Thermo-coupled
SSA/SIA "Hybrid" models
Vertically Integrated "Shelfy-stream" Thermo-coupled
Vertically Integrated Isothermal
Zeroth Order Thermo-coupled
From Asymptotic Analysis
F = F0ε0+F1ε1+F2ε2+...
Complexity

Figure 5.3: The relationship between several common varieties of ice sheet modesl. Complexity increases along the vertical axis.

- The numerical solution of the dynamical equations is fundamentally different. For the shallow-ice case, we need only local information (surface elevation and thickness) to solve for the velocity as a function of depth in a single column of ice. We do this pointwise for each location on the model domain (in map view), which is a relatively easy numerical problem. Each column of velocities leads to a tridiagonal matrix that is easy to invert (i.e., to solve for the velocities). This problem is *embarrassingly parallel*, since each column of unknown velocities results in a tridiagonal matrix that can be solved on a single processor. For higher-order models, however, the solution at any point also depends on the solution at neighboring points (in map view), leading to an elliptic system of equations in which every velocity must be solved for simultaneously with every other velocity. The result is a much larger system of equations, which are more difficult to solve on one processor and far more difficult to solve on multiple processors. Because large-scale higher-order model applications (e.g., whole-ice-sheet models) require efficient, robust solution and parallelization techniques, this is an active area of current research.

- The equations governing dynamics AND evolution in a shallow-ice model can be recast as a nonlinear diffusion equation for ice thickness. A single system of equations is solved to calculate the velocity field and evolve the ice sheet geometry. For higher-order models, we must first solve the momentum balance equations to obtain the velocity field, and then use some other scheme to evolve the ice thickness.

These differences imply that a higher-order model must be built in a fundamentally different way than a shallow-ice model. Most recent development work on CISM has focused on higher-order dynamics.

Using the constitutive law from the lower panel of Figure 5.2, the equations describing 3D higher-order flow[2] are:

$$\begin{aligned}
x &: \frac{\partial}{\partial x}\left(2\eta\left(2\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}\right)\right)+\frac{\partial}{\partial y}\left(\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{\partial}{\partial z}\left(\eta\frac{\partial u}{\partial z}\right)=\rho_i g\frac{\partial s}{\partial x},\\
y &: \frac{\partial}{\partial y}\left(2\eta\left(2\frac{\partial v}{\partial y}+\frac{\partial u}{\partial x}\right)\right)+\frac{\partial}{\partial x}\left(\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{\partial}{\partial z}\left(\eta\frac{\partial v}{\partial z}\right)=\rho_i g\frac{\partial s}{\partial y}.
\end{aligned} \tag{5.2}$$

[2]Specifically, the first-order accurate Stokes approximation discussed above. These will be derived in more detail in Section 5.2.

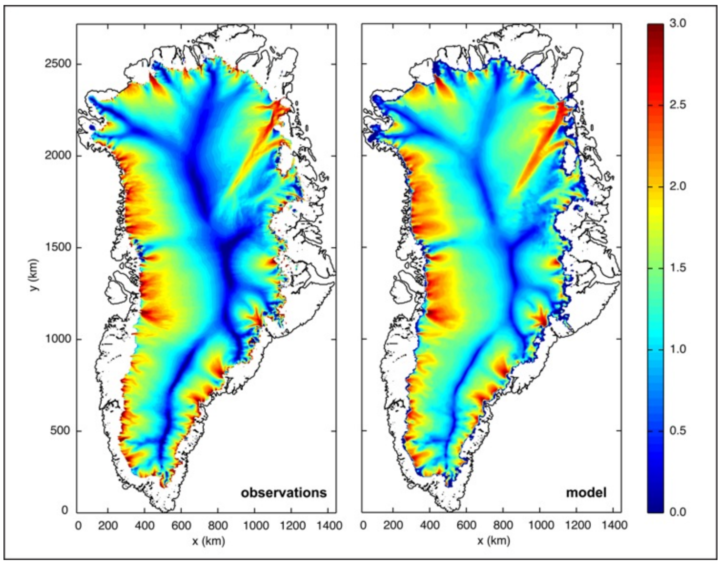


Figure 5.4: Observation-based balance velocities for Greenland (left) and depth-averaged speed from higher-order CISM (right) with basal sliding coefficients optimized to match the balance velocities (after Price et al. (2011)).

In these equations we can point out several important differences between shallow-ice and higher-order models:

1. The third term on the left-hand side represents the vertical diffusion of horizontal velocities. In a shallow-ice model, this term alone accounts for all ice dynamics and balances the entire body force in the x direction (the term on the right-hand side). Since the velocity gradients are only in the vertical direction, the solution is local (in map view).

2. The first-order equations must be solved for each of a set of horizontal layers (the number of which is defined by the resolution in the vertical, here dz). These layers communicate with each other through the vertical diffusion term.

3. The first and second sets of terms on the left-hand side represent the resistance to the body-force term from "membrane stresses" (i.e., stresses arising from horizontal velocity gradients). These horizontal gradients make the problem elliptic and non-local. The absence of membrane stresses in the shallow-ice momentum balance is the primary reason for their failure to realistically simulate the flow in outlet glaciers, ice streams, and ice shelves.

Other complications arise when we account for boundary conditions at the lateral margins of the domain and at the upper and lower ice surfaces. We describe both the governing equations and the boundary conditions in more detail below.

5.2 Higher-Order Momentum Balance

CISM'S higher-order dynamics scheme (some output from which is shown in Figure 5.4) is discussed in more detail in the following sections. First we describe the derivation of the equations themselves, followed by a discussion and derivation of the boundary conditions. We then describe generic solution methods for the nonlinear system of equations, followed by a brief discussion of the solution of the thickness evolution equation.

The higher-order scheme is based on the 3D first-order accurate Stokes approximation (often referred to as the "Blatter-Pattyn" model). The starting point is the nonlinear Stokes equations:

$$\begin{aligned} x: &\quad \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} - \frac{\partial P}{\partial x} = 0, \\ y: &\quad \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} - \frac{\partial P}{\partial y} = 0, \\ z: &\quad \frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z} - \frac{\partial P}{\partial z} = \rho_i g, \end{aligned} \tag{5.3}$$

where P is the pressure and τ is the deviatoric stress tensor. The latter is given by $\tau_{ij} = \sigma_{ij} + P\delta_{ij}$, where σ is the full stress tensor.

There are a number of ways to argue that, due to the shallowness of ice sheets (i.e., the small value of H/L, where H is the thickness and L is a relevant horizontal length scale), the Stokes equations can be reduced to the following first-order approximation:

$$\begin{aligned} x: &\quad \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} - \frac{\partial P}{\partial x} = 0, \\ y: &\quad \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yz}}{\partial z} - \frac{\partial P}{\partial y} = 0, \\ z: &\quad \frac{\partial \tau_{zz}}{\partial z} - \frac{\partial P}{\partial z} = \rho g. \end{aligned} \tag{5.4}$$

The arguments supporting this reduction are fairly complex and are based on either a variational analysis or an asymptotic analysis (see Schoof and Hindmarsh (2010) and Dukowicz et al. (2010) for details).

The third (vertical) balance equation in (5.4) can be integrated vertically to give an expression for the pressure:

$$P = \rho g\,(s - z) + \tau_{zz}(z). \tag{5.5}$$

This is simply a statement that the full vertical normal stress is balanced by the hydrostatic pressure (the so-called *hydrostatic assumption*). This expression can be substituted into the horizontal pressure gradient terms above to remove pressure from the equations. For example, for the x component of velocity we have

$$\begin{aligned} \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} &= \frac{\partial}{\partial x}\left[\rho g\,(s - z) + \tau_{zz}(z)\right] \\ \frac{\partial \tau_{xx}}{\partial x} - \frac{\partial \tau_{zz}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z} &= \rho g \frac{\partial s}{\partial x} \end{aligned} \tag{5.6}$$

Using the incompressibility constraint (5.1) and the assumption that stress and strain are aligned we can write

$$\tau_{zz} = -\tau_{xx} - \tau_{yy}. \tag{5.7}$$

Taking the gradient of (5.7) with respect to x, we can rewrite the vertical normal deviatoric stress in terms of horizontal normal deviatoric stresses:

$$-\frac{\partial \tau_{zz}}{\partial x} = -\frac{\partial}{\partial x}\left(-\tau_{xx} - \tau_{yy}\right) = \frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{yy}}{\partial x}. \tag{5.8}$$

Substituting (5.8) into (5.6), we obtain

$$\frac{\partial}{\partial x}\left(2\tau_{xx}+\tau_{yy}\right)+\frac{\partial \tau_{xy}}{\partial y}+\frac{\partial \tau_{xz}}{\partial z}=\rho g\frac{\partial s}{\partial x}. \tag{5.9}$$

Similarly, the y horizontal balance equation is

$$\frac{\partial}{\partial y}\left(2\tau_{yy}+\tau_{xx}\right)+\frac{\partial \tau_{xy}}{\partial x}+\frac{\partial \tau_{yz}}{\partial z}=\rho g\frac{\partial s}{\partial y}. \tag{5.10}$$

At this point we have removed the vertical balance equation entirely; it has been incorporated into the horizontal balance equations through incompressibility.

Next we write these equations in terms of the velocities for which we are ultimately solving. Stresses and velocities are linked through the constitutive law for ice, which relates strain rates to stresses (here we assume Nye's generalization of Glen's law), and through the definition of the strain rate tensor, which relates strain rates to velocity gradients:

$$\begin{aligned}
&1.\quad \tau_{ij}=B\dot{\varepsilon}_e^{\frac{1-n}{n}}\dot{\varepsilon}_{ij},\qquad B=B(T)\\
&2.\quad \dot{\varepsilon}_{ij}=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}+\frac{\partial u_j}{\partial x_i}\right)\\
&3.\quad 2\dot{\varepsilon}_e=\dot{\varepsilon}_{ij}\dot{\varepsilon}_{ij}\\
&4.\quad \eta\equiv\frac{1}{2}B\dot{\varepsilon}_e^{\frac{1-n}{n}}\\
&5.\quad \tau_{ij}=2\eta\dot{\varepsilon}_{ij}
\end{aligned} \tag{5.11}$$

In order, the five expressions above give:

1. Glen's flow law[3], where $B=A^{\frac{-1}{n}}$ is the temperature dependent rate factor
2. The definition of the strain-rate tensor in terms of velocity gradients
3. The definition of the effective strain rate, $\dot{\varepsilon}_e$, a norm of the strain-rate tensor
4. A definition of the "effective viscosity" (after rearranging some terms in (1))
5. Items (1)-(4) allow us to write the relationship between stress and strain in a standard "Newtonian" way, but with a non-Newtonian (nonlinear) viscosity.

Taking these definitions into the stress balance equations (5.9) and (5.10) and expanding in terms of strain rates and effective viscosity, we have (for the x direction only):

$$x:\quad 2\frac{\partial}{\partial x}\left(2\eta\dot{\varepsilon}_{xx}\right)+\frac{\partial}{\partial x}\left(2\eta\dot{\varepsilon}_{yy}\right)+\frac{\partial}{\partial y}\left(2\eta\dot{\varepsilon}_{xy}\right)+\frac{\partial}{\partial z}\left(2\eta\dot{\varepsilon}_{xz}\right)=\rho g\frac{\partial s}{\partial x}, \tag{5.12}$$

Replacing strain-rate components with velocity gradients, we obtain

$$x:\quad \frac{\partial}{\partial x}\left(4\eta\frac{\partial u}{\partial x}+2\eta\frac{\partial v}{\partial y}\right)+\frac{\partial}{\partial y}\left[\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]+\frac{\partial}{\partial z}\left(\eta\frac{\partial u}{\partial z}\right)=\rho g\frac{\partial s}{\partial x}. \tag{5.13}$$

An analogous expression gives the y-direction momentum balance:

$$y:\quad \frac{\partial}{\partial y}\left(4\eta\frac{\partial v}{\partial y}+2\eta\frac{\partial u}{\partial x}\right)+\frac{\partial}{\partial x}\left[\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right]+\frac{\partial}{\partial z}\left(\eta\frac{\partial v}{\partial z}\right)=\rho g\frac{\partial s}{\partial y}. \tag{5.14}$$

These are the basic equations to be discretized and solved.

[3]Technically, we are using the *inverse* form of the flow-law here.

5.3 Higher-Order Model Boundary Conditions

We now carry out an approximate derivation of the boundary conditions in a higher-order scheme. By "approximate" we mean that some of the derivation is guided by physical intuition and reasonable arguments, rather than rigorous mathematics. In the end, we derive the same set of equations as when following the more rigorous approach (e.g., Dukowicz et al. (2010)). We will consider the boundary conditions in three parts: (1) the free surface BC, (2) the basal traction BC, and (3) lateral BCs.

5.3.1 Stress-free surface

At the upper ice surface, a stress-free boundary condition is applied. The traction vector T_i must be continuous across the ice sheet surface and, assuming that atmospheric pressure and surface tension are small, we have

$$\begin{aligned} T_i &= -T_{i(\text{boundary})} \approx 0, \\ T_i &= \sigma_{ij} n_j = \sigma_{i1} n_1 + \sigma_{i2} n_2 + \sigma_{i3} n_3 = 0, \end{aligned} \tag{5.15}$$

where the n_i are the components of the outward-facing, unit normal vector in Cartesian coordinates.

For a function *F(x,y,z) = z – f(x,y) = 0*, where *z = f(x,y)* defines the surface, the gradient of *F(x,y,z)* gives the components of the surface normal vector. For the case of the ice sheet surface, $s = f(x, y)$ and the surface normal is given by

$$n_i = \left(-\frac{\partial s}{\partial x}, -\frac{\partial s}{\partial y}, 1 \right) \frac{1}{a}, \tag{5.16}$$

where

$$a = \sqrt{\left(\frac{\partial s}{\partial x}\right)^2 + \left(\frac{\partial s}{\partial y}\right)^2 + 1}. \tag{5.17}$$

Eq. (5.15) gives three equations that must be satisfied at the free surface:

$$\begin{aligned} i = x: &\quad T_x = \sigma_{xx} n_x + \sigma_{xy} n_y + \sigma_{xz} n_z = 0, \\ i = y: &\quad T_y = \sigma_{yx} n_x + \sigma_{yy} n_y + \sigma_{yz} n_z = 0, \\ i = z: &\quad T_z = \sigma_{zx} n_x + \sigma_{zy} n_y + \sigma_{zz} n_z = 0. \end{aligned} \tag{5.18}$$

Expanding the z equation and expressing stresses in terms of strain rates and pressures, where η is the effective viscosity defined above, gives

$$(2\eta\dot{\varepsilon}_{zx})\, n_x + (2\eta\dot{\varepsilon}_{zy})\, n_y + (2\eta\dot{\varepsilon}_{zz} - P)\, n_z = 0. \tag{5.19}$$

Solving for the pressure gives

$$P n_z = (2\eta\dot{\varepsilon}_{zz})\, n_z + (2\eta\dot{\varepsilon}_{zx})\, n_x + (2\eta\dot{\varepsilon}_{zy})\, n_y. \tag{5.20}$$

Expanding in terms of velocity gradients and normal vector components, we find

$$P = 2\eta\frac{\partial w}{\partial z} - \left(\eta\frac{\partial u}{\partial z}\right)\frac{\partial s}{\partial x} - \left(\eta\frac{\partial v}{\partial z}\right)\frac{\partial s}{\partial y}, \tag{5.21}$$

where we have made the usual first-order approximation that

$$\frac{\partial w}{\partial x} = \frac{\partial w}{\partial y} \approx 0. \tag{5.22}$$

and also assumed $a \approx 1$.

We use this expression for the pressure and expand the two horizontal boundary condition expressions

$$\begin{aligned} i = x: \quad & T_x = \sigma_{xx} n_x + \sigma_{xy} n_y + \sigma_{xz} n_z = 0, \\ i = y: \quad & T_y = \sigma_{yx} n_x + \sigma_{yy} n_y + \sigma_{yz} n_z = 0, \end{aligned} \tag{5.23}$$

in terms of velocity gradients and the effective viscosity to obtain

$$\begin{aligned} \hat{x}: \quad & -2\eta \frac{\partial u}{\partial x}\frac{\partial s}{\partial x} - \eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\frac{\partial s}{\partial y} + \eta\frac{\partial u}{\partial z} = \\ & \qquad - 2\eta\frac{\partial w}{\partial z}\frac{\partial s}{\partial x} + \eta\frac{\partial u}{\partial z}\left[\frac{\partial s}{\partial x}\frac{\partial s}{\partial x}\right] + \eta\frac{\partial v}{\partial z}\left[\frac{\partial s}{\partial y}\frac{\partial s}{\partial x}\right], \\ \hat{y}: \quad & -2\eta \frac{\partial v}{\partial y}\frac{\partial s}{\partial y} - \eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\frac{\partial s}{\partial x} + \eta\frac{\partial v}{\partial z} = \\ & \qquad - 2\eta\frac{\partial w}{\partial z}\frac{\partial s}{\partial y} + \eta\frac{\partial u}{\partial z}\left[\frac{\partial s}{\partial x}\frac{\partial s}{\partial y}\right] + \eta\frac{\partial v}{\partial z}\left[\frac{\partial s}{\partial y}\frac{\partial s}{\partial y}\right]. \end{aligned} \tag{5.24}$$

In both these expressions, the terms in square brackets are ~ 0 (because slopes on ice sheets are small and the slope squared is exceedingly small). From continuity, we also have

$$\frac{\partial w}{\partial z} = -\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}. \tag{5.25}$$

Using this expression for the normal vertical velocity gradient and removing the terms in square brackets, our two horizontal BC expressions become

$$\begin{aligned} \hat{x}: \quad & -2\eta \frac{\partial u}{\partial x}\frac{\partial s}{\partial x} - \eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\frac{\partial s}{\partial y} + \eta\frac{\partial u}{\partial z} = 2\eta\left(\frac{\partial u}{\partial x}\frac{\partial s}{\partial x} + \frac{\partial v}{\partial y}\frac{\partial s}{\partial x}\right), \\ \hat{y}: \quad & -2\eta \frac{\partial v}{\partial y}\frac{\partial s}{\partial y} - \eta\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\frac{\partial s}{\partial x} + \eta\frac{\partial v}{\partial z} = 2\eta\left(\frac{\partial u}{\partial x}\frac{\partial s}{\partial y} + \frac{\partial v}{\partial y}\frac{\partial s}{\partial y}\right). \end{aligned} \tag{5.26}$$

Moving terms to one side and dividing through by the effective viscosity η, we arrive at the final form of the first-order, free surface boundary conditions

$$\begin{aligned} \hat{x}: \quad & 4\frac{\partial u}{\partial x}\frac{\partial s}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial s}{\partial y} + 2\frac{\partial v}{\partial y}\frac{\partial s}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial s}{\partial y} - \frac{\partial u}{\partial z} = 0, \\ \hat{y}: \quad & 4\frac{\partial v}{\partial y}\frac{\partial s}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial s}{\partial x} + 2\frac{\partial u}{\partial x}\frac{\partial s}{\partial y} + \frac{\partial u}{\partial y}\frac{\partial s}{\partial x} - \frac{\partial v}{\partial z} = 0. \end{aligned} \tag{5.27}$$

5.3.2 Specified basal traction

Derivation of the basal boundary condition follows that above for the free surface except that

1. The right-hand side of the equation is not zero, but rather consists of an assumed basal traction vector with components

$$\tau_{bi} = (\tau_{bx}, \tau_{by}). \tag{5.28}$$

2. The outward-facing normal vector components now consist of horizontal gradients in the basal surface (with a resulting sign switch on the x, y, z components relative to the upper surface).

3. the effective viscosity does not disappear from the equations when we divide through.

Making these substitutions, we obtain the first-order basal boundary conditions for a specified basal traction:

$$\begin{aligned} \hat{x}: \quad & 4\frac{\partial u}{\partial x}\frac{\partial b}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial b}{\partial y} + 2\frac{\partial v}{\partial y}\frac{\partial b}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial u}{\partial z} = \frac{\tau_{bx}}{\eta}, \\ \hat{y}: \quad & 4\frac{\partial v}{\partial y}\frac{\partial b}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial b}{\partial x} + 2\frac{\partial u}{\partial x}\frac{\partial b}{\partial y} + \frac{\partial u}{\partial y}\frac{\partial b}{\partial x} - \frac{\partial v}{\partial z} = \frac{\tau_{by}}{\eta}. \end{aligned} \tag{5.29}$$

We assume that basal traction is linearly related to basal sliding velocity through a positive traction parameter, β, such that

$$\tau_{bx} = -\beta u|_{z=b}, \quad \tau_{by} = -\beta v|_{z=b}. \tag{5.30}$$

The minus sign in front of the β indicates that the traction vector is oriented parallel to and in the opposite direction of the sliding velocity vector.

Substituting this expression into (5.29) gives the final horizontal boundary conditions at the ice sheet base:

$$\begin{aligned} \hat{x}: \quad & 4\frac{\partial u}{\partial x}\frac{\partial b}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial b}{\partial y} + 2\frac{\partial v}{\partial y}\frac{\partial b}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial u}{\partial z} + \left(\frac{\beta}{\eta}\right) u = 0, \\ \hat{y}: \quad & 4\frac{\partial v}{\partial y}\frac{\partial b}{\partial y} + \frac{\partial v}{\partial x}\frac{\partial b}{\partial x} + 2\frac{\partial u}{\partial x}\frac{\partial b}{\partial y} + \frac{\partial u}{\partial y}\frac{\partial b}{\partial x} - \frac{\partial v}{\partial z} + \left(\frac{\beta}{\eta}\right) v = 0. \end{aligned} \tag{5.31}$$

The use of a linear traction parameter as in (5.30) can be modified to implement basal friction laws that are a function of effective pressure (ice overburden pressure minus water pressure at the bed) and generally are nonlinear in the sliding velocity. Two such basal friction laws are implemented in CISM. For friction laws that are nonlinear in sliding velocity, β in (5.30) becomes a function of sliding velocity. This is implemented by lagging the value of sliding velocity used to calculate an "effective" β in the fixed-point iterations used to solve the momentum balance.

The first is a Weertman-style friction law (Weertman, 1957, 1964) modified to include an effective pressure dependence (Bindschadler, 1983; Budd et al., 1979). Notation here follows the summary by Cuffey and Paterson (2010, p.240, eq. 7.17). The form of the friction law is:

$$\boldsymbol{u_b} = k\boldsymbol{\tau_b}^p N^{-q}, \tag{5.32}$$

which can be rearranged for $\boldsymbol{\tau_b}$ as:

$$\boldsymbol{\tau_b} = k^{\frac{-1}{p}} \boldsymbol{u_b}^{\frac{1}{p}} N^{\frac{q}{p}}, \tag{5.33}$$

where k is a friction coefficient based on thermal and mechanical properties of ice and is inversely related to bed roughness. Cuffey and Paterson (2010) suggest values of $p = 3$ and $q = 1$, making the basal friction nonlinear in velocity.

One problem with (5.32) is that it allows for unbounded basal traction. CISM also includes a physically-based basal friction law for sliding over hard beds that allows for cavitation and bounded basal drag (Schoof, 2005). In this law, the friction is related to the velocity and effective pressure by

$$\boldsymbol{\tau_b} = C\left(\frac{\boldsymbol{u_b}}{\boldsymbol{u_b} + N^n\Lambda}\right)^{1/n} N, \quad \Lambda = \frac{\lambda_{max} A}{m_{max}}. \tag{5.34}$$

In Eq. 5.34, C is a Coulomb friction coefficient, and λ_{max} and m_{max} are the wavelength (m) and maximum slope, respectively, of the dominant bedrock bumps. Near cavitation (i.e., when effective pressure approaches 0 at high water pressure), the friction law becomes a Coulomb friction law of the form $\boldsymbol{\tau_b} = CN$. Alternatively, at large effective pressures (i.e., low water pressure), the friction law takes a power law form, $\boldsymbol{\tau_b} \propto u_b^{1/n}$. The representation of bump geometry in the friction law is at a sub-grid scale and is not explicitly resolved by the model.

The release version of CISM does not currently include a hydrology model that computes effective pressure, so at present the effective pressure fields must be input as data to use these two basal friction laws.

5.3.3 Specified basal yield stress

The above implementation of a specified basal traction can be altered to simulate sliding over a sediment-covered bed for which the sediment has a plastic or “Coulomb friction” rheology. That is, sliding does not occur below the specified yield stress of the underlying material (e.g., a water-saturated subglacial till). When the yield stress, τ_0 is reached, sliding occurs at a rate that maintains but does not exceed that yield stress. Consider the x–direction boundary condition in (5.29), written out in terms of basal stress components:

$$\begin{aligned} \hat{x}: \quad & 4\eta\frac{\partial u}{\partial x}\frac{\partial b}{\partial x} + \eta\frac{\partial u}{\partial y}\frac{\partial b}{\partial y} + 2\eta\frac{\partial v}{\partial y}\frac{\partial b}{\partial x} + \eta\frac{\partial v}{\partial x}\frac{\partial b}{\partial y} - \eta\frac{\partial u}{\partial z} = \tau_{bx}, \\ & \tau_{bx} \approx \tau_0 = \tau_0\left(\frac{u}{|\mathbf{u}|}\right) = \tau_0\left(\frac{u}{\sqrt{u_0^2+v_0^2+\gamma}}\right). \end{aligned} \tag{5.35}$$

In this expression, u is the x component of the basal sliding velocity from the current iteration, u_0 and v_0 are components from the previous iteration, and γ is a regularization constant (a small number to avoid division by zero during early iterations). As the solution converges, velocities at the current and previous iteration are nearly equal, and the expression in parentheses approaches 1, in which case the basal stress and yield stress are approximately equal.

We can accommodate this plastic behavior in our earlier framework by treating β as nonlinear and dependent on the velocity:

$$\begin{aligned} \tau_{bx} &= -\beta u, \\ \beta &\equiv \frac{\tau_0}{\sqrt{u_0^2+v_0^2+\gamma}}, \\ \tau_{bx} &= -\left(\frac{\tau_0}{\sqrt{u_0^2+v_0^2+\gamma}}\right)u. \end{aligned} \tag{5.36}$$

This expression for nonlinear β can be substituted in the expression (5.31) for the basal boundary conditions. Figure 5.5 shows an example of this type of boundary condition for the case of an idealized ice stream simulated by CISM.

5.3.4 Lateral boundary conditions

Currently, the only significant lateral boundary condition implemented in CISM is that for floating ice; the depth-averaged stress resulting from an adjacent column of ocean water is applied at the location of an ice shelf (or ice tongue) front. The derivation largely follows those for the free surface and specified basal traction boundary conditions, except that the surface normal vectors, n_x and n_y, are taken as lying entirely in the x, y plane (i.e., they are perpendicular to a vertical shelf front). Thus we have

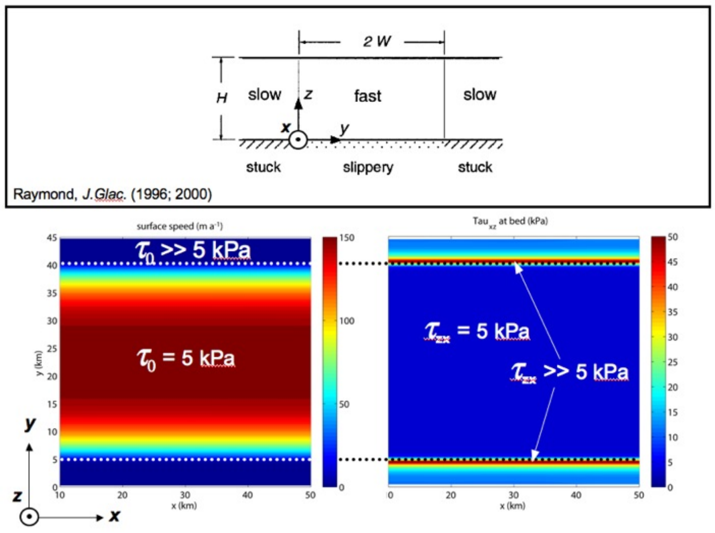


Figure 5.5: Idealized ice stream with plastic-bed sliding. Top panel shows a schematic of an idealized ice stream, frozen at the margins and thawed within the ice stream (flow is out of the page). Bottom panel (color) shows a modeled, idealized ice stream (flow is from left to right) with a yield stress of 5 kPa within the ice stream and much larger than 5 kPa outside of it. Bottom left shows the ice stream speed (m/yr) and the bottom right shows the basal drag (kPa). Within the ice stream, basal drag is equal to the yield stress. Outside the ice stream, stress transfer to the lateral margins results in basal drag that is much larger than the yield stress (and also much larger than the driving stress).

$$\begin{aligned}
\hat{x}: &\quad 4\eta\frac{\partial u}{\partial x}n_x + \eta\frac{\partial u}{\partial y}n_y + \eta\frac{\partial u}{\partial z}n_z = -2\eta\frac{\partial v}{\partial y}n_x - \eta\frac{\partial v}{\partial x}n_y + \rho g\left(s-z\right)n_x - S_x,\\
\hat{y}: &\quad 4\eta\frac{\partial v}{\partial y}n_y + \eta\frac{\partial v}{\partial x}n_x + \eta\frac{\partial v}{\partial z}n_z = -2\eta\frac{\partial u}{\partial x}n_y - \eta\frac{\partial u}{\partial y}n_x + \rho g\left(s-z\right)n_y - S_y.
\end{aligned} \tag{5.37}$$

Here, S_x and S_y are source terms from the pressure due to ocean water (to be defined below), and $\rho g\left(s-z\right)$ comes from including the first-order vertical stress balance:

$$\begin{aligned}
&\frac{\partial \sigma_{zz}}{\partial z} = \frac{\partial \tau_{zz}}{\partial z} - \frac{\partial P}{\partial z} = \rho g \quad \text{(integrate w.r.t. } z\text{)},\\
&P = \rho g\left(s-z\right) + \tau_{zz},\\
&P = \rho g\left(s-z\right) + 2\eta\dot{\varepsilon}_{zz} = \rho g\left(s-z\right) + 2\eta\left(-\frac{\partial u}{\partial x} - \frac{\partial v}{\partial y}\right),
\end{aligned} \tag{5.38}$$

with ρ being the ice density. In the last step above we have used the constitutive relation and incompressibility to expand the vertical, normal-deviatoric stress in terms of the effective viscosity and horizontal-normal strain rates. We calculate the source terms S_x and S_y as the depth-averaged stress at the ice shelf front due to the pressure of ocean water there. This value is given by

$$S_i = \left[\frac{1}{H}\frac{1}{2}\rho_w g\left(H-h_f\right)^2\right]n_i = \left[\frac{1}{2}\rho_w g H\left(\frac{\rho}{\rho_w}\right)^2\right]n_i, \qquad h_f = H\left(1-\frac{\rho}{\rho_w}\right), \tag{5.39}$$

where *i=x,y*, n_i is the shelf-front normal vector, H is the ice thickness, h_f is the "freeboard", or ice thickness above flotation, and ρ_w is the density of ocean water. Because we have taken a depth average for this source term, we take a depth average of the term $\rho g\left(s-z\right)$ above, which is $\frac{1}{2}\rho g H$.

Combining these two terms and inserting them in the horizontal boundary condition expressions above gives

$$\begin{aligned}
\hat{x}: &\quad 4\eta\frac{\partial u}{\partial x}n_x + \eta\frac{\partial u}{\partial y}n_y + \eta\frac{\partial u}{\partial z}n_z =\\
&-2\eta\frac{\partial v}{\partial y}n_x - \eta\frac{\partial v}{\partial x}n_y + \left[-\frac{1}{2}H\left(\frac{\rho}{\rho_w}\right)^2\rho_w g + \frac{1}{2}\rho g H\right]n_x,\\
\hat{y}: &\quad 4\eta\frac{\partial v}{\partial y}n_y + \eta\frac{\partial v}{\partial x}n_x + \eta\frac{\partial v}{\partial z}n_z =\\
&-2\eta\frac{\partial u}{\partial x}n_y - \eta\frac{\partial u}{\partial y}n_x + \left[-\frac{1}{2}H\left(\frac{\rho}{\rho_w}\right)^2\rho_w g + \frac{1}{2}\rho g H\right]n_y,
\end{aligned} \tag{5.40}$$

which can be rearranged to

$$\begin{aligned}
\hat{x}: &\quad 4\frac{\partial u}{\partial x}n_x + \frac{\partial u}{\partial y}n_y + \frac{\partial u}{\partial z}n_z + 2\frac{\partial v}{\partial y}n_x + \frac{\partial v}{\partial x}n_y = \frac{\rho g H}{2\eta}\left(1-\frac{\rho}{\rho_w}\right)n_x,\\
\hat{y}: &\quad 4\frac{\partial v}{\partial y}n_y + \frac{\partial v}{\partial x}n_x + \frac{\partial v}{\partial z}n_z + 2\frac{\partial u}{\partial x}n_y + \frac{\partial u}{\partial y}n_x = \frac{\rho g H}{2\eta}\left(1-\frac{\rho}{\rho_w}\right)n_y.
\end{aligned} \tag{5.41}$$

For an ice shelf, the surface and basal velocities are equal, in which case the vertical velocity gradient terms are ~ 0, giving the final form of the lateral boundary conditions:

$$\begin{aligned}
\hat{x}: &\quad 4\frac{\partial u}{\partial x}n_x + \frac{\partial u}{\partial y}n_y + 2\frac{\partial v}{\partial y}n_x + \frac{\partial v}{\partial x}n_y = \frac{\rho g H}{2\eta}\left(1 - \frac{\rho}{\rho_w}\right)n_x, \\
\hat{y}: &\quad 4\frac{\partial v}{\partial y}n_y + \frac{\partial v}{\partial x}n_x + 2\frac{\partial u}{\partial x}n_y + \frac{\partial u}{\partial y}n_x = \frac{\rho g H}{2\eta}\left(1 - \frac{\rho}{\rho_w}\right)n_y.
\end{aligned} \tag{5.42}$$

5.3.5 Summary

All the boundary conditions above have a similar form. In fact, all that differs among the equations for the free surface, the specified basal traction, and the lateral shelf boundary condition is (1) the definition of the normal vectors and (2) the existence and definition of a source term. Here are the x–direction equations again for the three cases:

$$\begin{aligned}
\hat{x}: &\quad 4\frac{\partial u}{\partial x}\frac{\partial s}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial s}{\partial y} + 2\frac{\partial v}{\partial y}\frac{\partial s}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial s}{\partial y} - \frac{\partial u}{\partial z} = 0, \\
\hat{x}: &\quad 4\frac{\partial u}{\partial x}\frac{\partial b}{\partial x} + \frac{\partial u}{\partial y}\frac{\partial b}{\partial y} + 2\frac{\partial v}{\partial y}\frac{\partial b}{\partial x} + \frac{\partial v}{\partial x}\frac{\partial b}{\partial y} - \frac{\partial u}{\partial z} = \frac{\tau_{bx}}{\eta}, \\
\hat{x}: &\quad 4\frac{\partial u}{\partial x}n_x + \frac{\partial u}{\partial y}n_y + 2\frac{\partial v}{\partial y}n_x + \frac{\partial v}{\partial x}n_y = \frac{\rho g H}{2\eta}\left(1 - \frac{\rho}{\rho_w}\right)n_x.
\end{aligned} \tag{5.43}$$

In the first equation (free surface), the normals are related to the ice sheet surface slope and the source term is zero (which allows us to divide through by the effective viscosity and remove it from the equations). In the second equation (specified basal traction), the normals are related to the bedrock slopes and the source term is related to the assumed relationship between the basal shear stress and the basal sliding rate. In the last equation, the normals are defined by the shape of the ice-shelf front in map view, the vertical velocity gradient terms are absent, and the source term is related to the pressure from the adjacent column of ocean water.

In CISM's earlier developmental dycore, Glam, these boundary conditions were implemented using finite differences, leading to significant complexity. The current dycore, Glissade, solves for ice velocity using a finite-element approach, in which boundary conditions are handled more simply. For example, the surface boundary condition is a "natural" BC that is satisfied automatically. See, e.g., Dukowicz et al. (2010) for details.

5.4 Numerical Solution of Higher-Order Equations

This section describes numerical methods for solving the higher-order equations discussed above. More specific details are given in Chapter 6.

5.4.1 Governing matrix equations

We would like to solve the stress-balance equations (5.13) and (5.14) for the horizontal velocity components u and v. Any number of standard methods (e.g., finite differences, finite volumes, or finite elements) can be used to discretize these governing equations: i.e., to re-formulate them from their continuous, PDE form to a discontinuous, piecewise, algebraic approximation based on a specific computational mesh. In the limit, as the mesh spacing goes to zero, the difference between the solution to the actual PDE and its algebraic approximation also goes to zero. The Glissade dynamical core uses the Finite Element Method (FEM) to discretize the governing equations, as discussed in Chapter 6. The result of the discretization is a set of linear algebraic equations that can be written in matrix form. An example is given below, where the coefficients of u and v are contained in block matrices $\mathbf{A}$, given by

$$\begin{bmatrix}\mathbf{A_{uu}} & \mathbf{A_{uv}}\\ \mathbf{A_{vu}} & \mathbf{A_{vv}}\end{bmatrix}\begin{bmatrix}\mathbf{u}\\ \mathbf{v}\end{bmatrix}=\begin{bmatrix}\mathbf{b_u}\\ \mathbf{b_v}\end{bmatrix},$$
$$\mathbf{A_{uu}u}+\mathbf{A_{uv}v}=\mathbf{b_u}, \qquad \mathbf{A_{vu}u}+\mathbf{A_{vv}v}=\mathbf{b_v}. \tag{5.44}$$

Here the **uu** subscript denotes block matrices containing coefficients for gradients of u in the equation for the x component of velocity (i.e., u). The subscript **uv** denotes block matrices containing coefficients for gradients of v in the equation for the x component of velocity (and similarly for the **vu** and **vv** subscripts). On the right-hand side, the subscripts **u** and **v** are attached to the geometric source terms for the x and y components of velocity, respectively.

5.4.2 Treating nonlinearity through a fixed-point iteration

The nonlinearity of the equations—the fact that the matrix coefficients (in particular, the effective viscosity) depend on the velocity (or more specifically, the velocity gradients)—is handled through a fixed-point iteration. A general fixed-point iteration for a vector of unknowns u can be written as

$$u^k = \mathbf{B}\left(u^{k-1}\right), \tag{5.45}$$

where k is the current iteration index and $\mathbf{B}$ is a matrix operation performed on the components of u obtained at the previous iteration, $k-1$. The fixed point occurs when the values of u at k and $k-1$ are equal to within some given tolerance, at which point the iteration process is halted. For a Picard iteration, which is used in Glissade, the matrix coefficients with a velocity dependence are simply based on the velocities from the previous iteration. In other words, velocities obtained during iteration $k-1$ are used to calculate the strain rates appearing in the expression for effective viscosity at iteration k.

Accounting for the Picard iteration on the effective viscosity, the final form of the matrix equations solved by Glissade becomes

$$\begin{bmatrix}\mathbf{A}^{k-1}_{\mathbf{uu}} & \mathbf{A}^{k-1}_{\mathbf{uv}}\\ \mathbf{A}^{k-1}_{\mathbf{vu}} & \mathbf{A}^{k-1}_{\mathbf{vv}}\end{bmatrix}\begin{bmatrix}\mathbf{u}^k\\ \mathbf{v}^k\end{bmatrix}=\begin{bmatrix}\mathbf{b}^k_{\mathbf{u}}\\ \mathbf{b}^k_{\mathbf{v}}\end{bmatrix},$$
$$\mathbf{A}^{k-1}_{\mathbf{uu}}\mathbf{u}^k+\mathbf{A}^{k-1}_{\mathbf{uv}}\mathbf{v}^k=\mathbf{b}^k_{\mathbf{u}}, \qquad \mathbf{A}^{k-1}_{\mathbf{vu}}\mathbf{u}^k+\mathbf{A}^{k-1}_{\mathbf{vv}}\mathbf{v}^k=\mathbf{b}^k_{\mathbf{v}},$$

These equations form a linear system; for the solution at any particular iteration k, the matrix coefficients that depend on the velocity components u and v are held frozen during the solution. This linear system can be solved using a variety of methods. For large, sparse systems, the preconditioned conjugate gradient (PCG) method or some related method (e.g., preconditioned BiCG or GMRES) is generally the most efficient. (Glissade forms a symmetric positive-definite matrix, as required for the PCG method. $\mathbf{A_{uu}}$ and $\mathbf{A_{vv}}$ are symmetric, and $\mathbf{A_{uv}} = \mathbf{A^T_{vu}}$.) In this case the linear system is solved not exactly, but to within some small tolerance of the true solution.

5.5 Thickness Evolution in Higher-Order Models

5.5.1 Conservation of mass

As mentioned previously, conservation of mass for an ice sheet can be expressed by

$$\frac{\partial H}{\partial t} = -\nabla\cdot(\boldsymbol{U}H)+B, \tag{5.46}$$

where $\boldsymbol{U} = (U, V)$ is the 2D, depth-integrated velocity vector, H is the ice thickness, $\boldsymbol{U}H$ is the 2D ice flux vector, and B is the sum of surface and basal mass balance terms. The change in thickness per unit time is given by the negative of the flux divergence plus a source term. The minus sign in front of the divergence (terms in parentheses on the right-hand side) ensures sensible behavior: consider a section of the ice sheet where the thickness is nearly uniform and there is no accumulation or ablation. If the velocity gradient along the flow is negative (the ice is slowing down), then we expect local thickening (left-hand side of the equation is > 0), and if the ice is speeding up along flow, we expect local thinning. This is the equation that needs to be solved to calculate the evolution of the ice sheet geometry.

5.5.2 A diffusive approach

For the case of a shallow-ice model, U and V are expressed in terms of ice thickness and elevation gradients, in which case the problem can be recast as a diffusion equation in ice thickness. In 1D, (5.46) becomes

$$\frac{\partial H}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial s}{\partial x}\right) + B, \quad D = \frac{2A}{n+2}\left(\rho g\right)^n H^{n+2}\left|\frac{\partial s}{\partial x}\right|^{n-1}, \tag{5.47}$$

where the diffusivity D is nonlinear (because it depends on the solution H), A is the rate factor in Glen's flow law, n is the power-law exponent, s is the ice surface elevation, and ρ and g are the ice density and the acceleration due to gravity. Importantly, we need only local information in order to solve the above equation. If the velocity cannot be solved locally, as in the case of higher-order models, we cannot easily use the above formulation to solve ice sheet evolution. In an attempt to use this form and retain a diffusion-solution approach to the problem (diffusion problems generally have favorable numerical properties), we could try the following approach (again, in 1D):

$$\frac{\partial H}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial s}{\partial x}\right) + B, \quad D = UH\left(\frac{\partial s}{\partial x}\right)^{-1}, \tag{5.48}$$

where the U in the expression for D is the depth-integrated velocity field from the higher-order model. This is the approach initially taken by Pattyn (2003) in one of the first higher-order modeling studies to deal with this particular problem. Notice that ice sheet surface slope is in the denominator of the diffusivity. As slopes become smaller (as they tend to do in fast-flowing ice streams and ice shelves), the diffusivity grows larger, approaching infinity as the slope goes to zero. The faster velocities in these regions appear in the numerator of the diffusivity, also making it larger. This is a severe restriction because, for explicit schemes, the diffusive CFL condition requires that

$$\Delta t < \frac{(\Delta x)^2}{2D}, \tag{5.49}$$

where Δt is the time step required for numerical stability and Δx is the grid spacing. As the diffusivity goes to infinity (i.e., for faster flows and shallower slopes), the stable time step goes to zero. Thus in practice, this approach has proven very difficult to use for calculating ice sheet evolution in the dynamically interesting areas of ice sheets. An alternate approach is needed.

5.5.3 Advection schemes

An alternate approach is to solve the evolution equation using an advection scheme. Numerically, advection schemes can be more problematic than diffusion schemes, but in cases like this one, they are difficult to avoid. One of the simplest advection schemes is a first-order, upwind-advection scheme (as above, "first-order" here refers to first-order accurate), and we will outline the implementation of such a scheme below.

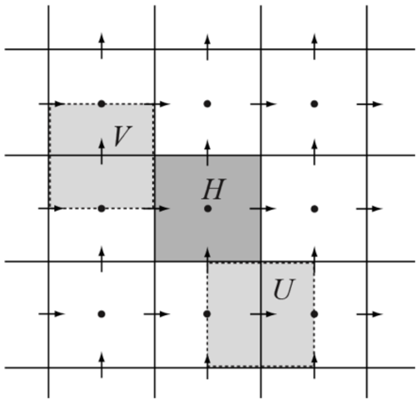


Figure 5.6: Staggered grid in two dimensions, showing scalars (like thickness, H) at grid cell centers and velocity components, U and V, at grid cell edges. This is a "C" grid. Another staggered-grid possibility is a "B" grid, in which both velocity components live at the grid cell corners. While CISM uses a B grid, averaging of corner velocities to edges allows one to express them on a C grid, if necessary.

Most ice sheet models (and fluid dynamics models in general) perform calculations on a "staggered" grid of the type shown below in Figure 5.6, where scalar components (e.g., temperature, pressure, and thickness) live on one grid, and velocity components live on a grid that is staggered by 1/2 grid cell in the horizontal direction. (This is essential for numerical stability for reasons not discussed here).

A control volume (or finite volume) approach to solving the equation

$$\frac{\partial H}{\partial t} = -\left(\frac{\partial (UH)}{\partial x} + \frac{\partial (VH)}{\partial y}\right) + B \tag{5.50}$$

would be to integrate the equation over a control volume centered on the scalar values (Figures 5.6 and 5.7). Ignoring source terms for now (assume $B = 0$), and assuming flow in the x direction only (assume $V = 0$) we have

$$\frac{\partial H}{\partial t} = -\frac{1}{\Delta y \Delta x}\int_{s}^{n}\int_{w}^{e}\frac{\partial (UH)}{\partial x}dxdy = -\frac{1}{\Delta y \Delta x}\left(HU_e - HU_w\right)\Delta y. \tag{5.51}$$

The "east" and "west" (subscripts e and w) faces of the control volume are shown in Figure 5.7.

In the above equation, we have deliberately left it vague as to which value of H is being advected across the east and west interfaces, into or out of the volume. This is where the term "upwinding" comes in – we choose the scalar value to advect across the interface based on an upwinding criterion. If, for example, the flow at interface e is from left to right ($U > 0$), we would advect the value of H at P *out* of the volume centered at P and *into* the volume centered at E. If, on the other hand, the flow at e was from right to left ($U < 0$), we would advect the value of H at E *into* the volume at P and *out* of the volume at E.

The product of velocity, thickness, and the grid spacing at each interface gives a volume flux in units of cubic meters per year. The sum of the volume fluxes over the total number of faces being considered (two in this case, but four for the 2-dimensional case) gives the total volume flux in (total>0) or out (total<0) of the volume. When this number is divided by the area of the volume, the result is the mean thickness change in that volume, per unit time, that is required to maintain conservation of mass.

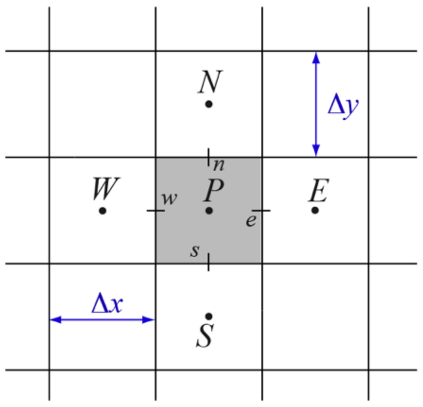


Figure 5.7: Staggered grid in two dimensions, showing locations of interfaces and control volume dimensions. Interfaces e, w, n and s connect the volume centered at P with those volumes to the east, west, north, and south (E, W, N, and S).

5.5.4 Explicit vs. implicit evolution schemes

Is this an explicit or implicit scheme? If we discretize the right-hand side of the primary equation in time we get

$$\begin{aligned}\frac{\partial H}{\partial t} &= -\nabla \cdot (UH) + B, \\ \frac{H_{t=1} - H_{t=0}}{\Delta t} &\approx -\nabla \cdot (UH) + B.\end{aligned} \tag{5.52}$$

This can be rearranged to

$$H_{t_{t=1}} \approx H_{t=0} + [-\nabla \cdot (UH) + B]_{t=0} \, \Delta t. \tag{5.53}$$

The thickness at the new time step $t = 1$ is a function only of variables evaluated at the previous time step $t = 0$. Thus, the scheme is *explicit* and, as such, is subject to the advective CFL condition for stability:

$$\Delta t < \frac{\Delta x}{U}. \tag{5.54}$$

Essentially, this means that we cannot take such a large time step that we advect mass through more than one grid cell per time step.

There is not much more to a first-order advection scheme other than extending it to two dimensions for non-zero V. The finite volume formulation guarantees that it will conserve mass (i.e., the mass moving out of one grid cell across a given cell face is equal to the mass moving into an adjacent grid cell across that face). There are many “higher-order” advection schemes (e.g., the incremental remapping scheme used in CISM and described in Chapter 6), but they are mainly based on the principles outlined here and rely on corrections to the simple upwind assumption in order to gain more accuracy.

Chapter 6

Higher-Order Ice Dynamics: Glissade Dynamical Core

6.1 Introduction

CISM includes a parallel, higher-order dynamical core called Glissade[1], the successor to the Glide shallow-ice dycore. Like Glide, Glissade solves equations for conservation of momentum (i.e., an appropriate approximation of Stokes flow), mass, and internal energy. Glissade numerics, however, differ substantially from Glide numerics:

- *Velocity:* Glide is limited to the shallow-ice approximation, whereas Glissade can solve several Stokes approximations, as described in Section 6.2. These include a 3D first-order Blatter-Pattyn solver (the most accurate and complex scheme) as well as simpler shallow-ice, shallow-shelf, and "L1L2" solvers. Unlike Glide, which computes the velocity using finite-difference techniques, Glissade uses a more robust and flexible finite-element method.

- *Temperature:* To evolve the ice temperature, Glide solves a prognostic equation that incorporates horizontal advection as well as vertical heat diffusion and internal dissipation. In Glissade, temperature advection is handled by the transport scheme, and a separate module solves for vertical diffusion and internal dissipation. The vertical solver, described in Section 6.3, is local; each ice column calculation is independent of other column calculations.

- *Mass and tracer transport:* Glide solves a diffusion equation for tranport of mass (i.e., thickness); this equation incorporates the local shallow-ice velocities. Since Glissade solves for higher-order flow that may have a large advective (as oppposed to diffusive) component, a different approach is needed. Glissade has two mass- and energy-conserving transport options, described in Section 6.4: incremental remapping (the more complex and accurate scheme) and first-order upwind. These schemes transport mass and internal energy in the horizontal direction, followed by a vertical remapping to sigma coordinates.

Glissade numerics are described in detail below.

[1]The name "Glissade" was originally an acronym, but as with "Glimmer" and "Glide" the acronym is rarely used. It can be pronounced either American–style (gliss–AID) or French–style (gliss–AHD).

6.2 Velocity Solver

Glissade computes the ice velocity by solving an appropriate approximation of the Stokes equations, given the 2D surface elevation and thickness fields, the 3D temperature field, and relevant boundary conditions. The directory `libglissade` contains the primary higher-order velocity module, `glissade_velo_higher.F90`, along with supporting modules for the various solver options. Section 6.2.1 describes assembly and solution of the matrix problem for the 3D first-order Blatter-Pattyn approximation. The following sections discuss simpler approximations, including the shallow-ice, shallow-shelf, and L1L2 approximations.

6.2.1 Blatter-Pattyn approximation

The basic equations of the Blatter-Pattyn approximation in Cartesian coordinates, repeated from Section 5.2, are

$$\begin{aligned} x: \quad & \frac{\partial}{\partial x}\left(2\eta\left(2\frac{\partial u}{\partial x}+\eta\frac{\partial v}{\partial y}\right)\right)+\frac{\partial}{\partial y}\left(\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{\partial}{\partial z}\left(\eta\frac{\partial u}{\partial z}\right)=\rho g\frac{\partial s}{\partial x}, \\ y: \quad & \frac{\partial}{\partial y}\left(2\eta\left(2\frac{\partial v}{\partial y}+\eta\frac{\partial u}{\partial x}\right)\right)+\frac{\partial}{\partial x}\left(\eta\left(\frac{\partial u}{\partial y}+\frac{\partial v}{\partial x}\right)\right)+\frac{\partial}{\partial z}\left(\eta\frac{\partial v}{\partial z}\right)=\rho g\frac{\partial s}{\partial y}, \end{aligned} \tag{6.1}$$

where u and v are the components of horizontal velocity, η is the effective viscosity, s is the ice surface elevation, ρ is the density of ice (assumed constant), and g is gravitational acceleration.

As in Glide, the equations are discretized on a structured 3D mesh. In the map plane the mesh consists of rectangular *cells.* These cells form an unstaggered 2D grid of dimension (nx, ny) (thus the number of cells is $(nx)(ny)$), together with a staggered grid of dimension $(nx-1, ny-1)$. The corners of each cell (where four rectangles meet) are called *vertices.* The vertical *levels* of the mesh are based on a terrain-following sigma coordinate system. We define $\sigma = (s-z)/H$, where H is the ice thickness, with $\sigma = 0$ at the top surface and $\sigma = 1$ at the bed. There are nz levels in the vertical direction, with $nz-1$ *layers* between these levels. An *element* is the region associated with a particular cell and layer; there are $(nx)(ny)(nz-1)$ elements on the mesh. A *node* is a point where eight elements intersect (or where four elements intersect at the upper or lower surface). There are $(nx-1)(ny-1)(nz)$ nodes on the mesh.

Scalar 2D fields such as H and s are defined for each cell. Scalar 3D fields such as ice temperature T lie at the center of each element (i.e., at the midpoint of each layer associated with each cell). Gradients of 2D scalar fields (e.g., the surface slope ∇s) are defined at vertices. The velocity components u and v live at nodes.

For problems solved in parallel, the domain is partitioned among multiple processors. Let nx_{local} and ny_{local} be the number of locally owned cells on each processor. Each processor holds data for two rows of halo cells (i.e., cells belonging to other processors) surrounding the block of locally owned cells. Thus the 2D grid on a given processor has dimensions $(nx_{\text{local}}+2, ny_{\text{local}}+2)$. Each locally owned cell is associated with a locally owned vertex lying at the northeast (upper right) corner of the cell.

An *active cell* is a cell that borders a locally owned vertex, whose ice thickness exceeds a minimum threshold. Each active cell is associated with a column of $nz-1$ *active elements.* (All the data in a given column resides on a single processor.) An *active vertex* is any vertex of an active cell. Each active vertex is associated with nz active nodes, including nodes at the surface and bed.

The effective viscosity is defined in each active element by

$$\eta \equiv \frac{1}{2}A^{\frac{-1}{n}}\dot{\varepsilon}_e^{\frac{1-n}{n}}, \tag{6.2}$$

where A is the temperature-dependent rate factor in Glen's flow law, and $\dot{\varepsilon}_e$ is the effective strain rate, given in the Blatter-Pattyn approximation by

$$\dot{\varepsilon}_e^2 = \dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2 + \dot{\varepsilon}_{xz}^2 + \dot{\varepsilon}_{yz}^2, \tag{6.3}$$

where

$$\dot{\varepsilon}_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). \tag{6.4}$$

To compute $A(T)$, we assume an Arrhenius relationship:

$$A(T_{\text{pmp}}) = ae^{-Q/RT_{\text{pmp}}}, \tag{6.5}$$

where T_{pmp} is the pressure melting point temperature, a is a temperature–independent material constant (given by Paterson and Budd (1982)), Q is the activation energy for creep and R is the universal gas constant.

Given T, s, H, and an initial guess for u and v, the problem is to solve Eq. (6.1) for u and v at each active node. (At inactive nodes we set $u = v = 0$.) This problem can be written in the form

$$\mathbf{Ax} = \mathbf{b}, \tag{6.6}$$

or more fully,

$$\begin{bmatrix} \mathbf{A_{uu}} & \mathbf{A_{uv}} \\ \mathbf{A_{vu}} & \mathbf{A_{vv}} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{v} \end{bmatrix} = \begin{bmatrix} \mathbf{b_u} \\ \mathbf{b_v} \end{bmatrix}. \tag{6.7}$$

Eq. (6.7) shows the four parts of the global matrix $\mathbf{A}$, with the solution and right-hand-side vectors separated into u and v components. In Glissade, $\mathbf{A}$ is always symmetric and positive-definite.

Since $\mathbf{A}$ depends (through η) on u and v, the problem is nonlinear and must be solved iteratively (see Section 5.4.2). For each nonlinear iteration, Glissade computes the 3D η field based on the current guess for the velocity field and solves a linear problem of the form (6.7). Then η is updated and the process is repeated until the solution converges to within a given tolerance. This procedure is known as Picard iteration.

The following sections describe how the matrix equations are assembled and solved.

Assembly

The coupled PDEs (6.1) are discretized using the finite-element method. Here we give a detailed but non-rigorous description of the method as applied to the Blatter-Pattyn approximation on the CISM mesh. We refer the reader to standard texts (e.g., Hughes, 2000) for a full discussion of finite elements.

The PDEs, with appropriate boundary conditions, are converted to a system of algebraic equations by dividing the full domain into subdomains (i.e., elements), representing the velocity solution on each element, and integrating over elements. On the CISM mesh, the elements are hexahedra (rectangles in map view), each of which has eight nodes shared with its neighbors. The solution is approximated as a sum over basis functions φ. Each active node is associated with a basis function whose value is $\varphi = 1$ at that node, with $\varphi = 0$ at all other nodes. The solution at a point within an element can be expanded in terms of basis functions and nodal values:

$$u(x,y,z) = \sum_n \varphi_n(x,y,z)u_n, \quad v(x,y,z) = \sum_n \varphi_n(x,y,z)v_n, \tag{6.8}$$

where the sum is over the nodes of the element, u_n and v_n are nodal values of the solution, and φ_n varies smoothly between 0 and 1 within the element. The fact that $\varphi = 0$ except in a small region ensures that the sum includes only as many terms as there are nodes per element.

Glissade uses standard finite-element techniques to represent the PDE on each element and assemble the element equations into a global set of algebraic equations of the form (6.7). The scheme is formally equivalent to that described by Perego et al. (2012) (henceforth PGB). Eq. (6.1) can be written as

$$\begin{aligned} -\nabla \cdot (2\eta\dot{\varepsilon}_1) &= -\rho g \frac{\partial s}{\partial x}, \\ -\nabla \cdot (2\eta\dot{\varepsilon}_2) &= -\rho g \frac{\partial s}{\partial y}, \end{aligned} \tag{6.9}$$

where

$$\dot{\varepsilon}_1 = \begin{bmatrix} 2\dot{\varepsilon}_{xx} + \dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{xy} \\ \dot{\varepsilon}_{xz} \end{bmatrix}, \quad \dot{\varepsilon}_2 = \begin{bmatrix} \dot{\varepsilon}_{xy} \\ \dot{\varepsilon}_{xx} + 2\dot{\varepsilon}_{yy} \\ \dot{\varepsilon}_{yz} \end{bmatrix}. \tag{6.10}$$

(These are Eqs. 12 and 13 in PGB.) We rewrite the equations in *weak form* (see PGB Eq. 15), which is obtained by multiplying (6.9) by the basis functions and integrating over the domain, using integration by parts to eliminate the second derivative:

$$\begin{aligned} x: &\int_\Omega 2\eta\dot{\varepsilon}_1(u,v)\cdot\nabla\varphi_1 \, \mathrm{d}\Omega + \int_{\Gamma_B} \beta u \varphi_1 \, \mathrm{d}\Gamma + \int_{\Gamma_L} p n_1 \varphi_1 \, \mathrm{d}\Gamma + \int_\Omega \rho g \frac{\partial s}{\partial x}\varphi_1 \, \mathrm{d}\Omega = 0, \\ y: &\int_\Omega 2\eta\dot{\varepsilon}_2(u,v)\cdot\nabla\varphi_2 \, \mathrm{d}\Omega + \int_{\Gamma_B} \beta u \varphi_2 \, \mathrm{d}\Gamma + \int_{\Gamma_L} p n_2 \varphi_2 \, \mathrm{d}\Gamma + \int_\Omega \rho g \frac{\partial s}{\partial y}\varphi_2 \, \mathrm{d}\Omega = 0, \end{aligned} \tag{6.11}$$

where Ω represents the domain volume, Γ_B denotes the lower boundary, Γ_L denotes the lateral boundary (e.g., the calving front of an ice shelf), β is a basal traction parameter, p is the pressure at the lateral boundary, and n_1 and n_2 are components of the normal to Γ_L. These equations can also be obtained from a variational principle as described by Dukowicz et al. (2010).

The four terms on the LHS of (6.11) describe internal ice streses, basal friction, lateral pressure, and the gravitational driving force, respectively. Next we describe how these terms are summed over elements and assembled into the global matrix $\mathbf{A}$ and right-hand side vector $\mathbf{b}$.

Internal ice stresses. We start with the internal stress term, which is the most complex. We rewrite the first term on the LHS of (6.11) in terms of velocity components:

$$\begin{aligned} x: &\int_\Omega 2\eta \left[2\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} \quad \frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \quad \frac{1}{2}\frac{\partial u}{\partial z} \right] \begin{Bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \\ \frac{\partial \varphi}{\partial z} \end{Bmatrix}, \\ y: &\int_\Omega 2\eta \left[\frac{1}{2}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \quad 2\frac{\partial v}{\partial y} + \frac{\partial u}{\partial x} \quad \frac{1}{2}\frac{\partial v}{\partial z} \right] \begin{Bmatrix} \frac{\partial \varphi}{\partial x} \\ \frac{\partial \varphi}{\partial y} \\ \frac{\partial \varphi}{\partial z} \end{Bmatrix}, \end{aligned} \tag{6.12}$$

where brackets are used for row vectors and braces for column vectors. Glissade evaluates (6.12) for each active element. Recall that hexahedral elements have eight nodes, with u and v to be determined at each active node. Inserting the velocity expressions (6.8) in (6.12), we obtain

$$\begin{aligned}
x &: \int\limits_{\Omega} 2\eta \left(\frac{\partial \varphi_i}{\partial x} \left(2 \left[\frac{\partial \varphi_j}{\partial x} \right] \{u_j\} + \left[\frac{\partial \varphi_j}{\partial y} \right] \{v_j\} \right) + \frac{\partial \varphi_i}{\partial y} \left(\frac{1}{2} \left[\frac{\partial \varphi_j}{\partial y} \right] \{u_j\} + \frac{1}{2} \left[\frac{\partial \varphi_j}{\partial x} \right] \{v_j\} \right) + \frac{\partial \varphi_i}{\partial z} \left(\frac{1}{2} \left[\frac{\partial \varphi_j}{\partial z} \right] \{u_j\} \right) \right) \\
y &: \int\limits_{\Omega} 2\eta \left(\frac{\partial \varphi_i}{\partial y} \left(2 \left[\frac{\partial \varphi_j}{\partial y} \right] \{v_j\} + \left[\frac{\partial \varphi_j}{\partial x} \right] \{u_j\} \right) + \frac{\partial \varphi_i}{\partial x} \left(\frac{1}{2} \left[\frac{\partial \varphi_j}{\partial x} \right] \{v_j\} + \frac{1}{2} \left[\frac{\partial \varphi_j}{\partial y} \right] \{u_j\} \right) + \frac{\partial \varphi_i}{\partial z} \left(\frac{1}{2} \left[\frac{\partial \varphi_j}{\partial z} \right] \{v_j\} \right) \right)
\end{aligned} \tag{6.13}$$

Each row or column vector has eight terms, one for each node of the element. These terms can be evaluated to form a set of four 8x8 element matrices. Each row of an element matrix is associated with u or v at a given node. The columns in that row contain terms linking that node to u or v at the other nodes of the element (with the diagonal term linking the node to itself).

In the x component of (6.13), the terms that multiply u_j are given by

$$\int\limits_{\Omega} \eta \left(4 \frac{\partial \varphi_i}{\partial x} \frac{\partial \varphi_j}{\partial x} + \frac{\partial \varphi_i}{\partial y} \frac{\partial \varphi_j}{\partial y} + \frac{\partial \varphi_i}{\partial z} \frac{\partial \varphi_j}{\partial z} \right) d\Omega \tag{6.14}$$

Letting i and j range from 1 to 8, (6.14) gives the 64 terms of the 8x8 element matrix $\mathbf{K_{uu}}$, which links the u value at each node to the u values at all eight nodes. Similarly, the 64 terms of element matrix $\mathbf{K_{uv}}$, which links u at each node to v at each of the eight nodes, are given by

$$\int\limits_{\Omega} \eta \left(2 \frac{\partial \varphi_i}{\partial x} \frac{\partial \varphi_j}{\partial y} + \frac{\partial \varphi_i}{\partial y} \frac{\partial \varphi_j}{\partial x} \right) d\Omega \tag{6.15}$$

Likewise, two 8x8 matrices are associated with the y component of (6.13). The terms of $\mathbf{K_{vu}}$, which connects v at each node to u at each of eight nodes, are given by

$$\int\limits_{\Omega} \eta \left(2 \frac{\partial \varphi_i}{\partial y} \frac{\partial \varphi_j}{\partial x} + \frac{\partial \varphi_i}{\partial x} \frac{\partial \varphi_j}{\partial y} \right) d\Omega \tag{6.16}$$

Finally, the terms of $\mathbf{K_{vv}}$, which links v at each node to v at each of eight nodes, are given by

$$\int\limits_{\Omega} \eta \left(4 \frac{\partial \varphi_i}{\partial y} \frac{\partial \varphi_j}{\partial y} + \frac{\partial \varphi_i}{\partial x} \frac{\partial \varphi_j}{\partial x} + \frac{\partial \varphi_i}{\partial z} \frac{\partial \varphi_j}{\partial z} \right) d\Omega \tag{6.17}$$

Because of the symmetry of the underlying PDEs, $\mathbf{K_{uu}}$ and $\mathbf{K_{vv}}$ are symmetric, and $\mathbf{K_{uv}} = \mathbf{K_{vu}}^T$. Note that $\mathbf{K_{vv}}$ can be obtained from $\mathbf{K_{uu}}$, and $\mathbf{K_{vu}}$ from $\mathbf{K_{uv}}$, by exchanging x and y. The terms containing z (i.e., the vertical shear stresses associated with the shallow-ice approximation) appear only in $\mathbf{K_{uu}}$ and $\mathbf{K_{vv}}$. The terms containing x and y (i.e., the membrane stresses) appear in all four element matrices.

Eqs. (6.14)–(6.17) lie at the heart of the code. Together with the expressions for the effective viscosity η (discussed below), these expressions contain the physical contents of the Blatter-Pattyn approximation.

In the weak form of the equations, each of the 64 coefficients in each element matrix must be integrated over the element. (Since φ varies over the element, the integrands in (6.14)–(6.17) have a different value at each point.) This is done for a given element by evaluating the integrand at each of eight *quadrature points* and summing over quadrature points. We first have to specify the form of the basis functions, then transform the basis functions to the geometry of the element (which is irregular in the vertical because of the sigma coordinate) and evaluate the basis function derivatives at the quadrature points.

Glissade uses trilinear basis functions defined on a reference cube. This cube is centered at the origin $(0, 0, 0)$ in local reference coordinates $(\hat{x}, \hat{y}, \hat{z})$. The eight nodes of the reference cube are located at $(\hat{x}, \hat{y}, \hat{z}) = (\pm 1, \pm 1, \pm 1)$. By convention, nodes 1–4 are the nodes of the lower face, proceeding counterclockwise from the southwest corner $(\hat{x}, \hat{y}) = (-1, -1)$. Nodes 5–8 are the nodes of the upper face, also moving counterclockwise from the southwest corner. Thus we have

$$\begin{aligned}
\varphi_1 &= (1-\hat{x})(1-\hat{y})(1-\hat{z})/8, \\
\varphi_2 &= (1+\hat{x})(1-\hat{y})(1-\hat{z})/8, \\
\varphi_3 &= (1+\hat{x})(1+\hat{y})(1-\hat{z})/8, \\
\varphi_4 &= (1-\hat{x})(1+\hat{y})(1-\hat{z})/8, \\
\varphi_5 &= (1-\hat{x})(1-\hat{y})(1+\hat{z})/8, \\
\varphi_6 &= (1+\hat{x})(1-\hat{y})(1+\hat{z})/8, \\
\varphi_7 &= (1+\hat{x})(1+\hat{y})(1+\hat{z})/8, \\
\varphi_8 &= (1-\hat{x})(1+\hat{y})(1+\hat{z})/8.
\end{aligned} \tag{6.18}$$

For each n we have $\varphi_n = 1$ at a single node, with $\varphi_n = 0$ at the other nodes.

The integrands in (6.14)–(6.17) are written in terms of real Cartesian coordinates (x, y, z) rather than reference coordinates $(\hat{x}, \hat{y}, \hat{z})$. Spatial derivatives in real coordinates are related to derivatives in reference coordinates by

$$\left\{ \begin{matrix} \frac{\partial \varphi_n}{\partial \hat{x}} \\ \frac{\partial \varphi_n}{\partial \hat{y}} \\ \frac{\partial \varphi_n}{\partial \hat{z}} \end{matrix} \right\} = \begin{bmatrix} \frac{\partial x}{\partial \hat{x}} & \frac{\partial y}{\partial \hat{x}} & \frac{\partial z}{\partial \hat{x}} \\ \frac{\partial x}{\partial \hat{y}} & \frac{\partial y}{\partial \hat{y}} & \frac{\partial z}{\partial \hat{y}} \\ \frac{\partial x}{\partial \hat{z}} & \frac{\partial y}{\partial \hat{z}} & \frac{\partial z}{\partial \hat{z}} \end{bmatrix} \left\{ \begin{matrix} \frac{\partial \varphi_n}{\partial x} \\ \frac{\partial \varphi_n}{\partial y} \\ \frac{\partial \varphi_n}{\partial z} \end{matrix} \right\} = [J] \left\{ \begin{matrix} \frac{\partial \varphi_n}{\partial x} \\ \frac{\partial \varphi_n}{\partial y} \\ \frac{\partial \varphi_n}{\partial z} \end{matrix} \right\}, \tag{6.19}$$

where $[J]$ is the Jacobian of the transformation between coordinate systems. Given the finite-element expansion

$$x = \sum_n \varphi_n x_n, \tag{6.20}$$

along with the spatial derivatives of φ at $(\hat{x}, \hat{y}, \hat{z})$ (which are easily derived from (6.18)), we can compute $[J(\hat{x}, \hat{y}, \hat{z})]$ as

$$[J] = \begin{bmatrix} \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{x}} x_n & \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{x}} y_n & \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{x}} z_n \\ \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{y}} x_n & \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{y}} y_n & \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{y}} z_n \\ \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{z}} x_n & \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{z}} y_n & \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial \hat{z}} z_n \end{bmatrix}. \tag{6.21}$$

We then invert (6.19) to obtain the basis function derivatives in terms of (x, y, z):

$$\left\{ \begin{matrix} \frac{\partial \varphi_n}{\partial x} \\ \frac{\partial \varphi_n}{\partial y} \\ \frac{\partial \varphi_n}{\partial z} \end{matrix} \right\} = [J^{-1}] \left\{ \begin{matrix} \frac{\partial \varphi_n}{\partial \hat{x}} \\ \frac{\partial \varphi_n}{\partial \hat{y}} \\ \frac{\partial \varphi_n}{\partial \hat{z}} \end{matrix} \right\}. \tag{6.22}$$

The left-hand side of (6.22) contains the spatial derivatives appearing in (6.14)–(6.17).

Eqs. (6.14)–(6.17) also contain the viscosity η, which is computed at each quadrature point. In the Blatter-Pattyn approximation, η is given by (6.2); it is a function of the flow factor A and the effective strain rate defined by (6.3). We approximate A by its value at the element center.

The (squared) effective strain rate, $\dot{\varepsilon}_e^2$, is evaluated at each quadrature point by summing over strain-rate components. The x components are given by

$$\frac{\partial u}{\partial x} = \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial x} u_n, \quad \frac{\partial v}{\partial x} = \sum_{n=1}^{8} \frac{\partial \varphi_n}{\partial x} v_n, \tag{6.23}$$

and similarly for the y and z components. The nodal velocities in (6.23) are values from the previous iteration; otherwise the resulting system of equations would be nonlinear.

We now have the information needed to compute the integrands (6.14)–(6.17) at quadrature points. To integrate over a hexahedron, we take a weighted sum of the values at each of eight quadrature points. These points are located at reference coordinates $(\hat{x}, \hat{y}, \hat{z}) = (\pm 1/\sqrt{3}, \pm 1/\sqrt{3}, \pm 1\sqrt{3})$. To evaluate an integral of the form

$$\int_\Omega \eta \left(\frac{\partial \varphi_i}{\partial z} \frac{\partial \varphi_j}{\partial z} \right) d\Omega \tag{6.24}$$

over element volume Ω, we compute the sum over quadrature points

$$\sum_{p=1}^{8} w_p \eta_p \left(\frac{\partial \varphi_i}{\partial z} \frac{\partial \varphi_j}{\partial z} \right)_p |J_p|, \tag{6.25}$$

where $|J|$ is the determinant of the Jacobian (6.21). For this choice of quadrature points, each point has $w_p = 1$.

The terms of the element matrices $\mathbf{K_{uu}}, \mathbf{K_{uv}}, \mathbf{K_{vu}}$ and $\mathbf{K_{vv}}$ are then inserted into the corresponding global matrices $\mathbf{A_{uu}}, \mathbf{A_{uv}}, \mathbf{A_{vu}}$ and $\mathbf{A_{vv}}$. This is mostly a matter of bookkeeping. For example, the first row of $\mathbf{K_{uu}}$ corresponds to a particular node of element (k, i, j) (specifically, the node with indices $(k-1, i-1, j-1)$, given our convention for numbering nodes within elements). This row is mapped to a row of the global matrix $\mathbf{A_{uu}}$, and each of the eight terms in the row is associated with a column of $\mathbf{A_{uu}}$. Glissade determines the correct column and adds the $\mathbf{K_{uu}}$ term to the corresponding term in $\mathbf{A_{uu}}$. This process proceeds until the code has looped over all the active elements and filled the global matrices.

If written in full, each global matrix would have as many rows and columns as there are active nodes. These matrices, however, are sparse, with a maximum of 27 nonzero terms per row (corresponding to a node and its 26 nearest neighbors in a 3D hexahedral lattice). Glissade therefore assembles and stores global matrices of dimension $(27, nz, nx-1, ny-1)$. The 27 terms of the first dimension are indexed such that each index has a geometric meaning. Suppose we are filling columns for the matrix row corresponding to node (k, i, j). Then, by convention, index 1 refers to the node with coordinates $(k-1, i-1, j-1)$, index 14 refers to the node itself (i.e., the diagonal term of the row), and index 27 refers to the node at $(k+1, i+1, j+1)$ (and similarly for the other indices). After assembly, these matrices can be converted as needed to the form required by a particular solver.

The remaining assembly consists of evaluating the other terms in (6.11) (i.e., the basal and lateral boundary conditions and the gravitational forcing) and implementing Dirichlet boundary conditions, if applicable. We consider these in turn.

Basal boundary conditions. At the basal boundary we assume a friction law of the form

$$\tau_\mathbf{b} = -\beta \mathbf{u_b}. \tag{6.26}$$

The coefficient β is defined at each vertex and can vary spatially. If β depends on the velocity, as in some friction laws, then it is calculated using the velocity from the previous iteration. See Sections 5.3.2 and 5.3.3 for a detailed discussion of basal traction in higher-order models.

The basal boundary terms to be evaluated in (6.11) are

$$\begin{aligned} x &: \int\limits_{\Gamma_B} \beta u \varphi_1 d\Gamma, \\ y &: \int\limits_{\Gamma_B} \beta v \varphi_2 d\Gamma. \end{aligned} \tag{6.27}$$

The basal face of each cell is a rectangle. To integrate over a rectangle, we sum over terms at four quadrature points. These points lie at $(\hat{x}, \hat{y}) = (\pm 1/\sqrt{3}, \pm 1/\sqrt{3})$ in a reference square with center $(0,0)$ and vertices $(\pm 1, \pm 1)$. (This reference square is the 2D analog of the reference cube discussed above.) We define four bilinear basis functions on the square (cf. (6.18)):

$$\begin{aligned} \varphi_1 &= (1-\hat{x})(1-\hat{y})/4, \\ \varphi_2 &= (1+\hat{x})(1-\hat{y})/4, \\ \varphi_3 &= (1+\hat{x})(1+\hat{y})/4, \\ \varphi_4 &= (1-\hat{x})(1+\hat{y})/4. \end{aligned} \tag{6.28}$$

Given these basis functions and their spatial derivatives, we can compute the Jacobian for the transformation between the reference square and the rectangular cell face, using the 2D versions of (6.21) and (6.22):

$$[J] = \begin{bmatrix} \sum\limits_{n=1}^{4} \frac{\partial \varphi_n}{\partial \hat{x}} x_n & \sum\limits_{n=1}^{4} \frac{\partial \varphi_n}{\partial \hat{x}} y_n \\ \sum\limits_{n=1}^{4} \frac{\partial \varphi_n}{\partial \hat{y}} x_n & \sum\limits_{n=1}^{4} \frac{\partial \varphi_n}{\partial \hat{y}} y_n \end{bmatrix}, \tag{6.29}$$

$$\begin{Bmatrix} \frac{\partial \varphi_n}{\partial x} \\ \frac{\partial \varphi_n}{\partial y} \end{Bmatrix} = [J^{-1}] \begin{Bmatrix} \frac{\partial \varphi_n}{\partial \hat{x}} \\ \frac{\partial \varphi_n}{\partial \hat{y}} \end{Bmatrix}. \tag{6.30}$$

The integrand at a quadrature point has the form

$$\beta \varphi_i \varphi_j, \tag{6.31}$$

where the second φ term arises from the finite-element expansion of u at a quadrature point:

$$u = \sum_{n=1}^{4} u_n \varphi_n. \tag{6.32}$$

We determine β at quadrature points from the values at cell vertices:

$$\beta = \sum_{n=1}^{4} \beta_n \varphi_n. \tag{6.33}$$

The integral of (6.31) over a cell is then computed as a sum over quadrature points:

$$\sum_{p=1}^{4} w_p \beta_p (\varphi_i \varphi_j)_p |J_p|, \tag{6.34}$$

where $w_p = 1$ for each point. This procedure generates a 4x4 matrix describing the connections between each node and its neighbors in the cell. Since the x term in (6.27) contains u but not v, and the y term contains v but not u, we form 2D matrices $\mathbf{K_{uu}}$ and $\mathbf{K_{vv}}$, but not $\mathbf{K_{uv}}$ and

$\mathbf{K_{vu}}$. Each term of $\mathbf{K_{uu}}$ is then inserted into the global matrix $\mathbf{A_{uu}}$, and similarly for $\mathbf{K_{vv}}$ and $\mathbf{A_{vv}}$.

This assembly procedure tends to smooth the β field. If it is necessary to resolve sharp discontinuities in β, as in the stream test problem of Section 8.2.3, Glissade allows an alternate assembly procedure in which the terms in a given row of the matrix depend only on β at the vertex associated with that particular row of the matrix. See option `which_ho_assemble_beta` in Section 7.3.

Lateral boundary conditions. The lateral boundary terms in (6.11) are

$$\begin{aligned} x &: \int_{\Gamma_L} p n_1 \varphi_1 \, \mathrm{d}\Gamma, \\ y &: \int_{\Gamma_L} p n_2 \varphi_2 \, \mathrm{d}\Gamma. \end{aligned} \tag{6.35}$$

Since these terms are independent of u and v, they contribute to the load vectors $\mathbf{b_u}$ and $\mathbf{b_v}$ on the right-hand side of (6.7). They are integrated over the lateral faces of floating cells that border the ocean. (Grounded cells are assumed to have no lateral forcing.)

The lateral faces bordering the ocean are quadrilaterals that can be mapped to a reference square. The integral over each face is found by summing over four quadrature points. Basis functions are given by (6.28), and the Jacobian of the reference square is computed using (6.29). We evaluate the ice thickness H at each quadrature point using

$$H = \sum_{n=1}^{4} H_n \varphi_n, \tag{6.36}$$

where the H_n are nodal values. The integrands have the form $p_n \varphi$, where p_n is the vertically averaged net pressure normal to the ice edge. (We use the vertically averaged pressure to avoid dealing with vertical shear at the ice edge.) The net pressure is equal to the pressure directed outward from the ice toward the ocean by the weight of the ice, minus the (smaller) pressure directed inward from the ocean to the ice by the hydrostatic water pressure. The outward pressure is obtained by integrating $\rho_i g(s-z)dz$ from $s-H$ to s and then dividing by H; it is given by

$$p_{\text{out}} = \frac{\rho_i g H}{2}. \tag{6.37}$$

The inward pressure is found by integrating $(-\rho_w g z dz)$ from $s-H$ to 0 and then dividing by $H-s$; it is given by

$$p_{\text{in}} = \frac{\rho_w g (s-H)^2}{2H} \tag{6.38}$$

Assuming hydrostatic balance, we have $s - H = (\rho_i/\rho_w)H$. Thus (6.38) becomes

$$p_{\text{in}} = \frac{\rho_i g H}{2} \frac{\rho_i}{\rho_w} \tag{6.39}$$

Combining (6.37) and (6.39) gives

$$p_{\text{net}} = \frac{\rho_i g H}{2} \left(1 - \frac{\rho_i}{\rho_w}\right), \tag{6.40}$$

directed from the ice to the ocean. The integral of the pressure terms over a lateral face is then found as a sum over quadrature points:

$$\sum_{p=1}^{4} \pm w_p (p_{\text{net}})_p (\varphi_i)_p |J_p|, \tag{6.41}$$

where the sign depends on the orientation of the face. The resulting pressure terms are inserted into the load vector (either $\mathbf{b_u}$ or $\mathbf{b_v}$, depending on the orientation) in the rows associated with each of the four nodes of the face.

Gravitational driving stress. The gravitational forcing terms in (6.11) are

$$\begin{aligned} x &: \int_\Omega \rho g \frac{\partial s}{\partial x} \varphi_1 \, \mathrm{d}\Omega, \\ y &: \int_\Omega \rho g \frac{\partial s}{\partial y} \varphi_2 \, \mathrm{d}\Omega. \end{aligned} \tag{6.42}$$

To compute these terms we evaluate $\frac{\partial s}{\partial x}$ and $\frac{\partial s}{\partial y}$ at each active vertex. A standard centered approximation at vertex (i,j) is

$$\frac{\partial s}{\partial x}(i,j) = \frac{s(i+1,j+1) + s(i+1,j) - s(i,j+1) - s(i,j)}{2\Delta x}, \tag{6.43}$$

and similarly for $\frac{\partial s}{\partial y}$. This approximation works well when the ice geometry is fixed but can cause problems when the geometry is evolving. These problems arise because checkerboard noise in s (which is common on structured meshes with the velocity at cell vertices) is invisible to the momentum balance; it is canceled out by the centered averaging in (6.43). Checkerboard noise can therefore persist and grow. To damp this noise, Glissade can use an upstream average:

$$\frac{\partial s}{\partial x}(i,j) = \frac{1.5(s(i+1,j+1) - s(i,j+1)) - 0.5(s(i+1,j+2) - s(i,j+2))}{\Delta x}. \tag{6.44}$$

Here, "upstream" means in the direction of increasing surface elevation. Both (6.43) and (6.44) are second-order accurate. The default is (6.43), but (6.44) can be chosen by setting `which_ho_gradient = 1` in the config file (see Section 7.3).

Eqs. (6.43) and (6.44) are ambiguous at the ice margin, where one or more of the four cells neighboring a vertex are ice-free. (Cells with very thin ice, $H <$ `thklim`, are considered ice-free by the velocity solver. CISM's default value is `thklim` $= 100$ m, which is appropriate for Glide, but Glissade is typically run with `thklim` $= 1$ m.) One option is to include all cells, including ice-free cells, in the gradient. This generally works well for land-based ice but gives large gradients with excessive ice speeds at floating shelf margins. A second option is to include only ice-covered cells in the gradient. For example, suppose cells (i,j) and $(i,j+1)$ have ice, but cells $(1+1,j)$ and $(i+1,j+1)$ are ice-free. Then, lacking the required information to compute a gradient in the x direction, we would set $\frac{\partial s}{\partial x} = 0$. The y gradient would be one-sided: $\frac{\partial s}{\partial y} = (s(i,j+1) - s(i,j))/\Delta y$. This option works well at shelf margins but tends to underestimate slopes at land margins. A third option is to include in the gradient any neighbor cells that are either ice-covered or land cells. (Land cells are cells with bedrock topography above sea level, whether ice-covered or ice-free.) Since this option (`which_ho_gradient_margin = 1`) works well for both land and shelf margins, it is the default.

The integrals in (6.42) are over 3D elements. Hence we map each hexahedral element to a reference cube as described above. Given $\frac{\partial s}{\partial x}$ at the nodes of a cell, the surface slope terms at quadrature points are

$$\frac{\partial s}{\partial x} = \sum_{n=1}^{8} \varphi_n \left(\frac{\partial s}{\partial x}\right)_n, \quad \frac{\partial s}{\partial y} = \sum_{n=1}^{8} \varphi_n \left(\frac{\partial s}{\partial y}\right)_n, \tag{6.45}$$

where the basis functions φ are given by (6.18) and the spatial derivatives are derived from (6.21) and (6.22). The integral of $\rho g \frac{\partial s}{\partial x}\varphi$ over an element is evaluated as a sum over quadrature points:

$$\sum_{p=1}^{8} w_p \rho g \left(\frac{\partial s}{\partial x}\right)_p (\varphi_i)_p |J_p|, \tag{6.46}$$

and similarly for the $\frac{\partial s}{\partial y}$ term. Glissade inserts these terms into the load vectors $\mathbf{b_u}$ and $\mathbf{b_v}$.

Dirichlet boundary conditions. Once the matrix has been assembled, it may need to be adjusted for Dirichlet boundary conditions (i.e., prescribed values of the velocity at certain nodes). The most common Dirichlet condition is to set $u = v = 0$ at the bed to enforce a no-slip boundary condition. (A no-slip condition can also be enforced by setting the basal traction coefficient β to a very large value, but formally this is not a Dirichlet condition.) Also, it may be desirable to set u and v to observed values at certain locations, as in the Ross Ice Shelf test case (Section 8.2.6).

Suppose that at node (k, i, j) we have $u = u_c$ and $v = v_c$, where u_c and v_c are prescribed values. Let nr be the row of $\mathbf{A_{uu}}$ associated with this node, and let nc range over the columns with nonzero entries in this row. To enforce the Dirichlet condition, we set $\mathbf{A_{uu}}(nr, nc) = \mathbf{A_{vv}}(nr, nc) = 0$ for all values of nc except $nc = nr$ (the diagonal term); we set $\mathbf{A_{uu}}(nr, nr) = \mathbf{A_{vv}}(nr, nr) = 1$. In addition, we set $\mathbf{A_{uv}}(nr, nc) = \mathbf{A_{vu}}(nr, nc) = 0$ for all nc, since these two matrices do not contain any terms on the diagonal of the full global matrix (i.e., $\mathbf{A}$ in (6.6)). On the right-hand side, we set $\mathbf{b_u}(nr) = u_c$ and $\mathbf{b_v}(nr) = v_c$. These operations convert the matrix rows associated with node (k, i, j) to the equations $1 \cdot u = u_c, 1 \cdot v = v_c$, which clearly have the desired solutions u_c and v_c.

A further step is needed to maintain matrix symmetry, as required when using a preconditioned conjugate gradient solver. Consider the term $\mathbf{A_{uu}}(nr, nc)$, where nc is a specific column associated with a neighboring node (say, node $(k + 1, i + 1, j + 1)$). We have already set $\mathbf{A_{uu}}(nr, nc) = 0$, so we need to set $\mathbf{A_{uu}}(nc, nr) = 0$ to maintain symmetry. The Dirichlet condition is $u(nr) = u_c$. In the matrix-vector product, the product $\mathbf{A_{uu}}(nc, nr)u(nr) = \mathbf{A_{uu}}(nc, nr)u_c$ contributes to $\mathbf{b_u}(nc)$. Thus we can replace $\mathbf{b_u}(nc)$ with $\mathbf{b_u}(nc) - \mathbf{A_{uu}}(nc, nr)u_c$ and set $\mathbf{A_{uu}}(nc, nr) = 0$ without altering the problem. We do this for all the terms in the columns associated with node (k, i, j) (i.e., all the terms multiplied by u_c or v_c in the matrix-vector product). Thus, both the rows and the columns associated with node (k, i, j) are filled with zeros, except for the diagonal term, and the full global matrix remains symmetric.

Solving the linear system

Once the matrices and right-hand side vectors have been assembled, we solve the linear problem (6.7). Glissade supports three kinds of solvers:

- A native Fortran 90 preconditioned conjugate gradient (PCG) solver
- Links to the Sparse Linear Algebra Package (SLAP), with options for the generalized minimum residual (GMRES) and biconjugate gradient methods
- Links to Trilinos, with options for PCG and GMRES, preconditioned by incomplete lower-upper (ILU) factorization or a multigrid method

We describe each solver in turn.

Preconditioned conjugate gradient solver. The native PCG solver works directly with the assembled matrices $\mathbf{A_{uu}}$, $\mathbf{A_{uv}}$, $\mathbf{A_{vu}}$, and $\mathbf{A_{vv}}$, along with the right-hand side vectors $\mathbf{b_u}$ and $\mathbf{b_v}$. These matrices are passed to one of two PCG solvers: a "standard" solver or a Chronopolous-Gear solver. The standard method for solving $\mathbf{Ax} = \mathbf{b}$ can be summarized as follows:

- Let x_0 denote the initial guess for x ($x_0 = 0$ if no guess is available); $r = b - Ax$ is the residual vector; d is a conjugate direction vector; M is a preconditioning matrix: α, β, η_0, η_1 and η_2 are scalars; q and z are work vectors; and (r, z) is the dot product of two vectors.
- $r_0 = b - Ax_0$, $d_0 = 0$, $\eta_0 = 1$
- Do until converged:

```
  ⋆ Solve $Mz = r$ for $z$ (the preconditioning step)
  ⋆ $\eta_1 = (r, z)$
  ⋆ $\beta = \eta_1/\eta_0$
  ⋆ $d = z + \beta d$
  ⋆ $\eta_0 = \eta_1$
  ⋆ $q = Ad$
  ⋆ $\eta_2 = (d, q)$
  ⋆ $\alpha = \eta_1/\eta_2$
  ⋆ $x = x + \alpha d$
  ⋆ $r = r - \alpha q$ (or periodically, set $r = b - Ax$ and check for convergence)
```

There are two dot products, one matrix-vector product, and one preconditioning step per iteration. The convergence condition is $\sqrt{(r,r)}/\sqrt{(b,b)} < \epsilon$, where ϵ is a small tolerance (10^{-8} by default). The convergence check is done every five iterations, as a compromise between the expense of an extra matrix-vector multiply and that of unnecessary iterations.

This solver works either in serial or in parallel. If run in parallel, the dot products require global sums, and a halo update for d is needed once per iteration. Halo updates in CISM operate on 2D arrays with horizontal indices i and j (or 3D/4D arrays with additional indices). This is the main reason for leaving the global matrices and vectors in standard Fortran arrays, instead of converting to a sparse matrix storage format such as compressed row storage. In array form, it is easy to do halo updates of vectors u, v, r_u, r_v, d_u, d_v, etc.

The PCG solver has two preconditioning options:

- Diagonal (also known as Jacoby) preconditioning, in which the preconditioning matrix $\mathbf{M}$ consists of the diagonal terms of $\mathbf{A}$, so that $\mathbf{M}$ is trivial to invert. Convergence with diagonal preconditioning can be slow.
- Shallow-ice-based preconditioning. In this case $\mathbf{M}$ includes only the terms in $\mathbf{A}$ that link a given node to itself and its immediate neighbors above and below. The matrix $\mathbf{M}$ is then tridiagonal, as in the shallow-ice approximation, and can be inverted efficiently. This preconditioner works well when the physics is dominated by vertical shear and the horizontal terms are small, but not as well when membrane stresses are important. Since the preconditioner is local (it consists of independent column solves), it scales well with an increasing number of processors.

Other preconditioning options could be added in the future. If the existing options are inefficient for a given problem, it may be better to use Trilinos (see below).

For small problems on a modest number of processors, the dominant cost is the matrix-vector multiply required during each iteration. For large problems on many processors, however, the global sums become increasingly expensive. To reduce the cost of global sums, the standard PCG algorithm can be replaced by the Chronopoulos-Gear algorithm, which rearranges operations such that both global sums are done with a single MPI call. We omit the details here, but refer the reader to the code comments in module `glissade_velo_higher_pcg.F90`.

SLAP. The Sparse Linear Algebra Package (SLAP) is a set of Fortran routines for solving sparse systems of linear equations. SLAP was part of the original Glimmer release and is still used to solve the thickness evolution and temperature advection problems in Glide. It can also be used to solve higher-order systems in Glissade, using either GMRES or a biconjugate gradient solver. SLAP, however, is limited to a single processor and thus is unsuited for large problems.

In order for Glissade to use CISM's SLAP routines, it must convert the global matrices and the right-hand-side and solution vectors to a SLAP-friendly format. First (before any matrix assembly), Glissade assigns a unique integer ID to each active node. Following assembly, the code loops through the matrices $\mathbf{A_{uu}}$, $\mathbf{A_{uv}}$, $\mathbf{A_{vu}}$ and $\mathbf{A_{vv}}$, putting each nonzero entry into a Fortran derived type containing the row, column and value of such entry. Similarly, the current solution and the right-hand side are placed in SLAP-friendly vectors. This information is passed to the SLAP solver and ultimately to the various SLAP subroutines. SLAP returns the velocity solution and residual vectors, which are copied back into Glissade data structures.

Trilinos. Trilinos is a set of solver packages and related software developed at Sandia National Laboratories. Under the DOE ISICLES project, the Glam higher-order dycore was parallelized and linked to Trilinos. The Trilinos links developed for Glam were later adapted for Glissade. Trilinos has been tested extensively on parallel architectures and has been installed on DOE high-performance computers where CISM is often run. See Section 2.4.2 for instructions on building CISM with Trilinos.

A C++ file, `libglimmer-trilinos/trilinosGlissadeSolver.cpp`, contains procedures that link Glissade to Trilinos:

- `initializetgs` sends Trilinos a global ID for each active node.
- `insertrowtgs` sends Trilinos a row of the matrix (specifically, the global row index, the number of potentially nonzero column entries, the index and value of each such entry, and the associated right-hand side term).
- `solvevelocitytgs` returns the velocity solution.

Various Glissade subroutines gather the information needed by Trilinos. Before assembly, Glissade computes global IDs for each node (k, i, j) on each processor, along with unique indices for each unknown (u and v) on each active node. Glissade also computes a logical array of dimension $(27, nz, nx-1, ny-1)$—the same dimension as the global matrices—whose value is `.true.` for the potential nonzero entries. After assembly of $\mathbf{A_{uu}}$, $\mathbf{b_u}$, etc., the nonzero matrix entries are passed to Trilinos, one row at a time. After solution, the Trilinos velocity result is copied to Glissade data structures.

Trilinos solver options are set in a file called `trilinosOptions.xml` in the directory where the code is executed. Among the key settings are the solver type and preconditioner type. The default solver is `Block GMRES`, and the default preconditioner is `Ifpack`, which applies ILU preconditioning. ILU generally works well for problems where shallow-ice physics is dominant, but can struggle for shelf-type problems where membrane stresses are dominant. For these problems the `ML` preconditioner type, which uses a multigrid method for preconditioning, is likely to work better. Optimal Trilinos settings for large, complex problems are an area of active research.

Choosing among the various solver options is something of an art. The default is native PCG using the Chronopoulos-Gear algorithm. The PCG solver runs efficiently on most platforms and scales well. On the other hand, its preconditioning options are limited, so convergence may be slow for problems with dominant membrane stresses. SLAP solvers are generally robust and efficient, but are limited to one processor. Trilinos is typically slower than the other solvers, in part because of the extra cost of setting up Trilinos data structures, as well as the slower performance of C++ code relative to Fortran on many platforms. On the other hand, Trilinos contains a vast range of solver options (only a few of which have so far been tested with CISM), some of which may be more flexible and robust than those implemented in SLAP and the native PCG solver.

Solving the nonlinear system

Each call to a linear solver (native PCG, SLAP, or Trilinos) returns a solution vector, along with the number of iterations and the error, defined as $\sqrt{(r,r)}/\sqrt{(b,b)}$. If the linear solution fails to converge after the maximum allowed number of iterations (typically 200), the solver exits with the last computed solution (which may be adequate despite being unconverged).

Then a new global matrix is assembled, using the latest velocity solution to compute the effective viscosity. The right-hand is adjusted as needed to incorporate Dirichlet boundary conditions. Glissade then computes the new residual $b - Ax$. If the L_2 norm of the residual (defined as $\sqrt{(r,r)}$) is smaller than a desired threshold (10^{-4} by default), the nonlinear system of equations is considered solved. (An absolute threshold of 10^{-4} may be too stringent for large problems. Alternatively, Glissade can use a relative threshold, based on the ratio of the L2 norm of the residual to the L2 norm of **b**; see Section 7.3.) Otherwise the linear solver is called again, until either the solution converges or the maximum number of nonlinear iterations (typically 100) is reached.

As mentioned above, the procedure of updating the matrix with the latest guess for the solution is known as Picard iteration. The older Glam dycore includes an option to solve the nonlinear system using a Jacobian-Free Newton-Krylov (JFNK) method. JFNK requires an extra residual evaluation per nonlinear iteration, but generally converges in fewer iterations than does the Picard method. Glissade does not yet have a JFNK option.

In addition to the 3D Blatter-Pattyn approximation, Glissade can solve the simpler shallow-ice and shallow-shelf approximations, as well as the vertically integrated, higher-order “L1L2” approximation (Schoof and Hindmarsh, 2010). These are described next.

6.2.2 Shallow-ice approximation

SIA: matrix form

By setting `which_ho_approx = 0` in the config file, Glissade's finite-element solver can be used as an SIA solver. The shallow-ice equations follow from the Blatter-Pattyn equations if the horizontal-stress terms are neglected. The SIA analogs of (6.1) are

$$
\begin{aligned}
x &: \frac{\partial}{\partial z}\left(\eta \frac{\partial u}{\partial z}\right) = \rho g \frac{\partial s}{\partial x}, \\
y &: \frac{\partial}{\partial z}\left(\eta \frac{\partial v}{\partial z}\right) = \rho g \frac{\partial s}{\partial y},
\end{aligned}
\tag{6.47}
$$

leading to the following internal stress terms in weak form:

$$\begin{aligned} x &: \int_{\Omega} 2\eta \frac{\partial \varphi_i}{\partial z} \left(\frac{1}{2} \left[\frac{\partial \varphi_j}{\partial z} \right] \{u_j\} \right), \\ y &: \int_{\Omega} 2\eta \frac{\partial \varphi_i}{\partial z} \left(\frac{1}{2} \left[\frac{\partial \varphi_j}{\partial z} \right] \{v_j\} \right). \end{aligned} \tag{6.48}$$

These terms are integrated over elements using (6.18) and (6.21) for the 3D basis functions and Jacobians. The resulting 8x8 element matrices $\mathbf{K_{uu}}$ and $\mathbf{K_{vv}}$ link each node to the eight nodes of the element, including itself. Since there are no horizontal stress terms, $\mathbf{K_{uv}} = \mathbf{K_{vu}} = 0$. In the expression for effective viscosity, the effective strain rate is given by

$$\dot{\varepsilon}_e^2 = \dot{\varepsilon}_{xz}^2 + \dot{\varepsilon}_{yz}^2. \tag{6.49}$$

The basal and lateral boundary conditions and gravitational loading are handled just as for the Blatter-Pattyn case.

The resulting SIA global matrices contain about half as many nonzero terms as the BP matrices, since $\mathbf{A_{uv}} = \mathbf{A_{vu}} = 0$. Each ice column, however, cannot be solved independently of the others; rather, each node is linked to its horizontal neighbors by terms that arise during element assembly. As a result, the matrix-based SIA solver is not dramatically faster than the BP solver. This solver can be useful for code testing but is not practical for productions runs, since a practical SIA solver should be many times faster than a BP solver. Glissade has a much faster alternative SIA solver, described next.

SIA: Local form

Glissade's local shallow-ice solver (in `glissade_velo_sia.F90`), is distinct from the finite-element solvers described in this section. It is local in the sense that u and v in each ice column are found independently of u and v in all other columns. It resembles the Glide shallow-ice solver described in Section 4.1. Glide, however, incorporates the velocity solution in a diffusion equation for ice thickness. Glissade's local SIA solver computes u and v only, with thickness evolving separately as described in Section 6.4.

The local SIA solver first computes the basal velocity given the basal traction coefficient t_b, which is defined as in Glide:

$$\mathbf{u_b} = t_b \tau_\mathbf{b}. \tag{6.50}$$

Note that $t_b = 1/\beta$, where the higher-order traction coefficient β is defined as in (6.26). There are several options for setting t_b: no sliding ($t_b = 0$); uniform traction ($t_b =$ constant); uniform traction where basal water is present (and no sliding elsewhere); and uniform traction where the bed is at the pressure melting point, $T_b = T_{pmp}$ (and no sliding elsewhere). As in Glide, the basal velocity is proportional to t_b and the gravitational driving stress:

$$\mathbf{u_b} = -t_b \rho_i g \bar{H} \nabla s, \tag{6.51}$$

where $\bar{H}$ is the ice thickness interpolated to cell vertices and ∇s is the surface slope at vertices.

To find the interior velocities, we first compute a vertically integrated factor $c(\sigma)$ for each level at each cell vertex:

$$c(\sigma) = -2(\rho g)^n \bar{H}^{n+1} |\boldsymbol{\nabla} s|^{n-1} \int_1^{\sigma} \bar{A} \tilde{\sigma}^n d\tilde{\sigma}, \tag{6.52}$$

where $\bar{A}$ is the flow factor interpolated to vertices, and a tilde distinguishes σ at a particular level from the integration variable $\tilde{\sigma}$. (The same term appears in the Glide SIA calculation; see (4.22).) Discretized in the vertical, this becomes

$$c(k) = -2(\rho g)^n \bar{H}^{n+1} |\nabla s|^{n-1} \sum_{l=nz-1}^{k} \bar{A}_l \left(\frac{\sigma_l + \sigma_{l+1}}{2} \right)^n (\sigma_{l+1} - \sigma_l) . \tag{6.53}$$

The factors $c(k)$ are interpolated from cell vertices to edges. The u and v components of velocity are computed at cell edges as a function of the surface slope:

$$\begin{aligned} \Delta u_E(k,i,j) &= \sum_{l=nz-1}^{k} \left(\frac{c(k,i,j) + c(k,i,j-1)}{2} \right) (s(i+1,j) - s(i,j)), \\ \Delta v_N(k,i,j) &= \sum_{l=nz-1}^{k} \left(\frac{c(k,i,j) + c(k,i-1,j)}{2} \right) (s(i,j+1) - s(i,j)). \end{aligned} \tag{6.54}$$

Here, Δu_E is the u velocity component on the east edge of a cell, relative to the basal velocity, and similarly for Δv_N on the north edge of the cell.

Finally, we average Δu_E and Δv_N to vertices and add the bed velocities (6.51) to determine the velocity in the ice column at each vertex:

$$\begin{aligned} u(k,i,j) &= u_b(i,j) + \left(\frac{\Delta u_E(k,i,j) + \Delta u_E(k,i,j+1)}{2} \right), \\ v(k,i,j) &= v_b(i,j) + \left(\frac{\Delta v_N(k,i,j) + \Delta v_N(k,i+1,j)}{2} \right). \end{aligned} \tag{6.55}$$

For serial problems there are no special advantages to using Glissade's local SIA solver in place of Glide. Glissade's local SIA solver, however, can run on multiple processors, whereas Glide cannot. For large SIA problems, parallel Glissade can hold more data in memory and may be faster.

6.2.3 Shallow-shelf approximation

Glissade's finite-element solver can also be used as a shallow-shelf solver. The SSA equations can be derived by vertically integrating the 3D Blatter-Pattyn equations. They are valid when the basal shear stress is small or zero and the velocity is (to a good approximation) vertically uniform. The shallow-shelf analog of (6.1) is

$$\begin{aligned} x: \quad & \frac{\partial}{\partial x}\left(2\bar{\eta}H \left(2\frac{\partial u}{\partial x} + \bar{\eta}H \frac{\partial v}{\partial y} \right) \right) + \frac{\partial}{\partial y}\left(\bar{\eta}H \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right) = \rho g \frac{\partial s}{\partial x}, \\ y: \quad & \frac{\partial}{\partial y}\left(2\bar{\eta}H \left(2\frac{\partial v}{\partial y} + \bar{\eta}H \frac{\partial u}{\partial x} \right) \right) + \frac{\partial}{\partial x}\left(\bar{\eta}H \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \right) = \rho g \frac{\partial s}{\partial y}, \end{aligned} \tag{6.56}$$

where $\bar{\eta}$ is the vertically averaged effective viscosity. The SSA equations in weak form resemble (6.11), but with internal stress terms

$$\begin{aligned} x: & \int_{\Omega} 2\bar{\eta}H \left(\frac{\partial \varphi_i}{\partial x} \left(2\left[\frac{\partial \varphi_j}{\partial x}\right]\{u_j\} + \left[\frac{\partial \varphi_j}{\partial y}\right]\{v_j\} \right) + \frac{\partial \varphi_i}{\partial y} \left(\frac{1}{2}\left[\frac{\partial \varphi_j}{\partial y}\right]\{u_j\} + \frac{1}{2}\left[\frac{\partial \varphi_j}{\partial x}\right]\{v_j\} \right) \right), \\ y: & \int_{\Omega} 2\bar{\eta}H \left(\frac{\partial \varphi_i}{\partial y} \left(2\left[\frac{\partial \varphi_j}{\partial y}\right]\{v_j\} + \left[\frac{\partial \varphi_j}{\partial x}\right]\{u_j\} \right) + \frac{\partial \varphi_i}{\partial x} \left(\frac{1}{2}\left[\frac{\partial \varphi_j}{\partial x}\right]\{v_j\} + \frac{1}{2}\left[\frac{\partial \varphi_j}{\partial y}\right]\{u_j\} \right) \right). \end{aligned} \tag{6.57}$$

The resulting element matrices are similar to (6.14)–(6.17), except that the terms containing $\frac{\partial \varphi_i}{\partial z}$ and $\frac{\partial \varphi_j}{\partial z}$ are missing, and the viscosity term η is replaced by $\bar{\eta} H$. The effective viscosity is defined as in (6.2), but with a vertically averaged flow factor and with (6.3) replaced by

$$\dot{\varepsilon}_e^2 = \dot{\varepsilon}_{xx}^2 + \dot{\varepsilon}_{yy}^2 + \dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy} + \dot{\varepsilon}_{xy}^2. \tag{6.58}$$

The integrals in (6.57) are taken over 2D cells rather than 3D elements, with basis functions and Jacobians given by (6.28) and (6.29).

The basal boundary terms are handled as in the 3D Blatter-Pattyn approximation. The lateral boundary and gravitational forcing terms are computed initially in 3D (as for the Blatter-Pattyn case), but then are summed in the vertical before being inserted into the 2D right-hand vectors $\mathbf{b_u}$ and $\mathbf{b_v}$. The solution procedure is the same as for the 3D case, except that the problem has no vertical dimension. When using the native PCG solver, the shallow-ice-based preconditioner is inappropriate and the diagonal preconditioner should be used instead.

In short, the SSA problem is similar to the Blatter-Pattyn problem except for the missing vertical shear terms and the reduction to 2D. The solution is much faster, because the SSA matrices have roughly $3 * nz$ times fewer nonzero entries than the Blatter-Pattyn matrices. The factor of 3 arises from the fact that the BP equations at each level include connections to the levels above and below, whereas the SSA equations are solved at a single level.

6.2.4 L1L2 approximation

The L1L2 approximation (Schoof and Hindmarsh, 2010) can be derived by vertically integrating the Blatter-Pattyn equations with the assumption that the horizontal gradients of the membrane stresses are uniform with depth. (The term "L1L2" is based on the classification scheme of Hindmarsh (2004).) This assumption leads to a 2D matrix system (like the SSA), which can be solved much more cheaply than the 3D BP system. The L1L2 approximation is about as accurate as the BP equations in regions of fast sliding. In regions of little or no sliding, it reduces to the shallow-ice approximation. Thus it is accurate where the SIA is accurate, but not in regions of slow sliding and rough bed topography, where the horizontal gradients of the membrane stresses vary strongly with height. For example, L1L2 performs poorly for ISMIP-HOM Test A.

Consider the Blatter-Pattyn equations (6.1), written in terms of deviatoric stresses instead of strain rates:

$$\begin{aligned}
x: \quad & \frac{\partial}{\partial x}\left(2\tau_{xx} + \tau_{yy}\right) + \frac{\partial}{\partial y}\left(\tau_{xy}\right) + \frac{\partial}{\partial z}\left(\tau_{xz}\right) = \rho g \frac{\partial s}{\partial x}, \\
y: \quad & \frac{\partial}{\partial y}\left(2\tau_{yy} + \tau_{xx}\right) + \frac{\partial}{\partial x}\left(\tau_{xy}\right) + \frac{\partial}{\partial z}\left(\tau_{yz}\right) = \rho g \frac{\partial s}{\partial y}.
\end{aligned} \tag{6.59}$$

Assuming that τ_{xx}, τ_{yy} and τ_{xy} are depth-independent, (6.59) can be integrated from b to s to obtain

$$\begin{aligned}
x: \quad & \frac{\partial}{\partial x}\left(\int_b^s (2\tau_{xx} + \tau_{yy}) dz\right) + \frac{\partial}{\partial y}\left(\int_b^s \tau_{xy} dz\right) - \beta u_b = \rho g H \frac{\partial s}{\partial x}, \\
y: \quad & \frac{\partial}{\partial y}\left(\int_b^s (2\tau_{yy} + \tau_{xx}) dz\right) + \frac{\partial}{\partial x}\left(\int_b^s \tau_{xy} dz\right) - \beta v_b = \rho g H \frac{\partial s}{\partial y},
\end{aligned} \tag{6.60}$$

where u_b and v_b are the components of basal velocity, and we have used Leibniz's rule to move the partial derivatives outside the integrals and eliminate several boundary terms. Since the deviatoric stresses are depth-independent, they can be written in terms of the basal strain rates. Taking these strain rates outside the integral, we obtain

$$\begin{aligned} x: &\quad 2\frac{\partial}{\partial x}\left(\bar{\eta}H\left(2\frac{\partial u_b}{\partial x}+\frac{\partial v_b}{\partial y}\right)\right)+\frac{\partial}{\partial y}\left(\bar{\eta}H\left(\frac{\partial u_b}{\partial y}+\frac{\partial v_b}{\partial x}\right)\right)-\beta u_b=\rho g H\frac{\partial s}{\partial x}, \\ y: &\quad 2\frac{\partial}{\partial y}\left(\bar{\eta}H\left(2\frac{\partial v_b}{\partial y}+\frac{\partial u_b}{\partial x}\right)\right)+\frac{\partial}{\partial x}\left(\bar{\eta}H\left(\frac{\partial u_b}{\partial y}+\frac{\partial v_b}{\partial x}\right)\right)-\beta v_b=\rho g H\frac{\partial s}{\partial y}, \end{aligned} \tag{6.61}$$

where the vertically averaged effective viscosity is

$$\bar{\eta}=\int_b^s \eta dz. \tag{6.62}$$

Note the similarity between (6.61) and (6.56). When (6.56) is vertically integrated and basal boundary terms are included, it is formally identical to (6.61). Thus the same methods used to assemble and solve the SSA equations can be applied to the L1L2 equations.

The effective viscosity, however, is treated differently for L1L2. (Here we follow Perego et al. (2012).) The parallel norm $|\cdot|_{||}$ is defined as

$$|\dot{\varepsilon}|_{||}^2=\dot{\varepsilon}_{xx}^2+\dot{\varepsilon}_{yy}^2+\dot{\varepsilon}_{xx}\dot{\varepsilon}_{yy}+\dot{\varepsilon}_{xy}^2 \tag{6.63}$$

and the perpendicular norm $|\cdot|_{\perp}$ as

$$|\dot{\varepsilon}|_{\perp}^2=\dot{\varepsilon}_{xz}^2+\dot{\varepsilon}_{yz}^2. \tag{6.64}$$

The constitutive law can be written as

$$\dot{\varepsilon}_{ij}=A\tau_e^2\tau_{ij}, \tag{6.65}$$

where we assume $n=3$. For the L1L2 approximation the effective stress is given by

$$\tau_e^2=|\tau|_{||}^2+\tilde{\tau}_{\perp}^2, \tag{6.66}$$

with $\tilde{\tau}_{\perp}$ estimated based on the SIA:

$$\tilde{\tau}_{\perp}^2=[\rho g(s-z)\nabla s]^2. \tag{6.67}$$

Inserting (6.66) in (6.65) and taking the parallel norm gives

$$|\dot{\varepsilon}|_{||}=A(|\tau|_{||}^2+|\tau|_{\perp}^2)|\tau|_{||}. \tag{6.68}$$

Given $|\dot{\varepsilon}|_{||}$ from (6.63), equation (6.68) can be written in the form $x^3+ax+b=0$, a cubic equation that can be solved for $|\tau|_{||}$ using standard techniques. (Of the three roots, the one of interest is real and positive; the other two are complex conjugates. Glissade currently computes $|\tau|_{||}$ only for $n=3$, although an iterative scheme for general n could be added later.) The effective viscosity at depth z is then given by

$$\eta=\frac{1}{2A\tau_e^2}, \tag{6.69}$$

where τ_e is obtained from (6.66), (6.67) and (6.68). Equation (6.69) follows from (6.65) and the relation

$$\tau_{ij}=2\eta\dot{\varepsilon}_{ij}, \tag{6.70}$$

which defines the effective viscosity.

After solving the 2D L1L2 system for the basal velocity, the velocity profile at each vertex can be found by vertical integration. First we estimate τ_{xz} and τ_{yz} in each layer. Since the depth-independent membrane stresses are determined by the basal velocity, τ_{xz} and τ_{yz} are now the only unknowns in (6.59). These equations can be integrated from the top surface (assumed to be stress-free) to depth z to give

$$\begin{aligned} x: \quad \tau_{xz}(z) &= -\rho g(s-z)\frac{\partial s}{\partial x} + 2\frac{\partial}{\partial x}\left[\left(2\frac{\partial u_b}{\partial x} + \frac{\partial v_b}{\partial y}\right)\left(\int_z^h \eta dz'\right)\right] + 2\frac{\partial}{\partial y}\left[\left(\frac{\partial u_b}{\partial y} + \frac{\partial v_b}{\partial x}\right)\left(\int_z^h \eta dz'\right)\right], \\ y: \quad \tau_{yz}(z) &= -\rho g(s-z)\frac{\partial s}{\partial y} + 2\frac{\partial}{\partial y}\left[\left(2\frac{\partial v_b}{\partial y} + \frac{\partial u_b}{\partial x}\right)\left(\int_z^h \eta dz'\right)\right] + 2\frac{\partial}{\partial x}\left[\left(\frac{\partial u_b}{\partial y} + \frac{\partial v_b}{\partial x}\right)\left(\int_z^h \eta dz'\right)\right], \end{aligned} \tag{6.71}$$

where again we use Leibniz's rule to move the partial derivatives outside the integrals and remove boundary terms. The bracketed terms are evaluated at cell centers, and then the gradient terms are computed at nodes using finite-difference formulas (e.g., (6.43)). Given τ_{xz} and τ_{yz} at each level, along with $|\dot{\varepsilon}|_{\parallel}^2$ from (6.63), the effective stress is given by

$$\tau_e^2 = 2\eta\,|\dot{\varepsilon}|_{\parallel}^2 + \tau_{xz}^2 + \tau_{yz}^2. \tag{6.72}$$

The velocity components can then be integrated upward from the bed using

$$\begin{aligned} x: \quad & \frac{1}{2}\frac{\partial u}{\partial z} = \dot{\varepsilon}_{xz} = A\tau_e^{n-1}\tau_{xz}, \\ y: \quad & \frac{1}{2}\frac{\partial v}{\partial z} = \dot{\varepsilon}_{yz} = A\tau_e^{n-1}\tau_{yz}, \end{aligned} \tag{6.73}$$

which imply

$$\begin{aligned} x: \quad & u(z) = u_b + 2\int_b^z A\tau_e^{n-1}\tau_{xz}dz, \\ y: \quad & v(z) = v_b + 2\int_b^z A\tau_e^{n-1}\tau_{yz}dz. \end{aligned} \tag{6.74}$$

6.3 Temperature Solver

As discussed in Section 4.2, the thermal evolution of the ice sheet is given by

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c}\nabla^2 T - \mathbf{u}\cdot\nabla T - w\frac{\partial T}{\partial z} + \frac{\Phi}{\rho c}, \tag{6.75}$$

where T is the temperature in °C, k is the thermal conductivity of ice, c is the specific heat of ice, ρ is the density, and Φ is the rate of heating due to internal deformation and dissipation. This equation describes the conservation of internal energy under horizontal and vertical diffusion (the first term on the RHS), horizontal and vertical advection (the second and third terms, respectively), and internal heat dissipation (the last term). Glide solves this equation in module `glide_temp.F90`. Glissade takes a different approach, dividing the temperature evolution into separate advection and diffusion/dissipation components. Module `glissade_temp.F90` solves for diffusion and internal dissipation:

$$\frac{\partial T}{\partial t} = \frac{k}{\rho c} \nabla^2 T + \frac{\Phi}{\rho c}, \tag{6.76}$$

as described in this section. The advective part of (6.75) is described in Section 6.4.

Glissade's vertical discretization of temperature is also different from that of Glide. In Glide, T is located at each of nz vertical levels. In Glissade, internal temperatures are located at the midpoints of the $nz - 1$ layers. Temperature is also defined at the upper and lower ice surface, giving a total of $nz + 1$ temperature points in each column. The upper surface temperature is denoted by T_0 and the lower surface temperature by T_{nz}.

This convention makes it fairly straightforward to advect temperature conservatively. The total internal energy in a column is the sum over layers of $\rho c T \Delta z$, where Δz is the layer thickness. This internal energy is conserved under transport (see Section 6.4). T_0 and T_{nz}, which are determined by the boundary conditions, are associated with infinitesimally thin layers that do not contain any internal energy.

The following sections describe how the terms in (6.76) are computed, how the boundary conditions are specified, and how the equation is solved.

6.3.1 Vertical diffusion

Computing the vertical diffusion term requires a discretization for $\nabla^2 T$. As in Glide (Section 4.2.1), we assume that horizontal diffusion is negligible compared to vertical diffusion:

$$\nabla^2 T \simeq \frac{\partial^2 T}{\partial z^2} = \frac{1}{H^2} \frac{\partial^2 T}{\partial \sigma^2}, \tag{6.77}$$

where the last equality follows from $\sigma = (s - z)/H$.

In σ–coordinates, the central difference formulas for first derivatives at the upper and lower interfaces of layer k are

$$\begin{aligned} \left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_k} &= \frac{T_k - T_{k-1}}{\tilde{\sigma}_k - \tilde{\sigma}_{k-1}}, \\ \left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_{k+1}} &= \frac{T_{k+1} - T_k}{\tilde{\sigma}_{k+1} - \tilde{\sigma}_k}, \end{aligned} \tag{6.78}$$

where $\tilde{\sigma}_k$ is the value of σ at the midpoint of layer k, halfway between σ_k and σ_{k+1}:

$$\tilde{\sigma}_k = \frac{\sigma_{k+1} - \sigma_k}{2}. \tag{6.79}$$

The second partial derivative, defined at the midpoint of layer k, is

$$\left.\frac{\partial^2 T}{\partial \sigma^2}\right|_{\tilde{\sigma}_k} = \frac{\left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_{k+1}} - \left.\frac{\partial T}{\partial \sigma}\right|_{\sigma_k}}{\sigma_{k+1} - \sigma_k} \tag{6.80}$$

Inserting (6.78) in (6.80), we obtain the required vertical diffusion term:

$$\begin{aligned} \left.\frac{\partial^2 T}{\partial \sigma^2}\right|_{\tilde{\sigma}_k} = \frac{T_{k-1}}{(\tilde{\sigma}_k - \tilde{\sigma}_{k-1})(\sigma_{k+1} - \sigma_k)} - T_k \left(\frac{1}{(\tilde{\sigma}_k - \tilde{\sigma}_{k-1})(\sigma_{k+1} - \sigma_k)} + \frac{1}{(\tilde{\sigma}_{k+1} - \tilde{\sigma}_k)(\sigma_{k+1} - \sigma_k)} \right) \\ + \frac{T_{k+1}}{(\tilde{\sigma}_{k+1} - \tilde{\sigma}_k)(\sigma_{k+1} - \sigma_k)}. \end{aligned} \tag{6.81}$$

6.3.2 Heat dissipation

In higher-order models the internal heating rate Φ in (6.76) is given by the tensor product of strain rate and stress:

$$\Phi = \dot{\varepsilon}_{ij}\tau_{ij}. \tag{6.82}$$

The effective strain rate and effective stress (cf. (6.3)) are defined by

$$\dot{\varepsilon}_e^2 = \frac{1}{2}\dot{\varepsilon}_{ij}\dot{\varepsilon}_{ij}, \quad \tau_e^2 = \frac{1}{2}\tau_{ij}\tau_{ij}. \tag{6.83}$$

It follows from (6.82) and (6.83) that

$$\Phi = 2\dot{\varepsilon}_e\tau_e. \tag{6.84}$$

Eq. (6.70), which defines the effective viscosity, implies

$$\dot{\varepsilon}_e = \frac{\tau_e}{2\eta}, \tag{6.85}$$

which can be substituted in (6.84) to give

$$\Phi = \frac{\tau_e^2}{\eta}. \tag{6.86}$$

Both terms on the RHS of (6.86) are available to the temperature solver, since the higher-order velocity solver computes η during matrix assembly and diagnoses τ_e from η and $\dot{\varepsilon}_{ij}$ at the end of the calculation.

6.3.3 Boundary conditions

The temperature T_0 at the upper boundary is set to the surface air temperature T_{air}. (In coupled climate applications, T_0 may be passed in by the climate model.) The diffusive heat flux at the upper boundary (defined as positive up) is

$$F_d^{\text{top}} = \frac{k}{H}\frac{T_1 - T_0}{\tilde{\sigma}_1}. \tag{6.87}$$

(The denominator contains just one term because $\sigma_0 = 0$.) Optionally, this flux can be returned to the climate model as a lower boundary condition in the land-surface model.

The lower ice boundary is more complex. For grounded ice there are three heat sources and sinks. First, the diffusive flux from the bottom surface to the ice interior (positive up) is

$$F_d^{\text{bot}} = \frac{k}{H}\frac{T_{nz} - T_{nz-1}}{1 - \tilde{\sigma}_{nz-1}}. \tag{6.88}$$

Second, there is a geothermal heat flux F_g to the lower boundary. This is typically prescribed as a constant (~ 0.05 W m^{-2}) or read from an input file. Finally, there is a frictional heat flux associated with basal sliding, given by (Cuffey and Paterson, 2010, p. 418)

$$F_f = \boldsymbol{\tau_b} \cdot \mathbf{u_b}, \tag{6.89}$$

where $\boldsymbol{\tau_b}$ and $\mathbf{u_b}$ are 2D vectors of basal shear stress and basal velocity, respectively. With a friction law of the form (6.26), this becomes

$$F_f = \beta\sqrt{u_b^2 + v_b^2}. \tag{6.90}$$

If the basal temperature $T_{nz} < T_{\text{pmp}}$ (where $T_{\text{pmp}} = T + 8.7 \cdot 10^{-4}(s - z)$ is the pressure melting point temperature), then the fluxes at the lower boundary must balance:

$$F_g + F_f = F_d^{\text{bot}}. \tag{6.91}$$

In other words, the energy supplied by geothermal heating and sliding friction is equal to the energy removed by vertical diffusion. If, on the other hand, $T_{nz} = T_{\text{pmp}}$, then the lower surface temperature is fixed and the net flux is used to melt or freeze ice at the boundary:

$$M_b = \frac{F_g + F_f - F_d^{\text{bot}}}{\rho L}, \tag{6.92}$$

where M_b is the melt rate and L is the latent heat of melting. Melting generates basal water, which may either stay in place or flow downstream (possibly replaced by water from upstream), depending on the parameterization chosen in the config file (`basal_water`; see 7.3). We hold $T_{nz} = T_{\text{pmp}}$ as long as basal water is present.

For floating ice the basal boundary condition is simpler; T_{nz} is simply set to the freezing temperature T_f of seawater. Optionally, a melt rate could also be prescribed at the lower surface, but this is not currently implemented in CISM.

6.3.4 Vertical temperature solution

Eq. (6.76) can be discretized for an ice layer k as

$$\frac{T_k^{n+1} - T_k^n}{\Delta t} = \frac{k}{\rho c H^2}\left[a_k T_{k-1}^{n+1} - (a_k + b_k) T_k^{n+1} + b_k T_{k+1}^{n+1}\right] + \frac{\Phi_k}{\rho c}, \tag{6.93}$$

where the coefficients a_k and b_k are given by (6.81), n is the current time level, and $n+1$ is the new time level. The vertical diffusion terms are evaluated at the new time level, making the discretization backward Euler (i.e., fully implicit) in time. A Crank–Nicolson formulation, in which the temperature terms are evaluated at time $n + 1/2$, is also available. Although Crank–Nicolson is second-order-accurate in time (compared to first-order for backward Euler), it can lead to temperature oscillations in thin ice. For this reason, backward Euler is the default.

Eq. (6.93) can be rewritten as

$$-\alpha_k T_{k-1}^{n+1} + (1 + \alpha_k + \beta_k) T_k^{n+1} - \beta_k T_{k+1}^{n+1} = T_k^n + \frac{\Phi_k \Delta t}{\rho c}, \tag{6.94}$$

where

$$\alpha_k = \frac{a_k \Delta t}{\rho c}, \quad \beta_k = \frac{b_k \Delta t}{\rho c}. \tag{6.95}$$

At the upper surface, $T_0 = T_{\text{air}}$. At the lower surface we have either a temperature boundary condition ($T_{nz} = T_{\text{pmp}}$ for grounded ice, or $T_{nz} = T_f$ for floating ice) or a flux boundary condition:

$$F_f + F_g - \frac{k}{H}\frac{T_{nz}^{n+1} - T_{nz-1}^{n+1}}{1 - \tilde{\sigma}_{nz-1}} = 0, \tag{6.96}$$

which can be rearranged to give

$$-T_{nz-1}^{n+1} + T_{nz}^{n+1} = \frac{(F_f + F_g)\, H\, (1 - \tilde{\sigma}_{nz-1})}{k}. \tag{6.97}$$

In each ice column the above equations form a tridiagonal system that is easily solved for T_k.

Occasionally, the solution T_k in one or more layers will exceed T_{pmp} for the layer. If so, we set $T_k = T_{\text{pmp}}$ and use the extra energy to melt ice internally. This melt is assumed (not very realistically) to drain immediately to the bed.

If (6.92) applies, we compute M_b and adjust the basal water depth. When the basal water goes to zero, T_{nz} is set to a value slightly below T_{pmp} so that the flux boundary condition will apply during the next time step.

6.3.5 Enthalpy model

An alternative vertical temperature solver based on enthalpy is under development. In this scheme, water that melts internally can be retained in the ice instead of draining instantly to the bed. The retained water reduces the ice viscosity compared to that of pure ice. Although CISM includes source code for an enthalpy model, the model is not yet scientifically supported.

6.4 Mass and Tracer Transport

Ice sheet models must solve a transport equation for ice thickness H:

$$\frac{\partial H}{\partial t} + \nabla \cdot (H\mathbf{U}) = B, \tag{6.98}$$

where $\mathbf{U}$ is the vertically averaged 2D velocity and B is the total surface mass balance. Eq. (6.98) describes the conservation of ice volume under horizontal transport. With the assumption of uniform density, volume conservation is equivalent to mass conservation. There is a similar conservation equation for the internal energy in each ice layer:

$$\frac{\partial (hT)}{\partial t} + \nabla \cdot (hT\mathbf{u}) = 0, \tag{6.99}$$

where h is the layer thickness, T is the temperature (defined at the layer midpoint), and $\mathbf{u}$ is the horizontal velocity. If other tracers (e.g., internal water fraction or ice age) are present, their transport is described by equations of the same form as (6.99). In the discussion below, everything said about temperature and internal energy applies also to other tracers and the associated conserved quantities.

Unlike Glide, which solves for mass transport and temperature advection independently, Glissade solves (6.98) and (6.99) in a coordinated way, one layer at a time. (Advection of tracers other than temperature is carried out in conjunction with temperature advection.) This coordination makes it possible to avoid numerical oscillations with undesirable effects, such as raising T above the melting point. The non–advective terms of the energy equation (6.75) (i.e., vertical diffusion and internal dissipation) are handled in a separate vertical column solve, as described in Section 6.3.

Glissade has two horizontal transport schemes: a first-order upwind scheme and a more accurate incremental remapping (IR) scheme (Dukowicz and Baumgardner, 2000; Lipscomb and Hunke, 2004). The transport driver is in `glissade_transport.F90`, and the IR scheme is in `glissade_remap.F90`. The IR scheme was originally implemented in the Los Alamos sea ice model, CICE, and has been adapted for CISM. It is fairly complex and is described in detail in Section 6.4.1.

Following horizontal transport, the mass balance is applied at each surface. Ice is added to or removed from the top surface depending on the sign of B_b, and similarly is removed from or added to the basal layer based on the sign of M_b. Any energy that is available for melt after an ice column has entirely melted is discarded, but in coupled applications could be returned to the climate model if needed to conserve energy.

After the transport scheme and mass-balance update have been applied, the new layer thicknesses generally do not have the desired spacing in σ coordinates. A vertical remapping scheme is used to move ice thickness (and associated tracers) between layers to restore σ layers in a way that conserves mass and energy. This scheme computes overlaps between the new layers and the target σ layers. If, for example, part of new layer k overlaps with target layer $k + 1$, the mass and internal energy are computed for the overlap region, with the assumption that temperature is uniform within each layer. (A more accurate reconstruction of temperature may be added in the future.) Mass and internal energy are then transferred between layers, and the new layer temperatures are derived from the ratio of internal energy to mass.

6.4.1 Incremental remapping

Next we describe the incremental remapping scheme, which has several desirable features:

- It conserves the quantities being transported (mass and energy).
- It is non-oscillatory; that is, it does not create spurious ripples in the transported fields.
- It preserves tracer monotonicity. That is, it does not create new extrema in temperature; the values at time $m + 1$ are bounded by the values at time m.
- It is second-order accurate in space and therefore is less diffusive than first-order schemes. The accuracy may be reduced locally to first order to preserve monotonicity.
- It is efficient for large numbers of tracers. Much of the work is geometrical and is performed only once per cell edge instead of being repeated for each quantity being transported. (This is more of an issue for CICE than for CISM, but for some applications it might be desirable to carry additional tracers in CISM.)

The upwind scheme, like IR, is conservative, non-oscillatory, and monotonicity-preserving, but because it is first-order it is highly diffusive.

The IR time step is limited by the requirement that trajectories projected backward from grid cell corners are confined to the four surrounding cells; this is what is meant by incremental remapping as opposed to general remapping. This requirement leads to a Courant–Friedrichs–Lewy (CFL) condition,

$$\frac{\max|\mathbf{u}|\Delta t}{\Delta x} \leq 1 \tag{6.100}$$

For highly divergent velocity fields the maximum time step must be reduced by a factor of two to ensure that trajectories do not cross. However, ice-sheet velocity fields usually have small divergence per time step relative to the grid size.

The remapping algorithm can be summarized as follows:

1. Given mean values of the ice thickness and tracer fields in each grid cell, construct linear approximations of these fields. Limit the field gradients to preserve monotonicity.
2. Given ice velocities at grid cell corners, identify departure regions for the fluxes across each cell edge. Divide these departure regions into triangles and compute the coordinates of the triangle vertices.
3. Integrate the area and tracer fields over the departure triangles to obtain the area, volume, and energy transported across each cell edge.
4. Given these transports, update the state variables.

These steps are carried out for each of $nz - 1$ ice layers, as described below.

Reconstructing area and tracer fields

First, using the known values of the state variables, the ice thickness and tracer fields are reconstructed in each grid cell as linear functions of x and y. For each field we compute the value at the cell center, along with gradients in the x and y directions. The gradients are limited to preserve monotonicity. When integrated over a grid cell, the reconstructed fields must have mean values equal to the known state variables, denoted by $\bar{h}$ for ice thickness and $\tilde{T}$ for temperature.

Consider first the ice thickness in a given layer. We contruct a field $h(\mathbf{r})$ whose mean is $\bar{h}$, where $\mathbf{r} = (x, y)$ is the position vector relative to the cell center. That is, we require

$$\int_A h\,dh = \bar{h}\,A, \tag{6.101}$$

where $A = \int_A dA$ is the grid cell area. Eq. (6.101) is satisfied if $h(\mathbf{r})$ has the form

$$h(\mathbf{r}) = \bar{h} + \alpha_h \,\langle \nabla h \rangle \cdot (\mathbf{r} - \bar{\mathbf{r}}), \tag{6.102}$$

where $\langle \nabla h \rangle$ is a centered estimate of the thickness gradient within the cell, α_h is a limiting coefficient that enforces monotonicity, and $\bar{\mathbf{r}}$ is the cell centroid in a local coordinate system whose origin lies at the cell center:

$$\bar{\mathbf{r}} = \frac{1}{A}\int_A \mathbf{r}\,dA.$$

On CISM's structured rectangular grid the cell center and centroid coincide, so that $\bar{\mathbf{r}} = 0$ and $h(\bar{\mathbf{r}}) = \bar{h}$.

Next consider the ice temperature field in a layer of thickness h. The reconstructed temperature must satisfy

$$\int_A h\,T\,dA = \bar{h}\,\tilde{T}\,A, \tag{6.103}$$

where $\tilde{T} = T(\tilde{\mathbf{r}})$ is the temperature at the center of ice mass. Eq. (6.103) is satisfied when $T(\mathbf{r})$ is given by

$$T(\mathbf{r}) = \tilde{T} + \alpha_T \,\langle \nabla T \rangle \cdot (\mathbf{r} - \tilde{\mathbf{r}}), \tag{6.104}$$

where $\langle \nabla T \rangle$ is a centered estimate of the thickness gradient, α_T is a limiting coefficient, and the center of ice mass $\tilde{\mathbf{r}}$ is given by

$$\tilde{\mathbf{r}} = \frac{1}{\bar{h}\,A}\int_A h\,\mathbf{r}\,dA. \tag{6.105}$$

Evaluating the integrals, we find that the components of $\tilde{\mathbf{r}}$ are

$$\begin{aligned} \tilde{x} &= \frac{h_c\overline{x} + h_x\overline{x^2} + h_y\overline{xy}}{\bar{h}}, \\ \tilde{y} &= \frac{h_c\overline{y} + h_x\overline{xy} + h_y\overline{y^2}}{\bar{h}}. \end{aligned} \tag{6.106}$$

where $h_c = \bar{h}$ is the thickness at the cell center, (h_x, h_y) is the limited thickness gradient, and the terms with overbars are geometric means. On a rectangular mesh, only the terms proportional to $\overline{x^2}$ and $\overline{y^2}$ are nonzero. From (6.104), the temperature at the cell center is given by

$$T_c = \tilde{T} - T_x\tilde{x} - T_y\tilde{y},$$

where (T_x, T_y) is the limited gradient of temperature.

We preserve monotonicity by limiting the gradients. If $\bar{\phi}(i,j)$ denotes the mean value of some field in grid cell (i,j), we first compute centered gradients of $\bar{\phi}$ in the x and y directions, then check whether these gradients give values of ϕ within cell (i,j) that lie outside the range of $\bar{\phi}$ in the cell and its eight neighbors. Let $\bar{\phi}_{\max}$ and $\bar{\phi}_{\min}$ be the maximum and minimum values of $\bar{\phi}$ over the cell and its neighbors, and let $\phi_{\max}$ and $\phi_{\min}$ be the maximum and minimum values of the reconstructed ϕ within the cell. Since the reconstruction is linear, $\phi_{\max}$ and $\phi_{\min}$ are located at cell corners. If $\phi_{\max} > \bar{\phi}_{\max}$ or $\phi_{\min} < \bar{\phi}_{\min}$, we multiply the unlimited gradient by $\alpha = \min(\alpha_{\max}, \alpha_{\min})$, where

$$\begin{aligned}\alpha_{\max} &= (\bar{\phi}_{\max} - \bar{\phi})/(\phi_{\max} - \bar{\phi}),\\ \alpha_{\min} &= (\bar{\phi}_{\min} - \bar{\phi})/(\phi_{\min} - \bar{\phi}).\end{aligned} \tag{6.107}$$

Otherwise the gradient need not be limited.

Locating departure triangles

The method for locating departure triangles is discussed in detail by Dukowicz and Baumgardner (2000). The basic idea is illustrated in Figure 6.1, which shows a shaded quadrilateral departure region whose contents are transported to the target or home grid cell, labeled H. The neighboring grid cells are labeled by compass directions: NW, N, NE, W, and E. The four vectors point along the velocity field at the cell corners, and the departure region is formed by joining the starting points of these vectors. Instead of integrating over the entire departure region, it is convenient to compute fluxes across cell edges. We identify departure regions for the north and east edges of each cell, which are also the south and west edges of neighboring cells. Consider the north edge of the home cell, across which there are fluxes from the neighboring NW and N cells. The contributing region from the NW cell is a triangle with vertices abc, and that from the N cell is a quadrilateral that can be divided into two triangles with vertices acd and ade. Focusing on triangle abc, we first determine the coordinates of vertices b and c relative to the cell corner (vertex a), using Euclidean geometry to find vertex c. Then we translate the three vertices to a coordinate system centered in the NW cell. This translation is needed in order to integrate fields in the coordinate system where they have been reconstructed. Repeating this process for the north and east edges of each grid cell, we compute the vertices of all the departure triangles associated with each cell edge.

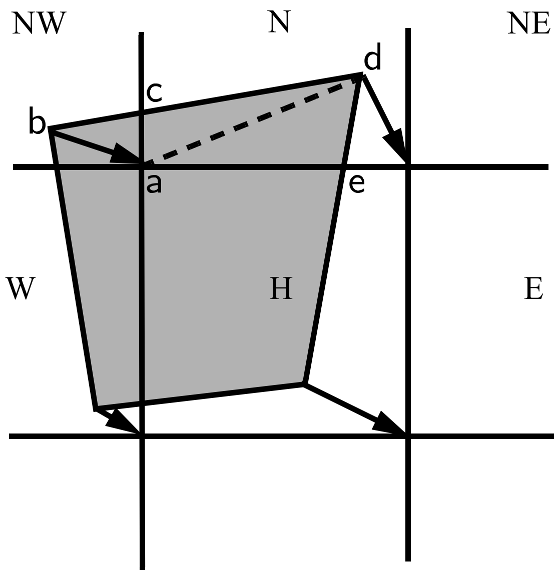


Figure 6.1: In incremental remapping, conserved quantities are remapped from the shaded departure region, a quadrilateral formed by connecting the backward trajectories from the four cell corners, to the grid cell labeled H. The region fluxed across the north edge of cell H consists of a triangle (abc) in the NW cell and a quadrilateral (two triangles, acd and ade) in the N cell.

Figure 6.2, reproduced from Dukowicz and Baumgardner (2000), shows all possible triangles that can contribute fluxes across the north edge of a grid cell. There are 20 triangles, which

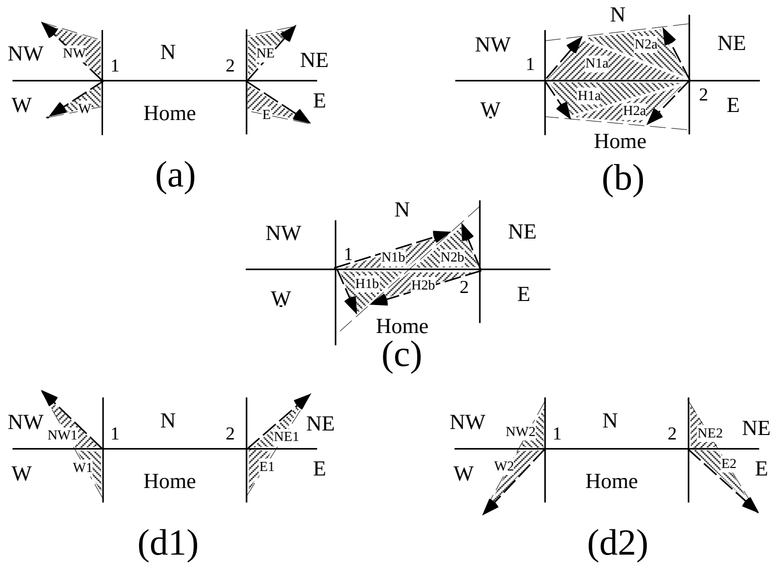


Figure 6.2: The 20 possible triangles that can contribute fluxes across the north edge of a grid cell.

can be organized into five groups of four mutually exclusive triangles as shown in Table 6.1. In this table, (x_1, y_1) and (x_2, y_2) are the Cartesian coordinates of the departure points relative to the northwest and northeast cell corners, respectively. The departure points are joined by a straight line that intersects the west edge at $(0, y_a)$ relative to the northwest corner and intersects the east edge at $(0, y_b)$ relative to the northeast corner. The east cell triangles and selecting conditions are identical except for a rotation through 90 degrees.

Departure triangles across a given cell edge are computed in a local coordinate system whose origin lies at the midpoint of the edge and whose vertices are at (-0.5, 0) and (0.5, 0). Intersection points are computed assuming Cartesian geometry with cell edges meeting at right angles. Let CL and CR denote the left and right vertices, which are joined by line CLR. Similarly, let DL and DR denote the departure points, which are joined by line DLR. Also, let IL and IR denote the intersection points (0, y_a) and (0,y_b) respectively, and let IC = (x_c, 0) denote the intersection of CLR and DLR. It can be shown that y_a, y_b, and x_c are given by

$$
\begin{aligned}
y_a &= \frac{x_{CL}(y_{DM} - y_{DL}) + x_{DM} y_{DL} - x_{DL} y_{DM}}{x_{DM} - x_{DL}}, \\
y_b &= \frac{x_{CR}(y_{DR} - y_{DM}) - x_{DM} y_{DR} + x_{DR} y_{DM}}{x_{DR} - x_{DM}}, \\
x_c &= (x_{DL} - y_{DL}) \frac{(x_{DR} - x_{DL})}{(y_{DR} - y_{DL})}.
\end{aligned}
\tag{6.108}
$$

Each departure triangle is defined by three of the seven points (CL, CR, DL, DR, IL, IR, IC).

In Dukowicz and Baumgardner (2000), departure points are defined by projecting cell corner velocities directly backward. That is,

$$
\mathbf{x_D} = -\mathbf{u}\Delta t, \tag{6.109}
$$

where $\mathbf{x}_D$ is the location of the departure point relative to the cell corner and $\mathbf{u}$ is the velocity at the corner. This approximation is only first-order accurate. In CISM, accuracy is improved by estimating the velocity at the midpoint of the trajectory.

Triangle group	Triangle label	Selecting logical condition
1	NW	$y_a > 0$ and $y_1 \geq 0$ and $x_1 < 0$
	NW1	$y_a < 0$ and $y_1 \geq 0$ and $x_1 < 0$
	W	$y_a < 0$ and $y_1 < 0$ and $x_1 < 0$
	W2	$y_a > 0$ and $y_1 < 0$ and $x_1 < 0$
2	NE	$y_b > 0$ and $y_2 \geq 0$ and $x_2 > 0$
	NE1	$y_b < 0$ and $y_2 \geq 0$ and $x_2 > 0$
	E	$y_b < 0$ and $y_2 < 0$ and $x_2 > 0$
	E2	$y_b > 0$ and $y_2 < 0$ and $x_2 > 0$
3	W1	$y_a < 0$ and $y_1 \geq 0$ and $x_1 < 0$
	NW2	$y_a > 0$ and $y_1 < 0$ and $x_1 < 0$
	E1	$y_b < 0$ and $y_2 \geq 0$ and $x_2 > 0$
	NE2	$y_b > 0$ and $y_2 < 0$ and $x_2 > 0$
4	H1a	$y_a y_b \geq 0$ and $y_a + y_b < 0$
	N1a	$y_a y_b \geq 0$ and $y_a + y_b > 0$
	H1b	$y_a y_b < 0$ and $\tilde{y}_1 < 0$
	N1b	$y_a y_b < 0$ and $\tilde{y}_1 > 0$
5	H2a	$y_a y_b \geq 0$ and $y_a + y_b < 0$
	N2a	$y_a y_b \geq 0$ and $y_a + y_b > 0$
	H2b	$y_a y_b < 0$ and $\tilde{y}_2 < 0$
	N2b	$y_a y_b < 0$ and $\tilde{y}_2 > 0$

Table 6.1: Evaluation of contributions from the 20 triangles across the north cell edge. The coordinates x_1, x_2, y_1, y_2, y_a, and y_b are defined in the text. We define $\tilde{y}_1 = y_1$ if $x_1 > 0$, else $\tilde{y}_1 = y_a$. Similarly, $\tilde{y}_2 = y_2$ if $x_2 < 0$, else $\tilde{y}_2 = y_b$.

Integrating fields

Next, we integrate the reconstructed fields over the departure triangles to find the total volume and internal energy transported across each cell edge. Volume transports are easy to compute since the ice thickness is linear in x and y. Given a triangle with vertices $\mathbf{x_i} = (x_i, y_i)$, $i \in \{1, 2, 3\}$, the triangle area is

$$A_T = \frac{1}{2} \left| (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \right| . \tag{6.110}$$

The integral F_a of any linear function $f(\mathbf{r})$ over a triangle is given by

$$F_a = A_T f(\mathbf{x_0}), \tag{6.111}$$

where $\mathbf{x}_0 = (x_0, y_0)$ is the triangle midpoint,

$$\mathbf{x}_0 = \frac{1}{3} \sum_{i=1}^{3} \mathbf{x}_i. \tag{6.112}$$

Thus, to compute the volume transport, we evaluate the thickness at the midpoint,

$$h(\mathbf{x}_0) = h_c + h_x x_0 + h_y y_0, \tag{6.113}$$

and multiply by A_T. By convention, northward and eastward transport is positive, while southward and westward transport is negative.

Eq. (6.111) cannot be used for energy transport, because the reconstructed internal energy is a quadratic function of position. (It is the product of two linear functions, for thickness and temperature.) The integral of a quadratic polynomial over a triangle requires function evaluations at three points,

$$F_h = \frac{A_T}{3} \sum_{i=1}^{3} f\left(\mathbf{x}_i'\right), \tag{6.114}$$

where $\mathbf{x}_i' = (\mathbf{x}_0 + \mathbf{x}_i)/2$ are points lying halfway between the midpoint and the three vertices.

Updating state variables

Finally, we compute new values of the state variables in each ice layer of each grid cell. The new ice thickness $h'(i,j)$ is given by

$$h'(i,j) = h(i,j) + \frac{F_E(i-1,j) - F_E(i,j) + F_N(i,j-1) - F_N(i,j)}{A(i,j)} \tag{6.115}$$

where $F_E(i,j)$ and $F_N(i,j)$ are the volume transports across the east and north edges, respectively, of cell (i,j), and $A(i,j)$ is the grid cell area. All transports added to one cell are subtracted from a neighboring cell; thus (6.115) conserves total ice volume.

The new internal energy in each layer is computed analogously. New temperatures are then given by the ratio of energy to volume. (Other tracers, if present, are updated in the same way.) Tracer monotonicity is ensured because

$$T' = \frac{\int_A h\,T\,dA}{\int_A h\,dA},$$

where T' is the new-time temperature, given by integrating the old-time thickness and temperature over a Lagrangian departure region with area A. That is, the new-time temperature is a weighted averages over old-time values, with non-negative weights h. Thus the new-time values must lie between the maximum and minimum of the old-time values.

6.4.2 CFL checks

As mentioned above, the time step used for explicit advection is limited by the advective CFL condition (6.100). Furthermore, for ice flux parallel to ∇s, ice thickness evolution is diffusive, giving rise to an additional diffusive CFL condition (Bueler and Brown, 2009):

$$\frac{\max D\Delta t}{\Delta x^{\frac{1}{2}}} \leq 2 \tag{6.116}$$

Flow governed by the shallow-ice approximation is always subject to this diffusive CFL, but some component of higher-order flow may not be because the ice velocity may not be parallel to ∇s (Bueler and Brown, 2009).

For these reasons, Glissade's transport scheme checks both the advective and diffusive CFL conditions and writes a warning to the log file if either is violated. These warnings indicate the i, j indices (on the global domain) of the point where the worst violation occurred, as well as the maximum allowable time step given the CFL condition(s).

The diffusivity is approximated using the component of velocity in the downslope direction:

$$D = \frac{(\boldsymbol{u} \cdot \nabla s)H}{|\nabla s|} \tag{6.117}$$

Future work may identify a less restrictive diffusivity, so the diffusive CFL warnings currently written may be overly restrictive. In other words, it is possible for a stable simulation to generate a diffusive CFL warning.

Advective CFL violations generally lead to stability problems with thickness evolution. However, since CFL violations may occur in portions of the domain that are not of interest to the user, these violations are not set to be a fatal error. Note, however, that the IR scheme may generate a "departure points out of bounds" error when advective CFL violations occur. This is a failure in the method that does result in a fatal error.

Eventually, CISM may include an adaptive time-stepping scheme that adjusts the time step based on the advective and diffusive CFL conditions. At present, however, the time step must be set manually and remain constant over a simulation. To aid in determining an appropriate time step, the `adv_cfl_dt` and `diff_cfl_dt` variables can be added to an output file. These variables are the maximum allowable time steps at each time in the model based on the advective and diffusive CFL conditions, respectively. In some cases, stability might require taking half of the reported values.

6.5 Other model physics

The Glissade modules for velocity, temperature, and transport replace the corresponding Glide modules when `dycore` = 2 is set in the config file. Other model physics, however, remains similar to Glide. In particular:

- Isostasy is treated in the same way as in Glide (see Section 4.4). The lithosphere can be treated as local or elastic, and the asthenosphere is either fluid or relaxing. Note that the elastic lithosphere calculation is non-local and has not been parallelized. Thus it is not possible to simulate an elastic lithosphere when running on more than one processor.

- Typically, the geothermal heat flux is set to a constant or prescribed from an input file. As in Glide, however, it is also possible to compute the geothermal heat flux from a model of heat flow in bedrock (see Section 4.3.2). This model includes non-local horizontal diffusion which, like the elastic lithosphere, has not been parallelized and therefore will not work on multiple processors.

- The calving options in Glissade are the same as those in Glide and are specified by the config variable `marine_margin`. These options are very simple (e.g., remove floating ice whenever the bedrock elevation is lower than a prescribed value). Since all the current options are local, they should work in parallel.
- The basal water options in Glissade are also the same as those in Glide. These are all local except for `basal_water` = 2, which computes a flux of basal water based on a steady-state routing calculation. The routing calculation is not parallel and has not been tested with Glissade.

A sophisticated basal hydrology model is under development and will likely be included in future versions of CISM. This model will output the effective pressure, which is a required input for the Coulomb friction law (`which_ho_babc` = 10) recently introduced in CISM. For now the effective pressure is either read from an input file or parameterized simply (using `basal_water` = 4). The Coulomb friction law is not yet scientifically supported and should be used at your own risk.

In summary, although CISM has a sophisticated higher-order dycore, the physics parameterizations remain fairly simple, especially for runs with multiple processors. This is an area of active development.

Chapter 7

Running CISM

7.1 Overview of Running CISM

Assuming you successfully completed the Installation instructions in Chapter 2, the executable for running the model, `cism_driver`, can be found in your build directory in a subdirectory called `cism_driver` (e.g., `./builds/mac-gnu/cism_driver/cism_driver`).

The build system creates the executable at this path but does not automatically make it available to other locations on your system. How you choose to do so depends on your situation. See the introduction to Chapter 8 for an overview of how to make the executable available to other locations on your system (e.g., symlinking, copying, or modifying your PATH environment variable).

Unlike previous versions of Glimmer, CISM 2.0 has a single executable, `cism_driver`, for running the model in all configurations. `cism_driver` can be invoked with a single argument specifying a CISM .config file to run CISM as a standalone ice sheet model without Glint climate forcing, or with two arguments (a CISM config file and a Glint config file) to run CISM with Glint climate forcing:

```
Call cism_driver with either 1 or 2 arguments. Examples:
cism_driver ice_sheet.config
cism_driver ice_sheet.config climate.config
```

The available options for the CISM configuration file and for the Glint climate interface configuration file are described in detail below.

To perform a parallel run with the parallel build of CISM, you must use the MPI run command, which is typically `mpirun` or `mpiexec` but may vary among MPI versions and installations. A standard syntax that is likely to work on most installations is

`mpirun -np N cism_driver ice_sheet.config` *`climate.config`*

where `N` is the number of processors you want to use, `ice_sheet.config` is the name of the CISM configuration file, and the optional argument *`climate.config`* is the name of the climate configuration file. For example:

`mpirun -np 4 cism_driver dome.config`

would run the dome test case on four processors.

When CISM runs, some basic information about its operation will be output to the screen (standard out). More verbose information about the run will be written to a log file which is named *`ice_sheet.config`*`.log`, where *`ice_sheet.config`* is the name of the .config file used to perform the run. (For example, if running the model with `./cism_driver dome.config`, the log file will be called `dome.config.log`.) The log file is an important reference, especially for debugging runs that do not behave as expected. For example, this file includes a list of configuration options and parameter values, which can be useful in diagnosing problems like typos in your .config file. The log file also indicates what files were used for input and output and at which times I/O occurred. The log file may contain warnings about potentially problematic configuration combinations or model behavior, such as the use of configurations settings that are not scientifically validated, or a CFL violation during advection. In contrast, fatal errors will kill the model and the error message will be written to both the screen and the log file.

Optionally, the log file also contains diagnostic information about the global state of the ice sheet (e.g., the total ice area and volume, the maximum surface and basal speeds, and the max and min temperatures), along with vertical profiles of speed and temperature at a user-specified grid point. This information is written at intervals specified by the config file variable `dt_diag`, for the diagnostic point `(idiag,jdiag)`.

In addition to the log file, the model will create any netCDF output files requested in the config file (see Section 7.4 below for details). If the model dies for some reason midway through a simulation, the netCDF files will still include output for the part of the simulation that was completed.

CISM2 (like its predecessor, Glimmer-CISM) has been coupled to the Community Earth System Model[1] and will be included in future CESM releases. Lipscomb et al. (2013) described the initial implementation of CISM in CESM, with one-way forcing of the Greenland ice sheet (using the shallow-ice approximation) by the surface mass balance computed in CESM's land model. Interactive coupling between ice sheets and other climate components has recently been implemented and is now being tested. This document, however, does not provide instructions for running CISM within CESM or other climate models. Please see here[2] for guidance on running CISM in CESM.

7.2 Overview of Configuration Files

Running CISM is managed through configuration files (*.config) that enable desired model features and control input of initial conditions and forcing and output of model results. This chapter summarizes the configuration options available for running CISM and is divided into sections on general Model Configuration, Input/Ouput Configuration, and optional Climate Forcing Configuration.

The format of CISM configuration files is taken from that used by the ConfigParser module in Python 2.x, which is similar to Windows `.ini` files and contains sections. Each section contains key/value pairs.

Comments:	Empty lines, or lines starting with a `#`, `;` or `!` are ignored. Comments can also be added on the same line as a key/value pair using these delimiters.
Sections:	A new section starts with the section name enclosed by square brackets, `[ ]` and can be up to 50 characters long, e.g., `[grid]`.
Key/Value Pairs:	Keys are separated from their associated values by `=` or `:`. The names can be up to 50 characters long. Values can be up to 400 characters long.

Sections and keys are case-sensitive and may contain white space. However, the configuration parser is very simple and thus the number of spaces within a key or section name also matters. Sensible defaults are used when a specific key is not found; defaults are shown in bold in the tables below.

Here is an example configuration file:

```
;a comment
[a section]
```

[1]www2.cesm.ucar.edu
[2]www.cesm.ucar.edu/models/cesm1.2/cism

```
an_int  : 1
a_float = 2.0
a_char  = hey, this is rather cool
an_array = 10. 20. -10. 40. 100.

[another section]
! more comments
foo : bar
```

7.3 Model Configuration

General model configuration options specify the grid and time-stepping used by the model, the dynamical core (dycore) used, and control various optional physics packages and parameter values. The `[grid]` and `[time]` configuration sections are always required. Also, while not required, in almost all situations `[options]` and (if using a higher-order dycore) `[ho_options]` will be included in .config files. The `[parameters]` and `[sigma]` sections are also commonly used. The `[GTHF]`, `[isostasy]`, and `[projection]` sections are needed only if the associated features are desired. Details of each of these sections, what they control, and the available options for each section are listed in the tables below.

`[grid]`	
Define model grid.	
`ewn`	(integer) number of nodes in x–direction
`nsn`	(integer) number of nodes in y–direction
`upn`	(integer) number of nodes in z–direction
`dew`	(real) node spacing in x–direction (m)
`dns`	(real) node spacing in y–direction (m)
`global_bc`	boundary conditions for the edges of the global domain **0** periodic 1 outflow
`sigma`	method for specifying sigma coordinates: **0** Use Glimmer's default spacing $\sigma_i = \frac{1-(x_i+1)^{-n}}{1-2^{-n}}$ with $x_i = \frac{\sigma_i - 1}{\sigma_n - 1}, n = 2.$ 1 use sigma coordinates defined in external file (named sigma_file) 2 use sigma coordinates given in configuration file 3 use evenly spaced sigma levels (required by the Glam dycore) 4 use Pattyn sigma levels

`[sigma]`	
Define the sigma levels used in the vertical discretization (`sigma`=2 above). This is an alternative to using a separate file (specified in section `[grid]` above). If neither is used, the levels are calculated as described above.	
`sigma_levels`	(real) list of sigma levels, in ascending order, separated by spaces. These run between 0.0 and 1.0.

`[time]`	
Configure time steps and diagnostic specifications	
`tstart`	(real) start time of the model in years
`tend`	(real) end time of the model in years

continued on next page

continued from previous page

`dt`	(real) size of time step in years
`subcyc`	(integer) number of time steps to subcycle evolution within dt using a steady velocity field
`ntem`	(real) time step multiplier setting the ice temperature update interval
`dt_diag`	(real) writing diagnostic variables to log file every dt_diag yrs
`idiag`	(int) x direction index for diagnostic grid point in log file
`jdiag`	(int) y direction index for diagnostic grid point in log file

`[options]`

Parameters set in this section determine how various components of the ice sheet model are treated. Configuration number options with a † are specific to the higher-order dycores (e.g., Glissade). Options with a ? are working, but are currently not scientifically supported, and are therefore for use at your own risk. Options with a ! are in development and will be supported in future code releases.

`dycore`	0	Glide (1-processor, 3d, shallow-ice-approximation dycore)
	1†?	Glam (parallel, 3d, FDM, 1st-order-accurate dycore)
	2†	Glissade (parallel, 3d, FEM, 1st-order-accurate dycore)
	3†!	FELIX (parallel, 3d, FEM, 1st-order-accurate dycore)
	4†!	BISICLES (parallel, quasi-3d, FVM, L1L2 dycore)
`evolution` (ice thickness)	**0**	pseudo-diffusion (Glide only)
	1	ADI scheme (Glide only)
	2	diffusion (Glide only)
	3†	incremental remapping
	4†	first-order upwind
	5†	evolve without changing ice thickness (Useful for running with a fixed geometry, e.g. for a temperature spinup. On each time step, geometry and tracers are evolved using incremental remapping, after which geometry is reset to its initial value. This evolution scheme is still subject to the advective CFL condition.)
`temperature`	0	Set each ice column to local surface air temperature
	1	prognostic temperature calculation
	2	hold temperature steady at initial value
	3!	prognostic temperature calculation using enthalpy-based formulation
`temp_init`	0	initial temperatures isothermal at 0°C
	1	initial column temperatures set to atmos. temperature
	2	initial column temperatures linearly interpolated between atmos. temperature and pressure melting point
`flow_law`	**0**	constant (using the value of `default_flwa`)
	1	temperature-dependent, Paterson and Budd (1982) ($T = -5°$C)
	2	temperature-dependent, Paterson and Budd (1982) (function of variable T)

continued on next page

continued from previous page

`basal_water`	**0**	none
	1	local water balance
	2?	compute the steady-state, routing-based, basal water flux and water layer thickness (NOTE: not supported for > 1 processor)
	3	use a constant basal water layer thickness everywhere, to enforce $T=T_{pmp}$ everywhere
	4!	ocean penetration parameterization for effective pressure from Leguy et al. (2014)
`basal_mass_balance`	**0**	ignore basal melt rate in mass balance calculation
	1	include basal melt rate in mass balance calculation
`slip_coeff`		slip coefficient (Glissade local SIA and Glide *only*)
	0	zero (no sliding)
	1	set to a non–zero constant everywhere
	2	set to constant where basal water (bwat) is nonzero
	3	set to constant where the ice base is melting
	4	set proportional to basal melt rate
	5	`tanh` function of basal water (bwat)
`marine_margin`	0	ignore marine margin
	1	set thickness to zero if floating
	2	lose fraction of ice from edge cells
	3	set thickness to zero if relaxed bedrock is below a given depth (variable “mlimit” in glide_types)
	4	set thickness to zero if present-day bedrock is below a given depth (variable “mlimit” in glide_types)
	5?	Huybrechts calving scheme
`vertical_integration`		(Glide *only*)
	0	standard integration (to obtain vertical velocity profile)
	1	constrained to obey kinematic velocity at upper surface boundary
`gthf`	**0**	prescribed, uniform geothermal heat flux
	1	read 2d geothermal heat flux field from input file
	2	calculate geothermal heat flux using 3d diffusion model
`isostasy`	**0**	no isostatic adjustment
	1	compute isostatic adjustment using lithosphere / asthenosphere model (see below for available options)
`topo_is_relaxed`	**0**	relaxed topography is read from a separate input variable, `relx`
	1	first time slice of input topography is assumed to be relaxed
	2	first time slice of input topography is assumed to be in isostatic equilibrium with ice thickness

continued on next page

continued from previous page

`restart`	*Note:* alternate keyword **hotstart** is retained for backwards compatibility. **0** normal start (initial values taken from input file or, if absent, using default options) 1 restart model using input from previous run; specific fields required for restart are dependent on chosen options (add "restart" to the `variable` list in the `[CF output]` section of the `.config` file to automatically save the appropriate restart fields.)
`ioparams`	(string) name of file containing netCDF I/O configuration. The main configuration file is searched for I/O related sections if no file name is given (default). In other words, you can remove sections `CF input`, `CF output`, and `CF forcing` from the primary configuration file and place them in a separate file, the path to which is specified here.

`[ho_options]`	
Options set in this section determine how various components of the higher-order extensions to the ice sheet model (e.g., Glissade) are treated. Defaults are indicated in bold. These options have no effect on the shallow-ice (Glide) dycore. In this section, options with a ? are working but are currently not scientifically supported (and are therefore for use at your own risk). Options marked with a * apply only to a serial build (or a parallel build if run on 1 processor). Options marked with a ! are under development and will be supported in future versions of the code (hence, these are also for use at your own risk).	
`which_ho_nonlinear`	**0** treat nonlinearity in momentum balance using Picard iteration 1? treat nonlinearity in momentum balance using Jacobian-Free Newton-Krylov iteration (Glam only)
`which_ho_sparse`	-1* solve sparse linear system using SLAP with incomplete Cholesky preconditioned conjugate gradient method 0* solve sparse linear system using SLAP with incomplete LU-preconditioned biconjugate gradient method 1* solve sparse linear system using SLAP with incomplete LU-preconditioned GMRES method 2 solve sparse linear system using preconditioned conjugate gradient method: standard algorithm (Glissade only) **3** solve sparse linear system using preconditioned conjugate gradient method: Chronopoulos-Gear algorithm (Glissade only) 4 solve sparse linear system using *Trilinos*, incomplete LU-preconditioned GMRES method (*Trilinos*-compatible build only)

continued on next page

continued from previous page

`which_ho_efvs`	0	use a constant value for the effective viscosity (i.e., linear viscosity). The default value is 2336041 Pa yr (as used by ISMIP-HOM Test F).
	1	set the effective viscosity to a value based on the flow rate factor: efvs $= 0.5 * A^{-1/n}$
	2	use the effective strain rate to compute the effective viscosity (i.e., full nonlinear treatment)
`which_ho_disp`	-1	no dissipation term included in temperature equation
	0	calculate dissipation in temperature equation assuming SIA ice dynamics
	1	calculate dissipation in temperature equation assuming first-order ice dynamics
`which_ho_babc`		Implementation of basal boundary condition in higher-order dycore
	0	constant value of “beta”
	1	specify a simple pattern for “beta” (hardcoded, mainly useful for debugging)
	2	read map of yield stress (in Pa) from input field “mintauf” to simulate sliding over a plastic subglacial till (Picard-based solution)
	3	calculate “beta” as linear (inverse) function of basal water thickness
	4	(virtually) no slip everywhere in domain (“beta” set to very large value)
	5	read map of “beta” from .nc input file using standard I/O
	6	no slip everywhere in domain (using Dirichlet basal BC)
	7!	read map of yield stress (in Pa) from input field “mintauf” to simulate sliding over a plastic subglacial till (Newton-based solution)
	8*	Spatial field of “beta” required for ISMIP-HOM Test C (avoids interpolation error associated with option 5; works for a single processor only)
	9!	Weertman-style power-law accounting for effective pressure (Eq. 5.33)
	10!	Coulomb friction law (Eq. 5.34)

continued on next page

continued from previous page

`which_ho_resid`	Residual calculation method for higher-order velocity solvers (e.g., Glissade). Nonlinear iterations are halted once the residual falls below a specified value. 0? use the maximum value of the normalized velocity vector update, defined by $r = \frac{\|vel_{k-1} - vel_k\|}{vel_k}$ 1? as in option 0 but omitting the basal velocities from the comparison (useful in cases where an approx. no slip basal BC is enforced) 2? as in option 0 but using the mean rather than the max **3** use the L2 norm of the system residual, defined by $r = Ax - b$ 4 use L2 norm of residual relative to rhs, $\|Ax - b\|/\|b\|$
`which_ho_approx`	Stokes-flow approximation to use with Glissade dycore -1 local shallow-ice approximation, Glide-type calculation (uses glissade_velo_sia) 0 3d matrix shallow-ice approximation, vertical-shear stresses only (uses glissade_velo_higher) 1 shallow-shelf approximation (SSA) with horizontal-plane stresses only (uses glissade_velo_higher; requires `which_ho_precond` <=1) **2** Blatter-Pattyn with both vertical-shear and horizontal-plane stresses (uses glissade_velo_higher) 3 depth-integrated (L1L2) approximation, with both vertical shear and horizontal-plane stresses (uses glissade_velo_higher; requires `which_ho_precond` <=1)
`which_ho_precond`	Preconditioner to use in the linear PCG solve of the Glissade dycore 0 no preconditioner 1 diagonal preconditioner **2** physics-based (shallow-ice) preconditioner (not valid for SSA and L1L2)
`which_ho_gradient`	Which spatial gradient operator to use in the Glissade dycore **0** centered gradient 1 upstream gradient (damps checkerboard noise in prognostic simulations)
`which_ho_gradient_margin`	Spatial gradient operator to use in the Glissade dycore at ice sheet margins. 0 use information from all neighboring cells, ice-covered or ice-free **1** use information from ice-covered and/or land cells, but not ice-free ocean cells 2 use information from ice-covered cells only

continued on next page

continued from previous page

`which_ho_assemble_beta`	Finite-element assembly method for basal boundary conditions that use “beta” field **0** Standard finite-element calculation, which effectively applies a smoothing to “beta” (and “mintauf”) 1 Apply the local “beta” (or “mintauf”) value at each vertex (no smoothing)
`glissade_maxiter`	**100** Maximum number of nonlinear (Picard) iterations in the Glissade dycore

`[parameters]`

Set values for various parameters. Parameters with a † are specific to the higher-order dycores (e.g., Glissade).

`log_level`	(integer) set to a value between 0, no messages, and 6, all messages are displayed to stdout. By default, messages are only logged to a file. The format for this filename is “configuration-file-name.config.log”
`ice_limit`	(real) below this limit ice is only accumulated/ablated; ice dynamics are switched on once the ice thickness is above this value. (default = 100.0 m)
`ice_limit_temp` †	(real) minimum thickness for computing vertical temperature (m). (default = 1.0 m)
`marine_limit`	(real) all ice is assumed lost (calved) once water depths reach this value (for `marine_margin`=3 or 4 in `[options]` above). Note, water depth is negative. (default = -200.0 m)
`calving_fraction`	(real) fraction of ice lost due to calving (for `marine_margin`=2). (default = 0.8)
`geothermal`	(real) constant geothermal heat flux, positive down by convention (hence < 0). (default = -0.05 W m^{-2})
`flow_factor`	(real) the flow law rate factor is multiplied by this factor (default = 1.0; in previous versions of Glimmer-CISM the default value was 3.0)
`default_flwa`	flow law parameter A to use in isothermal experiments (flow_law set to 0). Default value is 10^{-16} Pa^{-n} yr^{-1}. This overrides any temperature dependence.
`efvs_constant` †	Constant value of effective viscosity when using `which_ho_efvs`=0. Default value is 2336041 Pa yr, as in ISMIP-HOM Test F.
`basal_tract_const`	constant basal traction parameter. You can load a .nc file with a variable called `soft` if you want a spatially varying bed softness parameter (Glissade local SIA and Glide only)
`basal_tract_max`	max value for basal traction when using `slip_coeff`=4.
`basal_tract_slope`	slope value for basal traction relation when using `slip_coeff`=4. (Relation also uses `basal_tract_const`.)
`basal_tract_tanh`	(real(5)) basal traction factors. Basal traction is set to $B = \tanh(W)$ with the parameters (1) width of the tanh curve (2) W at midpoint of tanh curve [m] (3) B minimum [ma^{-1}Pa^{-1}] (4) B maximum [ma^{-1}Pa^{-1}] (5) multiplier for marine sediments

continued on next page

continued from previous page

`ho_beta_const` †	(real) spatially uniform beta used when `which_ho_babc` = 0. (default = 10.0 Pa yr m^{-1})
`friction_powerlaw_k` †	(real) friction coefficient k used for `which_ho_babc` = 9 (Eq. 5.33) (default = 8.4e-9 m y^{-1} Pa^{-2}, from Bindschadler (1983) converted to CISM units)
`coulomb_c` †	(real) Coulomb friction coefficient (unitless), C, used for `which_ho_babc` = 10 (Eq. 5.34) (default = 0.42, from Pimentel et al. (2010))
`coulomb_bump_wavelength` †	(real) wavelength (m) of the dominant bedrock bumps, λ_{max}, used for `which_ho_babc` = 10 (Eq. 5.34) (default = 2.0 m, from Pimentel et al. (2010))
`coulomb_bump_slope` †	(real) maximum slope (unitless) of the dominant bedrock bumps, m_{max}, used for `which_ho_babc` = 10 (Eq. 5.34) (default = 0.5 m, from Pimentel et al. (2010))
`p_ocean_penetration`	(real) p-exponent in ocean penetration parameterization for (`basal_water` = 4 (default = 0.0)
`periodic_offset_ew`†	(real) vertical offset between east and west edges of the global domain. (default = 0.0 m) (Primarily used for ISMIP-HOM and Stream test cases.)
`periodic_offset_ns`†	(real) vertical offset between north and south edges of the global domain. (default = 0.0 m) (Primarily used for ISMIP-HOM and Stream test cases.)

`[GTHF]`

Options related to lithospheric temperature and geothermal heat calculation. Ignored unless `gthf` = 1.

`num_dim`	can be either `1` for 1D calculations or 3 for 3D calculations.
`nlayer`	number of vertical layers (default: 20).
`surft`	initial surface temperature (default 2°C).
`rock_base`	depth below sea-level at which geothermal heat gradient is applied (default: -5000m).
`numt`	number time steps for spinning up GTHF calculations (default: 0).
`rho`	The density of lithosphere (default: 3300kg m^{-3}).
`shc`	specific heat capcity of lithosphere (default: 1000J kg^{-1} K^{-1}).
`con`	thermal conductivity of lithosphere (3.3 W m^{-1} K^{-1}).

`[isostasy]`

Options related to isostasy model. Ignored unless `isostasy` = 1. Options marked with a * work only with a serial build (or a parallel build if run on 1 processor).

`lithosphere`	**0** local lithosphere, equilibrium bedrock depression is found using Archimedes' principle 1* elastic lithosphere, flexural rigidity is taken into account
`asthenosphere`	**0** fluid mantle, isostatic adjustment happens instantaneously 1 relaxing mantle, mantle is approximated by a half-space
`relaxed_tau`	characteristic time constant of relaxing mantle (default: 4000.a)
`update`	lithosphere update period (default: 500.a)
`flexural_rigidity`	flexural rigidity of the lithosphere (default: 0.24e25 Pa m^3)

continued on next page

continued from previous page

[projection]	
Specify map projection for reference. The reader is referred to Snyder J.P. (1987) *Map Projections - a working manual.* USGS Professional Paper 1395.	
type	string that specifies the projection type (LAEA, AEA, LCC or STERE).
centre_longitude	central longitude in degrees east
centre_latitude	central latitude in degrees north
false_easting	false easting in meters
false_northing	false northing in meters
standard_parallel	location of standard parallel(s) in degrees north. Up to two standard parallels may be specified (depending on the projection).
scale_factor	non-dimensional; relevant only for the stereographic projection

7.4 Input/Output Configuration

NetCDF I/O can be configured in the main configuration file or in a separate file (see ioparams in the [options] section description above). Any number of input, forcing, and output files can be specified.

Input files are processed in the same order they appear in the configuration file, thus potentially overwriting previously loaded fields. The configuration section for an input file specifies which time slice from the input file should be used as the initial condition. (This is an integer specifying the time level, not the actual time.)

Input files can contain any of a large number of model fields that are specified as being "loadable". These fields are marked with asterisks in the netCDF model variable table in Appendix A. The option for a given variable to be "loadable" or not can be changed within the *_vars.def files (most commonly, glide_vars.def), as described in Appendix B.1. The standard test cases discussed in Chapter 8 give examples of variables that might be specified for standard model setups.

Forcing files are new in CISM 2.0. These are input files that are read on every time step to allow time-dependent forcing to be applied during a simulation. Any input fields specified in *_vars.def (again, most commonly, glide_vars.def) can be included in forcing files. Forcing files should have a "time" field which is used to assign values to each field in the file during the simulation. Forcing is applied in a piecewise constant fashion; the most recent time slice in the forcing file prior to the current model time is used on each time step. (Linear interpolatation of forcing may be available in the future but is not yet implemented.) If a field is present in both an input file and a forcing file at the start time, the value in the forcing file will overwrite the value from the input file because forcing files are read after input files. Forcing files are processed in the same order they appear in the configuration file on each time step, thus potentially overwriting previously loaded fields from other forcing files. The "dome" test case (tests/higher-order/dome) includes an optional setup of how to implement time-dependent forcing.

One special input field is the kinbcmask field, which is used to specify locations (i.e., indices in the horizontal grid plane) on the staggered (velocity) grid where Dirichlet boundary conditions are to be applied. This input field is used in conjunction with the uvel and vvel fields. At any location where kinbcmask = 1, the input field values of uvel and vvel for that same column will be applied as Dirichlet boundary conditions on the velocity solution. If uvel and vvel are not included in the input file, zero velocity will be specified throughout the column for any location where kinbcmask is set to 1. An example of this application can be seen in the confined shelf test case described in Chapter 8.2. The construction of the relevant input kinbcmask field is done by the python script that constructs the other input fields for this test case (the shelf-confined.py script). Also, the dome test case described in Chapter 8.2 includes a script and a config file demonstrating how kinbcmask, uvel, and vvel can be used in a forcing file to provide time-varying Dirichlet boundary conditions.

The following tables describe netCDF sections and parameters in the .config file.

`[CF default]`	
This section contains metadata describing the experiment. Any of these parameters can be optionally included in the `[CF output]` section for a specific output file, overriding the default values specified here.	
`title`	Title of the experiment
`institution`	Institution at which the experiment was run
`references`	References that might be useful
`comment`	Comments further describing the experiment

`[CF input]`	
Any number of input files can be specified in separate `[CF input]` sections. They are processed in the order they appear in the configuration file, potentially overwriting previously loaded variables with the same names from previous input files.	
`name`	Name of the netCDF file to be read. Typically, netCDF files end with `.nc`.
`time`	Time slice (not actual time) to be read from the netCDF file. The first time slice is read by default.

`[CF forcing]`	
Any number of forcing files can be specified in separate `[CF forcing]` sections. They are processed in the order they appear in the configuration file, potentially overwriting previously loaded variables. Each forcing file needs a "time" dimension and variable that indicates the model time associated with each time slice in the file.	
`name`	Name of the netCDF file to be read. Typically, netCDF files end with `.nc`.

`[CF output]`	
This section controls how often selected variables are written to files.	
`name`	Name of the output netCDF file. Typically, netCDF files end with `.nc`.
`start`	(real) Start writing to file when this time (years) is reached (default: first time slice).
`stop`	(real) Stop writing to file when this time (years) is reached (default: last time slice).
`frequency`	(real) The time interval in years, determining how often selected variables are written to file.
`xtype`	Set the floating point representation used in netCDF file. `xtype` can be one of `real`, `double` (default: `real`). (If the name `restart` is included in the list of `variables`, `xtype` is automatically set to `double` to ensure bit-reproducible restarts, overriding the value set here.)
`variables`	List of variables to be written to file. See Appendix A for a list of possible variables. Names should be separated by at least one space. The variable names are case-sensitive. The name `restart` selects all variables necessary for a restart based on the specified model configuration. (The name `hot` is also retained for this purpose for backwards compatability.) If the name `restart` is included, the `xtype` option is automatically set to `double` to ensure bit-reproducible restarts.

continued on next page

continued from previous page

7.5 Climate Forcing Configuration

The core ice sheet model is connected to the climate via the surface mass balance (`acab` field) and air temperature (`artm` field) and (optionally) a scalar value for eustatic sea level. This climate forcing can come from:

- climate schemes included in the CISM code:
  - EISMINT 1 and 2
  - annual and daily PDD schemes
- data input into the model:
  - directly as user-supplied `acab` and `artm` fields
  - from a global or regional climate model (e.g., CESM, GENIE, either from data or coupled) throught the Glint interface

7.5.1 EISMINT climate forcing

Like previous versions of Glimmer, CISM includes a set of idealized climate forcings used in the European Ice Sheet Modeling INiTiative (EISMINT) Phase 1 and 2 series of experiments. These forcings consist of surface mass balance and air temperature fields for predefined experiments. See Chapter 8 for details of how to run the individual tests, and see the EISMINT publications for a more detailed description of the tests and the forcings associated with each. Huybrechts et al. (1996) describe EISMINT Phase 1, and Payne et al. (2000) describe EISMINT Phase 2.

Configuration

The various EISMINT climate forcings are enabled by adding one of the following sections to the configuration file used for running CISM. See the files associated with the EISMINT test cases (Chapter 8) for examples of their use for each of the specific experiment setups described in Huybrechts et al. (1996) and Payne et al. (2000).

`[EISMINT-1 fixed margin]`	
EISMINT 1 fixed margin scenario. Some of the EISMINT-1 fixed margin tests use periodic, time-varying forcing.	
`temperature`	(real(2 values)) Temperature forcing $$T_{\text{surface}} = t_1 + t_2 d$$ where $$d = \max\{|x - x_{\text{summit}}|, |y - y_{\text{summit}}|\}$$

continued on next page

continued from previous page

`massbalance`	(real) Mass balance forcing
`period`	(real) Period of time–dependent forcing (switched off when set to 0) $$\Delta T = 10 \sin \frac{2\pi t}{T}$$ and $$\Delta M = 0.2 sin \frac{2\pi t}{T}$$
`mb_amplitude`	(real) Amplitude of the surface mass balance when `period` > 0

`[EISMINT-1 moving margin]`	
EISMINT 1 moving margin scenario. Some of the EISMINT-1 moving margin tests use periodic, time-varying forcing.	
`temperature`	(real(2 values)) Temperature forcing $$T_{\text{surface}} = t_1 - t_2 H$$ where H is the ice thickness
`massbalance`	(real(3 values)) Mass balance forcing $$M = \min\{m_1, m_2(m_3 - d)\}$$ where $$d = \sqrt{(x - x_{\text{summit}})^2 + (y - y_{\text{summit}})^2}$$
`period`	(real) Period of time–dependent forcing (switched off when set to 0) $$\Delta T = 10 \sin \frac{2\pi t}{T}$$ and $$M = \min\left\{m_1, m_2\left(m_3 + 100 sin \frac{2\pi t}{T} - d\right)\right\}$$
`mb_amplitude`	(real) Amplitude of the surface mass balance when `period` > 0

`[EISMINT-2]`	
EISMINT 2 climate forcing. Both surface mass balance and air temperature depend solely on position in the map plane and not on ice-surface elevation.	
`temperature`	(real(2 values)) Temperature forcing $$T_{\text{surface}} = t_1 - t_2 d$$ where d is the distance from the summit, $$d = \sqrt{(x - x_{\text{summit}})^2 + (y - y_{\text{summit}})^2}$$

continued on next page

continued from previous page	
`massbalance`	(real(3 values)) Mass balance forcing $$M = \min\{m_1, m_2(m_3 - d)\}$$ where d is the distance from the summit, $$d = \sqrt{(x - x_{\text{summit}})^2 + (y - y_{\text{summit}})^2}$$

7.5.2 Glint driver

Glint (originally an acronym for "Glimmer interface") allows CISM to be run with forcing from an external climate model or global data sets. Glint was originally developed as an interface between Glide and the GENIE Earth-system model, but is designed to be flexible enough to be used with a wide range of global climate models. In older versions of Glint, it was assumed that forcing from the climate model would include the temperature and precipitation fields required to force a positive-degree-day (PDD) scheme for the surface mass balance (SMB). In CISM2, PDD forcing is still supported, but Glint can also receive the SMB (typically computed in multiple elevation classes on the relatively coarse grid of a climate model) and downscale it directly to the ice sheet model grid.

A distinctive feature of Glint is the way it uses the object-oriented Glide architecture to enable multiple ice models to be coupled to the same climate model. This means that regional ice models can potentially run at high resolution over several parts of the globe (e.g., Greenland and Antarctica), without the expense of running a global ice sheet model.

Glint automates the processes required in coupling regional models to a global model, particularly the downscaling and upscaling of the fields that form the interface between the two models. The user may specify map projection parameters for each of the ice sheet models (known as *instances*). The different time steps of the global model, mass-balance scheme, and ice sheet model are handled automatically by temporal averaging or accumulation of quantities as appropriate.

Prerequisites

Glint users should bear the following in mind:

- Global input fields must be supplied on a latitude-longitude grid. The grid does not have to be uniform in latitude, meaning that Gaussian grids may be used. Irregular grids (e.g., icosahedral grids) are not currently supported. The boundaries of the grid boxes may be specified; if not, they are assumed to lie halfway between the grid points in lat-lon space.

- In the global field arrays, latitude must be indexed from north to south. That is, the first row of the array is the northernmost one. (Some flexibility might be introduced here in the future.)

- The global grid must not have grid points at either of the poles. This restriction is not expected to be permanent, and in the meantime can probably be overcome by moving the location of the polar points to be fractionally short of the pole (e.g. at 89.9° and -89.9°).

Initializing and calling

The easiest way to learn how Glint is used is by way of an example. Glint should be built automatically as part of CISM, and we assume here that this has been done successfully.

Typically, Glint will be called from the main program body of a climate model. (In CESM, Glint subroutines are called from a wrapper layer called **glc**.) To make this possible, the compiler needs to be told to use the Glint code, with "use" statements like the following:

```
use glint_main
```

The next task is to declare a variable of type `glint_params`, which holds everything relating to the model, including any number of ice-sheet instances:

```
type(glint_params) :: ice_sheet
```

Before the ice-sheet model may be called from the climate model, it must be initialized. This can be done with the following subroutine call[3]:

```
call initialise_glint(ice_sheet,lats,lons,time_step,paramfile)
```

These are the required arguments; many optional arguments are also possible. The required arguments are defined as follows:

- `ice_sheet` is the variable of type `glint_params` defined above;
- `lats` and `lons` are one-dimensional arrays giving the locations of the global grid-points in latitude and longitude, respectively;
- `time_step` is the intended interval between calls to Glint, in hours. This is known as the *forcing timestep.*
- `paramfile` is the name of the Glint configuration file. The contents of this file are discussed below.

If Glint is to be forced with an externally computed surface mass balance (rather than the fields that would drive a PDD scheme), the climate model should instead call the subroutine `initialise_glint_gcm`, which has the same required arguments but a different set of optional arguments.

After Glint is initialised, it may be called as part of the main climate model time-step loop:

```
  call glint(ice_sheet,time,temp,precip,orog)
```

where

- `ice_sheet` is the variable of type `glint_params` defined above;
- `time` is the current model time, in hours;
- `temp` is the daily mean 2 m global air temperature field, in °C;
- `precip` is the global daily accumulated precipitation field, in mm (water equivalent, making no distinction between rain, snow, etc.);
- `orog` is the global orography field, in m.

Many optional arguments may also be specified.

The latter three compulsory fields are needed to drive a PDD scheme. Glint includes two such schemes. One of these calculates the mass-balance for the whole year (the *annual PDD scheme*), while the other calculates on a daily basis (the *daily PDD scheme*). The annual scheme incorporates a stochastic temperature variation to account for diurnal and other variations, which means that if this scheme is used, Glint should be called such that short-term variations have been removed. In practice, this means calling Glint on a monthly basis, with monthly mean temperatures. For the daily scheme, no such restriction exists, and the scheme should be called at least every 6 hours.

If the SMB is computed externally by a climate model, the call to Glint would resemble this one:

[3]The spelling of some subroutine names reflects the British origins of the code.

```
    call glint_gcm(ice_sheet,time,qsmb,tsrf,topo)
```

where the last three arguments (all compulsory) are defined as follows:

- qsmb is the surface mass balance in kg m^{-2} s^{-1};
- tsrf is the surface ground temperature in $^{\circ}C$, used as an upper thermal boundary condition for the ice sheet; and
- topo is the surface elevation in m.

These fields contain two horizontal dimensions, along with a third dimension for elevation class. CESM typically computes these quantities in 10 elevation classes per glaciated grid cell in the climate model.

Among the optional arguments are the following output fields (returned by Glint to the climate model):

- gfrac is the fractional grid cell area covered by ice;
- gtopo is the mean surface elevation in m;
- ghflx is the upwelling heat flux at the ice sheet surface, in W m^2;
- grofi is the solid runoff flux (i.e., the calving flux) in kg m^{-2} s^{-1};
- grofl is the liquid runoff flux in kg m^{-2} s^{-1}. This includes basal and possibly internal melting, but not surface melting (which has already been computed by the climate model).

These fields are required by CESM to update its land-surface types and topography and to conserve heat and water when CISM is coupled interactively to the climate model. The first three fields are computed for each elevation class in each climate model grid cell, and the two runoff fluxes are averaged over each grid cell. The initial "g" denotes that these fields have been upscaled to the global climate grid.

Finishing off

After the desired number of time steps have been run, Glint may have some tidying up to do. To do this, the subroutine end_glint must be called:

```
  call end_glint(ice_sheet)
```

Configuration

Glint uses the same configuration file format as the rest of CISM. If there is only one ice sheet instance, all the configuration data for Glint and Glide (or Glissade) can reside in the same file. If two or more instances are used, a top-level file specifies the number of model instances and the name of a configuration file for each one. Configuration sections specific to Glint are as follows:

[Glint]	
Section specifying number of instances.	
n_instance	(integer) Number of instances (default=1)
[Glint instance]	
Specifies the name of an instance-specific configuration file. Unnecessary if we only have one instance whose configuration data is in the main config file.	

continued on next page

continued from previous page	
`name`	Name of instance-specific config file (required).
`[Glint climate]`	
Glint climate configuration	
`evolve_ice`	specify whether or not the ice sheet evolves in time: 0 Do not evolve ice sheet (hold fixed in time). This setting is appropriate if we want to analyze the SMB downscaled to the observed ice sheet geometry, without the complexity of evolution. **1** Allow the ice sheet to evolve
`precip_mode`	Method of precipitation downscaling: **1** Use large-scale precipitation rate 2 Use parameterization of *Roe and Lindzen*
`acab_mode`	Mass-balance model to use: 0 The surface mass balance is computed externally (e.g., by a climate model) **1** Annual PDD mass-balance model (Section 7.7.1) 2 Annual accumulation only 3 Hourly energy-balance model (NOTE: not supported) 4 Daily PDD mass-balance model (Section 7.7.2)
`ice_albedo`	Albedo of ice—may be used for coupling to climate model (default=0.4)
`lapse_rate`	Atmospheric temperature lapse-rate, used to correct the atmospheric temperature onto the ice model orography. This should be *positive* for temperature falling with height ($\mathrm{K\,km^{-1}}$) (default=8.0)
`data_lapse_rate`	Atmospheric temperature vertical lapse rate, to be used in the calculation of temperature at sea level. The variable `lapse_rate` is then used to adjust the temperature to the surface of the local ice sheet topography. If `data_lapse_rate` is not set, it is set to the value of `lapse_rate` by default.
`ice_tstep_multiply`	Ice time-step multiplier: allows asynchronous climate-ice coupling. See below for full explanation of Glint time-stepping. (default = 1)
`mbal_accum_time`	Mass-balance accumulation time (in years, default is equal to mass-balance timestep). See below for full explanation of Glint time-stepping.

Glint timestepping — an explanation

By default, the model accepts input on each forcing timestep (as specified in the call to `initialise_glint`). Input fields are accumulated over the course of a mass-balance time-step, whereupon the mass-balance model is called. The output from the mass-balance model is accumulated over the course of an ice sheet model time-step, and finally the ice sheet model (Glide or Glissade) is called.

This default behaviour can be altered, in two ways:

1. The number of ice sheet time steps executed for each accumulated mass-balance field may be increased—thus accelerating the ice sheet relative to the forcing. To do this, set `ice_tstep_multiply` in the `[Glint climate]` config section; this must be an integer. This acceleration is possible only if the mass balance is accumulated over an integer number of years.

2. The mass-balance accumulation period can be altered by setting `mbal_accum_time` in the `[Glint climate]` config section; this is a floating-point value in years.

The interaction of these two parameters is fairly complex, and permits sophisticated control of how the ice sheet model is forced. Various checks are made at run-time to make sure that sensible values are selected. Most importantly, all relevant time-steps must divide into one another appropriately. The model will (or should) stop if an un-sensible combination of values is detected.

Glint timestepping — further examples

To aid understanding of the time-stepping controls, here are some examples. First, suppose we have these time-step values:

forcing time-step:	6 hours
mass-balance time-step:	1 day
ice time-step:	0.5 year

By default, the model will accumulate 6 months of mass-balance calculations, and force the ice sheet model based on the 6-month average. This might not be desirable, so you could set:

```
mbal_accum_time = 1.0
```

This would make Glint accumulate 1 year of mass-balance output before forcing the ice sheet (at which point it would execute *two* ice sheet time-steps of 0.5 years each).

Having done that, you could accelerate the ice model by a factor of ten, by setting

```
ice_tstep_multiply = 10
```

In this scenario, 20 ice sheet time-steps of 0.5 years each would be done after each 12-month accumulation of mass-balance data.

For the second example, we consider the contrasting situation where we do not want to calculate a mass balance on all the available data (perhaps to save time). Consider these time-step values:

forcing time-step:	6 hours
mass-balance time-step:	1 day
ice time-step:	10 years

(Clearly this is a numerically stable and/or low-resolution ice sheet). To avoid running the daily PDD scheme c.3600 times (depending on the value of `days_in_year`), we can choose to use only the first two years of data:

```
mbal_accum_time = 2.0
```

Glint accumulates the mass balance for 2 years, then waits for 8 years (ignoring incoming data during this time) before calling the ice sheet. Ice sheet acceleration may be enabled with `ice_tstep_multiply` as before.

7.6 Glint: Using glint-example

To run CISM using the Glint climate driver, go to directory `tests/glint_example` and see the `README.md` file. This file gives directions for downloading a tar file, `glint-example.1.0.0.tar.gz`, that contains three data files used to force Glint:

- `ncep-doe_6h_climate.64x32.nc` consists of 6-hourly precipitation and surface temperature data on a coarse global grid;
- `orog.igcmgrid.64x32.nc` consists of surface elevation data on a global grid (these data are used to downscale temperature as a function of elevation); and

- `gland20.input.nc` includes thickness and topography data for the Greenland ice sheet on a 20-km mesh.

These files should be placed in directory `tests/glint_example`. This directory already includes the following config files:

- `greenland_20km.config.pdd` and `greenland_20km.config.smb` are ice sheet configuration files for PDD and SMB forcing, respectively;
- `glint_example.config.pdd` and `glint_example.config.smb` are the corresponding climate configuration files.

To apply the PDD scheme, type

```
cism_driver greenland_20km.config.pdd glint-example.config.pdd
```

and to apply the SMB scheme, type

```
cism_driver greenland_20km.config.smb glint-example.config.smb
```

The 6-hourly temperature and precipitation data are tailored for a PDD scheme. The same data, however, can also supply a crude surface mass balance and surface temperature in multiple elevation classes, imitating the forcing from a climate model such as CESM. (Accumulation is set equal to precipitation, and ablation is assumed to be linearly related to the downscaled temperature. This scheme is convenient for testing but obviously is not appropriate for science.) By default, two files are output at a specified frequency: a history file containing variables of interest (e.g., thickness, velocity, and SMB) is output every 10 years, and a "hot" file containing the variables required for exact restart is output every 1000 years.

7.7 Supplied mass-balance schemes

Users are free to supply their own mass-balance model for use with Glide. However, Glint includes two positive-degree-day models for mass balance, one annual and one daily. This section describes how to configure and call these models.

NOTE: CISM2 development has focused on CESM-style coupling where the SMB is computed externally. The two PDD schemes (and the following description) are identical to those in the original Glimmer code, but the schemes have not been extensively tested. Users should proceed with caution.

7.7.1 Annual PDD scheme

The annual PDD scheme is contained in the f90 module `glimmer_pdd`, and the model parameters are contained in the derived type `glimmer_pdd_params`. Configuration data is contained in a standard CISM config file, which needs to be read before initializing the mass-balance model. The model is initialized by calling the subroutine `glimmer_pdd_init`, and the mass-balance may be calculated annually by calling `glimmer_pdd_mbal`.

Example of use:

```
  use glimmer_pdd
  use glimmer_config

  ...

  type(glimmer_pdd_params) :: pdd_scheme
  type(ConfigSection),pointer :: config
```

```
...

call glimmer_pdd_init(pdd_scheme,config)

...

call glimmer_pdd_mbal(pdd_scheme,artm,arng,prcp,ablt,acab)
```

In the subroutine call to `glimmer_pdd_mbal`, apart from the parameter variable `pdd_scheme`, there are three input fields (`artm`, `arng` and `prcp`), which are, respectively, the annual mean air temperature, annual temperature half-range, and annual accumulated precipitation fields. The final two arguments are output fields — annual ablation (`ablt`) and annual mass-balance (`acab`). Temperatures are in degrees Celsius, and precipitation, ablation and mass-balance are measured in m of water equivalent.

Degree-day calculation

The greater part of the information in the `glimmer_pdd_params` derived type comprises a look-up table (the *PDD table*). The model is implemented this way for computational efficiency.

The table has two dimensions: mean annual air temperature (T_a) (as the second index) and annual air temperature half range (i.e., from July's mean to the annual mean ΔT_a) (as the first index). Following *Huybrechts and others* [1991], daily air temperatures (T_a') are assumed to follow a sinusoidal cycle

$$T_a' = T_a + \Delta T_a \cos\left(\frac{2\pi t}{A}\right) + \mathbf{R}(0, \sigma) \tag{7.1}$$

where A is the period of a year and R is a random fluctuation drawn from a normal distribution with mean 0 °C and standard deviation σ °C. *Huybrechts and others* [1991] indicate that the number of positive degree days (D, °C days) for this temperature series can be evaluated as

$$D = \frac{1}{\sigma\sqrt{2\pi}} \int_0^A \int_0^{T_a'+2.5\sigma} T_a \times \exp\left(\frac{-(T_a - T_a')^2}{2\sigma^2}\right) dT dt \tag{7.2}$$

where t is time. The table is completed by evaluating this integral using a public-domain algorithm (Romberg integration) by *Bauer* [1961]. The inner and outer integrals are coded as two subroutines (`inner_integral` and `pdd_integrand`), which call the Romburg integration recursively.

The main parameter needed is the assumed standard deviation of daily air temperatures, which can be set in the configuration file (the default is 5 °C).

The positive-degree days are then looked up in the table (as a function of T_a and ΔT_a). We take care to check that this look up is in done within the bounds of the table. The final value of P is determined using bilinear interpolation given the four nearest entries in the table to the actual values of T_a and ΔT_a.

The remainder of the loop completes the calculation of the ablation and accumulation given this value for P.

Mass balance calculation

We use the following symbols: a is total annual ablation; a_s is potential snow ablation; b_0 is the capacity of the snowpack to hold meltwater by refreezing; the total number of positive degree days (D); degree-day factors for snow and ice (f_s and f_i); and the fraction of snowfall that can be held in the snowpack as refrozen meltwater ($W_{\max}$). The degree-day factors have been converted from ice to water equivalents using the ratio of densities.

First, determine the depth of superimposed ice (b_0) that would have to be formed before runoff (mass loss) occurs as a constant fraction ($W_{\max}$) of precipitation (P):

$$b_0 = W_{\max} P. \tag{7.3}$$

Now determine the amount of snow melt by applying a constant degree-day factor for snow to the number of positive degree-days:

$$a_s = f_s D. \tag{7.4}$$

We now compare the potential amount of snow ablation with the ability of the snow layer to absorb the melt. Three cases are possible. First, all snow melt is held within the snowpack and no runoff occurs ($a = 0$). Second, the ability of the snowpack to hold meltwater is exceeded but the potential snow ablation is still less than the total amount of precipitation so that $a = a_s - b_0$. Finally, the potential snow melt is greater than the precipitation (amount of snow available), so that ice melt (a_i) has to be considered as well. The total ablation is therefore the sum of snow melt (total precipitation minus meltwater held in refreezing) and ice melt (from the total number of degree-days, deduct the number of degree-days needed to melt all snowfall and convert to ice melt):

$$a = a_s + a_i = P - b_0 + f_i \left(D - \frac{P}{f_s} \right). \tag{7.5}$$

We now have a total annual ablation, and can find total net mass balance as the difference between the annual precipitation and the annual ablation.

This methodology is fairly standard and stems from a series of Greenland papers by Huybrechts, Letreguilly and Reeh in the early 1990s.

Configuration

The annual PDD scheme is configured using a single section in the configuration file:

`[GLIMMER annual pdd]`	
Specifies parameters for the PDD table and mass-balance calculation	
`dx`	Table spacing in the x-direction (°C) (default=1.0)
`dy`	Table spacing in the y-direction (°C) (default=1.0)
`ix`	Lower bound of x-axis (°C) (default=0.0)
`iy`	Lower bound of y-axis (°C) (default=-50.0)
`nx`	Number of values in x-direction (default=31)
`ny`	Number of values in x-direction (default=71)
`wmax`	Fraction of melted snow that refreezes (default=0.6)
`pddfac_ice`	PDD factor for ice (m day^{-1} °C^{-1}) (default=0.008)
`pddfac_snow`	PDD factor for snow (m day^{-1} °C^{-1}) (default=0.003)

References

Bauer (1961) *Comm. ACM* **4**, 255.

Huybrechts, Letreguilly and Reeh (1991) *Palaeogeography, Palaeoclimatology, Palaeoecology (Global and Planetary Change)* **89**, 399-412.

Letreguilly, Reeh and Huybrechts (1991) *Palaeogeography, Palaeoclimatology, Palaeoecology (Global and Planetary Change)* **90**, 385-394.

Letreguilly, Huybrechts and Reeh (1991) *Journal of Glaciology* **37**, 149-157.

7.7.2 Daily PDD scheme

The other PDD scheme is a daily scheme. This is simpler than the annual scheme in that it does not incorporate any stochastic variations. The mass balance is calculated on a daily basis, given the daily mean temperature and half-range, and assuming a sinusoidal diurnal cycle. Consequently, the firn model is more sophisticated than with the annual scheme, and includes a snow-densification parameterization.

Configuration

The daily PDD scheme is configured using a single section in the configuration file:

`[GLIMMER daily pdd]`	
Specifies parameters for the PDD table and mass-balance calculation	
`wmax`	Fraction of melted snow that refreezes (default=0.6)
`pddfac_ice`	PDD factor for ice (m day^{-1} °C^{-1}) (default=0.008)
`pddfac_snow`	PDD factor for snow (m day^{-1} °C^{-1}) (default=0.003)
`rain_threshold`	Temperature above which precipitation is held to be rain (°C) (default=1.0)
`whichrain`	Which method to use to partition precipitation into rain and snow: **1** Use sinusoidal diurnal temperature variation 2 Use mean temperature only
`tau0`	Snow densification timescale (s) (default=10 years)
`constC`	Snow density profile factor C (m^{-1}) (default=0.0165)
`firnbound`	Ice-firn boundary as fraction of density of ice (default=0.872)
`snowdensity`	Density of fresh snow (kg m^{-3}) (default=300.0)

Chapter 8

Test Cases

Test cases for CISM include experiments with analytic solutions, standardized experiments without analytic solutions but for which community benchmarks are available, and some experiments specific to CISM which have been well characterized by CISM developers.

Here we organize test cases based on the velocity solver that is most appropriate for each test. Any velocity solver can be used with any test if the .config file settings are adjusted manually. In some cases, however, the results may be difficult to interpret.

Each test directory includes a README.md file with some technical details about how to run the test. Many tests have python scripts that are used to set up the initial condition and, in some cases, execute the model. Some tests have an additional python script for analyzing the CISM output.

The user must manually provide each test with access to the CISM executable. There are several ways to do this:

- Softlink the executable into the directory, e.g.:

  ```
  ln -s ../../../builds/mac-gnu/cism_driver/cism_driver ./
  ```

  This is the recommended procedure during development so that the test will always be using the most up-to-date version of the executable.

- Use the -e command line option to point to include an explicit path for the executable (for test case run scripts that support this option), e.g.:

  ```
  ./runDome.py -e ../../../builds/mac-gnu/cism_driver/cism_driver
  ```

  This is useful for quickly trying a different version of CISM (e.g., comparing serial and parallel executables).

- Add the directory containing the CISM executable to your environment PATH.

- Copy the executable into the directory. This is typically not the most efficient approach, but may make sense in some situations.

The python scripts generally have useful command line options that control their execution. Typically, you can see details by using the `--help` (or `-h`) command line option, e.g.:

```
./runDome.py --help
```

The various tests are described below.

8.1 Shallow-Ice Test Cases

These tests are primarily useful for testing the shallow-ice approximation (SIA) dynamical core, Glide (see Chapter 4). The Glissade dycore also has a shallow-ice option (see Section 6.2.2).

8.1.1 Halfar dome

The Halfar test case describes the time evolution of a parabolic dome of ice, as described by Halfar (1983). For a flat-bedded SIA problem, this case has an analytic solution for the time varying ice thickness. We start with the general SIA ice evolution equation,

$$\frac{\partial H}{\partial t} = \nabla \cdot (\Gamma H^{n+2} |\nabla H|^{n-1} \nabla H), \tag{8.1}$$

where n is the exponent in the Glen flow law, commonly taken as 3, and Γ is a positive constant:

$$\Gamma = \frac{2}{n+2} A(\rho g)^n. \tag{8.2}$$

For $n = 3$, the time-dependent solution is

$$H(t,r) = H_0 \left(\frac{t_0}{t}\right)^{\frac{1}{9}} \left[1 - \left(\left(\frac{t_0}{t}\right)^{\frac{1}{18}} \frac{r}{R_0}\right)^{\frac{4}{3}}\right]^{\frac{3}{7}}, \tag{8.3}$$

where

$$t_0 = \frac{1}{18\Gamma} \left(\frac{7}{4}\right)^3 \frac{R_0^4}{H_0^7}, \tag{8.4}$$

and H_0, R_0 are the central height of the dome and its radius at time $t = t_0$. For more details, see Halfar (1983), Bueler et al. (2005), and this link[1].

Provided files

Our implementation of the Halfar dome test has an initial radius of $R_0 = 21.2$ km and an initial thickness of $H = 707.1$ m. These values can be changed by editing the `halfarDome` function in `runHalfar.py`.

- README.md
  Information about the test case, including technical details about running it.

- halfar.config
  This is the config file defining CISM options. It is set up to run Glide.

- halfar-HO.config
  This alternative config file is set up to run Glissade using the Blatter-Pattyn approximation of Stokes flow. By manually setting `which_ho_approx` under `[ho_options]`, users can choose other approximations (see Section 7.3).

- runHalfar.py
  This python script generates the dome initial condition and runs CISM.

- halfar_results.py
  This script compares model results to the analytic solution.

[1] http://www.projects.science.uu.nl/iceclimate/karthaus/2009/more/lecturenotes/EdBueler.pdf

Running the test

One script sets up the initial condition and runs the model:

```
./runHalfar.py
```

Note that to run the test with the `halfar-HO.config` settings, you can use the `-c` commandline option for specifying a configuration file:

```
./runHalfar.py -c halfar-HO.config
```

Another script analyzes and plots the results:

```
./halfar_results.py
```

Results

With the default .config settings, this simulation should only take a few seconds and is a good first test for a working Glide dycore. With Glissade, the Blatter-Pattyn option takes a few minutes, but the SIA and L1L2 settings are much faster. As the dome of ice evolves, its margin advances and its thickness decreases (there is no surface mass balance to add new mass). The script `halfar_results.py` will plot the modeled and analytic thickness at a specified time (Figure 8.1), and also report error statistics. Invoke `halfar_results.py --help` for details on its use.

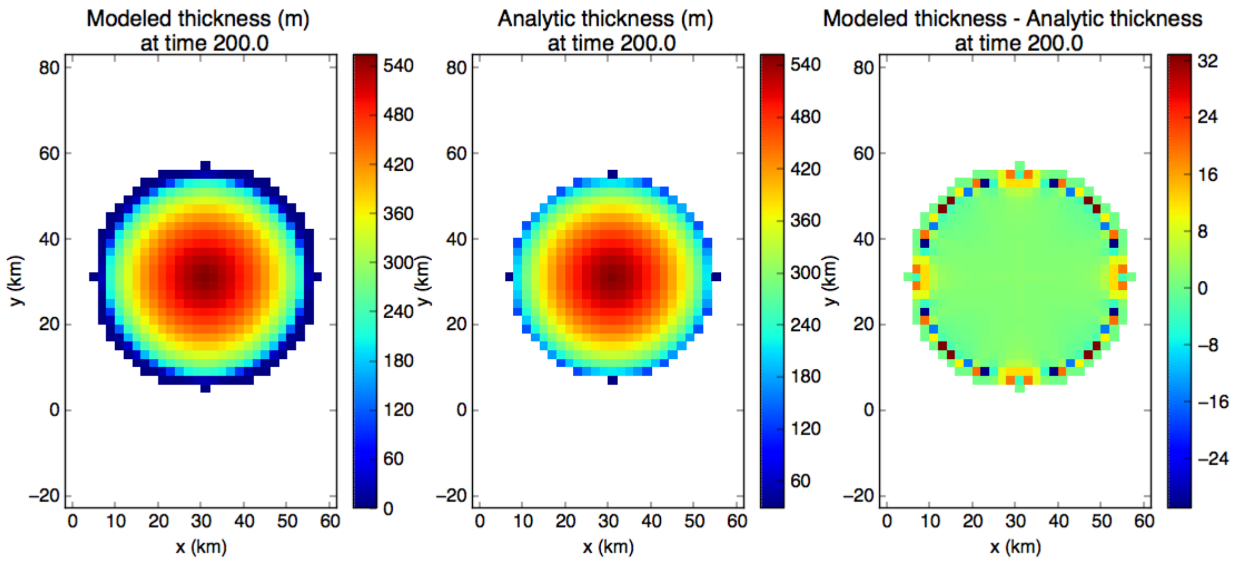


Figure 8.1: Halfar test case results (using Glide) after 200 years of dome evolution. This figure is generated by `halfar_results.py`.

8.1.2 EISMINT-1

This test case is from phase 1 of the European Ice Sheet Modelling INiTiative intercomparison experiments. These experiments are described in more detail here[2] and in Huybrechts et al. (1996).

Provided files

- README.md
  Information about the test case, including technical details about running it.

- *.config
  There are .config files for each of the six experiments in EISMINT 1: three fixed margin experiments (fm) and three moving margin experiments (mm).

Running the test

There is no script for running these experiments. They are simply run manually, e.g. using:

```
./cism_driver e1-fm.1.config
```

Results

These experiments are meant to be run to steady-state, and the supplied .config files are set up to run for long enough to do this. These simulations take more than a few minutes to complete. As the ice sheet evolves, its shape eventually equilibrates with the imposed surface mass balance. Currently there is no script for analyzing the model results. However, users can visually compare their results to those in Huybrechts et al. (1996).

8.1.3 EISMINT-2

This test case is from phase 2 of the European Ice Sheet Modelling INiTiative intercomparison experiments. These experiments are described in more detail here[3] and in Payne et al. (2000).

Provided files

- README.md
  Information about the test case, including technical details about running it.

- *.config
  There are 11 .config files: one for each of the experiments described in Payne et al. (2000).

- mound.nc, trough.nc
  These are input netCDF files used by the EISMINT-2 experiments.

Running the test

There is no script for running these experiments. They are run manually, e.g. using:

```
./cism_driver e2.a.config
```

[2]http://homepages.vub.ac.be/~phuybrec/eismint.html
[3]http://homepages.vub.ac.be/~phuybrec/eismint.html

Results

These experiments are meant to be run to steady-state, and the supplied .config files are set up to run for long enough to do this. Some experiments use the final state of a previous experiment as the initial condition (e.g., most experiments following experiment A use the final state from A as an initial condition). See the experimental descriptions in Payne et al. (2000) for details. These simulations take more than a few minutes to complete. As the ice sheet evolves, its shape equilibrates with the imposed surface mass balance. There is no script for analyzing model results, but users can visually compare their results to those in Payne et al. (2000), and also compare model diagnostics (e.g., ice area, volume, and maximum thickness) to Table 4 of that paper.

8.1.4 Glint example

Section 7.6 explains how to set up and run the Glint example test case. We summarize that information here.

Provided files

- README.md
  Information about the test case, including technical details about downloading the required data files and running the case.
- greenland_20km.config.pdd and glint_example.config.pdd
  These config files are used to run Glide with the surface mass balance computed by Glint's positive-degree-day scheme.
- greenland_20km.config.smb and glint_example.config.smb
  These config files are used to run Glide with the surface mass balance passed directly to Glint. (In real applications the SMB would be provided by a climate model, but here it is crudely approximated from temperature and precip data.)
- ncep-doe_6h_climate.64x32.nc, orog.igcmgrid.64x32.nc, gland20.input.nc
  These are data files that must be downloaded and placed in the `glint_example` test directory by the user. Users may also supply their own input data.

Running the test

There is no script for running these experiments. They are run manually, e.g. using:

```
./cism_driver greenland_20km.config.pdd glint_example.config.pdd
```

The two config files specify the ice sheet and climate configurations, respectively.

Results

The config files are set up to run Glide on a coarse Greenland grid for 10 model years, so that the default experiments complete quickly. The results are written to netCDF files. This test illustrates the application of CISM to a whole ice sheet but is not expected to be scientifically accurate.

8.2 Higher-Order Test Cases

The higher-order test cases are designed to test various aspects of higher-order dycores. By default they use the Glissade dycore with the Blatter-Pattyn approximation (see Section 6.2.1). As other higher-order dycores become available, they can be applied to these same tests. Additional test cases will be added as needed.

8.2.1 Dome

The dome test case is based on a parabolic dome of ice, similar to the Halfar test case. By default the dome has the same radius and center thickness as the Halfar case. However, it uses a simple square root function to define thickness as a function of distance from the dome center, resulting in a somewhat steeper profile. The dome test has been widely used for day-to-day testing because it is simple and relatively fast to run. It is a good test to confirm that basic higher-order model physics is working correctly, but does not strenuously test the physics and boundary conditions, or analytically verify the model.

Provided files

- README.md
  Information about the test case, including technical details about running it.
- runDome.py
  The script to set up and run the test.
- dome.config
  The default configuration settings for running CISM with the test case.
- dome-forcing.config
  An example configuration script that can be used to generate a forcing file and run CISM with it.

Running the test

One script sets up the initial condition and runs the model:

```
./runDome.py
```

There is no script for analyzing the results.

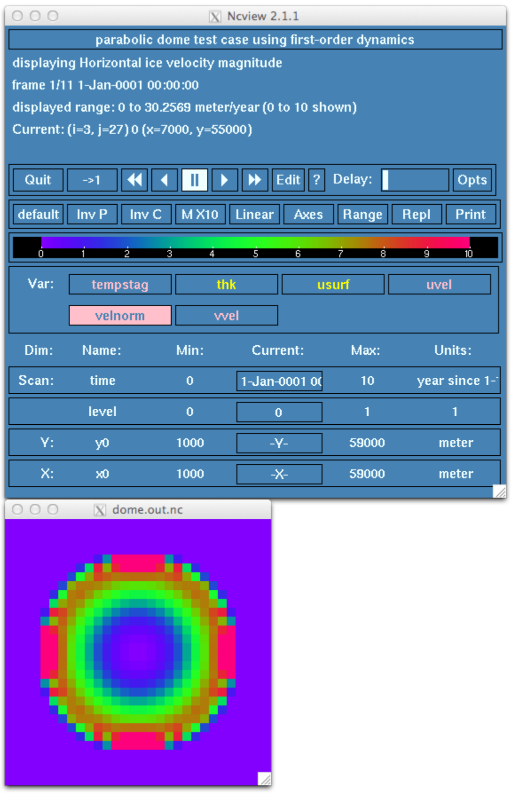
The dome test case can also be used to set up an example of CISM's time-dependent forcing capability. (See Section 7.4 for details.) By passing the runDome.py script a config file which has [CF Forcing] section, as found in dome-forcing.config, it will generate a dome.forcing.nc file that contains time-varying fields and then run CISM using this generated netCDF file. Run:

```
./runDome.py -c dome-forcing.config
```

Results

There is not an analytic solution for this test, nor is there a script to analyze the results. You can manually inspect the results using a tool such as `ncview`.[4] Sample output is shown in Figure 8.2.

[4]See section 2.2.4



parabolic dome test case using first-order dynamics
displaying Horizontal ice velocity magnitude
frame 1/11 1-Jan-0001 00:00:00
displayed range: 0 to 30.2569 meter/year (0 to 10 shown)
Current: (i=3, j=27) 0 (x=7000, y=55000)

Dim:	Name:	Min:	Current:	Max:	Units:
Scan:	time	0	1-Jan-0001 0(	10	year since 1-`
	level	0	0	1	1
Y:	y0	1000	-Y-	59000	meter
X:	x0	1000	-X-	59000	meter

Figure 8.2: Dome `velnorm` (i.e., the ice speed in m/yr) field at time 0 using default `dome.config` settings. This figure is a screenshot of ncview.

8.2.2 ISMIP-HOM

The Ice Sheet Model Intercomparison Project for Higher-Order Models (ISMIP-HOM) prescribes a set of experiments for testing the implementation of higher-order physics. For more information, see here[5] and the ISMIP-HOM description paper by Pattyn et al. (2008).

The python scripts provided (runISMIP_HOM.py and plotISMIP_HOM.py, referred to here as the ISMIP-HOM scripts) were created to run experiments A through F using CISM and compare the results with results from other models.

Note: The `README.md` file gives many additional details about running and analyzing the test case that are not described here.

Provided files

- README.md
  Information about the test case, including technical details about running it.
- ismip-hom.config
  A default configuration file used as a template for generating the .config file for each test. If you wish to run the tests with different solver settings, for example, you should edit this file.
- runISMIP_HOM.py
  The script used for running any/all of the ISMIP-HOM experiments. Invoke with '--help' to see the many command line options for controlling execution.
- plotISMIP_HOM.py
  The script used for analyzing/plotting any/all of the ISMIP-HOM experiments. Invoke with '--help' to see the many command line options for controlling execution.

Running the test

One script sets up the initial condition and runs the model:
```
./runISMIP_HOM.py
```
and another is used to analyze the results:
```
./plotISMIP_HOM.py
```

Results

The `plotISMIP_HOM.py` script will plot results relative to other models. None of the ISMIP-HOM tests have a useful analytic solution, so these tests are used as community benchmarks rather than actual model verification tests. The Pattyn et al. (2008) paper is useful for intepreting model results. An example output plot is shown in Figure 8.3.

[5] http://homepages.ulb.ac.be/~fpattyn/ismip/

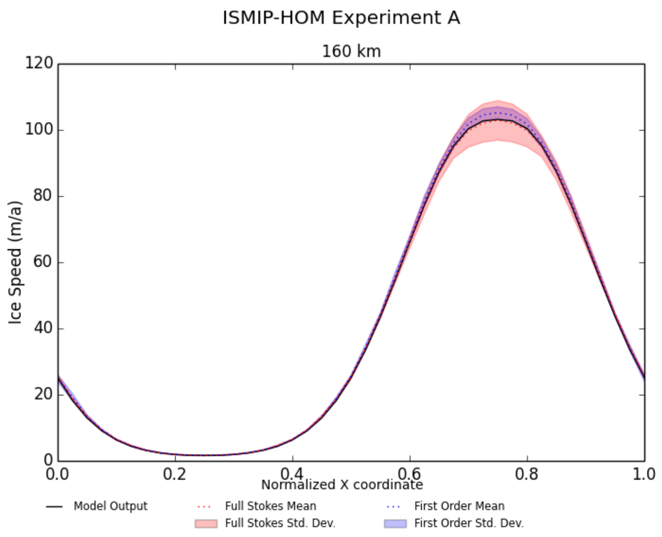


Figure 8.3: An example of the ISMIP-HOM test case output for test A at size L=160 km. CISM output is shown with a black line, and the range of output from other models is shown by colored bars. This figure was generated with `./plotISMIP_HOM.py -e a -s 160` after running the test with `./runISMIP_HOM.py -e a -s 160`. Additional options (e.g., running and plotting for multiple tests simultaneously) are described in the `README.md` and by invoking the '--help' option at the command line.

8.2.3 Stream

The stream test case simulates flow over an idealized ice stream underlain by a subglacial till with a known and specified yield stress distribution (see discussion in Section 5.3). For the two yield stress distributions specified in this test case, analytical solutions are available from Raymond (2000) and Schoof (2006).

For the Raymond test case, the yield stress within the ice stream is given a uniform value below the driving stress, and outside the ice stream it is given a uniform value much higher than the driving stress (i.e., the yield stress distribution is approximated by a step function). For the Schoof test case, the till yield stress across the ice stream is given by a continuously varying function.

In both cases, the basal properties vary in the across-flow direction only and are symmetric about the ice stream centerline. As a result, the velocity solutions are also uniform along flow and symmetric about the centerline.

Provided files

- README.md
  Information about the test case, including technical details about running it.

- runStream.py
  The script to set up and run the test.

- stream.config
  The default configuration settings for running CISM with the test case. Note that this file is parsed by the runStream.py script. Most of the relevant options that might be changed for this test case (e.g., grid spacing) can be done so from the command line, without having to edit the .config files (use `./runStream.py --help` for a description of available options). The choice of yield stress distribution (Raymond or Schoof) can be toggled by editing line 122 of `runStream.py`.

Running the test

One script sets up the initial condition and runs the model:

```
./runStream.py
```

and another is used to analyze the results:

```
./plotStream.py
```

Results

The `plotStream.py` script will plot model output relative to the analytical solutions in Raymond (2000) and Schoof (2006). The choice of analytical solution for comparison is automatic, based on the yield stress chosen in the runStream.py script. Figure 8.4 shows example output for both test cases.

Note: The excellent agreement between the CISM results and the Raymond analytical solution requires setting `which_ho_assemble_beta = 1` in the config file (see Section 7.3). With this setting, the step change in yield stress is well resolved. Otherwise there is some smoothing of the traction parameter β over neighboring nodes during finite-element assembly (Section 6.2.1), and the results are less accurate.

8.2.4 Confined shelf

The confined shelf test is based on tests 3 and 4 of the idealized (i.e., not Ross) EISMINT shelf test cases. It simulates the flow within an idealized, 500 m thick ice shelf in a confined, rectangular embayment. Grounded ice is not explicitly modeled but included in the model setup as Dirichlet boundary conditions for velocity along the ice shelf edges. More detailed information on this test case can be found here [6] in the "shelf-descr.pdf" document.

Note that the confined shelf and circular shelf experiments are both in the `tests/higher-order/shelf` directory and share some files.

Provided files

- README.md
  Information about the test case, including technical details about running it.
- runShelfConfined.py
  The script to set up and run the test.
- shelf-confined.config
  The default configuration settings for running CISM with the test case.

Running the test

One script sets up the initial condition and runs the model:

```
./runShelfConfined.py
```

Results

There is no script for analyzing the results. See the URL above for information about assessing the model output. You can manually inspect the results using a tool such as `ncview`. Figure 8.5 shows an example.

[6] http://homepages.vub.ac.be/~phuybrec/eismint/iceshelf.html

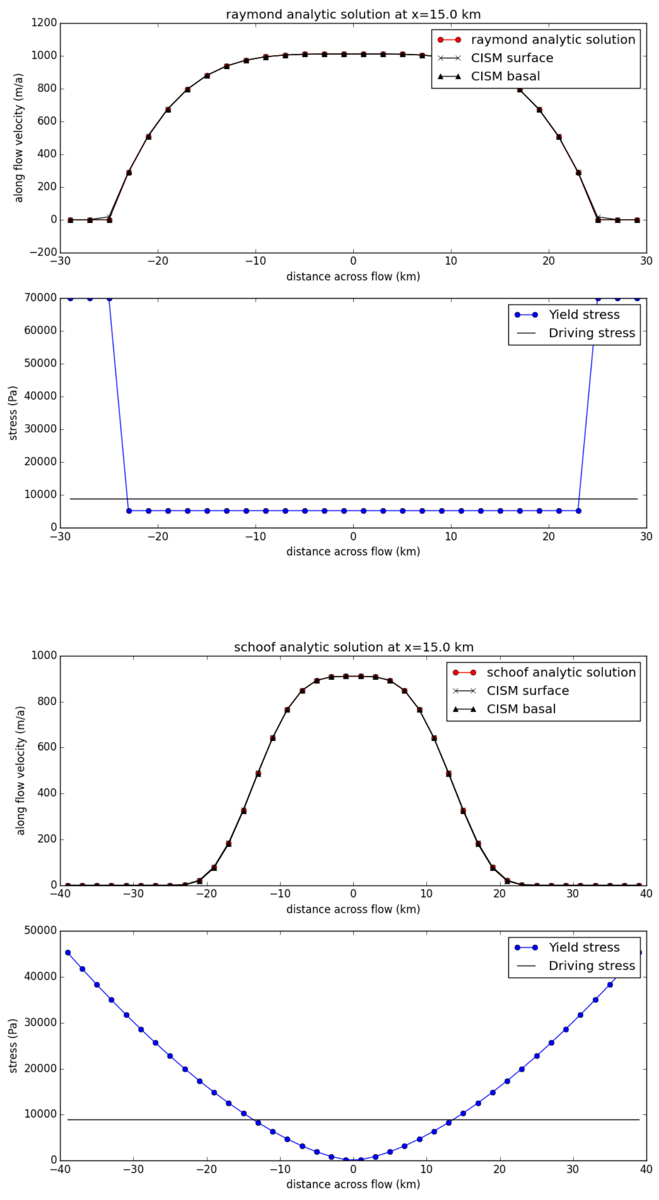
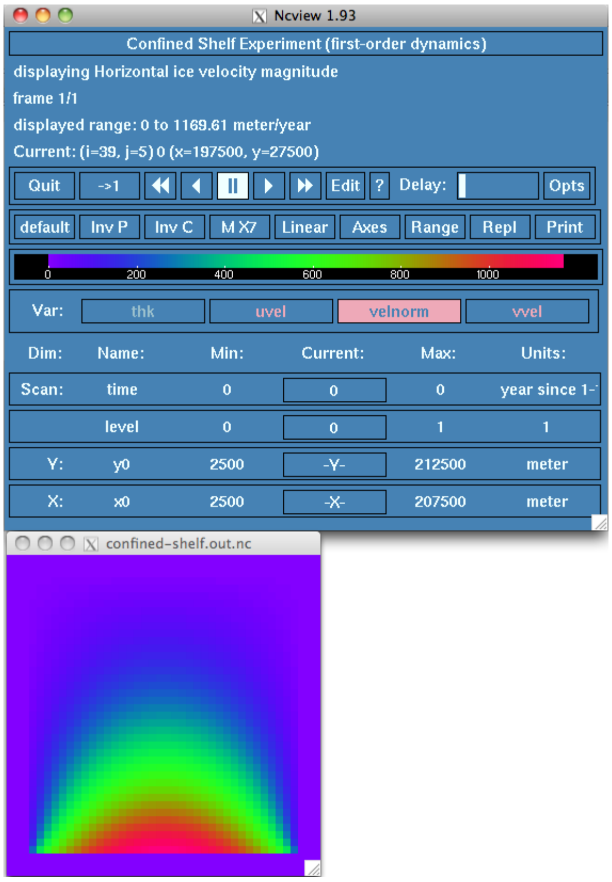


Figure 8.4: Comparison between CISM model output (black) and analytic solution (red) for the Raymond (top) and Schoof (bottom) stream test cases. This figure was generated with `./plotStream.py` after running the test with `./runStream.py`. Additional runtime options are described in the `README.md` and by invoking the '--help' option at the command line.



Confined Shelf Experiment (first-order dynamics)
displaying Horizontal ice velocity magnitude
frame 1/1
displayed range: 0 to 1169.61 meter/year
Current: (i=39, j=5) 0 (x=197500, y=27500)

Var: thk uvel velnorm vvel

```
Dim:   Name:   Min:   Current:   Max:     Units:
Scan:  time    0      0          0        year since 1-
       level   0      0          1        1
Y:     y0      2500   -Y-        212500   meter
X:     x0      2500   -X-        207500   meter
```

Figure 8.5: Confined shelf `velnorm` field using default `shelf-confined.config` settings. This figure is a screenshot of ncview.

8.2.5 Circular shelf

The circular shelf test case is a variant on the confined shelf discussed above. It simulates the flow within a circular ice shelf with a uniform thickness of 1000 m, which is grounded at a single grid point at its center. This test case confirms a working "floating ice" boundary condition in 2D (i.e., in map plane view) and also confirms radial symmetry.

Note that the confined shelf and circular shelf experiments are both in the `tests/higher-order/shelf` directory and share some files.

Provided files

- README.md
  Information about the test case, including technical details about running it.
- runShelfCircular.py
  The script to set up and run the test.
- shelf-circular.config
  The default configuration settings for running CISM with the test case.

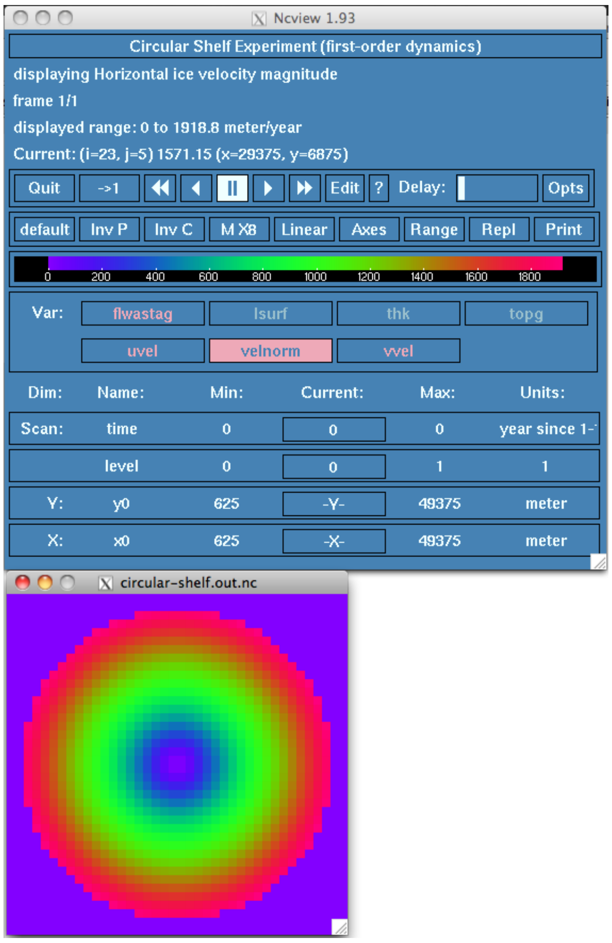
Running the test

One script sets up the initial condition and runs the model:

```
./runShelfCircular.py
```

Results

There is no script for analyzing the results. You can manually inspect the results using a tool such as `ncview`. Figure 8.6 shows an example.



Circular Shelf Experiment (first-order dynamics)
displaying Horizontal ice velocity magnitude
frame 1/1
displayed range: 0 to 1918.8 meter/year
Current: (i=23, j=5) 1571.15 (x=29375, y=6875)

Dim:	Name:	Min:	Current:	Max:	Units:
Scan:	time	0	0	0	year since 1-
	level	0	0	1	1
Y:	y0	625	-Y-	49375	meter
X:	x0	625	-X-	49375	meter

Figure 8.6: Circular shelf velnorm field using default `shelf-circular.config` settings. This figure is a screenshot of ncview.

8.2.6 Ross Ice Shelf

The Ross experiment is designed to simulate the flow of the Ross Ice Shelf of Antarctica under idealized conditions (e.g., constant and uniform flow-law rate factor). For more information about the experiment and its results, see here[7]. Also, see MacAyeal et al. (1996) for a discussion of the official model intercomparison results.

Using the Glissade solver with the Blatter-Pattyn approximation, this experiment typically takes about 10 minutes to run on a single processor. Using the shallow-shelf approximation instead, the results are very similar and are found more quickly (in less than a minute). For this reason, the SSA (`which_ho_approx = 1`) is the default.

Provided files

- README.md
  Information about the test case, including technical details about running it.
- runRoss.py
  The script to set up and run the test.
- ross.config
  The default configuration settings for running CISM with the test case.
- plotRoss.py
  The script to plot the test results.

Running the test

One script sets up the initial condition and runs the model:

```
./runRoss.py -r
```

(Without the "-r" flag, the script will set up the initial condition but not run CISM.) Another script can be used to visualize the results:

```
./plotRoss.py
```

Results

The `plotRoss.py` script will generate a figure of the velocity field calculated for the Ross Ice Shelf. The results should look very similar to Figures 8.7 and 8.8. You can compare these with similar figures in the paper by MacAyeal et al. (1996).

[7] http://homepages.vub.ac.be/~phuybrec/eismint/iceshelf.html

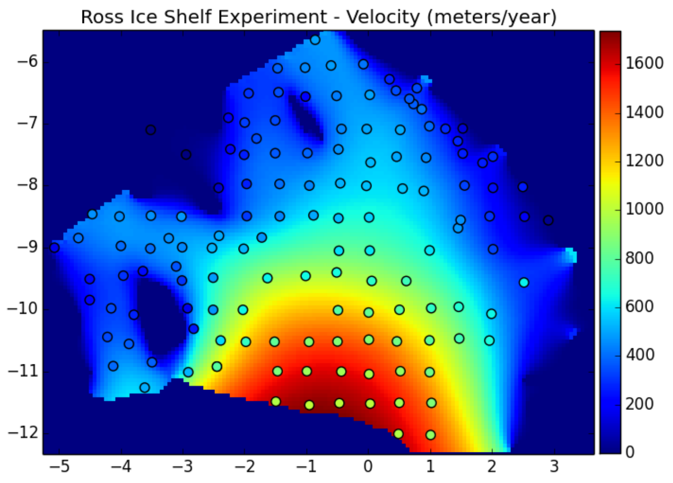


Figure 8.7: Ross Ice Shelf velocity field calculated by CISM. This figure is generated by `plotRoss.py`.

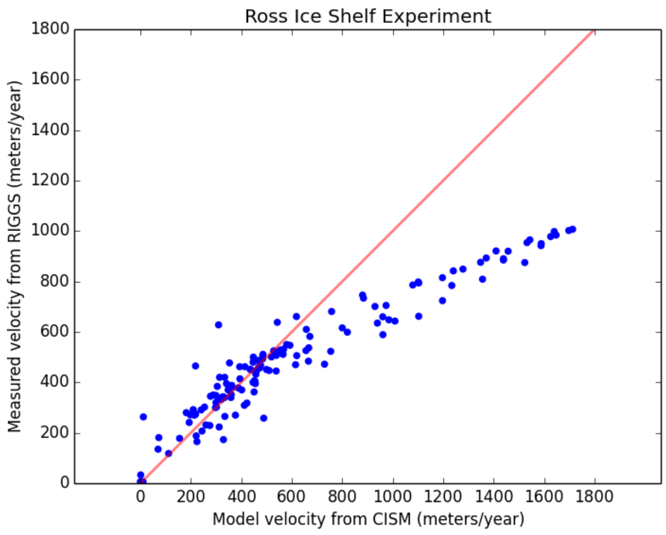


Figure 8.8: CISM-modeled Ross Ice Shelf speeds vs. those from observations. This figure is generated by `plotRoss.py`.

8.2.7 Other tests

Other higher-order tests that are still in development (e.g., "slab") are also included in the `./tests/higher-order/` directory. Instructions for running these tests are included in each directory. Since these tests have not yet been validated, they are for use at your own risk.

8.3 The build and test structure (BATS)

A build and test structure (BATS) has been included in the `./tests/regression/` directory. BATS is capable of automatically building CISM and then running a set of regression tests on a number of platforms. BATS is primarily intended to allow users and developers of CISM to quickly generate a set of regression tests for use with the Land Ice Verification and validation toolkit(LIVVkit) [8]. This allows users to quickly and easily build confidence in their installation of CISM and, by extension, their scientific results.

Provided files

- README.md
  Information about the build and test structure, including technical details about using it.
- `build_and_test.py`
  The main script to build CISM, and run a set of regression tests.
- `setup_hopper.bash` and `setup_titan.bash`
  Scripts to load the needed modules on the higher-performance-computing architectures Hopper (NERSC) and Titan (OLCF).
- `util/`
  A directory containing a set of helper python modules and files for `build_and_test.py`.

Using BATS

BATS works very similar to how you would build CISM and run one or more CISM tests. BATS will build a version of CISM, and then either run a set of regression tests if you are using a personal computer (PC), or setup a series of regression tests and generate a job submission script if you are using a high-performance-computing architecture (HPC).

For example, on the HPC Titan, if you wanted to build CISM using the gnu compiler, and run a set of regression tests, you would run these commands:

```
cd tests/regression/
source setup_titan.bash
./build_and_test.py -p titan -c gnu -b ./build
```

which will result in BATS generating all the CMake build files into a new directory called `build` and building CISM into the directory `build/cism_driver`. BATS will then setup a set of CISM's higher-order tests:

- Dome (at a variety of resolutions and processor counts)
- Circular and confined shelf
- ISMIP-HOM a and c (at 20 and 80 km resolutions)
- ISMIP-HOM f

[8]https://github.com/LIVVkit/LIVVkit

- Stream

All the files associated with each test will be output to a new directory called `reg_test/titan-gnu`, which has a directory structure that mirrors CISMS test directory structure:

```
reg_test/
    PLATFORM-COMPILER/
        CMakeCache.txt
        higher-order/
            dome/
               dome.RESO.pPRC.*
               timing/
                   dome-t?.RESO.pPRC.*
            ismip-hom
               ismip-hom-a.RESO.pPRC.*
               ismip-hom-c.RESO.pPRC.*
               ismip-hom-f.0100.pPRC.*
            shelf
               shelf-circular.RESO.pPRC.*
               shelf-confined.RESO.pPRC.*
            stream
                stream.RESO.pPRC.*
    --------------------------------------------
        Jobs/
            platform_job.small
            platform_job.small_timing_?
            platform_job.large
            platform_job.large_timing_?
        submit_all_jobs.bash
        clean_timing.bash
```

where `*` and `?` are POSIX regular expressions metacharacters, `RESO` is a number indicating the resolution the test was run at (the meaning of the number is test dependent), and `pPRC` is a number, prefixed by a `p`, indicating the number of processors used when running the model. This `reg_test` directory will be formatted to used with LIVVkit directly. Note: everything below the dashed line will only appear on HPC systems (tests are immediately run on PCs). `submit_all_jobs.bash` will submit all the jobs in the `jobs/` directory and `clean_timing.bash` will clean out any `higher-order/*/timing/` directory such that only the timing files remain (to be used after all jobs finish).

Advanced usage

BATS is designed to be flexible and work with any LIVVkit usage scenario. In order to do that, BATS provides a number of options to configure which system you are using, when/where/how CISM is built, the destination of the output directory, and which tests are run. For more information, see the LIVVkit wiki [9].

[9] https://github.com/LIVVkit/LIVVkit/wiki

Part II

Appendix

Appendix A

NetCDF Variables

The following list shows all the netCDF variable names used by CISM. Only variables marked with * are loaded (if present) by the input routines.

A.1 Glide/Glissade Variables

Name	Description	Units
`level`	sigma layers CF name: `land_ice_sigma_coordinate`	1
`lithoz`	vertical coordinate of lithosphere layer	meter
`staglevel`	stag sigma layers CF name: `land_ice_stag_sigma_coordinate`	1
`stagwbndlevel`	stag sigma layers with boundaries CF name: `land_ice_stag_sigma_coordinate_with_bnd`	1
`x0`	Cartesian x-coordinate, velocity grid	meter
`x1`*	Cartesian x-coordinate	meter
`y0`	Cartesian y-coordinate, velocity grid	meter
`y1`*	Cartesian y-coordinate	meter
`acab`*	accumulation, ablation rate CF name: `land_ice_surface_specific_mass_balance`	meter/year
`adv_cfl_dt`	advective CFL maximum time step	years
`artm`*	annual mean air temperature CF name: `surface_temperature`	degree_Celsius
`beta`*	higher-order bed stress coefficient	Pa yr/m
`bfricflx`*	basal friction heat flux	watt/meter2
`bheatflx`*	upward basal heat flux	watt/meter2
`bmlt`*	basal melt rate CF name: `land_ice_basal_melt_rate`	meter/year
`btemp`	basal ice temperature CF name: `land_ice_temperature`	degree_Celsius
`btractx`	basal traction (x-direction comp)	Pa
`btracty`	basal traction (y-direction comp)	Pa
`btrc`	basal slip coefficient	meter/pascal/year

continued on next page

continued from previous page

Name	Description	Units
`bwat`*	basal water depth	meter
`bwatflx`	basal water flux	meter3/year
`calving`	ice margin calving	meter
`diff_cfl_dt`	diffusive CFL maximum time step	years
`diffu`	apparent diffusivity	meter2/year
`dissip`	dissipation divided by rhoi Ci	deg C
`dthckdtm`	tendency of ice thickness (NOTE: Glide only)	meter/year
`dusrfdtm`	rate of upper ice surface elevation change (NOTE: Glide only)	meter/year
`dynbcmask`	2d array of higher-order model boundary condition mask values (NOTE: Glam ONLY)	1
`effecpress`*	effective pressure	Pa
`efvs`*	effective viscosity	Pascal * years
`eus`	global average sea level CF name: `global_average_sea_level_change`	meter
`flwa`*	Pre-exponential flow law parameter	pascal**(-n) year**(-1)
`flwastag`*	Pre-exponential flow law parameter	pascal**(-n) year**(-1)
`gravity`	gravitational acceleration CF name: `gravity`	meter/s/s
`iarea`	area covered by ice	km2
`iareaf`	area covered by floating ice	km2
`iareag`	area covered by grounded ice	km2
`ice_mask`	real-valued mask denoting ice (1) or no ice (0)	1
`ice_specific_heat`	ice specific heat CF name: `ice_specific_heat`	J/kg/K
`ice_thermal_conductivity`	ice thermal conductivity CF name: `ice_thermal_conductivity`	J/(K kg)
`ivol`	ice volume	km3
`kinbcmask`*	Mask of locations where uvel, vvel value should be held	1
`litho_temp`*	lithosphere temperature	degree_Celsius
`lsurf`	ice lower surface elevation	meter
`relx`*	relaxed bedrock topography	meter
`resid_u`	u component of residual Ax - b (NOTE: Glam only)	Pa/m
`resid_v`	v component of residual Ax - b (NOTE: Glam only)	Pa/m
`rho_ice`	ice density CF name: `rho_ice`	kg/meter3
`rho_seawater`	seawater density CF name: `rho_seawater`	kg/meter3
`rhs_u`	u component of b in Ax = b	Pa/m
`rhs_v`	v component of b in Ax = b	Pa/m
`seconds_per_year`	seconds per year CF name: `seconds_per_year`	s/yr
`soft`*	bed softness parameter	meter/pascal/year
`stagthk`	staggered ice thickness CF name: `stag_land_ice_thickness`	meter

continued on next page

continued from previous page

Name	Description	Units
`surftemp`*	annual mean surface temperature CF name: `surface_temperature`	degree_Celsius
`tau_eff`*	effective stress	Pa
`tau_xx`	x component of horiz. normal stress	Pa
`tau_xy`	horiz. shear stress	Pa
`tau_xz`	X component vertical shear stress	Pa
`tau_yy`	y component of horiz. normal stress	Pa
`tau_yz`	Y component vertical shear stress	Pa
`tauf`*	higher-order basal yield stress	Pa
`taux`	basal shear stress in x direction (NOTE: Glide only)	kilopascal
`tauy`	basal shear stress in y direction	kilopascal
`temp`*	ice temperature CF name: `land_ice_temperature`	degree_Celsius
`tempstag`*	ice temperature on staggered vertical levels with boundaries CF name: `land_ice_temperature_stag`	degree_Celsius
`thk`*	ice thickness CF name: `land_ice_thickness`	meter
`thkmask`*	mask	1
`topg`*	bedrock topography CF name: `bedrock_altitude`	meter
`ubas`*	basal slip velocity in x direction CF name: `land_ice_basal_x_velocity`	meter/year
`uflx`	flux in x direction (NOTE: Glide and Glam only)	meter2/year
`unstagbeta`*	higher-order bed stress coefficient on the unstaggered grid (NOTE: this will overwrite beta if both are input)	Pa yr/m
`usurf`*	ice upper surface elevation CF name: `surface_altitude`	meter
`uvel`*	ice velocity in x direction CF name: `land_ice_x_velocity`	meter/year
`uvel_extend`	ice velocity in x direction (extended grid) CF name: `land_ice_x_velocity`	meter/year
`vbas`*	basal slip velocity in y direction CF name: `land_ice_basal_y_velocity`	meter/year
`velnorm`	Horizontal ice velocity magnitude	meter/year
`vflx`	flux in x direction (NOTE: Glide and Glam only)	meter2/year
`vvel`*	ice velocity in y direction CF name: `land_ice_y_velocity`	meter/year
`vvel_extend`	ice velocity in y direction (extended grid) CF name: `land_ice_y_velocity`	meter/year
`wgrd`*	Vertical grid velocity	meter/year
`wvel`*	vertical ice velocity	meter/year
`wvel_ho`*	vertical ice velocity relative to ice sheet base from higher-order model (NOTE: Glam only)	meter/year

A.2 Glint Variables

Name	Description	Units
`ablt`	ablation	meter (water)/year
`arng`	air temperature half-range	degreeC
`global_orog`	orographic elevation provided by global model CF name: `surface_altitude`	meter
`hflx_tavg`*	heat flux to ice surface	W m-2
`inmask`	downscaling mask	1
`local_orog`	orographic elevation provided by local model CF name: `surface_altitude`	meter
`outmask`*	upscaling mask	1
`prcp`	precipitation CF name: `lwe_precipitation_rate`	meter (water)/year
`rofi_tavg`*	solid calving flux	kg m-2 s-1
`rofl_tavg`*	liquid runoff flux	kg m-2 s-1
`siced`*	superimposed ice depth	meter
`snowd`*	snow depth CF name: `surface_snow_thickness`	meter

Appendix B

Input and Output (I/O)

B.1 NetCDF I/O

The netCDF[1] library is used for platform-independent, binary file I/O. CISM uses the f90 netCDF interface. Several source files are automatically generated from template files and a variable definition file using a python script. The netCDF files adhere to the CF convention[2] for naming climate variables. The netCDF files also store parameters used to define the geographic projection.

B.1.1 Data structures

Information associated with each dataset is stored in the `glimmer_nc_stat` type. Variable and dimension IDs are retrived from the data set by using the relevant netCDF library calls. Metadata (such as title, institution and comments) are stored in the derived type `glimmer_nc_meta`.

Input and output files are managed by two separate linked lists. Elements of the input file list contain the number of available time slices and information describing which time slice(s) should be read. Output file elements describe how often data should be written and the current time.

B.1.2 Code generator

Much of the code needed to do netCDF I/O is repetitive and can therefore be automatically generated. The code generator, `generate_ncvars.py`, is written in python and produces source files from a template `ncdf_template.in` and the variable definition file (Section B.1.3). The templates are valid source files; the generator simply replaces special comments with the code generated from the variable file.

B.1.3 Variable definition file

All netCDF variables are defined in control files, `MOD_vars.def`, where `MOD` is the name of the model subsystem. Variables can be modified/added by editing these files to provide I/O functionality for additional variables beyond those listed in Appendix A. A common example would be to modify the file `glide_vars.def` in `./libglide` so that additional standard variables can be treated as "loadable" on input, or so that other standard variables internal to the code can be written to NetCDF output. Note that if `MOD_vars.def` is modified, the CMake configure script must be re-sourced before running "make" in order for those changes to be applied.

[1]`http://www.unidata.ucar.edu/packages/netcdf/`
[2]`http://cfconventions.org/`

The file is read using the python `ConfigParser` module. The format of the file is similar to Windows `.ini` files. Lines beginning with `#` or `;` or empty lines are ignored. These files must have a definition section `[VARSET]` (see Table B.1). A new variable definition block starts with the variable name in square brackets [ ]. Variables are further specified by parameter name/value pairs which are separated by `:` or `=`. Parameter names and their meanings are summarized in Table B.2. All parameter names not recognised by the code generator (i.e., not in Table B.2) are added as variable attributes.

name	description
`name`	Name of the model subsystem, e.g. `glide`. The f90 file is renamed based on this name. The f90 module and module procedures are prefixed with this name.
`datatype`	The name of the f90 type on which the netCDF variables depend.
`datamod`	The name of the f90 module in which the f90 type, `datatype`, is defined.

Table B.1: Each variable definition file must have a section, called `[VARSET]`, containing the parameters described above.

name	description
`dimensions`	List of comma-separated dimension names of the variable. C notation is used here, i.e. the slowest-varying dimension is listed first.
`data`	The variable to be stored/loaded. The f90 variable is assumed to be one dimension smaller than the netCDF variable, i.e. f90 variables are always snapshots of the present state of the model. Variables which do not depend on time are not handled automatically. Typically, these variables are filled when the netCDF file is created.
`factor`	Variables are multiplied with this factor on output and divided by this factor on input. Default: 1.
`load`	Set to 1 if the variable can be loaded from file. Default: 0.
`average`	Set to 1 if the variable should also be available as a mean over the write–out interval. Averages are calculated only if required. To store the average in a netCDF output file, append `_tavg` to the variable name.
`units`	UDUNITS compatible unit string describing the variable units.
`long_name`	A more descriptive name of the variable.
`standard_name`	The corresponding standard name defined by the CF standard.

Table B.2: List of accepted variable definition parameters.

Appendix C

Commonly Used Notation

The following table is a partial list of the notation for commonly used variables in this document.

Symbol	Variable
H	ice thickness
s	ice surface elevation
b	ice bottom elevation
b_r	bedrock elevation
B_s	surface mass balance (positive for accumulation)
M_b	basal melt rate (positive for melt)
B	combined surface and basal mass balance
u	x-component of ice velocity
v	y-component of ice velocity
w	z-component of ice velocity
η	effective viscosity

Part III

Bibliography

Bibliography

Robert Bindschadler. The importance of pressurized subglacial water in separation and sliding at the glacier bed. *Journal of Glaciology*, 29:3–19, 1983.

H. Blatter. Velocity and stress fields in grounded glaciers - a simple algorithm for including deviatoric stress gradients. *Journal Of Glaciology*, 41(138):333–344, 1995.

W.F. Budd, P.L. Keage, and N.A. Blundy. Empirical studies of ice sliding. *Journal of Glaciology*, 23(89):157–170, 1979.

Ed Bueler and Jed Brown. Shallow shelf approximation as a "sliding law" in a thermomechanically coupled ice sheet model. *Journal of Geophysical Research*, 114(F3):1–21, July 2009.

Ed Bueler, Craig S. Lingle, Jed A. Kallen-Brown, David N. Covey, and Latrice N. Bowman. Exact solutions and verification of numerical models for isothermal ice sheets. *Journal of Glaciology*, 51(173):291–306, March 2005. ISSN 00221430. doi: 10.3189/172756505781829449. URL http://openurl.ingenta.com/content/xref?genre=article&issn=0022-1430&volume=51&issue=173&spage=291.

K. Cuffey and Paterson. *The Physics of Glaciers*. Butterworth-Heinneman, Amsterdam, 4th edition, 2010.

J. K. Dukowicz and J. R. Baumgardner. Incremental remapping as a transport/advection algorithm. *J. Comput. Phys.*, 160:318–335, 2000.

J. K. Dukowicz, S. F. Price, and W. 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.

O Gagliardini, T Zwinger, F Gillet-Chaulet, G Durand, L Favier, B De Fleurian, R Greve, M Malinen, C Martin, P Råback, J Ruokolainen, M Sacchettini, M Schäfer, H Seddik, and J Thies. Capabilities and performance of Elmer/Ice, a new-generation ice sheet model. *Geoscientific Model Development*, 6(4):1299–1318, 2013.

M. K. M. Hagdorn. *Reconstruction of the Past and Forecast of the Future European and British Ice Sheets and Associated Sea Level Change*. PhD thesis, University of Edinburgh, 2003.

P Halfar. On the Dynamics of the Ice Sheets 2. *Journal of Geophysical Research*, 88(C10): 6043–6051, 1983.

R.C.A. Hindmarsh. A numerical comparison of approximations to the stokes equations used in ice sheet and glacier modeling. *Journal of Geophysical Research*, 109, 2004. doi: 10.1029/2003JF000065.

J. R. Holton. *An Introduction to Dynamic Meteorology*, volume 48 of *International Geophysics Series*. Academic Press, New York, 3rd edition, 1992.

T.J.R. Hughes. *The Finite Element Method: Linear Static and Dynamic Finite Element Analysis*. Dover Civil and Mechanical Engineering. Dover, Mineola, New York, 1st edition, 2000.

K. Hutter. *Theoretical Glaciology*. Mathematical Approaches to Geophysics. D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, 1983.

P. Huybrechts. A three–dimensional time–dependent numerical model for polar ice sheets; some basic testing with a stable and efficient finite difference scheme. Technical report, Geografisch Instituut, Vrije Universiteit Brussel, Belgium, 1986.

Philippe Huybrechts, Tony Payne, and The EISMINT Intercomparison Group. The EISMINT benchmarks for testing ice-sheet models. *Annals of Glaciology*, 23:1–12, 1996.

Nina Kirchner, Kolumban Hutter, Martin Jakobsson, and Richard Gyllencreutz. Capabilities and limitations of numerical ice sheet models: a discussion for Earth-scientists and modelers. *Quaternary Science Reviews*, 30(25-26):3691–3704, December 2011. ISSN 02773791. doi: 10.1016/j.quascirev.2011.09.012. URL http://linkinghub.elsevier.com/retrieve/pii/S0277379111002915.

K. Lambeck and S. M. Nakiboglu. Seamount loading and stress in the ocean lithosphere. *Journal of Geophysical Research*, 85:6403–6418, 1980.

G. R. Leguy, X. S. Asay-Davis, and W. H. Lipscomb. Parameterization of basal friction near grounding lines in a one-dimensional ice sheet model. *The Cryosphere*, 8(4):1239–1259, July 2014. ISSN 1994-0424. doi: 10.5194/tc-8-1239-2014. URL http://www.the-cryosphere.net/8/1239/2014/.

Wei Leng, Lili Ju, Max Gunzburger, Stephen Price, and Todd Ringler. A parallel high-order accurate finite element nonlinear Stokes ice sheet model and benchmark experiments. *Journal of Geophysical Research*, 117(F1), January 2012.

W. H. Lipscomb and E. C. Hunke. Modeling sea ice transport using incremental remapping. *Mon. Wea. Rev.*, 132:1341–1354, 2004.

W. H. Lipscomb, J. G. Fyke, M. Vizcaino, W. J. Sacks, J. Wolfe, M. Vertenstein, A. Craig, E. Kluzek, and D. M. Lawrence. Implementation and initial evaluation of the glimmer community ice sheet model in the community earth system model. *Journal of Climate*, 26:7352–7371, 2013. doi: 10.1175/JCLI-D-12-00557.1.

D R Macayeal. Large-Scale Ice Flow Over a Viscous Basal Sediment - Theory and Application to Ice Stream-B, Antarctica. *Journal of Geophysical Research*, 94(B4):4071–4087, 1989.

Douglas R MacAyeal, V Rommelaer, P Huybrechts, C L Hulbe, J determann, and C Ritz. An ice-shelf model test based on the Ross Ice Shelf, Antarctica. *Annals of Glaciology*, 23:46–51, 1996.

W. S. B. Paterson. *The Physics of Glaciers*. Butterworth–Heinemann, Oxford, 3rd edition, 1994.

W.S.B. Paterson and W. F. Budd. Flow parameters for ice sheet modeling. *Cold Reg. Sci. Technol.*, 6:175–177, 1982.

F Pattyn. A new three-dimensional higher-order thermomechanical ice sheet model: Basic sensitivity, ice stream development, and ice flow across subglacial lakes. *Journal Of Geophysical Research-Solid Earth*, 108:–, 2003.

F. Pattyn, L. Perichon, A. Aschwanden, B. Breuer, B. de Smedt, O. Gagliardini, G. H. Gudmundsson, R. C. A. Hindmarsh, A. Hubbard, J. V. Johnson, T. Kleiner, Y. Konovalov, C. Martin, A. J. Payne, D. Pollard, S. Price, M. Rückamp, F. Saito, O. Souček, S. Sugiyama, and T. Zwinger. Benchmark experiments for higher-order and full-Stokes ice sheet models (ISMIP-HOM). *The Cryosphere*, 2(2):95–108, August 2008. ISSN 1994-0424. doi: 10.5194/tc-2-95-2008. URL http://www.the-cryosphere.net/2/95/2008/.

A. J. Payne. A thermomechanical model of ice flow in West Antarctica. *Climate Dynamics*, 15: 115–125, 1999.

A. J. Payne and P. W. Dongelmans. Self–organisation in the thermomechanical flow of ice sheets. *Journal of Geophysical Research*, 102(B6):12219–12233, 1997.

A J Payne, P Huybrechts, R Calov, J L Fastook, R Greve, S J Marshall, I Marsiat, C Ritz, L Tarasov, and M P A Thomassen. Results from the EISMINT model intercomparison: the effects of thermomechanical coupling. *Journal of Glaciology*, 46(153):227–238, 2000.

W. R. Peltier, D. L. Goldsby, D. L. Kohlstedt, and L. Tarasov. Ice–age ice–sheet rheology: constraints from the Last Glacial Maximum form of the Laurentide ice sheet. *Annals of Glaciology*, 30:163–176, 2000.

M. Perego, M. Gunzburger, and J. Burkardt. Parallel finite-element implementation for higher-order ice sheet models. *Journal of Glaciology*, 58:76–88, 2012. doi: 10.3189/2012JoG11J063.

S. Pimentel, G. E. Flowers, and C. G. Schoof. A hydrologically coupled higher-order flow-band model of ice dynamics with a Coulomb friction sliding law. *Journal of Geophysical Research*, 115(F4):1–16, November 2010. ISSN 0148-0227. doi: 10.1029/2009JF001621. URL http://www.agu.org/pubs/crossref/2010/2009JF001621.shtml.

David Pollard and Robert M Deconto. Modelling West Antarctic ice sheet growth and collapse through the past five million years. *Nature*, 458(7236):329–332, September 2009.

W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in FORTRAN.* Cambridge University Press, 2nd edition, 1992.

Stephen F Price, Antony J Payne, Ian M Howat, and Benjamin E Smith. Committed sea-level rise for the next century from Greenland ice sheet dynamics during the past decade. *Proceedings of the National Academy of Sciences of the United States of America*, 108(22):8978–83, May 2011. ISSN 1091-6490. doi: 10.1073/pnas.1017313108. URL http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=3107304&tool=pmcentrez&rendertype=abstract.

C F Raymond. Energy balance of ice streams. *Journal Of Glaciology*, 46(155):665–674, 2000.

C. Ritz. Time dependent boundary conditions for calculation of temperature fields in ice sheets. In *The Physical Basis of Ice Sheet Modelling*, number 170 in IAHS Publications, pages 207–216, 1987.

I.C. Rutt, M. Hagdorn, N.R.J. Hulton, and A.J. Payne. The Glimmer community ice sheet model. *Journal of Geophysical Research*, 114:F02004, 2009.

C. Schoof. The effect of cavitation on glacier sliding. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 461(2055):609–627, March 2005. ISSN 1364-5021. doi: 10.1098/rspa.2004.1350. URL http://rspa.royalsocietypublishing.org/cgi/doi/10.1098/rspa.2004.1350.

C Schoof. A variational approach to ice stream flow. *Journal of Fluid Mechanics*, 556:227–251, 2006.

C Schoof and R C A Hindmarsh. Thin-Film Flows with Wall Slip: An Asymptotic Analysis of Higher Order Glacier Flow Models. *The Quarterly Journal of Mechanics and Applied Mathematics*, 63(1):73–114, January 2010.

Christian Schoof and Ian Hewitt. Ice-sheet dynamics. *Annual Review of Fluid Mechanics*, 45: 217–239, 2013.

L. Tarasov and W. R. Peltier. Impact of thermomechanical ice sheet coupling on a model of the 100kyr ice age cycle. *Journal of Geophysical Research*, 104(D8):9517–9545, 1999.

L. Tarasov and W. R. Peltier. Laurentide ice sheet aspect ratio in models based on Glen's flow law. *Annals of Glaciology*, 30:177–186, 2000.

J. Weertman. On the sliding of glaciers. *Journal of Glaciology*, 3:33–38, 1957.

J. Weertman. The theory of glacier sliding. *Journal of Glaciology*, 5(39):287–303, 1964.