PISM Installation Manual

The PISM Authors

Contents

Support by email: `help@pism-docs.org`.

Please see the *PISM User's Manual* for the full list of authors.

Manual date June 30, 2015. Based on PISM revision `stable v0.7.1-2-g79b8840`.

Get development branch source code: `git clone -b dev git@github.com:pism/pism.git pism-dev`

1 Introduction

This *Installation Manual* describes how to download the PISM source code and install PISM and the libraries it needs. Information about PISM, including a *User's Manual*, is on-line at

www.pism-docs.org

The fastest path to a fully functional PISM installation is to use a Linux system with a Debian-based package system (e.g. Ubuntu): Start by following subsections 3.1.1 about getting Debian packages for prerequisites, then 3.1.3 to install PETSc from source, then 3.2 to install PISM itself, and finally install Python packages following section 5.

> *WARNING*: PISM is an ongoing project. Ice sheet modeling is complicated and is generally not mature. Please don't trust the results of PISM or any other ice sheet model without a fair amount of exploration.
>
> Also, please don't expect all your questions to be answered here. Write to us with questions at `help@pism-docs.org`.

2 Libraries and programs needed by PISM

This table lists required dependencies for PISM alphabetically.

Required Program or Library		Comment
FFTW	http://www.fftw.org/	version $\geq$ 3.1
GSL	http://www.gnu.org/software/gsl/	version $\geq$ 1.15
MPI	http://www.mcs.anl.gov/mpi/	
NetCDF	http://www.unidata.ucar.edu/software/netcdf/	version $\geq$ 4.1
PETSc	http://www.mcs.anl.gov/petsc/petsc-as/	version $\geq$ 3.3
UDUNITS-2	http://www.unidata.ucar.edu/software/udunits/	

Before installing these "by hand", check the Debian and Mac OS X sections below for specific how-to. In particular, if multiple MPI implementations (e.g. MPICH and Open-MPI) are installed then PETSc can under some situations "get confused" and throw MPI-related errors. Even package systems have been known to allow this confusion.

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

Recommended Library		Comment
PROJ.4	http://trac.osgeo.org/proj/	
PnetCDF	http://trac.mcs.anl.gov/projects/parallel-netcdf	

Python (http://python.org/) is needed both in the PETSc installation process and in scripts related to PISM pre- and post-processing, while Git (http://git-scm.com/) is usually needed to download the PISM code. Both should be included in any Linux/Unix distribution.

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

Recommended Python Package		Comment
matplotlib	http://matplotlib.sourceforge.net/	used in some scripts
netcdf4-python	https://pypi.python.org/pypi/netCDF4/1.1.7	used in *most* scripts
numpy	http://numpy.scipy.org/	used in *most* scripts

3 Installation Cookbook

3.1 Installing PISM's prerequisites

3.1.1 Installing prerequisites by packages (Debian)

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

Package name	*Comments*
`cmake`	required to configure PISM
`libfftw3-dev`	required by PISM
`g++`	required to build PISM
`libgsl0-dev`	required by PISM
`netcdf-bin`	required: `ncgen` is used during the build process
`libnetcdf-dev`	required by PISM
`libudunits2-dev`	required by PISM
`cdo`	used in some pre-processing scripts
`cmake-curses-gui`	a text-based easy interface for CMake
`git`	used to get PISM source code
`nco`	used in many pre-processing scripts
`ncview`	view fields in NetCDF files
`libproj-dev`	used to compute ice area and volume
`python-dev`	(helps with scripts. . . perhaps not essential)
`python-pyproj`	used in some pre-processing scripts
`libx11-dev`	X windows is useful to get graphics through PETSc
`libblas-dev`	BLAS is required by PETSc
`liblapack-dev`	LAPACK is required by PETSc
`openmpi-bin`	MPI is required to run PISM in parallel
`libopenmpi-dev`	MPI is required to run PISM in parallel

Once done, see 3.1.3 to install PETSc from source and then 3.2 for building PISM itself.

3.1.2 Installing prerequisites from source

From now on, this manual assumes the use of the `bash` shell.

1. You will need Python and Git installed. To use the (recommended) graphical output of PISM you will need an X Windows server.

2. Generally the "header files" for its prerequisite libraries are required for building PISM. (This means that the "developer's versions" of the libraries are needed if the libraries are downloaded from package repositories like Debian; see section 2.)

3. PISM uses NetCDF (= *network Common Data Form*) as an input and output file format. If it is not already present, install it using the instructions at the web-page or using a package management system.

4. PISM uses the GSL (= *GNU Scientific Library*) for certain numerical calculations and special functions. If it is not already present, install it using the instructions at the web-page or using a package management system.

5. PISM uses the FFTW (= *Fastest Fourier Transform in the West*) library for the deformation of the solid earth (bed) under ice loads. Install FFTW version 3.1 or later, or check that it is installed already.

6. You will need a version of MPI (= *Message Passing Interface*). Your system may have an existing MPI installation, in which case the path to the MPI directory will be used when installing PETSc (see 3.1.3). The goal is to have the PETSc installation use the same version of MPI which is called by the `mpiexec` or `mpirun` executable.

   Once MPI is installed, you will want to add the MPI `bin` directory to your path so that you can invoke MPI using the `mpiexec` or `mpirun` command. For example, you can add it with the statement

   `export PATH=/home/user/mympi/bin:$PATH` (for `bash` shell)

   or

   `setenv PATH /home/user/mympi/bin:$PATH` (for `csh` or `tcsh` shell).

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

3.1.3 Installing PETSc from source

PISM uses PETSc (= *Portable Extensible Toolkit for Scientific Computation*).[1] Unfortunately, an up-to-date PETSc distribution is unlikely to be available in package repositories. Download the PETSc source by grabbing the current gzipped tarball at:

`http://www.mcs.anl.gov/petsc/`

PISM requires a version of PETSc which is `3.3` or later. The "lite" form of the tarball is fine if you are willing to depend on an Internet connection for accessing PETSc documentation.

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

[1] "PETSc" is pronounced "pet-see".

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

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

You need to define the environment variable `PETSC_DIR`—one way is shown here—*before* running the configuration script. Turning off the inclusion of debugging code and symbols can give a significant speed improvement, but some kinds of development will benefit from a `--with-debugging=1` configuration option. Using shared libraries may be unwise on certain clusters, etc.; check with your system administrator.

(ii) It is sometimes convenient to have PETSc grab a local copy of BLAS and LAPACK rather than using the system-wide version. So one may add "`--download-f2cblaslapack=1`" to the other configure options. Since there is no use of Fortran in PISM, Fortran can also be disabled using "`--with-fortran=0`" (PETSc ≤ 3.4), or "`--with-fc=0`" (PETSc ≥ 3.5).

(iii) If there is an existing MPI installation, for example at `/home/user/mympi/`, one can point PETSc to it by adding the option "`--with-mpi-dir=/home/user/mympi/`". The path used in this option must have MPI executables `mpicxx` and `mpicc`, and either `mpiexec` or `mpirun`, in sub-directory `bin/` and MPI library files in sub-directory `lib/`. If you get messages suggesting that PETSc cannot configure using your existing MPI, you might want to try adding the `--download-mpich=1` (or `--download-openmpi=1`) option to PETSc's configure command.

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

(v) Configuring PETSc takes at least a few minutes even when everything goes smoothly. A value for the environment variable `PETSC_ARCH` will be reported at the end of the configure process; take note of this value. One may always reconfigure with additional/new `PETSC_ARCH` as needed.

(vi) After `configure.py` finishes, you will need to `make all test` in the PETSc directory and watch the result. If the X Windows system is functional some example viewers will appear; as noted you will need the X header files for this to work.

3.2 Installing PISM itself

At this point you have configured the environment which PISM needs.

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

```
export PETSC_DIR=/home/user/petsc-3.4.0/
export PETSC_ARCH=linux-gnu-opt
export PATH=$PETSC_DIR/$PETSC_ARCH/bin/:$PATH
```

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

You are ready to build PISM itself, which is a much quicker procedure, as follows:

(i) Get the latest source for PISM using the Git version control system:

a) Check the website http://www.pism-docs.org/ for the latest version of PISM.

b) Do

```
$  git clone git://github.com/pism/pism.git pism0.7
```

c) A directory called "`pism0.7/`" will be created. Note that in the future when you enter that directory, `git pull` will update to the latest revision of PISM.[2]

(ii) Build PISM:[3]

```
$  mkdir pism0.7/build
$  cd pism0.7/build
$  PISM_INSTALL_PREFIX=~/pism cmake ..
$  make install
```

Here `pism0.7` is the directory containing PISM source code while `~/pism` is the directory PISM will be installed into. All the temporary files created during the build process will be in `pism0.7/build` created above.

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

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

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

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

If your operating system does not support shared libraries[4], then set `Pism_LINK_STATICALLY` to "ON". This can be done by either running

```
$  cmake -DPism_LINK_STATICALLY:BOOL=ON ..
```

[2]Of course, after `git pull` you will `make -C build install` to recompile and re-install PISM.
[3]Please report any problems you meet at these build stages by sending us the output: help@pism-docs.org.
[4]This might be necessary if you're building on a Cray XT5 or a Sun Opteron Cluster, for example.

or by using `ccmake`:[5] run

```
$  ccmake ..
```

and then change `Pism_LINK_STATICALLY` (and then press 'c' for "configure" then 'g' for "generate makefiles"). Then do `make install`.

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

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

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

(iv) Now see section 4 or the *Getting Started* section of the *User's Manual* to continue.

3.2.1 Installing PISM and prerequisites on a Cray XK6 system

Installing PISM on Cray systems deserves special mention for two reasons:

- building on a Cray requires static linking
- Cray uses non-standard locations for libraries and header files

This describes a successful PISM installation on a Cray XK6m-200 at ARSC (`http://www.arsc.edu/arsc/support/howtos/usingfish/`).

(i) Load appropriate modules. You may need to set module versions; note that PISM requires PETSc version $\geq$ 3.3. (Loading all the necessary modules allows compilers to find headers and libraries.)

```
module swap PrgEnv-pgi PrgEnv-gnu
# netcdf-hdf5parallel/4.2.0 requires GCC 4.7
module swap gcc/4.9.0 gcc/4.7.1

for module in \
    fftw/3.3.4.0 \
    petsc/3.3.00 \
    netcdf-hdf5parallel/4.2.0 \
    gsl/1.15.gnu;
do
    module load $module
```

[5]Install the `cmake-curses-gui` package to get `ccmake` on Ubuntu.

```
done

# petsc/3.3.00 does not work with the default tpsl
module swap tpsl/1.2.00 tpsl/1.3.00
```

(ii) Cray does not provide modules for GSL, UDUNITS-2, and CMake, but your system administrators can install them for you.

(iii) Get PISM sources and create a build directory

```
git clone git://github.com/pism/pism.git
mkdir pism/build
cd pism/build
```

(iv) Create a text file `pism_config.cmake` in the build directory you created. It should contain the following, edited to reflect locations of libraries on your system and the desired PISM install location.

```
# Compiler
set (CMAKE_C_COMPILER "cc" CACHE STRING "")
set (CMAKE_CXX_COMPILER "CC" CACHE STRING "")

# Disable testing for PISM's prerequisites
set (Pism_LOOK_FOR_LIBRARIES OFF CACHE BOOL "")

# Installation path
set (CMAKE_INSTALL_PREFIX "$ENV{HOME}/pism/" CACHE STRING "")

# General compilation/linking settings
set (Pism_ADD_FPIC OFF CACHE BOOL "")
set (Pism_LINK_STATICALLY ON CACHE BOOL "")

# No Proj.4 on fish.arsc.edu
set (Pism_USE_PROJ4 OFF CACHE BOOL "")
# No TAO on fish.arsc.edu
set (Pism_USE_TAO OFF CACHE BOOL "")
# No PNetCDF on fish (alas)
set (Pism_USE_PNETCDF OFF CACHE BOOL "" FORCE)

# Set the custom GSL location
set (GSL_LIBRARIES "/path/to/libgsl.a;/path/to/libgslcblas.a" CACHE STRING "" FORCE)
set (GSL_INCLUDES  "/path/to/gsl/include" CACHE STRING "" FORCE)
```

```
# Set the custom UDUNITS2 location
set (UDUNITS2_LIBRARIES "/path/to/libudunits2.a;/path/to/libexpat.a" CACHE STRING "" FORCE)
set (UDUNITS2_INCLUDES  "/path/to/udunits2/include" CACHE STRING "" FORCE)
```

(v) Configure and build PISM:

```
cmake -C pism_config.cmake ..
make install
```

Notes:

- Setting `Pism_LOOK_FOR_LIBRARIES` to "off" lets the module system manage all necessary compile flags. (This is only necessary on systems that install libraries in non-standard locations.)
- To set PROJ.4 location manually, set variables `PROJ4_INCLUDES` and `PROJ4_LIBRARIES` and makes sure `Pism_USE_PROJ4` is set to `ON`.
- To set PnetCDF location manually, set `PNETCDF_INCLUDES`, `PNETCDF_LIBRARIES`, and `Pism_USE_PNETCDF`.
- To set parallel HDF5 location manually, set `HDF5_C_INCLUDE_DIR`, `HDF5_LIBRARIES`, `HDF5_HL_LIBRARIES`, and `Pism_USE_PARALLEL_HDF5`.
- Extra compiler flags can be added by setting `CMAKE_CXX_FLAGS`, extra linker flags – `CMAKE_EXE_LINKER_FLAGS`.

3.2.2 Installing PISM and prerequisites on Mac OS X

This section adds information on installing PISM and its prerequisites on the Mac OS X operating system.

(i) As PISM is distributed as compilable source code only, you will need software developer's tools, XCode and the *X window system*, X11. Both packages can be installed by either downloading them from Apple Developer Connection or using the Mac OS X installation DVD.

(ii) The use of MacPorts or Fink is recommended, as it significantly simplifies installing many open-source libraries. Download a package from the MacPorts homepage (or Fink homepage), install and set the environment:

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

for MacPorts and

```
source /sw/bin/init.sh
```

for Fink.

(iii) It is not necessary to install Python, as it is bundled with the operating system. Some PISM scripts use SciPy; it can be installed using MacPorts or by downloading the Enthought Python Distributions.

(iv) If you are using MacPorts, do

```
$  sudo port install netcdf ncview gsl fftw-3 libproj4 git cmake
```

Fink users should use the following command instead (`ncview` is only available in the unstable branch).

```
$  fink install netcdf gsl fftw3 proj git cmake
```

(v) At this point, all the PISM prerequisites except PETSc are installed. Download the latest PETSc tarball from the PETSc website. Untar, then change to the directory just created. The next three commands complete the PETSc installation:

```
$  export PETSC_DIR=$PWD; export PETSC_ARCH=macosx;
$  ./config/configure.py --with-shared-libraries \
                         --with-fortran=0 --with-debugging=0
$  make all test
```

(vi) Now you can build PISM as described in section 3.2.

3.3 Common build problems and solutions (i.e. if it still does not work . . .)

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

Here are some issues we know about.

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

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

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

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

  To tell CMake where to look for libraries for the target system, see http://www.cmake.org/Wiki/CMake_Cross_Compiling and the paragraph about `CMAKE_FIND_ROOT_PATH` in particular.

  You may find section 3.2.1 to be useful in a case like this.

- Note that the PISM build system uses `ncgen` from the NetCDF package to generate `pism_config.nc`. This means that a working NetCDF installation is required on both the "host" and the "target" systems when cross-compiling PISM. If CMake finds `ncgen` for the target platform, try setting `CMAKE_FIND_ROOT_PATH_MODE_PROGRAM` to `NEVER`.

- Some systems support static libraries only. To build PISM statically and tell CMake not to try to link to shared libraries, set `Pism_LINK_STATICALLY` to `ON` using `ccmake`.

- You can set `Pism_LOOK_FOR_LIBRARIES` to "`OFF`" to disable all heuristics and set compiler flags by hand. See section 3.2.1 for an example.

4 Quick tests of the installation

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

(i) Try a MPI four process verification run:

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

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

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

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

```
$ pisms -y 5000 -view_map thk,temppabase,velsurf_mag
```

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

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

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

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

```
$ make test
```

in the build directory. The message at the bottom should say "`100% tests passed, 0 tests failed out of XX`" or similar. Feel free to send the output of `make test` to help@pism-docs.org if any failed tests cannot be resolved.

Next steps

Start with the *User's Manual*, which has a "Getting started" section. A copy is on-line at the PISM homepage and documentation page `www.pism-docs.org`, along with a source code *Browser* (HTML). Completely up-to-date documentation can be built from LaTeX source in the `doc/` sub-directory, as described in the last section.

A final reminder with respect to installation: Let's assume you have checked out a copy of PISM using Git, as in step 1a above. You can update your copy of PISM to the latest version by running `git pull` in the PISM directory and `make install` in your build directory.

5 Installing Python packages

If you're lucky, you might be able to install all the Python packages mentioned in section 2 using a package manager. On the other hand, the Python packages below are not currently available in Debian package repositories. They are easy to install using Python `setuptools` or `pip`, however; these tools are included with recent versions of Python.

Python module `netCDF4`, from package `netcdf4-python`

You can skip this paragraph if you have Enthought Python Distributions installed.

To install `netcdf4-python` providing the `netCDF4` module needed by PISM scripts, download a tarball from the project homepage `https://github.com/Unidata/netcdf4-python`.

```
$  wget https://pypi.python.org/packages/source/n/netCDF4/netCDF4-VERSION.tar.gz
$  tar -xzvf netCDF4-VERSION.tar.gz
```

Enter the directory you just untarred and install:

```
$  cd netCDF4-VERSION/
$  sudo python setup.py install
```

assuming that NetCDF was installed in the `/usr/` tree. If you are using python installed via MacPorts, you can get netdf4-python by doing

```
$  sudo port install py-netcdf4
```

Alternatively you can use pip:

```
pip install netCDF4
```

The scripts in directories `util/`, `examples/...`, and so on, which need `netCDF4`, should now work.

6 Rebuilding PISM documentation

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

LaTeX	`www.latex-project.org`	needed for rebuilding any of the documentation
`doxygen`	`www.doxygen.org`	required to rebuild the *Browser* from source
`graphviz`	`www.graphviz.org`	required to rebuild the *Browser* from source

To rebuild PISM documentation, change to the PISM build directory and do

`make manual`	to build the *User's Manual*, `manual.pdf`
`make forcing`	to build the *PISM's Climate Forcing Components* document, `forcing.pdf`
`make installation`	to build this document, the *Installation Manual*, `installation.pdf`
`make browser`	to build the *PISM Source Code Browser*,

To build documentation on a system without PISM's prerequisite libraries (such as MPI and PETSc), assuming that PISM sources are in `/pism0.7`, do the following:

```
$ cd ~/pism0.7
$ mkdir doc-build # create a build directory
$ cd doc-build
$ cmake ../doc
```

then "`make manual`", "`make installation`" and others (see above) will work as expected.

6.1 Building documentation for PISM's Python bindings and inversion tools

The documentation for PISM's Python bindings uses the documentation-generation tool Sphinx; see sphinx-doc.org. The bindings make scripting and interactive PISM possible, but many PISM users will not need them. Installing them is required to use PISM for inversion of surface velocities for basal shear stress and ice hardness. Building their documentation is strongly-recommended before use.

Sphinx can be installed via `apt-get` or `macports`. See `http://sphinx-doc.org/latest/install.html` for more details. For example, do

```
sudo apt-get install sphinx-common
```

The bindings documentation also requires the Sphinx extension called `sphinxcontrib.bibtex`, which may come with some Sphinx packages (but not with Debian packages at this time). Without it you will see this error when you try to build the bindings documentation:

```
Extension error:
Could not import extension sphinxcontrib.bibtex (exception: No module named bibtex)
```

To install it see http://sphinxcontrib-bibtex.readthedocs.org.

Note that if you install Sphinx using macports, you will install a version that depends on your python version, and its executables will have names that depend on the python version, e.g. `sphinx-build-2.7` rather than `sphinx-build` for Python 2.7. You will want to set up aliases so that the standard names work as well. To do this,

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

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

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

for example.

Now you can build the documentation. In the PISM build directory, do

```
make pismpython_docs
```

If you get an error like

```
make: *** No rule to make target ‘pismpython_docs’.  Stop.
```

then re-run `cmake ..` or `ccmake ..`, making sure that Sphinx is installed (see above); the `pismpython_docs` make target will then be present.

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