Fenics Ice Sheet Model User Guide

March 20, 2020

This document briefly outlines how to get started with the Fenics ice sheet model.

Contents

1 Installation

The ice sheet mdoel is built using the open source Python finite element software Fenics, and depends on the package tlm-adjoint for implementing inversion and error propagation capabilities. The simplest way to install Fenics and tlm-adjoint is to create a conda environment.

1.1   Installing Fenics

1. Install Anaconda. This can be either Anaconda itself, or miniconda, which is a stripped down version. Ensure the Python version is greater than 3.6. Installer can be found here: https://www.anaconda.com/distribution/

2. Add the conda-forge channel.
conda config –add channels conda-forge
conda config –set channel_priority strict

3. Create a new conda environment.
conda create -n fenics -c conda-forge fenics fenics-dijitso fenics-dolfin fenics-ffc fenics-fiat fenics-libdolfin fenics-ufl

4. Enter the conda environment:
source activate fenics

5. Make sure the pip package manager is up to date:
pip install --upgrade pip

6. Install the following packages:
conda install matplotlib numpy ipython scipy

7. Install hdf5 for python:
`http://docs.h5py.org/en/latest/index.html`
pip install h5py

8. Install pyrevolve:
`https://github.com/opesci/pyrevolve`

Change to directory where you would like to download pyrevolve to. You can delete the pyrevolve directory after finishing this step.

git clone `https://github.com/opesci/pyrevolve.git`
cd pyrevolve/
python setup.py install

9. Install mpi4py:
`http://mpi4py.scipy.org/docs/`
pip install mpi4py

10. To enter this environment:
source activate fenics

11. To exit:
source deactivate fenics

1.2 Installing tlm_adjoint

1. Clone the git repository to the local drive where you want it to live:
git clone `https://github.com/jrmaddison/tlm_adjoint.git`

1.3 Installing Fenics Ice

1. Clone the git repository to the local drive where you want it to live:
git clone `https://github.com/cpk26/fenics_ice.git`

1.4 Creating environment variables

Create an environment variable storing the fenics_ice base directory by adding the following
to .bashrc, amending the path appropriately for your system.
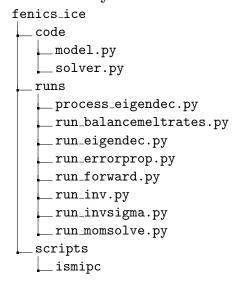FENICS_ICE_BASE_DIR="/XXXX/XXXX/XXXX/fenics_ice"
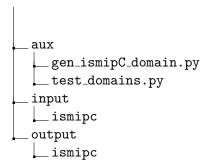export FENICS_ICE_BASE_DIR

1.5 Modifying the Python Path

Modify the default paths python looks for modules to include tlm_adjoint and fenics ice.
Add to the end of .bashrc:
PYTHONPATH=¨${PYTHONPATH}:/PATH/TO/tlm_adjoint/python:/PATH/TO/fenics_ice/code¨
export PYTHONPATH

2 Program structure

2.1 Directory Structure

```
fenics_ice
├── code
│   ├── model.py
│   └── solver.py
├── runs
│   ├── process_eigendec.py
│   ├── run_balancemeltrates.py
│   ├── run_eigendec.py
│   ├── run_errorprop.py
│   ├── run_forward.py
│   ├── run_inv.py
│   ├── run_invsigma.py
│   └── run_momsolve.py
└── scripts
    └── ismipc
```

```
    ├── aux
    │   ├── gen_ismipC_domain.py
    │   └── test_domains.py
    ├── input
    │   └── ismipc
    └── output
        └── ismipc
```

2.2 Overview

The core of the ice sheet model is in two files: `/code/model.py` and `/code/solver.py`. These are utilized by the python scripts in the `/runs` folder, which execute specific parts of a simulation. The python scripts there are generic to any simulation. Each new simulation then has its own primary folder in the `/scripts` folder, with simple bash scripts which call program files in `/runs` with specific parameters and data files.

The bash scripts in `/scripts` are where parameters and data file locations are specified; these are bash simple wrapper scripts for calling python scripts in `/runs`. The data and parameters are used by the program files in `/runs` to create a model object (via a class defined in `model.py`) and subsequently a solver object (via a class defined in `solver.py`). The model object contains all the necessary data for a simulation, such as topography, constants, and velocity observations for inversions. The solver object contains the ice sheet physics/inversion code. The model object is passed as a parameter to your solver object. This object then allows you to solve the SSA equations on your domain, invert for basal drag or B_{glen}, and perform uncertainty quantification. The options of any python script in the `/runs` folder can be viewed by typing 'python run_xxx.py –help'.

The `/aux` folder contains auxillary files; in here, the file `gen_ismipC_domain.py` generates the ismipC domain, based off definitions in `test_domains.py`. The `/input` folder is where input files, such as topography and ice thickness, for specific simulations are located. Similarily, the `/output` folder is where output is stored from specific simulations.

3   Tutorial: A Walkthrough of IsmipC

The Ice Sheet Model Intercomparison Project for Higher-Order ice sheet Models (ISMIP) provides a standardized set of idealized tests for ice sheet models. In this walkthrough, we apply Fenics_ice to the domain prescribed by experiment C (IsmipC). The original IsmipC is a static simulation, meaning time evolution is not considered. We'll extend it by running a dynamic simulation for the purposes of performing uncertainty quantification.

3.1 Generating the Domain

Navigate to the **/fenics_ice** base directory. Activate the fenics conda environment.

```
> source activate fenics
```

To begin, we'll generate the synthetic domain defined by the IsmipC experiment. The specifications are coded in the file **/aux/test_domains.py**. We'll use the python script 'gen_ismipC_domain.py' to create a domain with a given length and resolution.

```
> cd $FENICS_ICE_BASE_DIR/aux
> python gen_ismipC_domain.py -o ../input/ismipC -L 40000 -nx 100 -ny 100
```

This will generate a square domain with side-length 40km, at resolution of 100 x 100, placing the output in the folder **input/ismipc**. Let's observe the files that are generated.

```
> ls $FENICS_ICE_BASE_DIR/input/ismipC
B2.xml  Bglen.xml  alpha.xml  bed.xml  bmelt.xml  grid_data.npz  mask.xml  mesh.xml
 smb.xml  thick.xml
```

The .xml files contain discretized scalar fields over the IsmipC domain on a FEniCS mesh. The extension .npz indicates a numpy file format, and contains the domain resolution and length.

- B2.xml – β^2 coefficient for linear sliding law ($\boldsymbol{\tau_b} = \beta^2\boldsymbol{u}$);

- Bglen.xml – parameter in Glen's flow law

- alpha.xml – variable in sliding law

- bed.xml – basal topography

- bmelt.xml – basal melt.

- mask.xml – mask of our domain

- mesh.xml – FEniCS mesh

- smb.xml – surface mass balance

- thick.xml – ice thickness

3.2 Solving the Momentum Equations

Having generated the files which describe our domain, we can solve the SSA momentum equations to determine ice velocities.

```
> cd $FENICS_ICE_BASE_DIR/scripts/ismipc/
> ./forward_solve.sh
```

```
Generating new mesh
Building point search tree to accelerate distance queries.
Computed bounding box tree with 39999 nodes for 20000 points.
Solving nonlinear variational problem.
Newton iteration 0: r (abs) = 1.585e+03 (tol = 1.000e-08) r (rel) = 1.000e+00
(tol = 5.000e-02)
Newton iteration 1: r (abs) = 1.139e+02 (tol = 1.000e-08) r (rel) = 7.186e-02
(tol = 5.000e-02)
Newton iteration 2: r (abs) = 1.307e+02 (tol = 1.000e-08) r (rel) = 8.248e-02
(tol = 5.000e-02)
Newton iteration 3: r (abs) = 9.443e+01 (tol = 1.000e-08) r (rel) = 5.958e-02
(tol = 5.000e-02)
Newton iteration 4: r (abs) = 5.682e+01 (tol = 1.000e-08) r (rel) = 3.585e-02
(tol = 5.000e-02)
Newton solver finished in 5 iterations and 5 linear solver iterations.
Solving nonlinear variational problem.
Newton iteration 0: r (abs) = 6.650e+01 (tol = 1.000e-05) r (rel) = 1.000e+00
(tol = 1.000e-05)
Newton iteration 1: r (abs) = 4.913e+00 (tol = 1.000e-05) r (rel) = 7.387e-02
(tol = 1.000e-05)
Newton iteration 2: r (abs) = 4.393e-02 (tol = 1.000e-05) r (rel) = 6.606e-04
(tol = 1.000e-05)
Newton iteration 3: r (abs) = 5.647e-06 (tol = 1.000e-05) r (rel) = 8.492e-08
(tol = 1.000e-05)
Newton solver finished in 4 iterations and 4 linear solver iterations.
Time for solve: 4.667648553848267

ls $FENICS_ICE_BASE_DIR/input/ismipC/momsolve
...
```

The script automatically places the output in the subdirectory of `input/` since we'll use
the velocities we solved for in the next step, generating synthetic observations.

Opening `forward_solve.sh` with any text editor, we can confirm that this is a simple
wrapper script.

```
#!/bin/bash
set -e

BASE_DIR=$FENICS_ICE_BASE_DIR
RUN_DIR=$BASE_DIR/runs
```

```
INPUT_DIR=$BASE_DIR/input/ismipC
OUTPUT_DIR=$INPUT_DIR/momsolve

cd $RUN_DIR

python run_momsolve.py -b -q 0 -d $INPUT_DIR -o $OUTPUT_DIR
```

The bash script specifies key folders, that we are solving momentum equations on a domain with periodic boundary conditions (-b option), and that we are using a linear sliding law (-q 0).

3.3   Generating Synthetic Observations

IsmipC is a synthetic experiment, meaning we don't have observational data of ice velocities. We can generate pseudo-oberservations by adding gaussian noise to the solved velocities. We'll assume the noise is additive rather than a multiplicative factor.

The python script 'Uobs_from_momsolve.py' takes the vector field in U.xml and generates the files: data_mesh.xml, data_mask.xml, u_obs.xml, v_obs.xml, u_std.xml, and v_std.xml. The first file contains the data mesh, the second identifies where velocity data is available, the next two files contain the pseudo-observations in the x and y directions, with the final two files containing the standard deviation of the gaussian noise applied. A data_mask.xml and data_mask.pvd file are generated by forward_solve.sh since these are duplicates of the domain mask.

```
> cd $FENICS_ICE_BASE_DIR/aux/
> python Uobs_from_momsolve.py -b -L 40000 \
-d $FENICS_ICE_BASE_DIR/input/ismipC/momsolve
> find $FENICS_ICE_BASE_DIR/input/ismipC/momsolve \
-type f -regex '.*\(obs\|std\).xml'
/mnt/c/Users/ckozi/Documents/Python/fenics/fenics_ice/input/ismipC/u_obs.xml
/mnt/c/Users/ckozi/Documents/Python/fenics/fenics_ice/input/ismipC/u_std.xml
/mnt/c/Users/ckozi/Documents/Python/fenics/fenics_ice/input/ismipC/v_obs.xml
/mnt/c/Users/ckozi/Documents/Python/fenics/fenics_ice/input/ismipC/v_std.xml
```

Copy the five files generated into $FENICS_ICE_BASE_DIR/input/ismipC/. A study site in Antarctica or Greenland would require generating these files and the data_mask from a surface velocity dataset such as NSIDC MEaSUREs.

```
> cd $FENICS_ICE_BASE_DIR/input/ismipC/momsolve
> cp data_mask.xml data_mesh.xml mask_vel.xml u_*.xml v_*.xml ..
```

3.4   Uncertainty Quantification

Fenics-ice is developed with a view of understanding uncertainty in ice sheet models. Because inversions for key ice-sheet model variables (basal drag and B_{glen}) solve underdetermined systems of equations, the solutions may have large, and spatially varying error distributions. The novel capability of fenics-ice is to propogate this uncertainty through a forward stepping simulation, allowing us to calculate the probability distribution of a quantity-of-interest through time, rather than a point estimate.

The bash scripts starting with the prefix 'uq_' in `scripts/ismipc` perform the uncertainty quantification process. The suffixes relate to various values of parameters. Uncertainty quantification can be seperated into four parts, corresponding to to different python scripts in the `runs/` folder.

Beyond the specification of parameters and data sources, the uncertainty quantification scripts call the following (in order):

1. `run_inv.py` – Invert for basal drag

2. `run_forward.py` – Timestep the simulation forward in time

3. `run_eigendec.py` – Run the eigen-decomposition of the Hessian Matrix – allowing us to multiply by the inverse of the covariance matrix of basal drag.

4. `run_errorprop.py` – Run the errorpropogation code to calculate the uncertainty in a quantity of interest through time arising from uncertainty in the inverted values of basal drag

Steps (2) and (3) are independent of each other, and only depend on step (1). Hence their order can be switched.

There are five scripts pertaining to uncertainty quantification in `scripts/imsipc/`.

1. `uq_rc_1e4.sh`

2. `uq_rc_1e6.sh`

3. `uq_30x30.sh`

4. `uq_40x40.sh`

5. `run_all.sh`

The first two bash scripts specify simulations with different levels of regularization for the inversion. Script `uq_rc_1e4.sh` has less regularziation and results in high frequency features in basal drag. The script `uq_rc_1e4.sh` increases the level of regularization by two orders of magnitude, so that the inverted basal drag field mirrors the specification by IsmipC. Grid resolution is the focus of the next two scripts. The scripts `uq_30x30.sh` and

`uq_40x40.sh` use the regulariztion of simulation `uq_rc_1e6.sh`, but increase the resolution to 30x30 and 40x40 respectively. The final script – `run_all.sh` – simply runs the other four scripts.

We'll proceed with script `uq_rc_1e6.sh`. To run uncertainty quantification of IsmipC, simply call the bash script:

```
> cd $FENICS_ICE_BASE_DIR/scripts/ismipc/
> ./uq_rc_1e6.sh
...
```

There will be a significant amount of output.

On a Dell XPS laptop with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz and 32GB of RAM, running Python via the Windows 10 Linux Subsystem (WSL v1.0) the script had the following timings:

```
real        8m27.972s
user        32m10.906s
sys         21m14.797s.
```

Let's break down the contents of the `uq_rc_1e6.sh` script.

The script begins by defining the locations of inputs and outputs.

```
BASE_DIR=$FENICS_ICE_BASE_DIR
RUN_DIR=$BASE_DIR/runs

INPUT_DIR=$BASE_DIR/input/ismipC
OUTPUT_DIR=$BASE_DIR/output/ismipC/ismipC_inv6_perbc_20x20_gnhep_prior
EIGENDECOMP_DIR=$OUTPUT_DIR/run_forward
FORWARD_DIR=$OUTPUT_DIR/run_forward

EIGFILE=slepceig_all.p
```

The variable `OUTPUT_DIR` should be unique to this specific simulation. The other directories are standard and do not need to be modified. The variable `EIGFILE` specifies the name of the file where the output of eigendecomposition is stored.

The current name reflects the fact that the eigenvalues and eigenvectors where calculated using the library SLEPc, and that all eigenvectors/values where calculated. Future releases plan to offer additional libraries to solve the eigenvalue problem. For large domains, calculating all eigenvectors/values is not necessary, nor feasible.

Next in the script we define the values of parameters that will be used as command line

9

arguments. These will be discussed in the context of the python script they're applicable to.

```
RC1=1.0
RC2=1e-2
RC3=1e-2
RC4=1e6
RC5=1e6


T=30.0
N=120
S=5


NX=20
NY=20


QOI=1
```

The core of the script are the following lines. To see a complete list of options for each of the these python scripts, execute them with the '–help' flag (e.g. `python run_inv.py --help`).

```
cd $RUN_DIR


python run_inv.py -b -x $NX -y $NY -m 200 -p 0  -r $RC1 $RC2 $RC3 $RC4 $RC5
-d $INPUT_DIR -o $OUTPUT_DIR
python run_forward.py -t $T -n $N -s $S -i $QOI -d $OUTPUT_DIR -o $FORWARD_DIR
python run_eigendec.py -s -m -p 0   -d $OUTPUT_DIR -o $EIGENDECOMP_DIR -f $EIGFILE
python run_errorprop.py -p 0 -d $FORWARD_DIR -e $EIGENDECOMP_DIR -l $EIGFILE
-o $FORWARD_DIR
```

The first python script `run_inv.py` performs the inversion. The `-b` flag indicates that periodic boundary conditions should be applied at the domain boundary. Currently periodic boundary conditions cannot be specified on individual boundaries, but rather for the entire domain . Each of `-x` and `-y` specify the resolution in the number of cells in the x and y directions. At the present stage, resolution needs to be uniform in both axis. The `-m` option specifies that a maximum of 200 iterations of gradient descent be performed to minimize the cost-function, while the input -p 0 indicates we are optimizing basal drag. In the case of IsmipC, B_{glen} is assigned a constant value. Scaling constants in the cost function for the inversion are specified by the `-r` option. The first value scales the velocity misfit, RC2 and RC4 apply to the regularization of alpha, RC3 and RC4 apply to the regularization of beta. RC2 and RC3 specify the delta parameters in the cost function, and RC4 and RC5 specify the gamma parameters. The options `-d` and `-o` specify input and output directoies.

The second python script `run_forward.py` numerically integrates the simulation forward in time and calculates the adjoint of the quantity of interest with respect to the specified variable. The `-t` option determines the number years to run the simulation for, with `-n` number of timesteps. The $-s$ parameter specifies the numer of sensitivities to calculate. If the value is 1, than the sensitivity at the last timestep is calculated. Otherwise they are calculated at np.linspace(0, run_length, number_of_sensitivites). The sensitivities of a quantity of interest are calculated. Here, we specify the quantity of interest as the integral of the height squared with the `-i 1` option, as the IsmipC simulation is mass-conserving due to the periodic-boundary conditions. The other available option, suitable to real life domains, is volume above floation. Again, the options `-d` and `-o` specify input and output directories.

The third python script `run_eigendec.py` eigendecomposes the Hessian of the inversion cost function. The `-s` flag specifies that the SLEPc library should be used, presently the only functioning option. To consider only the velocity misfit portion of the cost function, we set the `-m` flag. As for `run_inv.py`, we set `-p 0` to consider only basal drag. The remaining options specify the input and output locations.

Lastly, the fourth python script `run_errorprop.py` assembles the output of `run_forward.py` and `run_errorprop.py` to determine the standard deviation of the quantity of interest through time. The options for this script duplicate those above.

3.5 Plotting

real	78m36.278s
user	337m34.531s
sys	214m18.828s