Fenics Ice Sheet Model User Guide

April 15, 2020

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

Contents

1 Introduction

Ice sheet models are important tools for not only generating knowledge, but also for operational forecasts. In this way, they are analagous to weather models and oceanographic models. Ice sheets currently contribute significantly to sea level rise, and this contribution is expected to increase over the coming century. With over 7 m of sea level rise equivalent stored in the Greenland Ice Sheet, and over 50 m sea level rise equivalent in the Antarctic Ice Sheet, forecasts of ice sheet evolution are crucial to informing our response to climate change. Accurately constraining the predicted ice mass loss, as well as the uncertainty in the predictions, is important for assessing risk and implementing succesful mitigation/adaption strategies.

Most models in glaciology approach the ice sheet forecasting problem from a deterministic perspective. That is, given a set of inputs, the model will produce a single output. The evolution of this approach is to consider the problem probabilistically, calculating the probability distribution of possible answers. Knowing the posterior probability distribution allows us to consider the relative likelihood of different outcomes and establish credible intervals for outputs. Fenics Ice is an ice sheet model developed from a Bayesian statistics perspective, providing uncertainty quantification of relevant quantitites such as mass loss.

Large scale models pose unique challenges for probabilistic modelling. A key problem is the computation expense of running large scale models. Ice sheet models requiring solve a highly non-linear set of equations over large domains. Ice is commonly modelled as a viscous, creeping, incompressible fluid, shear thinning fluid, where viscosity is a function

of both temperature and strain rate. Simplifications of the stokes equations based on phsyical assumptions are often used, including in Fenics Ice, for computational reasons. Real running time of a model over a relevant domain in Antartica can be on the order of hours - days, even with specialized solvers, parallelized programming, and variable resolution. A second key problem is the extremely high dimensionality of ice sheet models. The number of parameters can range from 10^5 - 10^9 depending on the model approximation, domain size, and grid resolution. These problems are not unique to glaciology, and are also encountered in models such as mantle convection, weather forecasting, and ocean circulation.

Computational expense and high dimensionality prohibit the effective use of monte carlo methods for large scale models. Two alternative approaches developed in the scientific community are Ensemble Kalman Filters and Variational Bayes. Ensemble Kalman Filters sample the posterior probability distribution using an ensemble of forecast states, estimating the forecast uncertainty based on the sampled uncertainty. The challenge is to suitably select the ensemble of model runs. Variational methods estimate the forecast uncertainty based on propogating uncertainty through the model by linearizing the model and using the adjoint. The challenge here is that that the model must be able to be automatically differentiatied.

Fenics Ice approaches uncertainty quantification of ice sheet models using the variational approach. The implementation mirrors that of [**?**], who developed a framework for end-to-end uncertainty quantification of problems consisting of both data assimilation and forward run. This involves 1) inferring unobserved model parameters from data; 2) determining the uncertainty of the inferred model parameters; 3) running the forward model to make a prediction about a quantity (e.g. ice mass loss); 4) propogating the uncertainty in inferred parameters to the model prediction. The key assumptions made in this approach are that probability distributions are Gaussian, and that the forward model can be reasonably approximated by a first-order Taylor approximation.

The main glaciology problem Fenics Ice sets out to address is how the uncertainty in inverted basal drag and β_{glen} (temperature dependent coefficient in stress-strain relationship) affect the model forecast. The aim is to understand when the uncertainty in the forecast is on the same order of magnitude as the forecast, determining a forecasting horizon. The same framework allows us to understand how uncertainty in quantities such as bed topography or ice thickness propogate through the model. Another question Fenics Ice is coded to investigate is the question of next-best-measurement. That is, given one additional measurment in a system, where would be the most informative location to place it? Fenics Ice is not limited to these questions, and can easily be adapted to address a variety of research questions.

Fenics Ice leverages the open source FEniCS computing platform for solving partial differential equations via finite element methods [**?**]. FEniCS does not much of the computational heavy lifting, including mesh generation, implementing appropriate function spaces, and

finite element assembly. Interfaces with PETSc and SLEPc provide efficient non-linear solvers and eigendecomposition algorithms. To generate the adjoint of the ice sheet model, and compute Hessian actions, Fenics Ice uses the tlm adjoint package [**?**].

2 Installation

The Fenics Ice model 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 script install.sh will attempt to create a suitable conda environment, install fenics_ice and test the installation. Manual installation instructions are provided below.

2.1 Installing Fenics

The simplest way to install FEniCS and tlm-adjoint is to create a conda environment.

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:
conda 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 seaborn

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:
conda activate fenics

11. To exit:
source deactivate fenics

2.2 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`

2.3 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`

2.4 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

2.5 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

3 Program structure

3.1 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.

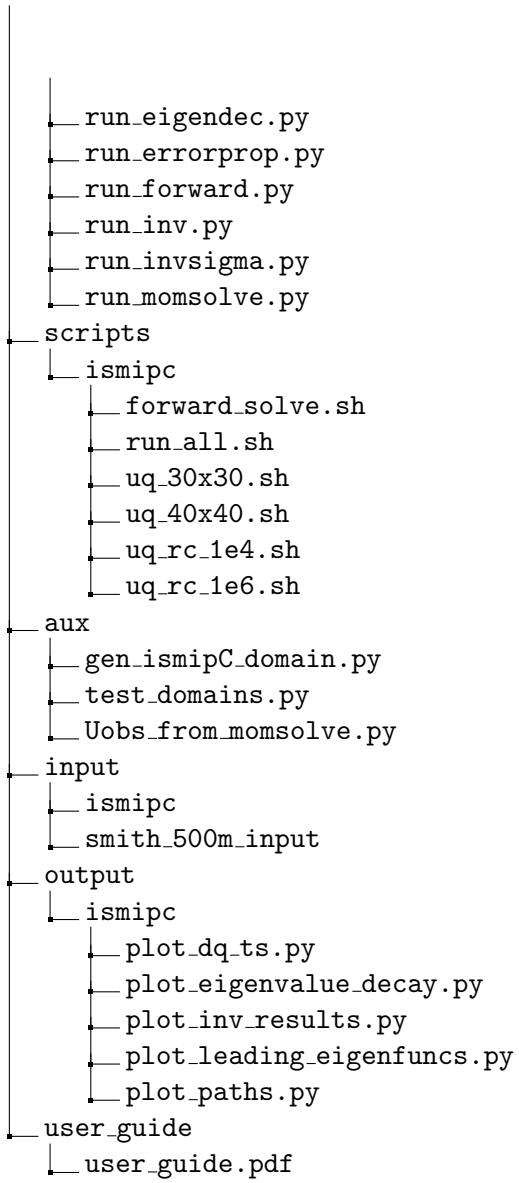
3.2 Directory Structure

The complete set of files and directories provided in the FenicsIce repository can be viewed online, using a file explorer, or with the following git command:

```
>git ls-tree -r HEAD --name-only
```

The core structure and key files are:

```
fenics_ice
├── code
│   ├── model.py
│   └── solver.py
└── runs
    ├── process_eigendec.py
    └── run_balancemeltrates.py
```



4 Tutorial: A Walkthrough of IsmipC

The Ice Sheet Model Intercomparison Project for Higher-Order ice sheet Models (ISMIP-HOM) provides a standardized set of idealized tests for ice sheet models. In this walkthrough, we apply FenicsIce to the domain prescribed by experiment C (IsmipC). A description of IsmipC is provided in Section **??** of this user guide. 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.

4.1 Generating the Domain

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

```
> conda 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 a grid resolution of 100 x 100 cells, 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  data_mask.xml  data_mesh.xml
 grid_data.npz 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

4.2 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/`. 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).

4.3 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: vel_mask.xml, u_obs.xml, v_obs.xml, u_std.xml, and v_std.xml. The first file 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. In this case the standard deviatiation has a constant value of 1.0.

```
> 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 from a surface velocity dataset such as NSIDC MEaSUREs.

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

4.4 Uncertainty Quantification

FenicsIce is developed with an aim of understanding uncertainty in ice sheet simulations. Because inversions for key ice-sheet model variables (basal drag and B_{glen}) solve under-determined systems of equations, the solutions may have large, and spatially varying error distributions. The novel capabality of FenicsIce 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 five 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 a hessian matrix – allowing us to multiply by the inverse of the covariance matrix of basal drag.
4. `run_errorprop.py` – Run the error-propogation code to calculate the uncertainty in a quantity of interest through time arising from uncertainty in the inverted values of basal drag
5. `run_invsigma.py` – Calculate the spatial distribution of the standard deviation of the inverted quantity.

Steps (2) and (3) are independent of each other, and only depend on step (1). Hence their order can be switched. Step (4) depends on (1)-(3), while step (5) depends on (1) and (2).

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_1e6.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.

These can be obtained in linux shell by executing `time ./uq_rc_1e6.sh`. 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 were calculated using the library SLEPc, and that all eigenvectors/values where calculated. Future re-

leases 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 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
python run_invsigma.py -p 0 -d $FORWARD_DIR -e $EIGENDECOMP_DIR -k $EIGENVECTOR_FILE
-l $EIGENVALUE_FILE -d $OUTPUT_DIR -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 RC5 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.

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

Lastly, the fifth python script `run_invsigma.py` processes the output of `run_eigendec.py` to determine the standard deviation of the inverted value across the domain. The options for this script duplicate those above.

4.5 Plotting

This section will go through plotting the results of the IsmipC experiments. It assumes you ran all the simulations in `ismipc/scripts/`. If not, you'll need to execute `run_all.sh` in that folder. The timings on the same machine as previously are:

real	79m31.631s
user	350m12.844s
sys	215m26.250s

4.5.1 Inversion Results

The first plot we'll create allows us to examine the inversion results. We'll do this by running the python script `plot_inv_results.py`. You can modify the simulation and output location at the top of the script. The default simulation is `uq_rc_1e6` and the default output location is a the folder `$FENICS_ICE_BASE_DIR/output/ismipC/uq_rc_1e6/plots`. The script will create the a file named `inv_results.pdf` there.

```
>cd $FENICS_ICE_BASE_DIR/output/ismipC/
>python plot_inv_results.py
```

There are five panels in the output plot. The inverted basal drag from the inversion is shown in panel **(a)**, and the uncertainty in panel **(b)**. Panels **(c)**-**(e)** visualize how well the inversion recreates the pseudo-observed velocities.

Figure **??** displays the results with higher regularization (`uq_rc_1e6`) while Figure **??** shows the results with lower regularization. Observe that less regularization results in higher frequencies in the pattern the basal drag coefficient.

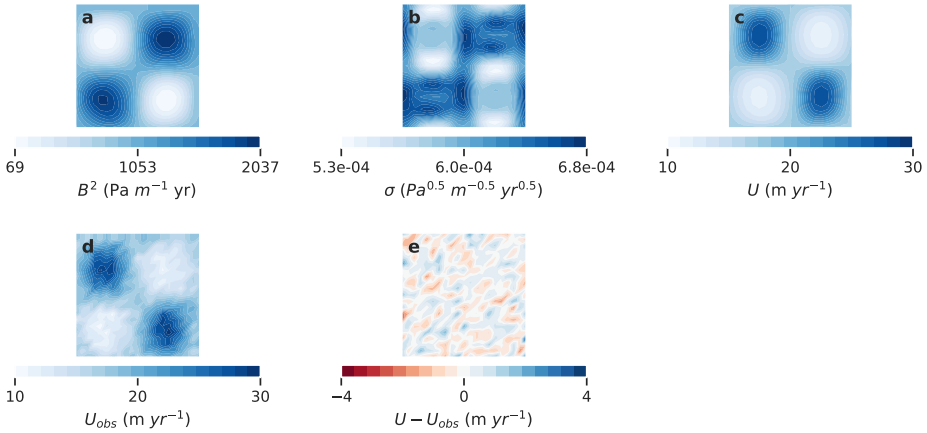


Figure 1: IsmipC inversion results for higher regularization. **(a)** linear coefficient in basal sliding law; **(b)** standard deviation of alpha (defined in this experiment as the square root of the linear drag coefficient); **(c)** modelled ice velocities using inverted basal drag; **(d)** pseudo-observed ice velocities, consisting of the solution to IsmipC and addtitive gaussian noise; **(e)** difference between ice velocities using inverted basal drag and observed ice velocities;

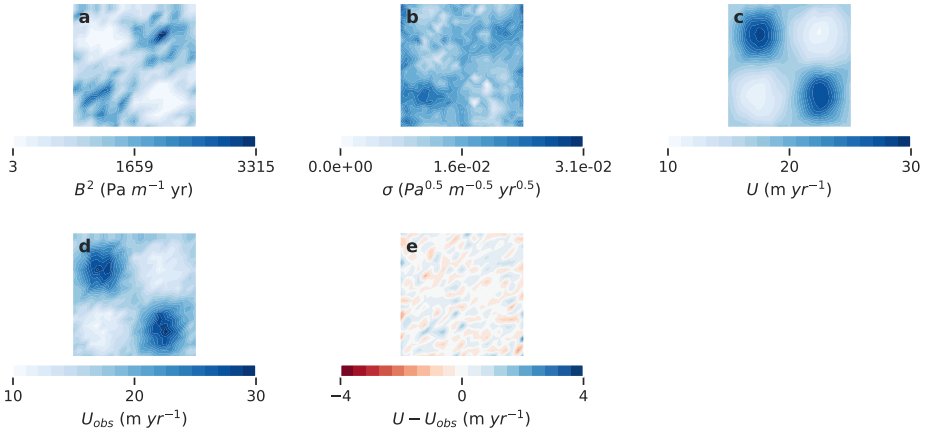


Figure 2: IsmipC inversion results for lower regularization. Panels as above.

4.5.2 Eigenvectors

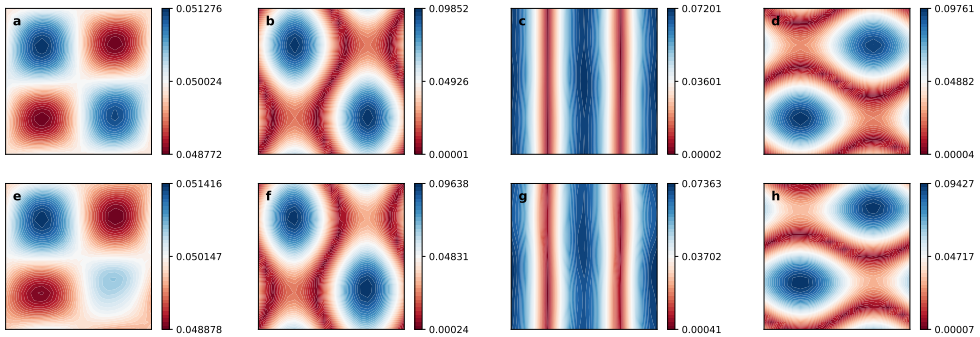


Figure 3: Leading four constrained modes of the inverted basal drag.

Which modes of the basal drag are well constrained by the data? These are described the eigendecomposition we performed earlier. Let's plot them for experiments `uq_rc_1e6` and `uq_rc_1e4`. The command required is:

```
>python plot_leading_eigenfuncs.py
```

At the top of the file you can specify the simulation folders and the output location. The default output folder is `output/ismipC/plots`. Four eigenvectors are plotted by default, with the parameter `e_offset` specifying the first eigenvector to plot. The order of the eigenvectors corresponds to how well constrained they are.

Two figures are shown. In each figure, the top row corresponds to simulation `uq_rc_1e6`, and the bottom panel to simulation `uq_rc_1e4`. The first four eigenvectors are shown in Figure **??**, while eigenvectors 30-33 are shown in Figure **??**. Set the parameter `e_offset` to 30 to reproduce the second plot. Observe that the top constrained modes between simulations are nearly identical. Also notice that well constrained modes correspond to lower frequencies.

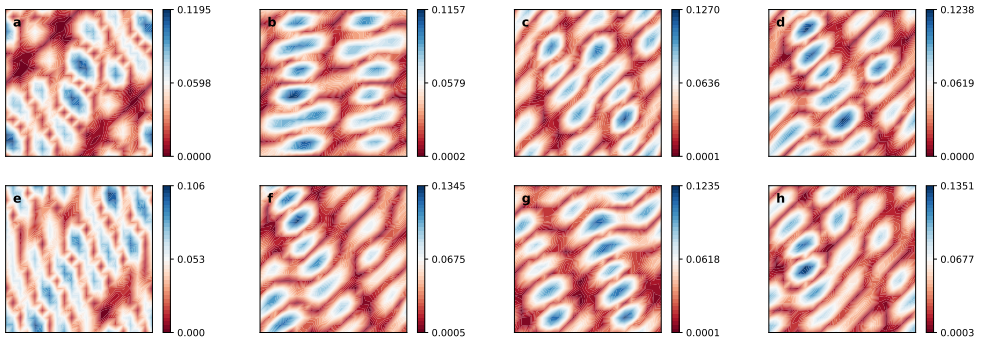


Figure 4: Eigenvectors 30-33, showing less well constrained modes of basal drag.

4.5.3 Eigenvalues

How well constrained are the eigenvectors we plotted previously? This information is given by the corresponding eigenvalues (Figure **??**). We'll plot the eigenvalues for the simulations: `uq_rc_1e6`, `uq_30x30`, and `uq_40x40` – which differ only in their resolution. The command is:

```
>python plot_eigenvalue_decay.py
```

Observe how the eigenvalues quickly drop-off in their magnitude, and overlay in each other. This reflects the fact that we expect the same low frequencies modes to be well constrained across different grid resolutions.

4.5.4 Quantity of Interest Probability Distribution through time

The quantity of interest for IsmipC was defined as the integral of ice thickness squared over the domain. Using uncertainty quantification techniques, FenicsIce determines not only a point estimate of the quantity of interest through time, but also a standard deviation (based on an assumption that it is distributed normally). Let's compare the results for simulations `uq_rc_1e4` and `uq_rc_1e6`:

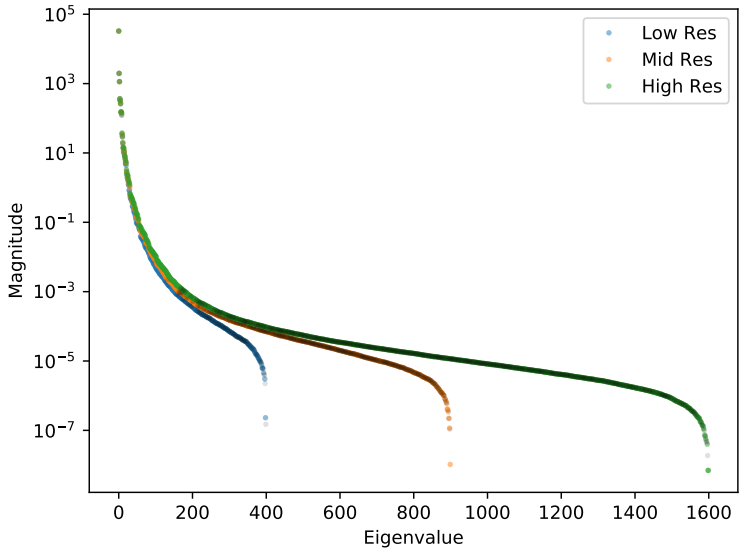


Figure 5: Eigenvalues of different modes of basal drag for IsmipC at low (20x20), medium (30x30), and high resolutions (40x40). Black eigenvalues correspond to negative values; they begin to appear at eigenvalues several orders of magnitude below the leading eigenvalues. Leading eigenvalues at different resoultions closely overlay each other.

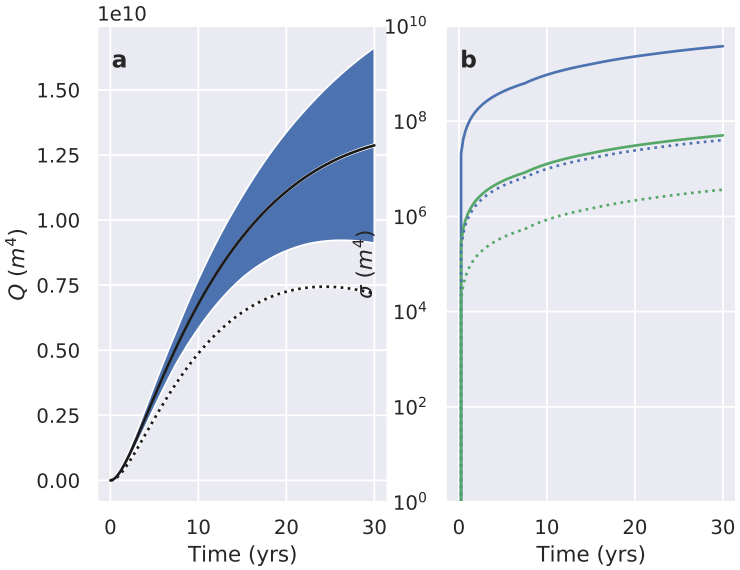


Figure 6: Probability distribution of the quantity of interest through time. The quantity of interest for IsmipC is the integral of height squared over the domain. **(a)** quantity of interest and a 2-sigma envelope. Dashed line shows simulation `uq_rc_1e6` and the solid line shows simulation `uq_rc_1e4`. The blue prior only envelope for `uq_rc_1e4` is the only one visible. **(b)** 2-sigma values for the quantity of interest through time. Solid line corresponds to `uq_rc_1e4` while dashed line indicates `uq_rc_1e6`. Blue lines correspond to the prior-only standard deviation, while the dashed line shows the standard deviation after data assimilation.

```
>python plot_paths.py
```

The plots show the 2-sigma envelope if we consider only the prior (regularization), as well as the prior plus information from the data. While the prior-only envelope for simulation `uq_rc_1e4` shows large uncertainty on the order of the quantity of interest, increasing the regularization by two orders of magnitude (`uq_rc_1e4`) collapses the uncertainty by 2-3 orders. When data is taken into consideration, it is clear that the basal drag modes relevant to the quantity of interest are well constrained.

4.5.5 Quantity of Interest with respect to alpha

FenicsIce can calculate how the quantity of interest at a given point in time depends on basal drag (parameterized by alpha) ($\frac{dQOI_t}{d\alpha}$). With this we can understand which parts of the basal drag field are the most important in determing the quantity of interest. To make this plot using the final timestep, run the following:

```
>python plot_dq_ts.py
```

5 IsmipC

Symbol	Constant	Value	Units
A	Ice-flow parameter	10^{-16}	$\mathrm{Pa^n\,yr^{-1}}$
ρ_i	Ice Density	910	$\mathrm{kg\,m^{-3}}$
g	Gravitational constant	9.81	$\mathrm{m\,s^{-2}}$
n	Exponent in Glen's Flow law	3	
t_y	Seconds per year	31556926	$\mathrm{s\,yr^{-1}}$

Table 1: Constants for ISMIP-HOM experiments

Ice Sheet Model Intercomparison Project for Higher-Order ice sheet Models (ISMIP-HOM) is a set of standardized simulations used in the glaciology community for model intercomparison. Due to its familiarity, and simple setup, Experiment C was selected for the tutorial in this user-guide. It allows many aspects of uncertainty quantification to be explored in a simple domain.

The domain of Experiment C is a square domain with periodic boundary conditions on all four boundaries. The surface and basal topography are prescribed as:

$$s(x, y) = -x \cdot tan(0.1^\circ) \tag{1}$$

$$b(x, y) = s(x, y) - 1000 \tag{2}$$

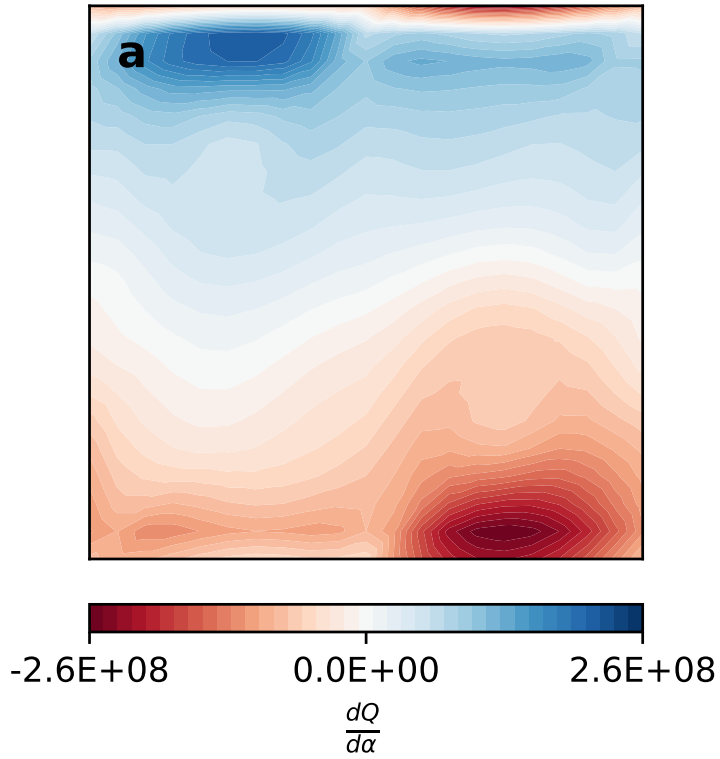


Figure 7: The derivative of the quantity of interest at the last timestep with respect to alpha.

In Experiment C, basal drag is parameterized with a linear sliding law, with the drag coefficient prescribed as:

$$\beta = [1000 + 1000sin(\omega x)cos(\omega y)] \cdot t_y^{-1} \tag{3}$$

where t_y is the number of seconds in a year, converting β to SI units.

6 Publications

This section will be updated as publications with Fenics Ice appear.