Concepts and capabilities of the Instructed Glacier Model (IGM v2.1.0)

G. Jouvet and al[*]

November 14, 2023

Abstract

We present the concept and capabilities of IGM (https://github.com/jouvetg/igm), a Python-based modeling tool designed for efficiently simulating glacier evolution across various scales. IGM integrates ice thermomechanics, climate-driven surface mass balance, mass conservation, and other processes. Within IGM, the update of all physical model components involves a series of mathematical operations on horizontal raster grids, performed by the Tensorflow library. This design choice results in high parallelization capabilities, particularly beneficial when executed on GPU hardware. The most challenging aspect of parallelization within IGM is the ice flow model, which leverages a physics-informed convolutional neural network trained from high-order ice flow physics. Conversely, components like the positive-degree day surface mass balance or the enthalpy thermal scheme take advantage of pixel-wise parallelization. Beyond its computational efficiency, IGM offers a user-friendly coding structure and modularity to promote community development, an OGGM-based module for accessing the data, data assimilation through underlying automatic differentiation tools, and post-processing vizualization routines. We present a comprehensive workflow, which includes data preprocessing, inverse and forward modeling, and rendering of results, enabling the rapid modeling of any mountain glacier globally by providing its RGI ID within a few minutes.

1 Introduction

Glacier evolution models play a crucial role in reconstructing the historical behavior of glaciers and their relationship with past climates. Additionally, they are indispensable for predicting how glaciers will evolve in the future and the consequent rise in sea levels due to climate warming (Pattyn, 2018). Over the last two decades, the glaciological community has dedicated substantial efforts to the development of these models (Zekollari et al., 2022). These models are designed to encompass a wide range of pertinent physical processes, including ice flow, thermodynamics, subglacial hydrology, and their intricate interactions with various factors such as atmospheric conditions (e.g., climate-driven surface mass balance), the Earth's lithosphere, and the ocean (e.g., iceberg calving or subaquatic melting). Prominent examples of such models include Full-Stokes such as Elmer/Ice (Gagliardini et al., 2013) for a generic usage, high-order such that PISM (Winkelmann et al., 2011), CISM (Lipscomb et al., 2019), ISSM (Larour et al., 2012), which are popular in the ice sheet modelling community, and SIA-based such as OGGM (Maussion et al., 2019) or PyGEM (Rounce et al., 2020), which were designed for global glacier modelling. However, the increasing complexity of models based on high-order mechanics comes with rising computational burdens, which can only be adressed using parallel computing.

In recent years, there has been a growing interest in employing Graphics Processing Units (GPUs) to tackle the computational bootleneck in ice flow modeling. GPUs are equipped with a larger number of cores, albeit at slower speeds compared to Central Processing Units (CPUs). GPUs have the potential of overcoming previously mentioned limitations in modeling ice flow and achieve substantial speed improvements (Räss et al., 2020). Effectively harnessing the power of GPUs hinges on the implementation of numerical methods that can be subdivided into numerous parallel tasks, a particularly challenging endeavor when dealing with the viscous behavior of ice and the underlying diffusion equations governing its motion. As far as our knowledge extends, two distinct approaches have been explored to achieve this high level of parallelization. The first approach involves the explicit time integration of the Shallow Ice Approximation (SIA) (Višnjević et al., 2020) and the Second Order SIA (Brædstrup et al., 2014). In contrast, the second approach entails solving the Stokes equations using finite differences, allowing for the utilization of numerical stencil-based techniques (Räss et al., 2020). While programming on GPU was a

[*]This working document intends to be submitted for publication in a model development journal. It is opened for contributions until 30 September 2024. Contributions typically include the development of modules, or the performing and reporting of numerical experiments that assess the capabilities of IGM. All contributors will be listed as co-authors of the final paper. To date, direct contributors are (listed alphabetically): Guillaume Cordonnier, Andreas Henz, Oskar Hermmann, Fabien Maussion, Jürgen Mey, Dirk Scherler, Ethan Welty. Contact `guillaume.jouvet@unil.ch` if you wish to contribute.

relative complex task in the past, the emergence of libraries such as TensorFlow and PyTorch in the popular Python language opens new opportunities for the development of efficient glacier ice flow model at relatively limited technical level.

Capitalizing on recent library for efficient computation on GPU and machine learning techniques, we outline the concept and demonstrate the capability of a Python-based glacier evolution model – the Instructed Glacier Model (IGM, Jouvet et al., 2022) – which couples ice thermomechanics, surface mass balance, and mass conservation. The specificity of IGM is that all physical model components are updated using relatively short sequence of operations on horizontal raster grids performed thanks to the Tensorflow library. As a result, IGM operations are highly parrallelized and therefore run very efficiently on GPU. Model components (such as the surface mass balanceor the Enthalpy models) that do not involve any horizontal diffusion are solvable pixel-wise, and therefore can be solved in parrallel. In contrast, we use a Convolutional Neural Network trained to minimise the energy associated with high-order ice flow physics to overcome solving high-order 3D ice flow partial differential equations and ensure parrallization (Jouvet and Cordonnier, 2023). Using Tensorflow to describe all operations has an other major advange for data assimilation as embedded automatic differentiation tools permits to access all their derivatives, and therefore to perform model inversion at low computational efforts (Jouvet, 2023). Lastly, similarly to OGGM and PyGEM, we have implemented IGM in the widely-used programming language, Python, to make it accessible to a large community of glaciologists and leverage the numerous Python libraries. Additionally, we have designed IGM in a modular fashion to facilitate community development and user customization of the model.

This paper is organized as follows. First, we describe the physical models taken from the litterature and implemented in IGM. Then, we describe the coding concept and the module-wise implementation of IGM. Last, we demonstrate its capabilities by presenting some examples of applications.

2 Model

In the following, we use the notations $b(x,y)$, $s(x,y,t)$, and $h(x,y,t)$ to represent the glacier bedrock, surface, and ice thickness. Here, (x,y) and t denote horizontal coordinates, and the time. We also introduce $\mathbf{u}(x,y,z,t) = (u_x, u_y, u_z)$ as the 3D velocity field of the ice, and $T(x,y,z,t)$ and $\omega(x,y,z,t)$ to represent temperature and water content, respectively.

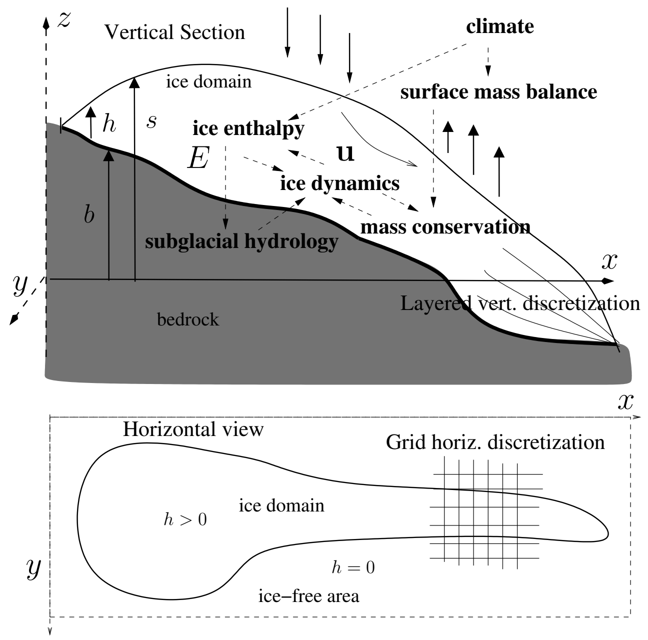

Figure 1: Cross-section and horizontal view of a glacier with notations, spatial discretization, main physical processes governing glacier evolution and their interactions.

2.1 Forward modelling

In the following sections, we describe all the physical models that are included in IGM and govern the evolution of the above-defined variables (Fig. 1). Note that some model components (e.g. the Enthalpy and the subglacial hydrology) may be ignored for simplicity. These models are based on the principles of conservation of mass, momentum and energy.

2.1.1 Mass conservation

The evolution of ice thickness, denoted as $h(x,y,t)$, starting from an initial glacier shape, is governed by mass conservation, which connects elevation change, ice dynamics and surface mass balance (Fig. 1) through:

$$\frac{\partial h}{\partial t} + \nabla \cdot (\bar{\mathbf{u}} h) = \mathrm{SMB}, \tag{1}$$

where symbol $\nabla\cdot$ represents the divergence operator for the horizontal variables (x,y), $\bar{\mathbf{u}} = (\bar{u}, \bar{v})$ denotes the vertically-averaged horizontal ice velocity field, while SMB represents the Surface Mass Balance function.

2.1.2 Ice dynamics

The momentum conservation equation (assuming negligible inertial terms) and the incompressibility condition are expressed as follows:

$$-\nabla \cdot \sigma = \rho \mathbf{g}, \tag{2}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{3}$$

where σ is the Cauchy stress tensor, $\mathbf{g} = (0, 0, -g)$, g is the gravitational constant. Let τ be the deviatoric stress tensor defined by

$$\sigma = \tau - pI, \tag{4}$$

where I is the identity tensor, p is the pressure field, with the requirement that $\mathrm{tr}(\tau) = 0$ so that $p = -(1/3)\mathrm{tr}(\sigma)$. Glen's flow law (Glen, 1953), which characterizes the mechanical behavior of ice, can be formulated as the following nonlinear relationship:

$$\tau = 2\mu D(\mathbf{u}), \tag{5}$$

where $D(\mathbf{u})$ denotes the strain rate tensor defined by

$$D(\mathbf{u}) = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T), \tag{6}$$

μ is the viscosity defined by

$$\mu = \frac{1}{2} A^{-\frac{1}{n}} |D(\mathbf{u})|^{\frac{1}{n}-1}, \tag{7}$$

where $|Y| := \sqrt{(Y:Y)/2}$ denotes the norm associated with the scalar product (:) (the sum of the element-wise product), $A = A(x, y, z, t)$ is the Arrhenius factor and $n > 1$ is the Glen's exponent. Note that A depends on the temperature of the ice (Paterson, 1994) *via* Glen-Paterson-Budd-Lliboutry-Duval law:

$$A(T, \omega) = A_c(T)(1 + 181.25\omega), \tag{8}$$

where $A_c(T)$ is given by the Paterson-Budd law:

$$A_c(T) = A \exp\left(-Q/(R\,T_{pa})\right) \tag{9}$$

where A and Q have different values below and above a threshold temperature:

$$A = 3.985 \times 10^{-13}\, s^{-1} Pa^{-3}, \quad \text{if } T < 263.15\,K \tag{10}$$
$$A = 1.916 \times 10^{3}\, s^{-1} Pa^{-3}, \quad \text{else.} \tag{11}$$

and

$$Q = 60\ \mathrm{kJmol}^{-1}, \quad \text{if } T < 263.15\,K \tag{12}$$
$$Q = 139\ \mathrm{kJmol}^{-1}, \quad \text{else.} \tag{13}$$

For simplicity, we may ignore temperature dependence of A for instance in the case the Enthalpy (Section 2.1.3) is not modelled.

Equations (2) to (7) describe a Stokes problem in which the unknowns are the 3D velocity field $\mathbf{u}$ and the pressure field p. To simplify the problem, we make the "hydrostatic assumption" as described by (Blatter, 1995) and neglect second-order terms in the aspect ratio of the ice domain (thickness versus length) within the strain rate tensor $D(\mathbf{u})$. By doing so and invoking the incompressibility equation, both the vertical velocity components u_z and the pressure p are eliminated from the momentum conservation equation. The resulting model, commonly referred to as the Blatter-Pattyn model (Blatter, 1995), conveniently transforms into a 3D nonlinear elliptic equation solely for the horizontal velocity components. This modification makes it easier to solve compared to the original Stokes model.

The boundary conditions that supplement (2), (3) are the following. Stress free force applies to the ice-air interface,

$$\sigma \cdot \mathbf{n} = 0, \quad p = 0, \tag{14}$$

where $\mathbf{n}$ is an outer normal vector along the surface. Along the lower surface interface, the nonlinear Weertman friction condition (e.g., Schoof and Hewitt, 2013) relates the basal shear stress τ_b to the sliding velocity $\mathbf{u}_b$ as follows:

$$\mathbf{u} \cdot \mathbf{n} = 0, \tag{15}$$
$$\tau_b = -c|\mathbf{u}_b|^{m-1}\mathbf{u}_b, \tag{16}$$

where $m > 0$, $c = c(x, y) > 0$, and $\mathbf{n}$ is the outward normal unit vector to the bedrock.

The sliding coefficient c in (16) is defined with the Mohr-Coulomb law (Cuffey and Paterson, 2010), that involves the effective pressure in the till N_{till} (Section 2.1.4):

$$c = \tau_c u_{th}^{-m} = N_{till} \tan(\phi) u_{th}^{-m}, \tag{17}$$

where ϕ is the till friction angle, and u_{th} is a parameter homegenous to ice velocity following Khroulev and the PISM Authors (2020). In the case the Enthalpy (Section 2.1.3) and the subglacial hydrology (Section 2.1.4) are not modelled, we may simply prescribe a constant sliding coefficient c, or get its spatial distribution by inverse modelling (Section 2.2).

2.1.3 Ice enthalpy

In this section, we model ice enthalpy following Aschwanden et al. (2012). This approach enables us to simultaneously model ice temperature and water content when the temperature reaches the pressure melting point, thereby conserving energy. Ice enthalpy influences the dynamical model in two ways: variations in temperature and water content lead to ice softening or hardening (Eq. (8)), while enthalpy affects basal sliding conditions throught basal till water layer (Eq. (17)). The enthalpy, denoted as E, is a variable defined throughout the ice and is a function of both temperature, T, and water content, ω:

$$E(T, \omega, p) = \begin{cases} c_i(T - T_{\mathrm{ref}}), & \text{if } T < T_{\mathrm{pmp}}, \\ E_{\mathrm{pmp}} + L\omega, & \text{if } T = T_{\mathrm{pmp}},\ 0 \le \omega, \end{cases} \tag{18}$$

where c_i is the heat capacity, T_{ref} is a reference temperature, and L is the latent heat of fusion. Additionally the temperature T_{pmp} and enthalpy E_{pmp} at pressure-melting point of ice are defined by

$$T_{\mathrm{pmp}} = T_0 - \beta p, \tag{19}$$
$$E_{\mathrm{pmp}} = c_i(T_{\mathrm{pmp}}(p) - T_{\mathrm{ref}}), \tag{20}$$

where $T_0 = 273.15$ K is the melting temperature at standard pressure, and $\beta = 7.9 \times 10^{-8}$ K Pa^{-1} is the Clausius-Clapeyron constant.

According to the definition of enthalpy provided above, we have two possible modes: i) When the ice is cold, meaning it is below the melting point, the enthalpy is simply proportional to the temperature minus a reference temperature. ii) When the ice is temperate, the enthalpy continues to increase. In this case, the additional component $L\omega$ accounts for the creation of water content through energy transfer. Therefore, one can infer the value of enthalpy, denoted as E, from both temperature, T, and water content, ω, and *vice-versa*.

The melting point temperature at pressure is adjusted for hydrostatic pressure $p = \rho g d$ using the following equation:

$$T_{pmp} = T_0 - \beta \rho g d, \tag{21}$$

where d represents the depth. Therefore, the "pressure-adjusted" temperature, denoted as T_{pa}, is defined as the temperature with a shift such that its melting point temperature reference is always zero:

$$T_{pa} = T + \beta \rho g z.$$

The enthalpy model consists of the following advection-diffusion equation, with horizontal diffusion being neglected:

$$\rho_i \left(\frac{\partial E}{\partial t} + u_x \frac{\partial E}{\partial x} + u_y \frac{\partial E}{\partial y} + u_z \frac{\partial E}{\partial z} \right) - \frac{\partial}{\partial z}\left(K_{c,t} \frac{\partial E}{\partial z} \right) \tag{22}$$
$$= \phi - \rho_w L D_w(\omega), \tag{23}$$

where ρ_i is the ice density, $K_{c,t}$ equals $K_c = k_i/c_i$ if the ice is cold ($E < E_{pmp}$) or $K_t = \epsilon k_i/c_i$ otherwise. Using Glen's flow law (Eq. (5)), the strain heating ϕ is defined by

$$\phi = D(\mathbf{U})\tau = A^{-1/n}|D(\mathbf{u})|^{1+1/n}. \tag{24}$$

The last source term $-\rho_w L D_w(\omega)$ in (23) permits to remove the water in temperate ice, $D_w(\omega)$ being a drainage function (Aschwanden et al., 2012).

At the top ice surface, the enthalpy equation is constrained by the surface temperature (or equivalently, the enthalpy) provided by the climate forcing, which is enforced as a Dirichlet condition. At the glacier bed, there are multiple boundary conditions for the enthalpy equation (Aschwanden et al., 2012; Wang et al., 2020):

- cold base and dry: $K_c \frac{\partial E}{\partial z} = Q_{\mathrm{geo}} + Q_{\mathrm{fh}}$ if $E_b < E_{\mathrm{pmp}}$ and $W_{till} = 0$,
- cold base and wet: $E_b = E_{\mathrm{pmp}}$ if $E_b < E_{\mathrm{pmp}}$ and $W_{till} > 0$,
- temperate base and cold ice: $E_b = E_{\mathrm{pmp}}$ if $E_b \geq E_{\mathrm{pmp}}$ and $W_{till} > 0$, zero temperate basal layer,
- temperate base, temperate ice: $K_t \frac{\partial E}{\partial z} = 0$ if $E_b \geq E_{\mathrm{pmp}}$ and $W_{till} > 0$, non-zero temperate basal layer,

where Q_{geo} and Q_{fh} are the geothermal heat flux, and the frictional heat flux, respectively. Using Weertmann law (16), the latter is computed as follows:

$$Q_{\mathrm{fh}} = \tau_b \cdot \mathbf{u}_b = c|\mathbf{u}_b|^{m+1}. \tag{25}$$

When the temperature hits the pressure-melting point at the glacier bed (i.e. $E \geq E_{\mathrm{pmp}}$), the basal melt rate is calculated via the following equation:

$$m_b = \frac{1}{\rho_i L}(Q_{fr} + Q_{geo} - K_{t,c}\frac{\partial E}{\partial z}). \tag{26}$$

The basal melt rate is further increased to account for the drainage of the water content generated throughout the entire column (last term of Eq. (23)).

2.1.4 Subglacial hydrology

Following Bueler and van Pelt (2015), the basal water thickness in the till W_{till} is computed from the basal melt rate as follows:

$$\frac{\partial W_{till}}{\partial z} = \frac{m_b}{\rho_w} - C_{dr}, \tag{27}$$

where C_{dr} is a simple drainage parameter. The till is assumed to be saturated when it reaches the value $W_{till}^{max} = 2$ m, therefore, the till water thickness is capped to this value. The effective thickness of water within the till N_{till} is computed from the saturation ratio $s = W_{till}/W_{till}^{max}$ by the formula (Bueler and van Pelt, 2015):

$$N_{till} = \min\left\{ p, N_0 \left(\frac{\delta P}{N_0}\right)^s 10^{(e_0/C_c)(1-s)} \right\}, \tag{28}$$

where p is the ice overburden pressure and the remaining parameters are constant.

2.1.5 Surface mass balance

IGM comes with several surface mass balance models.

The simplest one implements a classical linar relation between Equilibrium Line Altitude (ELA) and altitude:

$$SMB(z) = \min(\beta_{acc}(z - z_{ELA}), m_{acc}) \quad \text{if } z > z_{ELA}, \tag{29}$$

$$SMB(z) = \beta_{abl}(z - z_{ELA}) \quad \text{else,} \tag{30}$$

where z_{ELA} is the ELA, β_{abl} and β_{acc} are ablation and accumulation gradients, and m_{acc} is the maximum surface mass balance.

Second, IGM incorporates a temperature index model (Hock, 2003) to calculate the surface mass balance based on seasonal temperature and precipitation fields. In this model, surface accumulation equals solid precipitation when the temperature is below a threshold (usually 0°C) and decreases linearly to zero in a transition zone. Conversely, surface ablation is computed in proportion to the number of positive degree-days. Given monthly temperature T_i and precipitation P_i spatial fields, the yearly surface mass balance at elevation z is then computed with

$$SMB = \frac{\rho_w}{\rho_i} \sum_{i=1}^{12} \left(P_i^{sol} - d_f \max\{T_i - T_{melt}, 0\} \right), \tag{31}$$

where P_i^{sol} is the monthly solid precipitation, T_i is the monthly temperature and T_{melt} is the air temperature above which ice melt is assumed to occur, d_f is the melt factor, and $\frac{\rho_w}{\rho_i}$ is the ratio between water and ice density.

2.2 Inverse modelling

Inverse modelling (or data assimilation) serves to find optimal ice thickness, top ice surface, and ice flow parameters that align with observational data in a preliminary step. These observations can include surface ice speeds, ice thickness profiles, and top ice surface data. The goal is to find the fields that best explain the observed data while remaining consistent with the ice flow used in the forward modeling (Jouvet, 2023).

The optimization problem consists of finding spatially varying fields (h, c, s) that minimize the cost function

$$\mathcal{J}(h, c, s) = \mathcal{C}^u + \mathcal{C}^h + \mathcal{C}^s + \mathcal{C}^d + \mathcal{R}^h + \mathcal{R}^c + \mathcal{P}^h, \tag{32}$$

where $\mathcal{C}^u$ is the misfit between modeled $\mathbf{u}^s$ and observed $\mathbf{u}^{s,obs}$ surface ice velocities

$$\mathcal{C}^u = \int_\Omega \frac{1}{2\sigma_u^2} \left| \mathbf{u}^{s,obs} - \mathbf{u}^s \right|^2, \tag{33}$$

$\mathcal{C}^h$ is the misfit between modeled and observed h^{obs} ice thickness available profiles:

$$\mathcal{C}^h = \int_\Omega \frac{1}{2\sigma_h^2} |h^{obs} - h|^2, \tag{34}$$

where h^{obs} is a rasterized representation of ice thickness profiles (the pixels with missing data are ignored in the above integral), and $\mathcal{C}^s$ is the misfit between the modeled and observed s^{obs} top ice surface:

$$\mathcal{C}^s = \int_\Omega \frac{1}{2\sigma_s^2} \left| s - s^{obs} \right|^2, \tag{35}$$

where $\mathcal{C}^d$ is a misfit term between the flux divergence and its polynomial regression d with respect to the ice surface elevation $s(x, y)$ to enforce smoothness with dependence to s:

$$\mathcal{C}^d = \int_\Omega \frac{1}{2\sigma_d^2} \left| \nabla \cdot (h\bar{\mathbf{u}}) - d \right|^2, \tag{36}$$

where $\mathcal{R}^h$ is a regularization term to enforce anisotropic smoothness of $b = s - h$ and convexity of h:

$$\mathcal{R}^h = \alpha_h \int_{h>0} \left(|\nabla b \cdot \tilde{\mathbf{u}}^s|^2 + \beta |\nabla b \cdot (\tilde{\mathbf{u}}^s)^\perp|^2 - \gamma h \right), \tag{37}$$

where $\mathcal{R}^c$ is a regularization term to enforce smooth sliding coefficient c:

$$\mathcal{R}^c = \alpha_h \int_{h>0} \left(|\nabla c \cdot \tilde{\mathbf{u}}^s|^2 + \beta |\nabla c \cdot (\tilde{\mathbf{u}}^s)^\perp|^2 \right), \tag{38}$$

where $\mathcal{P}^h$ is a penalty term to enforce nonnegative ice thickness, and zero thickness outside a given mask:

$$\mathcal{P}^h = 10^{10} \times \left(\int_{h<0} h^2 + \int_{\mathcal{M}^{\text{ice-free}}} h^2 \right). \tag{39}$$

Here $\tilde{\mathbf{u}}_s$ is the horizontal modelled surface velocity field $\mathbf{u}_s$ after applying a gaussian smoothing ($\sigma = 3$), and $(\tilde{\mathbf{u}}_s^{obs})^\perp$ is its orthogonal field. Hereabove, we denote σ_u, σ_h, σ_d, σ_s as the user-defined confidence levels (possibly spatially varying) errors of observations for $\mathbf{u}_s^{obs}$, h_p^{obs}, d^{obs}, and s^{obs}, respectively, and $\alpha_h, \gamma, \alpha_c > 0$, $0 < \beta < 1$ are fixed parameters.

The optimization problem presented above closely resembles the one outlined by Jouvet (2023), with two notable enhancements: i) Instead of optimizing a variable that combines the sliding coefficient and the Arrhenius factor, it focuses solely on optimizing the single sliding coefficient c. ii) The regularization, which may be anisotropic, is applied to the bedrock rather than the ice thickness.

Note that the inverse modelling is not designed yet to be compatible with the Enthalpy and subglacial hydrology models.

3 Spatial and time discretization

In IGM, the horizontal modeled domain is assumed to be a rectangle. IGM deals with rastered data defined on a regular grid of dimensions $N_x \times N_y$ and uniform spacing along both the x and y axes (Fig. 1, bottom panel). Key variables such as ice thickness h, surface topography s, or sliding coefficient c are defined on this grid. It's important to note that our choice of a structured grid, rather than any other type of discretization, is crucial for representing variables as 2D arrays. This structure allows us to employ Convolutional Neural Networks (CNN) for emulating the mechanics of ice flow (Section 4.2.3). On the other hand, the discretization of ice thickness occurs vertically using a fixed number of points denoted as N_z (Fig. 1, top panel). These layers can be distributed in a non-uniform manner, e.g. to ensure finer discretization near the ice-bedrock interface, where the steepest gradients are expected, and coarser near the ice-surface interface following the strategy proposed in the Parallel Ice Sheet Model (PISM, Khroulev and the PISM Authors, 2020).

As a glacier evolution model, IGM employs a time-advancing algorithm that permits to update of ice thickness over time. To achieve this, i) the ice flow is computed by physics-informed deep learning (Jouvet and Cordonnier, 2023), ii) the surface mass balance is computed follwing given formula, iii) the enthalpy is solved column-wise by finite difference, and iv) the the mass conservation equation is solved using an explicit first-order upwind finite-volume scheme. For clarity and modularity, IGM separates these steps into distinct modules within the modeling framework.

4 IGM modules

Regardless of the specific application, glacier evolution models fundamentally involve several common tasks. These tasks include loading and optimizing geological and climate-related data, initializing fields that describe the glacier's geometry and thermo-mechanical state, updating these fields through a time iteration loop driven by external forcing, and outputting results at regular time intervals. Recognizing the similarity in the tasks performed by different model components, we have organized IGM in a module-wise fashion. Each module handles a specific aspect of the glacier evolution process, making the model modular and easy to customize.

There exist pre-processing, processing, post-processing IGM modules (Fig. 4), which contains functions for parameter definition, initialization, update and finalization – similarly to the Basic Model Interface proposed by Tucker et al. (2022) for the Community Surface Dynamics Modelling System. While the IGM python package is primarily composed of an ensemble of modules, the main Python script, `igm_run.py` plays a central role: `igm_run.py` loads the parameters of all modules, initialize all the modules, implement the time loop, during which all modules are updated, and finalize all the modules. This workflow is depicted in Figure 2.

```
import igm

modules = [...] # definition of modules

for module in modules:
    getattr(igm, "params_"+module)(parser)
params = parser.parse_args() # def params

state = igm.State() # def of state

for m in modules:
   getattr(igm,"initialize_"+m)(params,state)

while state.t < params.time_end:
   for module in modules:
      getattr(igm,"update_"+m)(params,state)

for module in modules:
   getattr(igm,"finalize_"+m)(params,state)
```

Figure 2: Sketch of the IGM main script `igm_run.py` that permits to run the glacier evolution model.

Var. names	Shape	Description	Unit
`t`		Time variable	y
`x,y`	nx	Coordinates	m
`thk`	ny	Ice thickness	m
`topg`	ny,nx	Basal topography	m
`usurf`	ny,nx	Surf. topography	m
`smb`	ny,nx	Surf. mass balance	m/y
`U`	2,nz,ny,nx	Ice velocity field	m/y
`arrhenius`	ny,nx	Arrhnius Factor	TODO
`slidingco`	ny,nx	Sliding Coeficient	TODO

Table 1: List, shape, description, and unit of of key variable names within IGM.

In the workflow depicted in Figure 2, there are three fundamental objects: i) `modules` is a list of modules representing the components of the glacier evolution model picked by the user, ii) `params` contains all the parameters needed for the simulation, organized as attributes (For example, `params.time_start` represents the initial simulation time) iii) `state` holds all the variables at a specific time (e.g., `state.U` and `state.thk` denotes the 3D ice flow and the 2D ice thickness fields). These variables are updated throughout the simulation to represent the evolving state of the glacier. In general, variable names adopts name convention of PISM, a minimal list of key variables is given in Table 1.

IGM is a Python package, which can be installed using pip directly from the source (`https://github.com/jouvetg/igm`) or from from the PyPi project (`https://pypi.org/`) using command:

```
pip install igm-model
```

which installs IGM for CPU usage. Running IGM on a GPU can significantly speed up computations, but it typically requires additional efforts, including the installation of GPU drivers, CUDA/CuDNN libraries, and TensorFlow with compatible versions.

The main IGM script `igm_run.py` accepts a wide range of parameters, and these parameters can be passed through the command line when running the script as follows:

```
igm_run --oggm_RGI_ID RGI60-11.01238
```

or in a companion JSON file as shown in Fig. 3.

```
{
  "modules_preproc": ["oggm_shop"],
  "modules_process": ["iceflow",...],
  "modules_postproc": ["plot2d"],
  "oggm_RGI_ID": "RGI60-11.01238",
  "time_start": 2023.0,
  "time_end": 2100.0
}
```

Figure 3: Example of JSON IGM parameter file.

The parameter values passed in the command line override those provided in the JSON parameter file, while the JSON parameter file, in turn, overrides the default IGM parameters.

In the following sections, we describe the most important IGM modules, each identified by a keyword.

4.1 Pre-processing modules

Pre-processing modules (Fig. 4) are essentially collecting data (`oggm_shop`), reading data from files (`load_ncdf` and `load_tif`). Optionally, a data assimilation step (`optimize` module) can be incorporated after reading the data.

4.1.1 `oggm_shop`

This module utilizes the Open Glacier Global Model (OGGM Maussion et al., 2019) (and then depends on python package `oggm`) to acquire data for runnin the inverse and forward glacier evolution model for present-day glaciers. Users provide the RGI ID (parameter `oggm_RGI_ID`) of the glacier they are interested in, typically obtained from sources such as `https://www.glims.org/maps/glims`). The data provided by OGGM is preprocessed with a spatial resolution of 100 m and a oggm_border size of 30 m, and available on servers. However, users can customize the spatial resolution and oggm_border size by setting parameter `oggm_preprocess` to `False`, and setting `oggm_dx` and `oggm_border` parameters to desired values. Upon running this module, its automatically downloads an ensemble of data related to the specified glacier, which is then stored in a folder named after the RGI ID. Users can select 2D gridded variables of interest from the available products (see Table 2). These selected variables are transformed into TensorFlow objects, making them accessible within the code as an attribute to the `state` variable, e.g. `state.thk` designes the ice thickness array, and `state.thk` designes the basal topgraphy. In addition, the module may also download available ice thickness profile data (GlaThiDa Consortium, 2020), and rasterize the data on the working grid into variable `state.thkobs` (referrring to observed thickness), which is filled with NaN values where no measurements are available. A copy of the selected variables is stored in a NetCDF file, which can be run using the `load_ncdf` module, eliminating the need to re-download the data.

Variable	Reference
Surface DEM	Copernicus DEM GLO-90
Ice thickness	(Millan et al., 2022)
Ice thickness	(Farinotti et al., 2019)
Surface ice speeds	(Millan et al., 2022)
Surface ice speeds	`its-live.jpl.nasa.gov`
Glacier mask	Randolph Glacier Inventory
Ice thickness profile	(GlaThiDa Consortium, 2020)
Glacier change	(Hugonnet et al., 2021)
Climate data	GSWP3_W5E5
Flowline	OGGM

Table 2: Products available with the `oggm_shop` module.

4.1.2 `load_ncdf` and `load_tif`

The `load_ncdf` module is responsible for loading spatial 2D raster data from a NetCDF file specified by the `lncd_input_file` parameter. It transforms all existing 2D fields within the NetCDF file into TensorFlow variables, which become attributes of the `state` object. The module is expected to import at least the basal topography, which is represented by the variable `state.topg`. Optionally, it can provide other fields such as the initial ice thickness `state.thk` and more. Any field present in the NetCDF file will be passed as TensorFlow variables, making them accessible as `state.field`. For example, a variable like `state.icemask` can be provided to delimit the accumulation area in the surface mass balance computation and prevent overflowing into neighboring catchments. During this stage, raster data can be resampled or coarsened by a certain factor using the `lncd_coarsen` param-

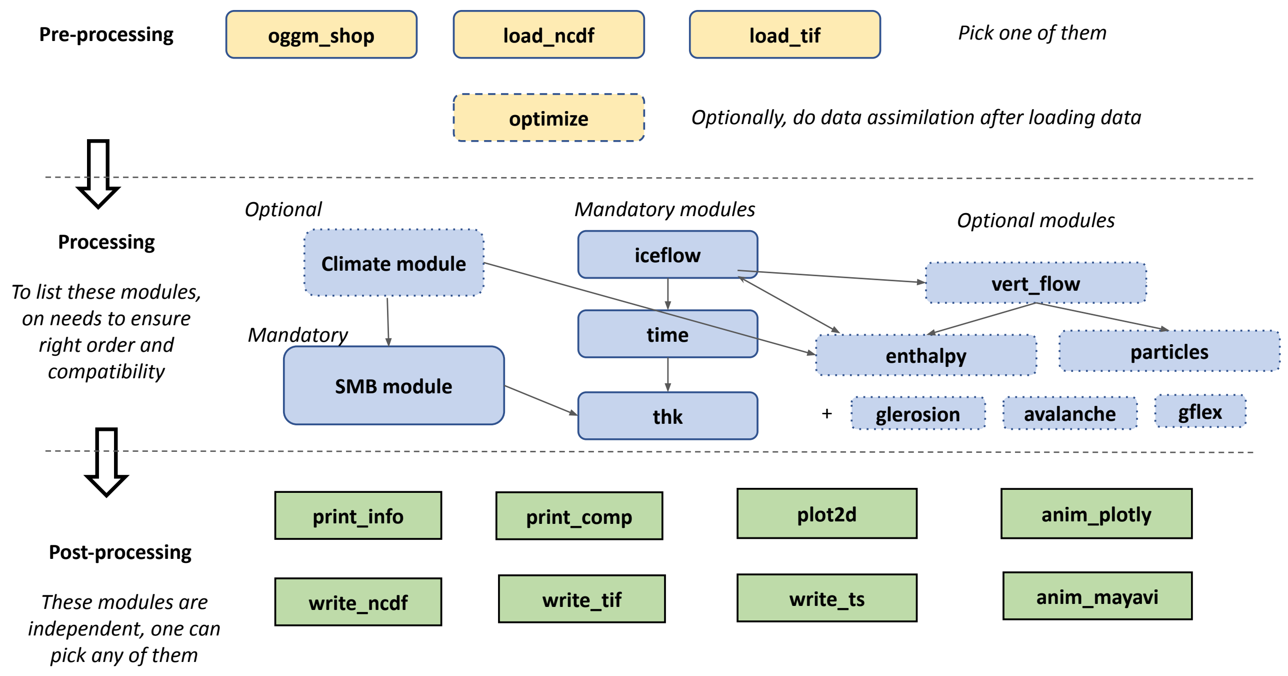

Figure 4: Flowchart of IGM modules.

eter. Additionally, it can be cropped to a specific area using the `lncd_crop` parameter, enabling quicker low-resolution runs or simulations in specific regions. The `load_tif` module essentially performs the same task as `load_ncdf`, but it reads all tif files (e.g., `topg.tif`) found in the working directory and transforms them into TensorFlow variables, named after the file name.

4.1.3 `optimize`

The module `optimize` implements the inverse modelling described in Section 2.2. Solving the optimization problem greatly leverages the automatic differentiation tools provided by the Tensorflow library, and the description of the ice flow model as a neural network (Section 4.2.3). We refer to Jouvet (2023) for a detailed explanation of the methodology. Here, we will describe how to use the `optimize` module for data assimilation as a preliminary step to a forward or prognostic model run. Note that his module was designed for isothermal ice without additional `enthalpy` module.

Before running the module, it is crucial to gather as much observational data as possible. This can be done using `oggm_shop` module (Tab. 2) or reading these data from data file with `load_ncdf` and `load_tif`.

Observational data follows a convention where variable names end with `obs`. Once optimized, the resulting variables are named according to the convention, but without the additional `obs` suffix. Typically, we may use observed horizontal ice surface ice velocities $\mathbf{u}^{s,obs}$, (named as `uvelsurfobs` and `vvelsurfobs`), top surface elevation s^{obs}, (`usurf`), ice thickness profiles h_p^{obs} (`thkobs`), glacier mask from RGI outlines (`icemask`) to enforce zero ice thickness outside the mask, and (`thkinit`) a formerly-inferred ice thickness field to initalize the inverse model (Tab. 2). All the observational data (including individual ice thickness profiles) are rasterized on the same grid. Pixels that do not contain data are ignored in the optimization, typically represented by NaN (not-a-number) values.

While the optimization problem is presented in its most general form (Section 2.2), the user can select a sub-ensemble of controls and constraints within the cost function based on the availability of data. Parameter `opti_control` defines the list of variables to optimize (e.g. `['thk','slidingco','usurf']`), while parameter `opti_cost` defines the list of cost components to minimize (e.g. `['velsurf','thk','usurf','icemask']`).

Finding a balance between controls and constraints is crucial to maintaining the well-posedness of the problem and guarding against multiple solutions. It is advisable to begin with a simple optimization approach, starting with just a single control (typically `thk`), and a few target/cost component (typically `velsurf`, and `icemask`). Once the simple optimization gives meaningfull results, the complexity of the optimization can be increased by adding controls and cost components. There are a couple parameters that may need tuning for each application:

- Confidence levels (i.e. tolerance to fit the data) $\sigma^u, \sigma^h, \sigma^s, \sigma^d$ may be changed to fit surface ice velocity, ice thickness, surface top elevation, or divergence of the flux by changing parameters `opti_velsurfobs_std`, `opti_thkobs_std`, `opti_usurfobs_std`, and `opti_divfluxobs_std`.

- Regularization parameters α^h, α^c control the regularization weights for the ice thickness h and sliding coefficient c (increasing α^h, α^c will make these fields spatially smoother). They can be changed throught parameters `opti_regu_param_thk` and `opti_regu_param_slidingco`.

- Parameters β and γ involved for regularizing the ice thickness h can be changed with parameters `opti_smooth_anisotropy_factor` and `opti_convexity_weight`. Setting β to 1 enforces isotropic smoothing, while reducing β increases anisotropy, promoting more smoothing along ice flow directions rather than across them. This behavior is motivated by expectations for the topography of a glacier bedrock, which has undergone long-term erosion. Adjusting the parameter γ to a lower value can be beneficial to improve the convexity into the system. This can be particularly useful when initializing the inverse model with zero thickness or when dealing with margin regions that lack of data.

The data assimilation scoring can be monitored during the inverse modelling in several ways: checking the decrease of misfit cost component, or monitoring the Root-Mean Square Errors between modelled and observation fields as well as the spatial distribution of these fields.

4.2 Processing modules

Processing modules implement update rules for all modeled variables within the time loop. Most modules correspond to specific physical components. For instance, the `iceflow` module updates 3D ice velocity, the `thk` module handles updates to ice thickness, and so on. Processing modules must be listed in order: for instance, `iceflow` must come before `thk` as the latter needs the first.

4.2.1 Climate modules

Climate modules implement climate forcing in IGM to be used to force the surface mass balance or enthalpy models. They output distributed field of air temperature (variable `air_temp` in °C), precipitation (variable `precipitation` in $kg\, m^{-2}\, y^{-1}$), air temperature variablity (variable `air_temp_std` in °C) at a given time resolution (e.g. daily, weekly or montly). Among them, module `clim_oggm` reads monthly time series of historical GSWP3_W5E5 climate data collected by the `oggm_shop` module, and generates monthly 2D raster fields of corrected precipitation, mean temperature, and temperature variability. To achieve this, we first apply a multiplicative correction factor for precipitation (parameter `prcp_fac`) and a biais correction for temperature (parameter `temp_bias`). Then, the module extrapolates temperature data to the entire glacier surface using a reference height and a constant lapse rate (parameter `temp_default_gradient`). In constrast, the point-wise data for precipitation and temperature variablity are extended to the entire domain without further correction. Module `oggm_shop` provides all calibrated parameters. Module `clim_oggm` can addionally generate climate outside the time frame of available data as necessary for future scenario-based simulations. To that aim, we define a reference period with parameter `clim_oggm_ref_period` to pick randomly years within this time interval (usually taken to be a climate-neutral period), and apply a biais in temperature and a scaling of precipitation defined by parameter `clim_oggm_clim_trend_array`.

4.2.2 Surface mass balance modules

These modules provide a rule to update the climatic surface climate mass balance (variable `smb`). Here we review two of these models, one computing the surface mass balance directly from given parameters, and one from climatic variables (Section 4.2.1).

Module `smb_simple` implements the "simple" surface mass balance model defined by (29)-(30). To that aim, the four parameters z_{ELA}, β_{abl}, β_{acc}, m_{acc} are given in the array parameter `smb_simple_array` for some times (alternatively, these parameters can be provided in an external file). Then, at time t, the module interpolates linearly the four parameters from those given in `smb_simple_array`.

Module `smb_oggm` implements a monthly temperature index model calibrated on geodetic mass balance data (Hugonnet et al., 2021) by OGGM (Maussion et al., 2019). The yearly surface mass balance at any point is computed with (31) from monthly temperature and precipitation fields as described in Section 4.2.1.

4.2.3 `iceflow`

This module models ice flow dynamics in 3D using a Convolutional Neural Network (CNN, Fig. 5) based on Physics Informed Neural Network as described by Jouvet and Cordonnier (2023). Our CNN predicts horizontal ice flow $(\mathbf{u}_H, \mathbf{v}_H)$ from input fields which includes ice thickness h_H, surface topography s_H, ice flow parameters A_H and sliding coefficient c_H, and spatial grid resolution H_H (Fig. 5). The CNN is trained to minimize the energy associated with the Blatter-Pattyn ice flow model (as described in Section 2.1.2), during the time iterations of a glacier evolution model. This energy reads

$$\mathcal{J}(\mathbf{v}) = \int_V 2\frac{A^{-\frac{1}{n}}}{1+\frac{1}{n}}|D(\mathbf{v})|^{1+\frac{1}{n}} dV + \int_\Gamma \frac{c}{1+m}|\mathbf{v}|_M^{1+m} dS + \rho g \int_V (\nabla s \cdot \mathbf{v}) dV, \tag{40}$$

where V and Γ denote the ice volume and bedrock interface, respectively (Jouvet and Cordonnier, 2023). This approach offers a computationally-efficient alternative to traditional solvers and exhibits the capability to handle a variety of ice flow regimes, along with the ability to memorize previous solutions. In the IGM strategy, we initially load a pretrained iceflow emulator provided with the IGM package. Subsequently, we retrain the iceflow emulator regularly within the time loop to maintain its accuracy, ensuring fidelity to the solver, and adapting it to updated glacier geometries. As retraining costs about 3 times more than a CNN evaluation, we strive to minimize the frequency of retraining (Jouvet and Cordonnier, 2023).

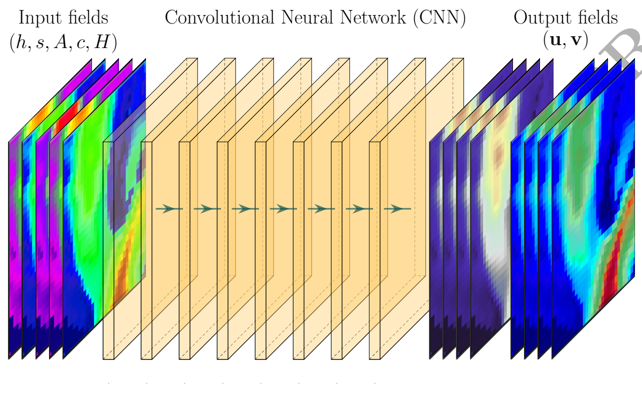
Input fields
(h, s, A, c, H)
Convolutional Neural Network (CNN)
Output fields
$(\mathbf{u}, \mathbf{v})$

Figure 5: Our iceflow emulator consists of a CNN that maps geometrical (thickness and surface topography), ice flow parameters (shearing and basal sliding), and spatial resolution inputs to 3D ice flow fields.

Parameters associated with this module are of several kinds:

- Physical parameters includes Glen's exponent n (`iflo_exp_glen`), Weertman's exponent m (`iflo_exp_weertman`), initial sliding coefficient c (`iflo_init_slidingco`) and initial arrhenius factor A (`iflo_init_arrhenius`) defined in Section 2.1.2. Note that A and c are automatically updated when using module `enthalpy`.
- Parameters associated with vertical discretization include the number of points N_z along the vertical direction (`iflo_Nz`) and the parameter `iflo_vert_spacing`, which controls the density of points near the bedrock. By default, `iflo_vert_spacing` is set to 4. This means that the layers are distributed according to a quadratic rule, resulting in four times finer discretization near the ice-bedrock interface, where the steepest gradients are expected, compared to the discretization near the ice surface.
- Parameter `iflo_retrain_emulator_lr` controls the learning rate for the retraining of the CNN: A low number results in a gentle learning with the downside that several iterations may be needed to achieve optimal learning. On the other hand, `iflo_retrain_emulator_freq` controls the retraining frequency, which needs to be sufficiently frequent to maintain accuracy, but not too much to mitigate computational costs. When treating large arrays, retraining must be done sequentially patch-wise for memory reason. The maximum size of patches is controlled by parameter `iflo_multiple_window_size`.
- One can choose between using the iceflow emulator (the default choice, achieved by setting `iflo_type` to `emulated`), the solver (set `iflo_type` to `solved`), or both (set `iflo_type` to `diagnostic`) to assess the fidelity of the iceflow emulator to the solver.
- One may choose between 2D Arrhenius factor (allowing horizontal variations only) by changing parameters between `iflo_dim_arrhenius` to 2 or 3. Turning it to 3 is necessary for the enthalpy model.

4.2.4 `time`

This module computes the time step with the following criteria: i) It is adjusted precisely to allow matching with specified saving times, whose the frquency is defined by parameter `time_save`. ii) It remains below a designated maximum time step, as determined by the parameter `time_step_max`. iii) It complies to the Courant-Friedrichs-Lewy (CFL) condition, as governed by the parameter `time_cfl`. Indeed, to ensure stability in the transport scheme for ice thickness evolution, it is essential that the time step respects the CFL condition. The module additionally updates the time `time` in addition to the time step `time`. Other key parameters of this module are `time_start` and `time_end`, which define the simulation starting and ending times.

4.2.5 `thk`

This module solves the mass conservation equation (1) to update ice thickness based on the results obtained from the iceflow module and the surface mass balance computed by another module. Equation (1) is solved using an explicit first-order upwind finite-volume scheme on the 2D working grid. In this scheme, ice mass is permitted to move between adjacent cells where thickness and velocities are defined. This movement is determined by edge-defined fluxes, which are inferred from depth-averaged velocities and ice thickness in the upwind direction. The resulting scheme is both mass-conserving and parallelizable, thanks to its fully explicit nature. However, it is subject to a Courant-Friedrichs-Lewy (CFL) condition, as explained in the previous section. This condition guarantees that no more than the entire content of one cell is transferred to a neighboring cell within a single iteration.

4.2.6 vert_flow

Since the iceflow module is based on the Blatter-Pattyn ice flow model and predicts only the horizontal components of ice velocity, an additional module, vert_flow is responsible for computing the vertical component, denoted as state.W, using the horizontal components state.U computed by the iceflow module. This computation is achieved by integrating the incompressibility condition (3). The vert_flow module is mandatory in various other modules, such as particle which involves 3D particle trajectory integration, and the enthalpy module, which is responsible for computing the 3D advection-diffusion of enthalpy.

4.2.7 enthalpy

The enthalpy module is responsible for updating the 3D enthalpy field based on the mechanical state by solving the advection-diffusion equation (22)-(23). This process includes applying the top and bottom boundary conditions described in Section 2.1.3. To achieve this, the variable E (and equivalently, T and ω) is defined on the same 3D structured grid than the ice flow field $\mathbf{u}$. Solving equations (22)-(23) is done in two steps through time splitting operators. First, an explicit first-order upwind finite-volume scheme is employed to solve the advective component in the horizontal direction, following a similar scheme as the one used in the mass conservation equation:

$$E^{n+\frac{1}{2}} = E^n - dt\left(u_x^n \frac{\partial E^n}{\partial x} + u_y^n \frac{\partial E^n}{\partial y}\right), \tag{41}$$

In a second time we solve the remaining advection-diffusion equation with respect to the vertical direction in an implicit manner:

$$\rho_i\left(\frac{E^{n+1} - E^{n+\frac{1}{2}}}{dt} + u_z \frac{\partial E^{n+1}}{\partial z}\right) \tag{42}$$

$$-\frac{\partial}{\partial z}\left(K_{c,t}\frac{\partial E^{n+1}}{\partial z}\right) = \phi^n - \rho_w L D_w(\omega^{n+\frac{1}{2}}). \tag{43}$$

This is done by finite differences, where the advection term is approximated using an upwind method, while the diffusion term is approximated using a centered scheme. Once the top and bottom boundary conditions are incorporated, the discretization leads to solving $N_y \times N_x$ tridiagonal systems, each with a size equal to the number of layers N_z. Solving these tridiagonal systems is accomplished using the tridiagonal matrix algorithm (or Thomas algorithm). This algorithm requires approximately $3 \times N_z$ operations per column, translating to a total of $3 \times N_z \times N_y \times N_x$ operations in total. Tensorflow ensures parallelism of operations between column-wise problems. Following solving the enthalpy equation, the process involves computing the basal melt rate using (26), updating it along with the enthalpy to account for water drainage along the ice column. This step also calculates the resulting water thickness from (27) and sliding coefficient from (17).

In summary, assuming known enthalpy E^n, ice thickness geometry h^{n+1}, iceflow $\mathbf{u}^{n+1}$, vertical ice flow u_z^{n+1}, updating the enthalpy at time t^{n+1} requires to perform the following sub-steps:

- Compute the mean surface temperature T_s^n to enforce the upper surface Dirichlet boundary condition. This temperature is capped at 0°C to maintain the temperature of ice below pressure-melting point.
- Compute the vertical discretization with the new ice geometry h^{n+1}.
- Compute the temperature T_{pmp}^n and enthalpy E_{pmp}^n at pressure melting point using (21).
- Compute the ice temperature T^n from the enthalpy E^n using (18).
- Compute the Arrhenius factor $A(T^n)$ from temperature T^n using (10)-(11).
- Compute the 3D strain heat ϕ^{n+1} from ice flow field $\mathbf{u}^{n+1}$ and Arrhenius factor $A(T^n)$ using (24).
- Compute the 2D basal frictional heat Q_{fh}^{n+1}, from basal velocity field $\mathbf{u}^{n+1}$ and sliding coefficient c^n using (25).
- Compute the surface enthalpy E_s^n from the surface temperature T_s^n using (18).
- Compute the updated enthalpy $E^{n+\frac{1}{2}}$ after solving one explicit step for the horizonal advection (Eq. (41)).
- Compute the updated enthalpy E^{n+1} field solving the advection-diffusion equation (Eq. (43)).
- Compute the basal melt rate from (26).

- Compute the water thickness in the till W^{n+1} solving (27) explicity.
- Compute the sliding parametrization c^{n+1} using (17).

4.2.8 `particles`

This module implements a particle tracking routine designed to calculate the time-space trajectory of virtual tracers that are advected by the ice flow, representing materials like debris or moraines. Thanks to TensorFlow's GPU implementation, a substantial number of particles can be efficiently computed in parallel. The routine encompasses particle seeding (the default seeds in the accumulation area at regular intervals), and particle tracking, achieved through advection using the 3D velocity field. For that purpose, one uses a simple explicit forward-in-time Euler scheme. The positions of particles are stored within vectors of length equal to the number of tracked particles: `state.xpos`, `state.ypos`, and `state.zpos`. There is currently no strategy in place for removing particles. Presently, there are two implementations available, controlled by the parameter `part_tracking_method`:

- `simple`: In the horizontal plane, particles are advected using the horizontal velocity field, with interpolation performed bilinearly. In the vertical direction, particles are tracked along the ice column, scaling their position between 0 and 1, where 0 represents the bed and 1 corresponds to the top surface. The evolution of the particles within the ice column over time is determined by the surface mass balance: when the surface mass balance is positive (indicating ice accumulation), the particles move deeper within the ice column, reducing their relative height. Conversely, when the surface mass balance is negative (indicating ice ablation), the particles rise within the ice column, increasing their relative height.
- `3d`: This method advects particles from the the 3D ice flow velocity field. Unlike the "simple" method, one needs to access the vertical ice velocity component by activating `vert_flow` module.

The seeding strategy is flexible and can be easily modified. To that aim, one simply needs to redefine the function `igm.seeding()` in a file `particles.py`. Note that particles can be plotted with `plot2d` module, or being written in a separate file.

4.3 Post-processing modules

IGM includes post-processing modules that allow for result monitoring. The frequency of saving is determined by the parameter `params.time_save`, which is defined within the `time` module.

4.3.1 `plot2d`

Based on `matplotlib`, the `plot2d` module generates 2D plan-view plots (possibly with overlayed particles), which can be displayed in real-time or saved as image files. These plots visualize variables defined by the parameter `params.plt2d_var`, which can be set to any 2D variables as defined in Tab. 1.

4.3.2 `write_ncdf` and `write_tif`

The `write_ncdf` and `write_tif` modules are responsible for writing 2D field variables that are specified in the parameter lists: `params.wncd_vars_to_save` and `params.wtif_vars_to_save`, respectively. These variables are written into the NetCDF output file specified by the parameter `params.wncd_output_file` and into TIF output files. For tif files, files are created in the working directory with names composed by the variable name and the time (e.g., thk-000040.tif, usurf-000090.tif).

4.3.3 `anim_plotly`

This module permits an interactive 3D vizualization of IGM results reading the NetCDF file produced by module `write_ncdf` based on libraries `dash` and `plotly`. It creates a dash app that can be accessed via a browser (the adress printed in the console is usually `http://127.0.0.1:8050/`). The app shows a 3D plot of the glacier's surface on top of the surrounding bedrock. The surface color shows either the ice thickness, the velocity magnitude of the surface or the surface mass balance. Variables can be chosen in the dropdown menu. The app also includes a slider to navigate the different time steps of the glacier simulation.

4.3.4 `anim_mayavi`

This module based on the `mayavi` and `pyqt5` libraries produces a 3D animated plot using from the NetCDF output file produced by module `write_ncdf`.

5 Applications

This section will contain a serie of numerical experiments and applications that will illustrate the capabilities of IGM.

References

Aschwanden, A., Bueler, E., Khroulev, C., and Blatter, H. (2012). An enthalpy formulation for glaciers and ice sheets. *Journal of Glaciology*, 58(209):441–457.

Blatter, H. (1995). Velocity and stress fields in grounded glaciers: a simple algorithm for including deviatoric stress gradients. *Journal of Glaciology*, 41(138):333–344.

Brædstrup, C. F., Damsgaard, A., and Egholm, D. L. (2014). Ice-sheet modelling accelerated by graphics cards. *Computers & Geosciences*, 72:210–220.

Bueler, E. and van Pelt, W. (2015). Mass-conserving subglacial hydrology in the parallel ice sheet model version 0.6. *Geoscientific Model Development*, 8(6):1613–1635.

Cuffey, K. and Paterson, W. (2010). *The physics of glaciers*. Academic Press. glacier depth average velocity.

Farinotti, D., Huss, M., Fürst, J. J., Landmann, J., Machguth, H., Maussion, F., and Pandit, A. (2019). A consensus estimate for the ice thickness distribution of all glaciers on earth. *Nature Geoscience*, 12(3):168–173.

Gagliardini, O., Zwinger, T., Gillet-Chaulet, F., Durand, G., Favier, L., de Fleurian, B., Greve, R., Malinen, M., Martín, C., Råback, P., Ruokolainen, J., Sacchettini, M., Schäfer, M., Seddik, H., and Thies, J. (2013). Capabilities and performance of Elmer/Ice, a new-generation ice sheet model. *Geoscientific Model Development*, 6(4):1299–1318.

GlaThiDa Consortium (2020). Glacier Thickness Database 3.1.0. *World Glacier Monitoring Service, Zurich, Switzerland.*

Glen, J. W. (1953). Rate of Flow of Polycrystalline Ice. *Nature*, 172:721–722.

Hock, R. (2003). Temperature index melt modelling in mountain areas. *Journal of Hydrology*, 282:104–115.

Hugonnet, R., McNabb, R., Berthier, E., Menounos, B., Nuth, C., Girod, L., Farinotti, D., Huss, M., Dussaillant, I., Brun, F., et al. (2021). Accelerated global glacier mass loss in the early twenty-first century. *Nature*, 592(7856):726–731.

Jouvet, G. (2023). Inversion of a stokes glacier flow model emulated by deep learning. *Journal of Glaciology*, 69(273):13–26.

Jouvet, G. and Cordonnier, G. (2023). Ice-flow model emulator based on physics-informed deep learning. *Journal of Glaciology*, pages 1–15.

Jouvet, G., Cordonnier, G., Kim, B., Lüthi, M., Vieli, A., and Aschwanden, A. (2022). Deep learning speeds up ice flow modelling by several orders of magnitude. *Journal of Glaciology*, 68(270):651–664.

Khroulev, C. and the PISM Authors (2020). PISM, a Parallel Ice Sheet Model v1.2: User's Manual.

Larour, E., Seroussi, H., Morlighem, M., and Rignot, E. (2012). Continental scale, high order, high spatial resolution, ice sheet modeling using the ice sheet system model (issm). *Journal of Geophysical Research: Earth Surface*, 117(F1).

Lipscomb, W. H., Price, S. F., Hoffman, M. J., Leguy, G. R., Bennett, A. R., Bradley, S. L., Evans, K. J., Fyke, J. G., Kennedy, J. H., Perego, M., et al. (2019). Description and evaluation of the community ice sheet model (cism) v2. 1. *Geoscientific Model Development*, 12(1):387–424.

Maussion, F., Butenko, A., Champollion, N., Dusch, M., Eis, J., Fourteau, K., Gregor, P., Jarosch, A. H., Landmann, J. M., Oesterle, F., et al. (2019). The open global glacier model (oggm) v1. 1. *Geoscientific Model Development*, 12(3):909–931.

Millan, R., Mouginot, J., Rabatel, A., and Morlighem, M. (2022). Ice velocity and thickness of the world's glaciers. *Nature Geoscience*, 15(2):124–129.

Paterson, W. S. B. (1994). *The Physics of Glaciers*. Pergamon, New York, third edition.

Pattyn, F. (2018). The paradigm shift in antarctic ice sheet modelling. *Nature communications*, 9(1):1–3.

Räss, L., Licul, A., Herman, F., Podladchikov, Y. Y., and Suckale, J. (2020). Modelling thermomechanical ice deformation using an implicit pseudo-transient method (FastICE v1. 0) based on graphical processing units (GPUs). *Geoscientific Model Development*, 13(3):955–976.

Rounce, D. R., Hock, R., and Shean, D. E. (2020). Glacier mass change in high mountain asia through 2100 using the open-source python glacier evolution model (pygem). *Frontiers in Earth Science*, 7:331.

Schoof, C. and Hewitt, I. (2013). Ice-sheet dynamics. *Annual Review of Fluid Mechanics*, 45(1):217–239.

Tucker, G. E., Hutton, E. W. H., Piper, M. D., Campforts, B., Gan, T., Barnhart, K. R., Kettner, A. J., Overeem, I., Peckham, S. D., McCready, L., and Syvitski, J. (2022). Csdms: a community platform for numerical modeling of earth surface processes. *Geoscientific Model Development*, 15(4):1413–1439.

Višnjević, V., Herman, F., and Prasicek, G. (2020). Climatic patterns over the european alps during the lgm derived from inversion of the paleo-ice extent. *Earth and Planetary Science Letters*, 538:116185.

Wang, Y., Zhang, T., Xiao, C., Ren, J., and Wang, Y. (2020). A two-dimensional, higher-order, enthalpy-based thermomechanical ice flow model for mountain glaciers and its benchmark experiments. *Computers & geosciences*, 141:104526.

Winkelmann, R., Martin, M. A., Haseloff, M., Albrecht, T., Bueler, E., Khroulev, C., and Levermann, A. (2011). The Potsdam parallel ice sheet model (PISM-PIK) – part 1: Model description. *The Cryosphere*, 5(3):715–726.

Zekollari, H., Huss, M., Farinotti, D., and Lhermitte, S. (2022). Ice-dynamical glacier evolution modeling—a review. *Reviews of Geophysics*, 60(2):e2021RG000754.