Inversion in the Ice-sheet and Sea-level System Model: Formulation, Discretisation, and Implementation

Technical note

August 14, 2026

Abstract

This note documents how the Ice-sheet and Sea-level System Model (ISSM) formulates and implements inverse ("control") methods for inferring poorly-constrained model inputs — principally basal friction and depth-averaged ice hardness — from remotely sensed surface velocity. We give the continuous PDE-constrained optimisation problem, derive the adjoint of the shelfy-stream approximation (SSA) explicitly, show precisely what the *incomplete adjoint* option discards and why the resulting operator is still symmetric positive definite, catalogue the cost-function library and its integer interface, describe the three optimisation drivers (Brent, TAO, M1QN3) and their box-constraint handling, and contrast the hand-derived adjoint path with the algorithmic-differentiation (AD) path built on ADOL-C/CoDiPack. A final section lists claims that should be checked against the specific ISSM revision in use, since several of the details below are version-dependent.

Contents

1 Scope and framing

ISSM [3] solves ice-flow problems in which several critical inputs are not observable: the basal friction coefficient beneath grounded ice, and the depth-averaged rigidity (hardness) $\bar{B}$ of floating ice, which absorbs the effects of temperature, fabric, and damage. Both control the stress balance at first order, and neither can be measured at ice-sheet scale. Surface velocity, by contrast, is available essentially everywhere at high resolution.

The strategy — introduced to glaciology by MacAyeal [6] — is to treat the unknown field as a distributed control and choose it to minimise the misfit between modelled and observed surface velocity, subject to the stress balance holding as a constraint. This is a PDE-constrained optimisation problem, solved in ISSM by gradient-based descent with gradients supplied by the adjoint of the stress-balance operator.

Two implementation routes coexist in the code base:

1. The **control method** proper: a hand-derived, hand-coded adjoint, exposed through the `md.inversion` model class and triggered by running the stress-balance solution with `md.inversion.iscontrol = 1`. This is the mature, production path and is what most ISSM inversions use.

2. **Algorithmic differentiation**: source transformation of the forward model by an operator-overloading AD tool (historically ADOL-C, now CoDiPack), exposed through the `md.autodiff` class. This produces exact discrete adjoints of arbitrary solution sequences, including transient runs, at higher memory and build cost.

Sections 2–7 concern the first route; Section 9 the second.

2 The forward problem

2.1 Stress balance

ISSM implements a hierarchy of stress-balance approximations: SSA (shelfy-stream), the Blatter–Pattyn higher-order (HO) model, L1L2, MOLHO, and full Stokes (FS). The adjoint machinery is analogous throughout; we present SSA because it is the case for which every step

can be written down compactly, and because it is the setting of the majority of published ISSM inversions.

Let $\Omega \subset \mathbb{R}^2$ be the horizontal footprint of the ice, H the thickness, s the surface elevation, and $\mathbf{u} = (u, v)^{\mathsf{T}}$ the depth-independent horizontal velocity. Write the strain-rate vector and the SSA constitutive matrix as

$$\mathbf{D}(\mathbf{u}) = \begin{pmatrix} \dot\varepsilon_{xx} \\ \dot\varepsilon_{yy} \\ 2\dot\varepsilon_{xy} \end{pmatrix}, \qquad \mathbf{C} = \begin{pmatrix} 4 & 2 & 0 \\ 2 & 4 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{1}$$

with $\dot\varepsilon_{xx} = \partial_x u$, $\dot\varepsilon_{yy} = \partial_y v$, $\dot\varepsilon_{xy} = \tfrac{1}{2}(\partial_y u + \partial_x v)$. The SSA effective strain rate satisfies the convenient identity

$$\dot\varepsilon_e^2 \;=\; \dot\varepsilon_{xx}^2 + \dot\varepsilon_{yy}^2 + \dot\varepsilon_{xx}\dot\varepsilon_{yy} + \dot\varepsilon_{xy}^2 \;=\; \tfrac{1}{4}\,\mathbf{D}(\mathbf{u})^{\mathsf{T}}\mathbf{C}\,\mathbf{D}(\mathbf{u}). \tag{2}$$

Glen's flow law with exponent n gives the effective viscosity

$$\mu(\mathbf{u}; \bar B) = \frac{\bar B}{2}\,\dot\varepsilon_e^{(1-n)/n}, \qquad \bar B = \bar A^{-1/n}, \tag{3}$$

where $\bar B$ is `md.materials.rheology_Bbar`.

Basal traction under grounded ice follows a friction law; ISSM's default (`friction` class, a Budd-type law) is

$$\boldsymbol\tau_b = -\,\alpha^2\, N^{\,r}\, \|\mathbf{u}\|^{\,s-1}\,\mathbf{u}, \qquad r = q/p, \quad s = 1/p, \tag{4}$$

with N the effective pressure and $\alpha =$ `md.friction.coefficient` the control variable. The defaults $p = q = 1$ give the linear-in-velocity Budd law $\boldsymbol\tau_b = -\alpha^2 N\mathbf{u}$. Note that ISSM inverts for α, not α^2; the squaring guarantees a non-negative drag coefficient and removes a sign ambiguity, at the cost of a factor 2α in the gradient (Section 6). Alternative laws (Weertman, Schoof, Coulomb-limited, hydrology-coupled, Tsai) are separate classes with their own control names.

The weak form is: find $\mathbf{u} \in V$ such that for all $\boldsymbol\varphi \in V$,

$$\underbrace{\int_\Omega H\,\mu(\mathbf{u})\,\mathbf{D}(\boldsymbol\varphi)^{\mathsf{T}}\mathbf{C}\,\mathbf{D}(\mathbf{u})\,\mathrm{d}\Omega}_{\text{viscous}} \;+\; \underbrace{\int_{\Omega_g} \alpha^2 N^r \|\mathbf{u}\|^{s-1}\,\mathbf{u}\cdot\boldsymbol\varphi\,\mathrm{d}\Omega}_{\text{basal}} \;+\; \underbrace{\int_\Omega \rho_i g H\,\nabla s\cdot\boldsymbol\varphi\,\mathrm{d}\Omega}_{\text{driving}} \;-\; \ell_\Gamma(\boldsymbol\varphi) \;=\; 0, \tag{5}$$

where Ω_g is the grounded portion and ℓ_Γ collects boundary terms (notably the water-pressure condition at the calving front). We denote the left-hand side by $R(\mathbf{u}, m; \boldsymbol\varphi)$, the weak residual, where m stands for whichever control field is active.

2.2 Discretisation

ISSM uses continuous Galerkin finite elements on unstructured triangles (2D) or prisms and tetrahedra (3D), with P1 nodal bases as the default for SSA; the FS solver uses stabilised or Taylor–Hood-type velocity/pressure pairs. Control fields are represented on the same nodal basis as the primal fields, so a control is a vector $\mathbf{m} \in \mathbb{R}^M$ of nodal values, M being the number of vertices (or the number of surface vertices for depth-averaged controls in 3D, extruded via `project3d`).

The nonlinearity in (5) is resolved by Picard (fixed-point on μ) iteration by default, with a Newton option; `md.stressbalance.restol`, `reltol`, `abstol` and `maxiter` govern convergence. Every cost-function evaluation during the inversion is a complete nonlinear forward solve — this dominates the cost of the whole procedure.

Discretely, write the forward problem as

$$\mathbf{R}(\mathbf{u}, \mathbf{m}) \;=\; \mathbf{K}(\mathbf{u}, \mathbf{m})\,\mathbf{u} - \mathbf{f}(\mathbf{m}) \;=\; \mathbf{0}, \qquad \mathbf{u} \in \mathbb{R}^N. \tag{6}$$

3 The inverse problem

3.1 Objective

ISSM minimises a weighted sum of *cost functions*,

$$J(\mathbf{u},\mathbf{m}) \;=\; \sum_{i=1}^{n_c} \int_{\Omega \text{ or } \Gamma_s} \gamma_i(\mathbf{x})\, j_i(\mathbf{u},\mathbf{m})\, \mathrm{d}\Omega, \tag{7}$$

where each j_i is one of the misfit or regularisation densities of Table 1 and $\gamma_i(\mathbf{x})$ is a *spatially varying* weight supplied as `md.inversion.cost_functions_coefficients`, an $(n_v \times n_c)$ array. The spatial dependence of γ_i is the mechanism by which observational coverage is encoded: setting $\gamma_i = 0$ over vertices with no velocity data removes those points from the misfit without changing the code path. It is also used to down-weight regions of known data artefacts and to strengthen regularisation where the data are sparse.

The reduced objective is $\hat{J}(\mathbf{m}) = J(\mathbf{u}(\mathbf{m}),\mathbf{m})$ with $\mathbf{u}(\mathbf{m})$ defined implicitly by (6). The optimisation problem actually solved is the box-constrained problem

$$\min_{\mathbf{m}} \hat{J}(\mathbf{m}) \quad \text{subject to} \quad \mathbf{m}^{\min} \le \mathbf{m} \le \mathbf{m}^{\max}, \tag{8}$$

with the bounds given per vertex by `md.inversion.min_parameters` and `max_parameters`. These are not cosmetic: without a lower bound the friction coefficient can be driven negative in the squared parameterisation's null direction or to zero in regions of no data, and without an upper bound $\bar{B}$ can run away where the velocity is under-determined.

3.2 Relation to Bayesian inference

Under a Gaussian likelihood with covariance Σ_{obs} and a Gaussian smoothness prior, the maximum a posteriori estimate minimises $\frac{1}{2}\|\mathbf{u}-\mathbf{u}^{\text{obs}}\|^2_{\Sigma_{\text{obs}}^{-1}} + \frac{1}{2}\|\mathbf{m}-\mathbf{m}_0\|^2_{\Sigma_{\text{prior}}^{-1}}$. ISSM's cost function (7) has this form when one uses cost function 101 with $\gamma_1 = \sigma^{-2}$ and a Tikhonov term as the prior. The control method therefore delivers a MAP point estimate; ISSM does *not* compute a posterior covariance or a Hessian-based uncertainty from the control solve. Uncertainty quantification is a separate facility (`md.qmu`, the Dakota interface) based on sampling and surrogate methods.

4 The cost-function library

4.1 Integer interface

Cost functions are selected by integer code in `md.inversion.cost_functions`. The integers are a user-facing API; they are translated to internal enum strings in `src/m/inversions/marshallcostfunctions.{m,py,js}` before being marshalled to the C++ core, and the admissible set is defined in `supportedcostfunctions.{m,py}`. In recent revisions the supported list is

$$[\,101{:}105,\;\; 201,\;\; 501{:}508,\;\; 510,\;\; 601{:}604\,].$$

The blocks are semantically meaningful: $1xx$ are velocity misfits, $2xx$ thickness misfits, $5xx$ regularisation and penalty terms, $6xx$ surface-elevation misfits.

4.2 Why more than one velocity misfit

The absolute misfit (101) is the maximum-likelihood term for homoscedastic Gaussian velocity errors, but on an ice sheet the velocity spans four to five orders of magnitude, so 101 is dominated

Table 1: Cost-function codes and densities. ϵ denotes the small regularising velocity (`epsvel` in the source), S the domain area, $\hat{\mathbf{u}}$ the unit flow direction. Codes 602–604 are supported but their densities should be confirmed against the revision in use.

Code	Internal name	Density j_i
101	`SurfaceAbsVelMisfit`	$\frac{1}{2}\left[(u-u^{\mathrm{obs}})^2+(v-v^{\mathrm{obs}})^2\right]$
102	`SurfaceRelVelMisfit`	$\frac{1}{2}\left[\frac{(u-u^{\mathrm{obs}})^2}{(u^{\mathrm{obs}})^2+\epsilon^2}+\frac{(v-v^{\mathrm{obs}})^2}{(v^{\mathrm{obs}})^2+\epsilon^2}\right]$
103	`SurfaceLogVelMisfit`	$\frac{1}{2}\left[\log\frac{\lVert\mathbf{u}\rVert+\epsilon}{\lVert\mathbf{u}^{\mathrm{obs}}\rVert+\epsilon}\right]^2$
104	`SurfaceLogVxVyMisfit`	$\frac{1}{2}\left[\log\frac{\lvert u\rvert+\epsilon}{\lvert u^{\mathrm{obs}}\rvert+\epsilon}\right]^2+\frac{1}{2}\left[\log\frac{\lvert v\rvert+\epsilon}{\lvert v^{\mathrm{obs}}\rvert+\epsilon}\right]^2$
105	`SurfaceAverageVelMisfit`	$S^{-1}\lVert\mathbf{u}-\mathbf{u}^{\mathrm{obs}}\rVert$
201	`ThicknessAbsMisfit`	$\frac{1}{2}(H-H^{\mathrm{obs}})^2$
501	`DragCoefficientAbsGradient`	$\frac{1}{2}\lVert\nabla\alpha\rVert^2$
502	`RheologyBbarAbsGradient`	$\frac{1}{2}\lVert\nabla\bar{B}\rVert^2$
503	`ThicknessAbsGradient`	$\frac{1}{2}\lVert\nabla H\rVert^2$
504	`ThicknessAlongGradient`	$\frac{1}{2}(\nabla H\cdot\hat{\mathbf{u}})^2$
505	`ThicknessAcrossGradient`	$\frac{1}{2}(\nabla H\cdot\hat{\mathbf{u}}^{\perp})^2$
506	`BalancethicknessMisfit`	mass-conservation residual misfit
507	`RheologyBAbsGradient`	$\frac{1}{2}\lVert\nabla B\rVert^2$ (3D, non-depth-averaged)
508	`RheologyBInitialguessMisfit`	$\frac{1}{2}(B-B_0)^2$
510	`ThicknessPositive`	one-sided penalty on $H<0$
601	`SurfaceAbsMisfit`	$\frac{1}{2}(s-s^{\mathrm{obs}})^2$
602–604	—	further surface/altimetry misfit variants

entirely by the fast outlet glaciers. Slow interior ice contributes essentially nothing and its friction field is left at its initial guess. This is not a subtle effect: it is the single most common failure mode of a first inversion attempt, and it is visible immediately by comparing $\log\lVert\mathbf{u}\rVert$ against $\log\lVert\mathbf{u}^{\mathrm{obs}}\rVert$ rather than the linear fields.

The logarithmic misfit (103) is scale-free and weights a 10% error in the interior the same as a 10% error at the terminus. The standard recipe is therefore a combination `[101, 103, 501]`, with γ_1 typically several orders of magnitude larger than γ_2 so that the two misfit terms are of comparable magnitude *at convergence* — the practical tuning rule is to run, inspect the individual terms reported in the cost-function history, and rescale.

Remark 1. *Note that 103 depends only on $\lVert\mathbf{u}\rVert$ and therefore carries no directional information, whereas 101 constrains u and v separately. A cost function built from 103 alone leaves the flow direction unconstrained. Conversely, 104 restores directional information in log space at the price of poor behaviour where a velocity component changes sign.*

5 The adjoint

5.1 Discrete derivation

Introduce the Lagrangian $\mathcal{L}(\mathbf{u},\mathbf{m},\boldsymbol{\lambda})=J(\mathbf{u},\mathbf{m})+\boldsymbol{\lambda}^{\mathsf{T}}\mathbf{R}(\mathbf{u},\mathbf{m})$. Stationarity with respect to $\mathbf{u}$ gives the adjoint equation

$$\left(\frac{\partial\mathbf{R}}{\partial\mathbf{u}}\right)^{\mathsf{T}}\boldsymbol{\lambda}\;=\;-\frac{\partial J}{\partial\mathbf{u}}, \tag{9}$$

and the reduced gradient follows as

$$\frac{\mathrm{d}\hat{J}}{\mathrm{d}\mathbf{m}} \;=\; \frac{\partial J}{\partial \mathbf{m}} \;+\; \left(\frac{\partial \mathbf{R}}{\partial \mathbf{m}}\right)^{\mathsf{T}} \boldsymbol{\lambda}. \tag{10}$$

The cost is one adjoint solve per gradient, independent of M — this is the entire reason the method is viable for $M \sim 10^5$–10^6 nodal controls.

5.2 The tangent operator, and what `incomplete_adjoint` discards

The only nontrivial ingredient is $\partial\mathbf{R}/\partial\mathbf{u}$. Differentiating the viscous term of (5) in a direction $\delta\mathbf{u}$, and using $\delta(\dot{\varepsilon}_e^2) = \tfrac{1}{2}\mathbf{D}(\mathbf{u})^{\mathsf{T}}\mathbf{C}\,\mathbf{D}(\delta\mathbf{u})$ from (2) together with

$$\frac{\partial \mu}{\partial \dot{\varepsilon}_e^2} \;=\; \frac{1-n}{2n}\,\frac{\mu}{\dot{\varepsilon}_e^2} \qquad \left(= -\frac{\mu}{3\dot{\varepsilon}_e^2} \text{ for } n=3\right), \tag{11}$$

gives the exact tangent bilinear form

$$\boxed{\begin{aligned} a'(\mathbf{u};\delta\mathbf{u},\boldsymbol{\varphi}) \;=\; & \int_\Omega H\mu\,\mathbf{D}(\boldsymbol{\varphi})^{\mathsf{T}}\mathbf{C}\,\mathbf{D}(\delta\mathbf{u})\,\mathrm{d}\Omega \\ & +\; \frac{1-n}{2n}\int_\Omega \frac{H\mu}{2\dot{\varepsilon}_e^2}\big(\mathbf{CD}(\mathbf{u})\cdot\mathbf{D}(\delta\mathbf{u})\big)\big(\mathbf{CD}(\mathbf{u})\cdot\mathbf{D}(\boldsymbol{\varphi})\big)\,\mathrm{d}\Omega \end{aligned}} \tag{12}$$

plus the analogous friction linearisation

$$b'(\mathbf{u};\delta\mathbf{u},\boldsymbol{\varphi}) = \int_{\Omega_g} \alpha^2 N^r \|\mathbf{u}\|^{s-1}\left[\delta\mathbf{u}\cdot\boldsymbol{\varphi} + (s-1)\frac{(\mathbf{u}\cdot\delta\mathbf{u})(\mathbf{u}\cdot\boldsymbol{\varphi})}{\|\mathbf{u}\|^2}\right]\mathrm{d}\Omega. \tag{13}$$

Setting `md.inversion.incomplete_adjoint = 1` discards the second integral in (12) (the documentation phrases this as "linear viscosity"): the viscosity is frozen at its converged value and the adjoint is solved with the Picard stiffness matrix $\mathbf{K}(\mathbf{u}^*)$ rather than the true Jacobian. Setting it to 0 retains the full term.

Three observations make the trade-off clear.

Proposition 1 (Symmetry)**.** *Both integrals in* (12) *are symmetric under $\delta\mathbf{u} \leftrightarrow \boldsymbol{\varphi}$; likewise b'. Hence $\partial\mathbf{R}/\partial\mathbf{u}$ is symmetric and* (9) *requires no transpose. This is not an accident: the SSA residual is the first variation of the convex dissipation potential $\Phi(\mathbf{u}) = \int \frac{2n}{n+1} H\bar{B}\,\dot{\varepsilon}_e^{(n+1)/n} + \frac{1}{s+1}\alpha^2 N^r\|\mathbf{u}\|^{s+1} + \rho_i g H\nabla s\cdot\mathbf{u}$, so the tangent operator is Φ''.*

Proposition 2 (Positive definiteness)**.** *Taking $\delta\mathbf{u} = \boldsymbol{\varphi}$ with $\mathbf{D}(\boldsymbol{\varphi}) = \mathbf{D}(\mathbf{u})$ in* (12) *and using $\mathbf{D}^{\mathsf{T}}\mathbf{CD} = 4\dot{\varepsilon}_e^2$ gives*

$$a'(\mathbf{u};\mathbf{u},\mathbf{u}) = \int_\Omega 4H\mu\,\dot{\varepsilon}_e^2\left[1 + \frac{1-n}{n}\right]\mathrm{d}\Omega = \frac{1}{n}\int_\Omega 4H\mu\,\dot{\varepsilon}_e^2\,\mathrm{d}\Omega > 0. \tag{14}$$

The exact tangent is positive definite but is a factor n softer than the Picard operator along $\mathbf{CD}(\mathbf{u})$ — shear thinning makes the ice easier to deform in the direction it is already deforming. The incomplete adjoint therefore systematically over*-stiffens the sensitivity in that direction.*

Remark 2 (Why the incomplete adjoint is nonetheless used)**.** *With the second term dropped, the adjoint matrix is exactly the matrix already assembled and factorised for the final Picard iterate of the forward solve. The adjoint solve is then a back-substitution rather than a fresh assembly and factorisation. For basal friction inversion the resulting gradient error is largely a directional scaling that a quasi-Newton method absorbs into its metric, and Morlighem et al. [8] found the difference in inferred friction to be small for a higher-order model. The gradient is nevertheless* not *the gradient of $\hat{J}$, so a finite-difference gradient check will fail with `incomplete_adjoint = 1`, and convergence tests based on $\|\nabla\hat{J}\|$ are testing the wrong quantity. For $\bar{B}$ inversion, where the control enters the viscosity directly, the approximation is less benign.*

5.3 Adjoint forcing

The right-hand side of (9) is $-\partial J/\partial \mathbf{u}$, assembled from the derivatives of the active misfit densities. For 101 this is simply $-\int \gamma_1(\mathbf{u}-\mathbf{u}^{\mathrm{obs}})\cdot\boldsymbol{\varphi}$; for 103,

$$-\frac{\partial j_{103}}{\partial \mathbf{u}}\cdot\boldsymbol{\varphi} = -\log\!\left(\frac{\|\mathbf{u}\|+\epsilon}{\|\mathbf{u}^{\mathrm{obs}}\|+\epsilon}\right)\frac{\mathbf{u}\cdot\boldsymbol{\varphi}}{\|\mathbf{u}\|\,(\|\mathbf{u}\|+\epsilon)}, \tag{15}$$

and analogously for the others. In the code these enter through the load-vector assembly of the adjoint analysis (`AdjointHorizAnalysis::CreatePVector` for SSA/HO/L1L2, `AdjointStokesAnalysis` for FS); the corresponding matrix assembly, which branches on `incomplete_adjoint`, is in `CreateKMatrix` of the same class. For 3D models the velocity misfits are surface responses and their derivatives load only the top-surface nodes.

6 Gradient assembly

6.1 Control-dependent terms

The second ingredient of (10) is $(\partial\mathbf{R}/\partial\mathbf{m})^{\mathsf{T}}\boldsymbol{\lambda}$, i.e. the explicit dependence of the residual on the control, contracted with the adjoint state. For the two standard controls this is local and cheap.

Friction coefficient. Only the basal term of (5) depends on α, and it does so pointwise:

$$\left.\frac{\mathrm{d}\hat{J}}{\mathrm{d}\alpha}\right|_{\mathrm{misfit}} = 2\,\alpha\,N^{r}\,\|\mathbf{u}\|^{\,s-1}\,(\mathbf{u}\cdot\boldsymbol{\lambda}) \qquad \text{on } \Omega_g, \tag{16}$$

zero on floating ice. The factor 2α is the chain rule through the squared parameterisation; it vanishes as $\alpha\to 0$, which is one reason a strictly positive `min_parameters` is advisable.

Ice hardness. Since μ is linear in $\bar{B}$ by (3), $\partial\mu/\partial\bar{B} = \mu/\bar{B}$ and

$$\left.\frac{\mathrm{d}\hat{J}}{\mathrm{d}\bar{B}}\right|_{\mathrm{misfit}} = \frac{H\mu}{\bar{B}}\,\mathbf{D}(\boldsymbol{\lambda})^{\mathsf{T}}\mathbf{C}\,\mathbf{D}(\mathbf{u}) = \tfrac{1}{2}H\,\dot{\varepsilon}_e^{(1-n)/n}\,\mathbf{D}(\boldsymbol{\lambda})^{\mathsf{T}}\mathbf{C}\,\mathbf{D}(\mathbf{u}). \tag{17}$$

The gradient density is thus the local viscous work pairing the forward and adjoint strain rates — it vanishes wherever the ice is not deforming, which is exactly why $\bar{B}$ is unidentifiable in slow, low-strain regions and why regularisation is mandatory there.

6.2 Regularisation terms

Terms 501–503 depend on the control only, not on $\mathbf{u}$, and contribute directly to $\partial J/\partial\mathbf{m}$. For $j_{501} = \frac{1}{2}\|\nabla\alpha\|^2$ the weak gradient is

$$\left\langle\frac{\partial J}{\partial\alpha},\delta\alpha\right\rangle = \int_\Omega \gamma_{501}\,\nabla\alpha\cdot\nabla\delta\alpha\,\mathrm{d}\Omega, \tag{18}$$

i.e. the discrete Laplacian applied to the control. Historically these were implemented as penalties; they were reorganised into the cost-function framework so that a single mechanism (the coefficient array) governs both data terms and regularisation, and so that their values appear alongside the misfits in the reported cost history. The relevant modules are `DragCoefficientAbsGradientx`, `RheologyBbarAbsGradientx`, `ThicknessAbsGradientx`.

6.3 Assembly, scaling, and constraints

The total gradient is summed in `Gradjx`, which loops over active cost functions and control parameters, and stored on the `ControlInput` object associated with each control. Three post-processing steps matter in practice:

Control scaling. `md.inversion.control_scaling_factors` rescales each control so that different controls are commensurate. This matters whenever more than one control is active: $\alpha \sim \mathcal{O}(10\text{–}10^2)$ while $\bar{B} \sim \mathcal{O}(10^8)\,\mathrm{Pa\,s^{1/n}}$, so an unscaled quasi-Newton method sees a condition number of $\sim 10^{14}$ purely from units and will make no progress on one of them.

Box constraints. The bounds in (8) are enforced by clipping the control vector to $[\mathbf{m}^{\min}, \mathbf{m}^{\max}]$ and suppressing gradient components at active bounds that would push further outside. M1QN3 itself is an unconstrained solver, so this projection is applied around it; TAO's `blmvm` handles bounds natively.

Vertical projection. For 3D models, depth-averaged controls live on the 2D footprint and are extruded; gradients are correspondingly collapsed. This is the `project3d` pathway.

Remark 3 (Discrete versus Riesz gradient)**.** *The quantity assembled is the derivative with respect to* nodal values*, $\partial\hat{J}/\partial m_i$, not the L^2 Riesz representative $\mathbf{M}^{-1}\partial\hat{J}/\partial\mathbf{m}$ with $\mathbf{M}$ the mass matrix. On a strongly graded mesh these differ by the local vertex area, so the effective step size — and hence the effective regularisation — is mesh-dependent. This is worth being aware of when comparing inversions at different resolutions, or when a mesh has been refined by a factor of ten in an outlet glacier. Verify against the revision in use whether any mass-matrix preconditioning is applied before the gradient is handed to the optimiser.*

7 Optimisation drivers

The choice of algorithm is made by the *class* assigned to `md.inversion`. Three classes exist, with different fields.

Table 2: Optimisation drivers and their controlling fields.

Class	Core file	Key fields
`inversion`	`control_core.cpp`	`nsteps`, `maxiter_per_step`, `step_threshold`, `gradient_scaling`
`taoinversion`	`controltao_core.cpp`	`algorithm`, `maxsteps`, `maxiter`, `fatol`, `frtol`, `gatol`, `grtol`, `gttol`
`m1qn3inversion`	`controlm1qn3_core.cpp`	`maxsteps`, `maxiter`, `dxmin`, `gttol`, `control_scaling_factors`

7.1 Brent search (legacy)

The original `inversion` class performs steepest descent with a Brent line search along $-\nabla\hat{J}$, scaled by `gradient_scaling`. `nsteps` counts gradient evaluations, `maxiter_per_step` bounds the line-search evaluations per step, and `step_threshold` sets the fractional decrease required to accept a step. It is robust but slow: steepest descent on a problem with the conditioning of (8) converges linearly with a rate governed by the (very large) condition number of the reduced Hessian.

7.2 TAO

`taoinversion` routes the reduced problem to PETSc's TAO optimisation package [9], with `blmvm` (bound-constrained limited-memory variable metric) the usual choice, and `lmvm`, `cg`, `tron` also exposed. The advantage is native handling of the bound constraints and access to PETSc's convergence machinery; the disadvantage is a hard dependency on a PETSc build with TAO enabled.

7.3 M1QN3 (default in practice)

`m1qn3inversion` interfaces M1QN3 [1], INRIA's limited-memory quasi-Newton code (an L-BFGS variant from the Modulopt library), following the design of the YAO variational-assimilation framework [10]. This is the recommended path for production inversions.

M1QN3 is a reverse-communication solver: it repeatedly returns control to a user-supplied simulator that must evaluate $\hat{J}$ and $\nabla\hat{J}$ at a requested point. In ISSM that simulator (in `controlm1qn3_core.cpp`) performs:

Algorithm 1 One M1QN3 simulator call in ISSM

1: receive scaled control vector $\mathbf{x}$ from M1QN3
2: unscale $\mathbf{x}$ by `control_scaling_factors` to get $\mathbf{m}$; clip to $[\mathbf{m}^{\min}, \mathbf{m}^{\max}]$
3: write $\mathbf{m}$ into the model inputs (`ControlInput` objects)
4: solve the *nonlinear* stress balance for $\mathbf{u}(\mathbf{m})$ ▷ Picard or Newton
5: evaluate each active cost function J_i; form $J = \sum_i \int \gamma_i j_i$
6: assemble adjoint load $-\partial J/\partial \mathbf{u}$ and matrix $\partial \mathbf{R}/\partial \mathbf{u}$
(matrix branches on `incomplete_adjoint`)
7: solve the linear adjoint system for $\boldsymbol{\lambda}$
8: assemble $\nabla\hat{J}$ via (10); add regularisation terms
9: apply control scaling; zero components at active bounds
10: return $(J, \nabla\hat{J})$ to M1QN3

M1QN3 then updates its limited-memory inverse-Hessian approximation from the accumulated $\{(\mathbf{s}_k, \mathbf{y}_k)\}$ pairs and performs a Wolfe-condition line search. Termination is governed by:

- `maxsteps` — maximum number of *gradient* evaluations (outer iterations);
- `maxiter` — maximum number of *function* evaluations (forward solves, including line-search trials; necessarily $\geq$ `maxsteps`);
- `dxmin` — two points closer than this in sup-norm are treated as identical, which in practice terminates the line search;
- `gttol` — relative gradient-norm reduction $\|\nabla\hat{J}_k\|/\|\nabla\hat{J}_0\| \leq$ `gttol`.

Remark 4. *Because the reduced problem is ill-posed, $\|\nabla\hat{J}\|$ typically does* not *fall by many orders of magnitude, and inversions are usually terminated on `maxsteps` rather than a gradient criterion. Convergence of the* misfit *is the practical criterion, and the cost-function history returned in `md.results.StressbalanceSolution.J` (one column per active cost function, plus the total) is the object to inspect. A misfit that has plateaued while the regularisation term is still falling means the regularisation weight is too high.*

8 Regularisation and its selection

Inverse problems of this type are ill-posed in the classical sense: the forward map is smoothing (basal friction perturbations at short wavelength are attenuated through the ice column by a factor decaying with wavenumber), so its inverse amplifies short-wavelength noise without bound. Absent regularisation the inferred field is dominated by mesh-scale oscillations that are artefacts, not structure.

ISSM's regularisation is Tikhonov of the first order, applied via cost functions 501/502/507:

$$\hat{J}_\gamma(\mathbf{m}) = \underbrace{\sum_{i\in\text{misfit}} \int \gamma_i j_i}_{\mathcal{M}(\mathbf{m})} + \frac{\gamma_{\text{reg}}}{2} \int \|\nabla \mathbf{m}\|^2. \tag{19}$$

The choice of γ_{reg} is a genuine modelling decision, not a numerical detail: it sets the resolution length scale of the recovered field. The standard objective procedure is the L-curve [2]: run the inversion over a logarithmic sweep of γ_{reg}, plot $\log \mathcal{M}$ against $\log \int \|\nabla \mathbf{m}\|^2$ at each converged solution, and select the corner of the resulting L-shaped locus, where further smoothing begins to cost misfit disproportionately. This requires $\mathcal{O}(10)$ full inversions and is worth budgeting for; it is not automated in ISSM.

Two practical caveats. First, the L-curve corner is sharp only when the noise is roughly homogeneous; on real velocity mosaics with spatially varying error it is often rounded, and the choice becomes partly subjective. Second, the tutorial's own conclusion is worth internalising: regularisation that is adequate to suppress noise will also smooth genuine sharp transitions, so an inferred friction field should not be read as resolving structure at the mesh scale.

9 Algorithmic differentiation

9.1 Motivation

The hand-coded adjoint of Section 5 covers the steady stress balance with a fixed set of controls. Anything outside that envelope — transient assimilation over a multi-year trajectory, sensitivity of a scalar diagnostic (ice volume, grounding-line flux) to an arbitrary input, coupled thermomechanical or hydrological sensitivities — requires an adjoint of the whole solution sequence, including time stepping. Deriving and maintaining that by hand is impractical. ISSM therefore also supports operator-overloading AD of the C++ core [5].

9.2 Mechanics

The AD build replaces the scalar type `IssmDouble` with an active AD type and records a tape of elementary operations during the forward run; the reverse sweep then propagates adjoints. Key elements:

- **Tools.** Originally ADOL-C with Adjoinable MPI; current builds use CoDiPack (with MeDiPack for MPI communication), enabled by `--with-codipack-dir` at configure time and detectable at runtime via `IssmConfig('_HAVE_CODIPACK_')`. AD builds are compiled separately from production builds — the active type propagates through the entire core, so one cannot mix.

- **External function for the linear solve.** Taping every operation inside a sparse direct or Krylov solver would be ruinous. Instead the linear solve is registered as an external differentiated function whose adjoint is supplied analytically: if $A\mathbf{x} = \mathbf{b}$, then $\bar{\mathbf{b}} = A^{-\mathsf{T}}\bar{\mathbf{x}}$ and $\bar{A} = -\bar{\mathbf{b}}\mathbf{x}^{\mathsf{T}}$. This is the `EDF_for_solverx` family of hooks and is the single most important design decision in the AD implementation.

- **Model interface.** `md.autodiff.isautodiff`, with `md.autodiff.independents` (a list of `independent` objects naming model fields, e.g. `md.geometry.thickness`, with a type of `'vertex'` or `'scalar'`) and `md.autodiff.dependents` (a list of `dependent` objects naming outputs, e.g. `IceVolume`, `MaxVel`).

- **Drivers.** `md.autodiff.driver` selects `'fos_forward'` (first-order scalar forward, one directional derivative), `'fov_forward'` (first-order vector forward, several directions at once, with `'fov_forward_indices'` on the independent), `'fos_reverse'` (first-order scalar reverse, with `'fos_reverse_index'` on the dependent), and `'fov_reverse'`. Reverse mode is the one that scales to large control vectors; forward mode is used chiefly for verification. Note the seed index migrated from the independent (forward) to the dependent (reverse) when CoDiPack support was added, so older scripts need adjustment.

- **Verification.** The `3xxx` series in `$ISSM_DIR/test/NightlyRun` exercises AD against step-wise finite differences; `test3015` and `test3019` are representative and show both branches (ADOL-C-era forward mode and CoDiPack reverse mode).

9.3 Trade-offs

AD gives the *exact discrete* adjoint by construction — no incomplete-adjoint approximation, no risk of the hand-coded derivative drifting out of sync with the forward code after a physics change. The costs are: a substantially larger memory footprint (the tape scales with the number of recorded operations, so long transients require checkpointing); a slower forward run; a more fragile build (AD tools impose C++ standard and compiler constraints that interact badly with some HPC toolchains, and the MPI layer must be intercepted consistently); and reduced portability across clusters. For a steady-state friction inversion the hand-coded control method is faster and entirely adequate; AD earns its cost for transient assimilation and for sensitivities that no hand-coded adjoint exists for.

10 Related inverse capabilities

Beyond the stress-balance control problem, the same optimisation infrastructure is reused for:

Balance thickness. Cost function 506 with `BalancethicknessThickeningRate` or `Vx`/`Vy` as controls solves a constrained mass-conservation problem, adjusting velocity or apparent mass balance so that $\nabla\cdot(H\mathbf{u}) = \dot{a} - \partial_t H$ is consistent with observed thickness. This is the machinery behind mass-conserving bed reconstructions.

Surface-elevation assimilation. The $6xx$ family assimilates altimetric surface elevation, used for example to infer surface mass balance jointly with friction from ICESat data [4].

Damage. `DamageDbar` can be taken as a control, inferring a damage field in place of (or alongside) a hardness anomaly on ice shelves.

Multiple simultaneous controls. `md.inversion.control_parameters` accepts a list; `min_parameters`/`max_parameters` then take one column per control. Simultaneous α and $\bar{B}$ inversion is possible but poorly conditioned — they trade off against each other in shear margins — and is normally avoided by restricting α to grounded ice and $\bar{B}$ to floating ice, which the gradients (16)–(17) already do naturally through their supports.

11 A minimal workflow

The following sketch shows the shape of a Python-interface inversion; it is illustrative rather than runnable.

Listing 1: Structure of a friction inversion.

```
from m1qn3inversion import m1qn3inversion
import numpy as np

nv = md.mesh.numberofvertices

md.inversion = m1qn3inversion(md.inversion)
md.inversion.iscontrol = 1
md.inversion.incomplete_adjoint = 1 # 1: frozen viscosity; 0: exact tangent

# --- observations ------------------------------------------------------
md.inversion.vx_obs = vx_obs # e.g. interpolated from ITS_LIVE / MEaSUREs
md.inversion.vy_obs = vy_obs
md.inversion.vel_obs = np.sqrt(vx_obs**2 + vy_obs**2)

# --- control -----------------------------------------------------------
md.inversion.control_parameters = ['FrictionCoefficient']
md.inversion.min_parameters = 1.0 * np.ones((nv, 1))
md.inversion.max_parameters = 200.0 * np.ones((nv, 1))

# --- objective: absolute + logarithmic misfit + Tikhonov ---------------
md.inversion.cost_functions = [101, 103, 501]
gamma = np.ones((nv, 3))
gamma[:, 0] = 3.0e3 # absolute misfit
gamma[:, 1] = 1.0 # log misfit
gamma[:, 2] = 1.0e-2 # regularisation <- sweep this for the L-curve
# zero the data weights where there are no observations:
nodata = np.nonzero(np.isnan(vx_obs))[0]
gamma[nodata, 0:2] = 0.0
md.inversion.cost_functions_coefficients = gamma

# --- optimiser ---------------------------------------------------------
md.inversion.maxsteps = 40 # gradient evaluations
md.inversion.maxiter = 40 # forward solves
md.inversion.dxmin = 1.0e-6
md.inversion.gttol = 1.0e-6

md = solve(md, 'Stressbalance')

alpha = md.results.StressbalanceSolution.FrictionCoefficient
Jhist = md.results.StressbalanceSolution.J # per-term cost history
md.friction.coefficient = alpha # carry into the forward model
```

Post-inversion, the inferred field must be written back into the relevant model class (`md.friction.coefficient` or `md.materials.rheology_Bbar`) before any forward run; the result object does not update the input automatically.

12 Verification

Two checks are worth performing on any new inversion setup.

Gradient check. Choose a random unit direction $\delta\mathbf{m}$ and verify

$$\left| \frac{\hat{J}(\mathbf{m} + h\,\delta\mathbf{m}) - \hat{J}(\mathbf{m})}{h} - \nabla\hat{J}(\mathbf{m}) \cdot \delta\mathbf{m} \right| = \mathcal{O}(h) \tag{20}$$

over several decades of h, with the classic V-shaped error curve bottoming out at $h \sim \sqrt{\epsilon_{\text{mach}}}$ before round-off dominates. Two provisos: this must be done with `incomplete_adjoint = 0`, since the incomplete gradient is not the gradient of $\hat{J}$; and the forward nonlinear solve must be converged tightly enough (`md.stressbalance.restol` small) that the residual tolerance does not contaminate the finite difference at the smallest h.

Twin experiment. Prescribe a synthetic control field, run the forward model, use the resulting velocities as observations, reinitialise the control to a constant, and check recovery. This is the structure of the official ISSM inversion tutorial and it exposes the regularisation trade-off directly: the pattern is recovered, sharp transitions are not.

13 Claims to verify against your revision

The following are version-sensitive; the ISSM code base has evolved substantially and file layouts have moved at least twice (SVN `trunk` → `trunk-jpl` → GitHub).

1. Cost-function codes 602–604: the supported list includes them but their densities are stated here only for 601. Check `src/m/inversions/marshallcostfunctions.m` and the corresponding `*x` modules.
2. The full contents of `supportedcontrols.{m,py}` — the set of admissible control names has grown with each new friction law (Schoof, Coulomb-limited, hydrology-coupled).
3. Whether any mass-matrix (Riesz) preconditioning is applied to the assembled gradient before it reaches the optimiser (see Remark on discrete versus Riesz gradients).
4. The exact mechanism by which box constraints are enforced around M1QN3 — clipping of the control, masking of the gradient, or both — in `controlm1qn3_core.cpp`.
5. The precise value and placement of `epsvel` in the relative and logarithmic misfits; this affects the behaviour of the inversion in slow-flow regions materially.
6. Whether the `inversion` (Brent) class is still the default constructed by `model()`; the online documentation says so, but tutorials and current practice use `m1qn3inversion` throughout.
7. Any transient/time-dependent inversion capability in the control-method path (as distinct from the AD path), which has been an active area of development.

References

[1] J. C. Gilbert and C. Lemaréchal. Some numerical experiments with variable-storage quasi-Newton algorithms. *Mathematical Programming*, 45(1–3):407–435, 1989.

[2] M. Habermann, M. Truffer, and D. Maxwell. Changing basal conditions during the speed-up of Jakobshavn Isbræ, Greenland. *The Cryosphere*, 2012. (See also Habermann et al., *J. Glaciol.*, 2012, on L-curve selection for basal inversions.)

[3] E. Larour, H. Seroussi, M. Morlighem, and E. Rignot. 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), 2012.

[4] E. Larour, J. Utke, B. Csatho, A. Schenk, H. Seroussi, M. Morlighem, E. Rignot, N. Schlegel, and A. Khazendar. Inferred basal friction and surface mass balance of the Northeast Greenland Ice Stream using data assimilation of ICESat surface altimetry and ISSM. *The Cryosphere*, 8:2335–2351, 2014.

[5] E. Larour, J. Utke, A. Bovin, M. Morlighem, and G. Perez. An approach to computing discrete adjoints for MPI-parallelized models applied to Ice Sheet System Model 4.11. *Geoscientific Model Development*, 9:3907–3918, 2016.

[6] D. R. MacAyeal. A tutorial on the use of control methods in ice-sheet modeling. *Journal of Glaciology*, 39(131):91–98, 1993.

[7] M. Morlighem, E. Rignot, H. Seroussi, E. Larour, H. Ben Dhia, and D. Aubry. Spatial patterns of basal drag inferred using control methods from a full-Stokes and simpler models for Pine Island Glacier, West Antarctica. *Geophysical Research Letters*, 37(14), 2010.

[8] M. Morlighem, H. Seroussi, E. Larour, and E. Rignot. Inversion of basal friction in Antarctica using exact and incomplete adjoints of a higher-order model. *Journal of Geophysical Research: Earth Surface*, 118(3):1746–1753, 2013.

[9] T. Munson, J. Sarich, S. Wild, S. Benson, and L. Curfman McInnes. TAO 2.0 Users Manual. Technical Report ANL/MCS-TM-322, Argonne National Laboratory, 2012.

[10] L. Nardi, C. Sorror, F. Badran, and S. Thiria. YAO: A software for variational data assimilation using numerical models. In *ICCSA 2009*, LNCS 5593, pages 621–636. Springer, 2009.

[11] ISSM Team. Inverse methods — ISSM documentation. `https://issm.jpl.nasa.gov/documentation/inversions/` and `https://issm.jpl.nasa.gov/documentation/tutorials/inversion/`.