PISM (a Parallel Ice Sheet Model) Reference Manual

beta

Generated by Doxygen 1.5.1

Fri Oct 5 14:38:34 2007

Contents

1 PISM (a Parallel Ice Sheet Model) Main Page

Author:

Ed Bueler

This Reference Manual is a greatly shortened version of the complete HTML PISM source browser at `pism/doc/doxy/html/index.html`. As with the source browser, this Manual was automatically generated by doxygen (`http://www.doxygen.org/`) from comments in the PISM source code.

This Manual documents the *continuum models* and the *numerical methods* in PISM. It is restricted to documenting the three classes MaterialType (and its derived classes), IceGrid (and its associated class IceParam), and the main PISM class IceModel. The source files from which this Manual is drawn are limited to files in the `pism/src/base/` subdirectory. Note that, by contrast, the source browser describes *all* classes and *all* class members and *all* source files in PISM.

Most users should start with the PISM User's Manual (`http://www.dms.uaf.edu/~bueler/manual.pdf` or `pism/doc/manual.pdf`) for help with installing and using PISM.

2 PISM (a Parallel Ice Sheet Model) Hierarchical Index

2.1 PISM (a Parallel Ice Sheet Model) Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

3 PISM (a Parallel Ice Sheet Model) Class Index

3.1 PISM (a Parallel Ice Sheet Model) Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

4 PISM (a Parallel Ice Sheet Model) Class Documentation

4.1 IceEISModel Class Reference

This derived class does EISMINT II simplified geometry experiments.

Inherits IceModel.

Inherited by IceRYANModel.

4.1.1 Detailed Description

This derived class does EISMINT II simplified geometry experiments.

These experiments only involve the thermomechanically coupled shallow ice approximation. See A. J. Payne and ten others, 200. *Results from the EISMINT model intercomparison: the effects of thermomechanical coupling*. J. Glaciol. 46(153), 227–238.

4.2 IceGrid Class Reference

Describes the PISM grid and the distribution of data across processors.

Public Member Functions

- PetscErrorCode createDA ()
- PetscErrorCode rescale (const PetscScalar lx, const PetscScalar ly, const PetscScalar lz)
- PetscErrorCode rescale (const PetscScalar lx, const PetscScalar ly, const PetscScalar lz, const PetscTruth truelyPeriodic)

4.2.1 Detailed Description

Describes the PISM grid and the distribution of data across processors.

This class creates and destroys the PETSc DAs (distributed arrays) which control how all PISM variables are distributed across multiple processors. This class also contains the parameters which describe the PISM computational box and the three-dimensional PISM finite difference grid.

4.2.2 Member Function Documentation

4.2.2.1 PetscErrorCode createDA ()

Create the PETSc DAs for the grid specified in ∗p (a pointer to IceParam).

4.2.2.2 PetscErrorCode rescale (const PetscScalar *lx*, const PetscScalar *ly*, const PetscScalar *lz*)

Rescale IceGrid based on new values for `Lx`, `Ly`, `Lz` (default version).

This method computes `dx`, `dy`, `dz`, and `Lbz` based on the current values of `Mx`, `My`, `Mz`, `Mbz` and the input values of `Lx`, `Ly`, and `Lz`. Note that `dz` is the vertical spacing both in the ice and in bedrock. Thus `Lbz = Mbz * dz` determines `Lbz`.

See the comment for `rescale(lx,ly,lz,truelyPeriodic)`.

4.2.2.3 PetscErrorCode rescale (const PetscScalar *lx*, const PetscScalar *ly*, const PetscScalar *lz*, const PetscTruth *truelyPeriodic*)

Rescale IceGrid based on new values for `Lx`, `Ly`,`Lz`, but optionally allowing for periodicity.

The grid used in PISM, in particular the PETSc DAs used here, are periodic in x and y. This means that the ghosted values `foo[i+1][j]`, `foo[i-1][j]`, `foo[i][j+1]`, `foo[i][j-1]` for all 2D Vecs, and similarly in the x and y directions for 3D Vecs, are always available. That is, they are available even if i,j is a point at the edge of the grid. On the other hand, by default, `dx` is the full width `2 * Lx` divided by `Mx - 1`. This means that we conceive of the computational domain as starting at the `i = 0` grid location and ending at the `i = Mx - 1` grid location, in particular. This idea is not quite compatible with the periodic nature of the grid.

The upshot is that if one computes in a truely periodic way then the gap between the `i = 0` and `i = Mx - 1` grid points should *also* have width `dx`. Thus we compute `dx = 2 * Lx / Mx`.

4.3 IceModel Class Reference

The base class for PISM. Contains all essential variables, parameters, and flags for modelling an ice sheet.

Inherited by IceCompModel, IceDragYieldModel, IceEISModel, IceExactSSAModel, IceGRNModel, IceHEINOModel, IceROSSModel, and ShelfModel.

Public Member Functions

- virtual PetscErrorCode run ()
- virtual PetscErrorCode diagnosticRun ()
- virtual PetscErrorCode setDefaults ()
- virtual PetscErrorCode setFromOptions ()
- PetscErrorCode velocity (bool updateSIAVelocityAtDepth)
- PetscErrorCode initFromFile (const char *)
- PetscErrorCode writeFiles (const char *defaultbasename, const PetscTruth forceFullDiagnostics)
- PetscErrorCode initFromFile_netCDF (const char *fname)

Protected Member Functions

- virtual PetscErrorCode createVecs ()
- virtual PetscErrorCode destroyVecs ()
- PetscErrorCode updateSurfaceElevationAndMask ()
- PetscErrorCode massBalExplicitStep ()

- PetscErrorCode updateYieldStressFromHmelt ()
- virtual PetscScalar basalVelocity (const PetscScalar x, const PetscScalar y, const PetscScalar H, const PetscScalar T, const PetscScalar alpha, const PetscScalar mu)
- PetscErrorCode writeMatlabVars (const char *fname)
- PetscErrorCode writeSSAsystemMatlab (Vec vNu[2])
- PetscErrorCode surfaceGradientSIA ()
- PetscErrorCode velocitySIAStaggered (bool faststep)
- PetscErrorCode frictionalHeatingSIAStaggered ()
- PetscErrorCode velocities2DSIAToRegular ()
- PetscErrorCode SigmaSIAToRegular ()
- PetscErrorCode horizontalVelocitySIARegular ()
- PetscErrorCode velocitySSA ()
- virtual PetscErrorCode computeEffectiveViscosity (Vec vNu[2], PetscReal epsilon)
- PetscErrorCode assembleSSAMatrix (Vec vNu[2], Mat A)
- PetscErrorCode assembleSSARhs (bool surfGradInward, Vec rhs)
- PetscErrorCode moveVelocityToDAVectors (Vec x)
- PetscErrorCode broadcastSSAVelocity ()
- PetscErrorCode correctSigma ()
- PetscErrorCode correctBasalFrictionalHeating ()
- PetscErrorCode saveUVBarForSSA (const PetscTruth firstTime)
- PetscErrorCode putSavedUVBarInUVBar ()
- PetscErrorCode temperatureAgeStep ()
- virtual PetscErrorCode temperatureStep ()
- PetscErrorCode ageStep (PetscScalar *CFLviol)
- PetscErrorCode excessToFromBasalMeltLayer (PetscScalar rho_c, PetscScalar z, PetscScalar *Texcess, PetscScalar *Hmelt)
- PetscErrorCode vertVelocityFromIncompressibility ()
- PetscErrorCode computeMax3DVelocities ()
- PetscErrorCode updateViewers ()

Static Protected Attributes

- static const titleNname tn [tnN]

4.3.1 Detailed Description

The base class for PISM. Contains all essential variables, parameters, and flags for modelling an ice sheet.

FIXME: add text about change of vertical variable

4.3.2 Member Function Documentation

4.3.2.1 PetscErrorCode run () [virtual]

Do the time-stepping for an evolution run.

This important routine can be replaced by derived classes; it is `virtual`.

This procedure has the main loop. The following actions are taken on each pass through the loop:

- the yield stress for the plastic till model is updated (if appropriate)
- the positive degree day model is invoked to compute the surface mass balance (if appropriate)
- a step of the bed deformation model is taken (if appropriate)
- the velocity field is updated; in some cases the whole three-dimensional field is updated and in some cases just the vertically-averaged horizontal velocity is updated; see velocity()
- the time step is determined according to a variety of stability criteria; see determineTimeStep()
- the temperature field is updated according to the conservation of energy model based (especially) on the new velocity field; see temperatureAgeStep()
- the thickness of the ice is updated according to the mass conservation model; see massBalExplicitStep()
- there is various reporting to the user on the current state; see summary() and updateViewers()

Note that at the beginning and ends of the loop there is a chance for derived classes to do extra work. See additionalAtStartTimestep() and additionalAtEndTimestep().

4.3.2.2 PetscErrorCode diagnosticRun () [virtual]

Calls the necessary routines to do a diagnostic calculation of velocity.

This important routine can be replaced by derived classes; it is `virtual`.

This procedure has no loop but the following actions are taken:

- the yield stress for the plastic till model is updated (if appropriate)
- the velocity field is updated; in some cases the whole three-dimensional field is updated and in some cases just the vertically-averaged horizontal velocity is updated; see velocity()
- there is various reporting to the user on the current state; see summary() and updateViewers()

4.3.2.3 PetscErrorCode setDefaults () [virtual]

Assigns default values to the many parameters and flags in IceModel.

4.3.2.4 PetscErrorCode setFromOptions () [virtual]

Read runtime (command line) options and set the corresponding parameter or flag.

This is called by a driver program, assuming it would like to use command line options.

In fact this procedure only reads the majority of the options. Some are read in initFromOptions(), writeFiles(), and setStartRunEndYearsFromOptions(), among other places.

Note there are no options to directly set `dx`, `dy`, `dz`, `Lbz`, and `year` as the user should not directly set these grid parameters. There are, however, options for directly setting `Mx`, `My`, `Mz`, `Mbz` and also `Lx`, `Ly`, `Lz`.

4.3.2.5 PetscErrorCode velocity (bool *updateVelocityAtDepth*)

Manage the computation of velocity and do the necessary communication.

This procedure calls the important routines velocitySIAStaggered(), velocitySSA(), and vertVelocityFromIncompressibility() according to various flags.

4.3.2.6 PetscErrorCode initFromFile (const char * *fname*)

Initialize from a saved PISM model state (in NetCDF format).

Calls initFromFile_netCDF() to do the actual work.

4.3.2.7 PetscErrorCode writeFiles (const char * *defaultbasename*, const PetscTruth *forceFullDiagnostics*)

Save model state in NetCDF format (and save variables in Matlab format if desired).

Optionally allows saving of full velocity field.

Calls dumpToFile_netCDF(), dumpToFile_diagnostic_netCDF(), and writeMatlabVars() to do the actual work.

4.3.2.8 PetscErrorCode initFromFile_netCDF (const char * *fname*)

Read a complete saved PISM model state for initialization of an evolution or diagnostic run.

When initializing from a NetCDF input file, the file determines the number of grid points (Mx,My,Mz,Mbz) and the dimensions of the computational box. The user is warned when their command line options "-Mx", "-My", "-Mz", "-Mbz" are overridden.

4.3.2.9 PetscErrorCode createVecs () [protected, virtual]

Allocate all Vecs defined in IceModel.

Initialization of an IceModel is confusing. Here is a description of the intended order:

- 1. The constructor for IceModel. Note IceModel has a member "grid", of class IceGrid. Note grid.p, an IceParam, is also constructed; this sets defaults for (grid.p->)Mx,My,Mz,Mbz,Lx,Ly,Lz,Lbz,dx,dy,dz,year.
- [1.5] derivedClass::setFromOptions() to get options special to derived class
- 2. setFromOptions() to get all options *including* Mx,My,Mz,Mbz
- [2.5] initFromFile_netCDF() which reads Mx,My,Mz,Mbz from file and overwrites previous; if this represents a change the user is warned
- 3. createDA(), which uses only Mx,My,Mz,Mbz
- 4. createVecs() uses DA to create/allocate Vecs
- [4.5] derivedClass:createVecs() to create/allocate Vecs special to derived class
- 5. afterInitHook() which changes Lx,Ly,Lz if set by user

Note driver programs call only setFromOptions() and initFromOptions() (for IceModel or derived class).

Note IceModel::setFromOptions() should be called at the end of derivedClass:setFromOptions().

Note 2.5, 3, and 4 are called from initFromFile() in IceModel.

Note 3 and 4 are called from initFromOptions() in some derived classes (e.g. IceCompModel) in cases where initFromFile() is not called.

Note step 2.5 is skipped when bootstrapping (-bif and bootstrapFromFile_netCDF()) or in those derived classes which can start with no input files, e.g. IceCompModel and IceEISModel. That is, 2.5 is only done when starting from a saved model state.

4.3.2.10 PetscErrorCode destroyVecs () [protected, virtual]

De-allocate all Vecs defined in IceModel.

Undoes the actions of createVecs().

4.3.2.11 PetscErrorCode updateSurfaceElevationAndMask () [protected]

Update the surface elevation and the flow-type mask when the geometry has changed.

This procedure should be called whenever necessary to maintain consistency of geometry. For instance, it should be called when either ice thickness or bed elevation change. In particular we always want $h = H + b$ to apply at grounded points, and, on the other hand, we want the mask to reflect that the ice is floating if the floatation criterion applies at a point.

There is one difficult case. When a point was floating and becomes grounded we generally do not know whether to mark it as `MASK_SHEET` so that the SIA applies or `MASK_DRAGGING` so that the SSA applies. For now there is a vote-by-neighbors scheme (among the grounded neighbors). When the `MASK_DRAGGING` points have plastic till bases this is not an issue.

4.3.2.12 PetscErrorCode massBalExplicitStep () `[protected]`

Update the thickness from the velocity field and the surface and basal mass balance.

The partial differential equation describing the conservation of mass in the map-plane (parallel to the geoid) is

$$\frac{\partial H}{\partial t} = M - S - \nabla \cdot \mathbf{q}$$

where

$$\mathbf{q} = \bar{\mathbf{U}} H.$$

In these equations H is the ice thickness, M is the surface mass balance (accumulation or ablation), S is the basal mass balance (e.g. basal melt), and $\bar{\mathbf{U}}$ is the vertically-averaged horizontal velocity of the ice. Note $\bar{\mathbf{U}} = (\bar{u}, \bar{v})$ in the notation used here. The map-plane flux of the ice $\mathbf{q}$ is defined by the above formula.

This procedure massBalExplicitStep() uses this conservation of mass equation to update the ice thickness.

This method is first-order explicit in time. The derivatives in $\nabla \cdot \mathbf{q}$ are computed by centered finite difference methods. In the case of the SIA the value of $\bar{\mathbf{U}}$ is already stored on the staggered grid by velocitySIAStaggered() and it is differenced in the obvious centered manner (with averaging of the thickness onto the staggered grid).

In the case of SSA locations the $\nabla \cdot \mathbf{q}$ term is computed by upwinding after expanding

$$\nabla \cdot \mathbf{q} = \bar{\mathbf{U}} \cdot \nabla H + (\nabla \cdot \bar{\mathbf{U}}) H.$$

That is, the mass conservation equation is regarded as an advection equation with source term,

$$\frac{\partial H}{\partial t} + \bar{\mathbf{U}} \cdot \nabla H = M - S - (\nabla \cdot \bar{\mathbf{U}}) H.$$

The product of velocity and the gradient of thickness on the left is computed by first-order upwinding. Note that the CFL condition for this advection scheme is checked; see determineTimeStep().

Note that if the point is flagged as `MASK_FLOATING_OCEAN0` then the thickness is set to zero. Note that the rate of thickness change $\partial H / \partial t$ is computed and saved, as

is the rate of volume loss or gain. Note updateSurfaceElevationAndMask() is called at the end.

4.3.2.13 PetscErrorCode updateYieldStressFromHmelt () [protected]

Update the till yield stress for the plastic till model, based on pressure and stored till water.

We implement formula (2.4) in C. Schoof 2006 "A variational approach to ice stream flow", J. Fluid Mech. vol 556 pp 227–251. That formula is

$$\tau_c = \mu(\rho g H - p_w)$$

We modify it by:

(1) adding a small till cohesion c_0 (see Paterson 3rd ed table 8.1);

(2) replacing $p_w \to \lambda p_w$ where $\lambda =$ Hmelt / DEFAULT_MAX_HMELT; thus $0 \le \lambda \le 1$ always while $\lambda = 0$ when the bed is frozen; and

(3) computing porewater pressure p_w as a fixed fraction φ of the overburden pressure $\rho g H$.

With these replacements our formula looks like

$$\tau_c = c_0 + \mu\left(1 - \lambda\varphi\right)\rho g H$$

Note also that $\mu = \tan(\theta)$ where θ is a "friction angle". The parameters c_0, φ, θ can be set by options -till_cohesion, -till_pw_fraction, and -till_friction_angle, respectively.

4.3.2.14 PetscScalar basalVelocity (const PetscScalar *x*, const PetscScalar *y*, const PetscScalar *H*, const PetscScalar *T*, const PetscScalar *alpha*, const PetscScalar *mu*) [protected, virtual]

Compute the coefficient for the basal velocity in SIA regions.

In SIA regions a basal sliding law of the form

$$\mathbf{U}_b = (u_b, v_b) = -C\nabla h$$

is allowed. Here $\mathbf{U}_b$ is the horizontal velocity of the base of the ice (the "sliding velocity") and h is the elevation of the ice surface. This procedure returns the *positive* coefficient C in this relationship. This coefficient can depend of the thickness, the basal temperature, and the horizontal location.

This procedure is virtual and can be replaced by any derived class.

The default version for IceModel here is location-independent pressure-melting-temperature-activated linear sliding.

4.3.2.15 PetscErrorCode writeMatlabVars (const char * *fname*) [protected]

Write out selected variables in Matlab .m format.

Writes out the independent variables `year`, `x`, and `y`. Then writes out variables selected with option `-matv` using single character names. See Appendix C of the User's Manual.

Writes these to `foo.m` if option `-mato foo` is given.

4.3.2.16 PetscErrorCode writeSSAsystemMatlab (Vec *vNu*[2]) [protected]

Write out the linear system of equations for the last nonlinear iteration (for SSA).

Writes out the matrix, the right-hand side, and the solution vector in a *Matlab-readable* format, a `.m` file.

4.3.2.17 PetscErrorCode surfaceGradientSIA () [protected]

Compute the surface gradient in advance of the SIA velocity computation.

There are two methods for computing the surface gradient. The default is to transform the thickness to something more regular and differentiate that. In particular, as shown in (Calvo et al 2002) for the flat bed and $n = 3$ case, if we define

$$\eta = H^{(2n+2)/n}$$

then η is more regular near the margin than H. So the default method for computing the surface gradient is to compute

$$\nabla h = \frac{n}{(2n+2)} \eta^{(-n-2)/(2n+2)} \nabla \eta + \nabla b,$$

recalling that $h = H + b$. This method is only applied when $\eta > 0$ at a given point; otherwise $\nabla h = \nabla b$.

We are computing this gradient by finite differences onto a staggered grid. We do so by centered differences using (roughly) the same method for η and b that (Mahaffy 1976) applies directly to the surface elevation h.

The optional method is to directly differentiate the surface elevation h by the (Mahaffy 1976) method.

4.3.2.18 PetscErrorCode velocitySIAStaggered (bool *faststep*) [protected]

Compute the vertically-averaged horizontal velocity according to the SIA.

See the comment for massBalExplicitStep() before reading the rest of this comment.

Note that one may write

$$\mathbf{q} = \bar{\mathbf{U}} H = D \nabla h + \mathbf{U}_b \cdot H$$

in shallow ice approximation (SIA) areas. Here h is the surface elevation of the ice $\mathbf{U}_b$ is the basal sliding velocity, and D is the diffusivity (which is computed in this method). At the end of this routine the value of the vertically-averaged horizontal velocity $\bar{\mathbf{U}}$ is known at all staggered grid points.

The surface slope ∇h is needed on the staggered grid although the surface elevation h itself is known on the regular grid. The scheme used for this is the one first proposed in the context of ice sheets by Mahaffy (1976). That is, the method is "type I" in the classification described in (Hindmarsh and Payne 1996).

This routine also calls the part of the basal dynamical model applicable to the SIA; see basalVelocity(). The basal sliding velocity is computed for all SIA points. This routine also computes the basal frictional heating and the volume strain-heating. Note that the SIA is used at all points on the grid in this routine but that the resulting vertically-averaged horizontal velocity is overwritten by different values at SSA points. See correctBasalFrictionalHeating() and correctSigma().

4.3.2.19 PetscErrorCode frictionalHeatingSIAStaggered () [protected]

Compute the rate of frictional heating where the base is sliding by assuming the shear stress is from SIA.

4.3.2.20 PetscErrorCode velocities2DSIAToRegular () [protected]

Average staggered-grid vertically-averaged horizontal velocity and basal velocity onto regular grid.

At the end of velocitySIAStaggered() the vertically-averaged horizontal velocity vuvbar[0],vuvbar[1] and the basal velocity ub,vb is available on the staggered grid. This procedure averages them onto the regular grid. Note that communication of ghosted values must occur between velocitySIAStaggered() and this procedure for the averaging to work. Only two-dimensional regular grid velocities are updated here. The full three-dimensional velocity field is not updated here but instead in horizontalVelocitySIARegular() and in vertVelocityFromIncompressibility().

4.3.2.21 PetscErrorCode SigmaSIAToRegular () [protected]

Put the basal frictional heating and the volume strain-heating onto the regular grid.

At the end of velocitySIAStaggered() the basal frictional heating and the volume strain-heating are available on the staggered grid. This procedure averages them onto the regular grid. Note that communication of ghosted values must occur between velocitySIAStaggered() and this procedure for the averaging to work.

4.3.2.22 PetscErrorCode horizontalVelocitySIARegular () [protected]

Update regular grid horizontal velocities u,v at depth for SIA regions.

In the current scheme the procedure velocitySIAStaggered() computes several scalar quantities at depth (the details of which are too complicated to explain). That is, these quantities correspond to three-dimensional arrays. This procedure takes those quantities and computes the three-dimensional arrays for the horizontal components u and v of the velocity field. The vertical component w of the velocity field is computed by vertVelocityFromIncompressibility().

4.3.2.23 PetscErrorCode velocitySSA () [protected]

Compute the vertically-averaged horizontal velocity from the shallow shelf approximation (SSA).

This is the main procedure implementing the SSA.

The outer loop (over index `k`) is the nonlinear iteration. In this loop the effective viscosity is computed by computeEffectiveViscosity() and then the linear system is set up and solved.

This procedure creates a PETSC `KSP`, it calls assembleSSAMatrix() and assembleSSARhs() to store the linear system in the `KSP`, and calls the PETSc procedure KSPSolve() to solve the linear system. Solving the linear system is also a loop, an iteration, but it occurs inside KSPSolve(). This inner loop is controlled by PETSc but the user can set the option `-ksp_rtol`, in particular, and that tolerance controls when the inner loop terminates.

The outer loop terminates when the effective viscosity is no longer changing much, according to the tolerance set by the option `-ssa_rtol`. (The outer loop also terminates when a maximum number of iterations is exceeded.)

In truth there is an outer outer loop (over index `l`). This one manages an attempt to over-regularize the effective viscosity so that the nonlinear iteration (the "outer" loop over `k`) has a chance to converge.

4.3.2.24 PetscErrorCode computeEffectiveViscosity (Vec *vNu*[2], PetscReal *epsilon*) [protected, virtual]

Compute the product of the effective viscosity ν and ice thickness H.

In PISM the product νH can be (i) constant, (ii) can be computed with a constant ice hardness $\bar{B}$ (temperature-independent) but with dependence of the viscosity on the strain rates, or (iii) it can depend on the strain rates and have a vertically-averaged ice hardness.

The flow law in ice stream and ice shelf regions must, for now, be a temperature-dependent Glen law. (That is, more general forms like Goldsby-Kohlstedt are not yet inverted.)

The effective viscosity is

$$\nu = \frac{\bar{B}}{2}\left[\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \frac{\partial u}{\partial x}\frac{\partial v}{\partial y} + \frac{1}{4}\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)^2\right]^{(1-n)/(2n)}$$

where in the temperature-dependent case

$$\bar{B} = (\text{FIXME: SOME INTEGRAL})$$

In the temperature dependent case the ice hardness is computed by the trapezoid rule.

4.3.2.25 PetscErrorCode assembleSSAMatrix (Vec *vNu*[2], Mat *A*) [protected]

Assemble the matrix (left-hand side) for the numerical approximation of the SSA equations.

The SSA equations are in their clearest form

$$-\frac{\partial T_{ij}}{\partial x_j} + \tau_{(b)i} = f_i$$

where i, j range over x, y, T_{ij} is a depth-integrated viscous stress tensor (i.e. equation (2.6) in (Schoof 2006), and following (Morland 1987)), and $\tau_{(b)i}$ are the components of the basal shear stress. Also f_i is the driving shear stress $f_i = -\rho g H \frac{\partial h}{\partial x_i}$. These equations determine velocity in a more-or-less elliptic equation manner. Here H is the ice thickness and h is the elevation of the surface of the ice.

More specifically, the SSA equations are

$$-2\frac{\partial}{\partial x}\left[\nu H\left(2\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}\right)\right] - \frac{\partial}{\partial y}\left[\nu H\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] + \tau_{(b)x} = -\rho g H \frac{\partial h}{\partial x},$$

$$-\frac{\partial}{\partial x}\left[\nu H\left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right)\right] - 2\frac{\partial}{\partial y}\left[\nu H\left(\frac{\partial u}{\partial x} + 2\frac{\partial v}{\partial y}\right)\right] + \tau_{(b)y} = -\rho g H \frac{\partial h}{\partial y},$$

where u is the x-component of the velocity and v is the y-component of the velocity. Note ν is the vertically-averaged effective viscosity of the ice.

For ice shelves $\tau_{(b)i} = 0$ (MacAyeal et al 1996). For ice streams with a basal till modelled as a plastic material, $\tau_{(b)i} = \tau_c u_i/|\mathbf{u}|$ where $\mathbf{u} = (u, v)$, $|\mathbf{u}| = \left(u^2 + v^2\right)^{1/2}$, and τ_c is the yield stress of the till (Schoof 2006). For ice streams with a basal till modelled as a linearly-viscous material, $\tau_{(b)i} = \beta u_i$ where β is the basal drag (friction) parameter (Hulbe & MacAyeal 1999).

Note that the basal shear stress appears on the *left* side of the above system. We believe this is crucial, because of its effect on the spectrum of the linear approximations of each stage. The effect on spectrum is clearest in the linearly-viscous till case (i.e.

Hulbe & MacAyeal 1999) but there seems to be an analogous effect in the plastic till case (Schoof 2006).

This method assembles the matrix for the left side of the SSA equations. The numerical method is finite difference. In particular [FIXME: explain f.d. approxs, esp. mixed derivatives]

4.3.2.26 PetscErrorCode assembleSSARhs (bool *surfGradInward*, Vec *rhs*) [protected]

Computes the right-hand side of the linear problem for the SSA equations.

The right side of the SSA equations is just

$$-\rho g H \nabla h$$

because, in particular, the basal stress is put on the left side of the system. (See comment for assembleSSAMatrix().)

The surface slope is computed by centered difference onto the regular grid, which may use periodic ghosting. (Optionally the surface slope can be computed by only differencing into the grid for points at the edge; see test I.)

Note that the grid points with mask value MASK_SHEET correspond to the trivial equations

$$\bar{u}_{ij} = \frac{uvbar[0][i-1][j] + uvbar[0][i][j]}{2},$$

and similarly for $\bar{v}_{ij}$. That is, the vertically-averaged horizontal velocity is already known for these points because it was computed (on the staggered grid) using the SIA.

4.3.2.27 PetscErrorCode moveVelocityToDAVectors (Vec *x*) [protected]

Move the solution to the SSA system into a `Vec` which is `DA` distributed.

Since the parallel layout of the vector `x` in the `KSP` representation of the linear SSA system does not in general have anything to do with the `DA`-based vectors, for the rest of IceModel, we must scatter the entire vector to all processors.

4.3.2.28 PetscErrorCode broadcastSSAVelocity () [protected]

At all SSA points, update the velocity field.

Once the vertically-averaged velocity field is computed by the SSA, this procedure updates the three-dimensional horizontal velocities u and v. (Note that w gets update later by vertVelocityFromIncompressibility().) The three-dimensional velocity field is needed, for example, so that the temperature equation can include advection. Basal velocities also get updated.

Here is where the flag doSuperpose controlled by option `-super` is relevant. If doSuperpose is true then the result of the SIA computation, already done, is added to the result of SSA.

This procedure also computes the maximum horizontal speed in the SSA areas so that the CFL condition for the upwinding (in massBalExplicitStep() and only for SSA points) can be computed.

4.3.2.29 PetscErrorCode correctSigma () [protected]

At SSA points, correct the previously-computed volume strain-heating.

4.3.2.30 PetscErrorCode correctBasalFrictionalHeating () [protected]

At SSA points, correct the previously-computed basal frictional heating.

4.3.2.31 PetscErrorCode saveUVBarForSSA (const PetscTruth *firstTime*) [protected]

Save the values of vertically-averaged horizontal velocity for the first guess in the SSA iteration.

4.3.2.32 PetscErrorCode putSavedUVBarInUVBar () [protected]

For all SSA points, put previous SSA velocities in vubar,vvbar.

4.3.2.33 PetscErrorCode temperatureAgeStep () [protected]

Manages the time-stepping and parallel communication for the temperature and age equations.

Note that both the temperature equation and the age equation involve advection and have a CFL condition (Morton & Mayers 1994). By being slightly conservative we use the same CFL condition for both (Bueler and others, 2007. "Exact solutions ... thermomechanically-coupled ...," J. Glaciol.). We also report any CFL violations; these can *only* occur when using the `-tempskip` option.

4.3.2.34 PetscErrorCode temperatureStep () [protected, virtual]

Takes a semi-implicit time-step for the temperature equation.

In summary, the conservation of energy equation is

$$\rho c_p \frac{dT}{dt} = k \frac{\partial^2 T}{\partial z^2} + \Sigma,$$

where $T(t, x, y, z)$ is the temperature of the ice. This equation is the shallow approximation of the full three-dimensional conservation of energy. Note dT/dt stands for

the material derivative, so advection is included. Here ρ is the density of ice, c_p is its specific heat, and k is its conductivity. Also Σ is the volume strain heating.

In summary, the numerical method is first-order upwind for advection and centered-differences with semi-implicitness for the vertical conduction term. Note that we work from the bottom of the column upward in building the system to solve (in the semi-implicit time-stepping scheme). Note that the excess energy above pressure melting is converted to melt-water, and that a fraction of this melt water is transported to the base according to the scheme in excessToFromBasalMeltLayer().

Here is a more complete discussion and derivation.

Consider a column of a slowly flowing and heat conducting material as shown in the left side of the next figure. (The left side shows a general column of flowing and heat conduction material showing a small segment V. The right side shows a more specific column of ice flowing and sliding over bedrock.) This is an *Eulerian* view so the material flows through a column which remains fixed (and is notional). The column is vertical. We will assume when needed that it is rectangular in cross-section with cross-sectional area $\Delta x \Delta y$.

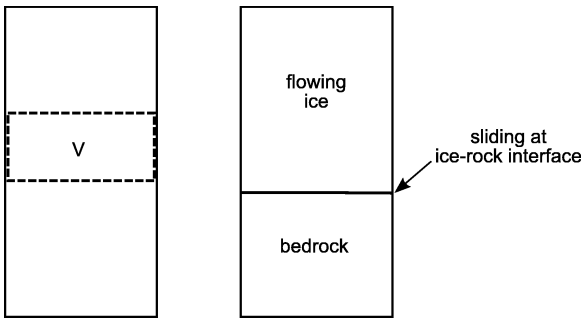

Figure 1: Left: a general column of material. Right: ice over bedrock.

[FIXME: CONTINUE TO MINE eqns3D.tex FOR MORE]

The application of the geothermal flux at the base of a column is a special case for which we give a finite difference argument. This scheme follows the equation (2.114) in Morton and Mayers (2005). We have the boundary condition

$$-k\frac{\partial T}{\partial z} = G(t,x,y)$$

where $G(t,x,y)$ is the applied geothermal flux, and it is applied at level $z = -B_0$ in the bedrock (which is the only case considered here). We *add a virtual lower grid point* $z_{-1} = z_0 - \Delta z$ and we approximate the above boundary condition at z_0 by the centered-difference

$$-k\frac{T_1 - T_{-1}}{2\Delta z} = G.$$

Here $T_k = T_{ijk}^{l+1}$. We also apply the discretized conduction equation at \$z_0\$. These two combined equations yield a simplified form

$$(1 + 2KR)T_0 - 2KT_1 = T_0 + \frac{2\Delta t}{\rho c_p \Delta z} G$$

where $K = k\Delta t(\rho c \Delta z^2)^{-1}$.

4.3.2.35 PetscErrorCode ageStep (PetscScalar * *CFLviol*) [protected]

Take an explicit time-step for the age equation. Also check the CFL for advection.

The age equation is $\frac{d\tau}{dt} = 1$, that is,

$$\frac{\partial \tau}{\partial t} + u\frac{\partial \tau}{\partial x} + v\frac{\partial \tau}{\partial y} + w\frac{\partial \tau}{\partial z} = 1$$

where $\tau(t, x, y, z)$ is the age of the ice and (u, v, w) is the three dimensional velocity field. This equation is hyperbolic (purely advective). The boundary condition is that when the ice fell as snow it had age zero. That is, $\tau(t, x, y, h(t, x, y)) = 0$ in accumulation areas, while there is no boundary condition elsewhere (as the characteristics go outward elsewhere). At this point the refreeze case, either grounded basal ice or marine basal ice, is not handled correctly.

The numerical method is first-order upwind.

4.3.2.36 PetscErrorCode excessToFromBasalMeltLayer (PetscScalar *rho_c*, PetscScalar *z*, PetscScalar * *Texcess*, PetscScalar * *Hmelt*) [protected]

Compute the melt water which should go to the base if T is above pressure-melting.

4.3.2.37 PetscErrorCode vertVelocityFromIncompressibility () [protected]

Compute vertical velocity using basal conditions and incompressibility of the ice.

The original statement of incompressibility is

$$\nabla \cdot \mathbf{U} + \frac{\partial w}{\partial z} = 0.$$

This is immediately equivalent to the integral

$$w(x, y, z, t) = -\int_{b(x,y,t)}^{z} \nabla \cdot \mathbf{U}\, d\zeta + w_b(x, y, t).$$

The basal kinematic equation is

$$w_b = \frac{\partial b}{\partial t} + \mathbf{U}_b \cdot \nabla b - S$$

where S is the basal melt rate. (The inclusion of the basal melt rate is optional.) This equation determines the vertical velocity of the ice at the base (w_b), which is needed in the incompressibility integral.

Note we do a change of vertical coordinate $\tilde{z} = z - b(x, y, t)$. Thus all derivatives, with respect to any variable x, y, z, t, change accordingly (see elsewhere). The revised form of the incompressibility integral is

$$w(x, y, \tilde{z}, t) = -\int_0^{\tilde{z}} \left(\nabla \cdot \mathbf{U} - \nabla b \cdot \left(\frac{\partial u}{\partial \tilde{z}}, \frac{\partial v}{\partial \tilde{z}} \right) \right) d\zeta + w_b(x, y, t).$$

The vertical integral is computed by the trapezoid rule.

4.3.2.38 PetscErrorCode computeMax3DVelocities () [protected]

Compute the maximum velocities for time-stepping and reporting to user.

4.3.2.39 PetscErrorCode updateViewers () [protected]

Update the runtime graphical viewers.

At every time step the graphical viewers are updated. The user specifies these viewers by the options `-d` *list* or `-dbig` *list* where *list* is a list of single character names of the viewers (a list with no spaces). See an appendix of the User's Manual for the names.

Most viewers are updated by this routing, but some other are updated elsewhere:

- see computeMaxDiffusivity() in iMutil.cc for diffusView ("-d D")
- see updateNuViewers() for nuView ("-d i" or "-d j") and lognuView ("-d n") and NuView ("-d N")
- see iceCompModel.cc for compensatory Sigma viewer (and redo of Sigma viewer) "-d PS".

4.3.3 Member Data Documentation

4.3.3.1 const titleNname tn [static, protected]

Holds variable names (for NetCDF and Matlab files) and short titles (for graphical viewers).

This array of strings holds names and titles. These names and titles are for *views* of the internal state of PISM. That is, they are *not* names and title for *parts* of the internal state of PISM. Thus, for example, there is *no* one-to-one correspondence between the internal variables stored in PETSc Vecs and the entries in this array; it is more complicated than that.

This array is used for both variable names *and* short titles in writeMatlabVars(), among other places. It is accessed for short titles *only* in createViewers().

This array is usually referenced by a single character. Note that by the ASCII-to-integer conversion, for instance, the variable name corresponding to the single character name 'A' is `tn[int('A') - int('0')].title`. Such indexing is facilitated by c-Index().

4.4 IceParam Class Reference

Collects the parameters for the grid and computational domain.

4.4.1 Detailed Description

Collects the parameters for the grid and computational domain.

Note that the default choices made when constructing an instance of IceParam will be overridden by PISM runtime options, by the input file (if an input file is used), and frequently by derived classes (of IceModel).

Index