Implementation of the Jacobian-free Newton-Krylov method for solving the first-order ice sheet momentum balance

Jean-François Lemieux*,a, Stephen Priceb, Katherine J. Evansc, Dana Knollb, Andrew G. Salingere, David Hollanda, Tony Payned

*a Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY, 10012-1185, USA
b Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545
c Oak Ridge National Laboratory, 1 Bethel Valley Road P.O. Box 2008, Oak Ridge, Tennessee 37831
d Center for polar observation and modelling, school of geophysical sciences, University of Bristol, Bristol, UK
e Sandia National Laboratories, New Mexico PO Box 5800, Albuquerque, NM 87185

Abstract

We have implemented the Jacobian-free Newton-Krylov (JFNK) method for solving the first-order ice sheet momentum equation in order to improve the numerical performance of the Community Ice Sheet Model (CISM), the land ice component of the Community Earth System Model (CESM). Our JFNK implementation is based on significant re-use of existing code. For example, our physics-based preconditioner uses the original Picard linear solver in CISM. For several test cases spanning a range of geometries and boundary conditions, our JFNK implementation is 1.84-3.62 times more efficient than the standard Picard solver in CISM. Importantly, this computational gain of JFNK over the Picard solver increases when refining the grid. Global convergence of the JFNK solver has been significantly improved by rescaling the equation for the basal boundary condition and through the use of an inexact Newton method. While a diverse set of test cases show that our JFNK implementation is usually robust, for some problems it may fail to converge with increasing resolution (as does the Picard solver). Globalization through parameter continuation did not remedy this problem and future work to improve robustness will explore a combination of Picard and JFNK and the use of homotopy methods.
1. Introduction

During the past decade, there have been major changes on both the Greenland and Antarctic ice sheets as a result of ice dynamics. In Greenland, many outlet glaciers underwent acceleration, thinning and retreat [1, 2, 3, 4, 5, 6] with a consequent increased contribution to global sea level. In Antarctica, land-based glaciers flowing into the Larsen A and Larsen B ice shelves sped up after those ice shelves collapsed [7, 8], and the Pine Island and Thwaites glaciers continued to accelerate and thin [9, 10]. For both Greenland and Antarctica, these dynamical changes are largely attributable to atmospheric and oceanic forcing [6, 11, 12, 13].

For 2000-2008, one half of the total sea level rise from Greenland can be attributed to ice dynamics [14]. In Antarctica, recent analytical and numerical modeling studies confirm the potential for large-scale, dynamical instability, which could increase Antarctica’s contribution to sea level by orders of magnitude [15, 16, 17]. The overall consensus in the scientific literature from the past decade is that Earth’s large ice sheets respond to external climate forcing much faster than was previously thought possible and that ice dynamics plays an important role in controlling the rate of mass loss to the oceans [18, 19, 20].

Because our understanding of ice sheet dynamics and the controlling physical processes is limited, it remains difficult to make plausible estimates for the magnitude of sea level rise associated with future changes in the Greenland and Antarctic ice sheets. In its Fourth Assessment Report (AR4), the Intergovernmental Panel on Climate Change [21] failed to provide a best estimate or even an upper bound for future sea level rise from ice sheets, largely due to this limited understanding. Our current inability to predict the future evolution of ice sheets is demonstrated by the fact that most existing ice sheet models fail to mimic or provide insight into the observed, dramatic changes occurring on ice sheets.

One reason for this failing is the prevalence of models based on the “Shallow Ice Approximation” (SIA) for ice dynamics (e.g. [22]), which assumes
that all of the geometric driving stress is balanced locally through vertical shearing in the ice column [23]. While the SIA greatly simplifies the numerical solution of the momentum and mass conservation equations, the underlying assumptions do not hold for the regions of the ice sheet that control the majority of the mass flux to the oceans (e.g. outlet glaciers, ice streams, and ice shelves). To accurately simulate the flow in these regions, a more complete description of the momentum balance is required, for example that given by solving the Stokes flow [24] or first-order equations [25, 26, 27]. However, the numerical implementation of these "higher order" momentum approximations is much more challenging than for the SIA approximation.

In this work, we focus on improving the numerical performance of ice dynamics represented in the Community Ice Sheet Model, Glimmer-CISM. The original model [22] was based on the SIA, but a first-order momentum balance has recently been added [28]. Unlike the SIA, the first-order formulation also accounts for the effects of longitudinal and lateral stress gradients and, assuming appropriate boundary conditions, is able to represent the full continuum of ice flow observed on ice sheets, from relatively slow inland flow to relatively fast ice stream and ice shelf flow. The model participated in the higher-order model inter-comparison benchmarking study (ISMIP-HOM) and, for tests A-E, all outputs were within one standard deviation of the mean defined by other models of its type (additional information on the results of the benchmarking study can be found in [29]).

The existing solution to the first-order momentum equations in Glimmer-CISM involves (1) a splitting of the momentum equation into its $u$ and $v$ components ($x$ and $y$ directions in map view) [25], (2) solution of the linearized system for $v$ by moving the $u$ terms to the right-hand side and treating them as known source terms (and vice versa), (3) a Picard iteration to handle the nonlinearity associated with the ice rheology (as discussed below), and (4) when appropriate, a correction to the solution from the Picard iteration using the "unstable manifold correction" scheme of [30]. However, as has been shown in previous work [31, 32], a Picard treatment of the nonlinearity generally leads to undesirably slow rates of convergence.

This paper describes the implementation of a more computationally efficient nonlinear solver, the Jacobian-Free Newton-Krylov (JFNK) method, into Glimmer-CISM. The JFNK method has many advantages: the rate of
convergence can be nearly quadratic in the vicinity of the solution, it is scalable (if care is taken with the preconditioning operator), and the Jacobian does not need to be explicitly formed and stored. This last point is especially important; for complicated problems, forming the Jacobian is a difficult development task. While the JFNK method has been applied successfully to other fields in the Earth sciences (e.g., [33, 34, 35]), to our knowledge this is the first time it has been applied to ice sheet modeling.

Our JFNK implementation re-uses much of the code from the existing nonlinear solver. This approach has two clear advantages. First, because it takes advantage of existing code it is easy to implement, and second, the majority of that code has already been extensively tested. Following this, the preconditioning operator is derived directly from the existing linear solver of the Picard scheme, an approach sometimes referred to as physics based preconditioning (e.g., [33]).

The next section (section 2) gives an overview of the first-order momentum equation and the related boundary conditions that govern the ice dynamics in Glimmer-CISM. Section 3 describes the standard Picard solver and the new JFNK solver in detail. In section 4, we describe a number of test cases used to compare the Picard and JFNK solvers. The results of that comparison, the computational efficiency and robustness of Picard versus JFNK, are given in section 5. Finally, in section 6 we summarize, make concluding remarks, and discuss ongoing work to further improve the numerical performance of Glimmer-CISM.

2. Ice sheet momentum equation with a first-order formulation

A complete description of the ice-sheet-evolution problem requires solution of the relevant mass, momentum, and energy conservation equations. Our numerical implementation involves a splitting in time such that ice thickness and temperature are treated explicitly in the momentum equation. Here, we focus only on the efficient solution of the momentum balance equation. Consistent with an incompressible viscous fluid in a low Reynolds number flow, the inertial and advective terms on the left-hand side of the momentum equations are ignored, (similarly, the Coriolis term is neglected).
A brief description of the first-order momentum equations and boundary conditions is given here. For more details, the reader is referred to [28] and [25]. More discussion on the derivation of the first-order equations (starting from the full Stokes equations) and their formal accuracy can be found in [26, 27]. In the discussion below we assume a right-handed Cartesian coordinate system with \(x, y,\) and \(z\) representing the two horizontal and vertical directions, respectively.

Consistent with a first-order scaling of the full Stokes equations [26, 27], the vertical normal stress is balanced by the hydrostatic pressure. This simplification, along with incompressibility and some rearranging reduces the full Stokes equations with four unknowns (the three velocity components \(u, v,\) and \(w\) and the pressure) to two equations for the two horizontal velocity components, \(u\) and \(v,\)

\[
\frac{\partial}{\partial x} \left( 2\sigma'_{xx} + \sigma'_{yy} \right) + \frac{\partial \sigma'_{xy}}{\partial y} + \frac{\partial \sigma'_{xz}}{\partial y} = \rho g \frac{\partial s}{\partial x} \tag{1}
\]

\[
\frac{\partial}{\partial x} \left( 2\sigma'_{yy} + \sigma'_{xx} \right) + \frac{\partial \sigma'_{xy}}{\partial y} + \frac{\partial \sigma'_{yz}}{\partial y} = \rho g \frac{\partial s}{\partial y} \tag{2}
\]

where the \(\sigma'_{ij}\) are the deviatoric stresses, \(\rho\) is the ice density, \(g\) is the gravitational acceleration and \(z = s(x, y)\) defines the upper surface of the ice sheet. The vertical component of velocity, \(w,\) can be recovered from the solutions for \(u\) and \(v\) through incompressibility.

The right-hand side (RHS) terms in equations (1) and (2) represent the volumetric body forces. For the constitutive equation (i.e. the relation between applied stresses and resulting deformations) we use Glen’s flow law [23] in a Newtonian form:

\[
\sigma'_{ij} = 2\eta \epsilon'_{ij} \tag{3}
\]

where the \(\epsilon_{ij}\) are the strain rates (spatial gradients of the velocity components \(u, v\) and \(w\)) and \(\eta\) is the effective viscosity, which is given by

\[
\eta = \frac{1}{2} A^{-1/n_g} (\dot{\epsilon} + \dot{\epsilon}_0)^{(1-n_g)/n_g} \tag{4}
\]
As is common practice, the value of the power-law exponent $n_g$ is set to 3 [23]. The minimum strain rate $\dot{\epsilon}_0$ in equation (4) is a mathematical requirement that prevents an infinite effective viscosity when $\dot{\epsilon}$, the second invariant of the strain rate tensor, tends to zero. The flow law rate factor $A_g$ depends on temperature and (weakly) on ice pressure. Because temperature is taken as a known quantity here (it is explicit), $A_g$ is also assumed known.

By assuming that $\partial w/\partial x \ll \partial u/\partial z$ and $\partial w/\partial y \ll \partial v/\partial z$, which is valid for small aspect ratios [26, 27], and combining equations (1), (2) and (3), the $v$ and $u$ momentum equations can be written as

\begin{align}
4 \frac{\partial \eta}{\partial y} \frac{\partial v}{\partial y} + \frac{\partial \eta}{\partial x} \frac{\partial v}{\partial x} + \frac{\partial \eta}{\partial z} \frac{\partial v}{\partial z} + \eta \left( 4 \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial z^2} \right) &= \rho g \frac{\partial s}{\partial x} - 2 \frac{\partial \eta}{\partial y} \frac{\partial u}{\partial x} - \frac{\partial \eta}{\partial x} \frac{\partial u}{\partial y} - 3 \eta \frac{\partial^2 u}{\partial x \partial y} \\
4 \frac{\partial \eta}{\partial x} \frac{\partial u}{\partial x} + \frac{\partial \eta}{\partial y} \frac{\partial u}{\partial y} + \frac{\partial \eta}{\partial z} \frac{\partial u}{\partial z} + \eta \left( 4 \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} \right) &= \rho g \frac{\partial s}{\partial x} - 2 \frac{\partial \eta}{\partial x} \frac{\partial v}{\partial y} - \frac{\partial \eta}{\partial y} \frac{\partial v}{\partial x} - 3 \eta \frac{\partial^2 v}{\partial x \partial y}
\end{align}

Equations (5) and (6) are written in "split" form, where all terms on the left-hand side (LHS) of the equation for $v$ involve $v$ and all terms on the RHS of the equation for $v$ involve $u$ (and vice versa for the equation for $u$). This is the form used by the standard Picard solver in Glimmer-CISM and the preconditioning operator for our JFNK implementation (the respective solution procedures are discussed further below).

2.1. Boundary conditions

The free surface boundary conditions for the $v$ and $u$ equations are respectively

\begin{align}
\left( 2 \sigma'_{yy}(s) + \sigma'_{xx}(s) \right) \frac{\partial s}{\partial y} + \sigma'_{xy}(s) \frac{\partial s}{\partial x} - \sigma'_{yz}(s) = 0 \\
\left( 2 \sigma'_{xx}(s) + \sigma'_{yy}(s) \right) \frac{\partial s}{\partial x} + \sigma'_{xy}(s) \frac{\partial s}{\partial y} - \sigma'_{xz}(s) = 0
\end{align}
Using the constitutive equation, equations (7) and (8) can be written as

$$\left(4 \frac{\partial v}{\partial y} + 2 \frac{\partial u}{\partial x}\right) \frac{\partial s}{\partial y} + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \frac{\partial s}{\partial x} - \frac{\partial v}{\partial z} = 0$$ (9)

$$\left(4 \frac{\partial u}{\partial x} + 2 \frac{\partial v}{\partial y}\right) \frac{\partial s}{\partial x} + \left(\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}\right) \frac{\partial s}{\partial y} - \frac{\partial u}{\partial z} = 0$$ (10)

Similarly, the basal boundary conditions for the $v$ and $u$ momentum equations are respectively

$$\tau_{by} = \sigma'_{yz}(b) - \left(2\sigma'_{yy}(b) + \sigma'_{xx}(b)\right) \frac{\partial b}{\partial y} - \sigma'_{xy}(b) \frac{\partial b}{\partial x}$$ (11)

$$\tau_{bx} = \sigma'_{xz}(b) - \left(2\sigma'_{xx}(b) + \sigma'_{yy}(b)\right) \frac{\partial b}{\partial x} - \sigma'_{xy}(b) \frac{\partial b}{\partial y}$$ (12)

where $z = b(x,y)$ defines the lower ice (basal) surface. $\tau_{by}$ and $\tau_{bx}$ are the components of the basal traction vector. They are formulated here as

$$\tau_{by} = -\beta^2 v(b)$$ (13)

$$\tau_{bx} = -\beta^2 u(b)$$ (14)

where $\beta^2 = \beta^2(x,y)$ is a sliding parameter. Its value controls the degree of basal sliding at any location; a very large number ($> 10^6$ Pa s m$^{-1}$) allows for a quasi no-slip condition, a smaller number ($< 10^2$ Pa s m$^{-1}$) allows for a moderate amount of basal sliding, and a value of $\sim 0$ allows for free slip at the ice base, as in the case of a freely floating ice shelf (i.e., water drag at the base is negligible).

3. Numerical implementation

3.1. The nonlinear system of equations

A ghost cell approach is used to impose the boundary conditions at the top and at the base. Equations (5) and (6) and the equations related to the boundary conditions are discretized using finite differences and can be written in a compact form as the nonlinear system of $n$ equations at time $t$. 

7
\[ F(v, u) = \begin{bmatrix} A_{vv}(v, u) & A_{vu}(v, u) \\ A_{uv}(v, u) & A_{uu}(v, u) \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} - \begin{bmatrix} s_v \\ s_u \end{bmatrix} = 0 \quad (15) \]

where \( v \) and \( u \) are vectors of size \( n/2 \), \( s_v \) and \( s_u \) include the purely geometric terms that do not depend on \( v \) and \( u \), the matrix \( A_{vv} (A_{uu}) \) is associated with the \( v (u) \) components of the \( v (u) \) equation and the off-diagonal matrix \( A_{vu} (A_{uv}) \) is associated with the \( u (v) \) components of the \( v (u) \) equation.

Notice that the matrices \( A_{vv}, A_{uu}, A_{vu} \) and \( A_{uv} \) are functions of \( v \) and \( u \). Together, these four \( n/2 \) block matrices form the \( n \times n \) matrix \( A \). \( F(v, u) \) is the residual vector.

### 3.2. The Picard solver

The Picard solver is based on a splitting of the \( v \) and \( u \) momentum equations. To describe this splitting, we first write the system of equations (15) as

\[ \begin{bmatrix} A_{vv}(v, u) & 0 \\ 0 & A_{uu}(v, u) \end{bmatrix} \begin{bmatrix} v \\ u \end{bmatrix} = \begin{bmatrix} b_v(v, u) \\ b_u(v, u) \end{bmatrix} \quad (16) \]

where \( b_v(v, u) = s_v - A_{vu}u \) and \( b_u(v, u) = s_u - A_{uv}v \).

This can be written as two similar coupled systems of equations of size \( n/2 \):

\[ A_{vv}(v, u)v = b_v(v, u) \quad (17) \]

and

\[ A_{uu}(v, u)u = b_u(v, u) \quad (18) \]

The Picard solver in Glimmer-CISM is based on an outer loop, the splitting described above and a linear solver. Here is the algorithm of the Picard solver:

1. Start with an initial iterate \( v^0, u^0 \)
   
   do \( k = 1, k_{max} \)
   
   2. Calculate \( \eta(v^{k-1}, u^{k-1}) \)
   
   3. ‘‘Solve’’ \( A_{vv}(v^{k-1}, u^{k-1})v^k = b_v(v^{k-1}, u^{k-1}) \) using a linear solver
4. ‘‘Solve’’, \( A_{uu}(v^{k-1}, u^{k-1})u^k = b_u(v^{k-1}, u^{k-1}) \) using a linear solver.

5. if \( \| F(v^k, u^k) \| < \gamma_{nl} \| F(v^0, u^0) \| \) stop

The initial iterate is the previous time step solution or the zero vector if \( t = 0 \). \( \gamma_{nl} \) defines the tolerance of the nonlinear solver. In the algorithm above, \( \| \| \) is the L2-norm. \( \| F(v^0, u^0) \| \) is the initial nonlinear residual norm.

In Glimmer-CISM, the matrix \( A_{vv}(v^{k-1}, u^{k-1}) \) and the vector \( b_v(v^{k-1}, u^{k-1}) \) are formed in order to solve \( A_{vv}(v^{k-1}, u^{k-1})v^k = b_v(v^{k-1}, u^{k-1}) \) for \( v^k \) with a linear solver. The same process is repeated to obtain \( u^k \). Notice that for step 4, the matrix \( A_{uu} \) and the vector \( b_u \) are formed using \( v^{k-1} \) not \( v^k \). When close to a converged solution, the "unstable manifold correction" scheme of [30] may also be applied after step 4. The standard linear solver, which is part of the SLAP package [36], is the Generalized Minimum RESidual (GMRES, [37]) method preconditioned by an Incomplete LU (ILU) factorization [38].

The criterion for convergence in step 3 is (an analogous criterion is applied to step 4):

\[
\frac{\| P_{ILU}^{-1}(b_v - A_{vv}v^k) \|}{\| P_{ILU}^{-1}b_v \|} < \gamma_l
\]  

(19)

where \( P_{ILU}^{-1} \) is the preconditioning operator for the Picard solver.

3.3. The JFNK solver

Unlike the Picard solver in Glimmer-CISM, the JFNK method does not split the \( v \) and \( u \) equations. However, there is a splitting in the preconditioning step because we use the Picard linear solver for this operation. We introduce the vector \( x = [v \ u]^T \), which is formed by stacking the \( v \) components of velocity followed by the \( u \) components. We then have the same nonlinear system of equations (15) to solve as discussed previously but written as

\[
F(x) = F(v, u) = \begin{bmatrix} F_v(v, u) \\ F_u(v, u) \end{bmatrix} = 0
\]  

(20)

The Newton method is based on a multivariate Taylor expansion around a previous iterate \((x^{k-1})\):

\[
F(x^{k-1} + \delta x^k) \approx F(x^{k-1}) + F'(x^{k-1})\delta x^k
\]  

(21)
where the higher order terms are neglected in the Taylor expansion.

Setting \( F(x^{k-1} + \delta x^k) = 0 \), the correction \( \delta x^k = x^k - x^{k-1} \) can be obtained by solving the linear system of \( n \) equations, here using a Krylov method:

\[
J(x^{k-1})\delta x^k = -F(x^{k-1})
\]

where the system matrix \( J \equiv F' \) is the Jacobian, an \( n \times n \) matrix with elements given by \( J_{ij} = \frac{\partial F_i(x^{k-1})}{\partial x^j_{k-1}} \) (with \( i = 1, n \) and \( j = 1, n \)).

The Newton-Krylov algorithm is:

1. Start with an initial iterate \( x^0 \)
   do \( k = 1, k_{\text{max}} \)
   2. Calculate \( \eta(x^{k-1}) \)
   3. ‘‘Solve’’ \( J(x^{k-1})\delta x^k = -F(x^{k-1}) \) using a Krylov method
   4. \( x^k = x^{k-1} + \delta x^k \)
   5. if \( \| F(x^k) \| < \gamma_{nl} \| F(x^0) \| \) stop
   enddo

As with Picard, the initial iterate is the previous time step solution or the zero vector if \( t = 0 \). In step 3, the convergence criterion for the linear solver is \( \| J(x^{k-1})\delta x^k + F(x^{k-1}) \| < \gamma_{nl}(k) \| F(x^{k-1}) \| \). The parameter \( \gamma_{nl}(k) \), which is a constant smaller than unity, defines the tolerance of the linear solver. Solving \( J(x^{k-1})\delta x^k = -F(x^{k-1}) \) to a very high accuracy (\( \gamma_{nl}(k) \rightarrow 0 \)) might overall increase the total CPU time in finding the nonlinear approximate solution and might even decrease the robustness of the solver. When the approximate solution to the linear system of equations is not “accurate”, the approach is referred to as an inexact Newton method [39]. This is the approach adopted in this work, as discussed further below.

Our implementation of JFNK relies on code re-use from the existing Picard solver within the Glimmer-CISM model. First, the residual vector \( F(x^{k-1}) \) can be obtained as

\[
F(x^{k-1}) = \begin{bmatrix}
A_{vv}(x^{k-1})v^{k-1} - b_v(x^{k-1}) \\
A_{uu}(x^{k-1})u^{k-1} - b_u(x^{k-1})
\end{bmatrix}
\]

To calculate the residual vector \( F \), we just had to code a simple MATVEC
subroutine as we reuse the part of the Glimmer-CISM code that calculates the matrices $A_{vv}$ and $A_{uu}$ and the vectors $b_v$ and $b_u$. As opposed to the Picard solver implementation which requires only one matrix to be stored (because of the splitting), both matrices $A_{vv}(x^{k-1})$ and $A_{uu}(x^{k-1})$ are stored for JFNK because they are used for the preconditioning step.

Obtaining the Jacobian matrix is a difficult development task for complicated problems such as the one considered here. Moreover, forming this system matrix could be computationally expensive. For these reasons, we adopt a Jacobian-free approach, which is possible when using a Krylov method as the linear solver. Indeed, Krylov methods approximate the linear solution in a subspace of the form $(r_0, Jr_0, J^2r_0...)$ where $r_0$ is the initial linear residual given by $J(x^{k-1})\delta x_0^k + F(x^{k-1})$ [39], with an initial guess $\delta x_0^k$ usually taken to be zero. Because Krylov methods require only the product of the system matrix (the Jacobian) and a vector, the Jacobian matrix does not need to be formed and stored explicitly. Rather, only its action on a vector is required. This property is fundamental for the implementation of a Jacobian-free approach, which relies on the fact that the product of $J$ times a vector $w$ can be approximated by a first-order Taylor series expansion

$$J(x^{k-1})w \sim \frac{F(x^{k-1} + \epsilon w) - F(x^{k-1})}{\epsilon}$$

where $\epsilon$ is a small number ($10^{-7}$ in our implementation).

The approximation of $J$ times a vector can be calculated using the machinery previously described in equation (23).

Krylov methods for solving linear systems of stiff equations are likely to converge very slowly and to exhibit robustness issues unless preconditioning is applied [40]. By using preconditioning, one solves an equivalent system of equations that has the same solution as the original system but which is numerically easier to solve. In this work we use the Flexible GMRES (FGMRES) approach which relies on right preconditioning [40]. In this case, equation (22) becomes

$$J(x^{k-1})P_p^{-1}\delta z = -F(x^{k-1})$$

where $\delta z = P_p\delta x^k$ and $P_p^{-1}$ is referred to as the preconditioning operator.
The subscript $p$ indicates that the Picard solver is used for the JFNK preconditioning operator (as discussed further below).

Note that $P_p^{-1}$ is an operator that should not be necessarily considered as the inverse of a matrix. We are looking for an operator that approximates the inverse of the system matrix ($J^{-1}$). As before, we are looking for a Jacobian-free approach.

The matrix $J$ can be divided into two matrices [35]

$$J(x^{k-1}) = A(x^{k-1}) + G(x^{k-1})$$  \hspace{1cm} (26)$$

where $A(x^{k-1})$ is the matrix described in equation (15, linearized with $x^{k-1}$) and $G(x^{k-1})$ is a matrix with entries made up of sums of terms like $x_j^{k-1} \partial a_{ij}(x^{k-1})/\partial(x_j^{k-1})$ (the $a_{ij}$ being the elements of the matrix $A$). The matrix $G$ reflects the nonlinear nature of the ice sheet rheology, i.e., the fact that the viscosity depends on the velocity field as opposed to a Newtonian fluid. Using the block matrices introduced in equation (15), we can write $J(x^{k-1})$ as

$$J(x^{k-1}) = \begin{bmatrix} A_{vv}(x^{k-1}) & 0 \\ 0 & A_{uu}(x^{k-1}) \end{bmatrix} + \begin{bmatrix} 0 & A_{vu}(x^{k-1}) \\ A_{uv}(x^{k-1}) & 0 \end{bmatrix} + G(x^{k-1})$$  \hspace{1cm} (27)$$

While the off-diagonal matrices $A_{vu}(x^{k-1})$ and $A_{uv}(x^{k-1})$ are never formed within the existing Picard solver and getting $G(x^{k-1})$ is very complicated (the reason why the system matrix $J(x^{k-1})$ is so difficult to obtain), we do have access to the matrices $A_{vv}(x^{k-1})$ and $A_{uu}(x^{k-1})$. It is therefore possible to use these two matrices for the preconditioning operator. Given a vector $q$, the preconditioning operator applied to it leads to the vector $r = P_p^{-1}q$ where $q$ and $r$ are vectors formed during the Krylov iteration process. In more details, based on the splitting approach of the Picard solver and the matrices $A_{vv}(x^{k-1})$ and $A_{uu}(x^{k-1})$, the preconditioning operator leads to

$$r_v = \tilde{A}_{vv}^{-1}q_v$$  \hspace{1cm} (28)$$

$$r_u = \tilde{A}_{uu}^{-1}q_u$$  \hspace{1cm} (29)$$
where \( \mathbf{q} = [\mathbf{q}_v \ \mathbf{q}_u]^T \) and \( \mathbf{r} = [\mathbf{r}_v \ \mathbf{r}_u]^T \), and the tilde indicates that we solve equations (28) and (29) to a very loose tolerance \( (\gamma_l = 10^{-3}, \text{see equation (19)} \) for details) in other words, \( \mathbf{r}_v \) is the approximate solution of \( \mathbf{A}_{vv} \mathbf{t}_v = \mathbf{q}_v \)

where \( \mathbf{t}_v \) would be the exact solution.

3.4. Robustness of the solvers

For the Picard solver, \( \gamma_l \) in equation (19) is set to \( 10^{-12} \). This is a very small value of \( \gamma_l \). Tests have shown that this high tolerance increases the robustness of the Picard solver [28]. Note that the same criterion is used for the \( u \) equation in step 4.

We have included two small modifications to our JFNK solver to improve its robustness. As was observed by [41] and [35], a small value of the forcing term \( \gamma_l \) (see equation (30)) in early Newton iterations increases the CPU time and can even prevent convergence of the approximate solution. We have substantially improved the robustness of our JFNK solver by using an inexact Newton method. Following [42], the value of the parameter \( \gamma_l(k) \) in our inexact Newton approach depends on previous values of the L2-norm. It is given by

\[
\gamma_l(k) = \alpha \left( \frac{\|F(x^{k-1})\|}{\|F(x^{k-2})\|} \right)^m
\]

where the parameters \( \alpha \) and \( m \) are 1 and 2, respectively. We have not done a thorough investigation to optimize these parameters. For the first Newton iteration, \( \gamma_l(k) = 0.9 \) and it is limited to \( 0.01 \leq \gamma_l(k) \leq 0.9 \) for subsequent iterations. As discussed later, this progressive tolerance (equation (30)) might only be needed for a few time steps at the beginning of the run when the initial iterate is far from the solution. Later in the simulation, a more aggressive tolerance (e.g., \( \gamma_l(k) = 0.01 \)) could be used to improve the computational efficiency without affecting the robustness.

The second modification is related to the basal boundary condition. Strong friction at the base \( (\beta^2 \gg 1) \) leads to large off-diagonal elements in the \( \mathbf{A}_{vv} \) and \( \mathbf{A}_{uu} \) matrices. A rescaling of the elements in \( \mathbf{A}_{vv} \) and \( \mathbf{A}_{uu} \) associated with the basal boundary conditions has proven to significantly improve the robustness of the JFNK solver. Note that the same scaling is used here for
<table>
<thead>
<tr>
<th>test case</th>
<th>domain</th>
<th>problem size</th>
<th>( \beta^2 ) (Pa s m(^{-1}))</th>
</tr>
</thead>
<tbody>
<tr>
<td>dome-2km</td>
<td>30x30x10</td>
<td>6640</td>
<td>( 10^{10} )</td>
</tr>
<tr>
<td>dome-1km</td>
<td>60x60x10</td>
<td>27760</td>
<td>( 10^{10} )</td>
</tr>
<tr>
<td>dome-0.5km</td>
<td>120x120x10</td>
<td>111520</td>
<td>( 10^{10} )</td>
</tr>
<tr>
<td>dome-0.25km</td>
<td>240x240x10</td>
<td>448960</td>
<td>( 10^{10} )</td>
</tr>
<tr>
<td>ISMIP-HOM-C</td>
<td>91x91x11</td>
<td>182116</td>
<td>variable and periodic</td>
</tr>
<tr>
<td>shelf</td>
<td>51x51x11</td>
<td>48576</td>
<td>( 10^{-5} )</td>
</tr>
<tr>
<td>GIS-20km</td>
<td>76x141x11</td>
<td>94644</td>
<td>( 10^{10} )</td>
</tr>
<tr>
<td>GIS-10km</td>
<td>151x281x11</td>
<td>371602</td>
<td>( 10^{10} )</td>
</tr>
<tr>
<td>GIS-5km</td>
<td>301x561x11</td>
<td>1483196</td>
<td>( 10^{10} )</td>
</tr>
</tbody>
</table>

Table 1: The different test cases

the Picard solver but that it does not affect its robustness and performances.

4. The test cases

We have used a suite of test cases to assess the robustness and computational efficiency of the JFNK method and to compare these to the Picard solver. A wide range of problem sizes, configurations, and boundary conditions are covered. Table 1 gives relevant details about the different test cases.

The name of each test case is given in the first column of Table 1. The horizontal (East-West and North-South) dimensions and the number of vertical levels are given in the second column and the size of the problem is given in the third column. The size of the problem is defined as the number of grid cells where ice is present (at the beginning of the simulation) times two (for the two components of velocity). The last column gives some information about the sliding coefficient \( \beta^2 \).

The first four test cases correspond to the same problem: a parabolic dome of ice with a circular, 60 km diameter base. The horizontal spatial resolutions studied are 2 km, 1 km, 0.5 km and 0.25 km, and there are 10 vertical levels. For this set of experiments, a quasi no-slip basal condition is imposed by setting \( \beta^2 = 10^{10} \) Pa s m\(^{-1}\). A zero-flux boundary condition is applied at the dome margins. We refer to this set of experiments as the
Table 2: Physical constants used in the simulations

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ρ</td>
<td>ice density</td>
<td>910 kg m$^{-3}$</td>
</tr>
<tr>
<td>g</td>
<td>gravitational acceleration</td>
<td>9.81 m s$^{-2}$</td>
</tr>
<tr>
<td>$n_g$</td>
<td>exponent in Glen’s law</td>
<td>3</td>
</tr>
<tr>
<td>$A_g$</td>
<td>flow rate parameter</td>
<td>$10^{-16}$ Pa$^{-n_g}$yr$^{-1}$</td>
</tr>
</tbody>
</table>

“dome” test cases. The next test case (“shelf”) involves a flat ice shelf of uniform thickness at flotation that is confined on three sides by solid walls but open to the ocean on the fourth side. Along the confined walls a zero-flux boundary condition is applied and along the open front the stress is balanced by the hydrostatic pressure due to a column of ocean water. The drag on the shelf bottom is negligible as $\beta^2$ is set to $10^{-5}$. In the ISMIP-HOM-C test case, the $\beta^2$ value is doubly periodic as described in [29]. The last set of test cases represent the Greenland ice sheet (GIS) at different spatial resolutions (20 km, 10 km and 5 km), based on the 5 km digital elevation model of [43]. A quasi-no slip boundary condition is applied at the bed. As with the dome test cases, a zero-flux boundary condition is applied at the lateral margins. We refer to these respectively as GIS-20km, GIS-10km and GIS-5km. In all test cases, the ice is taken as isothermal with a constant and uniform rate factor of $10^{-16}$ Pa$^{-n_g}$yr$^{-1}$.

Table 2 gives the value of the physical constants used by the model for the simulations presented here.

We treat the evolution of the ice thickness using the "incremental remapping" scheme of [44], a higher-order advection scheme that has been applied successfully to the thickness evolution of sea ice [45].

5. Results

All simulations were performed on a desktop computer (Intel(R) dual-core(TM) E8400 3.00 GHz CPU, cache of 6144 KB with a RAM of 4 Gb). The fortran compiler is gfortran 4.1.2, 64 bits and the O3 optimization is used. We use revision 2002 of a Glimmer-CISM developmental branch (called the parallel branch) that will be released to the public in the near future.
Table 3: Number of outer iterations and CPU time required to calculate a diagnostic velocity field when using either the Picard solver or the JFNK solver and computational gain of JFNK over Picard depending on the test case.

<table>
<thead>
<tr>
<th>test case</th>
<th>ite JFNK</th>
<th>ite Picard</th>
<th>CPU JFNK</th>
<th>CPU Picard</th>
<th>gain</th>
</tr>
</thead>
<tbody>
<tr>
<td>dome-2km</td>
<td>11</td>
<td>51</td>
<td>1.39</td>
<td>3.13</td>
<td>2.25</td>
</tr>
<tr>
<td>ISHOM-HOM-C</td>
<td>15</td>
<td>58</td>
<td>170.97</td>
<td>522.81</td>
<td>3.06</td>
</tr>
<tr>
<td>shelf</td>
<td>8</td>
<td>60</td>
<td>37.09</td>
<td>101.42</td>
<td>2.73</td>
</tr>
<tr>
<td>GIS-20km</td>
<td>10</td>
<td>51</td>
<td>17.89</td>
<td>45.01</td>
<td>2.51</td>
</tr>
<tr>
<td>GIS-10km</td>
<td>15</td>
<td>55</td>
<td>136.36</td>
<td>251.42</td>
<td>1.84</td>
</tr>
</tbody>
</table>

5.1. Robustness

In the first series of experiments, we assess the robustness of the JFNK and Picard solvers in calculating a diagnostic velocity field. The initial iterate is the zero vector, which provides a good test to evaluate the global convergence\(^1\) properties (and therefore robustness) of a nonlinear solver as this initial iterate is far from the solution. Changing the size of the time step for a diagnostic solution does not affect the robustness results presented here because the inertial term is neglected in the momentum equations (i.e. a steady-state solution given a temperature and thickness fields). For all the following robustness experiments, the nonlinear tolerance \((\gamma_{nl})\) is set to \(10^{-8}\).

Figure 1 shows the evolution of the L2-norm for the Picard solver (dashed curves) and the JFNK solver (solid curves) with the number of outer iterations. Table 3 summarizes the results for the number of outer iterations and CPU time required to get the approximate solution. The current focus is robustness so we assess if the solvers are able (or not) to calculate the required approximate solution. We also give the number of iterations and CPU time required for the simulations, but given that the first step is a very particular case, the iteration count and computational gain of JFNK over Picard might not be representative.

\(^1\)an iterative method is said to be globally convergent if the approximate solution converges to the true solution given some arbitrary initial iterate.
Figure 1: L2-norm as a function of the number of outer loop iterations for the Picard solver (dashed curves) and the JFNK solver (solid curves). The test cases are: a) dome-2km, b) ISHOM-HOM-C c) shelf d) GIS-20 km, e) GIS-10 km and f) GIS-5 km.

Both solvers converge to the solution for the dome-2km, ISHOM-HOM-C, shelf, GIS-20km and GIS-10km test cases. However, Picard and JFNK fail to converge at 5 km resolution for the Greenland ice sheet test case. For both solvers, the residual initially decreases by almost three orders of magnitude but then levels off. A plot of the residual on the grid shows that this convergence issue only affects a few grid cells (not shown). We discuss several potential strategies for addressing this problem in the conclusions.

The JFNK solver requires between 3.7 to 7.5 times fewer outer iterations than the Picard solver to get the solution. Nevertheless, because a Newton
iteration involves more computations, the computational gain is not as high
as the iteration count ratio. Here, we find that the JFNK solver is between
1.84-3.06 times faster than the Picard solver depending on the particular test

case.

5.2. Computational efficiency

Because the initial iterate is far from the solution, both solvers have a
harder time solving the nonlinear system of equations at the beginning of
the runs (for a few time steps). After this transition phase, the behavior of
each solver is fairly constant (i.e., the CPU time and number of outer iter-
ations required per time step do not vary significantly). The main goal in
developing a JFNK solver is to significantly reduce the computational time
for long-term simulations. In this section, we assess the computational gain
of JFNK over the Picard solver in the context of long-term simulations by
looking at the performances of both solvers after the initial transition phase.
We are also interested in understanding how this computational gain evolves
as the problem size increases.

To this end, we perform two sets of experiments. In the first one, we
compare the computational efficiency of JFNK to the one of Picard for the
dome test cases with increasing horizontal spatial resolution. In the second
set of experiments, we investigate the performance of JFNK and Picard in
simulating the Greenland ice sheet (under the conditions given above) at 20
km and 10 km resolutions. For all these experiments, \( \gamma_{nl} = 10^{-6} \) and the
time step is 1 year. The simulations are run for 20 years. The mean CPU
time per time step required by Picard is compared to the mean CPU time
needed by JFNK. The mean values are calculated based on time steps 11 to
20 (i.e. sufficiently far from the initial iterate).

As an aside, Figure 2 shows a typical velocity field solution. It is the sur-
face ice speed calculated with the JFNK solver for the GIS-10km test case
at \( t = 10 \) years. In accordance with observations, the fastest velocities are
found close to ice margins.

Figure 3 shows the relative computational gain of JFNK versus Picard
as a function of the problem size for the dome test cases (solid curve with *
data points) and the Greenland ice sheet experiments (the + data points).
The computational gain is calculated as the mean CPU time per time step required by Picard divided by the mean CPU time per time step for JFNK. The JFNK solver is clearly more efficient than Picard for all the spatial resolutions tested and for both sets of experiments. Furthermore, the computational gain increases as the grid is refined. For the dome experiments, JFNK is 2.6 times faster than Picard for a 2 km resolution. This gain increases to 3.62 at 0.25 km resolution. Although the computational gains are lower, JFNK is still significantly faster than Picard for the Greenland experiments: the gain is 2.14 at 20 km and 2.44 at 10 km resolution. The computational gain of JFNK over the Picard solver also increases when a more accurate solution (a smaller $\gamma_{nl}$) is required (results not shown). Note that these numbers (the computational gain) are slightly biased in favor of
the Picard solver because the final L2-norm of the JFNK solver is often significantly lower than that from the Picard solver; from one iteration to the next, the L2-norm drops by a larger amount for a single Newton iteration than for a single Picard iteration (see Figure 5 for an example of this).

Figure 3: Mean CPU time needed by Picard divided by the mean CPU time for JFNK as a function of the problem size for the dome test cases (•) and the Greenland ice sheet simulations (+).

Figure 4 gives additional information on the behavior of JFNK as the grid is refined. First, Figure 4a shows on a log-log plot how the CPU time evolves as the resolution is increased. The relationship between the CPU time and the problem size is linear (on a log-log plot) but the slope is slightly larger than 1 (for both sets of experiments). The “extra work” is performed by the inner GMRES (Figure 4c) as the number of FGMRES iterations is fairly constant when changing the spatial resolution (Figure 4b). At 2 km resolution with the “dome” test case, the percentage of CPU time associated with the preconditioning operator is slightly larger than 25% while this percentage increases to 45% with a 0.25 km grid. The use of a more efficient precondi-
tioner such as an algebraic multigrid will likely result in better scaling (slope close to 1).

Figure 4: a) Average CPU time per time step required by JFNK as a function of the problem size. b) Average number of FGMRES iterations per time step for the JFNK solver as a function of the problem size. c) Average number of GMRES (the preconditioner) iterations per time step for the JFNK solver as a function of the problem size. The data points for the “dome” test cases are the * while the + correspond to the Greenland ice sheet simulations.

Figure 5 shows a typical evolution of the L2-norm of the nonlinear system of equations when using either JFNK or the Picard solver for the GIS-10km test case at \( t = 10 \) years. The L2-norm as a function of the number of outer iterations for the Picard solver is shown with the dashed curve. The solid curve with the * data points shows the L2-norm with the number of Newton iterations for JFNK when using the progressive linear tolerance given by equation (30) (the default approach). Because this progressive linear tolerance is only needed for the first few time steps to increase robustness, more aggressive tolerance values can be used for later times. We have done this here by setting \( \gamma_l(k) = 0.01 \) at \( t = 10 \) years, for the solid curve with the + data points on Figure 5. With this tighter tolerance, JFNK only needs 3 Newton iterations to reach the convergence criterion while 5 Newton iterations are required with the progressive tolerance. In terms of computational efficiency, JFNK is 2.45 faster than Picard with the progressive tolerance and 2.95 faster.
with a fixed $\gamma_l(k)$ of 0.01. As expected from theory, the convergence rate of Picard is linear. For JFNK, the convergence rate is not quadratic because an inexact Newton approach is used. Asymptotic quadratic convergence could be achieved by using very small values of $\gamma_l(k)$ [39].

![Figure 5: L2-norm as a function of the number of outer iterations for the Picard solver, JFNK with $\gamma_l(k)$ given by equation (30) and JFNK with $\gamma_l(k) = 0.01$. This is for the GIS-10km test case at $t = 10$ years.](image)

The use of the Picard solver (GMRES+ILU) as a preconditioner has lead to a quick implementation of the JFNK solver. One might argue however that a more computationally efficient approach would be to simply use ILU for the preconditioning step and therefore put aside the inner GMRES. Results indicate that at low resolution, JFNK with ILU as the preconditioner (referred here as JFNK-ILU) is as efficient as our standard JFNK (with GMRES+ILU for the preconditioning step). However, as the resolution is increased, JFNK-ILU is less and less efficient (results not shown). This is caused by a significant increase in the number of FGMRES iterations when

...
compared to our standard JFNK approach. This is detrimental on the CPU
time as function evaluations in the approximation of the Jacobian times a
vector (equation (24)) are costly. As it was observed by others, ILU does
not perform well when refining the grid as it leads to a lot more of Krylov
iterations (e.g., [46]).

6. Conclusion

We have implemented a Jacobian-Free Newton-Krylov (JFNK) method
to solve the first-order ice sheet momentum equations. This new solver has
been implemented into the framework of the Community Ice Sheet Model
(Glimmer-CISM), the land ice component of the Community Earth System
Model (CESM). It was developed by re-using many parts of the existing
Picard solver in Glimmer-CISM. Specifically, we have developed a physics
based preconditioner with the existing Glimmer-CISM Picard solver. The
re-use of previously tested code has led to a quick implementation.

Two minor modifications considerably improved the robustness of our
JFNK implementation. First, we apply a loose linear tolerance during the
early stage of the Newton process. As observed by others (e.g., [41], [35]),
the use of a tight tolerance in the early Newton iterations can be detrimental
in terms of CPU time and can even prevent the method from converging. A
loose tolerance of $\gamma_1(k) = 0.9$ is applied for the first Newton iteration and
subsequent values are calculated according to the evolution of the L2-norm
(based on [42]). Second, the imposition of the boundary condition at the
base can lead to very large matrix coefficients. To improve the robustness,
a rescaling of the rows of the matrix associated with this basal boundary
condition is employed.

With these modifications, JFNK converges for a range of different test
cases even when starting with an initial iterate that is far from the solution.
However, our tests also indicate that JFNK sometimes does not converge
for high resolution simulations. We note that the existing Picard solver also
exhibits a lack of robustness for these same fine grid test cases. Additional
work needs to be done to eliminate this problem. A parameter continuation
method [39] has not been able to resolve this issue. This was done by slowly
adjusting the value of the exponent in Glen’s law (equation (3)) during the
Newton iteration. We are currently investigating the use of a combination
Picard (in the early stage) and Newton (at the end of the process) solver to
improve the overall solver robustness [32]. We are also investigating homo-
topy and pseudo-transient continuation [39] methods as possible globalization
approaches.

The computational efficiency of JFNK (and its comparison to the one of
Picard) was studied for two different sets of experiments with variable hori-
zontal spatial resolution. These tests indicate that JFNK is between 2.14 to
3.62 times faster than the Picard solver. The computational gain of JFNK
over Picard increases as the grid is refined. Moreover, the computational
gain increases when a tighter nonlinear tolerance is required. In other words,
the JFNK solver is increasingly more efficient compared to Picard when a
more accurate solution is desired. The JFNK method is therefore a signif-
icant improvement in terms of computational efficiency when compared to
the standard solver of Glimmer-CISM (the Picard solver).

Because the percentage of CPU time associated with the preconditioning
step increases with the spatial resolution of the problem, a more efficient pre-
conditioner, such as algebraic multigrid, could further improve the results.
We are currently implementing the Trilinos solver package [47], after which
we will replace our “homemade” JFNK solver with the Trilinos NOX solver,
using multi-level (ML) for the preconditioning step. We are also developing
a multi-pass Picard preconditioner (which involves the use of the off-diagonal
block matrices) to reduce the CPU time associated with the preconditioning
operator. Current work also includes a complete parallelization of the model.

Acknowledgments

We would like to thank Carl Gladish for his help in setting up and compil-
ing Glimmer-CISM and William H. Lipscomb for helpful discussions. Jean-
François Lemieux is grateful to FQRNT and NSERC for Postdoctoral fel-
lowships. This work has been funded by the Department of Energy Office
of Advanced Scientific Computing project, A Scalable, Efficient, and Accu-
rate Community Ice Sheet Model, within the ISICLES initiative. Sandia is a
multiprogram laboratory operated by Sandia Corporation, a Lockheed Mar-
tin Company, for the United States Department of Energy under contract
References


