

A Dimensional Reduction Algorithm and Software for Acyclically Dependent Constraints

P. Areias, A. Vidinha-Alves, M. Pereira dos Santos & J. Carrilho Lopes

To cite this article: P. Areias, A. Vidinha-Alves, M. Pereira dos Santos & J. Carrilho Lopes (2019): A Dimensional Reduction Algorithm and Software for Acyclically Dependent Constraints, International Journal for Computational Methods in Engineering Science and Mechanics

To link to this article: <https://doi.org/10.1080/15502287.2019.1566284>



Published online: 01 Feb 2019.



Submit your article to this journal [↗](#)



View Crossmark data [↗](#)

A Dimensional Reduction Algorithm and Software for Acyclically Dependent Constraints

P. Areias^{a,b}, A. Vidinha-Alves^a, M. Pereira dos Santos^a, and J. Carrilho Lopes^c

^aDepartment of Physics, University of Évora, Colégio Luís António Verney, Évora, Portugal; ^bCERIS/Instituto Superior Técnico, University of Lisbon, Lisbon, Portugal; ^cDepartment of Geosciences, University of Évora, Colégio Luís António Verney, Évora, Portugal

ABSTRACT

For discrete equations of motion with acyclic equality constraints and within the context of the null-space method, an original Algorithm is introduced. By first permuting and then topologically ordering the degrees-of-freedom in the constraint gradient matrix, the saddle point problem can be solved with a sparse triangular system for the constraint equations. In this work, we show that saddle problems resulting from constrained (nonlinear) mechanical problems can always be set in this form, with constraint pivots being selected a priori. Given n discrete motion equations and m equality constraints, the original square sparse $(n + m)^2$ system is replaced by a sparse system $(n - m)^2$ and a sparse triangular solve with m^2 coefficients and $n - m$ right-hand sides. This triangular solve, which involves three sparse matrices (in existing literature only two of the three matrices are sparse), is here discussed in detail. Seven sparse operations are addressed (five standard and two nonstandard) in addition to some specific ad-hoc operations. Algorithms, source code and examples are presented in this work.

KEYWORDS

Coupling of multibody dynamics and finite elements; Equality constraints; Nonlinear problems; Sparse matrices

1. Introduction



According to the form of contribution to the force vector and Jacobian matrix (in quasi-statics, the consistent *stiffness* matrix), discretizations of continuum engineering problems generate constituents belonging to two classes: additive (finite and meshless elements including loading, contact elements, and other smooth and nonsmooth force elements) and multiplicative (certain equality constraints, master-slave relations, rigid parts, and arc-length constraints). This classification leaves room for some overlapping, and a rational choice can be made on grounds of efficiency. It is also worth noting that general nonlinear equality constraints contribute both additively and multiplicatively to the force vector and Jacobian matrix.

Combinations of finite elements and rigid-body regions (rigidity is enforced by master-slave relations) are highly relevant in current multiphysics simulations. For example, when a full thermo-mechanical analysis is performed and it is not time-efficient to consider the deformation of certain parts of the domain, but a full heat conduction simulation is

required, these parts can be made rigid by using master-slave relations.

Specific formulations of many of such constituents are provided in the book by Belytschko et al. [1] and in many papers, see e.g. [2]. Details concerning the solution of problems resulting from systematic creation and combination of new constituents (made possible with tools such as Mathematica [3] with the AceGen add-on [4]), has not been shown with Algorithmic depth in the literature. A systematization of the technical implementation of models of mechanics, in the sense of Klarbring [5],¹ after discretization, is the aim of this work. This perspective is shared by the governing equations, constraints and solution methods.

Although a comprehensive solution is introduced in this work, contributions by other groups deserve mentioning. A preliminary work was shown in the papers by Abel and Shephard [6] and [7], but it did not include dependence between constraints. Ainsworth [8] generalized this approach and showed the demand for constraint ordering. In Chow et al.

CONTACT P. Areias  pmaa@uevora.pt  Department of Physics, University of Évora, Colégio Luís António Verney, Rua Romão Ramalho, 59, 7002-554 Évora, Portugal.

¹Chapter 12 shows continuum applications of constraints, some as “*constitutive assumptions*”

Color versions of one or more of the figures in the article can be found online at www.tandfonline.com/ucme.

[9], the Authors extended previous works, but without topologically ordering the constraints. In [10], a linear iterative solver is adopted with the projection method (see, e.g. [11]). Recently in [2] and [12], the Authors have performed the operations at the clique level. Notwithstanding, for more general constraints, a complete sparse formulation was found to be preferable, and this is the approach of the present work. Our present Algorithm internally reorders the constraints.

By using direct methods for indefinite sparse matrices, the full saddle-point problem could in theory be solved monolithically. However, for large number of constraints, this can be uneconomical if a direct sparse solver is adopted (see, e.g. [13]). Essential boundary conditions are a good example of the effectiveness of multiplicative components used in many commercial and academic codes.² The same applies to rod and shell parametrization: director inextensibility is imposed with multiplicative constituents (at the *continuum* level by coordinate transformation, see, e.g. Antman [14] and [15]).

We here are concerned in imposing nodal trajectories, rigid body constraints and more complex interactions such as frictional contact. Generality is limited by the resulting DOF graph, as we shall see, but also the well-posedness of the resulting discrete system (dependent on the values of the coefficients).

Contact and friction constituents, which introduce complementarity conditions are adequately treated with additive elements since they are often part of a active-set Algorithm. Other behavior, such as rigid motion, kinematic links, periodicity boundary conditions (see, e.g. [16] for such an application) are best treated with the methods herein described. In the context of multibody dynamics, these methods are also known as *coordinate reduction methods* [17] and [18]. These techniques have been increasingly relevant in recent years for unit cell analysis in multiscale methodologies. Another obvious application is static condensation, very convenient for mixed and hybrid FE element technology and also nodeless degrees of freedom. The presence of “condensable” degree-of-freedom should be detected by the solver prior to decomposition and the subsequent postprocessing of slave degrees-of-freedom must be effected without user intervention. Static condensation of the *nodeless* degrees of freedom is simple to program and can provide substantial savings [19]. In this context, efficient methods are available for iterative sparse linear solvers [20] since the transformation matrix (\mathbf{Z} in our

notation) can premultiply the iterative solution, allowing considerable savings. From an applied Mathematics perspective, a in-depth review is provided by Benzi et al. [11].

Our aim here is to provide a complete solution (algorithm and software) to prototype problems. We assess the software with a rigid-body constraint in 3 D within a visco-plasticity medium of magma with olivine.

2. Constrained dynamic systems

We consider $t \in \mathbb{R}_0^+$ as the time when inertia is present or pseudo-time in quasi-statics and $\mathbf{q} \in \mathbb{R}^n$ as the unconstrained degree-of-freedom vector. In addition, $\dot{\mathbf{q}} = \text{d}\mathbf{q}/\text{d}t$ and $\ddot{\mathbf{q}} = \text{d}^2\mathbf{q}/\text{d}t^2$. A n -dimensional dynamic system established by the discrete equations of motion:

$$\mathbf{r}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) = 0 \quad (1)$$

subject to a set of m equality constraints³

$$\mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, t) = 0 \quad (2)$$

The combination of (1) and (2) in discrete form is:

$$\begin{aligned} \delta \mathbf{q} \cdot \mathbf{r}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) &= 0 \\ \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, t) &= 0 \end{aligned} \quad (3)$$

with $\delta \mathbf{q}$ belonging to the null space of the constraint gradients

$$\underbrace{\nabla \mathbf{g}}_{\partial \mathbf{g}} \cdot \delta \mathbf{q} = 0 \quad (4)$$

where ∇ is the gradient with respect to \mathbf{q} including the dependence on $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$. Enforced property (4) can be interpreted as a filter of certain equations in \mathbf{r} which are then replaced by equations in \mathbf{g} . An example appears in the linear finite element literature where essential boundary conditions are often applied by removing rows and columns of the stiffness matrix.

A relation with classical notation for the mass matrix \mathbf{M} can be established from (1): $\mathbf{M} = \partial^2 \mathbf{r} / \partial \ddot{\mathbf{q}}$. We now make use of a *general* form of integration for a given time step k such that velocity and acceleration variations are related to displacement variations: $\delta \dot{\mathbf{q}} \equiv c \dot{\mathbf{q}} \delta \mathbf{q}$ and $\delta \ddot{\mathbf{q}} \equiv c \ddot{\mathbf{q}} \delta \mathbf{q}$. For completeness (and the starting procedure, as will become apparent) we also introduce $\delta \mathbf{q} = c_q \delta \mathbf{q}$. Using a set of Lagrange multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$, we can combine (1) and (2) as:

$$\delta \mathbf{q} \cdot \mathbf{r}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}, t) - \delta \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, t) \cdot \boldsymbol{\lambda} = 0 \quad (5)$$

²usually, the affected coefficients are implicitly multiplied by zero, which is equivalent to the removal of the equations as will become apparent

³discrete forms of essential boundary conditions, rigid body constraints, etc.

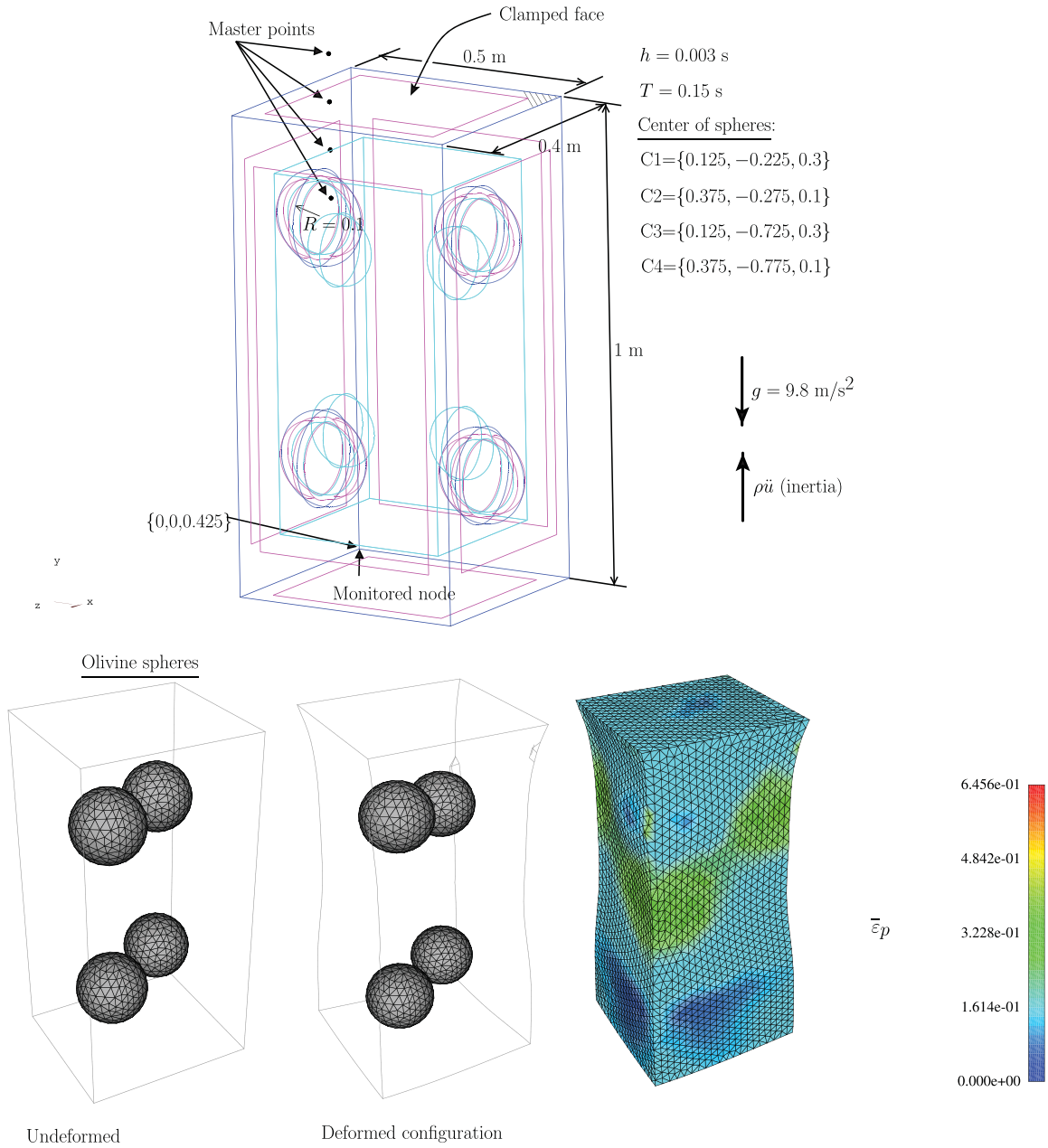


Figure 1. Magma/olivine geometry and boundary conditions. Also shown are the olivine spheres in undeformed and deformed configurations and the effective plastic strain contour plot over the deformed magma geometry.

$$\delta \lambda \cdot \mathbf{g}(\mathbf{q}, \dot{\mathbf{q}}, t) = 0 \quad (6)$$

Omitting the arguments of \mathbf{r} and \mathbf{g} , Newton iteration on (5–6) provides:

$$\begin{bmatrix} \underbrace{\nabla \mathbf{r} - \lambda \cdot \nabla^2 \mathbf{g}}_K & -\nabla \mathbf{g}^T \\ \underbrace{-\nabla \mathbf{g}}_B & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \Delta \mathbf{q} \\ \Delta \lambda \end{Bmatrix} = \begin{Bmatrix} -\mathbf{r} + \lambda \cdot \nabla \mathbf{g} \\ \mathbf{f}_\lambda \\ \mathbf{g} \end{Bmatrix} \quad (7)$$

where:

$$\nabla \mathbf{r} = c_q \nabla_q \mathbf{r} + c\dot{q} \nabla_{\dot{q}} \mathbf{r} + c\ddot{q} \nabla_{\ddot{q}} \mathbf{r} \quad (8)$$

and

$$\nabla \mathbf{g} = c_q \nabla_q \mathbf{g} + c\dot{q} \nabla_{\dot{q}} \mathbf{g} \quad (9)$$

$$\nabla^2 \mathbf{g} = c_q^2 \nabla_q^2 \mathbf{g} + 2c_q c\dot{q} \nabla_{q\dot{q}}^2 \mathbf{g} + c\dot{q}^2 \nabla_{\dot{q}\dot{q}}^2 \mathbf{g} \quad (10)$$

In most problems it is preferable to move the term $\lambda \cdot \nabla \mathbf{g}$ to the left-hand side, resulting as:

$$\begin{bmatrix} K & -\nabla \mathbf{g}^T \\ -\nabla \mathbf{g} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \Delta \mathbf{q} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} -\mathbf{r} \\ \mathbf{f} \\ \mathbf{g} \end{Bmatrix} \quad (11)$$

This system can be in the quadratic programming form:

$$\begin{aligned} \min_{\Delta \mathbf{q}} \left[\frac{1}{2} \Delta \mathbf{q}^T \mathbf{K} \Delta \mathbf{q} + \Delta \mathbf{q}^T (\mathbf{r} - \nabla \mathbf{g}^T \boldsymbol{\lambda}) \right] \\ \text{s.t. } \nabla \mathbf{g}^T \Delta \mathbf{q} + \mathbf{g} = 0 \end{aligned} \quad (12)$$

We focus on this problem as follows:

Given $\mathbf{r} \in \mathbb{R}^n$, $\nabla \mathbf{r} \in \mathbb{R}^{n \times n}$ (in a sparse format), $\mathbf{g} \in \mathbb{R}^m$ where constraints can be topologically ordered, $\nabla \mathbf{g} \in \mathbb{R}^{m \times n}$ (in a sparse format), and the product $\boldsymbol{\lambda} \cdot \nabla^2 \mathbf{g}$ (in a sparse format), determine $\Delta \mathbf{q}$ and $\boldsymbol{\lambda}$ solving a $(n-m) \times (n-m)$ sparse linear system and a $m \times m$ triangular sparse system with $(n-m)$ right-hand sides.

3. Solution by use of the constraint gradient null space

If $\text{rank}[\nabla \mathbf{g}] = m$, we use a basis for the null space of $\nabla \mathbf{g}$, $\mathbf{Z} \in \mathbb{R}^{n \times (n-m)}$ where columns of \mathbf{Z} are basis vectors. Introducing $\mathbf{Y} \in \mathbb{R}^{n \times m}$ such that $[\mathbf{Z} \mid \mathbf{Y}]$ spans \mathbb{R}^n , we partition $\Delta \mathbf{q}$ as the sum of a particular solution $\Delta \hat{\mathbf{q}}$ and a term $\Delta \tilde{\mathbf{q}}$ depending on free parameters:

$$\Delta \mathbf{q} = \Delta \hat{\mathbf{q}} + \Delta \tilde{\mathbf{q}} \quad (13)$$

with $\Delta \hat{\mathbf{q}} = \mathbf{Y} \Delta \mathbf{q}_1$ and $\Delta \tilde{\mathbf{q}} = \mathbf{Z} \Delta \mathbf{q}_2$. A depiction of this decomposition is shown in Figure 1. Using the property $\nabla \mathbf{g} \mathbf{Z} = 0$ and introducing a new matrix $\mathbf{B} \in \mathbb{R}^{m \times n}$, we obtain the solution [21] in four steps:

1. Determine $\Delta \mathbf{q}_1$ by solving the system $\mathbf{B} \mathbf{Y} \Delta \mathbf{q}_1 = \mathbf{g}$.
2. Determine $\Delta \mathbf{q}_2$ by solving the system $\mathbf{Z}^T \mathbf{K} \mathbf{Z} \Delta \mathbf{q}_2 = \mathbf{Z}^T (\mathbf{f} - \mathbf{K} \mathbf{Y} \Delta \mathbf{q}_1)$.
3. Determine the complete set of unknowns $\Delta \mathbf{q} = \mathbf{Y} \Delta \mathbf{q}_1 + \mathbf{Z} \Delta \mathbf{q}_2$.
4. Determine $\boldsymbol{\lambda}$ by solving the constraint equation $\mathbf{Y}^T \mathbf{B}^T \boldsymbol{\lambda} = \mathbf{Y}^T (\mathbf{f} - \mathbf{K} \Delta \mathbf{q})$.

In the finite-element literature, $\Delta \mathbf{q}_1$ is known as a set of slave degrees-of-freedom and $\Delta \mathbf{q}_2$ as the set of master degrees-of-freedom. Obtaining matrices \mathbf{Y} and \mathbf{Z} is a task that can be accomplished by partitioning of degrees of freedom corresponding to specific columns of \mathbf{B} . Fletcher and Johnson [22] call this *direct elimination*. By partitioning (by permutation) \mathbf{B} in two sub-matrices: a nonsingular $\mathbf{B}_1 \in \mathbb{R}^{m \times m}$ and $\mathbf{B}_2 \in \mathbb{R}^{m \times (n-m)}$: $\mathbf{B} = [\mathbf{B}_1 \mid \mathbf{B}_2]$ we obtain the *fundamental* basis \mathbf{Z} and the matrix \mathbf{Y} :

$$\mathbf{Y} = \begin{bmatrix} \mathbf{B}_1^{-1} \\ 0 \end{bmatrix} \in \mathbb{R}^{[m+(n-m)] \times m} \quad (14)$$

$$\mathbf{Z} = \begin{bmatrix} -\mathbf{B}_1^{-1} \mathbf{B}_2 \\ \mathbf{I} \end{bmatrix} \in \mathbb{R}^{[m+(n-m)] \times (n-m)} \quad (15)$$

The solution for $\Delta \mathbf{q}$ and $\boldsymbol{\lambda}$ arises from the column permutation of \mathbf{B} such that a well-conditioned \mathbf{B}_1 is obtained. This is not the sole requirement. Since two

reduced systems are present: $m \times m$ $\mathbf{B} \mathbf{Y} \Delta \mathbf{q}_1 = \mathbf{g}$ and the null-space system $(n-m) \times (n-m)$, both matrices are sparse and retain some sparsity from the original problem to take advantage of existing direct or iterative sparse solvers. With the choice for \mathbf{Y} (14), $\Delta \mathbf{q}_1 = \mathbf{g}$, but sparsity concerning the result of the product depends on the specific form of the constraint gradients. Minimization of fill-in for these problems is discussed by Benzi et al. [11].

This approach to saddle point solution is known (cf. [22]) although not in the context of equations of motion. We further explore this line of solution and propose an efficient approach:

1. Determine $\Delta \mathbf{q}_2$ by solving the reduced system:

$$\begin{aligned} (\mathbf{B}_2^T \mathbf{B}_1^{-T} \mathbf{K}_{11} \mathbf{B}_1^{-1} \mathbf{B}_2 - \mathbf{K}_{21} \mathbf{B}_1^{-1} \mathbf{B}_2 - \mathbf{B}_2^T \mathbf{B}_1^{-T} \mathbf{K}_{12} + \mathbf{K}_{22}) \Delta \mathbf{q}_2 \\ = \mathbf{f}_2 - \mathbf{B}_2^T \mathbf{B}_1^{-T} \mathbf{f}_1 + (\mathbf{B}_2^T \mathbf{B}_1^{-T} \mathbf{K}_{11} \mathbf{B}_1^{-1} - \mathbf{K}_{21} \mathbf{B}_1^{-1}) \mathbf{g} \end{aligned} \quad (16)$$

2. Assemble $\Delta \mathbf{q}$:

$$\Delta \mathbf{q} = \begin{Bmatrix} \underbrace{\mathbf{B}_1^{-1} (\mathbf{g} - \mathbf{B}_2 \Delta \mathbf{q}_2)}_{\Delta \mathbf{q}_1} \\ \Delta \mathbf{q}_2 \end{Bmatrix} \quad (17)$$

3. Determine $\boldsymbol{\lambda}$ by solving

$$\boldsymbol{\lambda} = \mathbf{B}_1^{-T} \mathbf{f}_1 - \mathbf{B}_1^{-T} \mathbf{K}_{11} \Delta \mathbf{q}_1 - \mathbf{B}_1^{-T} \mathbf{K}_{12} \Delta \mathbf{q}_2 \quad (18)$$

where $\mathbf{B}_1^{-T} = [\mathbf{B}_1^{-1}]^T$. Note that, in (17), $\Delta \mathbf{q}_1^* = \mathbf{B}_1^{-1} (\Delta \mathbf{q}_1 - \mathbf{B}_2 \Delta \mathbf{q}_2)$. Various alternatives for solution of this problem are presented by Rees and Scott [21]. Our choice for this approach emerges from the fact that, in our applications, \mathbf{B}_1 can be lower-triangular and sparse, resulting in a sparse \mathbf{Z} if the constraint gradient matrix \mathbf{B} graph is acyclic.

4. Constraint ordering and local preassignment of pivots

One important conclusion concerning the solution with the *fundamental basis* is that the partition of \mathbf{B} (by selection of \mathbf{B}_1) is the crucial ingredient to obtain this solution. We therefore focus on the method for efficiently determining \mathbf{Z} with the generality to be applied in large-scale discretization software. It is important that \mathbf{Z} is sparse so that the result $\mathbf{Z}^T \mathbf{K} \mathbf{Z}$ is also sparse. This issue has been studied by Gilbert and Heath [23] who focused on the sparsity of the result. A useful remark in [23] is that, if the QR decomposition is used in \mathbf{B} , that is equivalent to a permutation. An alternative to the method of Gilbert and Heath was proposed by Coleman and Pothen who achieved a triangular form for the lower submatrix of $\mathbf{B}_1^{-1} \mathbf{B}_2$. Alternatively, Gotsman and Toledo [24] used a LU

factorization with partial pivoting to obtain the basis. That work is somehow related to ours but in our case pivots are preassigned by the analyst.

We find a lower-triangular form of \mathbf{B}_1 (and therefore \mathbf{B}_1^{-1}) by identifying the pivot order a priori⁴ and performing a topological ordering of degrees-of-freedom and therefore of constraints (by the a priori pivots) in the matrix \mathbf{B} . The directed graph of a triangular matrix is acyclic.

We start by writing the array \mathbf{g} in component form and select a given $i(j)$ such that Δq_i is the pivot variable of the equation $\nabla g_j \cdot \Delta \mathbf{q} = -g_j$. First, we select pivots so that each row of \mathbf{B} corresponds to a pivot, i.e. the i^{th} pivot is the value $B_{i,1}$ for $i = 1, \dots, m$. This is performed by permuting degrees-of-freedom (columns of \mathbf{B}). Then we transform \mathbf{B}_1 to a triangular form by performing a topological ordering of the m constraints. Given this ordering, a lower triangular sparse matrix is produced. In full (dense) format, the topologically ordered \mathbf{B} , \mathbf{B}_p has the following form:

$$\mathbf{B}_p = -[\nabla \mathbf{g}_1 | \nabla \mathbf{g}_2] = [\mathbf{L}_1 | \mathbf{B}_2] \quad (19)$$

$$= \begin{bmatrix} B_{1,1} & 0 & \cdots & 0 & B_{1,m+1} & \cdots & B_{1,n} \\ B_{2,1} & B_{2,2} & \cdots & 0 & B_{2,m+1} & \cdots & B_{2,n} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ B_{m,1} & B_{m,2} & \cdots & B_{m,m} & B_{m,m+1} & \cdots & B_{m,n} \end{bmatrix} \quad (20)$$

where both constraint permutations and degree-of-freedom permutation of \mathbf{B} were performed. We remark that the dense form for \mathbf{B}_p was employed in (20) for representation purposes only. We have \mathbf{Y} and \mathbf{Z} defined as:

$$\mathbf{Y} = \begin{bmatrix} \mathbf{L}_1^{-1} \\ 0 \end{bmatrix} \quad (21)$$

$$\mathbf{Z} = \begin{bmatrix} \mathbf{Z}_t \\ \mathbf{I} \end{bmatrix} \quad (22)$$

where \mathbf{Z}_t is determined by solving the triangular system with $n - m$ right-hand sides:

$$\mathbf{L}_1 \mathbf{Z}_t = -\mathbf{B}_2 \quad (23)$$

We note that (23) involves three sparse matrices and is not, to the best of our knowledge, a standard operation in the sparse bibliography, since \mathbf{L}_1 is sparse and triangular and \mathbf{Z}_t is also sparse. In the literature, (see, e.g. [25]), the solution \mathbf{Z}_t is dense. In addition, it is possible to directly solve for \mathbf{Z} by changing \mathbf{L}_1 so that a $(n - m)^2$ identity matrix appears beneath \mathbf{Z}_t :

$$\mathbf{L}_1^* \mathbf{Z} = -\mathbf{B}_2 \quad (24)$$

which we describe here. The explicit form (21) of \mathbf{Y} is not required in our Algorithm. In summary, we have:

1. After the determination of \mathbf{Z} , we calculate $\Delta \mathbf{q}_2$ by solving the following reduced system:

$$(\mathbf{Z}^T \mathbf{K} \mathbf{Z}) \Delta \mathbf{q}_2 = \underbrace{\mathbf{Z}^T \left[\mathbf{f} - \mathbf{K} \begin{Bmatrix} \Delta \hat{\mathbf{q}} \\ \mathbf{L}_1^* \mathbf{g} \\ 0_{nm} \end{Bmatrix} \right]}_{\mathbf{f}_2^*} \quad (25)$$

2. Then we determine $\Delta \mathbf{q}$ as follows:

$$\Delta \mathbf{q} = \Delta \hat{\mathbf{q}} + \mathbf{Z} \Delta \mathbf{q}_2 \quad (26)$$

3. Finally, λ is calculated by the following equality (it consists of a sparse triangular solve with dense right-hand side)

$$\lambda = [\mathbf{L}_1^{-T} \quad | 0] \underbrace{[\mathbf{f} - (\mathbf{K} \Delta \mathbf{q})]}_{\mathbf{f}_1^*} \quad (27)$$

where $\mathbf{L}_1^{-T} = [\mathbf{L}_1^{-1}]^T$. Besides permutations and a specific topological ordering, sparse operations required in this Algorithm are:

1. Sparse matrix sum $\nabla \mathbf{r} + (-\lambda \cdot \nabla^2 \mathbf{g})$.
2. Sparse pre- and postmultiplication $\mathbf{Z}^T \mathbf{K} \mathbf{Z}$ with sparse result.
3. Sparse linear solution $(\mathbf{Z}^T \mathbf{K} \mathbf{Z}) \Delta \mathbf{q}_2 = \mathbf{f}_2^*$.
4. Sparse matrix/dense vector multiplication, e.g. $\mathbf{K} \Delta \mathbf{q}$ with dense result.
5. Modified sparse triangular solve with multiple sparse right-hand sides $\mathbf{L}_1^* \mathbf{Z} = -\mathbf{B}_2$ with sparse result.
6. Sparse triangular solve $\mathbf{L}_1^{-1} \mathbf{g}$ with dense result.
7. Sparse triangular solve $\mathbf{L}_1^{-T} \mathbf{f}_1^*$ with dense result.

We use compact sparse row format (CSR) for sparse representation. Operation 1 is in essence a standard sparse merge, Operation 2 (sparse linear solution) can be performed with either direct or iterative solvers. The BiCGStab(2) solver is used here [26]. Operation 3 is a variation on the sparse matrix-matrix multiplication, not requiring an explicit transpose of \mathbf{Z} . This is discussed here in detail. Operation 4 is a traditional sparse matrix-vector multiplication. Operation 5 is not standard and is discussed here. Operations 6 and 7 are somehow standard, but we still show the corresponding Algorithms which have some particularities. Note that, due to topological ordering, the Lagrange multiplier vector λ has a precise physical meaning, depending on the problem under study.

⁴First active column for each row of \mathbf{B} is the selected pivot degree-of-freedom.

5. Complete algorithm and reference implementation

With the purpose of avoiding redundancy in each Newton iteration step, we separate the symbolic from the numeric calculations. Often, to ensure memory alignment, two symbolic operations are required before proceeding to the numeric stage. An example of this requirement is the sparse triangular solve with multiple sparse right-hand sides. For guidance, we introduce the following notation for the nonzero pattern of a given matrix A in CSR form (Fortran 2003 conventions are used):

1. NRA is the number of rows of A .
2. NCA is the number of columns of A , which can be determined as $NCA = \max_{LA \in [1, IA[NRA+1]-1]} [JA[LA]]$, see below.
3. IA[1:NRA + 1] is the row start array. Row RA starts at index IA[RA] and ends at index IA[RA + 1] - 1.
4. JA[1:IA[NRA + 1] - 1] is the active column number list. Active columns numbers for row RA are JA[IA[RA]]...JA[IA[RA + 1] - 1].
5. VA[1:IA[NRA + 1] - 1] stores the active coefficients of A in row-by-row order.
6. IAT[1:NCA + 1] is the column start array (or the row start of A^T).
7. JAT[1:IAT[NCA + 1] - 1] is the active row number list (or the active column list of A^T).
8. IJA[1:IAT[NCA + 1] - 1] is the index list for the coefficients of A^T in VA. Given column CA and the local position KA, which is the (KA - 1)th active row for IAT[CA], we have $VAT[IAT[CA] - 1 + KA] = VA[IA[JAT[IAT[CA] - 1 + KA]] - 1 + IJA[IAT[CA] - 1 + KA]]$. Therefore, if VA exists, dedicated storage for VAT is not required.
9. IJAT[1:IA[NRA + 1] - 1] is the index list for the coefficients of A in VAT. Given row RA and the local position LA, which is the (LA - 1)th active column for IA[RA], we have $VA[IA[RA] - 1 + LA] = VAT[IAT[JA[IA[RA] - 1 + LA]] - 1 + IJAT[IA[RA] - 1 + LA]]$. Therefore, if VAT exists, dedicated storage for VA is not required.
10. Index RA indicates a row of A , as index CA indicates a column of A .

Since the rows and columns of A represent members of certain sets (constraints, degrees-of-freedom, elements, etc.) and can be unequivocally defined by the set

$$GR[A] = \{NRA, IA, JA\} \quad (28)$$

Table 1. Input/output parameters for algorithm 1.

| Variable | Description |
|----------|---|
| MG | Number of constraints |
| MJ | Number of rows and columns of the Jacobian |
| R | Residual values (r) |
| G | Constraint values (g) |
| IJ | Row start array for the Jacobian |
| JJ | Active column numbers for the Jacobian |
| VJ | Coefficients of the Jacobian (∇r) |
| IG | Row start array for the constraint gradients |
| JG | Active column numbers for the constraint gradients |
| VG | Coefficients for the constraint gradients (∇g) |
| IG2 | Row start array for the constraint Hessians |
| JG2 | Active column numbers for the constraint Hessians |
| VG2 | Coefficients for the constraint Hessians ($\lambda \nabla^2 g$) |
| L | Lagrange multipliers (λ) |
| S | Solution (Δq) |

We call this set the graph of A . Using a simplified version of Fortran 2003-like syntax, we describe the operations and leave the sparse solution to the reader, as this aspect is highly dependent on the problem structure and conditioning. The source code is available in GitHub, cf. [27].

Using the notation in Table 1, Algorithm 1 determines the solution to the stated problem. Lines in Algorithm 1 are the following:

- Lines 2 and 3 calculate $K = \nabla r - \lambda \nabla^2 g$.
- Line 7 performs the column permutation and topological ordering to obtain a lower-triangular B_1 .
- Lines 8 and 9 perform the permutation of r and g .
- Line 10 performs the permutation of rows and columns of K .
- Line 11 numerically transposes ∇g .
- Lines 12 and 13 calculate $L_1^* Z = B_2$ in transposed form.
- Line 14 performs $Z \leftarrow -Z$.
- Line 15 solves $L_1 \Delta \hat{q} = g$.
- Line 16 calculates $K \Delta \hat{q}$.
- Line 17–19 calculates $f_2 = -r - K \Delta \hat{q}$.
- Line 20 calculates $f_2^* = Z^T f_2$.
- Lines 21 and 22 calculate $K^* = Z^T K Z$.
- Line 23 solves the problem $K^* \Delta q_2 = f_2^*$.
- Line 24 calculates $Z \Delta q_2$.
- Lines 25–27 calculates $\Delta q = \Delta \hat{q} + Z \Delta q_2$.
- Line 28 calculates $K \Delta q$.
- Lines 29–31 calculate $f_1^* = -r - K \Delta q$.
- Lines 32–35 solve $L_1 \lambda = f_1^*$.
- Lines 36–38 permute r , Δq and λ .

The reordering of constraints and degrees-of-freedom present in the constraint gradient matrix is shown in Algorithm 2. It starts with column permutation so that each row of the matrix B corresponds to the first active degree-of-freedom, then a topological ordering is performed and all relevant arrays are

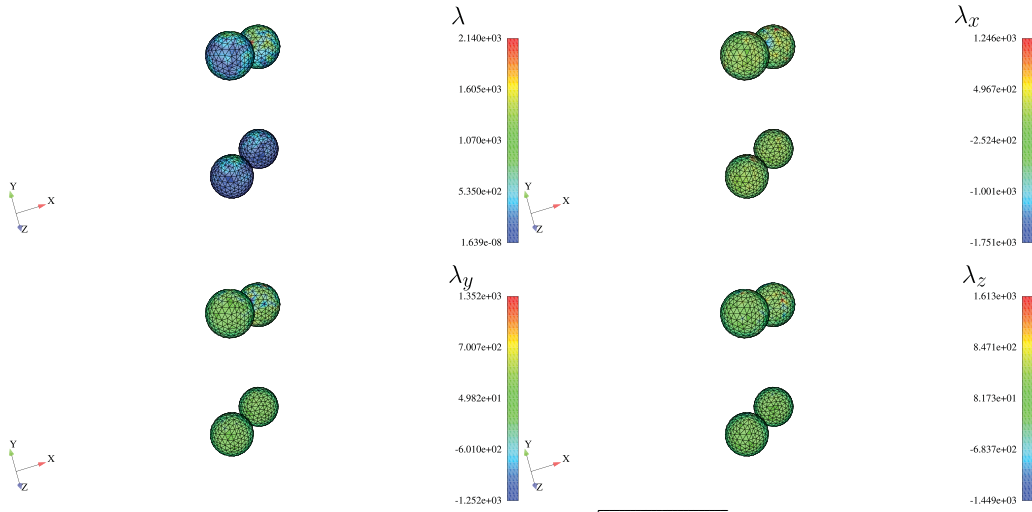


Figure 2. Values of Lagrange multipliers λ at the olivine spheres. $\lambda = \sqrt{\lambda_x^2 + \lambda_y^2 + \lambda_z^2}$. The deformed configuration is shown.

permuted. The permutation algorithms are exhibited in Algorithm 3. Finally, the modified topological ordering Algorithm is shown in 4. This allows a rectangular sparse matrix to be ordered, which was not the case of the original algorithm in [28].

The sparse sum is divided in Algorithms 5 and 6 and the transpose of a sparse relation is shown in 7. The triple product $\mathbf{Z}^T \mathbf{K} \mathbf{Z}$ is divided in 8 (symbolic) and 9 (numeric). This operation is not standard and does not consist of two sequential sparse products.

The solution $\mathbf{L}^* \mathbf{X} = -\mathbf{b}$ with three sparse matrices is described in Algorithms 10–12. The complete operation is nonstandard and therefore we describe in detail. In 10, the sequence is the following:

- In lines 3–12, we obtain the list of degrees-of-freedom reached by each non-zero element of the right-hand side \mathbf{b} . This is performed for each column of \mathbf{b} and therefore of \mathbf{X} .
- Line 13 changes IXT from the number of non-zeros to the starting indices.
- Lines 14–30 fill JXT with the column numbers.

The use of a modified depth-first algorithm (11) is here centered in marking the reachable degrees-of-freedom in MK.

Triangular solve and triangular multiplication are shown in Algorithms 13 and 14. These are relatively standard, but are shown for completeness.

6. Graph of $\nabla \mathbf{r}$ from sum of cliques and projected Hessians

It is known since the seminal work of Gustavson [29] that the graph structure of the *assembling* process in

Finite Element literature can be established by a specific sparse multiplication of the DOF connectivities:

$$\text{GR}[\nabla \mathbf{r}] = \text{GR}[\text{EDOF}^T \text{EDOF}] \quad (29)$$

where EDOF is the element-DOF connectivity relation. Note that $\nabla \mathbf{r} \neq \text{EDOF}^T \text{EDOF}$, only the graph is coincident. The values for $\nabla \mathbf{r}$ are obtained from a clique sum (ne elements):

$$[\nabla \mathbf{r}]_{ij} = \sum_{e=1}^{ne} [\nabla \mathbf{r}]_{eij} \quad (30)$$

where the index e identifies the given element and indices i and j identify the corresponding degrees-of-freedom (rows and columns of $\nabla \mathbf{r}$) of element e . As for the term $\lambda \cdot \nabla^2 \mathbf{g}$, it is a sum of constraint second-derivative matrices, as follows:

$$[\lambda \cdot \nabla^2 \mathbf{g}]_{ij} = \sum_{l=1}^m \lambda_l \nabla^2 g_{lij} \quad (31)$$

where the two last indices of a given constraint l Hessian, $\nabla^2 g_{lij}$, correspond to degrees-of-freedom i and j . Since both (30) and (31) are sums of clique graphs (e and l for elements and constraints, respectively), these terms are treated as clique contributions to the graph of $\nabla \mathbf{r}$. In practice, the terms in (31) are treated as additional elements.

The symbolic and numeric process of assembling without search has been described in our previous work [2] where a different approach was adopted. As in that work, direct addressing is avoided by use of a source pointer. The two Algorithms 15 and 16 show the procedures, making use of previous Algorithms.

The numeric assembling follows the previous strategy, cf. [2] and is omitted here. The complete source code is available at GitHub [27].

7. Application to second-order problems with time integration

The application of the Algorithm within an implicit time integrator is straightforward. Using two consecutive time steps t_k and t_{k+1} and $h = t_{k+1} - t_k$ we use the Newmark family [30] start by establishing the update for \mathbf{q} and $\dot{\mathbf{q}}$:

$$\mathbf{q}_{k+1} = \mathbf{q}_k + h\dot{\mathbf{q}}_k + \frac{h^2}{2}[(1 - 2\beta)\ddot{\mathbf{q}}_k + 2\beta\ddot{\mathbf{q}}_{k+1}] \quad (32)$$

$$\dot{\mathbf{q}}_{k+1} = \dot{\mathbf{q}}_k + h[(1 - \gamma)\ddot{\mathbf{q}}_k + \gamma\ddot{\mathbf{q}}_{k+1}] \quad (33)$$

where β and γ are integrator parameters. Based on consistency with variational integrators [30], we choose $\beta = \frac{1}{4}$ and $\gamma = \frac{1}{2}$. For these values, inverting the expressions we obtain the formulas for updated $\dot{\mathbf{q}}$ and $\ddot{\mathbf{q}}$:

$$\dot{\mathbf{q}}_{k+1} = -\dot{\mathbf{q}}_k + \frac{2(\mathbf{q}_{k+1} - \mathbf{q}_k)}{h} \quad (34)$$

$$\ddot{\mathbf{q}}_{k+1} = -\ddot{\mathbf{q}}_k - \frac{4\dot{\mathbf{q}}_k}{h} + \frac{4(\mathbf{q}_{k+1} - \mathbf{q}_k)}{h^2} \quad (35)$$

In the starting procedure ($t_0 = 0$) we obtain the acceleration $\ddot{\mathbf{q}}_0$ from the constrained system and a vector of Lagrange multipliers $\dot{\boldsymbol{\lambda}}_0$ corresponding to the constraint enforcement:

$$\begin{aligned} & [\nabla\dot{\mathbf{q}}\mathbf{r} - \boldsymbol{\lambda}_0 \cdot \nabla\dot{\mathbf{q}}^2\mathbf{g} - \nabla\dot{\mathbf{q}}\mathbf{g}^T - \nabla\dot{\mathbf{q}}\mathbf{g}\boldsymbol{\theta}] \begin{Bmatrix} \Delta\dot{\mathbf{q}}_0 \\ \dot{\boldsymbol{\lambda}}_0 \end{Bmatrix} \\ & = \underbrace{\left\{ -\mathbf{r}(q_0, \dot{q}_0, \ddot{q}_0) \mathbf{g}(q_0, \dot{q}_0, \ddot{q}_0) \right\}}_f \end{aligned} \quad (36)$$

Note that, since a total solution for the Lagrange multipliers is adopted, time-integration is not required for $\boldsymbol{\lambda}$. From these relations, we have the following three constants, c_q , $c\dot{q}$, and $c\ddot{q}$:

$$c_q = \begin{cases} 0 & k=0 \\ 1 & k>0 \end{cases} \quad (37)$$

$$c\dot{q} = \begin{cases} 0 & k=0 \\ \frac{2}{\Delta t} & k>0 \end{cases} \quad (38)$$

$$c\ddot{q} = \begin{cases} \frac{1}{4} & k=0 \\ \frac{1}{\Delta t^2} & k>0 \end{cases} \quad (39)$$

8. Numerical example: creep of basaltic magma with rigid inclusions of rigid olivine

We use an example with inertia and rate-dependence to assess the source code (which is available in GitHub [27]). A basaltic magma using Bingham

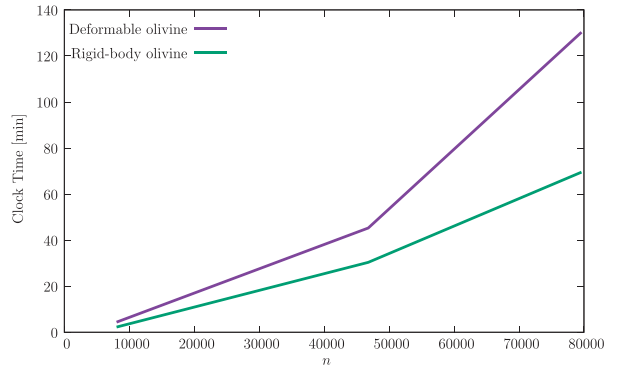


Figure 3. Computing clock time for the three meshes and two cases (deformable and rigid). A desktop with a Intel i5-4690K cpu with 16 GB of memory is adopted.

rheology is employed. Since olivine has a much higher yield stress, the representation of four spheres of olivine inside the magma is based on one of two cases:

Case I where olivine spheres are considered deformable.

Case II where rigid body constraints are applied to the olivine spheres, with four master points and four spheres of slave points.

Figure 2 shows the relevant properties and also the deformed configurations. The block of magma is subjected only to the self-weight.

For the rigid body constraint, we consider two steps in sequence: k and $k+1$. For two nodes, a master (m) and a slave (s), we introduce the notation for the displacements \mathbf{u}_{k+1}^m and \mathbf{u}_k^m of the master node and \mathbf{u}_{k+1}^s and \mathbf{u}_k^s of the slave node. The positions (in step k) of the master/slave nodes are \mathbf{x}_k^m and \mathbf{x}_k^s , respectively. The rigid body constraint is introduced by the residual as:

$$\begin{aligned} r_l = & \left\{ (\mathbf{u}_{k+1}^s - \mathbf{u}_k^s) - (\mathbf{u}_{k+1}^m - \mathbf{u}_k^m) \right. \\ & \left. + [\mathbf{I} - \mathbf{R}(\boldsymbol{\theta}_{k+1}^m - \boldsymbol{\theta}_k^m)] (\mathbf{x}_k^s - \mathbf{x}_k^m) \right\}_l \end{aligned} \quad (40)$$

where $\mathbf{R}(\Delta\boldsymbol{\theta})$ is the rotation matrix with $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}_{k+1}^m - \boldsymbol{\theta}_k^m$. In (40), l is the coordinate index. We use the standard notation and $\Delta\theta = \|\Delta\boldsymbol{\theta}\|_2$. The classical form of the Rodrigues rotation matrix is the following (software to calculate the rotation matrix and its first and second derivatives is available in [27]):

$$\begin{aligned} \mathbf{R}(\Delta\boldsymbol{\theta}) = & \mathbf{I} + \frac{\sin(\Delta\theta)}{\Delta\theta} \begin{bmatrix} 0 & -\Delta\theta_3 & \Delta\theta_2 \\ \Delta\theta_3 & 0 & -\Delta\theta_1 \\ -\Delta\theta_2 & \Delta\theta_1 & 0 \end{bmatrix} + \\ & \frac{2 \sin^2\left(\frac{\Delta\theta}{2}\right)}{\Delta\theta^2} \begin{bmatrix} -\Delta\theta_2^2 - \Delta\theta_3^2 & \Delta\theta_1\Delta\theta_2 & \Delta\theta_1\Delta\theta_3 \\ \Delta\theta_1\Delta\theta_2 & -\Delta\theta_1^2 - \Delta\theta_3^2 & \Delta\theta_2\Delta\theta_3 \\ \Delta\theta_1\Delta\theta_3 & \Delta\theta_2\Delta\theta_3 & -\Delta\theta_1^2 - \Delta\theta_2^2 \end{bmatrix} \end{aligned} \quad (41)$$

Table 2. Constitutive properties for basaltic magma.

Basaltic magma: mechanical properties obtained from [33]–[35]

| Prop. | Description | Value | Units |
|-------------------------------------|--|--|--------------------|
| κ | Bulk modulus | 24.2×10^9 | Pa |
| ν | Poisson coefficient | 0.25 | – |
| E | Elasticity modulus ($= 3\kappa(1-2\nu)$) | 36.3×10^9 | Pa |
| η | Viscosity [31]. $T \in [1448, 1623]$ K | $\exp(-38 + \frac{62 \times 10^3}{T})$ | Pa s |
| y | Initial yield stress (τ_1 in Chevrel et al. [34]) see also Piombo [36] | 3.49×10^3 | Pa |
| ρ | Mass density [33] | 4500 | kg m^{-3} |
| Olivine: mechanical properties [37] | | | |
| Prop. | Description | Value | Units |
| κ | Bulk modulus | 129.4×10^9 | Pa |
| ν | Poisson coefficient | 0.249 | – |
| E | Elasticity modulus ($= 3\kappa(1-2\nu)$) | 195×10^9 | Pa |
| ρ | Mass density | 3355 | kg m^{-3} |

For the magma we use a J_2 plasticity formulation with rate dependence, corresponding to the intended behavior. Therefore, we have the pressure-deviatoric split of the stress \mathbf{S} :

$$\mathbf{S} = -\bar{p}\mathbf{I} + \mathbf{S}^d \quad (42)$$

with \mathbf{S}^d is the deviatoric stress and \bar{p} is the pressure. We use the viscosity for Grímsvötn basaltic magma [31], focusing on the lower temperature $T = 1448$ K:

$$\eta = 123.678 \text{ Pa}\cdot\text{s} \quad (43)$$

The yield function is now given by:

$$f = \sqrt{\frac{1}{2} \mathbf{S}^d : \mathbf{S}^d} - y - \eta \dot{\epsilon}_p \quad (44)$$

where y is the quasi-static yield stress. This yield function is a variant of the von-Mises [32] and subject to the same treatment. Specifically, the Algorithm in page 152 of Simo and Hughes [32] is adopted. Properties of basaltic magma and olivine are summarized in Table 2 with the corresponding sources.

The savings resulting from the use of a rigid-body assumption are shown in Figure 3. The ifort-2013 Fortran compiler is used with the -O2 flag in a Intel i5-4690K cpu with 16 GB of memory. A BiCGStab(2) [26] solver was used. Three meshes are used with {2681, 15598, 26540} nodes (including masters) and {13214, 83424, 144247} elements. It can be observed that for $n = 79620$ (initial degrees-of-freedom), the nonlinear problem is solved in roughly half the time with the rigid assumption. Reactions as function of time are shown in Figure 4. Oscillations due to the presence of inertial forces are visible, and after dissipation caused by plasticity becomes predominant, these are attenuated. The displacement of the monitored node is presented in Figure 5.

We conclude that the rigid assumption produces slightly stiffer results but substantial savings are achieved with the rigid-body assumption.

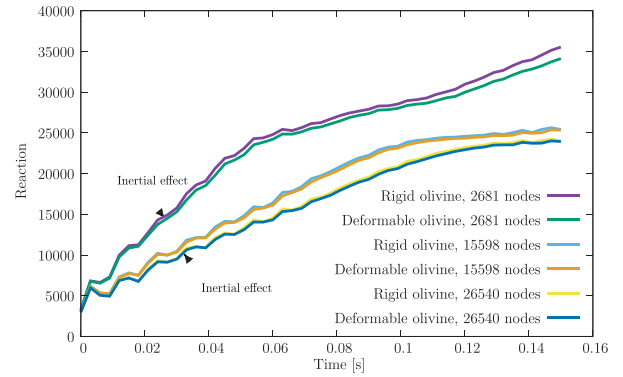


Figure 4. Reactions as a function of time for the three meshes and two olivine cases.

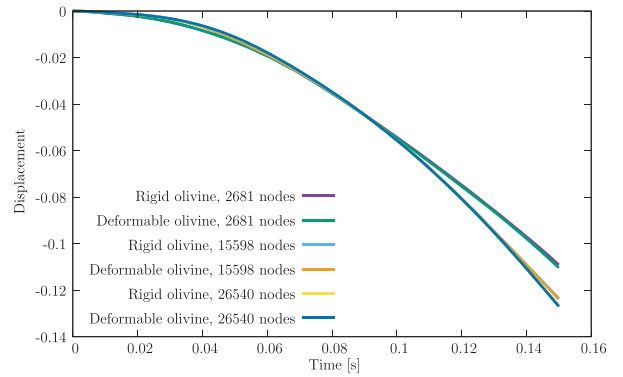


Figure 5. Monitored node displacement as a function of time for the three meshes and two cases.

9. Conclusions

We created an algorithm and corresponding Fortran 2003 code for efficiently inserting a specific class of equality constraints in a finite element code. In contrast with performing the operation by cliques (see [12]) we here explicitly use complete sparse matrices, with several new and adapted sparse algorithms. With triangularization of the slave sub-matrix, we obtained substantial savings in computational cost when comparing a rigid-body approach with the stiff deformable solution. The source code is available in GitHub [27].

Algorithms

Algorithm 1 Overall Algorithm to solve the equality-constrained nonlinear sparse problem in the acyclic case. Cf. Table 1 for the parameter list.

```

1 SOLVECONSTRAINEDSPARSE(MG,MJ,R,G,IJ,JJ,VJ,IG,JG,VG,IG2,JG2,VG2,L,S)
2 CALL APLUSBFIRSTPART(MJ,IJ,JJ,IG2,JG2,IS)
3 CALL APLUSBSECONDPART(MJ,1.0d00,IJ,JJ,VJ,-1.0d00,IG2,JG2,VG2,IS,JS,VS)
4 DO KG=1,IG(MG+1)-1
5   VG(KG)=-VG(KG)
6 END DO
7 CALL TREATCONSTRAINTMATRIX(MG,MJ,IG,JG,VG,NOLD,IGN,JGN,VGN)
8 CALL RCHANGEPERMUTATION(MJ,NOLD,R,2)
9 CALL RCHANGEPERMUTATION(MJ,NOLD,G,2)
10 CALL ROWCOLPERM(MJ,IS,JS,VS,INS,JNS,VNS,NOLD)
11 CALL SPARSETRANSP(MG,MJ,IGN,JGN,VGN,IGT,JGT,VGT)
12 CALL SPARSEPARSELXBSYMB(MG,IGT,JGT,IZT,JZT,MJ-MG,IGT(MG+1),JGT)
13 CALL SPARSEPARSELXBNUM(MG,IGT,JGT,VGT,IZT,JZT,VZT,MJ-MG,IGT(MG+1),JGT,VGT)
14 VZT=-VZT
15 CALL SPARSEDENSELXBCSC(MG,IGT,JGT,VGT,Q1,G)
16 CALL ATIMESV(MJ,INS,JNS,VNS,Q1,F2)
17 DO I=1,MJ
18   F2(I)=-R(I)-F2(I)
19 END DO
20 CALL ATIMESV(MJ-MG,IZT,JZT,VZT,F2,F2R)
21 CALL SPARSEZKZSYMB(MJ,MJ-MG,IZ,JZ,IZT,JZT,IJZT,INS,JNS,IREDS,JREDS)
22 CALL SPARSEZKZNUM(MJ,MJ-MG,IZ,JZ,IZT,JZT,VZT,IJZT,INS,JNS,VNS,IREDS,JREDS,VREDS)
23 CALL ITERSOLUC(MJ-MG,IREDS,JREDS,VREDS,Q2,F2R)
24 CALL ATTIMESV(MJ-MG,IZT,JZT,VZT,Q2,S)
25 DO I=1,MJ
26   S(I)=S(I)+Q1(I)
27 END DO
28 CALL ATIMESV(MJ,INS,JNS,VNS,S,F1)
29 DO I=1,MJ
30   F1(I)=-R(I)-F1(I)
31 END DO
32 CALL SPARSEDENSEUXBCSR(MG,IGT,JGT,VGT,L,F1)
33 DO I=MG+1,MJ
34   L(I)=0.0D00
35 END DO
36 CALL RCHANGEPERMUTATION(MJ,NOLD,R,1)
37 CALL RCHANGEPERMUTATION(MJ,NOLD,S,1)
38 CALL RCHANGEPERMUTATION(MJ,NOLD,L,1)
39 END SOLVECONSTRAINEDSPARSE

```

Algorithm 2 Reordering of degrees-of-freedom and constraints to obtain a lower triangular sub-matrix.

```

TREATCONSTRAINTMATRIX(MGRAD,MJAC,IGRAD,JGRAD,VGRAD,NEWOLD,IGRADNEW,JGRADNEW,VGRADNEW)
DO RJAC=1,MJAC
  NEWOLD(RJAC)=0
END DO
DO RGRAD=1,MGRAD
  CGRAD=JGRAD(IGRAD(RGRAD))
  NEWOLD(CGRAD)=RGRAD
END DO
NK=0
DO RJAC=1,MJAC
  IF(NEWOLD(RJAC).EQ.0) THEN
    NK=NK+1
    NEWOLD(RJAC)=MGRAD+NK
  END IF
END DO
DO KGRAD=1,IGRAD(MGRAD+1)-1
  JGRAD(KGRAD)=NEWOLD(JGRAD(KGRAD))
END DO
CALL DOFTOPRECTANGLE(MGRAD,MJAC,IGRAD,JGRAD,ACYCLIC,TOPD)
DO K=MGRAD+1,MJAC
  TOPD(K)=K
END DO
DO K=1,MJAC
  ITOPD(TOPD(K))=K
END DO
CALL ROWCOLPERM(MGRAD,IGRAD,JGRAD,VGRAD,IGRADNEW,JGRADNEW,VGRADNEW,ITOPD)
DO RJAC=1,MJAC
  NEWOLD(RJAC)=ITOPD(NEWOLD(RJAC))
END DO
END TREATCONSTRAINTMATRIX

```

Algorithm 3 Sparse permutation Algorithm and dense vector permutation.

```

ROWCOLPERM (NRA , IA , JA , VA , IANEW , JANEW , VANEW , OLDNEW)
...
DO RA=1 , NRA
  RANEW=OLDNEW (RA)
  IANEW (RANEW+1)=IA (RA+1) - IA (RA)
END DO
IANEW (1)=1
DO RA=1 , NRA
  IANEW (RA+1)=IANEW (RA+1) + IANEW (RA)
END DO
DO RA=1 , NRA
  KANEW=IANEW (OLDNEW (RA)) - 1
  DO KA=IA (RA) , IA (RA+1) - 1
    KANEW=KANEW+1
    JANEW (KANEW)=JA (KA)
    VANEW (KANEW)=VA (KA)
  END DO
END DO
DO KA=1 , IANEW (NRA+1) - 1
  JANEW (KA)=OLDNEW (JANEW (KA))
END DO
END ROWCOLPERM

RCHANGEPERMUTATION (N , IPERM , LIS , IJOB)
SELECT ( IJOB)
CASE (1)
  DO I=1 , N
    LIS (I)=LIST (IPERM (I))
  END DO
CASE (2)
  DO I=1 , N
    LIS (IPERM (I))=LIST (I)
  END DO
END SELECT
END RCHANGEPERMUTATION

```

Algorithm 4 Topological ordering of a rectangular matrix (or many-to-many relation) in CSR format. A modified version of the Algorithm by D. Jungnickel [28]

```

DOFTOPRECTANGLE (NRA , MCA , IA , JA , ACYCLIC , INCREASINGORDER)
M=1
DO RA=1 , NRA
  DO KA=IA (RA) , IA (RA+1) - 1
    IF (JA (KA) . NE . RA . AND . JA (KA) . LE . NRA) IND (JA (KA))=IND (JA (KA)) + 1
  END DO
END DO
IK=0
DO RA=1 , NRA
  IF (IND (RA) . EQ . 0) THEN
    IK=IK+1
    L (NRA+1 - IK)=RA
  END IF
END DO
MK=NRA
DO WHILE (IK . NE . 0)
  RA=L (MK)
  MK=MK-1
  IK=IK-1
  INCREASINGORDER (M)=RA
  M=M+1
  DO J=IA (RA) , IA (RA+1) - 1
    CA=JA (J)
    IF (CA . NE . RA) THEN
      IF (CA . LE . NRA) THEN
        IND (CA)=IND (CA) - 1
        IF (IND (CA) . EQ . 0) THEN
          IK=IK+1
          L (MK+1 - IK)=CA
        END IF
      END IF
    END IF
  END DO
  END DO
  IF (M . EQ . NRA+1) THEN
    ACYCLIC=.TRUE.
  ELSE
    ACYCLIC=.FALSE.
  END IF
  DO RA=1 , NRA/2
    I1=INCREASINGORDER (NRA+1 - RA)
    INCREASINGORDER (NRA+1 - RA)=INCREASINGORDER (RA)
    INCREASINGORDER (RA)=I1
  END DO
END DOFTOPRECTANGLE

```

Algorithm 5 First part of $C = \alpha_A A + \alpha_B B$ in CSR format.

```

APLUSBFIRSTPART (NRA , IA , JA , IB , JB , IC)
...
LEN=0
IC(1)=1
DO RA=1 , NRA
  DO KA=IA (RA) , IA (RA+1) -1
    LEN=LEN+1
    CA=JA (KA)
    IW (CA)=LEN
  END DO
  DO KB=IB (RA) , IB (RA+1) -1
    CB=JB (KB)
    IF (IW (CB) . EQ . 0) THEN
      LEN=LEN+1
      IW (CB)=LEN
    END IF
  END DO
  DO KA=IA (RA) , IA (RA+1) -1
    IW (JA (KA))=0
  END DO
  DO KB=IB (RA) , IB (RA+1) -1
    IW (JB (KB))=0
  END DO
  IC (RA+1)=LEN+1
END DO
...
END APLUSBFIRSTPART

```

Algorithm 6 Second part of $C = \alpha_A A + \alpha_B B$ in CSR format.

```

APLUSBSECONDPART (NRA , ALPHA_A , IA , JA , VA , ALPHA_B , IB , JB , VB , IC , JC , VC)
...
LEN=0
IC(1)=1
DO RA=1 , NRA
  DO KA=IA (RA) , IA (RA+1) -1
    LEN=LEN+1
    CA=JA (KA)
    JC (LEN)=CA
    VC (LEN)=ALPHA_A *VA (KA)
    IW (CA)=LEN
  END DO
  DO K=IB (RA) , IB (RA+1) -1
    CB=JB (K)
    IF (IW (CB) . EQ . 0) THEN
      LEN=LEN+1
      JC (LEN)=CB
      VC (LEN)=ALPHA_B *VB (K)
      IW (CB)=LEN
    ELSE
      VC (IW (CB))=VC (IW (CB))+ALPHA_B *VB (K)
    END IF
  END DO
  DO KC=IC (RA) , LEN
    IW (JC (KC))=0
  END DO
  IC (RA+1)=LEN+1
END DO
...
END APLUSBSECONDPART

```

Algorithm 7 Transpose and addressing of transpose in CSR format.

```

SPARSETRANSP (NRA , NCA , IA , JA , VA , IAT , JAT , VAT)
...
DO RA=1 , NRA
  DO KA=IA (RA) , IA (RA+1) -1
    CA=JA (KA)
    IAT (CA)=IAT (CA)+1
  END DO
END DO
CALL MUDLIS (NCA , IAT)
ITEMP=IAT (NCA+1) -1
...
DO RA=1 , NRA
  DO KA=IA (RA) , IA (RA+1) -1
    CA=JA (KA)
    NEXTV=IAT (CA)
    IAT (CA)=NEXTV+1
    JAT (NEXTV)=RA
    VAT (NEXTV)=VA (KA)
  ENDDO
ENDDO
DO CA=NCA , 1 , -1
  IAT (CA+1)=IAT (CA)
END DO
IAT (1)=1
END SPARSETRANSP

SPARSETRANSPINDEX (NRA , NCA , IA , JA , IJA , IAT , JAT)
...
DO RA=1 , NRA
  DO KA=IA (RA) , IA (RA+1) -1
    CA=JA (KA)
    IAT (CA)=IAT (CA)+1
  END DO
END DO
CALL MUDLIS (NCA , IAT)
ITEMP=IAT (NCA+1) -1
...
DO RA=1 , NRA
  IK=0
  DO KA=IA (RA) , IA (RA+1) -1
    IK=IK+1
    CA=JA (KA)
    NEXTV=IAT (CA)
    IAT (CA)=NEXTV+1
    JAT (NEXTV)=RA
    IJA (NEXTV)=IK
  ENDDO
ENDDO
DO CA=NCA , 1 , -1
  IAT (CA+1)=IAT (CA)
END DO
IAT (1)=1
END SPARSETRANSPINDEX

MUDLIS (N , LIST)
LOL=LIST (1)
LIST (1)=1
DO IN=1 , N
  NEWV=LIST (IN)+LOL
  IN1=IN+1
  LOL=LIST (IN1)
  LIST (IN1)=NEWV
ENDDO
END MUDLIS

```

Algorithm 8 Symbolic $K_r = Z^T K Z$ in CSR format.

```

SPARSEZKZSYMB (NRZ , NCZ , IZ , JZ , IZT , JZT , IJZT , IK , JK , IKR , JKR)
...
CALL SPARSETRANSPINDEX (NCZ , NRZ , IZT , JZT , IJZT , IZ , JZ)
NRKR=NCZ
DO RKR=1 , NRKR
  IW=0
  LEN=0
  DO K1=IZT (RKR) , IZT (RKR+1) -1
    RZ=JZT (K1)
    DO K2=IK (RZ) , IK (RZ+1) -1
      CK=JK (K2)
      DO K3=IZ (CK) , IZ (CK+1) -1
        CZ=JZ (K3)
        IF (IW (CZ) . EQ .0) THEN
          LEN=LEN+1
          IW (CZ)=1
        END IF
      END DO
    END DO
  END DO
  IKR (RKR)=LEN
END DO
CALL MUDLIS (NRKR , IKR)
LEN=0
DO RKR=1 , NRKR
  IW=0
  DO K1=IZT (RKR) , IZT (RKR+1) -1
    RZ=JZT (K1)
    DO K2=IK (RZ) , IK (RZ+1) -1
      CK=JK (K2)
      DO K3=IZ (CK) , IZ (CK+1) -1
        CZ=JZ (K3)
        IF (IW (CZ) . EQ .0) THEN
          LEN=LEN+1
          JKR (LEN)=CZ
          IW (CZ)=LEN
        END IF
      END DO
    END DO
  END DO
END DO
END SPARSEZKZSYMB

```

Algorithm 9 Numeric $K_r = Z^T K Z$ in CSR format.

```

SPARSEZKZNUM (NRZ , NCZ , IZ , JZ , IZT , JZT , VZT , IJZT , IK , JK , VK , IKR , JKR , VKR)
...
LEN=0
NRKR=NCZ
DO RKR=1 , NRKR
  IW=0
  DO K1=IZT (RKR) , IZT (RKR+1) -1
    RZ=JZT (K1)
    VALZT=VZT (K1)
    DO K2=IK (RZ) , IK (RZ+1) -1
      CK=JK (K2)
      VALK=VK (K2)
      DO K3=IZ (CK) , IZ (CK+1) -1
        CZ=JZ (K3)
        VALZ=VZT (IJZT (K3)+IZT (CZ) -1)
        IF (IW (CZ) . EQ .0) THEN
          LEN=LEN+1
          VKR (LEN)=VKR (LEN)+VALZT*VALK*VALZ
          IW (CZ)=LEN
        ELSE
          LG=IW (CZ)
          VKR (LG)=VKR (LG)+VALZT*VALK*VALZ
        END IF
      END DO
    END DO
  END DO
END DO
END SPARSEZKZNUM

```

Algorithm 10 Specific symbolic sparse triangular solve with multiple sparse right-hand-sides $L^*X = -b$ with sparse result.

```

1  SUBROUTINE SPARSESPARSELXBSYMB (NCL, ILT, JLT, IXT, JXT, NCB, IBT, JBT)
2    NCX=NCL
3    DO CB=1, NCB
4      MK=0
5      NM=0
6      DO KB=IBT(CB), IBT(CB+1)-1
7        RB=JBT(KB)
8        CL=RB
9        CALL DEPTHFIRSTSEARCH(NCL, ILT, JLT, CL, NM, MK)
10     END DO
11     IXT(CB)=NM+1
12   END DO
13   CALL MUDLIS(NCX, IXT)
14   DO CB=1, NCB
15     MK=0
16     NM=0
17     DO KB=IBT(CB), IBT(CB+1)-1
18       RB=JBT(KB)
19       CL=RB
20       CALL DEPTHFIRSTSEARCH(NCL, ILT, JLT, CL, NM, MK)
21     END DO
22     L=0
23     DO CL=1, NCL
24       IF (MK(CL).EQ.1) THEN
25         L=L+1
26         JXT(IXT(CB)-1+L)=CL
27       END IF
28     END DO
29     JXT(IXT(CB+1)-1)=CB+NCL
30   END DO
31 END SPARSESPARSELXBSYMB

```

Algorithm 11 Depth-first search for dependencies and update of degree-of-freedom marking for row I. NM is updated as the number of reachable degrees-of-freedom.

```

DEPTHFIRSTSEARCH(NRA, IA, JA, I, NM, MK)
  RA=I
  NSTACK=1
  STACK(NSTACK)=RA
  DO
    IF (NSTACK.EQ.0) EXIT
    RA=STACK(NSTACK)
    NSTACK=NSTACK-1
    IF (MK(RA).EQ.0) THEN
      NM=NM+1
      MK(RA)=1
      DO KA=IA(RA), IA(RA+1)-1
        CA=JA(KA)
        NSTACK=NSTACK+1
        STACK(NSTACK)=CA
      END DO
    END IF
  END DO
END DEPTHFIRSTSEARCH

```

Algorithm 12 Specific numeric sparse triangular solve with multiple sparse right-hand-sides $L^*X = -b$ with sparse result.

```

SPARSESPARSELXBNUM(NRL, ILT, JLT, VLT, IXT, JXT, VXT, NCB, IBT, JBT, VBT)
NRX=NRX
NCX=NCB
NRB=NRX
DO CB=1, NCB
  CX=CB
  DO K=IXT(CX), IXT(CX+1)-2
    RX=JXT(K)
    PLACE(RX)=K
  END DO
  DO K=IBT(CB), IBT(CB+1)-1
    RB=JBT(K)
    L=PLACE(RB)
    VXT(L)=VBT(K)/VLT(ILT(RB))
  END DO
  CX=CB
  DO KX=IXT(CX), IXT(CX+1)-2
    RX=JXT(KX)
    CL=RX
    DO KL=ILT(CL), ILT(CL+1)-1
      RL=JLT(KL)
      IF(RX.lt.RL) then
        L=PLACE(RL)
        VXT(L)=VXT(L)-VXT(KX)*VLT(KL)/VLT(ILT(RL))
      ENDIF
    END DO
  END DO
  VXT(IXT(CX+1)-1)=-1.0D00
END DO
END SPARSESPARSELXBNUM

```

Algorithm 13 Sparse triangular solves, $x = L^{-1}b$ or $x = L^{-T}b$ with dense result.

```

SPARSEDENSELXBCSC(NRL, IL, JL, VX, X, B)
X=B
DO RL=1, NRL
  X(RL)=X(RL)/VX(IL(RL))
  T=X(RL)
  DO KL=IL(RL)+1, IL(RL+1)-1
    X(JL(KL))=X(JL(KL))-T*VX(KL)
  END DO
END DO
END SPARSEDENSELXBCSC

SPARSEDENSEUXBCSR(NRU, IU, JU, VU, X, B)
X(NRU)=B(NRU)/VU(IU(NRU))
DO RU=NRU-1, 1, -1
  T=B(RU)
  DO KU=IU(RU)+1, IU(RU+1)-1
    T=T-VU(KU)*X(JU(KU))
  END DO
  X(RU)=T/VU(IU(RU))
END DO
END SPARSEDENSEUXBCSR

```

Algorithm 14 Sparse matrix/dense vector $w = Av$ and transposed version $w = A^T v$.

```

ATIMESV(NRA, IA, JA, VA, V, W)
  DO RA=1, NRA
    T=0.0D00
    DO KA=IA(RA), IA(RA+1)-1
      T=T+VA(KA)*V(JA(KA))
    END DO
    W(RA)=T
  END DO
END ATIMESV

ATTIMESV(NRA, IA, JA, VA, V, W)
  W=0
  DO RA=1, NRA
    DO KA=IA(RA), IA(RA+1)-1
      W(JA(KA))=W(JA(KA))+V(RA)*VA(KA)
    END DO
  END DO
END ATTIMESV

```

Algorithm 15 Symbolic assembling.

```

SYMASSEMB(NEL, IEDOF, JEDOF, ICLIQUE, JCLIQUE, NEQ, IDOFDOF, JDOFDOF)
  ...
  CALL SPARSETRANSINDEX(NEL, NEQ, IEDOF, JEDOF, IJLE, IDOFE, JDOFE)
  CALL CLQADDRESS(NEL, IEDOF, IEDOF, ICLIQUE)
  CALL ATIMESBSYMB(NEQ, IDOFE, JDOFE, NEL, IEDOF, JEDOF, IGASH, IDOFDOF)
  L=0
  DO IEQ=1, NEQ
    DO KDOFE=IDOFE(IEQ), IDOFE(IEQ+1)-1
      IEL=JDOFE(KDOFE)
      IGL=IJLE(KDOFE)
      JGL=0
      DO KEDOF=IEDOF(IEL), IEDOF(IEL+1)-1
        JGL=JGL+1
        IDOFDOFB=JEDOF(KEDOF)
        IP=IW(IDOFDOFB)
        IF(IP.EQ.0) THEN
          L=L+1
          JDOFDOF(L)=IDOFDOFB
          IW(IDOFDOFB)=L
          LLP=INDSTIFF(ICLIQUE, IEDOF, IEL, IGL, JGL)
          JCLIQUE(LLP)=L
        ELSE
          LLP=INDSTIFF(ICLIQUE, IEDOF, IEL, IGL, JGL)
          JCLIQUE(LLP)=IP
        END IF
      END DO
    END DO
  END DO
  DO IZC=IDOFDOF(IEQ), L
    IW(JDOFDOF(IZC))=0
  END DO
END DO
END SYMASSEMB

INDSTIFF(ICLIQUE, IEDOF, IEL, IGL, JGL)
  INDSTIFF=ICLIQUE(IEL)-1+ID2D(IEDOF(IEL+1)-IEDOF(IEL), IGL, JGL)
END FUNCTION INDSTIFF

INTEGER FUNCTION ID2D(M, I, J)
  ID2D=I+(J-1)*M
END FUNCTION ID2D

```

Algorithm 16 Indices for multiplication and clique addressing.

```

ATIMESBSYMB (NA, IA, JA, NB, IB, JB, NC, IC)
  NCB=NUMINJ (NB, IB, JB)
  NC=NA
  DO RA=1, NA
    LDG=0
    LLAST=-1
    DO KA=IA (RA), IA (RA+1) -1
      CA=JA (KA)
      DO KB=IB (CA), IB (CA+1) -1
        CB=JB (KB)
        IF (IW (CB).EQ.0) THEN
          LDG=LDG+1
          IW (CB)=LLAST
          LLAST=CB
        END IF
      END DO
    END DO
    IC (RA)=LDG
    DO K=1, LDG
      KA=IW (LLAST)
      IW (LLAST)=0
      LLAST=KA
    END DO
  END DO
  CALL MUDLIS (NC, IC)
END ATIMESBSYMB

```

```

CLQADDRESS (NEL, IEDOF, IEDOF2, ISTIF)
  DO IEL=1, NEL
    ITEMPO=IEDOF (IEL)
    ITEMP1=IEDOF (IEL+1)
    JTEMPO=IEDOF2 (IEL)
    JTEMP1=IEDOF2 (IEL+1)
    ISTIF (IEL)=(ITEMP1-ITEMPO)*(JTEMP1-JTEMPO)
  END DO
  CALL MUDLIS (NEL, ISTIF)
END CLQADDRESS

```

```

NUMINJ (NUM, IPO, LIS)
  NUMINJ=0
  DO I=1, IPO (NUM+1) -1
    NUMINJ=MAX (NUM2, LIS (I))
  END DO
END NUMINJ

```

Disclosure statement

No potential conflict of interest was reported by the authors.

References

- [1] T. Belytschko, W. K. Liu, and B. Moran, *Nonlinear Finite Elements for Continua and Structures*, Chichester: John Wiley & Sons, 2000.
- [2] P. Areias, T. Rabczuk, D. Dias da Costa, and E. B. Pires, "Implicit Solutions With Consistent Additive and Multiplicative Components," *Finite Elem. Anal. Des.*, vol. 57, pp. 15–31, 2012.
- [3] Wolfram Research Inc., *Mathematica*, Champaign, IL: Wolfram Research Inc., 2007.
- [4] J. Korelc, "Multi-Language and Multi-Environment Generation of Nonlinear Finite Element Codes," *Eng. Comput.*, vol. 18, no. 4, pp. 312–327, 2002.
- [5] A. Klarbring, *Models of Mechanics*, Dordrecht: Springer, 2006.
- [6] J. F. Abel, and M. S. Shephard, "An Algorithm for Multipoint Constraints in Finite Element Analysis," *Int. J. Numer. Methods Eng.*, vol. 14, no. 3, pp. 464–467, 1979.
- [7] M. S. Shephard, "Linear Multipoint Constraints Applied Via Transformation as Part of a Direct Stiffness Assembly Process," *Int. J. Numer. Methods Eng.*, vol. 20, no. 11, pp. 2107–2112, 1984.
- [8] M. Ainsworth, "Essential Boundary Conditions and Multi-Point Constraints in Finite Element Analysis," *Comput. Methods Appl. Mech. Eng.*, vol. 190, no. 48, pp. 6323–6339, 2001.
- [9] E. Chow, T. A. Manteuffel, C. Tong, and B. K. Wallin, "Algebraic Elimination of Slide Surface Constraints in Implicit Structural Analysis," *Int. J. Numer. Methods Eng.*, vol. 57, no. 8, pp. 1129–1144, 2003.
- [10] P. Saint-Georges, Y. Notay, and G. Warzée, "Efficient Iterative Solution of Constrained Finite Element Analyses," *Comput. Methods Appl. Mech. Eng.*, vol. 160, no. 1–2, pp. 101–114, 1998.
- [11] M. Benzi, G. H. Golub, and J. Liesen, "Numerical Solution of Saddle Point Problems," *Acta Numer.*, vol. 14, pp. 1–137, 2005.
- [12] P. Areias, T. Rabczuk, and J. I. Barbosa, "The Extended Unsymmetric Frontal Solution for Multiple-Point Constraints," *Eng. Comput.*, vol. 31, no. 7, pp. 1582–1607, 2014.
- [13] N. Ian, and M. Gould, "On Modified Factorizations for Large-Scale Linearly Constrained Optimization," *SIAM J. Opt.*, vol. 9, no. 4, pp. 1041–1063, 1999.
- [14] S. S. Antman, and R. S. Marlow, "Material Constraints, Lagrange Multipliers, and Compatibility," *Arch. Rational Mech. Anal.*, vol. 116, no. 3, pp. 257–299, 1991.
- [15] S. S. Antman, *Nonlinear Problems of Elasticity*, 2nd ed., New York: Springer, 2005.
- [16] P. Areias, and K. Matous, "Finite Element Formulation for Modeling Nonlinear Viscoelastic Elastomers," *Comput. Methods Appl. Mech. Eng.*, vol. 197, no. 51–52, pp. 4702–4717, 2008.
- [17] F. Amirouche, *Fundamentals of Multibody Dynamics Theory and Applications*, Switzerland: Birkhäuser, 2006.
- [18] P. E. Nikravesh, *Computer-Aided Analysis of Mechanical Systems*, Upper Saddle River, NJ: Prentice Hall, 1988.
- [19] I. S. Duff, A. M. Erisman, and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford: Clarendon Press, 1986.
- [20] N. I. M. Gould, M. E. Hribar, and J. Nocedal, "On the Solution of Equality Constrained Quadratic Programming Problems Arising in Optimization," *SIAM J. Sci. Comput.*, vol. 23, no. 4, pp. 1376–1395, 2001.
- [21] T. Rees, and J. Scott, "A Comparative Study of Null-Space Factorizations for Sparse Symmetric Saddle Point Systems," *Numer. Linear Algebra Appl.*, vol. 25, pp. e2103, 2017.
- [22] R. Fletcher, and T. Johnson, "On the Stability of Null-Space Methods for KKT Systems," *SIAM J. Matrix Anal. Appl.*, vol. 18, no. 4, pp. 938–958, 1997.
- [23] J. R. Gilbert, and M. T. Heath, "Computing a Sparse Basis for the Null Space," *SIAM J. Alg. Disc. Meth.*, vol. 8, no. 3, pp. 446–459, 1987.
- [24] C. Gotsman, and S. Toledo, "On the Computation of Null Spaces of Sparse Rectangular Matrices," *SIAM J. Matrix Anal. Appl.*, vol. 30, no. 2, pp. 445–463, 2008.
- [25] T. A. Davis, *Direct Methods for Sparse Linear Systems*, Philadelphia, PA: SIAM, 2006.
- [26] S. G.L.G. Sleijpen and D. R. Fokema, "BICGSTAB(l) for Linear Equations Involving Unsymmetric Matrices With Complex Spectrum," *Electron. Trans. Numer. Anal.*, vol. 1, pp. 11–32, 1993.
- [27] P. Areias, "Sparse acyclic library." Available: <https://github.com/PedroAreias/sparseacyclic/>, Accessed: Jan. 21, 2019.
- [28] D. Jungnickel, *Graphs, Networks and Algorithms*, Volume 5 of *Algorithms and Computation in Mathematics*, 2nd ed., Berlin: Springer, 2005.
- [29] F. G. Gustavson, "Two Fast Algorithms for Sparse Matrices: Multiplication and Permuted Transposition," *ACM Trans. Math. Softw.*, vol. 4, no. 3, pp. 250–269, 1978.
- [30] C. Kane, J. E. Marsden, M. Ortiz, and M. West, "Variational Integrators and the Newmark Algorithm for Conservative and Dissipative Mechanical Systems," *Int. J. Numer. Methods Eng.*, vol. 49, no. 10, pp. 1295–1325, 2000.
- [31] M. Hobiger, I. Sonder, R. Büttner, and B. Zimanowski, "Viscosity Characteristics of Selected Volcanic Rock Melts," *J. Volcanol. Geother. Res.*, vol. 200, no. 1–2, pp. 27–34, 2011.
- [32] J. C. Simo, and T. J. R. Hughes, *Computational Inelasticity*, corrected second printing edition, Berlin: Springer, 2000.
- [33] F. J. Spera, Physical Properties of Magma, in *Encyclopedia of Volcanoes*, H. Sigurdsson, B.F. Houghton, S.R. McNutt, H. Rymer, and J. Stix, (Eds.). Cambridge, MA: Academic Press, 1999, pp. 171–190.
- [34] M. O. Chevrel, T. Platz, E. Hauber, D. Baratoux, Y. Lavallée, and D. B. Dingwell, "Lava Flow Rheology:

- A Comparison of Morphological and Petrological Methods,” *Earth Planet. Sci. Lett.*, vol. 384, pp. 109–120, 2013.
- [35] E. Rivalta, and P. Segall, “Magma Compressibility and the Missing Source for Some Dike Intrusions,” *Geophys. Res. Lett.*, vol. 35, no. L04306, pp. 1–5, 2008.
- [36] A. Piombo, and M. Dragoni, “Evaluation of Flow Rate for a One-Dimensional Lava Flow With Power-Law Rheology,” *Geophys. Res. Lett.*, vol. 36, no. 22, pp. L22306, 2009.
- [37] E. H. Abramson, J. M. Brown, L. J. Slutsky, and J. Zaug, “The Elastic Constants of San Carlos Olivine to 17 GPa,” *J. Geophys. Res.*, vol. 102, no. B6, pp. 12253–12263, 1997.