Assessing the potential of Jacobian-free Newton-Krylov methods for cell-centred finite volume solid mechanics

Philip Cardiff[1,2,3*] and Ivan Batistić[4]

[1*]School of Mechanical and Materials Engineering, University College Dublin, Ireland.

[2]UCD Centre for Mechanics, University College Dublin, Ireland.

[3]SFI I-Form Centre, University College Dublin, Ireland.

[4]University of Zagreb, University of Zagreb, Croatia.

*Corresponding author. E-mail: philip.cardiff@ucd.ie

Abstract

In this study, we explore the efficacy of Jacobian-free Newton-Krylov methods within the context of finite-volume solid mechanics. Traditional Newton-based approaches to solving nonlinear systems typically require explicit formation and storage of the Jacobian matrix, which can be computationally expensive and memory-intensive. The Jacobian-free Newton-Krylov method circumvents this by employing Krylov subspace iterative solvers, such as GMRES, in conjunction with a Newton iteration scheme that approximates the action of the Jacobian through finite difference evaluations. A further potential advantage of the Jacobian-free Newton-Krylov method is that it is readily applicable to existing segregated finite volume frameworks, where forming and storing the exact Jacobian would require major code refactoring. This article research systematically evaluates the performance of Jacobian-free Newton-Krylov methods by benchmarking them against conventional segregated methods on a suite of benchmark cases of varying geometric dimension, geometric nonlinearity, dynamic response, and material behaviour. Key metrics such as computational cost, memory and robustness are analysed. Additionally, we investigate the impact of various solution algorithm choices, such as preconditioning strategy, on the efficiency of the Jacobian-free Newton-Krylov method. Our findings indicate that Jacobian-free Newton-Krylov methods can achieve comparable/superior/XXX convergence behaviour relative to traditional segregated methods, particularly in cases where YYYY. Summarise key findings: time, memory, important choices, JFNK vs SEG vs FE, ... The results suggest that Jacobian-free Newton-Krylov methods are promising for advancing finite-volume solid mechanics simulations and are particularly attractive for existing segregated frameworks where minimal code changes would be required to exploit openly available Jacobian-free Newton-Krylov implementations. The described implementations are made publicly available in the solids4foam toolbox for OpenFOAM, allowing the community to examine, extend and compare the procedures with the our codes.

Keywords: Jacobian-free Newton-Krylov, Finite volume method, GMRES, OpenFOAM

1 Introduction

Finite volume formulations for solid mechanics are heavily influenced by their fluid mechanics counterparts, favouring segregated implicit and fully explicit methods. Segregated approaches, where the governing momentum equation is temporarily decomposed into scalar component equations, offer memory efficiency and simplicity of implementation, but the outer coupling Picard iterations often suffer from slow convergence. Explicit formulations are straightforward to implement and offer superior robustness but are only efficient for high-speed dynamics, where the physics requires small time increments. In contrast, the finite element community commonly employs Newton-Raphson-type solution algorithms, which necessitate repeated assembly of the Jacobian matrix and solution of the resulting block-coupled non-diagonally dominant linear system. A disadvantage of traditional Newton-based approaches is that they typically require explicit formation and storage of the Jacobian matrix, which can be computationally expensive and memory-intensive. A further disadvantage from a finite volume perspective is that extending existing code frameworks from segregated algorithms to a coupled Newton-Raphson-type approach is challenging in terms of the required assembly, storage, and solution of the resulting block-coupled system. In addition, the derivation of the true Jacobian matrix is non-trivial. Consequently, similar block-coupled solution finite volume methods are rare in the literature [**? ? ?**]. The motivation of the current work is to seek (or exceed) the robustness and efficiency of block-coupled Newton-Raphons approaches in a way that can be easily incorporated into existing segregated solution frameworks. To this end, the current article examines the efficacy of *Jacobian-free* Newton-Krylov methods, where the quadratic convergence of Newton methods can potentially be achieved without deriving, assembling and storing the exact Jacobian.

Jacobian-free Newton-Krylov methods circumvent the need for the Jacobian matrix by combining the Newton-Raphson method with Krylov subspace iterative linear solvers, such as GMRES, and noticing that such Krylov solvers do not explicitly require the Jacobian matrix. Instead, only the action of the Jacobian matrix on a solution-type vector is required. The key step in Jacobian-free Newton-Krylov methods is the approximation of products between the Jacobian matrix and a vector using the finite difference method; that is

$$\boldsymbol{J}\boldsymbol{v} \approx \frac{\boldsymbol{F}(\boldsymbol{x} + \epsilon\boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{x})}{\epsilon} \tag{1}$$

where $\mathbf{J}$ is the Jacobian matrix, $\mathbf{x}$ is the current solution vector (e.g. nodal displacements), $\mathbf{v}$ is a vector (e.g., from a Krylov subspace), and ϵ is a small scalar perturbation. With an appropriate choice of ϵ (balancing truncation and round-off errors), the characteristic quadratic convergence of Newton methods can be achieved without the Jacobian, hence the modifier *Jacobian-free*. This approach promises significant memory savings over Jacobian-based methods, especially for large-scale, but also potentially for execution time, with appropriate choice of solution components.

A crucial aspect of ensuring the efficiency and robustness of the Jacobian-free Newton-Krylov method is the choice of a suitable preconditioner for the Krylov iterations. This preconditioner is often derived from the exact Jacobian matrix in traditional Newton methods. However, the Jacobian-free approach does not allow direct access to the full Jacobian matrix, necessitating an alternative strategy to approximate its action. To this end, and to extend existing segregated frameworks, we propose using a compact-stencil approximate Jacobian as the preconditioner. This approximate Jacobian corresponds to the matrix typically employed in segregated approaches; similar approaches are successful in fluid mechanics applications [**?** **?**]; however, it is unclear if such an approach is suitable for solid mechanics - a question which we hope to answer in this work. By leveraging this compact-stencil approximate Jacobian, we aim to effectively precondition the Krylov iterations, enhancing convergence while maintaining the memory and computational savings that define the Jacobian-free and segregated methods. Similarly, if such an approach is efficient, it would naturally fit into existing segregated frameworks, as existing matrix storage and assembly can be reused.

The remainder of the paper is structured as follows: Section 2 summarises a typical solid mechanics mathematical model and its cell-centred finite volume discretisation. Section 3 presents the solution algorithms, starting with the classic segregated solution algorithm, followed by the proposed Jacobian-free Newton-Krylov solution algorithm. The performance of the proposed Jacobian-free Newton-Krylov approach is compared with the segregated approach on several varying benchmark cases in Section 4, where the effect of several factors are examined, including problem dimension, mesh, material model, nonlinear geometry, choice of preconditioner, and other solution parameter. Finally, the article ends with a summary of the main conclusions of the work.

2 Mathematical Model and Numerical Methods

2.1 Governing Equations

In this work, we restrict our interest to Lagrangian formulations of the conservation of linear momentum. Assuming small strains, the linear geometry formulation is expressed in strong integral form as:

$$\int_{\Omega} \rho \frac{\partial^2 \boldsymbol{u}}{\partial t^2} \, d\Omega = \oint_{\Gamma} \boldsymbol{n} \cdot \boldsymbol{\sigma} \, d\Gamma + \int_{\Omega} \rho \boldsymbol{g} \, d\Omega \tag{2}$$

where Ω is the volume of an arbitrary body bounded by a surface Γ with outwards pointing normal $\boldsymbol{n}$. The density is ρ, $\boldsymbol{u}$ is the displacement vector, $\boldsymbol{\sigma}_s$ is the engineering (small strain) stress tensor, and $\boldsymbol{g}$ is a body force per unit mass, e.g., gravity.

More generally, linear momentum conservation can be expressed in a nonlinear geometry form, which is suitable for finite strains. Two equivalent nonlinear geometry forms are common: the *total* Lagrangian form:

$$\int_{\Omega_o} \rho_o \frac{\partial^2 \boldsymbol{u}}{\partial t^2} d\Omega_o = \oint_{\Gamma_o} \left(J \boldsymbol{F}^{-T} \cdot \boldsymbol{n}_o \right) \cdot \boldsymbol{\sigma} \, d\Gamma_o + \int_{\Omega_o} \rho_o \boldsymbol{g} \, d\Omega_o \tag{3}$$

and the *updated* Lagrangian form:

$$\int_{\Omega_u} \frac{\partial}{\partial t}\left(\rho_u \frac{\partial \boldsymbol{u}}{\partial t}\right) d\Omega_u = \oint_{\Gamma_u} (j\boldsymbol{f}^{-T} \cdot \boldsymbol{n}_u) \cdot \boldsymbol{\sigma}\ d\Gamma_u + \int_{\Omega_u} \rho_u \boldsymbol{g}\, d\Omega_u \tag{4}$$

where subscript o indicates quantities in the initial reference configuration, and subscript u indicates quantities in the updated configuration. The true (Cauchy) stress tensor is indicated by $\boldsymbol{\sigma}$.

The deformation gradient is defined as $\boldsymbol{F} = \mathbf{I} + (\boldsymbol{\nabla u})^T$ and its determinant as $J = \det(\boldsymbol{F})$. Similarly, the *relative* deformation gradient is given in terms of the displacement *increment* as $\boldsymbol{f} = \mathbf{I} + [\boldsymbol{\nabla}(\Delta \boldsymbol{u})]^T$ and its determinant as $j = \det(\boldsymbol{f})$. The displacement increment is the change in displacement between the current time step and the previous time step when the time interval is discretised into a finite number of steps.

The definition of the engineering stress ($\boldsymbol{\sigma}_s$) and true stress ($\boldsymbol{\sigma}$) in Equations 2, 3 and 4 is given by a chosen mechanical law, e.g. linear elasticity. Several mechanical laws are considered in this work, as briefly described in Section 4.

2.2 Newton-Type Solution Methods

To facilitate the comparison between classic segregated solution algorithms and the proposed Jacobian-free Newton-Krylov algorithm, the governing linear momentum conservation (Equations 2, 3 and 4) is expressed in the general form:

$$\mathcal{R}(\boldsymbol{u}) = \mathbf{0} \tag{5}$$

where $\mathcal{R}$ represents the *residual* (imbalance) of the equation, which is a function of the primary unknown field. For example, in the linear geometry case, the residual is given as

$$\boldsymbol{R}(\boldsymbol{u}) \ = \ \oint_{\Gamma} \boldsymbol{n} \cdot \boldsymbol{\sigma}(\boldsymbol{u})\, d\Gamma + \int_{\Omega} \rho \boldsymbol{g}\, d\Omega - \int_{\Omega} \rho \frac{\partial^2 \boldsymbol{u}}{\partial t^2}\, d\Omega \ = \ \mathbf{0} \tag{6}$$

where the dependence of the stress tensor on the solution vector is made explicitly clear: $\boldsymbol{\sigma}(\boldsymbol{u})$.

In Newton-type methods, a Taylor expansion about a current point $\boldsymbol{u}_k$ can be used to solve Equation 5 [**?**]:

$$\boldsymbol{R}(\boldsymbol{u}_{k+1}) = \boldsymbol{R}(\boldsymbol{u}_k) \ + \ \boldsymbol{R}'(\boldsymbol{u}_k)(\boldsymbol{u}_{k+1} - \boldsymbol{u}_k) \ + \ \text{H.O.T.} = \mathbf{0} \tag{7}$$

Neglecting the higher-order terms (H.O.T.) yields the strict Newton method in terms of an iteration over a sequence of linear systems:

$$\begin{aligned} \boldsymbol{J}(\boldsymbol{u}_k)\delta\boldsymbol{u} &= -\boldsymbol{R}(\boldsymbol{u}_n), \\ \boldsymbol{u}_{k+1} &= \boldsymbol{u}_k + s\,\delta\boldsymbol{u}, \\ k &= 0, 1, ... \end{aligned} \tag{8}$$

where $\boldsymbol{J} \equiv \boldsymbol{R}'$ is the Jacobian matrix. Starting the Newton procedure requires the specification of $\boldsymbol{u}_0$. The scalar $s > 0$ can be chosen to improve convergence, for example, using a line search or under-relaxation procedure, and is equal to unity in the classic Newton-Raphson approach. Iterations are performed over this system until the residual $\boldsymbol{R}(\boldsymbol{u}_n)$ and solution correction $\delta\boldsymbol{u}$ are sufficiently small, with appropriate normalisation.

For problems with N scalar equations and N scalar unknowns, the residual $\boldsymbol{R}$ and solution $\boldsymbol{u}$ vectors have dimensions of $N \times 1$. The components of the $N \times N$ Jacobian are

$$J_{ij} = \frac{\partial R_i(\boldsymbol{u})}{\partial u_j} \tag{9}$$

In the current work, we are interested in vector problems, where the governing momentum equation is formulated in terms of the unknown displacement solution vector. In this case, Equation 9 refers to the individual scalar components of the residual, solution, and Jacobian. That is, for 3-D analyses, the residual takes the form

$$\boldsymbol{R}(\boldsymbol{u}) = \{R_1^x, R_1^y, R_1^z, R_2^x, R_2^y, R_2^z, ..., R_n^z\} \tag{10}$$

and the solution takes the form

$$\boldsymbol{u} = \{u_1^x, u_1^y, u_1^z, u_2^x, u_2^y, u_2^z, ..., u_n^z\} \tag{11}$$

In practice, it is often more practical and efficient to form and store the residual, solution and Jacobian in a *blocked* manner, where the residual and solution can be considered as vectors of vectors. Similarly, the Jacobian can be formed in terms of sub-matrix block coefficients.

In the strict Newton procedure, the residuals converge at a quadratic rate when the current solution is close to the true solution; that is, the iteration error decreases proportionally to the square of the error at the previous iteration. Once the method gets sufficiently close to the true solution, the number of correct digits in the approximation roughly doubles with each iteration. However, quadratic convergence is only possible when using the exact Jacobian. In contrast, a quasi-Newton method uses an approximation to the Jacobian, sacrificing strict quadratic convergence in an attempt to produce an overall more computationally efficient procedure. From this perspective, the segregated solution algorithm commonly employed in finite volume solid mechanics can be viewed as a quasi-Newton method, where an approximate Jacobian replaces the exact Jacobian:

$$\tilde{\boldsymbol{J}}(\boldsymbol{u}_k)\ \delta\boldsymbol{u} = -\boldsymbol{R}(\boldsymbol{u}_k) \tag{12}$$

In this case, the approximate Jacobian $\tilde{\boldsymbol{J}}$ comes from the compact stencil discretisation of a simple diffusion (Laplacian) term. A benefit of this approach is that the inter-component coupling is removed from the Jacobian, allowing the solution of three smaller scalar systems rather than one larger vector system in 3-D (or two smaller systems in 2-D).

A fully explicit procedure can also be viewed from this perspective by selecting an approximate Jacobian which is diagonal $\tilde{\boldsymbol{D}}$, making solution of the linear system trivial:

$$\tilde{\boldsymbol{D}}(\boldsymbol{u}_k)\ \delta\boldsymbol{u} = -\boldsymbol{R}(\boldsymbol{u}_k) \tag{13}$$

2.3 Cell-Centred Finite Volume Discretisation

In this work, a nominally second-order cell-centred finite volume discretisation is employed, as described previously, for example, [**? ? ? ?**]. Consequently, only a summary of the discretisation is presented below.

The solution domain is discretised in both space and time. The total simulation period is divided into a finite number of time increments, denoted as Δt, and the discretised governing momentum equation is solved iteratively in a time-marching fashion. The spatial domain is partitioned into a finite number of contiguous convex polyhedral cells.

The conservation equation (Equations **??**, **??**, or **??**) is applied to each cell (control volume) in the computational mesh and discretised in terms of the displacement at the cell centre/centroid $\boldsymbol{u}_P$ and at the centres of the neighbouring cells N_i.

To complete the discretisation, the volume integrals and surface integrals in the governing equation must be approximated by algebraic equations. Starting first with the volume integrals, assuming a linear variation of the integrand, the mid-point rule approximates the integral in terms of the cell centre value. Consequently, the inertia term (e.g. left-hand side term of Equation 2) becomes

$$\int_\Omega \rho \frac{\partial \boldsymbol{u}}{\partial t} d\Omega \ \approx\ \rho_P \left(\frac{\partial^2 \boldsymbol{u}}{\partial t^2}\right)_P \Omega_P \tag{14}$$

and, similarly, the body force term (e.g. the second term on the right-hand side of Equation 2) becomes:

$$\int_\Omega \rho\, \boldsymbol{g}\, d\Omega \ \approx\ \rho_P\, \boldsymbol{g}\, \Omega_P \tag{15}$$

where subscript P indicates a quantity at the cell centre. The discretisation of the acceleration in time in Equation 14 can be achieved using the finite difference method, e.g. first-order Euler, second-order backwards, second-order Newmark-beta. Maybe we should give the temporal discretisation for completeness: i.e. 2nd order backwards

The surface integral term (e.g. first term on the right-hand side of Equation 2), corresponding to the divergence of stress, is discretised by assuming that the stress varies linearly across the face, allowing the mid-point rule to be used:

$$\oint_\Gamma \boldsymbol{n} \cdot \boldsymbol{\sigma}\ d\Gamma \approx \sum_{f \in N_f} \boldsymbol{\Gamma}_f \cdot \boldsymbol{\sigma}_f \tag{16}$$

where subscript f indicates a quantity at the centre of a cell face, and N_f represents the set of neighbouring cells which share a face with cell P. The stress at a face, $\boldsymbol{\sigma}_f$, is calculated by linearly interpolating from the adjacent cell centres. Stress is calculated at the cell centres as a function of the displacement gradient, $(\boldsymbol{\nabla u})_f$, and the cell-centre gradients are determined using a least squares method [**?**].

The discretisation is complete but, in its current form, is known to suffer from zero-energy modes, i.e. checkerboarding oscillations. Here, a Rhie-Chow-type stabilisation term [**?**] is added to the residual (Equation 5) to quell such oscillations. The Rhie-Chow stabilisation term, first used for finite volume solid mechanics by **?**], consists of the numerical difference between a diffusion (Laplacian) term calculated using compact and larger computational stencils. The term introduces numerical diffusion to the discretisation, which reduces at a third-order rate. In the current approach, the Rhie-Chow stabilisation term $\mathcal{D}_{\text{Rhie-Chow}}$ for a cell P takes the following form:

$$\mathcal{D}_{\text{Rhie-Chow}} = \sum_{f \in N_f} \alpha \bar{K}_f \left[|\boldsymbol{\Delta}_f| \frac{\boldsymbol{u}_{N_f} - \boldsymbol{u}_P}{|\boldsymbol{d}_f|} - \boldsymbol{\Delta}_f \cdot (\boldsymbol{\nabla u})_f \right] |\boldsymbol{\Gamma}_f| \tag{17}$$

where $\alpha > 0$ is a user-defined parameter for globally scaling the amount of stabilisation. Parameter $\bar{K}_f$ is a stiffness-type parameter that gives the stabilisation an appropriate scale and dimension. Here, $\bar{K}_f = \frac{4}{3}\mu + \kappa = 2\mu + \lambda$ following previous work [**?** **?** **?**], where μ is the shear modulus (first Lamé parameter), κ is the bulk modulus, and λ is the second Lamé parameter. Vector $\boldsymbol{d}_f$ connects cell centre P with the other cell sharing face f, and $\boldsymbol{n}_f$ is the outward-facing unit normal to the face f. The vector $\boldsymbol{\Delta}_f = \frac{\boldsymbol{d}_f}{\boldsymbol{d}_f \cdot \boldsymbol{n}_f}$ is termed the *over-relaxed orthogonal* vector [**?**] and increases in magnitude as the deviation between the $\boldsymbol{d}_f$ and $\boldsymbol{n}_f$ vectors increases. In this way, the amount of stabilisation increases on distorted meshes. Should we mention Nishikawa alpha scheme? Very similar: but scales differently with mesh distortion

In Equation 17, the first term within the brackets on the right-hand side represents a compact stencil (two-node) approximation of the face normal gradient, while the second term represents a larger stencil approximation. These two terms cancel out in the limit of mesh refinement (or if the solution varies linearly); otherwise, they produce a stabilisation effect that tends to smooth the solution fields. As the term reduces at a third-order rate, it does not affect the overall scheme's second-order accuracy.

All dependent variables must be specified at the initial time. Boundary conditions must be applied to the faces that coincide with the boundary of the solution domain. The discretised expressions on boundary faces are modified to account for either the known displacement components in Dirichlet conditions or the known traction for Neumann conditions.

comment on traction boundaries extrapolate to get value or use constitutive law

3 Solution Algorithms

3.1 Segregated Solution Algorithm

The classic segregated solution algorithm can be viewed as a quasi-Newton method, where a compact-stencil approximation of a diffusion term is employed as the approximate Jacobian:

$$\begin{aligned}\tilde{\boldsymbol{J}} &= \oint_{\Gamma} \bar{K}\, \boldsymbol{n} \cdot \boldsymbol{\nabla} \boldsymbol{u}\, d\Gamma \\ &\approx \sum_{f \in N_f} \bar{K}_f\, |\boldsymbol{\Delta}_f| \left(\frac{\boldsymbol{u}_{N_f} - \boldsymbol{u}_P}{|\boldsymbol{d}_f|} \right) |\boldsymbol{\Gamma}_f| \end{aligned} \tag{18}$$

When a diffusion term is typically discretised using the cell-centre finite volume method, non-orthogonal corrections are included in a deferred correction manner to preserve the order of accuracy on distorted grids. However, in the Newton method case, the approximate Jacobian's exact value does not affect the final converged solution, but only the convergence behaviour. Consequently, non-orthogonal corrections are not included in the approximate Jacobian here. However, grid distortion is appropriately accounted for in the calculation of the residual. Nonetheless, as a result, it is expected that the convergence behaviour of the segregated approach may degrade as mesh non-orthogonality increases.

The linearised system (Equation 12) is formed for each cell in the domain, resulting in a system of algebraic equations:

$$\tilde{\boldsymbol{J}}(\boldsymbol{u}_n)\, \delta \boldsymbol{u} = -\boldsymbol{R}(\boldsymbol{u}_n) \tag{19}$$

where $\tilde{\boldsymbol{J}}$ is a symmetric, weakly diagonally dominant, $M \times M$ stiffness matrix, where M is three times the number of cells in 3-D and twice the number of cells in 2-D. By design, matrix $\tilde{\boldsymbol{J}}$ contains no inter-component coupling; consequently, three equivalent smaller linear systems can be formed and solved for the Cartesian components of the displacement correction (or two in 2-D), e.g.

$$\tilde{\boldsymbol{J}}_x(\boldsymbol{u}_n)\, \Delta \boldsymbol{u}_x = -\mathcal{R}_x(\boldsymbol{u}_n) \tag{20}$$

$$\tilde{\boldsymbol{J}}_y(\boldsymbol{u}_n)\, \Delta \boldsymbol{u}_y = -\mathcal{R}_y(\boldsymbol{u}_n) \tag{21}$$

$$\tilde{\boldsymbol{J}}_z(\boldsymbol{u}_n)\, \Delta \boldsymbol{u}_z = -\mathcal{R}_z(\boldsymbol{u}_n) \tag{22}$$

where $\bullet_x$ represents the components in the x direction, $\bullet_y$ represents the components in the y direction, and $\bullet_z$ represents the components in the z direction. An additional benefit of the segregated approach, from a memory perspective, is that matrices $\tilde{\boldsymbol{J}}_x$, $\tilde{\boldsymbol{J}}_y$ and $\tilde{\boldsymbol{J}}_z$ are identical, except for the effects from including boundary conditions. From an implementation perspective, this allows a single scalar matrix to be formed and stored, where the boundary condition contributions are inserted before solving a particular component.

The *inner* linear sparse systems (Equations 20, 21 and 22) can be solved using any typical direct or iterative linear solver approach; however, an incomplete Cholesky pre-conditioned conjugate

gradient method [**?**] is often preferred as the weakly diagonally dominant characteristic leads to good convergence characteristics. Algebraic multigrid can be used to accelerate convergence.

In literature, the segregated solution algorithm is typically formulated in terms of the total displacement vector (or its difference between time steps) as the primary unknown; in contrast, in the quasi-Newton interpretation presented here, the primary unknown is the correction to the displacement vector, which goes to zero at convergence. Nonetheless, both approaches are equivalent and neither formulation displays superior performance.

The current procedure is implemented and publicly shared in the solids4foam toolbox of OpenFOAM. Add a section about code sharing: appendix?

Comment: we have two implementations of segregated: native OpenFOAM (solves Eqs 16-18) and PETSc SNES (solves Eq 15) Do we need to comment on this? Maybe we should use only PETSc SNES for a fair comparison Or we could use both on the verification case and then stick with just one afterwards

3.2 Jacobian-free Newton-Krylov Algorithm

As noted in the introduction, the Jacobian-free Newton-Krylov avoids the need to construct the Jacobian matrix explicitly by approximating its action on a solution vector using the finite difference method, repeated here:

$$\boldsymbol{J}\boldsymbol{v} \approx \frac{\boldsymbol{F}(\boldsymbol{x} + \epsilon\boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{x})}{\epsilon} \tag{23}$$

The derivation of this approximation can be shown for a 2×2 system as [**?**]:

$$\begin{aligned}
\frac{\mathbf{F}(\mathbf{x} + \epsilon\mathbf{v}) - \mathbf{F}(\mathbf{x})}{\epsilon} &= \begin{pmatrix} \frac{F_1(x_1+\epsilon v_1, x_2+\epsilon v_2) - F_1(x_1,x_2)}{\epsilon} \\ \frac{F_2(x_1+\epsilon v_1, x_2+\epsilon v_2) - F_2(x_1,x_2)}{\epsilon} \end{pmatrix} \\
&\approx \begin{pmatrix} \frac{F_1(x_1,x_2) + \epsilon v_1 \frac{\partial F_1}{\partial u_1} + \epsilon v_2 \frac{\partial F_1}{\partial u_2} - F_1(x_1,x_2)}{\epsilon} \\ \frac{F_2(x_1,x_2) + \epsilon v_1 \frac{\partial F_2}{\partial u_1} + \epsilon v_2 \frac{\partial F_2}{\partial u_2} - F_2(x_1,x_2)}{\epsilon} \end{pmatrix} \\
&\approx \begin{pmatrix} v_1 \frac{\partial F_1}{\partial u_1} + v_2 \frac{\partial F_1}{\partial u_2} \\ v_1 \frac{\partial F_2}{\partial u_1} + v_2 \frac{\partial F_2}{\partial u_2} \end{pmatrix} \\
&\approx \boldsymbol{J}\boldsymbol{v}
\end{aligned} \tag{24}$$

where a first-order truncated Taylor series expansion about $\boldsymbol{u}$ was used to approximate $\boldsymbol{F}(\boldsymbol{x} + \epsilon\boldsymbol{v})$. As noted above, choosing an appropriate value for ϵ is non-trivial, and care must be taken to balance truncation error (reduced by decreasing ϵ) and round-off error (increased by decreasing ϵ).

The purpose of preconditioning the Jacobian-free Newton-Krylov method is to reduce the number of inner linear solver iterations. In the current work, the GMRES linear solver is used for the inner system. Using right preconditioning, the finite difference approximation of Equation 23

becomes

$$\boldsymbol{J}\boldsymbol{P}^{-1}\boldsymbol{v} \approx \frac{\boldsymbol{F}(\boldsymbol{x} + \epsilon\boldsymbol{P}^{-1}\boldsymbol{v}) - \boldsymbol{F}(\boldsymbol{x})}{\epsilon} \tag{25}$$

where $\boldsymbol{P}$ is the preconditioning matrix or process. In practice, only the action of $\boldsymbol{P}^{-1}$ on a vector is required, and the $\boldsymbol{P}^{-1}$ may not be explicitly formed. Concretely, the preconditioner needs to approximately solve the linear system $\boldsymbol{y} = \boldsymbol{P}^{-1}\boldsymbol{v}$.

In the current work, we proposed to use the compact-stencil approximate Jacobian from the segregated algorithm $\tilde{\boldsymbol{J}}$ as the preconditioning matrix $\boldsymbol{P}$ for the preconditioned Jacobian-free Newton-Krylov method. Comment on literature: used before but not yet for solids. This preconditioning approach can be considered as a "physics-based" preconditioner in the classifications of [**?**]. The approach is conceptually similar to an approximation of the Jacobian of a higher-order advection scheme by a compact-stencil lower-order upwind scheme. A benefit of the proposed approach is that existing segregated frameworks can re-use their existing discretisation and storage implementations. Concretely, the Jacobian-free Newton-Krylov method requires only a procedure for forming this preconditioning matrix and a procedure for explicitly evaluating the residual. Both routines are easily implemented in an existing segregated framework. The only additional required procedure is an interface to an existing Jacobian-free Newton-Krylov implementation. In the current work, the PETSc toolbox [**?**] is used as the nonlinear solver, where driven by a finite volume solver in the OpenFOAM toolbox [**?**].

Several preconditioners are available in the literature, incomplete Cholesky/LU being popular; however, multigrid methods offer the greatest potential for large-scale problems. **?**] noted that algorithmic simplifications within a multigrid procedure, which may result in loss of convergence for multigrid as a solver, have a much weaker effect when multigrid is the preconditioner. In this work, three preconditioners are considered:

1. ILU(N): incomplete LU with fill-in N. The segregated solver uses ILU(0), that is, ILU with zero fill-in.
2. Multigrid: here, we use the HYPRE Boomerang multigrid implementation.
3. LU: for comparison, we consider a direct LU decomposition solver as the preconditioner.

A challenge with Newton-type methods, including Jacobian-free versions, is convergence can be poor when far from the true solution, and divergence is often a real possibility. Globalisation refers to steering an initial solution towards the quadratic convergence range of the Newton method. Several strategies are possible, and it is common to combine approaches [**?**]. In the current work, a line search procedure is used to select the α parameter in the solution update step (the second line in Equations 8). Line search methods assume the Newton update direction is correction and aim to find a scalar $s > 0$ that decreases the residual $\boldsymbol{R}(\boldsymbol{u}_k + s\delta\boldsymbol{u}) < \boldsymbol{R}(\boldsymbol{u}_k)$. The scalar s is typically ≤ 1, but extrapolation (> 1) is also possible for accelerating convergence, albeit at the expense of robustness.

In addition to a line search approach, a *transient continuation* globalisation approach is used in the current work, where a prediction for the solution (displacement) field at time $t+\Delta t$ is performed

at the start of a new time step, based on a truncated second-order Taylor series expansion:

$$\boldsymbol{u}_{t+\Delta t} = \boldsymbol{u}_t + \Delta t \left(\frac{\partial \boldsymbol{u}}{\partial t}\right)_t + \frac{1}{2}\Delta t^2 \left(\frac{\partial^2 \boldsymbol{u}}{\partial t^2}\right)_t \tag{26}$$

where Δt is the time increment (assumed constant here), $\left(\frac{\partial \boldsymbol{u}}{\partial t}\right)_t$ is the velocity at time t, and $\left(\frac{\partial^2 \boldsymbol{u}}{\partial t^2}\right)_t$ is the acceleration at time t. In this way, for highly nonlinear problems, the user can decrease the time step size Δt as a globalisation approach to improve the performance of the Newton method. The predictor step in Equation 26 has been chosen to be consistent with the assumed discretisation of the temporal term in the governing equation; that is, the second order backwards scheme is assumed.

A final comment on the Jacobian-free Newton-Krylov solution algorithm is the potential importance of *oversolving*. Here, oversolving refers to solving the linear system to too tight a tolerance during the early Newton iterations, essentially wasting time when the solution is far from the true solution. In addition, some authors [**?**] have shown Newton convergence to be worse when the earlier iterations are solved to too tight a tolerance. The concept of oversolving also applies to segregated solution procedures and has been well-known since the early work of Demirdžić and co-workers [**?**], where the residuals are typically reduced by one order of magnitude in the inner linear system. The optimal choice of residual reduction for a Jacobian-free Newton-Krylov finite volume solid mechanics procedure is explored in Section 4. Check: do we examine this?

4 Test Cases

This section assesses the performance of the proposed Jacobian-free Newton-Krylov solution approach and several benchmark cases. The cases have been chosen to exhibit a variety of characteristics in terms of

- Geometric dimension (2-D vs. 3-D),
- Geometric nonlinearity (small strain vs. large strain)
- Statics vs. dynamics, and
- Material behaviour (elasticity, elastoplasticity, hyperelasticity).

In addition, through the analysis of the benchmark cases above, the effect of several parameters will be examined, including the mesh types, Rhie-Chow stabilisation scaling, preconditioner choices, linear solver settings, the effect of globalisation strategies, and multi-CPU-core parallelisation. The performance of the Jacobian-free Newton-Krylov algorithm is compared with that of the segregated algorithm in terms of computational time and memory requirements. Metrics from a commercial finite element software (Abaqus) are included for reference ask Dylan to run Abaqus cases once we have our results.

The presented analyses aim to be extensive; however, they are not exhaustive. Several common features of modern solid mechanics procedures, including contact mechanics and incompressibility, where mixed formulations are required, are left for future work.

4.1 Description of the Selected Benchmark Cases

Here, for the selected 4-6 cases, we should describe: - geometry (+ image) and meshes (maybe image of one type of mesh) - material properties - loading conditions

I suggest we aim to keep the descriptions above concise we can use tables where appropriate and can include the loading conditions in the geometry figure

4.2 Accuracy and Order of Accuracy Verification

This should show that JFNK does not affect the answers: one case is fine

4.3 Time and Memory Requirements

Present a table of time and memory for meshes and cases

4.4 Effect of XXXX

There are many parameters we could examine; here are ideas

- **Mesh types**: tet, hex, poly. MAYBE. We would pick one case and show its timings and memory (and maybe compare residual convergence) for tet (Gmsh is a good option), hex and poly (Gmsh + polyDualMesh, or maybe Fluent?). Alternatively, we can use different meshes for different cases in the "time and memory requirements" section, e.g. ellipticPlate could use hex, but narrowTmember uses tet. However, then it would not be clear if the differences come from the cases or the mesh.
- **Rhie-Chow stabilisation scaling**: MAYBE. from a quick check, the value of s greatly affects the convergence of the linear solver: $s = 1$ shows good convergence, $s = 0.1$ is poor convergence. It might be a nice insight to show that smoothing is equally important for the linear solver and the solution stability.
- **Preconditioner choices**: YES: compare ILU, LU, and multigrid. This should be easy to do. We can use multigrid for all other sections (I think it will be the best), but here, we can show the differences (time and memory) for some or all of the cases and mesh densities.
- **Linear solver settings**: MAYBE: In GMRES, we must choose the value of the "restarts" parameter; the default in PETSc of 30 does not always seem good enough (I have been setting it to 100). We could show its affect on one case. I think the optimal choice is linked with the choice of preconditioner (a better preconditioner means we can use a small restart value) so maybe we include this with the "preconditioner choices" section.
- **Linear solver oversolving**: MAYBE: this can also be considered a linear solver setting. For segregated, we always use a relative tolerance of 0.1, but initial tests for JFNK show that 0.1 is too loose and 1e-3 seems better, although maybe there is no difference with using 1e-6.
- **The effect of globalisation strategies**: MAYBE: we could simply say that we use line search and a predictor step, or we could show their effect. Maybe pick one or more cases

- **Multi-CPU-core parallelisation**: YES: final sub-section
- **Something else?**

Let's prioritise these and complete the first few. I have added my thoughts above in CAPITALS.

4.5 Parallelisation

Here, for one or more cases, we should show: - strong scaling: largest mesh, speed vs number of cores: I can use Meluxina - (optional) weak scaling: time vs number of cores as we keep the work-per-core constant

Comments from [**?**]: "The first is a scalable implementation, in the sense that time per iteration is reduced in inverse proportion to the number of processors (strong scaling), or that time per iteration is constant as problem size and processor number are scaled proportionally (weak scaling). The second is good per processor performance on contemporary cache-based microprocessors. The third is algorithmic scalability, in the sense that the number of iterations to convergence does not grow with increased numbers of processors (or problem size). The third factor arises because the requirement of a scalable implementation generally forces parameterized changes in the algorithm as the number of processors grows. If the convergence is allowed to degrade, however, the overall execution is not scalable, and this must be countered algorithmically."

5 Conclusions

In the current work, a Jacobian-free Newton-Krylov solution algorithm has been proposed for solid mechanics problems discretised using the cell-centred finite volume method. A compact-stencil discretisation of the diffusion term is proposed as the preconditioner matrix, allowing a straightforward extension of existing segregated solution frameworks. The key findings of the work are:

- Main results
- Main important choices
- How it compares to segregated methods and FE (Abaqus)

Acknowledgments. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant Agreement No. 101088740). Financial support is gratefully acknowledged from the Irish Research Council through the Laureate programme, grant number IRCLA/2017/45, from Bekaert through the University Technology Centre (UTC phases I and II) at UCD (www.ucd.ie/bekaert), from I-Form, funded by Science Foundation Ireland (SFI) Grant Numbers 16/RC/3872 and RC2302_2, co-funded under European Regional Development Fund and by I-Form industry partners, and from NexSys, funded by SFI Grant Number 21/SPP/3756. Additionally, the authors wish to acknowledge the DJEI/DES/SFI/HEA Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support (www.ichec.ie), and part of this work has been carried out

using the UCD ResearchIT Sonic cluster which was funded by UCD IT Services and the UCD Research Office. Ivan: add any additional acknowledgements here

References