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

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

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

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 $\boldsymbol{f}_b$ is a body force per unit volume, e.g., $\rho \boldsymbol{g}$, where $\boldsymbol{g}$ is 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

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

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

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 s 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 on 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 but not exhaustive. Several common features of modern solid mechanics procedures are left for future work, including contact mechanics and incompressibility.

The remainder of this section is structured as follows: the order of accuracy of the proposed discretisation is assessed on a 3-D linear elastic case with a known analytical solution. This demonstrates that the predictions are unaffected by the solution algorithm choice. Subsequently, the remaining sub-sections assess the effect of several key parameters on the Jacobian-free Newton Krylov approach, including XXX.

4.1 Description of the Selected Benchmark Cases

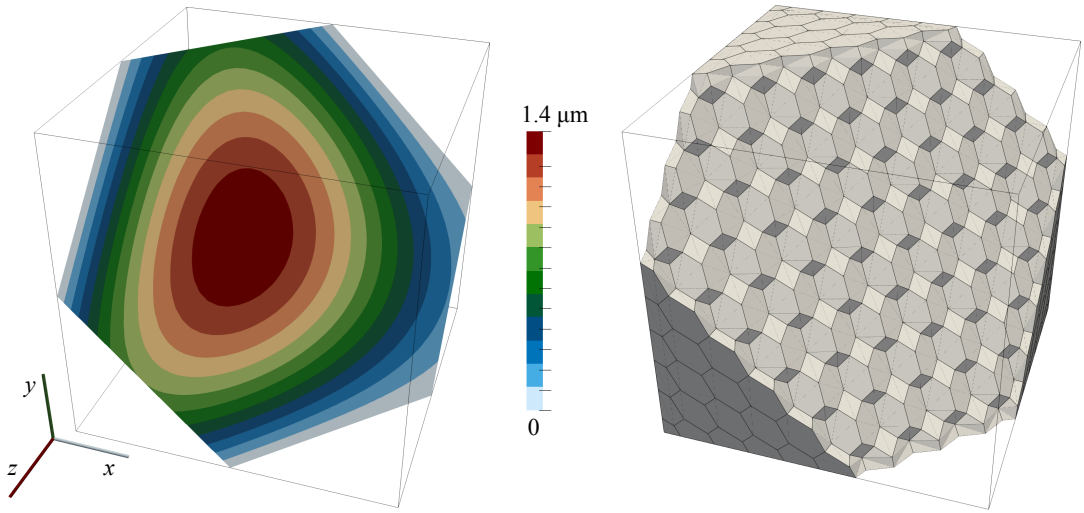
This section concisely describes the benchmark cases examined in subsequent sections. Details of the geometry, mesh, loading conditions, material properties, and relevant numerical settings are given so that the results can be reproduced. We may need to drop some of these cases if we have too many: cases 2 and 3 are very similar (3-D, static, linear elastic)

Case 1: Order Verification via the Manufactured Solution Procedure

The first test case consists of a $0.2 \times 0.2 \times 0.2$ m cube with linear elastic ($E = 200$ GPa, $\nu = 0.3$) properties. A manufactured solution for displacement (Figure 1(a)) is employed of the form:

$$\boldsymbol{u} = \begin{pmatrix} a_x \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \\ a_y \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \\ a_z \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \end{pmatrix} \tag{27}$$

where $a_x = 2\times10^{-6}$ m, $a_y = 4\times10^{-6}$ m, and $a_z = 6\times10^{-6}$ m. The Cartesian coordinates are given by x, y and z. The resulting body force to force the manufactured solution is given in Appendix A.



(a) Magnitude of the manufactured displacement solution (b) Polyhedral mesh with 1 000 cells

Fig. 1 A cut plane through the cube case geometry showing the magnitude of the manufactured displacement solution (left) and a polyhedral mesh (right). The cut plane passes through the centre of the cube and has the unit normal $\boldsymbol{n} = (1/\sqrt{3} \quad 1/\sqrt{3} \quad 1/\sqrt{3})$.

The manufactured displacement solution is applied at the domain's boundaries, and inertial effects are neglected. Two forms of mesh are examined: (i) structured, uniform hexahedra and (ii) regular polyhedra (Figure 1(b)). To create the polyhedral meshes, structured tetrahedra meshes are first generated using Gmsh [1] and then converted to their dual polyhedral representations using the OpenFOAM `polyDualMesh` utility. Starting from an initial mesh spacing of 0.04 m, six meshes are created by successively halving the spacing. The cell numbers (hexahedra and polyhedral meshes) are 125, 1 000, 8 000, 64 000, 512 000, and 4 096 000.

Case 1b: Spherical Cavity in an Infinite Solid Subjected to Remote Stress

This 3-D case consists of a spherical cavity with radius $a = 0.2$ m (Figure 2) in an infinite, isotropic linear elastic solid ($E = 200$ GPa, $\nu = 0.3$). Far from the cavity, the solid is subjected to a tensile stress $\sigma_{zz} = T = 1$ MPa, with all other stress components zero. The analytical expressions for the

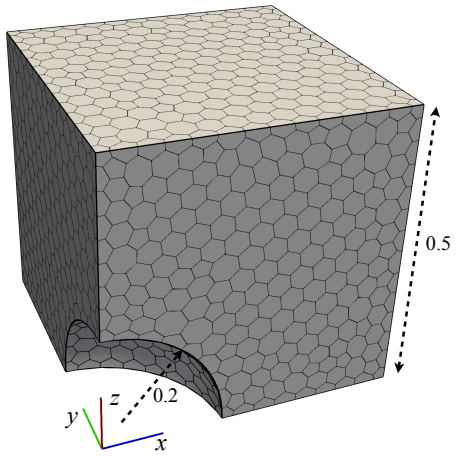


Fig. 2 Spherical cavity case geometry, showing the polyhedral mesh with 2,257 cells Update to graded mesh

stress distributions around the cavity, first derived by [**?**], are

$$\sigma_{rr} = \frac{T}{14-10\nu}\frac{a^3}{R^3}\left[9-15\nu-12\frac{a^2}{R^2}-\frac{r^2}{R^2}\left(72-15\nu-105\frac{a^2}{R^2}\right)+15\frac{r^4}{R^4}\left(5-7\frac{a^2}{R^2}\right)\right], \tag{28}$$

$$\sigma_{\theta\theta} = \frac{T}{14-10\nu}\frac{a^3}{R^3}\left[9-15\nu-12\frac{a^2}{R^2}-15\frac{r^2}{R^2}\left(1-2\nu-\frac{a^2}{R^2}\right)\right], \tag{29}$$

$$\sigma_{zz} = T\left[1-\frac{1}{14-10\nu}\frac{a^3}{R^3}\left\{38-10\nu-24\frac{a^2}{R^2}-\frac{r^2}{R^2}\left(117-15\nu-120\frac{a^2}{R^2}\right)+15\frac{r^4}{R^4}\left(5-7\frac{a^2}{R^2}\right)\right\}\right], \tag{30}$$

$$\sigma_{zr} = \frac{T}{14-10\nu}\frac{a^3 zr}{R^5}\left[-3(19-5\nu)+60\frac{a^2}{R^2}+15\frac{r^2}{R^2}\left(5-7\frac{a^2}{R^2}\right)\right]. \tag{31}$$

where $r^2 = x^2 + y^2$ is the cylinderical radial coordinate, $R^2 = r^2 + z^2$ is the spherical radial coodinate, and x, y, z are the Cartesian coordinates.

Add displacement equations from Goodier We can move the analytical solutions to an appendix

The solution domain is taken as one-eighth of a $1 \times 1 \times 1$ m cube centred on the sphere. The analytical tractions are applied at the far boundaries of the domain to mitigate the effects of finite

geometry. Unstructured polyhedral meshes of varying densities are employed (state average cell widths, and number of cells in each mesh). The mesh with 2,257 cells is shown in Figure 2. Initially, unstructured tetrahedral meshes were generated using the Gmsh meshing utility [1], followed by conversion to their dual polyhedral representations using the OpenFOAM `polyDualMesh` utility.

Case 2: Out-of-plane bending of an elliptic plate

This 3-D, static, linear elastic test case (Figure 3) consists of a thick elliptic plate (0.6 m thick) with a centred elliptic hole, with the inner and outer ellipses given as

$$\left(\frac{x}{2}\right)^2 + \left(\frac{y}{1}\right)^2 = 1 \text{ inner ellipse} \tag{32}$$

$$\left(\frac{x}{3.25}\right)^2 + \left(\frac{y}{2.75}\right)^2 = 1 \text{ outer ellipse} \tag{33}$$

The case has been described by the National Agency for Finite Element Methods and Standards (NAFEMS) [**?**], and analysed using finite volume procedures by **?**] and **?**]. Symmetry allows one quarter of the geometry to be simulated. A constant pressure of 1 MPa is applied to the upper surface, and the outer surface is fully clamped. The mechanical properties are: $E = 210$ GPa, $\nu = 0.3$. Comment on the meshes used.

placeholder

Fig. 3 Elliptic Plate case geometry, mesh and loading conditions

Case 3: Narrow T-section component under tension

This case, proposed by **?**], consists of a narrow engineering component with a T cross-section (Figure 4). The case is 3-D, static, with linear elastic material behaviour. Symmetry allows one quarter of the geometry to be simulated. A constant negative pressure of 1 MPa is applied to the lower surface and the upper left surface is fully clamped. The Young's modulus is $E = 210$ GPa and Poisson's ratio is $\nu = 0.3$. A hole of radius 5 mm is located at the expected stress concentration. Comment on the meshes

placeholder

Fig. 4 Inflation of an idealised ventricle case geometry, mesh and loading conditions

Case 4: Inflation of an idealised ventricle

Inflation of an idealised ventricle (Figure 5) was proposed by **?**] as a benchmark problem for cardiac mechanics software. The case is 3-D, static, with finite hyperelastic strains. The initial geometry is defined as a truncated ellipsoid:

$$x = r_s \sin(u)\cos(v), \quad y = r_s \sin(u)\sin(v), \quad z = r_l \cos(u) \tag{34}$$

where on the inner (endocardial) surface $r_s = 7$ mm, $r_l = 17$ mm, $u \in \left[-\pi, -\arccos\left(\frac{5}{17}\right)\right]$ and $v \in [-\pi, \pi]$, while on the outer (epicardial) surface $r_s = 10$ mm, $r_l = 20$ mm, $u \in \left[-\pi, -\arccos\left(\frac{5}{20}\right)\right]$ and $v \in [-\pi, \pi]$. The base plane $z = 5$ mm is implicitly defined by the ranges for u. The hyperelastic material behaviour is decsribed by the transversely isotropic constitutive proposed by **?**] law, where the parameters $C = 10$ kPa, $b_f = b_t = b_{fs} = 1$. The chosen parameters produce isotropic behaviour. A pressure of 10 kPa is applied to the inner surface, and the base plane is fixed. The geometry is meshed using structured approach and is predominantly composed of hexahedra, with prism cells forming the apex. Four succesively refined meshes are examined: 1,620 (shown in Figure 5), 12,960, 103,680, and 829,440 cells.

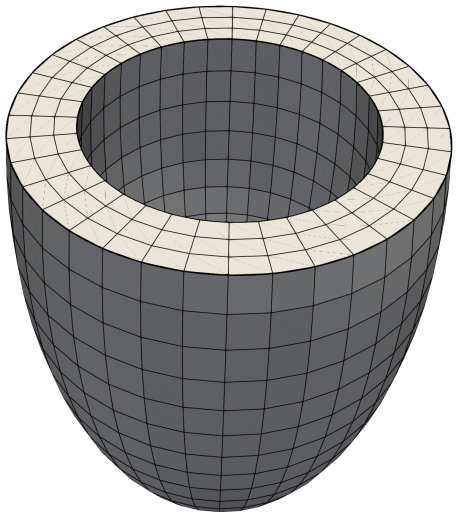


Fig. 5 Idealised ventricle case geometry, showing the mesh with 1,620 cells

Case 5: Cook's membrane

Cook's membrane (Figure 6) is a well-known bending-dominated benchmark case used in linear and non-linear analysis. In current work, the finite strain elastoplastic version [**?**] is considered. We could also consider the small-strain elastic version if needed The 2-D tapered panel (trapezoid) is fixed on one side and subjected to uniform shear traction on the opposite side. The prescribed shear traction is $\tau = 0.3125$ MPa. The vertices of the trapezoid (in mm) are (0, 0), (48, 44), (48, 60), and (0, 44). There are no body forces, and the problem is solved as statically as 2-D plane strain. The problem is solved as static, using 30 equally-sized loading increments.

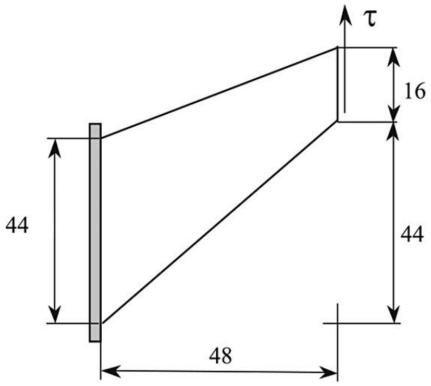


Fig. 6 Cook's membrane case geometry (dimensions in mm) and loading conditions (fixed left, traction τ applied to right)

The Young's modulus $E = 206.9$ MPa and Poisson's ratio $\nu = 0.29$, with the yield stress σ_y given as [**?**]

$$\sigma_y = \sigma_Y + H\bar{\varepsilon}_p + (\sigma_\infty - \sigma_Y)(1 - e^{-\delta\bar{\varepsilon}_p}) \tag{35}$$

The plastic yielding parameters are $\sigma_Y = 0.45$ MPa, $\sigma_\infty = 0.715$ MPa, $\delta = 16.93$, and $H = 0.12924$ MPa, where the hardening variable $\bar{\varepsilon}_p$ corresponds to equivalent plastic strain.

Case 6: Vibration of a 3-D Cantilevered Beam

This 3-D, dynamic, finite strain case geometry consists of a $2 \times 0.2 \times 0.2$ cuboid column (Figure 7), taken from **?**]. A sudden, constant traction $\boldsymbol{T} = (|\boldsymbol{T}|/\sqrt{2})\,(1, 1, 0)$ Pa is applied to the upper surface, where the magnitude $|\boldsymbol{T}|$ is defined in terms of a dimensionless load factor $\mathcal{F}$ as

$$|\boldsymbol{T}| = \frac{\mathcal{F}EI}{AL^2} \text{ Pa} \tag{36}$$

In the current study, $\mathcal{F} = 1$. A St. Venant-Kirchoff material is assumed with $E = 15.293$ MPa, $\nu = 0.3$ and density $\rho = 1000$ kg m^{-3}, while the area $A = (0.2)(0.2) = 0.04$ m^2 and the second moment of area $I = 0.0001333$ m^4. The geometric and material parameters were chosen for the first natural frequency to be 1 Hz. Comment on the meshes A fixed time step size of XXX s is chosen, corresponding to a YY on the coarsest mesh update as required. The total time period is 10 s, corresponding to 10 expceted oscillation periods.

placeholder

Fig. 7 Vibration of a 3-D Cantilevered Beam case geometry, mesh and loading conditions

Case 7: Vibration of an Axial Turbine Blade

This 3-D, dynamic, finite strain case geometry consists of a twisted axial turbine blade, 0.8 m in height, with a cord length of 0.2 m (Figure 8), and is taken from **?**]. Zeljko probably has this geometry or a reference for where it came from The blade is constrained at one end and subjected to a sudden, constant traction of $\boldsymbol{T} = (\text{ZZZ,ZZZ,ZZZ})$ Pa at the other end. A St. Venant-Kirchoff material is assumed with $E = 15.293$ MPa and $\nu = 0.3$, while the area $A = (0.2)(0.2) = 0.04$ m^2 and the second moment of area $I = 0.0001333$ m^4. Comment on the meshes A fixed time step size of XXX s is chosen, corresponding to a YY on the coarsest mesh update as required. The total time period is XXX s, corresponding to XX expceted oscillation periods. An advantage of this case over the cantilever is that it is complex geometry; a disadvantage is that we do not have a reference solution A mitigation for this disadvantage is we could run it in Abaqus to provide a credible reference

placeholder

Fig. 8 Vibration of an axial turbine blade case geometry, mesh and loading conditions

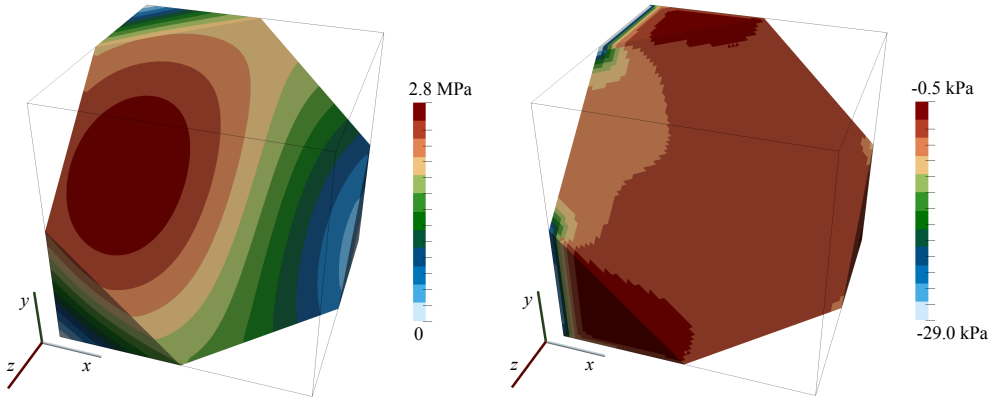
4.2 Order of Accuracy Verification

This section assesses the order of accuracy of the discretisation using a manufactured solution on a cube domain (case 1). A secondary purpose is to demonstrate that the choice of solution algorithm (Jacobian-free Newton-Krylov vs segregated) does not affect the predictions, assuming iteration errors are small.

The predicted σ_{xx} stress distribution for hexahedral mesh with 512 000 cells is shown in Figure 9(a). The corresponding cell-wise σ_{xx} error distribution is shown in Figure 9(b), where the errors of greatest magnitude (-29 kPa) occur at the ends of the boundary.

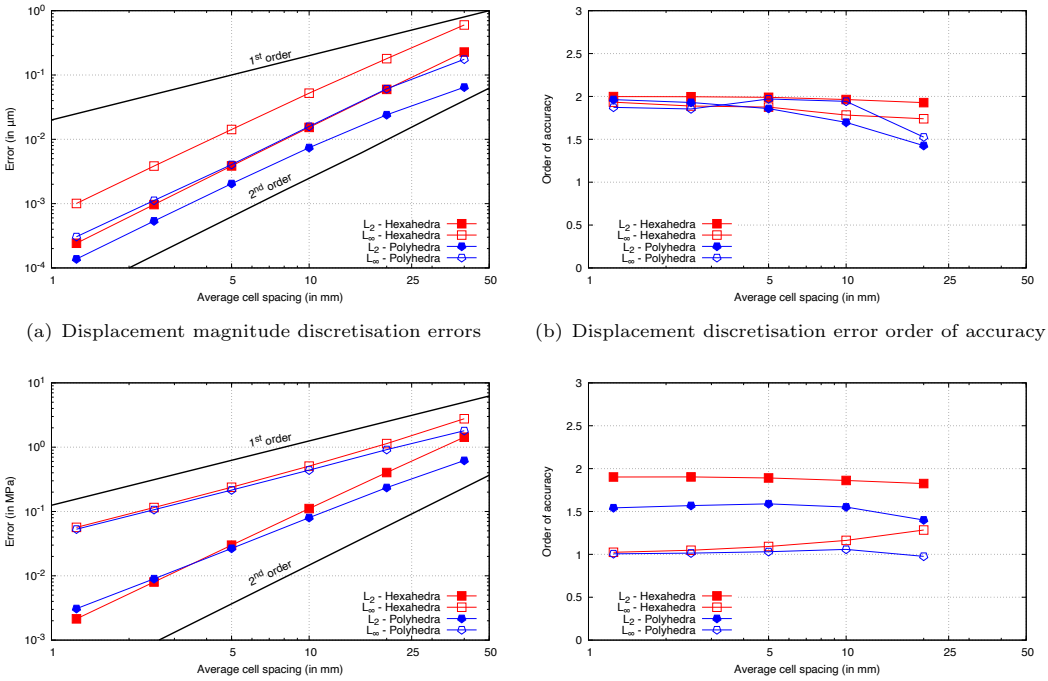
Figure 10(a) shows the displacement magnitude discretisation errors (L_2 and L_∞) as a function of the average cell width for both the hexahedral and polyhedral meshes. Figure 10(b) shows the corresponding order of accuracy plots. The maximum (L_∞) and average (L_2) discretisation errors are seen to reduce at an approximately second-order rate for both hexahedral and polyhedral meshes, where the errors are smaller on the hexahedral meshes check. The discretisation errors in the σ_{xx} component of stress are shown in Figure 10(b), and the order of accuracy in 10(d). The order of accuracy for the average stress magnitude error is seen to be XX for the hexahedral meshes and just over 1.5 for the polyhedral meshes. In contrast, the maximum stress error order of accuracy is seen to be 1 for both mesh types check.

The presented results have been generated using the Jacobian-free Newton-Raphson solution algorithm; however, minimal differences were seen when using the segregated solution algorithm,



(a) Predicted σ_{xx} stress distribution (b) Cell-wise σ_{xx} error distribution

Fig. 9 Manufactured solution cube case: the predicted σ_{xx} stress distribution on a cut-plane for the hexahedral mesh with 512 000 cells (left).



(a) Displacement magnitude discretisation errors (b) Displacement discretisation error order of accuracy

(c) Stress magnitude discretisation errors (d) Stress discretisation error order of accuracy

Fig. 10 Manufactured solution cube case: the accuracy and order of accuracy for displacement and stress

and hence, the results are not shown check. Analysis of time and memory requirements is left to subsequent sections.

4.3 Time and Memory Requirements

This section presents the time and memory requirements of the proposed Jacobian-free Newton Krylov approach for all six cases described in Section 4.1, where several mesh densities are examined. Time and memory requires from the classic segregated approach are given for comparison, along with the requirements from commercial finite element software Abaqus (version XXX): ask

Dylan to create the Abaqus results when we are happy the paper will not change. The finite element approach uses a coupled solution algorithm and direct linear solver, where a Newton-Raphson method is used for resolving nonlinearities. Take care when comparing times between different hardware

Figure 11 provides a collage of the equivalent (von Mises) stress fields for the six cases.

placeholder

Fig. 11 Collage of the equivalent (von Mises) stress fields for the six cases

Table 1 lists the wall clock times (time according to a clock on the wall) and maximum memory usage according to the GNU time utility. All clock times and memory usage were generated using hardware details using one CPU core. Times and maximum memory usage were recorded using the GNU time utility. Give details of relevant numerical settings, e.g. linear solver, preconditioner, Rhie-Chow scaling, globalisation strategy if any. Multi-CPU-core parallelisation is examiend in Section 4.8.

Case	Number of Cells	Time (in s)	Memory (in MB)
Spherical cavity	1,234	345	1,234
Spherical cavity	12,345	345	1,234
Spherical cavity	123,456	345	1,234
Case 2, etc	123,456	345	1,234
Case 3, etc	123,456	345	1,234
Case 4, etc	123,456	345	1,234

Table 1 Execution times and maximum memory usage speed-up column?

The trends shown in Table 1 are common to all typical modern computer systems, however, the exact values depend on the particular details of the CPU, the memory configuration and the operating system. To highlight the effect of hardware specification, Table 2 lists the time and memory usage for a different system: e.g. Mac M2/M3 seem to give excellent performance vs Meluxina AMD EPYC cores. Describe the difference, e.g. 2-3 faster but same memory usage Additional comments/insights, e.g. time trends are the same for both systems We could even include multiple systems here, e.g. Mac Studio vs Meluxina AMD vs Zagreb Intel workstation vs Intel i9/i7

Hardware	Case	Number of Cells	Time (in s)	Memory (in MB)
Mac Studio, M2 Ultra	Spherical cavity	1,234	345	1,234
Mac Studio, M2 Ultra	Spherical cavity	12,345	345	1,234
Mac Studio, M2 Ultra	Spherical cavity	123,456	345	1,234
Meluxina, AMD EPYC	Spherical cavity	1,234	345	1,234
Meluxina, AMD EPYC	Spherical cavity	12,345	345	1,234
Meluxina, AMD EPYC	Spherical cavity	123,456	345	1,234

Table 2 Execution times and maximum memory usage for Hardware Spec 2

4.4 Effect of Mesh Type

To highlight the effect of mesh type on the performance of the proposed Jacobian-free Newton-Krylov approach, case X is re-examined using several mesh types: (i) structured hexahedral; (ii) structured tetrahedral; (iii) unstructured tetehrahedral; and (iv) unstructured poyhedral. The hexahedral and tetahedral meshes have been generated using the Gmsh utility [1] and the polyhedral mesh has been created by converting a tetrahedral mesh using the OpenFOAM `polyDualMesh` utility.

Table 3 lists the execution times and total number of linear solver iterations for the different mesh types and densities. Comment on the results: e.g. fastest, slowest, most/least iterations, etc Provide insights, if any: e.g. why is one faster or slower Conclusion: mesh type does or does not affect the performance.

Mesh Type	Number of Cells	Time (in s)	Linear Solver Iterations
Structured hexahedral	1,234	345	1,234
Structured hexahedral	12,345	345	1,234
Structured hexahedral	123,456	345	1,234
Structured tetrahedral	1,234	345	1,234
Structured tetrahedral	12,345	345	1,234
Structured tetrahedral	123,456	345	1,234
Unstructured tetrahedral	1,234	345	1,234
Unstructured tetrahedral	12,345	345	1,234
Unstructured tetrahedral	123,456	345	1,234
Unstructured polyhedral	1,234	345	1,234
Unstructured polyhedral	12,345	345	1,234
Unstructured polyhedral	123,456	345	1,234

Table 3 Execution times on Case X for different mesh types

4.5 Effect of the Preconditioner Choice

A critical factor contributing to the performance of Jacobian-free Newton Krylov methods is the formulation of the linear solver preconditioner and related linear solver settings. In this section, three choices of preconditioner are compared:

- **LU** - A LU decomposition direct solver Is this MUMPS? To be checked. A direct solver is expected to be the more robust but suffer from excessive time and memory requirements for larger numbers of unknowns.
- **ILU(N)** - Incomplete LU decomposition with N fill-in. ILU(N) is expected to have lower memory requirements but at the expense of robustness. Additionally, as the system of unknowns becomes larger, the number of ILU(N) iterations is expected to increase. As the fill-in factor N is increased, the performance of ILU(N) approaches that of the LU direct solver.
- **Algebraic multigrid** - The HYPRE Boomerang parallelised multigrid preconditioner. Multigrid approaches have the potential to offer superior performance than other methods for larger problems, with near linear scaling of time and memory requirement.

An additional consideration when selecting a preconditioner is its ability to scale in parallel as the number of CPU cores increases. From this perspective, the iterative approaches (ILU(N) and

multigrid) are expected to show better parallel scaling than direct methods (LU). Analysis of this point is left to Section 4.8.

Table 4 presents the execution times and memory requirements for cases X, Y and Z for different mesh densities. Comment on the results: e.g. fastest, slowest, most/least iterations, etc Provide insights, if any: e.g. why is one faster or slower Conclusion: which preconditioner if 'best' for which situation: probably LU is best for 'small' cases and multigrid for 'large' cases

Case	Number of Cells	Preconditioner	Time (in s)	Memory (in MB)
Case X	1,234	LU	345	1,234
Case X	12,345	LU	345	1,234
Case X	123,456	LU	345	1,234
Case X	1,234	ILU(0)	345	1,234
Case X	12,345	ILU(0)	345	1,234
Case X	123,456	ILU(0)	345	1,234
Case X	1,234	Multigrid	345	1,234
Case X	12,345	Multigrid	345	1,234
Case X	123,456	Multigrid	345	1,234
Case Y	1,234	LU	345	1,234
Case Z	1,234	LU	345	1,234

Table 4 Execution times and maximum memory usage for different preconditioners on varying cases and mesh densities Try other value of N for ILU?

Other linear solver settings: mention these but no need for analyses

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. I suggest we do not add an analysis of this: instead we can just mention it

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. I suggest we do not add an analysis of this: instead we can just mention it

4.6 Effect of Rhie-Chow Stabilisation

This section highlights a key effect in the performance of the proposed Jacobian-free Newton Krylov method: the choice of the global scaling factor α in the Rhie-Chow stabilisation (Equation 17). Without this (or a similar) stabilisation term, zero-energy oscillations, such as checkerboarding, may appear in the solution fields. As the amount of stabilisation is increased (increasing α), these numerical modes are quelled and the solution is stabilisation; however, at some point, further increases in amount of stabilisation reduce the accuracy of the discretisation due to oversmoothing. Consequently, a good choice of the stabilisation global scaling factor α is one which is sufficiently high to quell oscillations but not higher. This challenge is related to the discretisation and hence is common to all solution algorithms, including segregated and Jacobian-free Newton Krylov methods.

Nonetheless, the amount of stabilisation can affect the convergence of the linear solver in Jacobian-free Newton Krylov methods, for example, as examined by **?**] for a XXX Euler flow finite volume formulation.

Table 5 presents the execution times and total number of linear solver iterations for the case XX for $\alpha = [0.01, 0.1, 1]$ for several mesh densities. Comment on the results: e.g. fastest, slowest, most/least iterations, if any cases diverged or needed a higher 'restarts' value, etc Provide insights, if any: e.g. why is one faster or slower

Case	Number of Cells	α	Time (in s)	Linear Solver Iterations
Idealised Ventricle	1,234	0.01	345	1,234
Idealised Ventricle	12,345	0.1	345	1,234
Idealised Ventricle	123,456	1.0	345	1,234
Axial Turbine	1,234	0.01	diverged	diverged
Axial Turbine	12,345	0.1	345	1,234
Axial Turbine	123,456	1.0	345	1,234

Table 5 Execution times and total number linear solver iterations for different values of the Rhie-Chow stabilisation factor α

Figure 12 shows a comparison of the stress (or displacement) along the line XXX -¿ we need to show the effect on accuracy for different values of alpha

placeholder

Fig. 12 Stress predictions along the line XXX for the YYY case using different values of the Rhie-Chow stabilisation factor α

4.7 Effect of Globalisation Strategies

We already have many section so it may be better to just mention this rather than providing an analysis

4.8 Parallelisation

In this final analysis, the multi-CPU-core parallel scaling performance of the proposed Jacobian-free Newton Krylov method is examined. Two types of parallel scaling analyses are performed [**?**]:

- Strong scaling study: Strong scaling measures how the execution time of a fixed problem size (fixed number of cells) decreases as the number of CPU cores increases. Good strong scaling indicates that an application effectively utilises additional CPU cores without significant overhead. In contrast, poor strong scaling suggests that adding more cores does not significantly reduce the execution time, often due to increased communication or synchronisation overhead.

- Weak scaling study: Weak scaling measures how the execution time changes as the problem size (the number of cells) and the number of CPU cores increase proportionally, where the problem size per CPU core remains (approximately) constant. Good weak scaling indicates that the application can efficiently manage larger workloads with more cores without a significant increase in execution time. Poor weak scaling implies that the application struggles to maintain performance as the problem size and core count increase.

The results from the strong scaling study are shown in Figure 13. The speedup S is defined as $S = {}^{t_1}/_{t_p}$, where t_1 is the clock time on one CPU core and t_p is the clock time on P CPU cores. In the ideal case, the speedup should double when the number of cores is doubled. In reality, inter-CPU-core communication reduces the parallel scaling efficiency below the ideal. Comment on the results: e.g. fastest, slowest, most/least iterations, best/worst scaling Provide insights, if any: e.g. why is one faster or slower

(a) Clock times vs number of CPU cores

(b) Speedup S vs number of CPU cores

Fig. 13 Strong parallel scaling study comparing the performance of three preconditioning strategies: a LU direct solver, and ILU(N) and multigrid iterative solvers

The weak scaling results are shown in Figure 14. The weak scaling efficiency η_w is defined as $\eta_w = {}^{t\lfloor w \rceil_1}/_{t\lfloor w \rceil_p}$, where $t\lfloor w \rceil_1$ is the clock time on one CPU core and $t\lfloor w \rceil_p$ is the clock time on P CPU cores where the problem size per CPU core is constant. In the ideal case, the efficiency should be unity, but inter-CPU-core communication reduces it. Comment on the results: e.g. fastest, slowest, most/least iterations, best/worst scaling Provide insights, if any: e.g. why is one faster or slower

(a) Clock times vs number of CPU cores

(b) Weak scaling efficiency η_w vs number of CPU cores

Fig. 14 Weak parallel scaling study comparing the performance of three preconditioning strategies: a LU direct solver, and ILU(N) and multigrid iterative solvers

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/Zeljko: add any additional acknowledgements here

Appendix A Body Force for the Method of Manufactured Solutions Case

The body force for the manufactured solution case is

$$\boldsymbol{f}_b = \begin{pmatrix} \lambda \left[8a_y\pi^2 \cos(4\pi x)\cos(2\pi y)\sin(\pi z) \right. \\ \quad +4a_z\pi^2 \cos(4\pi x)\cos(\pi z)\sin(2\pi y) \\ \quad \left. -16a_x\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \right] \\ +\mu \left[8a_y\pi^2 \cos(4\pi x)\cos(2\pi y)\sin(\pi z) \right. \\ \quad +4a_z\pi^2 \cos(4\pi x)\cos(\pi z)\sin(2\pi y) \\ \quad \left. -5a_x\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \right] \\ -32a_x\mu_\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \\ \\ \lambda \left[8a_x\pi^2 \cos(4\pi x)\cos(2\pi y)\sin(\pi z) \right. \\ \quad +2a_z\pi^2 \cos(2\pi y)\cos(\pi z)\sin(4\pi x) \\ \quad \left. -4a_y\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \right] \\ +\mu \left[8a_x\pi^2 \cos(4\pi x)\cos(2\pi y)\sin(\pi z) \right. \\ \quad +2a_z\pi^2 \cos(2\pi y)\cos(\pi z)\sin(4\pi x) \\ \quad \left. -17a_y\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \right] \\ -8a_y\mu_\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \\ \\ \lambda \left[4a_x\pi^2 \cos(4\pi x)\cos(\pi z)\sin(2\pi y) \right. \\ \quad +2a_y\pi^2 \cos(2\pi y)\cos(\pi z)\sin(4\pi x) \\ \quad \left. -a_z\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \right] \\ +\mu \left[4a_x\pi^2 \cos(4\pi x)\cos(\pi z)\sin(2\pi y) \right. \\ \quad +2a_y\pi^2 \cos(2\pi y)\cos(\pi z)\sin(4\pi x) \\ \quad \left. -20a_z\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \right] \\ -2a_z\mu_\pi^2 \sin(4\pi x)\sin(2\pi y)\sin(\pi z) \end{pmatrix} \tag{A1}$$

where μ and λ are the first and second Lamé parameters, respectively.

References

[1] Geuzaine, C., Remacle, J.-F.: Gmsh: A 3-d finite element mesh generator with built-in pre- and post-processing facilities. International journal for numerical methods in engineering **79**(11), 1309–1331 (2009) https://doi.org/10.1002/nme.2579