Iterative solvers within sequences of large linear systems in non-linear structural mechanics

Stefan Hartmann¹,*, Jurjen Duintjer Tebbens², Karsten J. Quint³, and Andreas Meister⁴.

¹ Institute of Applied Mechanics, Clausthal University of Technology, Adolph-Roemer-Str. 2a, 38678 Clausthal-Zellerfeld, Germany
² Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou Věží 2, 18207 Prague 8, Czech Republic
³ Institute of Mechanics, University of Kassel, Mönchebergstr. 7, 34125 Kassel, Germany
⁴ Department of Mathematics, University of Kassel, Heinrich-Plett Str. 40, 34125 Kassel, Germany

Received XXXX, revised XXXX, accepted XXXX
Published online XXXX

Key words iterative solver, non-symmetric matrices, sequences of linear systems, finite strains, finite elements

MSC (2000) 04A25

This article treats the computation of discretized constitutive models of evolutionary-type (like models of viscoelasticity, plasticity, and viscoplasticity) with quasi-static finite elements using diagonally implicit Runge-Kutta methods (DIRK) combined with the Multilevel-Newton algorithm (MLNA). The main emphasis is on promoting iterative methods, as opposed to the more traditional direct methods, for solving the non-symmetric systems which occur within the DIRK/MLNA. It is shown that iterative solution of the arising sequences of linear systems can be substantially accelerated by various techniques that aim at sharing part of the computational effort throughout the sequence. In this way iterative solution becomes attractive at clearly lower dimensions than the dimensions where direct solvers start to fail for memory reasons. The applications are related to small strain viscoplasticity of a smooth constitutive model for plastics and a finite strain plasticity model with non-linear kinematic hardening developed for metals.

1 Introduction

The treatment of constitutive models of evolutionary-type within implicit finite elements leads after the spatial discretization to a system of differential-algebraic equations (DAE-system) where the algebraic part results from the discretized weak formulation of the equilibrium conditions and the differential part is the outcome of the assemblage of all constitutive model’s evolutionary equations at all spatial integration points — Gauss-points (see, for example, [18], [25]). Usually, this is solved by a Backward-Euler method. In this article, however, use is made of time-adaptive stiffly accurate diagonally implicit Runge-Kutta methods, which contain the Backward-Euler approach as a particular case. In any case, this leads at each point in time of the time integration procedure to a coupled system of non-linear equations for the unknown nodal displacements and the internal variables at all Gauss-points of the entire finite element structure. To solve the coupled non-linear system the Multilevel-Newton algorithm (MLNA, see [42, 26]) is applied. This procedure yields a solution scheme with two levels and is not related to multigrid solvers. For the computation of the increment of the nodal displacements on the global level (outer loop) one has to solve an embedded non-linear system which is done on the local level. This system of non-linear equations is Gauss-pointwise decoupled so that small systems can be solved on Gauss-point or element level determining the internal variables. The outer loop leads in each iteration step to a linear equation system, where the coefficient matrix and the right-hand side are frequently called the tangential stiffness matrix and the residual force vector, respectively. Both determine the unknown increment of the nodal displacement. This system has to be solved within each time step (stage, load step) a number of times until the Multilevel-Newton algorithm has converged, i.e. the linear systems have to be solved several hundred times in total. Concerning this procedure we are interested in computing medium-sized linear systems (currently, this is approximately $n_u = 50000 - 250000$) instead of applying a strategy for solving only one system with several millions of unknowns per time step.

To solve the arising linear systems one can use direct or iterative solvers. Frequently it is stated that for smaller linear systems of equations direct solution schemes are superior (see, for an overview in the context of finite elements, [11] and

* Corresponding author, e-mail: stefan.hartmann@tu-clausthal.de, Phone: +00 49 5323 72 2774 Fax: +00 49 5323 72 2203
and iterative solvers do not seem to be applied often in structural mechanics unless memory demands make direct solution impossible. As both direct and iterative solution techniques have been substantially improved in the recent past, a comparison of current algorithms and available software would be of large interest. On the one hand, very efficient direct solvers for sparse non-symmetric linear systems can be used within the Multilevel-Newton algorithm, for example, SuperLU [14], UMFPACK [12] or PARDISO [47]. These packages are based on sparse storage schemes so that only necessary data are stored. However, these methods are limited to a certain size of the linear system and with growing dimensions iterative solvers become of more interest. Krylov-subspace solvers as GMRES [44] and BiCGSTAB [54] can be very efficient for large, sparse and non-symmetric matrices, especially if appropriate preconditioning techniques are applied. There are some publications reporting on iterative solution in the finite element literature on geometrical and physical non-linear problems. In [53] iterative solvers are applied to sheet forming processes within explicit and implicit integration schemes for dynamical and inelastic systems under consideration. There, conjugate gradient (CG, [30]), BiCGSTAB and GMRES solvers with and without symmetric successive overrelaxation (SSOR), incomplete Cholesky (IC) and Jacobi preconditioners are investigated for symmetric systems. For other applications using iterative solvers for symmetric applications in finite elements we refer to [46], [1], and [2]. Note that these articles address the case of symmetric linear systems only, whereas they sometimes apply methods like GMRES and BiCSTAB which were designed for the non-symmetric case. We feel that the computational experience with iterative solvers in structural mechanics is somehow restricted and that hints to exploit the many possibilities offered by this important class of linear solvers may be beneficial. In view of helpful textbooks on iterative solvers, we refer to [22], [43] and to [36, in German].

The main purpose of this paper is to show that with an appropriate combination of acceleration techniques iterative solvers may outperform direct solvers in structural mechanics problems surprisingly easily. In particular, the intersection between the efficiency curves of direct and iterative solvers can occur at rather moderate dimensions. We demonstrate this here in the context of the MLNA but suspect that similar conclusions can be drawn for other solution approaches. Of course, the efficiency of linear solvers depends to an important extent on the problem under consideration. The condition number of the system matrix, for instance, can strongly influence the convergence behavior. The main tools we will use to accelerate the convergence are related to two aspects of iterative methods that seem to have been little exploited for the considered applications before, namely flexible stopping criteria and flexible preconditioning. These tools are closely connected with the fact that we are here solving sequences of linear systems; this may in part explain why they are little known in structural mechanics. In addition, the required accuracy (stopping criteria) of both the interior loop of the MLNA and the iterative solver are connected to the accuracy of the global loop (see, for example, [42], [31]). Furthermore, we spend considerable time on the description of more basic aspects of and useful hints for iterative solution techniques (like, for instance, preconditioning in general). We also give a brief description of modern direct solvers and in our study we compare the most advantageous combination of techniques for both iterative and direct solution. Additional optimization can be achieved by parallelizing all algorithms, but we considered this out of the scope of the article. It depends on the constitutive model whether the total computational time can be reduced essentially due to acceleration of the linear solver. This will be investigated in detail because if on local, i.e. element level the computational time dominates the overall efficiency, the improvement of the iterative solvers is only of minor importance.

In [40] a multigrid method is applied to various models of viscoplasticity, whereas a similar approach treats the more smooth problem of porous media in [55, 56]. The single-grid algorithms are based on the GMRES and BiCGSTAB algorithms. In this respect, the investigations here are restricted to single-grid applications and focus on an approach where essential changes in the program have not to be carried out due to the multigrid idea. However, all studies can be embedded into multigrid schemes as well (in the context of multigrid applications in finite elements see, for example, [35], [49], and [39]).

The investigations are carried out for two different constitutive models. The first one was developed for small strain viscoplasticity for a plastic called polyoxymethylene (POM) representing a smooth problem; although non-linear, it is locally iteration-free, see [27]. The second problem comes from a highly non-linear finite strain plasticity model which is based on a von Mises-type yield function and non-linear kinematic hardening. This model-type depends on a case distinction whether the stress state results from an elasticity relation (algebraic equations) or the plastic strains and hardening variables evolve according to a DAE-system itself, see [29]. Both models are summarized in Appendix A.

The article is structured as follows: in Section 2 the global MLNA is explained. Afterwards, in Section 3, some ideas of iterative solution schemes are recapped and the acceleration strategies employed later on are described. Numerical studies including a comparison of the overall performance of iterative and direct solvers are presented in Section 4 and some concluding remarks can be found in Section 5.
2 Global solution procedure

In [18] the classical implicit finite element approach using constitutive models as viscoelasticity, elastoplasticity or viscoplasticity, is connected to the procedure of solving DAEs, see also [57] and [21]. In view of finite elements we have the systems

\[ F(t, y(t), y'(t)) := \begin{cases} g(t, u(t), q(t)) \\ q'(t) - r(t, u(t), u(t), q(t)) \end{cases} = 0 \]  

(1)

with

\[ y(t) := \begin{cases} u(t) \\ q(t) \end{cases}, \quad \text{and the initial conditions} \quad y(t_0) := \begin{cases} u(t_0) \\ q(t_0) \end{cases} =: y_0. \]

Here, \( g \in \mathbb{R}^{n_u} \) defines the discretized weak formulation depending on the approximation of the space discretization (small or finite strains, mixed formulation, etc.), see for different applications [25, 27] and [28]. Thereby, \( u \in \mathbb{R}^{n_u} \) are the unknown nodal displacements and \( q \in \mathbb{R}^{n_q} \) the internal variables at all spatial integration points. If \( n_q \) is the number of elements, \( n_q \) the number of all internal variables at one Gauss-point (for the models in Appendix A we have \( n_q = 12 \)) and \( n_e \) the number of Gauss-points in an element, then the number of all internal variables is \( n_Q = n_q \times n_e^e \times n_e \). Frequently, \( n_Q \) is much larger than the dimension of the linear systems \( n_q \). The internal variables evolve according to ordinary differential equations which are defined locally, i.e. on each Gauss-point, and are only formally assembled into the global system (1).

The application of stiffly accurate diagonally implicit Runge-Kutta methods (DIRK-methods, see, for example, [23]) to solve DAEs leads at each stage \( T_{ni} = t_n + c_i \Delta t_n, i = 1, \ldots, s \), with \( \Delta t_n = t_{n+1} - t_n \), where \( s \) defines the number of stages, to a system of non-linear equations:

\[ F\left(T_{ni}, Y_{ni}, \frac{Y_{ni} - S_{ni}}{\Delta t_n a_{ni}} \right) = \begin{cases} G_{ni}(U_{ni}, Q_{ni}) \\ L_{ni}(U_{ni}, Q_{ni}) \end{cases} := \begin{cases} g(T_{ni}, U_{ni}, Q_{ni}) \\ U_{ni} - S_{ni} \end{cases} = 0. \]

(3)

The generally non-linear equation system (3) stems from the implicit time-integration of (1), \( c_i, a_{ij} \) and \( b_i, i = 1, \ldots, s, \) are the elements of the Butcher array characterizing the DIRK-method under consideration.

\[ S_{ni}^u = u_n + \Delta t_n \sum_{j=1}^{i-1} a_{ij} \dot{u}_{nj} \]

(4)

\[ S_{ni}^q = q_n + \Delta t_n \sum_{j=1}^{i-1} a_{ij} \dot{q}_{nj} \]

(5)

are the starting vectors containing only known quantities at stage \( i \). Due to stiff accuracy, \( a_{si} = b_i, \) the quantities at time \( t_{n+1} \) are computed with the last stage so that the evaluation

\[ u_{n+1} = u_n + \Delta t_n \sum_{i=1}^{s} b_i \dot{u}_{ni} \]

(6)

\[ q_{n+1} = q_n + \Delta t_n \sum_{i=1}^{s} b_i \dot{q}_{ni} \]

(7)

is only required for estimating the local integration error using an embedded scheme.

One major point within the DIRK-method, which has to be addressed, is related to the solution of the coupled non-linear system (3) at each stage \( i \). It has become a habit in finite elements to apply a two-level strategy which is related to [42]. This is explained in the context of finite elements in [24] or in [26], see also the discussion in [48] and [32]. The Multilevel-Newton algorithm makes use of the implicit function theorem and is recapitulated in Tab. 1. This procedure consists of two levels. In the first level (local level), the integration step for the internal variables (3) is solved for given nodal displacements \( \dot{U}_{ni}^{(m)} \), \( m \) is the iteration index. This can be done on Gauss-point level since the equations are Gauss-pointwise decoupled so that only small systems of non-linear equations – of the dimension \( n_q \) – have to be solved, see Eq.(8) in Tab. 1. Additionally, the matrix \( dQ/dU \) is required, which represents the computation of small systems of linear equations of dimension \( n_q \) with several right-hand sides, \( Q_{ni}^{(m)} \) and \( dQ/dU_q \) have to be inserted into the coefficient matrix (tangential stiffness matrix) and the right-hand side (residual force vector) of Eq.(10) in Tab. 1 to get \( \Delta U_{ni} \).
The overall efficiency of the solution scheme to solve large DAE-systems in the context of finite elements and material non-linearities, is not only influenced by the linear solvers discussed in Sec. 3. However, particular emphasis has to be given on the non-linear solution scheme itself; here, the MLNA. One essential point of the overall efficiency concerns the estimation of the starting vector in the MLNA of Tab. 1. It is well known that the classical Newton-Raphson method or the MLNA converge quadratically and are, with respect to the convergence rate, superior to all other non-linear iteration methods, [31, 42]. This is also the reason for many authors to put a lot of analytical and numerical effort in obtaining the so-called consistent tangent, see [50], or more general in view of the MLNA, [26]. But the domain of convergence and therefore the performance of the Newton-Raphson method is restricted by the fact that the start vector is in the vicinity of the solution and thus highly influenced by the quality of the initial guess.

Remark 2.1 A first and very important efficiency hint is related to the initial guess of the non-linear solver. In standard textbooks on non-linear finite elements such as [3], [4] and [58] the authors suggest to use as initial guess the value which was obtained from the previous time step \( t_n \). The extension to the DIRK-method implies the starting vector of the last stage quantity (displacement). [3] also recommends to use the exact tangent stiffness matrix and to decrease the step-size if convergence problems of the Newton-iteration occur. [4] mention that the major restrictions on the size of the time-step in implicit methods arise (besides accuracy requirements) from the decreasing robustness of the Newton-procedure. He points out that large time-steps impair the convergence of the Newton-method and the starting iterate may be far from the solution. To get in the range of quadratic convergence and to significantly reduce the number of iterations an approximation of the solution should be used. This approximation is often referred to as extrapolator or predictor and these initial guesses can be based on linear or quadratic extrapolation from the previous increments [19], [51], [38]. In the case of DIRK methods also a so-called stage extension extrapolation (as introduced in [9] and studied in [10] for the case of electro- and magneto-quasistatic field simulations) can be used. In this paper we use a linear extrapolation from the last two stages (time

### Table 1

<table>
<thead>
<tr>
<th>Multilevel-Newton algorithm in the ( i )-th stage of time-step ( t_n ) ( \sim ) ( t_{n+1} )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Given:</strong> ( U_{ni}^{(0)} ) from initial guess of Eq.(12), ( Q_{ni}^{(0)} = q_n, \Delta t_n, T_{ni}, a_{ii}, S_{ni} )</td>
</tr>
<tr>
<td><strong>Repeat</strong> ( m = 0, \ldots )</td>
</tr>
<tr>
<td>( local \ level ) (given: ( U_{ni}^{(m)} ), argument vector ( y := (T_{ni}, U_{ni}^{(m)}, Q_{ni}^{(m)}) ))</td>
</tr>
<tr>
<td>local integration step</td>
</tr>
<tr>
<td>( L(U_{ni}^{(m)}, Q_{ni}^{(m)}) = 0 ) ( \sim ) ( Q_{ni}^{(m)} )</td>
</tr>
<tr>
<td>consistent linearization</td>
</tr>
<tr>
<td>( \left[ \frac{\partial L}{\partial Q} \right]_y \frac{dQ}{dU}_y = - \frac{\partial L}{\partial U}_y ) ( \sim ) ( \frac{dQ}{dU}_y )</td>
</tr>
<tr>
<td><strong>global level</strong></td>
</tr>
<tr>
<td>solve linear system of equations</td>
</tr>
<tr>
<td>( \left[ \frac{\partial G}{\partial U} \right]_y + \left[ \frac{\partial G}{\partial Q} \right]<em>y \frac{dQ}{dU}<em>y \right] \Delta U</em>{ni} = - G(y) ) ( \sim ) ( \Delta U</em>{ni} )</td>
</tr>
<tr>
<td>Update of global variables</td>
</tr>
<tr>
<td>( U_{ni}^{(m+1)} \leftarrow U_{ni}^{(m)} + \Delta U_{ni} ) ( \sim ) ( U_{ni}^{(m+1)} )</td>
</tr>
<tr>
<td>Until the convergence criterion is fulfilled</td>
</tr>
</tbody>
</table>
The notation is more a symbolic notation because the times $t_{(i)}$ define the stage times (implying in particular cases stage times of the last time-interval) and the displacement vectors are the quantities concerned. This extrapolation can be done at almost no expense and yields great savings; in many examples the number of iterations is halved and the MLNA is stabilized essentially. In the very first time step the prescribed step size is reduced to $\Delta t = 10^{-3} \Delta t_0$ to start the process when no information from previous stages is available. In our approach the starting vector of the internal variables is not extrapolated, only the starting vector of the nodal displacement vector is estimated using (12).

In the next section we concentrate on the efficient computation of Eq.(10) in Tab. 1.

3 Solution of linear system sequences

The sequence of linear algebraic equations of the form Eq.(10) can be solved with direct or iterative methods. Here, we have in mind standard direct and iterative approaches, namely LU-factorization and Krylov subspace methods respectively. A different direct approach like QR-factorization, for instance, is unnecessarily expensive for the purpose of solving linear systems and QR-factorization generates a full orthogonal matrix which is often too large to store. Generally, the so-called stationary iterative schemes like Jacobi- or Gauss-Seidel iteration converge slower than Krylov subspace methods and they lack the property of termination after at most $N$ steps in exact arithmetics, where $N$ is the system dimension. As mentioned in the introduction, we do not consider multigrid techniques in this paper.

In the following, we first give a useful overview of the main advantages and disadvantages of direct and iterative approaches and describe the direct and iterative methods we will use in the numerical experiments. Then, we focus on two important techniques that we exploit to accelerate iterative solution of the linear system sequence: We present a discussion on a stopping criterion within the computation of systems of non-linear equations, and we elaborate on preconditioning techniques.

3.1 Direct versus iterative solution

A classical LU-factorization is computed with at most $O(N^2)$ flops and allows solution of a linear system by forward and backward substitution solves with the right-hand side. The forward and backward solves cost at most $O(N^2)$ flops each. For sparse matrices these flop counts may be significantly lower, depending on the number of zero entries in the factorization. In the worst case fill-in, i.e. non-zero entries created during factorization at positions where the original matrix had zero entries, causes a very sparse matrix to be factorized into triangular factors without any zero entries. Probably the most important disadvantage of direct solution is its unpredictable storage costs, with the risk of not being able to store the factorization at all.

The amount of fill-in can be reduced by appropriate permutations (pivot choices) of the system matrix, which are an integral part of modern LU-packages. As finding permutations with minimum fill-in is an NP-hard problem [59], all packages use some kind of heuristic. Pivoting strategies are also used to enhance the numerical stability of the factorization process and most heuristics contain some compromise between pivoting for sparsity and pivoting for stability. A current strategy is to compute, in the so-called symbolic factorization phase, column permutations to reduce fill-in and to compute row permutations for stability during the numerical factorization.

Assuming that the growth factor of the matrix entries during factorization is moderate, LU-factorization with standard pivoting strategies is numerically stable in the sense that it computes the exact solution of a nearly linear system [22, Chapter 3]. It is often said that direct solvers with robust pivoting strategies may be less sensitive to high condition numbers of system matrices than other solvers. Also, LU-factorization is independent of the given right-hand side. Thus, in linear system sequences with identical matrices but varying right-hand sides (such sequences arise, e.g. in the chord and Shamanskii’s method; for the terminology see [31]) the factorization needs to be computed just once, with only the backward solves differing for every new linear system. In the case where the system matrices of the sequence do differ but their sparsity pattern is constant, one can save costs by performing the symbolic factorization only at the beginning of the sequence.

Krylov subspace methods are based on projection to subspaces which are generated through repeated multiplication of vectors (related to the right-hand side) with the system matrix. In favorable cases this reveals dominant properties of the
system at an early stage and we may expect rapid convergence. In most methods, the computational costs per iteration are dominated by the costs of the matrix-vector product and the main storage costs result from storing the system matrix. Hence Krylov subspace methods are particularly suited for large, sparse linear systems. However, convergence of the iterative process is not guaranteed in finite precision arithmetic. On the other hand every iteration produces an approximation to the exact solution that is perhaps not accurate but it may suffice for the purposes of the whole computation. A classical example of a computational process where inexact solution can be exploited is a non-linear system of equations solved by a Newton-type method; we elaborate this later.

In many applications Krylov subspace methods must be preconditioned for satisfactory convergence. The computation of a preconditioner can be totally for free (for some physics-based or splitting-based preconditioners) or it is done with significantly less effort than full factorization. We think for instance of incomplete LU-type factorizations where considerable computational as well as storage costs are saved by dropping nonzero entries during the process. The application of the preconditioner consists often of a backward and forward solve whose costs are related to the number of non-zeros of the triangular factors. These costs add to the overall costs per iteration of a preconditioned Krylov subspace method.

When solving a sequence of linear systems, Krylov subspaces generated in previous systems may be used to improve solution of the current system (see, e.g., [41]). This Krylov-subspace recycling is expected to be efficient mainly when the system matrices and right-hand sides generating the subspaces vary slowly from system to system. In addition, preconditioners may be recycled (or frozen) and to enhance the power of a frozen factorization an approximate preconditioner update may be beneficial [37, 5, 7, 15]. In some cases updates are efficient even with strongly varying system matrices, see e.g. [8].

An important advantage of Krylov subspace methods, that we will not exploit in this paper, is that they allow a matrix-free implementation. Storage of and computations with the system matrices can be avoided as long as we have a subroutine that approximates the matrix-vector product with the system matrix. For instance a matrix-vector product with a Jacobian may be replaced with a divided difference scheme based on function evaluations.

The question which solution strategy is most efficient, direct or iterative, depends evidently on the properties of the given linear systems. We will compare the two approaches for the particular systems from Eq.(10) generated in our applications. An important aspect is the efficient solution of the whole sequence of systems. Below we give descriptions of the methods and strategies used to test the two approaches; we tried to maximally exploit the individual advantages of the approaches.

### 3.1.1 Employed methods for direct solution

In our numerical experiments we test the direct solution with two popular modern LU-packages based on different factorization strategies, namely UMFPACK [12] and PARDISO [47]. UMFPACK is a standard example of a multifrontal solver. A frontal solver builds the LU decomposition of a sparse matrix given as the assembly of submatrices by assembling the matrix and eliminating equations only on a subset of rows and columns at a time. This subset is called the front and only parts of the matrix are assembled as they enter the front. Processing the front involves dense matrix operations, which use the CPU efficiently. In a typical implementation, only the front is in memory, while the factors in the decomposition are written into files. A multifrontal solver is an improvement of the frontal solver that uses several independent fronts at the same time. PARDISO uses a combination of left- and right-looking Level-3 BLAS supernode techniques. The algorithms are based on a Level-3 BLAS update and pipelining parallelism is exploited with left/right supernode techniques to improve sequential and parallel performance. The pivoting methods are also parallelized and allow complete supernode pivoting in order to compromise numerical stability and scalability during the factorization process. Even though PARDISO supports parallel execution this is not exploited in this paper. Both UMFPACK and PARDISO use minimum degree column ordering during the symbolic factorization phase to minimize fill-in. This ordering depends on the sparsity pattern of the matrix only and as the pattern is identical for all matrices of the form Eq.(10) it will be the same throughout the sequence. Thus, it needs to be computed only for the very first matrix, i.e. we use the option to recycle the initial minimum degree column ordering and skip symbolic factorization in the experiments with both UMFPACK and PARDISO.

### 3.1.2 Employed methods for iterative solution

We will present results with two very popular Krylov subspace methods for non-symmetric linear systems, GMRES and BiCGSTAB. An appealing property of GMRES is that it produces a non-increasing sequence of residual norms (the residual norm is the Euclidean distance between right-hand side and system matrix times approximate solution). However, storage and computational costs grow proportionally to the iteration number. Therefore, one often uses a restarted version of GMRES. Restarted GMRES processes do not possess the property that they terminate after at most \( N \) steps in exact arithmetics, and the processes risk to fully stagnate. The residual norms of BiCGSTAB can be oscillating and the process
can break down without having found the exact solution (such breakdowns are rare in practice). But unless such a break- 
down occurs, in exact arithmetics residual norms are zero by the \( N \)th step [43, Chapter 6]. The advantage of BiCGSTAB is 
that it does not need to be restarted because generating the Krylov subspaces is done with a short recurrence, thus requiring 
the storage of and computations with only a small number of vectors in every iteration. A GMRES iteration step performs 
one matrix-vector product whereas BiCGSTAB needs two matrix-vector products per iteration.

### 3.2 Stopping criteria

As mentioned above, an important advantage of iterative methods is that they can be stopped as soon as the approximation 
is good enough for the purpose of the whole computational process. For instance during the solution of partial differential 
equations with finite differences, volumes or elements, the discretization error resulting from discretizing the continuous 
problem may be important. Then, it does not make sense to solve the corresponding linear system on a significantly higher 
level of accuracy than the discretization accuracy.

Similarly, in the context of Newton methods for non-linear systems of equations, it is often not necessary to solve the 
linear systems to high accuracy in order to achieve satisfactory convergence of the non-linear residual. This is the main idea 
of inexact Newton methods [13]. Finding stopping criteria for the iterative solvers of linear systems in an inexact Newton 
method asks for balancing between moderately accurate solution of the linear problems and reasonably fast convergence 
of the non-linear process. The standard paper containing useful stopping criteria (or \textit{forcing terms}) for inexact Newton 
processes is [16]. Here, we were inspired by the tolerances of [31, Chapter 6]. Consider the non-linear system of equations 
\( \mathbf{g}(\mathbf{u}) = \mathbf{0} \) for a function \( \mathbf{g} \) with the Newton updates \( \mathbf{s}^{(m)} \) defined through the solutions of the linear systems 

\[ \mathbf{K}^{(m)}\mathbf{s}^{(m)} = -\mathbf{g}(\mathbf{u}^{(m)}), \]

where \( \mathbf{K}^{(m)} := \frac{d\mathbf{g}}{d\mathbf{u}}|_{\mathbf{u}=\mathbf{u}^{(m)}} \) is the Jacobian (tangential stiffness matrix) evaluated at the current approximation \( \mathbf{u}^{(m)} \). 
The residual norm of an approximation \( \mathbf{s} \) to \( \mathbf{s}^{(m)} \) is \( \|\mathbf{g}(\mathbf{u}^{(m)}) + \mathbf{K}^{(m)}\mathbf{s}\| \) and the stopping criterion for the relative residual 
norm of the linear system can be written as 

\[ \|\mathbf{g}(\mathbf{u}^{(m)}) + \mathbf{K}^{(m)}\mathbf{s}\| \leq \varepsilon^{(m)}\|\mathbf{g}(\mathbf{u}^{(m)})\|. \]

In [31, Chapter 6] the values \( \varepsilon^{(m)} \) are obtained with 

\[ \varepsilon^{(m)} = \gamma\|\mathbf{g}(\mathbf{u}^{(m)})\|^2/\|\mathbf{g}(\mathbf{u}^{(m-1)})\|^2 \]

for a parameter \( \gamma \in (0, 1] \) as 

\[ \varepsilon^{(m)} = \begin{cases} \varepsilon_{\text{max}}, & k = 0 \\ \min(\varepsilon_{\text{max}}, \varepsilon^{(m)}_A), & k > 0, \gamma\varepsilon^{(m-1)}_A \leq 0.1 \\ \min(\varepsilon_{\text{max}}, \max(\varepsilon^{(m)}_A, \gamma\varepsilon^{(m-1)}_A)), & k > 0, \gamma\varepsilon^{(m-1)}_A > 0.1 \end{cases} \]

(13)

for a value \( \varepsilon_{\text{max}} \) that bounds away the sequence \( \varepsilon^{(m)} \) from 1, e.g. \( \varepsilon_{\text{max}} = 0.9 \). See [31, Chapter 6] for more details on these 
choices.

For the last linear system of the non-linear process one may avoid over-solving with equation (6.20) in [31, Chapter 6] 
or one may simply monitor the magnitude of \( \|\mathbf{g}(\mathbf{u}^{(m)})\| \) and switch to a stopping criterion for this \textit{absolute} residual norm 
as soon as it comes close to the desired accuracy. In our experiments, however, for simplicity we did not switch to a new 
strategy for the last linear system of the non-linear process but used (13) throughout the whole process.

**Remark 3.1** A further improvement to minimize the overall numerical costs is related to the solution of Eq.(8). The 
exact fulfillment of the local systems of non-linear equations at each Gauss-point, which represents the integration step of 
the internal variables, is not necessary, since the global iteration loop leads to large residuals. [42] suggest to introduce a 
stopping criterion for the local level iteration scheme, which, of course, should depend on the accuracy of the global level. 
Our investigations within the finite element framework, see [26], show that if there is no additional global Newton-iteration, 
this procedure is more efficient because the number of unknown internal variables \( n_Q \) is frequently larger than the unknown 
nodal displacements \( n_u \). However, one can not exclude circumstances where one or several additional global iterations 
might occur due to the "bad" estimation of the global coefficient matrix in (10) by the term coming from (9). In such cases 
the overall solution time is uncertain and can be larger than the original version of Tab. 1. Thus, we investigate only the 
influence of estimation (13).
3.3 Preconditioners

Apart from physics-based preconditioners, which we will not consider here, the most popular algebraic preconditioning techniques are incomplete factorization and splitting-based factorization. In a splitting-based factorization the system matrix is split in its strict lower triangular, main diagonal and strict upper triangular part as

$$K = L + D + U,$$

where $K$ is the system matrix. Parts of the splitting are then used, in the same way as in stationary iterative methods and often in combination with a relaxation parameter $\omega$, to build a preconditioner. For instance the preconditioner derived from SSOR has the form

$$\frac{\omega}{2 - \omega} \left( L + \frac{D}{\omega} \right) D^{-1} \left( U + \frac{D}{\omega} \right).$$

The computation of these factorizations is virtually for free as the factors are immediately obtained from the splitting. On the other hand, their accuracy as a factorization is rather uncontrollable and more or less a question of good or bad luck. For example, the accuracy of the SSOR-factorization above is the norm of

$$K - \frac{\omega}{2 - \omega} \left( L + \frac{D}{\omega} \right) D^{-1} \left( U + \frac{D}{\omega} \right),$$

which is easily seen to equal

$$\frac{1}{2 - \omega} \left\| (1 - \omega) K - \frac{1 - \omega}{\omega} D - \omega LD^{-1} U \right\|.$$

Clearly, the minimum over all $\omega$ may still be relatively large.

Incomplete factorization offers the possibility to balance between computational costs and accuracy of the factorization. In a simple drop tolerance-based incomplete LU-decomposition, entries are dropped during the factorization process if their norms are smaller than a prescribed tolerance; hence the smaller the tolerance the closer one comes to a complete LU-decomposition. Alternatively, entries can be dropped according to sparsity requirements or according to a combination of tolerance, sparsity and possibly other criteria. The ILU(0)-decomposition requires only that the sparsity pattern of the factorization is the same as that of the original matrix. Note that ILU(0) is one of the many existing incomplete LU-preconditioners; it appears sometimes to be wrongly called the ILU-factorization as if others are not worth considering. Apart from ILU(0) we will also perform our numerical experiments with the ILUT conditioner, which combines dropping tolerances with a prescribed number of allowed fill-in entries per row, [45].

For completeness we also mention a preconditioning technique that is closely related to incomplete LU; we will not use it in the experiments. Approximate inverse factorization computes approximations to the inverses of the triangular factors of an LU-decomposition [6]. Hence the application of such a preconditioner does not consist of back- and forward solves but of direct multiplication with the approximate inverse factors. The computation of an approximate inverse factorization is in general more expensive than incomplete LU-factorization but it can be a much more powerful preconditioner in some applications.

It is not necessary to compute a preconditioner for every system from Eq.(10) of the sequence from scratch. On the contrary, periodic recomputing of a preconditioner and freezing during subsequent systems yields satisfactory convergence speed of the iterative methods. In many examples, it even suffices to compute the preconditioner only once, at the beginning, for a whole sequence of hundreds of linear systems. We observed this in some of our experiments and think it might be related to the fact that we compute a solution to the non-linear systems through extrapolation which is already quite accurate (see (12)).

4 Examples

In the following, the application of iterative and direct solvers is studied by means of various experiments with the two models described in the introduction, i.e. the small strain viscoplasticity model for POM and the finite strain viscoplasticity model for metal plasticity. First, the influence of the local effort (i.e. the effort on element level) on the overall solution time is treated. Second, the adaptive stopping tolerance within the MLNA is investigated. Third, issues concerning computation and periodic recomputation of preconditioners are addressed, and, finally, a comparison of direct and iterative solvers is presented.
4.1 Global/ local computation time

In the first example the influence of the discretization and the material model on the local, global, and total computation time is studied. See Tab. 1 in reference to operations performed on local and global level. This information is crucial because many algorithms, as the chord-MLNA, reduce the global computational cost at the expense of the local computational cost and vice versa.

A monotonous loading, Fig. 1(a), of a plate with hole, Fig. 1(b), is considered. Due to multiple symmetries only one eighth of the body needs to be considered. Note that the body and the boundary conditions are also symmetric in the thickness-direction. This setting serves as a benchmark problem to study various strategies in the following examples. Four different meshes with varying sizes are investigated, Fig. 2. The number of unknown displacements ranges from \( n_u = 5006 \) up to \( n_u = 100520 \) and the corresponding number of internal variables ranges from \( n_Q = 126720 \) up to \( n_Q = 2965248 \).

For the time integration the method of Ellsiepen is used (see, [17, 18]), which is of second order and has two stages. As mentioned in Appendix A the local integration step (8) is very cheap for the POM-model (only function evaluations, see [27]) and expensive in the case of the applied metal plasticity model, where local iterations have to be performed (see [29]). The corresponding computation times are given in Fig. 3. Displayed is the normalized CPU-time over the number of unknown displacements \( n_u \). For the polymer POM of Appendix A.1 the local computation time, see Eqns.(8) and (9), increases linearly with the number of unknowns but the global computation time grows more rapidly and outweights the local computation time, Fig. 3(a). For the large strain metal plasticity model of Appendix A.2 the local computation time increases also linearly. But in this case it is larger than the global computation time of Eq.(10) and dominates the total effort, Fig. 3(b). Reducing the global computation cost has only a minor effect in this case. Therefore, it has to be borne in mind that the weight of overall improvement achieved by linear solvers is influenced by the constitutive model, i.e. the “stress algorithm” on Gauss-point level.

4.2 Adaptive stopping tolerance for iterative solvers

In this subsection the usage of the adaptive stopping tolerance for the iterative solver is investigated (here, we make use of an ILUT-preconditioning for each linear system). As mentioned in Sec. 3.2, at the beginning of the Multilevel-Newton iteration the solution of the linear system of equations is not required to be very accurate, but the accuracy has to increase when coming closer to the sought-after solution. Here, the viscoplasticity model of POM is used, see Sec. A.1. The total time is decomposed into eleven time steps consisting of two time stages each (integration with the method of Ellsiepen).

As initial value for the stopping tolerance, \( \varepsilon_{\text{max}} = 10^{-3} \) in (13) proved its worth. Using the value \( \varepsilon_{\text{max}} = 0.25 \) or even \( \varepsilon_{\text{max}} = 0.5 \) as suggested in [31] led to more global iterations at the beginning of the Newton process and thus spoiled the savings of the iterative solver. The second parameter, which serves to compute a measure of the degree to which the non-linear iteration approximates the solution, is set to \( \gamma = 0.9 \).

We consider again the monotonous loading path in Fig. 1(a) applied to the plate with a hole, Fig. 2(d), having \( n_u = 100520 \) unknown displacement degrees of freedom. In Fig. 4(a) the stopping tolerance \( \varepsilon \) is given over the Newton-iteration number. In the standard method the stopping tolerance is fixed to \( \varepsilon = 10^{-6} \). When using the adaptive stopping tolerance the initial value is \( \varepsilon = 10^{-3} \). This value is used for the first Newton iteration. Then the stopping tolerance is reduced eventually \( \varepsilon = 10^{-6} \). After the fourth MLNA-iteration the solution of the non-linear system is found and time is
advanced to the next time stage yielding a new non-linear system of equations. The stopping tolerance is reset to $\varepsilon = 10^{-3}$ and the process starts again. The reason for using an adaptive stopping tolerance is to reduce the number of iterations of the Krylov solver per Newton iteration. This number is displayed over the Newton-iteration number in Fig. 4(b). Using the adaptive stopping tolerance reduces the number of iterations of the linear solver, particularly, at the beginning of each Newton-process. It can also be seen from the graph that the total number of Newton-iterations increases slightly when the adaptive stopping tolerance is used. Here, the standard (constant) stopping tolerance led to a total of 82 non-linear iterations.
while the adaptive stopping tolerance needed 94 iterations. On the other hand the number of iterations of the BiCGSTAB solver was reduced from 6350 (standard) to 5384 (adaptive), even though more linear systems had to be solved. Comparing the computation times, the total saving when using the adaptive stopping tolerance is in this example about 14%.

Remark 4.1 The application of the adaptive stopping tolerance in the case of the finite strain viscoplasticity model of Appendix A.2 does not show any advantages, because the increase of global Newton-iteration steps is computationally very costly due to the expensive computation of the tangential stiffness matrix (coefficient matrix) in Eq.(10).

4.3 Precondition strategy

Computing the preconditioner for the iterative solver of the linear system can consume a significant amount of the total solution time of the linear system. As mentioned in Sec. 3.3 it is not necessary to recompute the preconditioner for every system but the preconditioner can be kept constant during the Newton-process or even over some time interval of the time integration. Before we present results with periodically computed preconditioners, we describe and legitimate the choice of the preconditioner type.

In all computations the ILUT incomplete LU-decomposition with dual-threshold strategy (see [45]) is used as preconditioner. Gauss-Seidel and SSOR preconditioners (see Sec. 3.3) have proved too weak for the problems at hand in terms of convergence (they led to ten respectively five times more BiCGSTAB-iterations than using the ILUT preconditioner.) Therefore, despite the negligible costs to compute these preconditioners, they were not considered in the following. To determine the parameters lfil (level of fill-in) and droptol (dropping tolerance) a pretest is performed with the BiCGSTAB solver on an example linear system. This system comes from the first Newton-step within the solution of the monotonous loading of the plate with hole (material model POM and \( n_u = 100520 \) unknown displacement degrees of freedom). In Fig. 5 the computation time is given over the dropping tolerance for four different numbers of fill-in. In Fig. 5(a) the total computation time to solve the linear system, consisting of a preconditioner computation plus a BiCGSTAB run, is plotted. The plot indicates that the level of fill-in has a minor influence and suggests using a rather rough dropping tolerance. In Fig. 5(b) only the computation time of the iterative solver is considered. This plot shows that the computation time decreases with smaller drop tolerance and reaches a minimum at about droptol = \( 10^{-4} \). On the other hand the computation time rises quickly for a drop tolerance greater than \( 10^{-2} \). Since the preconditioner is to be reused and to stay away from the large increase of computation time, we choose the parameters lfil=6 and droptol=\( 10^{-4} \). Note that Figures 5(a) and 5(b) indicate that the „classical“ ILU(0) decomposition, with parameters lfil=0 and droptol=0, is not efficient and that it is necessary to consider more sophisticated ILU-decompositions for satisfactory convergence speed. For completeness, we also display a comparison of the computational costs of the preconditioner and the iterative solver in dependence of the number of unknowns, see Fig. 6(a). Again the computation time of a single system coming from a Newton-step as described above is considered (now for the four different meshes of Fig. 2).

Next, results with periodic preconditioner recomputation with two different periods are presented: recomputation for every system, and utilizing a constant preconditioner. The number of iterations needed to solve each linear system (\( n_u = 100520 \) unknown displacement degrees of freedom). In Fig. 5 the computation time is given over the dropping tolerance for four different numbers of fill-in. In Fig. 5(a) the total computation time to solve the linear system, consisting of a preconditioner computation plus a BiCGSTAB run, is plotted. The plot indicates that the level of fill-in has a minor influence and suggests using a rather rough dropping tolerance. In Fig. 5(b) only the computation time of the iterative solver is considered. This plot shows that the computation time decreases with smaller drop tolerance and reaches a minimum at about droptol = \( 10^{-4} \). On the other hand the computation time rises quickly for a drop tolerance greater than \( 10^{-2} \). Since the preconditioner is to be reused and to stay away from the large increase of computation time, we choose the parameters lfil=6 and droptol=\( 10^{-4} \). Note that Figures 5(a) and 5(b) indicate that the „classical“ ILU(0) decomposition, with parameters lfil=0 and droptol=0, is not efficient and that it is necessary to consider more sophisticated ILU-decompositions for satisfactory convergence speed. For completeness, we also display a comparison of the computational costs of the preconditioner and the iterative solver in dependence of the number of unknowns, see Fig. 6(a). Again the computation time of a single system coming from a Newton-step as described above is considered (now for the four different meshes of Fig. 2).
Fig. 5 Influence of droptol and lfil on computation times

100520 (within the Multilevel-Newton iteration) is displayed for two different periods in Fig. 6(b). When working with the constant preconditioner, i.e., computing the preconditioner only once at the beginning of the simulation, the number of Krylov subspace iterations increases with time and the algorithm can even break down. Nevertheless more than half of the total computation time is saved. Recomputing the preconditioner every forty-fifth system yields a comparable number of iterations as when recomputing the preconditioner for every system. This strategy saves 58% of the total computation time. In other words, the recycled preconditioner is as powerful as the preconditioner from scratch during the MLNA iterations and is insensitive to several time-steps. This makes additional updating of the recycled preconditioner, with techniques mentioned in Section 3.1, unnecessary.

Remark 4.2 The application of the preconditioner periodic recycling technique is possible for the finite strain viscoplasticity model as well. However, in this case, where on Gauss-point level 12 equations with 12 unknowns have to be
computed, the reduction of the computational costs does not lead to an essential improvement since the local computation dominates the problem. For problems where an analytical reduction of the number of unknowns is possible, see, for example, [34], the use is recommended.

4.4 Comparison of direct and iterative solver

Finally, the performances of direct and iterative solvers for sparse linear systems are compared for the plate with hole of Fig. 1(b) described with the material model of POM and subjected to the loading of Fig. 1(a). The computation times are given for two direct solvers, UMFPACK 5.03 and PARDISO 3.2, and for the iterative solvers BiCGSTAB and GMRES (both modified from Youcef Saad’s Sparskit2). For the BiCGSTAB and the GMRES solvers the incomplete LU factorization with threshold and parameters as described in Sect. 4.3 are used. A restarted version of the GMRES solver is used as described in Sect. 3.1.2. Here the restart parameter is set to ten. To obtain high performance for the iterative solvers the preconditioner is computed periodically with a period of 40 linear systems and the adaptive stopping criteria is used with the parameters $\gamma = 0.9$, $\varepsilon_{\text{max}} = 10^{-3}$ and $\varepsilon_{\text{min}} = 10^{-6}$. Similarly, the user-specific parameters in the direct solver packages where tuned to obtain optimal performance. As can be seen from the graphs of Fig. 7, the iterative solvers outperform the direct ones when solving large systems. In particular, the BiCGSTAB solver performs very well and is superior as soon as systems have more than about 20000 unknowns. For GMRES the superiority comes somewhat later, but still the slope of the curve for GMRES is clearly less steep than for the LU-packages. Note that the condition numbers (in the 1-norm) of the initial linear systems of the considered sequences are already of the order $10^5$; this does not seem to have a negative influence on the performance of the iterative linear solvers.

5 Conclusions

In this article several aspects of accelerating the computations which are related to the application of iterative linear solvers within implicit finite elements based on constitutive models of evolutionary-type are studied. First of all, one can state that only in the case of computational cheap local effort due to the time-integration procedure of the internal variables one can expect a large profit using iterative linear solvers. In this case up to 80% of the total computation time is spend by the solver of the linear system. Iterative solvers depend on a user-prescribed stopping tolerance and we found out that an adaptive tolerance can lead to savings of more than 15%. The main improvement is given by reducing the costs of the preconditioner. Although it is common to compute the preconditioner each time before the iterative solver is started, one can freeze the preconditioner for a larger number of solution calls. It turns out that there is a certain insensitiveness so that in some problems it is sufficient to compute the preconditioner only once at the very first time-step. Even if the preconditioner is computed for every fortieth call of the linear solver one can save an overall computational time of more than 50% in our application. Finally, it turns out that in modern three-dimensional computations PARDISO is (at the moment) an efficient direct solver. However, iterative solvers such as GMRES or BiCGSTAB with ILUT-preconditioning are faster even for moderately sized problems where PARDISO is not disqualified because of storage costs.

Fig. 7 Comparison of direct and iterative solvers
A Constitutive models

A.1 Small strain viscoplasticity for POM

A small strain viscoplasticity model which was developed in [27] for polyoxymethylene is based on an additive decomposition of the linearized strain tensor $\mathbf{E}$ into an elastic and a viscous part, $\mathbf{E} = \mathbf{E}^e + \mathbf{E}^s$, $\mathbf{E} = \mathbf{E}^e + \mathbf{E}^s$. Additionally, the stress-state $\mathbf{T} = \mathbf{T}^e + \mathbf{T}^h$ is assumed to decompose into an equilibrium and an overstress part, $\mathbf{T} = \mathbf{T}^e + \mathbf{T}^h$, where the equilibrium stress part consists of an elastic, $\mathbf{T}^e = \mathbf{T}^e \mathbf{T}$, and a hysteretic part, $\mathbf{T}^h = \mathbf{T}^h \mathbf{T}$. The elastic equilibrium and the overstress part $\mathbf{T}^o$ are defined by elasticity relations

$$
\mathbf{T}^e = \mathbf{K}_0 (\text{tr} \mathbf{E}^e) \mathbf{I} + 2G_0 \mathbf{E}^e, \quad \mathbf{T}^h = \mathbf{T}^h \mathbf{T},
$$

(14)

$$
\mathbf{T}^e = (\mathbf{K}_T f_T (\mathbf{I}_E) + \mathbf{K}_C f_C (\mathbf{I}_E)) \mathbf{I} + G(\mathbf{I}_E) (f_T (\mathbf{I}_E) + \beta f_C (\mathbf{I}_E)) \mathbf{E}^D
$$

(15)

with both the deformation-dependent shear modulus $G(\mathbf{I}_E)$ and the smoothing functions $f_T (\mathbf{I}_E)$ as well as $f_C (\mathbf{I}_E)$

$$
G(\mathbf{I}_E) = \frac{\alpha_1}{\alpha_2 + \sqrt{\mathbf{I}_E}}, \quad f_T (\mathbf{I}_E) = \frac{1}{2} (1 + \tanh(a \mathbf{I}_E)), \quad f_C (\mathbf{I}_E) = \frac{1}{2} (1 - \tanh(a \mathbf{I}_E)).
$$

Here, the invariants $\mathbf{I}_E = \text{tr} \mathbf{E}$ and $\mathbf{J}_E = \mathbf{E}^D$. $\mathbf{E}^D$ are defined, where $\text{tr} \mathbf{E} = E^k_k$ symbolizes the trace operator, and the dot denotes the inner product, $\mathbf{A} \cdot \mathbf{B} = a_i^j b^i_j$. The superscript $D$ defines the deviator operator $\mathbf{A}^D = \mathbf{A} - (\text{tr} \mathbf{A})/3 \mathbf{I}$, where $\mathbf{I}$ is the second order identity tensor. Furthermore $a$ is a smoothing parameter and $\mathbf{K}_0$, $G_0$, $\mathbf{K}_T$, $\mathbf{K}_C$, $\alpha_1$, and $\alpha_2$ are material parameters which have to be adapted to experimental data, see [27]. The smoothing functions are introduced to represent the tension-compression asymmetry occurring in the experiments, controlled by the material parameter $\beta$. The remaining viscous strains $\mathbf{E}^s$ and the hysteretic part of the equilibrium stresses $\mathbf{T}^h$ are given by evolutions equations, i.e. ordinary differential equations of first order:

$$
\dot{\mathbf{T}}^h = c \dot{\mathbf{E}}^D - b s \mathbf{T}^h, \quad \dot{\mathbf{E}} = \frac{\beta_2 \| \dot{\mathbf{E}} \|^2 j + 1}{\eta_0 (1 + \beta_1 \| \dot{\mathbf{E}} \| / J)} \mathbf{T}^o
$$

(16)

(17)

Table 2 Material parameters of small strain viscoplasticity model identified in [27]

<table>
<thead>
<tr>
<th>$K_T$</th>
<th>$K_C$</th>
<th>$\alpha_1$</th>
<th>$\alpha_2$</th>
<th>$\beta_1$</th>
<th>$\beta_2$</th>
<th>$\beta_3$</th>
<th>$\eta_0$</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPa</td>
<td>MPa</td>
<td>MPa</td>
<td>/</td>
<td>l</td>
<td>l</td>
<td>l</td>
<td>MPa</td>
</tr>
<tr>
<td>7200</td>
<td>7200</td>
<td>35</td>
<td>0.04</td>
<td>1.8</td>
<td>200</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>MPa</td>
<td>MPa</td>
<td>MPa</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>700</td>
<td>10$^7$</td>
<td>7 $\times$ 10$^7$</td>
<td>0.96</td>
<td>10</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

conclusion, the constitutive model fits into the structure

$$
\mathbf{T} = h(\mathbf{E}, \mathbf{q}), \quad \dot{\mathbf{q}} = r(\mathbf{E}, \dot{\mathbf{E}}, \mathbf{q}),
$$

(18)

(19)

with $\mathbf{q} = \{ T^h, \dot{\mathbf{E}} \} \in \mathbb{R}^{12}$ in the three-dimensional case, i.e. the stress state is defined by the current state of deformation and the hardening state, where for the latter ordinary differential equations are defined.

Although the constitutive model is non-linear, a resulting diagonally-implicit Runge-Kutta step for constant strains leads to pure function evaluations since the internal variables occur linearly, see [27]. Thus, on Gauss-point level there are no iterations (see function $\mathbf{L}$ in Eq.(8) of Tab. 1), i.e. only function evaluations have to be performed on Gauss-point level.
A.2 Finite strain viscoplasticity for metal plasticity

A much more complicated model occurs in the field of finite strain viscoplasticity. Here, we assume according to the proposals of [33] and [52] a multiplicative decomposition of the deformation gradient into an elastic and a viscous part, where the latter is additionally decomposed multiplicatively into an energy storing and a dissipative part: \( \mathbf{F} = \hat{\mathbf{F}}_e \mathbf{F}_p = \hat{\mathbf{F}}_p \hat{\mathbf{F}}_e \). Based on this decomposition strain tensors relative to the inelastic intermediate configurations are formulated. Relative to the plastic intermediate configuration an elasticity relation is formulated, which can be expressed by quantities relative to the reference configuration. The kinematic hardening behavior is modeled by strain-like quantities, where an additional elasticity relation defines a backstress tensor occurring in a von Mises yield function. Both the plastic and the energy storing intermediate configurations are controlled by flow rules for strain tensors. Relative to the reference configuration they can be expressed by the plastic right Cauchy-Green tensor \( \mathbf{C}_p \) and the quantity \( \mathbf{C}_t = \mathbf{F}_e^T \mathbf{F}_e \). The extension to viscoplasticity is done by a Perzyna-type model. In Tab. 3 the model is summarized showing that 12 internal variables control the inelastic deformation process, where \( \mathbf{S} = (\det \mathbf{F})^{-1} \mathbf{T} \mathbf{F}^{-T} \) is the 2nd Piola-Kirchhoff tensor. \( \mathbf{T} \) designates the Cauchy-stress tensor. The aspect of integration, particularly, the incorporation of the plastic incompressibility condition is discussed in [29]. The chosen material parameters are depicted in Tab. 4. The integration step of the internal variables is done iteratively on Gauss-point level by solving 12 unknowns (and one for the yield function) if \( \mathbf{F} < 0 \) holds. Accordingly, the numerical effort is much larger than in the POM model.

Table 3 Constitutive model expressed with quantities relative to the reference configuration

<table>
<thead>
<tr>
<th>loading condition</th>
<th>( F &lt; 0 )</th>
<th>( F &gt; 0 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>elasticity relation</td>
<td>( \mathbf{S} = K (J-1) J_0^{-1} + 2 c_{10} J^{-2/3} \left( \mathbf{C}_p^{-1} - \frac{1}{3} \text{tr} (\mathbf{C}_p^{-1}) \mathbf{C}^{-1} \right) )</td>
<td>( \dot{\mathbf{C}}_p = 0 )</td>
</tr>
<tr>
<td>flow rule 1</td>
<td>( \dot{\mathbf{C}}_p = \frac{\alpha}{2} \left( c_3 \mathbf{S} - \mathbf{C}_p \right) \mathbf{C}_p^{-1} )</td>
<td>( \dot{\mathbf{C}}_t = \lambda \sqrt{\frac{3}{2}} \left( \mathbf{C}_t \mathbf{Z} - (\text{tr} (\mathbf{Z}_t)) / 3 \mathbf{C}_t \right) )</td>
</tr>
<tr>
<td>flow rule 2</td>
<td>( \dot{\mathbf{C}}_t = 0 )</td>
<td>( \dot{\mathbf{C}}_t = \lambda \sqrt{\frac{3}{2}} \left( \mathbf{C}_p \mathbf{Z} \mathbf{C}_p^{-1} - (\text{tr} (\mathbf{Z}_p)) / 3 \mathbf{C}_p \right) )</td>
</tr>
<tr>
<td>abbreviations</td>
<td>( \mathbf{Z} = \frac{1}{2} \left( (c_1 - \frac{1}{3}) (3 - \text{tr} (\mathbf{C}_p^{-1})) \right) \mathbf{C}_p^{-1} + \frac{1}{2} \left( \mathbf{C}_p^{-1} - \mathbf{C}_p^{-1} \mathbf{C}_p \mathbf{C}_p^{-1} \right) )</td>
<td>( F = \left( \frac{1}{2} \left( (c_3 \mathbf{S} \mathbf{C}_p^{-1} - \mathbf{C}_p) \cdot (\mathbf{S} - \mathbf{C}_p^{-1} \mathbf{C}_p \mathbf{C}_p^{-1}) - \frac{1}{2} \text{tr} (\mathbf{S} - \mathbf{C}_p \mathbf{C}_p^{-1} \mathbf{C}_p \mathbf{C}_p^{-1}) \right) - k )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \alpha = F + k ), ( \lambda = 1/\eta (F/\sigma_0)^{\gamma} )</td>
</tr>
</tbody>
</table>

Table 4 Material parameters of finite strain viscoplasticity model

<table>
<thead>
<tr>
<th>( K )</th>
<th>( \mu )</th>
<th>( k )</th>
<th>( c_1 )</th>
<th>( c_2 )</th>
<th>( \beta_1 )</th>
<th>( r )</th>
<th>( \sigma_0 )</th>
<th>( \eta )</th>
</tr>
</thead>
<tbody>
<tr>
<td>166666.67</td>
<td>76923.08</td>
<td>200</td>
<td>66666.67</td>
<td>20000</td>
<td>0.0025</td>
<td>1.0</td>
<td>1.0</td>
<td>0.1</td>
</tr>
</tbody>
</table>

References


Copyright line will be provided by the publisher