Dynamic Matrix Compression: A New Perspective on Two-Phase Flows

Mia Ohlrogge[1,2][0009−0007−7816−600X], Torsten F. Bosse[1][0000−0002−4894−0742], and H. Martin Bücker[1,3][0000−0002−5210−0789]

[1] Institute for Computer Science, Friedrich Schiller University Jena, Jena, Germany
[2] Faculty of Engineering Sciences, University of Innsbruck, Innsbruck, Austria
[3] Michael Stifel Center Jena for Data-driven and Simulation Science, Friedrich Schiller University Jena, Jena, Germany

Abstract. Simulations to predict two-phase flows in porous media require the solution of large systems of nonlinear equations, typically addressed using a Newton-type method. A major computational cost in this process is often the assembly, storage, and evaluation of the underlying sparse Jacobian matrices. It is well-known that automatic differentiation is capable of exploiting this sparsity structure by first representing the Jacobian by a matrix compression from which the Jacobian is then reconstructed. This work introduces a novel matrix compression strategy that enables the efficient construction of a sequence of Jacobian matrices with varying sparsity patterns. In contrast to a standard approach which, in a static fashion, computes a matrix compression from scratch at every Newton iteration, the new dynamic strategy reduces the computational effort by updating all sparsity-exploiting data structures incrementally. For an immiscible flow of two fluids moving through the pore space without mass exchange, the resulting dynamic approach is shown to speed up the construction of matrix compressions by a factor of up to 400.

Keywords: Two-Phase Flow · Automatic Differentiation · Sparse Matrices · Matrix Compression

1 Introduction

Modeling two-phase flow in porous media is fundamental to understanding a range of subsurface processes, including groundwater transport [5], hydrocarbon recovery [13], geological carbon storage [1], and geothermal energy [10]. Accurately predicting these flows requires solving highly nonlinear partial differential equations that couple mass and momentum conservation for multiple fluid phases within heterogeneous geological formations. After spatial and temporal discretization, the governing equations result in large, sparse, and strongly nonlinear algebraic systems [8]. The solution of these systems is frequently based on linearization using a Newton-type iterative method. In each Newton iteration, a system of linear equations is solved whose coefficient matrix is a large, sparse Jacobian matrix representing the sensitivities of the residual with respect to the unknown physical quantities such as pressure and saturation.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

Rather than relying on approximations of derivatives, techniques of automatic differentiation (AD) [7] are free from any truncation error offering the advantage of increased accuracy. Besides reducing human effort for deriving and implementing analytic derivatives, they are also particularly suitable for exploitation of sparsity [6, 4]. In particular, AD is capable of reducing time and memory of Newton iterations by carefully partitioning the columns of a sparse Jacobian into groups of structurally orthogonal columns, a process referred to as *matrix compression*. Today, the state-of-the-art technique for matrix compression is to partition the columns statically. That is, given a sparsity pattern of the current iteration, a matrix compression is computed via AD for each iteration from scratch.

However, the sparsity patterns often do not change significantly from one iteration to the next. Therefore, we propose a dynamic approach for matrix compression that updates the matrix compression of the previous iteration efficiently. The key idea behind the new approach is to overestimate the sparsity pattern of the current iteration by taking into account the knowledge of the previous iteration. If the overestimation stays reasonably moderate throughout the whole fluid flow simulation, a dynamic matrix compression can reduce the running time significantly at the cost of increasing the storage requirement only slightly. The new strategy is put into practice in a two-phase flow simulation, but is also relevant in other areas of computational science.

The organization of the article is as follows. In Sect. 2, the mathematical model representing two-phase flow is briefly reviewed. Its discretization using the finite volume method is then sketched in Sect. 3. The standard (static) approach based on sparsity-exploiting techniques of AD is summarized in Sect. 4. The novel dynamic approach is presented in Sect. 5 and results of numerical experiments are reported in Sect. 6.

2 Mathematical Model for Two-Phase Flow

Consider an immiscible flow in a porous medium, where two fluids move simultaneously through the pore space without mass exchange between them. The wetting fluid such as water preferentially wets the solid matrix and is denoted by w. The non-wetting fluid such as oil is denoted by o. The *mass conservation* for a phase $a \in \{w, o\}$ is represented by

$$\nabla \cdot (\rho_a \mathbf{v}_a) + \frac{\partial u_a}{\partial t} = q_a, \tag{1}$$

where ρ_a is the fluid density and $\mathbf{v}_a$ is the Darcy velocity of phase a. The term $\nabla \cdot (\rho_a \mathbf{v}_a)$ represents the divergence of the mass flux, quantifying the net outflow or inflow due to fluid motion. The phase mass per unit volume $u_a = \phi S_a \rho_a$ is a conserved quantity, with ϕ describing the porosity of the medium and S_a the saturation. Thus, (1) reflects how the divergence of the mass flux plus the rate of change of fluid mass stored in the pores with respect to time t must balance any external sources or sinks, represented by the symbol q_a; see [8] for more details.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

The conservation equations (1) involve the unknown phase velocities $\mathbf{v}_a$ and fluid densities ρ_a, from which all other quantities can be determined. To close the system of equations, a link between $\mathbf{v}_a$ and pressure p and gravity g is necessary in the form of a constitutive relation. Let $\mathbf{g}$ denote the gravitational acceleration vector, say, $\mathbf{g} = (0, 0, -g)^T$. Then, the relevant relation is known as Darcy's law

$$\mathbf{v}_a = -\lambda_a \mathbb{K}(\nabla p_a - \rho_a \mathbf{g}), \tag{2}$$

where $\lambda_a = k_{ra}(S_a)/\mu_a$ is the mobility of phase a in which the symbol k_{ra} denotes the relative permeability as a function of saturation and μ_a is the dynamic fluid viscosity. Furthermore, $\mathbb{K}$ is the tensor of absolute permeabilities and ∇p_a is the pressure gradient [2].

Coupling (1) with (2) yields a system of nonlinear partial differential equations on a domain containing four unknowns, S_w, S_o, p_w, and p_o. Assuming no capillary forces and neglecting dispersion, the pressures of both phases are identical, $p = p_w = p_o$, and the saturations satisfy $S_w + S_o = 1$. Choosing p and S_w as primary variables, (1) and (2) reduce to

$$-\nabla \cdot \left[\rho_w \frac{k_{rw}}{\mu_w} \mathbb{K}(\nabla p - \rho_w \mathbf{g}) \right] + \frac{\partial}{\partial t} (\phi S_w \rho_w) = q_w, \tag{3}$$

$$-\nabla \cdot \left[\rho_o \frac{k_{ro}}{\mu_o} \mathbb{K}(\nabla p - \rho_o \mathbf{g}) \right] + \frac{\partial}{\partial t} (\phi (1 - S_w) \rho_o) = q_o. \tag{4}$$

The mathematical model employs physically consistent boundary and initial conditions. Injection boundaries specify an inflow rate (Neumann condition) on the inlet boundary, representing water injection. Pressure boundaries prescribe a reference pressure (Dirichlet condition) at production wells or open boundaries. No-flow boundaries impose zero normal flux on impermeable surfaces. Initial conditions define the starting distributions of pressure and saturation throughout the domain. A detailed derivation of this model is given in [8].

3 Numerical Methods

Equations (3) and (4) are discretized in space and time as follows. Space discretization is based on the finite volume method, in which each control volume represents a cell where the conservation law is enforced in integral form. Applying Gauss's theorem converts the volume integrals into surface fluxes between neighboring cells. Here, the phase mass per unit volume u_a as well as the source term q_a are assumed to be constant within each control volume V_i. As outlined in [12], this assumption for a cell i allows to simplify the mass balance over V_i to

$$|V_i| \frac{\partial u_{a,i}}{\partial t} + \sum_{j \in \mathcal{H}(i)} f_{a,ij} = |V_i| q_{a,i},$$

where the notation $j \in \mathcal{H}(i)$ indicates that a cell j is a neighbor of a cell i and the symbol $f_{a,ij}$ denotes the flux from a cell i to a neighboring cell j. Here, the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

sum $\sum_{j\in\mathcal{H}(i)} f_{a,ij}$ represents the net flux of phase a across all faces of a cell i. The two-point flux approximation (TPFA) [11] is used to compute intercell fluxes based on pressure differences and upwinded phase mobilities as

$$f_{a,ij} = -\rho_a T_{ij} \lambda_{a,ij} \big(p_{a,i} - p_{a,j} - \rho_a g(z_i - z_j)\big),$$

where T_{ij} is the transmissibility across the interface, $\lambda_{a,ij}$ is the upwinded phase mobility at the interface, $p_{a,i}$ and $p_{a,j}$ are the phase pressures at the cell centers, and z_i and z_j are the elevations of the cell centers. The transmissibility T_{ij} represents the ease of fluid flow between cells. The spatial difference $z_i - z_j$ arises from discretizing the gravitational contribution in Darcy's law along the line connecting the centers of cells i and j; see [12] for details.

The total flux $f_a(\mathbf{x})$ across the boundary of the control volume is expressed in terms of the primary variables, p and S_w, collected in a global state vector

$$\mathbf{x} = \big(p_1, p_2, \ldots, p_N, S_{w,1}, S_{w,2}, \ldots, S_{w,N}\big)^\top \in \mathbb{R}^{2N},$$

where $N \in \mathbb{N}$ is the number of grid cells.

Time integration uses the fully implicit (backward Euler) scheme for stability in stiff, multiphase systems. For each time step $t \in \mathbb{N}$ of length $0 < \delta_t \in \mathbb{R}$, the discrete residual in control volume i for phase a is given by

$$F_{a,i}(t, \mathbf{x}^{[t]}, \mathbf{x}^{[t+1]}) := \frac{u_{a,i}^{t+1} - u_{a,i}^{t}}{\delta_t} + \frac{1}{|V_i|} \sum_{j\in N(i)} f_{a,ij}^{t+1} - q_{a,i}^{t+1},$$

where $\mathbf{x}^{[t]}$ and $\mathbf{x}^{[t+1]} \in \mathbb{R}^{2N}$ are the state vectors of primary variables at time step t and $t+1$, respectively. Assembling the residuals over all cells and both phases yields the system of nonlinear equations

$$\mathbf{F}(t, \mathbf{x}^{[t]}, \mathbf{x}^{[t+1]}) = \begin{pmatrix} F_{w,1}(t, \mathbf{x}^{[t]}, \mathbf{x}^{[t+1]}) \\ \vdots \\ F_{w,N}(t, \mathbf{x}^{[t]}, \mathbf{x}^{[t+1]}) \\ F_{o,1}(t, \mathbf{x}^{[t]}, \mathbf{x}^{[t+1]}) \\ \vdots \\ F_{o,N}(t, \mathbf{x}^{[t]}, \mathbf{x}^{[t+1]}) \end{pmatrix} = \mathbf{0} \in \mathbb{R}^{2N}, \tag{5}$$

which, in each time step $t+1$, is solved by Newton's method for the state $\mathbf{x}^{[t+1]}$.

4 Static Matrix Compression

Consider a standard approach at a given time step $t+1$ to solve the nonlinear system (5) for the state $\mathbf{x} := \mathbf{x}^{[t+1]}$ using Newton's method [9]. Starting from an initial guess $\mathbf{x}$ for $\mathbf{x}^{[t+1]}$, it generates a sequence of linear systems of the form

$$J(\mathbf{x})\Delta\mathbf{x} = \mathbf{b}(\mathbf{x}), \tag{6}$$

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

where the $2N \times 2N$ Jacobian matrix and the right-hand side given by

$$J(\mathbf{x}) := \partial \mathbf{F}(t, \mathbf{x}^{[t]}, \mathbf{x})/\partial \mathbf{x} \qquad \text{and} \qquad \mathbf{b}(\mathbf{x}) := -\mathbf{F}(t, \mathbf{x}^{[t]}, \mathbf{x})$$

do not only depend on the state $\mathbf{x}$ but also on the state $\mathbf{x}^{[t]}$ from the previous time step. In each iteration of the Newton iteration, the solution $\Delta\mathbf{x}$ of the linear equations (6) is then used to update the current approximation $\mathbf{x}$ by $\mathbf{x} \leftarrow \mathbf{x}+\Delta\mathbf{x}$. Once the Newton iteration is converged to a solution $\mathbf{x}$ for (5), this solution is set to the state at the next time step $t+1$, namely, $\mathbf{x}^{[t+1]} \leftarrow \mathbf{x}$.

Each Newton step requires the solution of a linear system (6), where the coefficient matrix is a large, sparse, and non-symmetric Jacobian

$$J = \begin{bmatrix} \frac{\partial \mathbf{F}_w}{\partial \mathbf{p}} & \frac{\partial \mathbf{F}_w}{\partial \mathbf{S}_w} \\ \frac{\partial \mathbf{F}_o}{\partial \mathbf{p}} & \frac{\partial \mathbf{F}_o}{\partial \mathbf{S}_w} \end{bmatrix}$$

whose 2×2 block structure reflects the coupling between the pressure and saturation variables for the two fluid phases.

The Jacobian is accurately and efficiently computed by AD. These techniques transform a given program representing a mathematical function $\mathbf{F} : \mathbb{R}^n \to \mathbb{R}^n$ into another program capable of evaluating the function and a Jacobian-vector product $J\mathbf{z}$ for a given direction $\mathbf{z} \in \mathbb{R}^n$ simultaneously in a single *AD pass* [7]. The $n \times n$ Jacobian can then be evaluated by choosing the direction $\mathbf{z}$ successively as the Euclidean unit vector $\mathtt{e}_i \in \mathbb{R}^n$ for $i = 1, 2, \ldots, n$ requiring n AD passes. However, when the Jacobian J is sparse and its sparsity pattern $\mathtt{J}$ is known from the discretization, AD can exploit this structural information and reduce the number of AD passes and the storage requirement by matrix compression [7].

To illustrate the rationale behind matrix compression, assume that the columns J_i of the Jacobian J are partitioned into c distinct groups of pairwise *structurally orthogonal* columns. Here, two columns J_i and J_j are called structurally orthogonal, denoted by the notation

$$J_i \perp J_j,$$

if there is no row in which their sparsity patterns $\mathtt{J}_i$ and $\mathtt{J}_j$ are both nonzero. A partitioning of n columns into a set $\mathcal{C} = \{1, 2, \ldots, c\}$ of disjoint and nonempty groups is represented by a vector

$$\gamma = (\gamma(1), \gamma(2), \ldots, \gamma(n))^T \in \mathcal{C}^n,$$

called *grouping*, where $\gamma(i) = g$ indicates that column i is assigned to group g.

Performing a logical OR operation over all directions given by the Euclidean unit vectors $\mathtt{e}_i$ associated to a group $g \in \mathcal{C}$ results in a binary vector

$$\mathtt{S}_g = \bigvee_{\substack{\text{column index } i \\ \text{s.t. } \gamma(i)=g}} \mathtt{e}_i \quad \in \{0,1\}^n$$

enabling the computation of the nonzero entries of all columns belonging to this group g in one single AD pass. For each of the c groups $g \in \mathcal{C}$, we collect the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

vector $\mathtt{S}_g$ as a column g of a binary *seed pattern* $\mathtt{S} \in \{0,1\}^{n \times c}$. Then, AD only requires c passes to compute the *compressed Jacobian*

$$J^{\text{cmp}} = J \cdot \mathtt{S} \in \mathbb{R}^{n \times c}, \tag{7}$$

where this Jacobian-matrix product is computed in a matrix-free fashion, meaning without explicitly assembling the Jacobian J. This property motivates compression strategies that seek to minimize the number of groups c.

Given a sequence of Jacobian sparsity patterns, a matrix compression needs to be recomputed whenever a pattern changes. If this recomputation is carried out each time from scratch, we call this approach *static* matrix compression.

5 Dynamic Matrix Compression

In transient simulations, the Jacobian pattern may vary not only between successive time steps but also across Newton iterations within a single time step. Statically recomputing a grouping at every iteration may therefore be computationally expensive. To mitigate this cost, we propose an incremental strategy that updates an existing grouping by reusing information on the grouping from the previous iteration. This dynamic strategy is based on an approximation `A` that overestimates the pattern `J` for the Jacobian J. Here, a sparsity pattern `A` is said to overestimate another pattern `J` if the pattern `A` contains a nonzero at least for every nonzero entry of `J` at the corresponding position. The overestimation property guarantees that any valid grouping for an overestimator `A` into groups of pairwise structurally orthogonal columns is also a valid grouping of the columns of `J`. As a result, any seed pattern `S` that is constructed from the grouping of `A` is also a valid seed pattern for `J`. Thus, a seed pattern `S` constructed from an overestimator is appropriate for computing the compressed representation (7) in a matrix-free fashion, from which the Jacobian J is recovered.

An algorithmic description of the overall simulation for the two-phase flow problem incorporating the dynamic strategy is provided in Algorithm 1. It illustrates the time integration for the flow simulation, employing Newton methods at each time step. The main distinction between a dynamic and a static approach, which always recomputes the grouping and seed pattern from scratch by existing sparse AD techniques [6], is reflected by the methods in lines 4, 13, and 14. These methods encapsulate the core idea of the proposed dynamic strategy.

Specifically, the first method, `InitMatrixCompression`, computes an initial overestimator `A` of the sparsity pattern of the Jacobian matrix J evaluated at the input $\mathbf{x}$, for example, $\mathtt{A} = \mathtt{J}$. Also, it determines an initial grouping γ for the overestimator `A` and a corresponding seed pattern `S`, together with an overestimator `C` for the sparsity pattern of the compressed Jacobian J^{cmp}.

The second method, `GetDifference`, detects differences between the pattern `J` and its overestimator `A` after the update of $\mathbf{x}$. These differences are represented by the pattern `D` having nonzero entries at all locations where `A` is zero but `J` is nonzero. To capture these differences induced by variations in the input variables t, $\mathbf{x}^{[t]}$, and $\mathbf{x}$, it is not unreasonable to assume that this method relies on user information or an efficient algorithm for sparsity detection returning `D`.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

Algorithm 1 Two-phase flow simulation via dynamic matrix compression

procedure TWO-PHASEFLOW(State $\mathbf{x}^{\text{init}}$ at initial time, Permeability $\mathbb{K}$, ...)
  Set $\mathbf{x} \leftarrow \mathbf{x}^{\text{init}}$
  $\texttt{InitModel}(\mathbf{x}, \mathbb{K}, \dots)$
  $\texttt{C}, \texttt{A}, \gamma, \texttt{S} \leftarrow \texttt{InitMatrixCompression}(\mathbf{x})$
  for $t = 0, 1, 2, \dots$ **do** ▷ Time stepping
    Set $\mathbf{x}^{[t]} \leftarrow \mathbf{x}$
    while Nonlinear solution $\mathbf{x}$ not converged **do** ▷ Newton iteration
      Set $\texttt{A}^{\text{old}} \leftarrow \texttt{A}$ and recompute fluid properties that depend on $\mathbf{x}$
      Set $\mathbf{b} \leftarrow -\mathbf{F}(t, \mathbf{x}^{[t]}, \mathbf{x})$ and $J^{\text{cmp}} \leftarrow J(t, \mathbf{x}^{[t]}, \mathbf{x}) \cdot \texttt{S}$ ▷ Matrix-free AD
      $J \leftarrow \texttt{recover}(J^{\text{cmp}}, \texttt{S})$
      $\Delta\mathbf{x} \leftarrow J \backslash \mathbf{b}$
      Update $\mathbf{x} \leftarrow \mathbf{x} + \Delta\mathbf{x}$
      $\texttt{D} \leftarrow \texttt{GetDifference}(t, \mathbf{x}^{[t]}, \mathbf{x}, \texttt{A}^{\text{old}})$
      $\texttt{C}, \texttt{A}, \gamma, \texttt{S} \leftarrow \texttt{UpdateMatrixCompression}(\texttt{C}, \texttt{A}, \texttt{D}, \gamma, \texttt{S})$
    end while
  end for
end procedure

The last method, `UpdateMatrixCompression`, uses previous information to update the overestimator $\texttt{A}$. For the updated overestimator, it also computes an updated grouping γ besides updated versions for the seed pattern $\texttt{S}$ and the corresponding overestimator $\texttt{C}$ of the compressed Jacobian. This method for the updates of the overestimators $\texttt{A}$ and $\texttt{C}$ is summarized in Algorithm 2.

The logic behind this algorithm is as follows. Suppose there exists a column $\texttt{D}_i$ in $\texttt{D}$ in which new nonzero entries have been added to the sparsity pattern $\texttt{J}$ that were not previously covered by the overestimator $\texttt{A}$. In this case, it is necessary to inspect whether the column i with its new entries from $\texttt{D}_i$ can remain in its original group $\gamma(i)$ or needs to be assigned to another group. The corresponding analysis is based on checking structural orthogonality as follows.

There are three different types of updating the data structures. If column i augmented with the additional nonzero entries identified in $\texttt{D}_i$ stays structurally orthogonal to all other columns in its group $\gamma(i)$, then column i remains in its group. Namely, column i remains in its group if the condition

$$\texttt{C}_{\gamma(i)} \perp \texttt{D}_i$$

is satisfied. Notice that the pattern of $\texttt{A}_i$ is already included in $\texttt{C}_{\gamma(i)}$. We refer to this update type as "Keep Group."

Otherwise, the column i needs to be assigned to another group. Assigning it to an existing group $k \neq \gamma(i)$ can only be done if the updated column i is structurally orthogonal to all columns associated to that group k, i.e.,

$$\texttt{A}_i \vee \texttt{D}_i \perp \texttt{C}_k.$$

This update type is called "Change Group."

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

Algorithm 2 Update of data structures in dynamic matrix compression

procedure UpdateMatrixCompression($\mathtt{C}$, $\mathtt{A}$, $\mathtt{D}$, γ, $\mathtt{S}$)
  Let c denote the (current) number of columns in $\mathtt{C}$.
  Let $\mathcal{C} = \{1, 2, \ldots, c\}$ be the indices of all columns in $\mathtt{C}$.
  Let $\mathcal{D}$ be the set of indices of all columns in $\mathtt{D}$ having nonzero entries.
  while $(\mathcal{D} \neq \emptyset)$ **do**
    Pick index $i \in \mathcal{D}$ of a column that is (currently) assigned to group $\gamma(i)$.
    Check structural orthogonality to see if groups need to be changed:
    if $\mathtt{C}_{\gamma(i)} \perp \mathtt{D}_i$ **then** ▷ Keep group
      Set $g \leftarrow \gamma(i)$, i.e., column i keeps its group $\gamma(i)$.
      Set $\mathtt{v} \leftarrow \mathtt{D}_i$.
    else if $\exists\, k \in \mathcal{C} \setminus \{\gamma(i)\}$ s.t. $\mathtt{C}_k \perp \mathtt{A}_i \vee \mathtt{D}_i$ **then** ▷ Change group
      Set $g \leftarrow k$, i.e., column i changes from group $\gamma(i)$ to group k.
      Set $\mathtt{v} \leftarrow \mathtt{A}_i \vee \mathtt{D}_i$.
    else ▷ New group
      Set $c \leftarrow c + 1$ to increase the number of groups.
      Set $\mathcal{C} \leftarrow \mathcal{C} \cup \{c\}$.
      Set $g \leftarrow c$, i.e., column i changes from group $\gamma(i)$ to a new group.
      Set $\mathtt{v} \leftarrow \mathtt{A}_i \vee \mathtt{D}_i$.
      Append zero column $\mathtt{C} \leftarrow [\mathtt{C}, 0]$ to compressed pattern.
      Append zero column $\mathtt{S} \leftarrow [\mathtt{S}, 0]$ to seed pattern.
    end if
    Update grouping $\gamma(i) \leftarrow g$.
    Replace row i of the seed pattern $\mathtt{S}$ by the g-th Euclidean unit vector $\mathtt{e}_g$.
    Update column i of overestimated pattern $\mathtt{A}_i \leftarrow \mathtt{A}_i \vee \mathtt{D}_i$.
    Update column g of compressed overestimated pattern $\mathtt{C}_g \leftarrow \mathtt{C}_g \vee \mathtt{v}$.
    Update index set $\mathcal{D} \leftarrow \mathcal{D} \setminus \{i\}$, i.e., remove column index i from $\mathcal{D}$.
  end while
  return $\mathtt{C}$, $\mathtt{A}$, γ, $\mathtt{S}$
end procedure

If a column can neither keep its original group nor be assigned to another already existing group, it has to be assigned to a new group. This situation is related to the update type "New Group."

6 Numerical Experiments

In this section, the proposed dynamic approach is evaluated by means of numerical experiments. It is implemented using the MATLAB Reservoir Simulation Toolbox (MRST) [11]. As a starting point, the open-source code from the project *fas_mrst* [14] is used, which was subsequently modified to incorporate the proposed algorithm and the specific requirements of this study. The adapted code can be made available upon request. Algorithmic differentiation is performed using the software package ADiMat [3, 15].

The partial differential equations (3) and (4) are solved on a three-dimensional grid consisting of 1554 nodes and $N = 1080$ cells. In each of 100 time steps, the

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

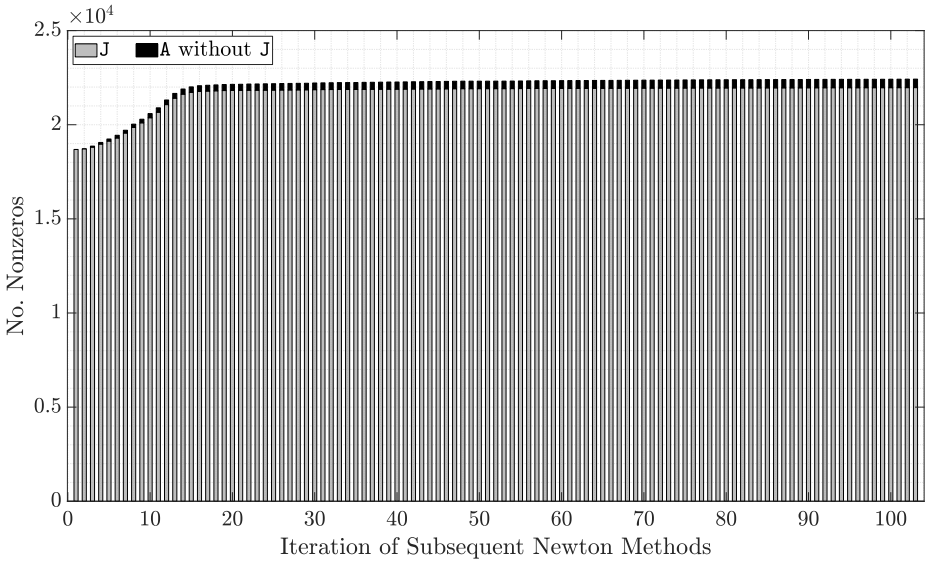

Fig. 1. Number of nonzeros, `J`, used in the static approach and the additional number of nonzeros, `A` without `J`, also stored in `A` in the dynamic approach.

resulting nonlinear system (5) is solved using a Newton method leading to a linear system (6) of order $2N = 2160$, which is solved directly using the backslash operator. For all time steps, Newton's method converges robustly. The norm of the residual, $||\mathbf{F}||_2$, decreases monotonically from approximately 10^4 in the initial iterations to below 10^{-8} at convergence, confirming that the proposed dynamic approach preserves the convergence behavior of a static approach. Therefore, the following discussion concentrates on the novel aspects of the dynamic approach.

In Fig. 1, the number of nonzeros in the Jacobian sparsity pattern `J` is shown in gray as a function of the iteration index. Here, the iteration index is defined as the concatenation of all Newton iterations across successive time steps, resulting in a single, linearly ordered sequence of nonlinear iterations over the entire simulation. As expected, the number of nonzeros of `J` increases rapidly during the initial iterations, reflecting the activation of couplings in the system. After this initial phase, the growth rate decreases significantly and the sparsity pattern stabilizes. The largest increase occurs between iteration 1 and 15. Beyond this point, only minor changes in the sparsity pattern are observed. This stagnation increases our confidence that the Jacobian structure changes only marginally once the simulation has progressed beyond the initial transient phase, making repeated recomputations of groupings from scratch unnecessarily complex.

Figure 1 also depicts information on the number of nonzeros in the overestimated data structure `A` as black segments stacked on top of the gray bars related to `J`. By construction, the number of nonzeros in `A` are monotonically increasing. Consequently, `A` can contain additional entries that are not present in the exact

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

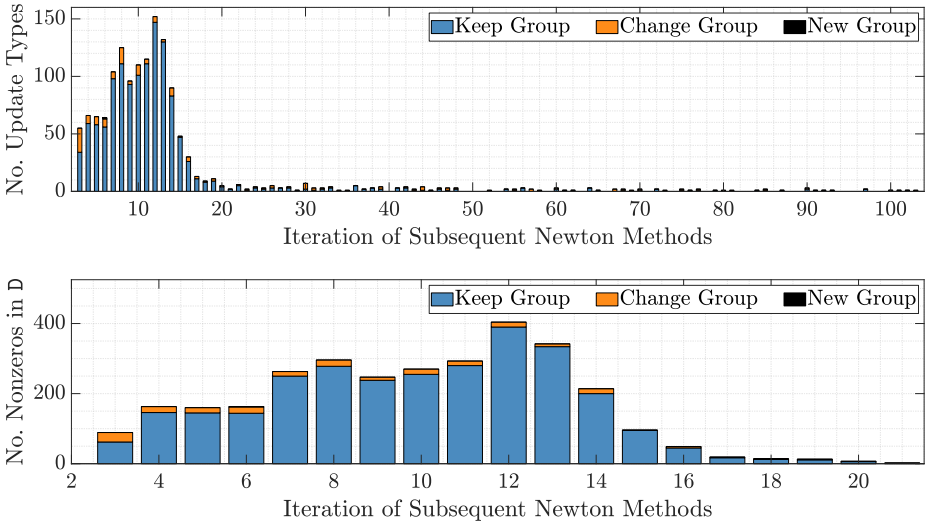

Fig. 2. Number of occurrences of different update types (top) and number of nonzeros added by them during the iterations 3 to 21 (bottom).

Jacobian sparsity pattern. The resulting gap between the overestimated and the actual number of nonzeros and is drawn in black in that figure and represents the additional memory overhead introduced by the proposed dynamic approach. However, this overhead remains small, below 2%, throughout the simulation. This moderate overestimation of nonzero elements indicates that the dynamic approach provides a viable strategy for large-scale simulations.

To further analyze the dynamic approach, the top plot of Fig. 2 examines the number of occurrences of each of the three different update types introduced in Algorithm 2. The horizontal axis starts at iteration 3 because we switch from the static to the dynamic approach only after the second iteration. The stacked bar graph indicates that updates primarily occur in the beginning of the Newton iterations; there are hardly any occurrences of updates after iteration, say 20. Furthermore, the overwhelming majority of updates are of type "Keep Group." Updates of type "New Group" are extremely rare. For instance, there is a small black segment on top of the other two segments in iteration 6, indicating that a minute number of such "New Group" updates occur in that experiment.

Consider the bottom plot to analyze whether a small number of updates possibly introduces a large number of nonzero entries. That bottom plot zooms in on iterations 3 to 21 and, now, it reports the number of additionally introduced nonzero entries for each update type. The analysis is performed on the column-wise sparsity patterns in `D`, where we accumulate newly introduced nonzeros in the affected columns resulting from each update. The results show that the majority of additional nonzero entries are introduced by "Keep Group" updates. In contrast, "Change Group" updates account for comparatively few additional nonzeros. The creation of "New Group" updates contributes only a

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

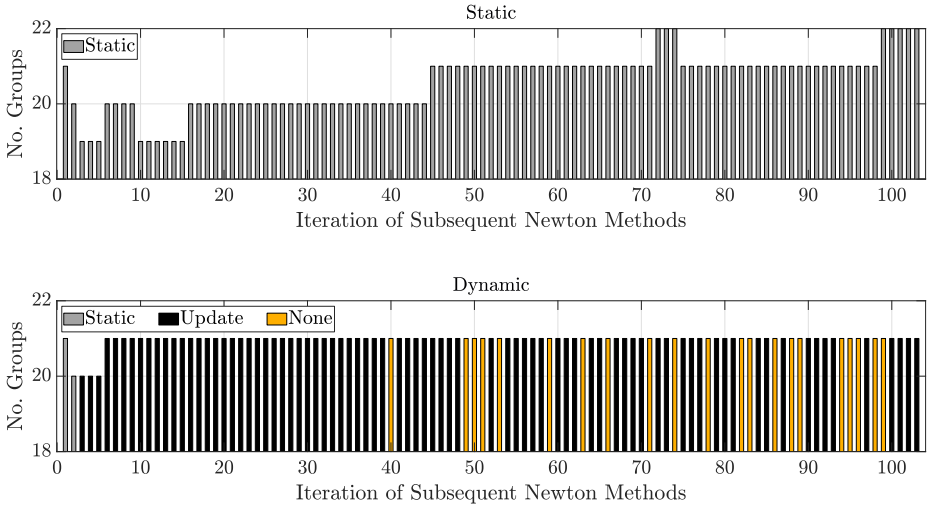

Fig. 3. Number of groups needed by the static (top) and dynamic (bottom) approaches.

tiny fraction to the overall increase. Hence, additional nonzero entries are incorporated predominantly within existing groups and most structural changes in the compressed system can be accommodated without modifying the existing grouping.

This behavior is reflected in the evolution of the number of groups over the iterations. In Fig. 3, the number of groups required for both the static and the dynamic approach are displayed as a function of the iteration index. In the static approach depicted in the top figure, the number of groups starts at 21, then varies between 19 and 20 until iteration 45, and then returns back to 21. The maximum number of groups, 22, occurs between iterations 72–74 and 99–103.

By contrast, the dynamic grouping given in the bottom plot is turned on in iteration 3 where it starts with 20 groups, then increases at iteration 6 remaining consistently at 21 for the remainder of the simulation. These results indicate that in both cases the number of columns of the seed pattern remains moderate, avoiding an uncontrolled growth in computations and storage requirements. Orange bars in the dynamic approach indicate iterations in which no grouping is performed because no nonzero entries are to be added.

All measurements of running times are taken at one of the partitions of the Ara cluster of Friedrich Schiller University Jena. This centrally-installed system offers dual-socket compute nodes with two Intel Xeon E5-2650v4 12 core chips, a clock rate of 2.2 GHz, and RAM memory of 1 TB. Each experiment is carried out serially on a dedicated compute node, eliminating potential effects caused by any other users or the underlying network architecture. All computations are repeated 99 times and running times reported in the following refer to the median of each set of these 99 running times.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

We are interested in assessing the performance of the novel dynamic approach in more detail. To this end, for each Newton iteration, the running times of the following parts of Algorithm 1 are measured separately.

- "Grouping": This is the core part of the dynamic approach and refers to line 14 of the algorithm. Essentially, it provides an update of the three patterns `C`, `A`, and `S` by executing the procedure `UpdateMatrixCompression`. Given the information on how the sparsity pattern `A` changes to $\mathtt{A} \vee \mathtt{D}$, it updates a given column grouping encoded by the patterns `C` and `S`.
- "Jacobian": Given a binary seed pattern `S`, this part first computes the compressed Jacobian J^{cmp} by a Jacobian-matrix product using AD in a matrix-free fashion and then recovers the Jacobian J from its compressed version J^{cmp}. It corresponds to lines 9 and 10 of the algorithm.
- "Remaining": This part accumulates all remaining times of a Newton iteration, ignoring sparsity detection. Recall the assumption that the sparsity pattern is detected by the black box procedure `GetDifference` for which we do not measure any timing. However, this part includes the solution of the linear system (6) and the recomputation of fluid properties, which are needed in the part "Jacobian" to evaluate $\mathbf{F}$ and the Jacobian-matrix product.

We set these running times into perspective with a state-of-the-art approach using static matrix compression previously sketched in Sect. 4. While the two parts "Remaining" and "Jacobian" are identical in the static and dynamic approach, the part "Grouping" differs. In the static approach, this part first solves a matrix compression problem with a different sparsity pattern in each iteration using some sparsity-exploiting AD technique [6, 4] and then encodes the resulting solution in the form of a seed pattern.

The running times of the static and dynamic approaches are visualized in Fig. 4. In the static approach, there is a significant fraction of the running time of each iteration that is spent in the "Grouping" part. This fraction remains roughly constant over the iterations and is caused by the solution of a new matrix compression problem in each iteration. However, the dynamic approach almost eliminates the running time related to "Grouping" after switching from a static to a dynamic setting after the first two iterations. Although the bottom figure shows a small running time associated with "Grouping" in a range of about 15 iterations immediately after turning on the dynamic approach, there is hardly any such time for all remaining iterations. This observation is in line with the occurrence of the updates shown in the top of Fig. 2.

The performance improvement of the dynamic approach as compared to the static approach is quantified in Fig. 5. Here, the corresponding ratios of the running times of the three parts are plotted versus the iterations. As motivated by the discussion of the previous figure, the largest improvements are observed for "Grouping." Here, the running time of the dynamic approach is smaller by factors up to 400. As expected from their definitions, the factors for the parts "Jacobian" and "Remaining" are consistently close to the value 1. This figure contains the additional line plot labeled "Newton" displaying the ratio of the running time of a complete Newton iteration defined as the sum of the running

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

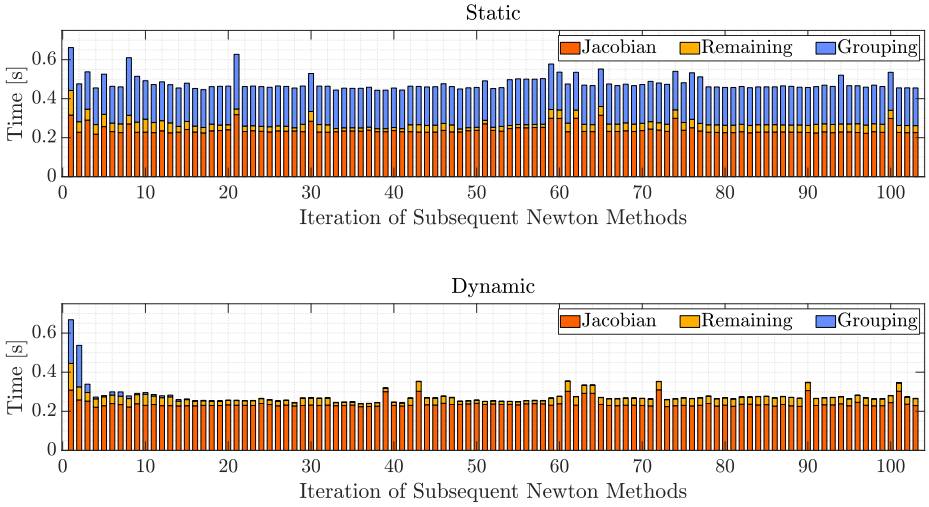

Fig. 4. Running time of different parts needed by the static (top) and dynamic (bottom) approaches.

times of the three parts "Grouping," "Jacobian," and "Remaining." Overall, the dynamic approach is faster than the static approach for all but the first two or three iterations by a factor of roughly 1.7.

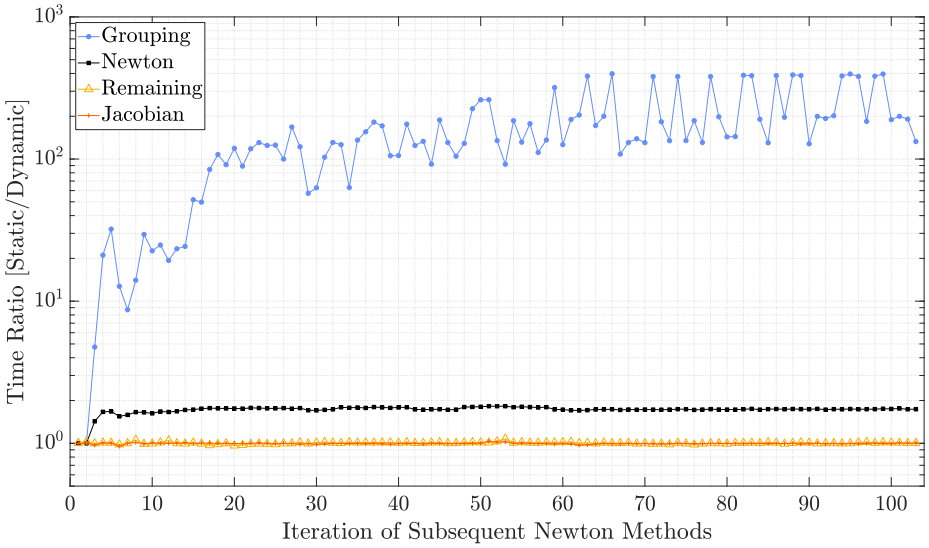

Fig. 5. Time ratio of the static and dynamic approach for different parts.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

7 Concluding Remarks

In this study, a novel strategy for a two-phase flow simulation in porous media is introduced. To solve the resulting nonlinear systems, it combines the forward mode of automatic differentiation with a sequence of compressed representations of Jacobian matrices. This way, the new dynamic approach enables the efficient assembly of a sequence of sparse linear systems required by fully coupled Newton methods. Two-phase flow simulations are particularly well-suited for this strategy, as their governing equations lead to large sparse Jacobian matrices. This sparsity originates from the local nature of the finite-volume discretization and the coupling between pressure and saturation variables. Exploiting this structure is crucial for reducing the cost of derivative evaluations, which frequently constitutes the dominant fraction of the overall computational expense.

The key contribution of this work consists of a dynamic approach to matrix compression. For a sequence of given sparsity patterns, dynamic matrix compression partitions the columns of each Jacobian into different groups of pairwise structurally orthogonal columns by an incremental update of the data structures used to represent the groupings. Rather than recomputing a grouping statically from scratch whenever a pattern changes, the proposed algorithm performs an update of an overestimator for the true Jacobian pattern, resolving potential conflicts locally in each Newton step. Columns are retained in their existing groups whenever possible, reassigned to alternative groups if necessary, and new groups are introduced only as a last resort.

Numerical experiments show that most updates can be handled without modifying these groupings, resulting in a bounded number of groups and a compact seed pattern throughout the simulation. Although the compressed representation introduces a mild overestimation of the true sparsity pattern, this overhead remains below 2% and does neither compromise the efficiency nor decrease robustness of the Newton iterations. Overall, the proposed dynamic approach provides a scalable and robust framework for Jacobian construction in Newton-based two-phase flow simulations. Compared to a corresponding static approach, it speeds up matrix compression by factors of up to 400. Future work is needed to extend the method to more complex multiphase and compositional models, as well as to investigate its performance in parallel implementations and practical settings on more massive scales.

Acknowledgments. This research was funded in part by the Carl Zeiss Foundation within the project P2021-02-005 "Interactive Inference," by the German Federal Ministry of Education and Research (BMBF) within the project THInKI, project number 16DHBKI084, and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 273418955 and 359757177.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37

References

1. Abidoye, L.K., Khudaida, K.J., Das, D.B.: Geological carbon sequestration in the context of two-phase flow in porous media: A review. Critical Reviews in Environmental Science and Technology **45**(11), 1105–1147 (2015). https://doi.org/10.1080/10643389.2014.924184
2. Bear, J.: Dynamics of fluids in porous media. Dover Publications Inc. (1988)
3. Bischof, C.H., Bücker, H.M., Lang, B., Rasch, A., Vehreschild, A.: Combining source transformation and operator overloading techniques to compute derivatives for MATLAB programs. In: Proceedings Second IEEE International Workshop on Source Code Analysis and Manipulation. pp. 65–72 (2002). https://doi.org/10.1109/SCAM.2002.1134106
4. Coleman, T.F., Moré, J.J.: Estimation of sparse Jacobian matrices and graph coloring problems. SIAM Journal on Numerical Analysis **20**(1), 187–209 (1983). https://doi.org/10.1137/0720013
5. Forsyth, P.A.: Comparison of the single-phase and two-phase numerical model formulation for saturated-unsaturated groundwater flow. Computer Methods in Applied Mechanics and Engineering **69**(2), 243–259 (1988). https://doi.org/10.1016/0045-7825(88)90190-9
6. Gebremedhin, A.H., Manne, F., Pothen, A.: What color is your Jacobian? Graph coloring for computing derivatives. SIAM Review **47**(4), 629–705 (2005). https://doi.org/10.1137/S0036144504444711
7. Griewank, A., Walther, A.: Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. No. 105 in Other Titles in Applied Mathematics, SIAM, Philadelphia, PA, USA, 2nd edn. (2008). https://doi.org/10.1137/1.9780898717761
8. Jansen, J.D.: A systems description of flow through porous media, vol. 570. Springer (2013). https://doi.org/10.1007/978-3-319-00260-6
9. Kelley, C.T.: Solving Nonlinear Equations with Newton's Method. SIAM, Philadelphia, PA, USA (2003). https://doi.org/10.1137/1.9780898718898
10. Lai, C., Bodvarsson, G., Truesdell, A.: Modeling studies of heat transfer and phase distribution in two-phase geothermal reservoirs. Geothermics **23**(1), 3–20 (1994). https://doi.org/10.1016/0375-6505(94)90042-6
11. Lie, K.A.: An introduction to reservoir simulation using MATLAB/GNU Octave: User guide for the MATLAB Reservoir Simulation Toolbox (MRST). Cambridge University Press, Cambridge, UK (2019). https://doi.org/10.1017/9781108591416
12. Monteagudo, J.E.P., Firoozabadi, A.: Control-volume method for numerical simulation of two-phase immiscible flow in two-and three-dimensional discrete-fractured media. Water Resources Research **40**(7) (2004). https://doi.org/10.1029/2003WR002996
13. Saunders, J.H., Jackson, M.D., Pain, C.C.: A new numerical model of electrokinetic potential response during hydrocarbon recovery. Geophysical Research Letters **33**(15), L15316:1–L15316:6 (2006). https://doi.org/10.1029/2006GL026835
14. Toft, R.: twoPhaseAD.m. `https://github.com/raymondToft/fas_mrst` (2023), accessed: 2025-06-02
15. Willkomm, J., Bischof, C.H., Bücker, H.M.: A new user interface for ADiMat: Toward accurate and efficient derivatives of Matlab programs with ease of use. International Journal of Computational Science and Engineering **9**(5/6), 408–415 (2014). https://doi.org/10.1504/IJCSE.2014.064526

ICCS Camera Ready Version 2026
To cite this paper please use the final published version:
DOI: 10.1007/978-3-032-29921-5_37