Fast MATLAB evaluation of nonlinear energies using FEM in 2D and 3D: Nodal elements

Alexej Moskovka [a], Jan Valdman [b,c,*]

[a] *Department of Mathematics, Faculty of Applied Sciences, University of West Bohemia, Technická 8, 30100 Plzeň, Czechia*
[b] *Institute of Information Theory and Automation, Czech Academy of Sciences, Pod vodárenskou věží 4, 18200 Praha 8, Czechia*
[c] *Department of Computer Science, Faculty of Science, University of South Bohemia, Branišovská 1760, 37005 České Budějovice, Czechia*

ARTICLE INFO

ABSTRACT

Nonlinear energy functionals appearing in the calculus of variations can be discretized by the finite element (FE) method and formulated as a sum of energy contributions from local elements. A fast evaluation of energy functionals containing the first order gradient terms is a central part of this contribution. We describe a vectorized implementation using the simplest linear nodal (P1) elements in which all energy contributions are evaluated all at once without the loop over triangular or tetrahedral elements. Furthermore, in connection to the first-order optimization methods, the discrete gradient of energy functional is assembled in a way that the gradient components are evaluated over all degrees of freedom all at once. The key ingredient is the vectorization of exact or approximate energy gradients over nodal patches. It leads to a time-efficient implementation at higher memory-cost. Provided codes in MATLAB related to 2D/3D hyperelasticity and 2D p-Laplacian problem are available for download and structured in a way it can be easily extended to other types of vector or scalar forms of energies.

© 2022 Elsevier Inc. All rights reserved.

1. Introduction

Given a domain $\Omega \in \mathbb{R}^{\dim}$, where $\dim \in \{1, 2, 3\}$ is the space-dimension, we consider a minimization problem

$$J(\mathbf{u}) = \min_{\mathbf{v} \in V} J(\mathbf{v}), \tag{1}$$

where V is a space of trial functions and $J : V \rightarrow \mathbb{R}$ represents an energy functional in the form

$$J(\mathbf{v}) = J_{grad}(\mathbf{v}) + J_{lin}(\mathbf{v}), \tag{2}$$

where $J_{grad}(\mathbf{v})$ denotes its first-gradient part and $J_{lin}(\mathbf{v})$ its linear part. Examples of such minimization problems are numerous and their study is a general subject of the Calculus of variations. There are models with higher order derivatives (such as plate problems with the second derivative in their formulation) available but not considered in this contribution.

As the main example we recall a class of vector nonlinear elasticity problems represented by minimizations of energies of hyperelastic materials [10,12]. The trial space is chosen as

$$V = W_D^{1,p}(\Omega, \mathbb{R}^{\dim})$$

---

* Corresponding author.
  E-mail address: jvaldman@jcu.cz (J. Valdman).

i.e., the (vector) Sobolev space of $L^p(\Omega)$ integrable functions with the first weak derivative being also $L^p(\Omega)$ integrable and satisfying (in the sense of traces) Dirichlet boundary conditions $\mathbf{v}(\mathbf{x}) = \mathbf{u}_D(\mathbf{x})$ at the domain boundary $\mathbf{x} \in \partial\Omega$ for a prescribed function $\mathbf{u}_D : \partial\Omega \to \mathbb{R}^{\dim}$. A primal variable is the deformation mapping $\mathbf{v} \in V$ describing the relocation of any point $\mathbf{x} \in \Omega$ during the deformation process.

Then the gradient deformation tensor $\mathbf{F} \in L^p(\Omega, \mathbb{R}^{\dim \times \dim})$ is defined as

$$
\mathbf{F}(\mathbf{v}) = \nabla\mathbf{v} = \begin{bmatrix} \frac{\partial v^{(1)}}{\partial x_1} & \cdots & \frac{\partial v^{(1)}}{\partial x_{\dim}} \\ \vdots & & \vdots \\ \frac{\partial v^{(\dim)}}{\partial x_1} & \cdots & \frac{\partial v^{(\dim)}}{\partial x_{\dim}} \end{bmatrix}.
\tag{3}
$$

The first-gradient and the linear parts of the energy functional (2) read

$$
J_{grad}(\mathbf{v}) = \int_\Omega W\big(\mathbf{F}(\mathbf{v}(\mathbf{x}))\big)\,\mathrm{d}\mathbf{x}, \qquad J_{lin}(\mathbf{v}) = -\int_\Omega \mathbf{f}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x})\,\mathrm{d}\mathbf{x},
$$

where $W : \mathbb{R}^{\dim \times \dim} \to \mathbb{R}$ defines a strain-energy density function and $\mathbf{f} : \Omega \to \mathbb{R}^{\dim}$ a loading functional. We assume the compressible Neo-Hookean density

$$
W(\mathbf{F}) = C_1\big(I_1(\mathbf{F}) - \dim - 2\log(\det\mathbf{F})\big) + D_1(\det\mathbf{F} - 1)^2,
\tag{4}
$$

where $I_1(\mathbf{F}) = |\mathbf{F}|^2$ uses the Frobenius norm $|\cdot|$, and $\det(\cdot)$ is the matrix determinant operator. An extension to other gradient densities as the St. Venant Kirchhoff is possible.

As the second example we recall a scalar p-Laplacian problem [7] with the energy functional defined as

$$
J(v) = \frac{1}{p} \int_\Omega |\nabla v(\mathbf{x})|^p\,\mathrm{d}\mathbf{x} - \int_\Omega f(\mathbf{x})\,v(\mathbf{x})\,\mathrm{d}\mathbf{x},
\tag{5}
$$

where $V = W_D^{1,p}(\Omega, \mathbb{R})$. The functional $J(v)$ is then known to be strictly convex in V for $p \in (1, \infty)$ and it has therefore a unique minimizer $u(\mathbf{x}) \in V$.

The main motivation of this contribution is to describe how nonlinear energy functionals can be efficiently and automatically evaluated by the finite element method (FE). We provide vectorization concepts and MATLAB implementation for

- evaluations of the value $J(\mathbf{v})$,
- evaluations of the gradient vector $\nabla J(\mathbf{v})$

expressed for a trial function $\mathbf{v} \in V$. These two objects can be passed to an external optimization method of the first order to find a minima of the energy $\mathbf{u} \in V$ and the corresponding minimal energy value $J(\mathbf{u})$. We utilize the trust-region optimization method [4] available in the MATLAB Optimization Toolbox [14] for benchmarking. Our implementation is built on the top of our own vectorized codes [3,6,15] developed primarily for assemblies of finite element matrices. There are no explicit loops over mesh elements in evaluations of $J(\mathbf{v})$ and $\nabla J(\mathbf{v})$, all necessary data such as gradients of basis functions and energy densities are computed all at once. It leads to a significant computational speedup, but also it is memory intensive. Practically, the user specifies the form of the energy $J(\mathbf{v})$ and the corresponding gradient vector $\nabla J(\mathbf{v})$ is evaluated approximately by a central difference scheme. Alternatively, if the user is willing to apply some differential calculus to a particular form of the energy, then the exact gradient can be assembled. This exact gradient approach is not versatile, but leads to the further performance speedup in our tests. We set up a set of six benchmarks as a base for future tests and improvements:

- The mesh and the nodal patches data are preprocessed in Benchmark 1.
- Evaluations of $J(\mathbf{v})$ and $\nabla J(\mathbf{v})$ for a given $\mathbf{v} \in V$ are provided in Benchmarks 2 and 3.
- The full minimizations of 2D/3D hyperelasticity and 2D p-Laplacian energies are shown in Benchmarks 4, 5 and 6.

Authors are not aware of any similar MATLAB implementation. This is also our first attempt in this direction apart from our own contribution [13] focusing on p-Laplacian energy minimization. There is a growing number of MATLAB vectorized implementations of the second order linear partial differential equations eg. [9,17] or particular nonlinear partial differential equations [6,16].

The paper is structured in the following way: Section 2 summarizes useful notation and Section 3 basics of FEM. Section 4 includes implementation of two structures: mesh and (nodal) patches. Section 5 is focused on implementation of energy evaluation and Section 6 on implementation of the gradient of energy. The final Section 7 reports on the solutions of particular minimization problems.

2. Notation

Index mappings in the construction of FEM:

$I_{LG} : \mathbb{N} \to \mathbb{N}$ - (local to global) mapping which for a local basis function on an element returns the index of the corresponding global basis function

$I_{DN} : \mathbb{N} \to \mathbb{N}$ - (degree to node) mapping which for the n-th degree of freedom returns the index i of the corresponding node, see (22)

Domain triangulations are described by the following parameters:

$\mathcal{T}, \mathcal{N}$ - a set of elements, a set of nodes
$\mathcal{M} \subset \mathcal{N}$ - a set of (at least partially) free nodes
$\mathcal{T}^i, \mathcal{T}^{(n)}$ - the i-th nodal patch, the $\left(I_{DN}(n)\right)$-th nodal patch
$|\mathcal{T}|, |\mathcal{N}|, |\mathcal{M}|, |\mathcal{T}^i|, |\mathcal{T}^{(n)}|$ - the sizes of sets $\mathcal{T}, \mathcal{N}, \mathcal{M}, \mathcal{T}^i, \mathcal{T}^{(n)}$

Both scalar and vector problems are treated together as a vector problem with d components, where $d = 1$ for scalar problems, $d = \dim$ for vector problems and dim is the space dimension. The following indices are frequently used:

i - the index of a node ($i \in \{1, \ldots, n_b\}$, also $i \in \{1, \ldots, |\mathcal{N}|\}$ or $i \in \{1, \ldots, |\mathcal{M}|\}$)
j - the index of a vector component ($j \in \{1, \ldots, d\}$)
k - the index of an element ($k \in \{1, \ldots, |\mathcal{T}|\}$), also ($k \in \{1, \ldots, |\mathcal{T}^i|\}$) or ($k \in \{1, \ldots, |\mathcal{T}^{(n)}|\}$)
ℓ - the index of a local basis function ($\ell \in \{1, \ldots, \dim + 1\}$)
m - the index of a spatial component ($m \in \{1, \ldots, \dim\}$)
n - the index of a global degree of freedom ($n \in \{1, \ldots, d|\mathcal{N}|\}$) or an active degree of freedom ($n \in \{1, \ldots, d|\mathcal{M}|\}$)
r - the index of a global patches matrix row ($r \in \{1, \ldots, |\mathcal{M}|\}$)

Nodal basis functions are used in several ways:

$\varphi_i(\mathbf{x}), \boldsymbol{\varphi}_i(\mathbf{x})$ - a scalar global nodal basis function, a vector global nodal basis function
$\boldsymbol{\varphi}_{k,\ell}(\mathbf{x})$ - the ℓ-th local basis function on the k-th element

A trial vector function is addressed in several ways:

$\mathbf{v}(\mathbf{x}), v^{(j)}(\mathbf{x})$ - a trial vector function and its j-th component
$\mathbf{V}$ - a matrix of the coefficients of $\mathbf{v}(\mathbf{x})$ in the nodal finite element basis
$\mathbf{v}_i, \mathbf{v}^{(j)}, v_i^{(j)}$ - the i-th row of $\mathbf{V}$, the j-th column of $\mathbf{V}$, the (i,j) element of $\mathbf{V}$
$\mathbf{v}$ - a vector reshaped from $\mathbf{V}$
$\hat{\mathbf{v}}$ - the restriction of $\mathbf{v}$ to free nodes

Given matrices $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{p \times q}$, the following operators are used:

$\mathrm{tr}(\mathbf{A})$ - the trace of matrix (for $p = q$)
$\det(\mathbf{A})$ - the determinant of matrix (for $p = q$)
$\mathbf{A} \odot \mathbf{B}$ - the elementwise (Hadamard) product defined as a matrix $\mathbf{C} \in \mathbb{R}^{p \times q}$, where $c_{i,j} = a_{i,j} b_{i,j}$
$\mathbf{A} : \mathbf{B}$ - the scalar product defined as $\mathbf{A} : \mathbf{B} = \sum_{i=1}^{p} \sum_{j=1}^{q} a_{i,j} b_{i,j} = \sum_{i=1}^{p} \sum_{j=1}^{q} \mathbf{A} \odot \mathbf{B}$

3. Finite element discretization

The finite element method [5] is applied for the discretization of (1). We assume a trial function and a trial space of the form

$$\mathbf{v}(\mathbf{x}) = (v^{(1)}(\mathbf{x}), \ldots, v^{(d)}(\mathbf{x})), \qquad V = V^{(1)} \times \ldots \times V^{(d)}$$

and approximate $\mathbf{v}(\mathbf{x}) \in V$ in the finite-dimensional subspace

$$V_h = V_h^{(1)} \times \ldots \times V_h^{(d)} \subset V,$$

where $V_h^{(1)} = \ldots = V_h^{(d)} := V_h^s$ and the scalar basis space V_h^s is generated from the scalar basis functions

$$\varphi_i(\mathbf{x}) \in V_h^s, \qquad i \in \{1, \ldots, n_b\},$$

where n_b denotes their number. Hence, any component of $\mathbf{v}(\mathbf{x})$ is given by a linear combination

$$v^{(j)}(\mathbf{x}) = \sum_{i=1}^{n_b} v_i^{(j)} \varphi_i(\mathbf{x}), \qquad \mathbf{x} \in \Omega, \quad j \in \{1, \ldots, d\}. \tag{6}$$

The coefficients $v_i^{(j)}$ from (6) are assembled in a matrix $\mathbf{V} \in \mathbb{R}^{n_b \times d}$ given as

$$\mathbf{V} = \begin{pmatrix} v_1^{(1)} & \cdots & v_1^{(d)} \\ \vdots & & \vdots \\ \vdots & & \vdots \\ v_{n_b}^{(1)} & \cdots & v_{n_b}^{(d)} \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1 \\ \vdots \\ \vdots \\ \mathbf{v}_{n_b} \end{pmatrix} = \begin{pmatrix} \mathbf{v}^{(1)} \cdots \mathbf{v}^{(d)} \end{pmatrix} \tag{7}$$
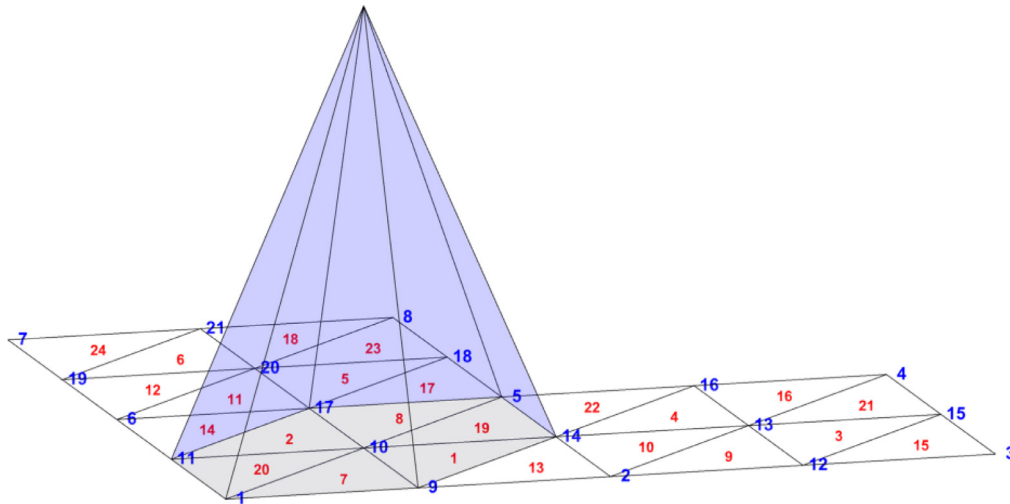
Fig. 1. Example of a scalar nodal basis function for dim = 2.

and latter two equivalent forms assume a row vector $\mathbf{v}_i = (v_i^{(1)}, \ldots, v_i^{(d)})$, $i \in \{1, \ldots, n_b\}$, and a column vector $\mathbf{v}^{(j)} = (v_1^{(j)}, \ldots, v_{n_b}^{(j)})^T$, $j \in \{1, \ldots, d\}$. Using a row vector basis function

$$\boldsymbol{\varphi}_i(\mathbf{x}) = \underbrace{(\varphi_i(\mathbf{x}), \ldots, \varphi_i(\mathbf{x}))}_{d-\text{times}}, \qquad i \in \{1, \ldots, n_b\}$$

one can rewrite (6) in a compact way for all components $j \in \{1, \ldots, d\}$ as

$$\mathbf{v}(\mathbf{x}) = \sum_{i=1}^{n_b} \mathbf{v}_i \odot \boldsymbol{\varphi}_i(\mathbf{x}), \qquad x \in \Omega, \tag{8}$$

where the symbol $\odot$ represents an elementwise (or Hadamard) multiplication (here multiplication of components with the same index j). Formula (8) is the key tool for a vectorized implementation, since MATLAB provides the elementwise multiplication.

The domain Ω is then approximated by its triangulation $\mathcal{T}$ into closed elements in the sense of Ciarlet [5]. The simplest possible elements are considered, i.e., triangles for dim = 2 and tetrahedra for dim = 3. The elements are geometrically specified by their nodes (or vertices) belonging to the set of nodes $\mathcal{N}$. The nodes are also clustered into elements edges (for dim $\geq$ 2) and faces (for dim = 3). The numbers of elements and nodes are denoted as $|\mathcal{T}|$ and $|\mathcal{N}|$.

Given a node $N_i \in \mathcal{N}, i \in \{1, \ldots, |\mathcal{N}|\}$, we define a nodal patch $\mathcal{T}^i$ by

$$\mathcal{T}^i = \{T \in \mathcal{T} : N_i \in T\}$$

and the number of its elements by $|\mathcal{T}^i|$. The nodal patch $\mathcal{T}^i$ consists of elements denoted as T_k^i, $k \in \{1, \ldots, |\mathcal{T}^i|\}$, which are adjacent to the node N_i. The same nodal patch $\mathcal{T}^i$ can be alternatively denoted by $\mathcal{T}^{(n)}$, where $i = I_{DN}(n)$ and n is the index of one of the corresponding degrees of freedom.

We consider only the case where $V_h^s = P^1(\mathcal{T})$ is the space of nodal, elementwise linear and globally continuous scalar basis functions. Then the number of basis functions is equal to the number of nodes

$$n_b = |\mathcal{N}|,$$

but some coefficients of the trial function $\mathbf{v}(\mathbf{x})$ are known due to the Dirichlet boundary conditions.

Example 1. One regular triangulation $\mathcal{T}$ of an L-shape domain Ω is shown in Fig. 1. The triangulation is specified by $|\mathcal{T}| = 24$ and $|\mathcal{N}| = 21$. The graph of the global scalar basis function $\varphi_{10}(\mathbf{x})$ is displayed. The function has a hexagonal pyramid shape and the support on the nodal patch

$$\mathcal{T}^{10} = \{T_1, T_2, T_7, T_8, T_{19}, T_{20}\}.$$

The restrictions of $\varphi_{10}(\mathbf{x})$ to its six supporting triangles in $\mathcal{T}^{10}$ are given as linear functions with values 1 at the node N_{10} and values 0 at the two remaining nodes.

Additionally, for the scalar case ($d = 1$) the node N_{10} has only one corresponding degree of freedom with index 10 and, therefore, $\mathcal{T}^{(10)} = \mathcal{T}^{10}$ as long as $I_{DN}(10) = 10$. For the vector case ($d = 2$) the same node N_{10} has two corresponding degrees of freedom with indices 19,20, hence, $\mathcal{T}^{(19)} = \mathcal{T}^{(20)} = \mathcal{T}^{10}$ as long as $I_{DN}(19) = I_{DN}(20) = 10$. This alternative notation is essential in Section 6.

3.1. The first-gradient energy term $J_{grad}(v)$

Since the first gradient of any scalar basis function $\varphi_i(\mathbf{x}) \in P^1(\mathcal{T})$, $i \in \{1, \ldots, |\mathcal{N}|\}$, is a piecewise constant function on each element, the gradient part of the discrete energy can be written as a sum over the elements

$$J_{grad}(\mathbf{v}) = \int_\Omega W(\mathbf{F}(\mathbf{v}(\mathbf{x}))) \, d\mathbf{x} = \sum_{k=1}^{|\mathcal{T}|} \int_{T_k} W(\nabla \mathbf{v}(\mathbf{x})) \, d\mathbf{x} = \sum_{k=1}^{|\mathcal{T}|} |T_k| W\left(\nabla \mathbf{v}(\mathbf{x})|_{T_k}\right), \tag{9}$$

and $|T_k|$ denotes the size of the element T_k, $k \in \{1, \ldots, |\mathcal{T}|\}$ (equal to its length in 1D, its area in 2D or its volume in 3D). In order to evaluate (9), we need to assemble the gradient $\mathbf{v}(\mathbf{x})|_{T_k}$ on every element. To do it, we define a (local-global) mapping $I_{LG} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ which for the k-th element and its ℓ-th node returns the global index i of this node. Then a local basis function is given as

$$\boldsymbol{\varphi}_{k,\ell}(x) = \boldsymbol{\varphi}_i\big|_{T_k}(x), \qquad \text{where } i = I_{LG}(k, \ell) \tag{10}$$

for $k \in \{1, \ldots, |\mathcal{T}|\}$, $\ell \in \{1, \ldots, dim+1\}$. Hence, any partial derivative of $\mathbf{v}(\mathbf{x})$ with respect to x_m reads

$$\frac{\partial \mathbf{v}(\mathbf{x})}{\partial x_m}\bigg|_{T_k} = \sum_{\ell=1}^{dim+1} \mathbf{v}_{k,\ell} \odot \frac{\partial \boldsymbol{\varphi}_{k,\ell}(\mathbf{x})}{\partial x_m}, \qquad m \in \{1, \ldots, dim\}, \tag{11}$$

where $\mathbf{v}_{k,\ell} = \mathbf{v}_{I_{LG}(k,\ell)}$ represents the values of $\mathbf{v}$ in the ℓ-th node of the k-th element.

3.2. The linear energy term J_{lin}

Furthermore, if $\mathbf{f}(\mathbf{x}) \in V_h$, then the linear term of the energy (2) rewrites as

$$J_{lin}(\mathbf{v}) = \int_\Omega \mathbf{f}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}) \, d\mathbf{x} = \sum_{j=1}^d \int_\Omega f^{(j)}(\mathbf{x}) v^{(j)}(\mathbf{x}) \, d\mathbf{x} =$$

$$= \sum_{j=1}^d \int_\Omega \left(\sum_{i_1=1}^{|\mathcal{N}|} f_{i_1}^{(j)} \varphi_{i_1}(\mathbf{x}) \sum_{i_2=1}^{|\mathcal{N}|} v_{i_2}^{(j)} \varphi_{i_2}(\mathbf{x}) \right) d\mathbf{x} =$$

$$= \sum_{j=1}^d \sum_{i_1=1}^{|\mathcal{N}|} \sum_{i_2=1}^{|\mathcal{N}|} f_{i_1}^{(j)} v_{i_2}^{(j)} \int_\Omega \left(\varphi_{i_1}(\mathbf{x}) \varphi_{i_2}(\mathbf{x}) \right) d\mathbf{x}. \tag{12}$$

All integral terms in the formula above can be assembled in a sparse and symmetric mass matrix $\mathbf{M} \in \mathbb{R}^{|\mathcal{N}| \times |\mathcal{N}|}$ with entries

$$\mathbf{M}_{i_1, i_2} = \int_\Omega \varphi_{i_1}(\mathbf{x}) \varphi_{i_2}(\mathbf{x}) \, d\mathbf{x}, \qquad i_1, i_2 \in \{1, 2, \ldots, |\mathcal{N}|\}. \tag{13}$$

Then we can define vectors $\mathbf{b}^{(j)} = \mathbf{M} \mathbf{f}^{(j)} \in \mathbb{R}^{|\mathcal{N}|}$, where $\mathbf{f}^{(j)} = (f_1^{(j)}, \ldots, f_{|\mathcal{N}|}^{(j)})^T \in \mathbb{R}^{|\mathcal{N}|}$, $j \in \{1, \ldots, d\}$ and it is easy to check the exact formula

$$J_{lin}(\mathbf{v}) = \int_\Omega \mathbf{f}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}) \, d\mathbf{x} = \sum_{j=1}^d \mathbf{b}^{(j)} \cdot \mathbf{v}^{(j)} \tag{14}$$

which allows us to represent the linear part of the discrete energy $J(\mathbf{v})$ as a linear function.

4. Implementation: Mesh and nodal patches

We describe the typical properties of meshes and nodal patches needed in our computational techniques. Considered finite element meshes consist of triangles (in 2D) and tetrahedra (in 3D), however detailed explanations of Sections 4, 5, 6 address 3D version only.

4.1. The 'mesh' structure

The topology and important attributes of the computational domain are stored in the structure-type data object '**mesh**'. For the example of a tetrahedral mesh displayed in Fig. 2, a vector energy model and full Dirichlet boundary conditions (specified in all three directions and indicated by the nodes in red circles) it provides the following information:
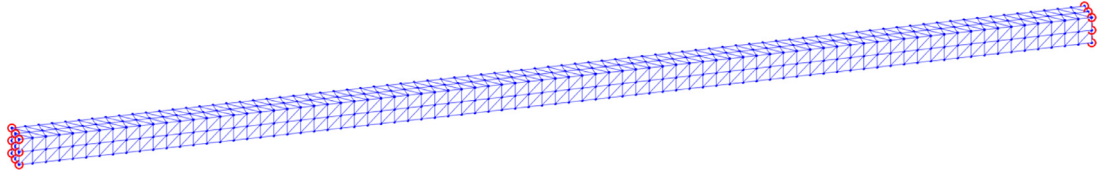
Fig. 2. Example: A tetrahedral mesh of a 3D bar domain with red boundary nodes. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

```
mesh =
 struct with fields:
 dim:                3
  level:             1
  nn:                729
  ne:                1920
 elems2nodes:        [1920x4 double]
 nodes2coord:        [729x3 double]
 volumes:            [1920x1 double]
  dphi:              {[1920x4 double] [1920x4 double] [1920x4 double]}
 nodesDirichlet:     [18x1 double]
 nodesMinim:         [711x1 double]
 dofsDirichlet:      [54x1 double]
 dofsMinim:          [2133x1 double]
```

Parameter **dim** represents the domain dimension (here $dim = 3$) of the problem and **level** the level of a uniform refinement. Higher levels of refinement lead to finer uniformly refined meshes with more elements. The numbers of mesh nodes and elements are provided in **nn** and **ne**. Mesh nodes belonging to each tetrahedron are collected in a matrix **'elems2nodes'** and the Cartesian coordinates of mesh nodes in a matrix **'nodes2coord'**.

Once the last two matrices are given, the codes of [15] generate the **'volumes'** of all tetrahedra together with the restrictions of the partial derivatives (gradients) of all P1-basis functions to tetrahedra stored in a cell **'dphi'** whose components are matrices corresponding to the partial derivates with respect to every x_m, $m \in \{1, \ldots, dim\}$.

The indices of Dirichlet boundary nodes are stored in a vector **'nodesDirichlet'** and the remaining free nodes in a vector **'nodesMinim'**. Vectors **'dofsDirichlet'** and **'dofsMinim'** denote the indices of the degrees of freedom corresponding to Dirichlet and free nodes.

Remark 1. Vectors **'dofsDirichlet'** and **'dofsMinim'** are equal to vectors **'nodesDirichlet'** and **'nodesMinim'** in case of a scalar energy formulation.

4.2. The 'patches' structure

This object is only relevant if the knowledge of gradient $\nabla J(\mathbf{v})$ is required, e.g. as an input of the trust-region method of Section 7. Then, the additional structure-type data object **'patches'** is constructed for the evaluation of its gradient part given by (9).

Remark 2. Only the components of $\nabla J(\mathbf{v})$ corresponding to free nodes are evaluated in our implementation, and the remaining components belonging to the full Dirichlet boundary conditions are omitted. Thus, the nodes with at least one free degree of freedom also belong to the set of free nodes and the gradient is evaluated in all of their components and finally restricted to free degrees of freedom.

We denote by $\mathcal{M}$ a set of all free nodes and by $|\mathcal{M}|$ their number. Then, the node index $i, i \in \{1, \ldots, |\mathcal{M}|\}$, goes exclusively through the free nodes. A nodal patch $\mathcal{T}^i$, $i \in \{1, \ldots, |\mathcal{M}|\}$ is implemented as a vector of elements indices **'elems_i'**, a vector of their volumes **'volumes_i'**, a matrix of the corresponding elements nodes stored as **'elems2nodes_i'**, values of the gradients of local basis functions stores as a cell **'dphi_i'** with matrices components **dphi_i**{1}, ..., **dphi_i**{dim}. All these matrices and vectors are of size $|\mathcal{T}^i| \times (dim + 1)$ and $|\mathcal{T}^i| \times 1$, respectively.

The data of all nodal patches $\mathcal{T}^i$, $i \in \{1, \ldots, |\mathcal{M}|\}$ are then collected in the corresponding long global matrices or vectors with the number of rows equal to

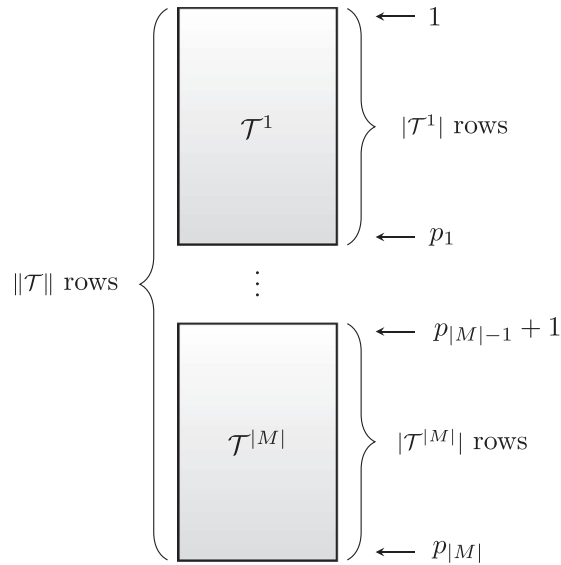$$\|\mathcal{T}\| = \sum_{i=1}^{|\mathcal{M}|} |\mathcal{T}^i|.$$

Fig. 3. All data from nodal patches are stored in long matrices or vectors.

For $i \in \{1, \ldots, |\mathcal{M}|\}$ we define the indices

$$p_i = \sum_{r=1}^{i} |\mathcal{T}^r|, \tag{15}$$

and additionally $p_0 = 0$. Then the submatrix or subvector extracted from rows $(p_{i-1} + 1), \ldots, p_i$ of the global matrices or vector above corresponds to the i-th nodal patch. It is shown schematically in Fig. 3.

Thus we obtain the global vectors **'elems'**, **'volumes'** of size $\|\mathcal{T}\| \times 1$ and the global matrices **'elems2nodes'**, **dphi**$\{1\}$, ..., **dphi**$\{$dim$\}$, of size $\|\mathcal{T}\| \times ($dim $+ 1)$.

Remark 3. For the gradient evaluation of Section 6 we will need to extend a set of p_i, $i \in \{0, \ldots, |\mathcal{M}|\}$, indices up to $\{0, \ldots, d|\mathcal{M}|\}$. Thus, we additionally define

$$\begin{aligned} p_n &= p_{n-|\mathcal{M}|} + \|\mathcal{T}\|, & n &\in \{ |\mathcal{M}| + 1, \ldots, 2|\mathcal{M}|\} \\ p_n &= p_{n-2|\mathcal{M}|} + 2\|\mathcal{T}\|, & n &\in \{2|\mathcal{M}| + 1, \ldots, 3|\mathcal{M}|\}. \end{aligned} \tag{16}$$

In order to maintain the right ordering of the local basis functions within each nodal patch, an additional logical-type matrix of zeros and ones **'logical'** is provided. If the n-th row corresponds to the i-th patch, then **logical**$(n, \ell) = 1$ means that **elems2nodes**$(n, \ell) = i$. Therefore, in every row of **'logical'** matrix the value '1' has exactly one single occurrence.

Below we provide an example of the **'patches'** structure along with the same **'mesh'** one introduced in Subsection 4.1 and corresponding to the domain in Fig. 2.

```
patches =
 struct with fields:
      lengths: [711x1 double]
        elems: [7584x1 double]
      volumes: [7584x1 double]
 elems2nodes: [7584x4 double]
         dphi: {[7584x4 double] [7584x4 double] [7584x4 double]}
      logical: [7584x4 logical]
```

The first vector **'lengths'** is of size $|\mathcal{M}| \times 1$ with entries **lengths**$(i) = |\mathcal{T}^i|$, $i \in \{1, \ldots, |\mathcal{M}|\}$. Here, its size is equal to $711 = 729 - 18$, where $|\mathcal{N}| = 729$ is the number of mesh nodes and 18 is the number of nodes with full Dirichlet boundary conditions.

Benchmark 1. The script benchmark1_start.m generates a sequence of the structures mesh and patches corresponding to each of the uniform mesh refinements. Table 1 provides the assembly times and the memory requirements of both objects.

Table 1

Benchmark 1 - setup times and memory consumption in 3D.

mesh level	number of nodes $|\mathcal{N}|$	number of elems. $|\mathcal{T}|$	number of free dofs	**mesh** setup time [s]	**patches** setup time [s]	**mesh** memory size [MB]	**patches** memory size [MB]
1	729	1920	2133	0.01	0.01	0.30	1.13
2	4025	15,360	11,925	0.02	0.02	2.32	9.07
3	26,001	122,880	77,517	0.06	0.13	18.17	72.73
4	185,249	983,040	554,013	0.51	0.98	144.07	582.53
5	1,395,009	7,864,320	4,178,493	5.56	10.62	1147.67	4663.18

Note that the most memory consuming part of both structures is given by substructures '**dphi**' containing the values of the precomputed gradients of basis functions.

5. Implementation: Energy evaluation

The following matrix-vector transformation is frequently used: matrices $\mathbf{B}, \mathbf{V} \in \mathbb{R}^{|\mathcal{N}| \times d}$ are stretched to the isomorphic vectors

$$\mathbf{b}, \mathbf{v} \in \mathbb{R}^{d|\mathcal{N}|} : \qquad \mathbf{b}_n = \mathbf{B}_{i,j}, \quad \mathbf{v}_n = \mathbf{V}_{i,j}, \qquad n \in \{1, \ldots, d|\mathcal{N}|\}, \tag{17}$$

where $i = (n-1)/d + 1$ and $j = (n-1)\%d + 1$. Here, / symbol is the integer division operator and % is the modulo operator. Put simply, for any $i \in \{1, \ldots, |\mathcal{N}|\}$ the elements of $\mathbf{v}$ with indices $d(i-1) + 1, \ldots, d(i-1) + d$ corresponds to the values of the trial function $\mathbf{v}(\mathbf{x})$ in the i-th node in the directions $1, \ldots, d$.

5.1. The linear energy term J_{lin}

The linear part of the energy (14) rewrites equivalently as

$$J_{lin}(\mathbf{v}) = \int_\Omega \mathbf{f}(\mathbf{x}) \cdot \mathbf{v}(\mathbf{x}) \, d\mathbf{x} = \mathbf{B} : \mathbf{V} = \mathbf{b} \cdot \mathbf{v}, \tag{18}$$

where : denotes the scalar product of matrices.

5.2. The first-gradient energy term J_{grad}

The gradient part of the discrete energy (9) is given as a sum of the energy contributions from every element T_k, $k \in \{1, \ldots, |\mathcal{T}|\}$ and its evaluation in MATLAB is performed effectively by using operations with vectors and matrices only. The energy evaluation for a trial vector $\mathbf{v} \in \mathbb{R}^{d|\mathcal{N}|}$ is performed by the main function:

```matlab
function [e, densities] = energy(v, mesh, params)
% components of deformation
v_cell = createCellFromVector(v, mesh.dim);

% components of deformation on elements
v_elems = CellAtMatrixOfIndices(v_cell, mesh.elems2nodes);

% deformation gradients on elements
F_elems = evaluate_F(mesh, v_elems);

% gradient densities on elements
densities.Gradient = densityGradientVector_3D(F_elems, params);

% total gradient energy
e = sum(mesh.volumes.*densities.Gradient);
end
```

The structure '**mesh**' is described in Section 5 and the '**params**' contains material parameters apart from some other parameters (e.g. visualization parameters). The code above is vectorized and generates the objects:

- A cell '**v_elems**' containing matrices **v_elems**{1}, **v_elems**{2}, **v_elems**{3} of size $|\mathcal{T}| \times 4$ providing the restrictions of nodal deformations to all elements.
- A cell '**F_elems**' of size $\dim \times \dim$ storing the deformation gradients (see (3)) in all elements. In particular, **F_elems**{d}{m} is then a vector of size $|\mathcal{T}| \times 1$ evaluating the partial derivatives of the d-th component of deformation with respect to the m-th variable in all elements.
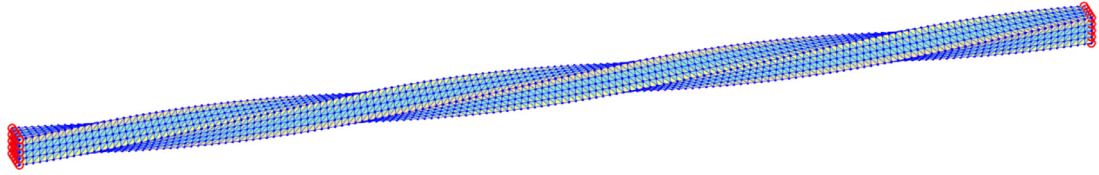
Fig. 4. A bar domain twisted by the prescribed deformation (19).

A vector '**densities.Gradient**' of size $|\mathcal{T}| \times 1$ containing gradient densities in all elements.

The energy **e** is given as a sum of the gradient energy contributions over the elements (the gradient part $J_{grad}(\mathbf{v})$ given by (9)) subtracted by the linear energy term $J_{lin}(\mathbf{v})$ (given by (14)).

The Neo-Hookean density function $W = W(F)$ from (4) is implemented as

```
1  function densities=densityGradientVector_3D(F,params)
2  % determinant term
3  DET = F{1,1}.*F{2,2}.*F{3,3} + F{1,3}.*F{2,1}.*F{3,2} + ...
4        F{1,2}.*F{2,3}.*F{3,1} − F{1,3}.*F{2,2}.*F{3,1} − ...
5        F{1,2}.*F{2,1}.*F{3,3} − F{1,1}.*F{2,3}.*F{3,2};
6
7  % I1 term (Frobenius norm squared)
8  I1 = F{1,1}.^2 + F{1,2}.^2 + F{1,3}.^2 + ...
9       F{2,1}.^2 + F{2,2}.^2 + F{2,3}.^2 + ...
10      F{3,1}.^2 + F{3,2}.^2 + F{3,3}.^2;
11
12 % gradient densities
13 densities = params.C1*(I1−3−2*log(DET)) + params.D1*(DET−1).^2;
14 end
```

Benchmark 2. Assume a bar domain $\Omega = (0, l_x) \times (-\frac{l_y}{2}, \frac{l_y}{2}) \times (-\frac{l_z}{2}, \frac{l_z}{2})$, where $l_x = 0.4, l_y = l_z = 0.01$, specified by the material parameters $E = 2 \cdot 10^8$ (Young's modulus) and $\nu = 0.3$ (Poisson's ratio) and deformed by the prescribed deformation $\mathbf{v}(\mathbf{x}) = \mathbf{v}(x, y, z)$ given by

$$
\begin{aligned}
v^{(1)}(x, y, z) &= x, \\
v^{(2)}(x, y, z) &= \cos(\alpha \tfrac{x}{l_x}) y + \sin(\alpha \tfrac{x}{l_x}) z, \\
v^{(3)}(x, y, z) &= -\sin(\alpha \tfrac{x}{l_x}) y + \cos(\alpha \tfrac{x}{l_x}) z,
\end{aligned}
\tag{19}
$$

or, equivalently, using matrix operations

$$
\begin{pmatrix} v^{(1)}(x, y, z) \\ v^{(2)}(x, y, z) \\ v^{(3)}(x, y, z) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha \tfrac{x}{l_x}) & \sin(\alpha \tfrac{x}{l_x}) \\ 0 & -\sin(\alpha \tfrac{x}{l_x}) & \cos(\alpha \tfrac{x}{l_x}) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}.
\tag{20}
$$

Here, $\alpha = 2\pi$ means that the right Dirichlet wall is twisted once around the x-axes (Fig. 4).

Constants C_1, D_1 are transformed according to the formulas $C_1 = \frac{\mu}{2}, D_1 = \frac{K}{2}$, where $\mu = \frac{E}{2(1+\nu)}$ is the Shear modulus and $K = \frac{E}{3(1-2\nu)}$ is the bulk modulus. The exact evaluation shows that the corresponding gradient energy using the Neo-Hookean density function (4) reads

$$
J_{grad}(\mathbf{v}) = \frac{\alpha^2 C_1 l_y l_z (l_y^2 + l_z^2)}{12 l_x} \approx 6.326670.
$$

The script `benchmark2_start.m` evaluates an approximation of $J_{grad}(\mathbf{v})$ on the sequence of the uniform mesh refinements defined in Benchmark 1. In order to provide higher accuracy of the evaluation times, the energies are recomputed 10 times. Table 2 provides evaluation times and values of the energy approximations (times for setting up the '**mesh**' structure are not included).

6. Implementation: Energy gradient evaluation

Evaluation of the full gradient $\nabla J(\mathbf{v})$, where $\mathbf{v} = (v_1, \dots, v_{d|\mathcal{N}|}) \in \mathbb{R}^{d|\mathcal{N}|}$, requires in general computation of all partial derivatives $\frac{\partial J(\mathbf{v})}{\partial v_n}$, $n \in \{1, \dots, d|\mathcal{N}|\}$. Some minimization methods (e.g. trust-region mentioned applied in Section 7) require the knowledge of gradient, but only restricted to free degrees of freedom.

Table 2
Benchmark 2 - energy evaluation times in 3D.

mesh level	number of free dofs	evaluation (10x) of $J_{grad}(\mathbf{v})$: time [s]	value of $J_{grad}(\mathbf{v})$
1	2133	0.01	12.6623
2	11,925	0.02	7.9083
3	77,517	0.08	6.7219
4	554,013	0.76	6.4255
5	4,178,493	12.82	6.3514

The gradient $\nabla J_{lin}(\mathbf{v})$ easily reads using (18)

$$\nabla J_{lin}(\mathbf{v}) = \mathbf{b}, \tag{21}$$

where $\mathbf{b}$ is of size $d|\mathcal{N}|$, given by (17) and its restriction to free degrees of freedom is then trivial.

The evaluation of $\nabla J_{grad}(\mathbf{v})$ is technically more involved. Firstly, it is evaluated with respect to Remark 2 for all degrees of freedom belonging to all (at least partially) free nodes. Secondly, it is restricted to free degrees of freedom only. In order to determine to which node the n-th active degree of freedom belongs we define the index mapping

$$I_{DN} : \{1, \ldots, d|\mathcal{M}|\} \to \{1, \ldots, |\mathcal{M}|\}, \qquad I_{DN}(n) = (n-1)/d + 1 \tag{22}$$

which for the n-th active degree of freedom returns the corresponding i-th free node (here / is an integer division operator). By

$$\mathcal{T}^{(n)}, \qquad n \in \{1, \ldots, d|\mathcal{M}|\},$$

we denote the set of elements adjacent to node $N_{I_{DN}(n)}$ belonging to the n-th active degree of freedom and by $|\mathcal{T}^{(n)}|$ their number. The gradient of the nonlinear part $J_{grad}(\mathbf{v})$ can be computed in two different ways:

1. numerically, where the partial derivatives are computed approximately by using a difference scheme.
2. exactly by taking the explicit partial derivatives.

Deriving the exact partial derivatives can be demanding and it depends on the particular problem. On the contrary, the numerical approach is more general and is feasible regardless of the complexity of the function representing the corresponding discrete energy. Hence, we first describe the numerical approach by using the central difference scheme and then explain the gradient evaluation by deriving the explicit form of the partial derivatives.

6.1. Numerical approach to evaluate $\nabla J_{grad}(\mathbf{v})$

By using the central difference scheme, one can write

$$\frac{\partial}{\partial v_n} J_{grad}(\mathbf{v}) \approx \frac{J_{grad}(\mathbf{v} + \varepsilon \mathbf{e}_n) - J_{grad}(\mathbf{v} - \varepsilon \mathbf{e}_n)}{2\varepsilon}, \tag{23}$$

where $\mathbf{e}_n$ is the n-th canonical vector in $\mathbb{R}^{d|\mathcal{M}|}$ and ε is a small positive number. Both summands in the numerator above can be directly evaluated by taking the energy evaluation procedure introduced in the previous subsection as

$$J_{grad}(\mathbf{v} + \varepsilon \mathbf{e}_n) - J_{grad}(\mathbf{v} - \varepsilon \mathbf{e}_n) = \sum_{k=1}^{|\mathcal{T}|} \int_{T_k} W(\nabla(\mathbf{v} + \varepsilon \mathbf{e}_n)) \, d\mathbf{x} - \sum_{k=1}^{|\mathcal{T}|} \int_{T_k} W(\nabla(\mathbf{v} - \varepsilon \mathbf{e}_n)) \, d\mathbf{x}.$$

However, this approach is ineffective as long as the step of the central difference scheme ε occurs only in a few summands of the sums representing $J_{grad}(\mathbf{v} + \varepsilon \mathbf{e}_n)$ and $J_{grad}(\mathbf{v} - \varepsilon \mathbf{e}_n)$ given by (9), while the remaining summands are the same and therefore vanish. Hence, we can further simplify

$$J_{grad}(\mathbf{v} + \varepsilon \mathbf{e}_n) - J_{grad}(\mathbf{v} - \varepsilon \mathbf{e}_n) =$$

$$= \sum_{k=1}^{|\mathcal{T}^{(n)}|} \int_{T_k^{(n)}} W(\nabla(\mathbf{v} + \varepsilon \mathbf{e}_n)) \, d\mathbf{x} - \sum_{k=1}^{|\mathcal{T}^{(n)}|} \int_{T_k^{(n)}} W(\nabla(\mathbf{v} - \varepsilon \mathbf{e}_n)) \, d\mathbf{x} =$$

$$= \sum_{k=1}^{|\mathcal{T}^{(n)}|} |T_k^{(n)}| W\left(\nabla(\mathbf{v} + \varepsilon \mathbf{e}_n)\big|_{T_k^{(n)}}\right) - \sum_{k=1}^{|\mathcal{T}^{(n)}|} |T_k^{(n)}| W\left(\nabla(\mathbf{v} - \varepsilon \mathbf{e}_n)\big|_{T_k^{(n)}}\right). \tag{24}$$

By using the substitutions

$$J_{grad}^{(n),-}(\mathbf{v}) = |T_k^{(n)}| W\left(\nabla(\mathbf{v} - \varepsilon \mathbf{e}_n)\big|_{T_k^{(n)}}\right), \quad J_{grad}^{(n),+}(\mathbf{v}) = |T_k^{(n)}| W\left(\nabla(\mathbf{v} + \varepsilon \mathbf{e}_n)\big|_{T_k^{(n)}}\right), \tag{25}$$

one can rewrite (24)

$$J_{grad}(\mathbf{v} + \varepsilon\mathbf{e}_n) - J_{grad}(\mathbf{v} - \varepsilon\mathbf{e}_n) = \sum_{k=1}^{|\mathcal{T}^{(n)}|} J_{grad}^{(n),+}(\mathbf{v}) - \sum_{k=1}^{|\mathcal{T}^{(n)}|} J_{grad}^{(n),-}(\mathbf{v}) \qquad (26)$$

and evaluate the whole $\nabla J_{grad}(\mathbf{v})$ via the simple for-loop over its components.

Remark 4. The energy evaluation procedure has to be called in every loop over the vector components $n \in \{1, \ldots, d|\mathcal{M}|\}$ and it turned out to cause multiple self built-in times that slowed down performance. To avoid that, the original energy evaluation procedure is modified so that the multiple input vectors can be processed simultaneously and the energy evaluation procedure be called only once.

The outer gradient evaluation procedure is simple:

```
1  eps = 1e−8;   % finite difference step size
2
3  % local patch energies for +eps and −eps
4  es_minsplus = energies(v,[−eps eps],mesh,patches,params,indx);
5
6  % the gradient vector by the central difference scheme
7  g = diff(es_minsplus,1,2)/eps/2;
```

Here '**es_minsplus**' is a matrix of size $3\|\mathcal{T}\| \times 2$ with the components given by (25):

es_minsplus $(n, 1) = J_{grad}^{(n),-}(\mathbf{v})$,

es_minsplus $(n, 2) = J_{grad}^{(n),+}(\mathbf{v})$.

The gradient vector **g** is then assembled by using the central difference scheme (23) with a constant central difference step size ε for all $n \in \{1, \ldots, d|\mathcal{M}|\}$. Obviously, a generalization of the above code to higher accuracy difference schemes is possible.

We recall a cell '**v_elems**' containing matrices

v_elems {1}, **v_elems**{2}, **v_elems**{3}   of size $|\mathcal{T}| \times 4$ is assembled in the energy evaluation procedure **energy** from Subsection 5.2. For the evaluation of (23) by using (26) we need to assemble a cell-structure '**v_patches**' containing matrices

v_patches {1}, **v_patches**{2}, **v_patches**{3}   of size $\|\mathcal{T}\| \times 4$ that provide the restrictions of nodal deformations to all patches. In fact, the structure '**v_patches**' copies parts of '**v_elems**' to the particular positions.

The extended procedure **energies** evaluates all ε-perturbed values of energies:

```
1  function e = energies(v,eps,mesh,patches,params,indx)
2  % components of deformation
3  v_cell = createCellFromVector(v,mesh.dim);
4  % components of deformation on elements
5  v_elems = CellAtMatrixOfIndices(v_cell,mesh.elems2nodes);
6  % deformation gradients on elements
7  F_elems = evaluate_F(mesh,v_elems);
8
9  % deformations on patches
10 v_patches = CellAtMatrixOfIndices(v_cell,patches.elems2nodes);
11 % deformation gradients on patches
12 F_patches = CellAtMatrixOfIndices(F_elems,patches.elems);
13
14 v_patches_eps = cell(3,1);
15 e = zeros(3*numel(mesh.nodesMinim),size(eps,2));
16 for comp=1:size(eps,2)  % loop over epsilon perturbations
17     % perturbations of deformations on patches
18     v_patches_eps{1} = v_patches{1} + eps(comp)*patches.logical;
19     v_patches_eps{2} = v_patches{2} + eps(comp)*patches.logical;
20     v_patches_eps{3} = v_patches{3} + eps(comp)*patches.logical;
21
22     % deformation gradients of perturbation
23     GG = evaluate_GG(patches,v_patches_eps,F_patches);
24
25     % densities
26     densities_patches = densityGradientVector_3D(GG,params);
27     % energies
28     e_patches = [patches.volumes; patches.volumes; patches.volumes].*
                densities_patches;
29     % final energy values on patches
30     csep = cumsum(e_patches);
31     e([1:3:end 2:3:end 3:3:end],comp) = [csep(indx(1)); diff(csep(indx))];
32 end
33 e = e(mesh.dofsMinim_local,:);   % dropping out the Dirichlet dofs
34 end
```
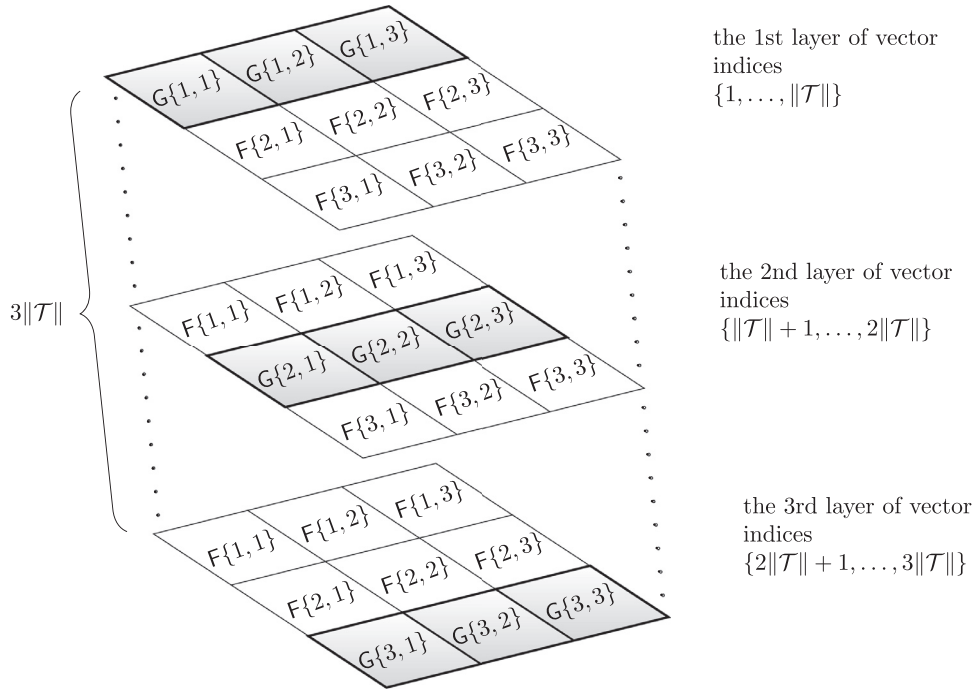
the 1st layer of vector
indices
$\{1, \ldots, \|\mathcal{T}\|\}$

the 2nd layer of vector
indices
$\{\|\mathcal{T}\| + 1, \ldots, 2\|\mathcal{T}\|\}$

the 3rd layer of vector
indices
$\{2\|\mathcal{T}\| + 1, \ldots, 3\|\mathcal{T}\|\}$

Fig. 5. The '**GG**' structure.

The cells '**v_elems**' and '**F_elems**' are the same as in the first-gradient energy evaluation procedure from Subsection 5.2. The cell '**v_patches_eps**' contains matrices

v_patches_eps {1}, **v_patches_eps**{2}, **v_patches_eps**{3}   of size $\|\mathcal{T}\| \times 4$ that provide the restrictions of nodal deformations to all patches, but here these deformations are perturbed by the value of the central difference step ε. This cell is used for the construction of the key cell '**GG**' of size $\dim \times \dim$ containing vectors

GG $\{j, m\}$ of length $\dim \|\mathcal{T}\|$ storing the partial derivatives of deformations of the j-th component with respect to the m-th variable. The structure of '**GG**' is in details displayed in Fig. 5.

All three layers of vector indices have the same size of $\|\mathcal{T}\|$ entries and the following meaning:

- the 1st layer corresponds to the ε-perturbation of the component $\mathbf{v}^{(1)}$,
- the 2nd layer corresponds to the ε-perturbation of the component $\mathbf{v}^{(2)}$,
- the 3rd layer corresponds to the ε-perturbation of the component $\mathbf{v}^{(3)}$.

The cell '**G**' has the same size as '**F**', but contains deformation gradient matrices corresponding to the deformations perturbed by ε. The most important feature is using the values of the precomputed input cell '**F**' for the efficient construction of the '**GG**' cell. Note that if the numeric difference step ε is added or subtracted from the first component of displacements, the deformation gradients of the rest two components remain the same and their values are already stored in the '**F**' cell.

A vector '**csep**' of length $d\|\mathcal{T}\|$ contains the cumulative sums of '**e_patches**' with elements

$$\mathbf{csep}(n) = \sum_{r=1}^{n} \mathbf{e\_patches}(r).$$

An output matrix $\mathbf{e}$ of size $3|\mathcal{M}| \times 2$ contains all energy contributions and for any $comp \in \{1, 2\}$

$$\mathbf{e}(n, comp) = \sum_{r=p_{n-1}+1}^{p_n} \mathbf{e\_patches}(r) = \mathbf{csep}(p_n) - \mathbf{csep}(p_n - 1). \tag{27}$$

Note that vector '**e_patches**' changes its value inside the loop over the components $comp$. Therefore, the whole vector $\mathbf{e}$ is evaluated at once using the Matlab difference function **diff** with the input vector '**indx**' of length $3|\mathcal{M}|$, where $\mathbf{indx}(n) = p_n, n \in \{1, \ldots, d|\mathcal{M}|\}$, with p_n defined in (15) and additionally in (16).

The procedure for the construction of **'GG'** is listed below:

```
1  function GG = evaluate_GG(patches, v_patches, F)
2  % deformation gradients on patches
3  G = evaluate_F(patches, v_patches);
4
5  % deformation gradient on patches and on 3 components
6  GG = cell(3,3);
7  GG{1,1} = [G{1,1}; F{1,1}; F{1,1}];
8  GG{1,2} = [G{1,2}; F{1,2}; F{1,2}];
9  GG{1,3} = [G{1,3}; F{1,3}; F{1,3}];
10 GG{2,1} = [F{2,1}; G{2,1}; F{2,1}];
11 GG{2,2} = [F{2,2}; G{2,2}; F{2,2}];
12 GG{2,3} = [F{2,3}; G{2,3}; F{2,3}];
13 GG{3,1} = [F{3,1}; F{3,1}; G{3,1}];
14 GG{3,2} = [F{3,2}; F{3,2}; G{3,2}];
15 GG{3,3} = [F{3,3}; F{3,3}; G{3,3}];
16 end
```

6.2. Exact approach to evaluate $\nabla J_{grad}(\mathbf{v})$

Evaluation of the exact $\nabla J(\mathbf{v})$ requires an explicit deriving of every $\frac{\partial J(\mathbf{v})}{\partial v_n}$, $n \in \{1, \ldots, d|\mathcal{M}|\}$. The gradient of the linear part is trivial and is given by (18). Using (9) one can write

$$\frac{\partial J_{grad}(\mathbf{v})}{\partial v_n} = \sum_{k=1}^{|\mathcal{T}|} |T_k| \frac{\partial}{\partial v_n} W\left(\mathbf{F}(\mathbf{v})|_{\mathbf{T_k}}\right). \tag{28}$$

Note that the only elements whose energy contributions depend on v_n are those belonging to the i-th patch, where $i = I_{DN}(n)$. Therefore, we can simplify the equation above as

$$\frac{\partial J_{grad}(\mathbf{v})}{\partial v_n} = \sum_{k=1}^{|\mathcal{T}^{(n)}|} |T_k| \frac{\partial}{\partial v_n} W\left(\mathbf{F}(\mathbf{v})|_{\mathbf{T_k}}\right). \tag{29}$$

By using the chain rule one can write

$$\frac{\partial}{\partial v_n} W\left(\mathbf{F}(\mathbf{v})\right) = \sum_{j=1}^{d} \sum_{m=1}^{\dim} \frac{\partial W}{\partial \mathbf{F}_{j,m}}\left(\mathbf{F}(\mathbf{v})\right) \frac{\partial \mathbf{F}_{j,m}}{\partial v_n}(\mathbf{v}), \tag{30}$$

where assuming the Neo-Hookean density from (4)

$$\frac{\partial W(\mathbf{F})}{\partial \mathbf{F}_{j,m}} = C_1 \left(\frac{\partial \mathbf{I}_1(\mathbf{F})}{\partial \mathbf{F}_{j,m}} - \frac{2}{\det(\mathbf{F})} \frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}_{j,m}}\right) + 2D_1\left(\det(\mathbf{F}) - 1\right)\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}_{j,m}}. \tag{31}$$

Using the row or the column expansion rule for calculating the determinant of a 3×3 matrix, one can express

$$\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}_{j,m}} = (-1)^{j+m} \det(\mathbf{F}_{j,m}^{sub}), \tag{32}$$

where $\mathbf{F}_{j,m}^{sub}$ is the 2×2 submatrix of $\mathbf{F}$ given by dropping the j-th row and the m-th column of $\mathbf{F}$. Futhermore, $\frac{\partial \mathbf{I}_1}{\partial \mathbf{F}_{j,m}}$ can be also simplified by

$$\frac{\partial \mathbf{I}_1(\mathbf{F})}{\partial \mathbf{F}_{j,m}} = 2\mathbf{F}_{j,m}. \tag{33}$$

From Subsection 6.1 we already know how to assemble $\mathbf{v}|_{T_k}$ and $\mathbf{F}(\mathbf{v})|_{T_k}$. Here, in addition, we need to express $\frac{\partial \mathbf{I}_1(\mathbf{F})}{\partial \mathbf{F}}$, $\frac{\partial \det(\mathbf{F})}{\partial \mathbf{F}}$ and $\frac{\partial \mathbf{F}}{\partial v_n}$.

Similarly to the numeric gradient evaluation, the exact gradient evaluation procedure uses the same precomputed structures **'v_cell'**, **'v_elems'** and **'F_elems'**.

```
1  % components of deformation
2  v_cell = createCellFromVector(v, mesh.dim);
3  % components of deformation on elements
4  v_elems = CellAtMatrixOfIndices(v_cell, mesh.elems2nodes);
5  % deformation gradients on elements
6  F_elems = evaluate_F(mesh, v_elems);
```

Table 3
Benchmark 3 - energy gradient evaluation times in 3D.

mesh level	number of free dofs	evaluation (10x) of exact $\nabla J(\mathbf{v})$: time [s]	evaluation (10x) of num. $\nabla J(\mathbf{v})$: time [s]
1	2133	0.12	0.08
2	11,925	0.14	0.15
3	77,517	1.18	1.51
4	554,013	12.47	15.64
5	4,178,493	487.34	533.24

The main procedure for the exact gradient evaluation is provided below:

```
1  % deformations on patches
2  v_patches = CellAtMatrixOfIndices(v_cell,patches.elems2nodes);
3  % deformation gradients on patches
4  F_patches = CellAtMatrixOfIndices(F_elems,patches.elems);
5
6  % deformation gradients of perturbation
7  GG = evaluate_GG(patches,v_patches,F_patches);
8
9  % densities
10 D_densities_patches = density_D_GradientVector_3D(GG,D_F,params);
11 % energies
12 D_e_patches = [patches.volumes; patches.volumes; patches.volumes].*
      D_densities_patches;
13 % final energy values on patches
14 csDep = cumsum(D_e_patches);
15 g = zeros(3*numel(nodesMinim),1);
16 g([1:3:end 2:3:end 3:3:end]) = [csDep(indx(1)); diff(csDep(indx))];
17 g = g(dofsMinim_local);
```

Here, **'D_densities_patches'** is a vector of size $3\|\mathcal{T}\| \times 1$ containing all $\frac{\partial W(\mathbf{F}(\mathbf{v}))}{\partial v_n}$ for every $n \in \{1, \ldots, d|\mathcal{M}|\}$ given by (30). Note that the function **density_D_GradientVector_3D** uses the input 3×3 cell structure **'D_F_patches'**, where **D_F_patches**$\{j, m\}(n) = \frac{\partial \mathbf{F}_{j,m}}{\partial v_n}$.

Benchmark 3. The script benchmark3.m evaluates approximations of $\nabla J_{grad}(\mathbf{v})$ on the sequence of the uniform mesh refinements defined in Benchmark 2. In order to provide higher accuracy of evaluation times, the energies are recomputed 10 times. Table 3 provide evaluation times of the gradient approximations both in the exact and numerical case (the times for setting up **'mesh'** and **'patches'** structures are not included).

7. Application to practical energy minimizations

For a practical energy minimization, the trust-region method [4] available in the MATLAB Optimization Toolbox is utilized. Standard stopping criteria (the first-order optimality, tolerance on the argument and tolerance on the function) equal to 10^{-6} are considered in all benchmarks. The method also allows to specify the sparsity pattern of the Hessian matrix $\nabla^2 J(v)$ in free degrees of freedom, i.e., only positions (indices) of nonzero entries. The sparsity pattern is directly given by a finite element discretization.

Example 2. The sparsity pattern related to the 3D bar domain of Fig. 2 is displayed in Fig. 6 (left). It corresponds to the FEM mesh not taking the Dirichlet boundary conditions into account. Therefore, it is of size 729×729. In practical computations, it is first extended from a scalar to a vector problem and then restricted to the **'mesh.dofsMinim'** indices (right). Since there are 18 Dirichlet nodes fixed in all three directions, the corresponding number of rows and columns of the right hessian sparsity pattern is $(729 - 18) * 3 = 2133$.

7.1. Benchmark 4: Time-dependent hyperelasticity in 3D

We consider the elastic bar specified in Benchmark 2 with a time-dependent nonhomogeneous Dirichlet boundary conditions on the right wall ($x = l_x$). The torsion of the right wall is described by the formula (19) for $x = l_x = 0.4$, where the rotation angle is assumed to be a time dependent function $\alpha = \alpha(t) = \alpha_{\max} t/T$ for a sequence of integer discrete times $t \in \{1, 2, \ldots, T\}$. Here we assume the final time $T = 24$ and the maximal angle of rotation $\alpha_{\max} = 8\pi$ ensuring four full rotations of the right Dirichlet wall around the x-axis. Altogether we solve a sequence of T minimization problems (1) with the Neo-Hook energy density (4) and no loading ($\mathbf{f} = 0$).

The trust-region method accepts an initial deformation approximation. For the first discrete time we run iterations from the identity $v(x) = x, x \in \Omega$ and for the next discrete time the minimizer of the previous discrete time problem serves as its initial approximation. We output the found energy minimizer for each minimization problem with some minimizers shown in form of the deformed meshes in Fig. 7. Additionally, we also provide the minimization time, the number of the trust-region iterations and the value of the corresponding minimal energy.
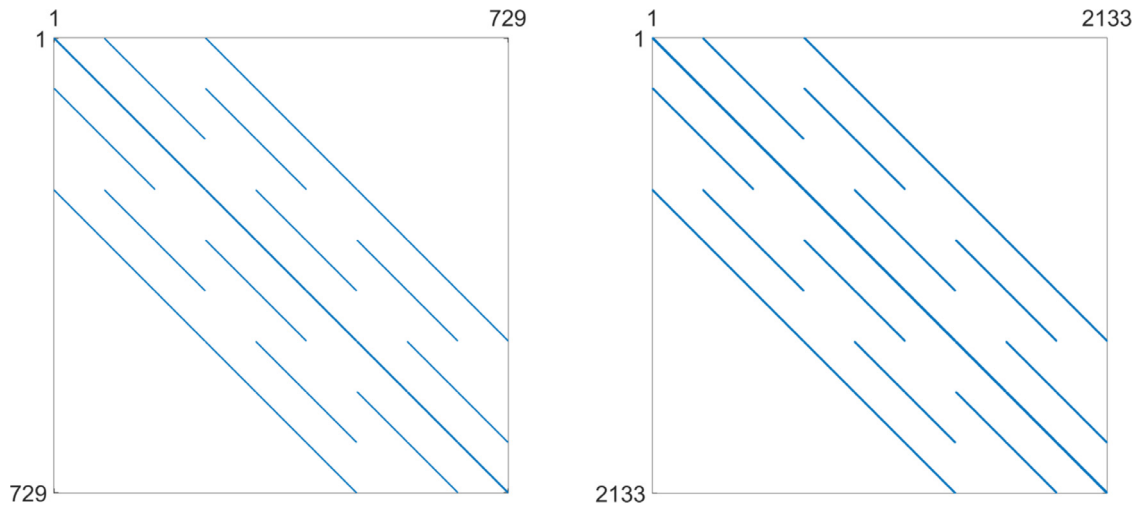
Fig. 6. The sparsity pattern of the 3D bar domain based on nodes connectivity (left) and its extension to the vector problem (right).

Table 4
Benchmark 4 - performance of hyperelasticity minimizations in 3D.

step	level 1, 2133 free dofs			level 2, 11,925 free dofs			level 3, 77,517 free dofs		
	time [s]	iters	$J_{grad}(\mathbf{u})$	time [s]	iters	$J_{grad}(\mathbf{u})$	time [s]	iters	$J_{grad}(\mathbf{u})$
t=3	6.65	40	3.1177	66.23	47	1.8244	2148.62	93	1.4631
t=6	10.42	60	12.4423	117.74	76	7.2962	1404.68	68	5.8533
t=9	10.33	60	27.8993	125.76	85	16.4079	1441.84	73	13.1735
t=12	18.43	104	49.5506	102.11	68	29.1664	1277.04	65	23.4286
t=15	15.51	81	77.3859	127.17	87	45.5677	1130.02	58	36.6236
t=18	18.31	97	111.3369	110.65	75	65.5470	1540.90	76	52.7641
t=21	15.74	82	151.4946	74.47	51	89.1672	5523.64	290	71.7883
t=24	16.37	91	197.7577	91.79	63	116.3261	2660.44	131	93.7238

The overall performance is explained in Table 4 for separate computations on tetrahedral meshes of levels 1, 2, 3 applying the numerical gradient evaluation of Subsection 6.1 only with the choice $\varepsilon = 10^{-6}$. We notice that the minimizations of finer meshes require only slightly higher number of iterations, which is acceptable. Comparison of the values $J_{grad}(\mathbf{u})$ in each line of Table 4 indicates convergence in space of the energy minimizers.

7.2. Benchmarks 5 and 6: Hyperelasticity and p-Laplacian in 2D

Although our exposition is mainly focused on implementation details in 3D, a reduction to 2D (dim = 2) is straightforward.

As the first example we consider a square domain with lengths $l_x = l_y = 2$ perforated by a hole with radius $r = 1/3$ (Fig. 8 left) and whose center is located at the center of the square, subjected to zero Dirichlet boundary condition on the bottom and left edges, a constant loading $\mathbf{f}(\mathbf{x}) = (-3.5 \cdot 10^7, -3.5 \cdot 10^7)$ and the elastic parameters specified in Benchmark 2. We assume the choice $\varepsilon = 10^{-6}$ in the evaluation of the numerical gradient. The resulting deformation and the deformation gradient densities are depicted in Fig. 8 (right).

As the second example we minimize the p-Laplacian energy functional (5) over the L-shape domain (Fig. 9 left). The constant loading $f = -10$ is assumed together with zero Dirichlet boundary conditions on the full domain boundary. For the numerical gradient we assume $\varepsilon = 10^{-5}$. The solution is displayed in Fig. 9 (right). Table 6 depicts performance of all options for the power $p = 3$ and confirms faster evaluation times in comparison to our former contribution [13]. This is mainly due to the improved vectorization concepts here, due to the faster CPU and finally due to updates of the trust-region in-built implementation in the latest version of Matlab.

7.3. Implementation remarks and outlooks

Assembly times in all benchmarks were obtained on a MacBook Air (M1 processor, 2020) with 16 GB memory running MATLAB R2021a. Our implementation is available at https://www.mathworks.com/matlabcentral/fileexchange/97889
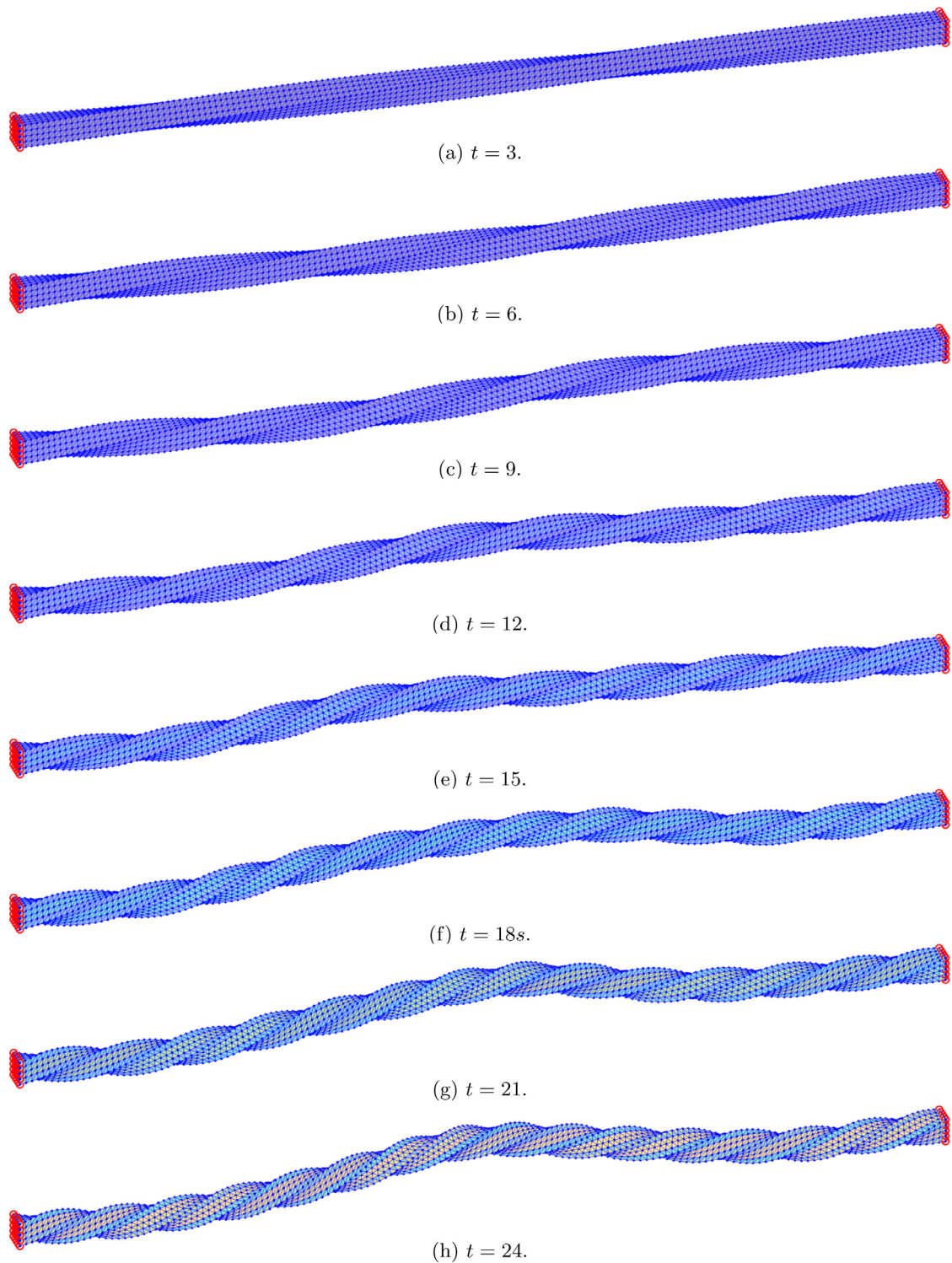
(a) $t = 3$.

(b) $t = 6$.

(c) $t = 9$.

(d) $t = 12$.

(e) $t = 15$.

(f) $t = 18s$.

(g) $t = 21$.

(h) $t = 24$.

Fig. 7. Benchmark 4 - deformation of the tetrahedal mesh subjected to the right time-rotating Dirichlet plane with the underlying Neo-Hookean density.
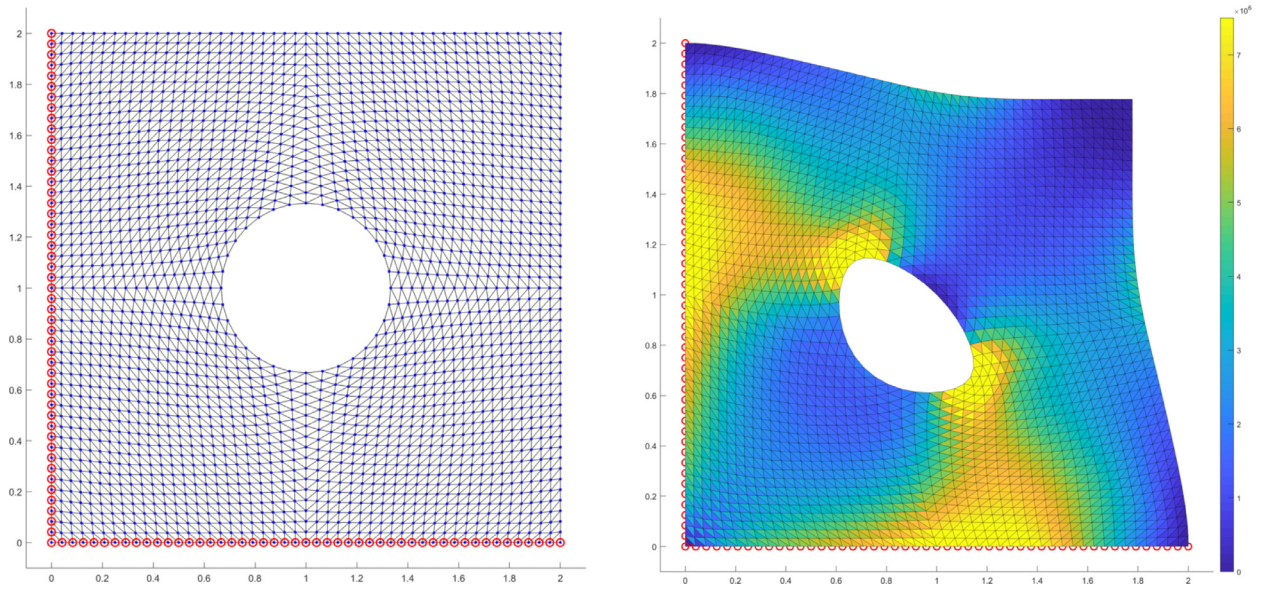
Fig. 8. Benchmark 5 - a triangulation of the square domain perforated by a hole (left) and its deformation with the underlying Neo-Hookean density (right).
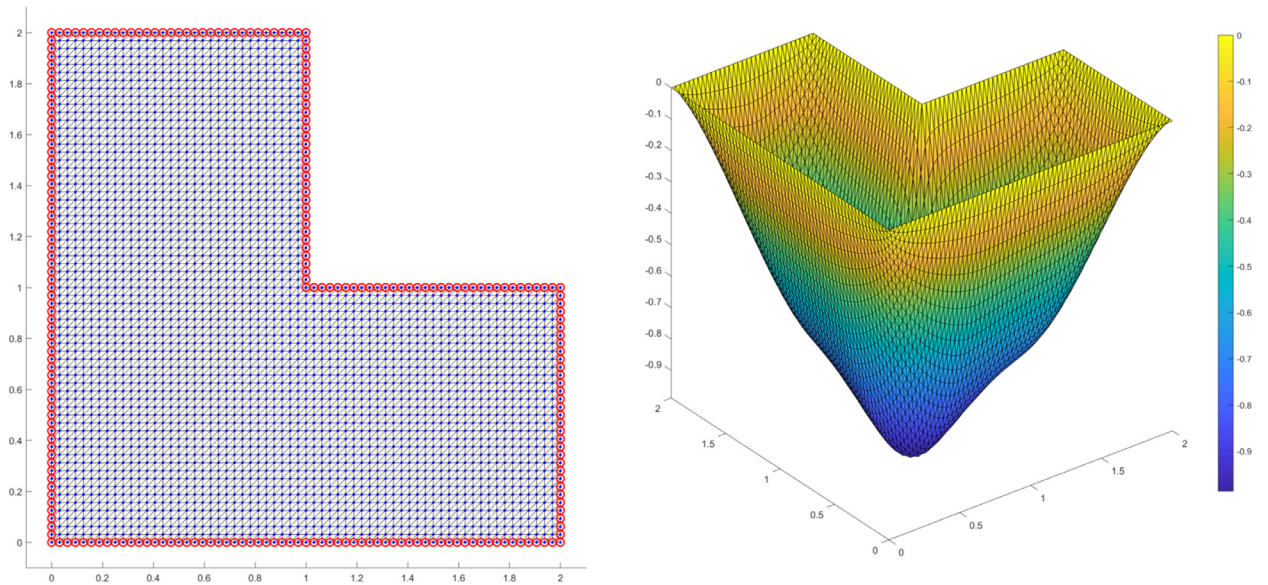

Fig. 9. Benchmark 6 - a triangulation of the L-shape domain (left) and the solution p-Laplacian for the power $p = 3$ and a constant loading $f(\mathbf{x}) = -10$ (right).

Table 5
Benchmark 5 - performance of hyperelasticity minimizations in 2D. True values of the energies are multiplied by the scale factor of 10 million.

		exact gradient			numerical gradient		
level	free dofs	time [s]	iters	$J(\mathbf{u})$	time [s]	iters	$J(\mathbf{u})$
1	278	0.36	4	24.5745	0.13	4	24.5745
2	1102	0.15	4	24.3558	0.16	4	24.3558
3	4382	0.54	4	24.2957	0.54	4	24.2957
4	17,470	2.35	5	24.2799	2.65	5	24.2799
5	69,758	11.27	5	24.2758	12.39	5	24.2758
6	278,782	83.27	6	24.2748	88.74	6	24.2748

Table 6
Benchmark 6 - performance of p-Laplacian minimizations for $p = 3$ in 2D.

		exact gradient			numerical gradient		
level	free dofs	time [s]	iters	$J(\mathbf{u})$	time [s]	iters	$J(\mathbf{u})$
1	33	0.02	8	−7.5353	0.03	8	−7.5353
2	161	0.05	11	−7.9729	0.12	15	−7.9729
3	705	0.11	11	−8.1039	0.19	11	−8.1039
4	2945	0.30	11	−8.1445	0.56	12	−8.1445
5	12,033	1.50	11	−8.1578	2.08	12	−8.1578
6	48,641	6.30	12	−8.1625	9.48	12	−8.1625
7	195,585	48.61	13	−8.1642	60.62	12	−8.1642
8	784,385	617.92	22	−8.1649	672.04	16	−8.1649

for download and testing. It is based on own codes of [3,6,15] used primarily for assemblies of finite element matrices. It also utilizes the function **mcolon** from the reservoir simulator [11]. The names of mesh attributes were initially motivated by codes of [1] and further modified.

The 3D cuboid mesh is generated by the code of [6] and 2D meshes (the square with the hole and the L-shape) by our own code. Vectorized evaluations of gradients of basis functions are taken from [3,15]. The general structure of the code was initially taken from [13] and extended to the implementation of hyperelasticity in 2D and 3D. Many parts of this code (mainly related to the gradient evaluation) have been significantly improved since then.

The code is designed in a modular way allowing to add different scalar (e.g. Ginzburg-Landau [1]) and vector (e.g. topology optimization with elastoplasticity [2], shape memory alloys [8]) problems involving the first gradient energy terms.

Acknowledgments

References

[1] J. Alberty, C. Carstensen, S. Funken, Remarks around 50 lines of Matlab: short finite element implementation, Numer Algorithms 20 (1999) 117–137.
[2] S. Almi, U. Stefanelli, Topology optimization for incremental elastoplasticity: a phase-field approach, SIAM J. Control Optim. 59 (1) (2021) 339–364.
[3] I. Anjam, J. Valdman, Fast MATLAB assembly of FEM matrices in 2d and 3d: edge elements, Appl Math Comput 267 (2015) 252–263.
[4] A.R. Conn, N.I.M. Gould, P.L. Toint, Trust-Region Methods, SIAM, Philadelphia, 2000.
[5] P.G. Ciarlet, The Finite Element Method for Elliptic Problems, SIAM, Philadelphia, 2002.
[6] M. Čermák, S. Sysala, J. Valdman, Efficient and flexible MATLAB implementation of 2d and 3d elastoplastic problems, Appl Math Comput 2019 (355) (2019) 595–614.
[7] P. Drábek, J. Milota, Methods of Nonlinear Analysis: Applications to Differential Equations, (second edition), Birkhauser, 2013.
[8] M. Frost, B. Benešová, P. Sedlák, A microscopically motivated constitutive model for shape memory alloys: formulation, analysis and computations, Math. Mech. Solids 21 (3) (2016) 358–382.
[9] J. Koko, Fast MATLAB assembly of FEM matrices in 2D and 3D using cell-array approach, International Journal of Modeling, Simulation, and Scientific Computing 7 (3) (2016).
[10] M. Kružík, T. Roubíček, Mathematical Methods in Continuum Mechanics of Solids, Springer, 2019.
[11] K.A. Lie, An introduction to reservoir simulation using MATLAB: user guide for the Matlab reservoir simulation toolbox (MRST), Technical report, SINTEF ICT, December, 2016. http://www.sintef.no/projectweb/mrst/
[12] J.E. Marsden, T.J.R. Hughes, Mathematical foundations of elasticity, Dover Publications, 1994.
[13] C. Matonoha, A. Moskovka, J. Valdman, Minimization of P-Laplacian via the finite element method in MATLAB, in: I. Lirkov, S. Margenov (Eds.), LSSC 2021, LNCS 13127, 2022, pp. 496–503.
[14] MATLAB, documentation to minimization with gradient and hessian sparsity pattern, https://www.mathworks.com/help/optim/ug/minimization-with-gradient-and-hessian-sparsity-pattern.html.
[15] T. Rahman, J. Valdman, Fast MATLAB assembly of FEM matrices in 2D and 3D: nodal elements, Appl Math Comput 2013 (7151) (2013) 7151–7158.
[16] G.K. Rose, Computational methods for nonlinear systems analysis with applications in mathematics and Engineering, Doctor of Philosophy (PhD), Dissertation, Mechanical & Aerospace Engineering, Old Dominion University, 2017.
[17] T. Weinberg, B. Sousedík, Fast implementation of mixed RT0 finite elements in MATLAB, SIAM Undergraduate Research Online 12 (2018).