Engineering with Computers
https://doi.org/10.1007/s00366-023-01877-w

ORIGINAL ARTICLE

Sonics: develop intuition on biomechanical systems through interactive error controlled simulations

Arnaud Mazier[1] · Sidaty El Hadramy[2] · Jean-Nicolas Brunet[2] · Jack S. Hale[1] · Stéphane Cotin[2] · Stéphane P. A. Bordas[1]

Received: 15 September 2022 / Accepted: 12 July 2023

Abstract
We describe the SOniCS (SOFA + FEniCS) plugin to help develop an intuitive understanding of complex biomechanics systems. This new approach allows the user to experiment with model choices easily and quickly without requiring in-depth expertise. Constitutive models can be modified by one line of code only. This ease in building new models makes SOniCS ideal to develop surrogate, reduced order models and to train machine-learning algorithms for enabling real-time patient-specific simulations. SOniCS is thus not only a tool that facilitates the development of surgical training simulations but also, and perhaps more importantly, paves the way to increase the intuition of users or otherwise non-intuitive behaviors of (bio)mechanical systems. The plugin uses new developments of the FEniCSx project enabling automatic generation with FFCx of finite-element tensors, such as the local residual vector and Jacobian matrix. We verify our approach with numerical simulations, such as manufactured solutions, cantilever beams, and benchmarks provided by FEBio. We reach machine precision accuracy and demonstrate the use of the plugin for a real-time haptic simulation involving a surgical tool controlled by the user in contact with a hyperelastic liver. We include complete examples showing the use of our plugin for simulations involving Saint Venant–Kirchhoff, Neo-Hookean, Mooney–Rivlin, and Holzapfel Ogden anisotropic models as supplementary material.

Keywords SOFA · FEniCS · Automatic differentiation · Biomechanics · Hyperelasticity · Benchmark

1 Introduction

Designing efficient finite-element (FE) simulation software is a challenging task. Indeed, as FEM is a vast field, numerous pieces of software emerged to fill different gaps. For instance, commercial software, such as Abaqus [2] or Ansys [3], focuses on user-friendly Graphical User Interfaces (GUIs) guiding the user from pre-processing to post-processing. This has the advantage of allowing users to perform complex simulations with a relatively basic theoretical knowledge of FE. Meanwhile, other pieces of software focused on specific domains, such as Gmsh [4] for FE meshing, Paraview [5] for FE visualization, OpenFoam [6] for CFD (Computational Fluid Dynamic) simulations, OpenXFEM [7] for extended finite elements [8], collocation methods [9], meshfree methods [10], multi-scale problems [11], or material point methods (MPM) [12]. This historical perspective explains the countless FE solvers which makes an exhaustive state-of-the-art review quasi-impossible. Before selecting one piece of FE software, user

Arnaud Mazier and Sidaty El Hadramy have contributed equally to this work.

✉ Stéphane P. A. Bordas
stephane.bordas@alum.northwestern.edu

Arnaud Mazier
mazier.arnaud@gmail.com

Sidaty El Hadramy
sidaty.el-hadramy@inria.fr

Jean-Nicolas Brunet
jnbrunet2000@gmail.com

Jack S. Hale
jack.hale@uni.lu

Stéphane Cotin
stephane.cotin@inria.fr

[1] Institute of Computational Engineering, Department of Engineering, Université du Luxembourg, 2 Avenue de l'Universite, Esch-sur-Alzette 4365, Luxembourg

[2] MIMESIS team, Inria, 1 Place de l'Hôpital, Strasbourg 67000, France

Published online: 30 September 2023

should consider the benefits and disadvantages associated with each (i.e., meshing, parallel support, solvers, coding language, and visualization). However, generally, simplicity of use and the ability to easily test modeling hypotheses appear like the most important considerations in selecting such a computational tool.

In the medical simulation context, several specific aspects have to be considered.

- The material model complexity: Contrary to engineering materials, such as steel or copper, the mechanical properties of living organs were only recently quantified [13, 14] (early 90s against 1700 for copper) and show immense variability [15]. Various models have been proposed, e.g., anisotropic [16–20], hyperelastic [21–25], viscoelastic [26–29], or poroelastic [30–36] to accurately depict their complex mechanical behaviors. Indeed, predicting the deformations of bio-materials can only be achieved through complex material models (sometimes even multi-scale), which are rarely implemented in commercial software. One major difficulty of this implementation remains the differentiation of highly non-linear equations. Indeed, predicting the deformations of such materials can only be achieved through complex material models, rarely implemented in commercial software. One major difficulty of this implementation remains the linearization of highly non-linear equations. For instance, hyperelasticity equations can be written as a minimization of a tensorial function. In most cases, this minimization is solved using gradient-descent algorithms requiring the first and second derivatives of the functional. Therefore, obtaining such high-order non-linear derivatives is not straightforward and can be prone to manual errors.
- The complexity of the simulation: In addition to the complexity of the material model, the simulation setup itself can be problematic. For instance, the material parameters are patient-specific and require data-driven or inference methods [37, 38]. The simulations can also (partially) involve unknown boundary conditions [39], contact with other organs [40, 41], or surgical tools [42, 43]. The problem can also require multi-physics models [such as Fluid–Structure Interactions (FSI) [44, 45]] or incompressibility [46, 47], where classical displacement-based finite elements are prone to locking.
- The error control and uncertainty of the solution: When dealing with biomechanical simulations, several uncertainties always arise from the material parameters, loads, geometry, or boundary conditions. This is mainly due to the difficulty in estimating the mechanical properties through ex vivo methods or the topology of biological tissues using medical imaging. Similarly, error control and mesh adaptivity are necessary to ensure homogeneous convergence of the solution over the domain and that the mesh is optimal given a quantity of interest [48]. Therefore, quantifying the uncertainty or controlling the error on quantities of interests often requires making a very large number of simulations [49, 50] which is incompatible with surgical timing [51, 52]. Meanwhile, those approaches could be functional in clinical settings using accelerated simulations [53] to build surrogate models or/and machine learning models for faster solutions of those highly non-linear parametric problems.
- The real-time aspect: In addition to the previous point, for clinical environments, the run time of the numerical simulation is crucial. When performing an operation, the surgeon cannot, in general, spend minutes waiting for the model predictions. Eventually, this aspect is also applicable to artificial intelligence. Indeed, to build an efficient machine-learning model, a significant amount of data is necessary. Using numerical simulations to create synthetic data is now standard and directly dependent on the simulation time [54, 55]. Consequently, the run time of the simulation can be considered a principal feature of biomechanics simulations, and indeed, of any non-intuitive non-linear problems subject to significant uncertainties in loading, boundary and initial conditions, and parameters. Nowadays, gaming engines, such as Unity3D [56] or Unreal Engine [57], offer real-time animation where physics-based algorithms can be included [58–60].
- The interaction with the user: In a surgical simulation setting, external variables can impact the simulation during the execution. For example, the exact movement of the surgeon's tool influences the simulation at run time. Thus, the parameters of the simulation must be tuned, "live", to integrate interactions between the user and the simulation [61, 62].
- The visual rendering: Depending on the research field, visualization can play a critical role in understanding the results. For instance, in the computer graphics community, photo-realistic visualization is one of the main objectives [63, 64]. Contrastingly, in the mechanical engineering culture, visualization is a manner of extracting and understanding quantitatively a solution or data set (i.e., stress or displacement fields). For medical simulations, an optimal solution must combine both the accuracy of the results and photo-realistic rendering reflecting the clinical ground truth all within clinical time frames [65].

According to the state-of-the-art, Simulation Open Framework Architecture (SOFA) [66] appears to be a suitable compromise. Indeed, SOFA employs efficient rendering while providing the possibility to interact in real time with the running simulation. One can note that real-time computing is only possible depending on the complexity of the problem. Indeed, using excessive numbers of degrees of freedom (DOFs) or solving a highly non-linear problem cannot result in a real-time simulation (without using model order reduction [67–69] or machine learning [55]). SOFA can also manage complex simulations through an efficient implementation of contact [70–72], for example, or enabling multi-physics coupling. At the time of writing, only a few finite elements are available in SOFA. We remark that in this the manuscript, the finite element refers to the choice of basis functions via the Ciarlet triple (cell topology, polynomial space, and dual basis) plus a specific choice of weak form. A similar issue is observed for material models where numerous implementations only focused on a three-dimensional isotropic behavior. Therefore, coding a new finite element (Ciarler triple + weak form) in SOFA requires advanced C++ skills that may discourage individuals from using the software.

To alleviate the problem of complex material models, FEniCS [73] seems like an appropriate solution. Indeed, FEniCS may not possess all of SOFA's features, but definitely overcomes SOFA's capabilities for material model complexity. With FEniCS, the user can generate any material model, regardless of the element's geometry or interpolation. Plus, it authorizes an export of the pertinent finite-element tensors in C code to be efficiently plugged into SOFA. The benefits of the synergy are considerable. Using SOFA's interactivity and real-time features, the user can easily prototype a real-time simulation. Indeed, by modifying "live" various boundary conditions, geometries, or topologies, the user can effortlessly and rapidly verify modeling hypotheses for a specific problem. Combined with the specificities of FEniCS, the user can additionally smoothly prototype complex material models for modeling elaborate scenarios. Such feature has already been used by coupling FEniCS and Acegen but with different objectives [74]. To the authors' knowledge, this paper is the first to use FEniCS code generation capabilities for such an endeavor and is the first coupling between FEniCS and SOFA.

This paper has the following outline. We will first briefly introduce SOFA and FEniCS, highlighting the relative advantages and design choices in each. Section 2 will detail the plugin functionalities and a short tutorial for importing a Saint Venant–Kirchhoff model from FEniCS in SOFA. Then, Sect. 3 will focus on confirming our implementation for various numerical tests. We will use a manufactured solution in Sect. 3.1 as validation and compare our solutions for a cantilever beam problem with SOFA in Sect. 3.2. The last test consists in implementing a new material model (Mooney–Rivlin) in SOFA and benchmarking it with FEBio [75] in Sect. 3.3. Finally, in Sect. 4, we will use our plugin in a complex haptic simulation that cannot be implemented in FEniCS, using a custom material model inexistent in SOFA.

1.1 SOFA

SOFA was created in 2007 by a joint effort from Inria, CNRS, USTL, UJF, and MGH. This piece of software aims to provide an efficient framework dedicated to research, prototyping, and the development of physics-based simulations. It is an open-source library distributed under the LGPL license, hosted on GitHub at https://github.com/sofa-framework/sofa, and developed by an international community. SOFA is modular. Users can create public or private plugins to include additional features.

SOFA is a C++ library, including Python wrappers for a user-friendly prototyping interface. It was originally designed for deformable solid mechanics but has been extended to various domains, such as robotics, registration, fluid simulations, model-order reduction, and haptic simulations [76–78]. SOFA exhibits many attractive features, but among them, the combination of multi-model representations and mappings differentiates it from other software.

Multi-model representation: Most classical FE software uses an identical discretization for the whole model. Consequently, if one user wants to refine the mesh along a contact surface, the FE mesh will undergo the same refinement in the contact region. It can induce slow simulations for solving the FE system, while the user was initially only interested in the contact part. Conversely, utilizing a multi-model representation approach, users can split the principal model into three distinct sub-models: deformation, collision, and visual. Thus, the user can decide to have high-fidelity deformations with flawed contact detection while maintaining a fine rendering, or vice versa. Similarly, an object can be made of several deformation models. For example, one can model a muscle by the interaction of FE 3D tetrahedra for the volume and 1D beams for modeling ligaments, using two different solvers.

Mappings: In SOFA, the "mappings" are responsible for the communication between the different models. The models have parent–child relationships constructing a hierarchy

(and a DOF hierarchy by extension). It enables propagating the positions, velocities, accelerations, and forces across the different models. For example, if the contact model calculates a force, it is mapped on the deformation model that will communicate back the computed displacement.

Finally, by combining the multi-model representation with the mappings, SOFA can build complex real-time simulations with high-fidelity rendering. In addition to a scenegraph structure and visitors (responsible for going through the model hierarchy) implementation, it can account for interactivity with the users.

Despite the advantages provided by SOFA, some drawbacks have to be acknowledged. First, in terms of solid mechanics simulations, only a few elements and material models are available. Indeed, SOFA only proposes Lagrange linear elements, and the cell topologies are limited to segments, triangles, quadrangles, tetrahedra, and hexahedra. The following material models are coded: Boyce and Aruda [79], Costa [80], isotropic and anisotropic Hookean, Mooney–Rivlin (two invariants) [81], classical and stabilized Neo-Hookean, Ogden [82], Saint Venant–Kirchhoff, and Veronda Westman [83]. Despite a reasonable number of mechanical models, a few of them are actually implemented for each cell topology. Second, the benefits provided by the mappings can also turn out to be a disadvantage when it comes to implementation. Indeed, the structure of the mappings is usually complex for unexperimented C++ users, and the mechanical tensors such as the Cauchy–Green or Piola–Kirchhoff are rarely computed. The two previous drawbacks are associated with the same flaw: the strong coupling between the material models and the topology of the element assumed within SOFA's architecture. This coupling implies that changing an element's topology or interpolation will involve a new mapping or the rewriting of the material model, even in the case of a similar material model.

1.2 The modular mechanics plugin (Caribou)

The initial goal of the plugin (called Caribou at the time of writing, https://github.com/mimesis-inria/caribou) was to quickly implement new shape functions and their derivatives for different Immersed-Boundary and meshless domain discretization while keeping the compatibility with the existing SOFA surgical simulations [84]. Besides, the plugin enabled to effortlessly implement different volumetric quadrature schemes and several hyperelastic material models. Hence, the software design had to be generic enough to combine all the previous requirements. It also had to be efficient enough to avoid the creation of a bottleneck that would prevent the biomechanical model from meeting its computational speed requirement. Hence, the plugin was made as an extension to SOFA, bringing a redesigned software architecture.

In the plugin, the authors implemented a compile-time polymorphism design using generic C++ template programming. The idea is to write the code as close as possible to equations found in the traditional FE books. Then, the C++ compiler optimizes the set of operations executed during the simulation while keeping an object-oriented code. In this design, the "Element" concept was created as a generic computational class that would be inherited by all element types. Similarly to OpenXFEM++ [7], it provides a flexible implementation to add interpolation and quadrature numerical procedures quickly. Since standard isoparametric elements have a number of nodes, quadrature points, and shape functions already known at compile-time, most modern compilers will be able to aggressively inline the code to optimize the computation.

Finally, the plugin allows the creation of additional material models by simply defining three methods per material: the strain energy density function, the second Piola–Kirchhoff stress tensor function, and its derivative functions. These three functions are evaluated at a given integration point automatically provided by the plugin. This design delivers an undeniable advantage: writing a new material model is now independent of the topology and integration scheme. However, it comes with a non-negligible cost. The author of the new material model has to manually differentiate the strain energy twice and write it in C++. This manual intervention is error-prone and can quickly become a substantial drawback for complex materials.

1.3 FEniCS

The FEniCS Project (FEniCS) [73] is a collection of tools for the automated solution of partial differential equations using the finite-element method. Like SOFA, the FEniCS components are distributed under open-source licenses (LGPL v3 or later, and MIT) and development is hosted on GitHub at https://github.com/fenics.

A distinguishing feature of FEniCS is the ability to allow the user to write variational of weak formulations of finite-element methods in a high-level Python-based domain-specific language (DSL), the Unified Form Language (UFL) [85]. Subsequently, that high-level description can be compiled/transformed using the FEniCS Form Compiler (FFC) [86] into low-level and high-performance kernels. These kernels can calculate the corresponding local finite-element

tensor for a given cell in the mesh. UFL is also used by other finite-element solvers with independently developed automatic code generation capabilities, notably Firedrake [87], Dune [88]. The AceGen software also supports automatic differentiation and support for automaticaly generating low-level code [89]. Compared with SOFA, FEniCS is limited in scope; its primary focus is the specification and solution of partial differential equations via the finite-element method, leaving difficult problems like mesh generation, post-processing, and visualization to leading third-party packages, such as Gmsh [4] and Paraview [5].

In the context of implementing finite-element models of hyperelastic materials, this automatic approach has a number of advantages over the traditional route used by most finite-element codes (including, to some extent, SOFA); differentiating analytical expressions for the residual (first derivative) and Jacobian (second derivative) of the energy functional for the hyperelastic model, picking a suitable finite-element basis, and then hand-coding the corresponding finite-element kernels in a low-level language (C, Fortran, and C++) for performance. Specifically:

1. The symbolic residual and Jacobian can be derived automatically using the symbolic differentiation capabilities of UFL. By contrast, taking these derivatives by hand can be tedious and error-prone.
2. The compilation of the UFL description of the problem by FFC into the associated low-level kernels is entirely automated. Again, this step is often time-consuming and difficult to perform manually.
3. Because of the high-level description of the problem, it is possible to experiment quickly with different concrete finite-element formulations (material models, basis functions, element topology, etc.) without manually modifying low-level kernel code.

The potential of this high-level approach for solid mechanics was recognized early on in the development of FEniCS, with two chapters in the FEniCS Book [90, 91] promoting this direction. Since then, FEniCS has been used in a large number of publications on the topic of hyperelastic large-deformation elasticity, e.g., [92–96].

Recently, the FEniCS Project has undergone a major redevelopment, resulting in the new FEniCSx components; DOLFINx (the finite-element problem solving environment, replacing DOLFIN), FFCx (the FEniCSx Form Compiler, replacing FFC), and Basix [97] (a finite element basis function tabulator, replacing FIAT [98]). UFL is largely unchanged from the version used in the old FEniCS components and Firedrake.

In this work, we do not use DOLFINx. DOLFINx contains the basic finite-element data structures and algorithms (e.g., meshes, function spaces, assembly, interfacing with linear algebra data structures in, e.g., PETSc [99]), and therefore, there is a significant overlap with the functionality already available in SOFA. Directly interfacing DOLFINx and SOFA at the Application Programming Interface (API) level would be a significant technical challenge due to the substantial differences in their internal data structures. Dedicated weak coupling libraries such as PreCICE [100] could be an interesting alternative to API coupling, but it is not a path that we explore in this work.

Instead, the approach taken by SOniCS is to only use UFL and FFCx (which in turn depends on Basix) to convert the high-level description of the finite-element problem into low-level C code, which are then called using SOFA's existing C++ finite-element data structures and algorithms. Compared with coupling DOLFINx and SOFA directly, our approach creates a relatively light compile–time coupling between FEniCS (specifically, the generated C code) and SOFA (a large complex C++ code with many dependencies). Consequently, no additional runtime dependencies required for SOFA. This methodology will be familiar to users of the DOLFINx C++ interface where C finite-element kernels are generated in a first step using UFL and FFCx and are then integrated into the DOLFINx solver in a second step through a standard compile/include/link approach. Without going into excessive detail, three changes in the redeveloped FEniCSx components have made SOniCS significantly easier to realize:

1. FFCx (and also DOLFINx) now support code generations using finite elements defined on quadrilateral and hexahedral cells, the default in SOFA.
2. Basix and FFCx have full support for Serendipity finite elements of arbitrary polynomial order following the construction of Arnold and Awanou [101]. Serendipity elements are used in SOFA and there was a desire to continue supporting Serendipity basis function due to their lower number of degrees of freedom per cell and generally lower number of local computations compared with standard tensor-product Lagrange elements. We remark that despite the widespread use of Serendipity elements in many solvers, they can only obtain optimal order convergence on affinely mapped meshes; see, e.g., [102] for more details.
3. FFCx outputs C99 compliant code according to the UFCx interface, which is specified as a C header file included with FFCx. This is in contrast with FFC, which outputs C++03 compliant code conforming to an

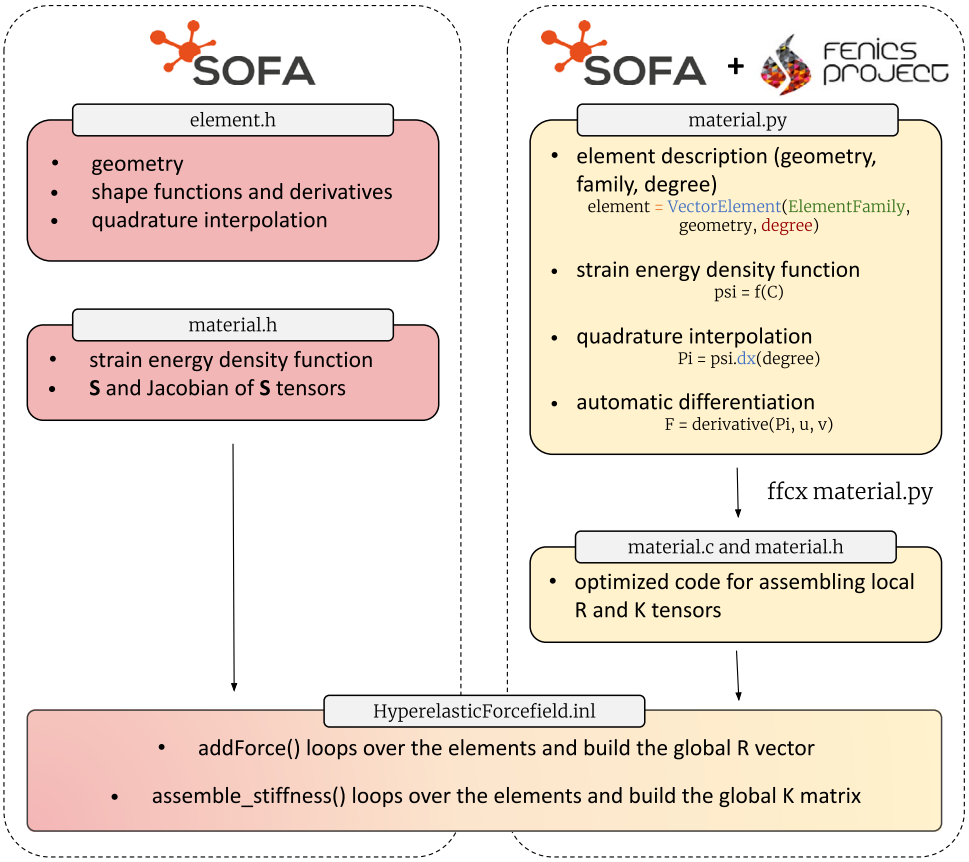


Fig. 1 Description of the SOniCS pipeline (on the right) and differences with SOFA (on the left). In SOFA, each element has to be defined, embedding cell geometry, shape functions (including derivatives), and the quadrature scheme and degree. In SOniCS, it has been replaced by two Python lines of code for describing the element and its quadrature. The same benefit goes for the material model description. In SOFA, each material has to be created in a separate file stating its strain energy, derivating by hand the second Piola–Kirchhoff tensor (S) and its Jacobian. It was replaced in SOniCS by only defining the strain energy of the desired material model in UFL. The derivative of the strain energy will then be automatically calculated using the FFCX module. Finally, both plugins share the same Forcefield methods for assembling the global residual vector ($\Omega \subset R^3$) and stiffness matrix ($\boldsymbol{u} : \Omega \rightarrow R^3$)

interface specified with a C++ header file. This switch makes it significantly easier to call FFCx generated kernels from libraries with a C Foreign Function Interface (FFI) such as Python and Julia, or any language which can easily call functions with a C ABI (e.g., Fortran). Although SOFA is a C++ libraries and could certainly call C++ generated kernels, the C interface is simpler to use, consisting only of structs containing basic native data types and functions.

2 SOniCS

In this section, we present more in-depth the SOniCS (SOFA + FEniCS). The plugin is available on a development branch in GitHub https://github.com/mimesis-inria/caribou/tree/FeniCS-features and will soon be merged into the main branch. We first introduce the procedure for defining the material model, the element, and the quadrature rule or degree using the UFL (Python) syntax. For simplicity, we only focused on a Saint Venant–Kirchhoff model. However, the method can be generalized to all element types following the pipeline shown in Fig. 1. Second, we explain the methodology for converting the UFL script into efficient C kernels. Finally, we show the interface between the SOniCS plugin and SOFA, stating the conceptual and coding differences. For simplicity and as the modular mechanics plugin's name (Caribou) might change, we use the name SOFA to denote the combination of SOFA and the modular mechanics plugin.

2.1 UFL: from FE model to Python code

The first step is to define the FE model using the UFL syntax in a Python script. As an example, in listing 1, we describe the 3D simplest hyperelastic model and element: Saint Venant–Kirchhoff with linear Lagrange finite elements on tetrahedra. In the context of hyperelastic simulations, we will exclusively describe features that users could be interested in customizing.

```python
# material.py
from ufl import (Coefficient, Constant, Identity,
                 TestFunction, TrialFunction, inner, ds,
                 VectorElement, derivative, dx, grad,
                 tetrahedron, tr, variable)

# Function spaces
cell = tetrahedron
d = cell.geometric_dimension()
element = VectorElement("Lagrange", cell, 1)

# Trial and test functions
du = TrialFunction(element)  # Incremental displacement
v = TestFunction(element)  # Test function

# Functions
u = Coefficient(element)  # Displacement from previous iteration
B = Coefficient(element)   # Body forces
B = Coefficient(element)   # Traction forces

# Kinematics
I = Identity(d)  # Identity tensor
F = variable(I + grad(u))  # Deformation gradient
C = variable(F.T * F)  # Right Cauchy-Green tensor
E = variable(0.5 * (C - I))  # Green-Lagrange tensor

# Elasticity parameters
young = Constant(cell)
poisson = Constant(cell)
mu = young / (2 * (1 + poisson))
lmbda = young * poisson / ((1 + poisson) * (1 - 2 * poisson))

# Stored strain energy density (compressible Neo-Hookean model)
psi = (lmbda / 2) * tr(E) ** 2 + mu * tr(E * E)

# Total potential energy
Pi = psi * dx(degree=1) - inner(B, u) * dx(degree=1) - inner(T, u) * ds(
    degree=1)

# First variation of Pi (directional derivative about u in the direction of
    v)
F = derivative(Pi, u, v)

# Compute Jacobian of F
J = derivative(F, u, du)

# Export forms
forms = [F, J, Pi]
```

Listing 1: Python code example (`material.py`) of a Saint Venant-Kirchhoff material model using Lagrange linear tetrahedron.

Listing 1: Python code example (material.py) of a Saint Venant-Kirchhoffmaterial model using Lagrange linear tetrahedron.

Element: After importing the necessary packages, we can define the element geometry. In listing 1, on line 10, we used a linear Lagrange tetrahedron. The user can easily modify different parameters of the element, such as the geometry, the family type, or the interpolation degree. For example, by only changing line 10 to

```
element = VectorElement("Serendipity", hexahedron, 2)
```

the element is now a quadratic Serendipity hexahedron. Note that `VectorElement` creates by default a function space of vector field equal to the spatial dimension. A complete list of the element available in the Basix documentation [97].

Material model: By definition, boundary value problems for hyperelastic media can be expressed as minimization problems. For a domain $\Omega \subset R^3$, the goal is to find the displacement field $\boldsymbol{u} : \Omega \rightarrow R^3$ that minimizes the total potential energy Π. The potential energy is given by

$$\Pi(\boldsymbol{u}) = \int_{\Omega} \psi(\boldsymbol{u})\,\mathrm{d}x - \int_{\Omega} \boldsymbol{B}\cdot\boldsymbol{u}\,\mathrm{d}x - \int_{\partial\Omega} \boldsymbol{T}\cdot\boldsymbol{u}\,\mathrm{d}s, \tag{1}$$

where ψ is the elastic stored energy density, $\boldsymbol{B}$ is a body force (per unit reference volume), and $\boldsymbol{T}$ is a traction force (per unit reference area) prescribed on a boundary $\partial\Omega$ (of measure $\mathrm{d}s$) of the domain Ω (of measure $\mathrm{d}x$).

In listing 1, at line 37, we define the strain energy of a Saint Venant–Kirchhoff material model as

$$\psi = \frac{\lambda}{2}\mathrm{tr}(\boldsymbol{E})^2 + \mu\mathrm{tr}(\boldsymbol{E}^2), \tag{2}$$

where λ and μ are Lamé material constants, while $\boldsymbol{E}$ is the Green Lagrange strain tensor defined as $\frac{1}{2}(\boldsymbol{C}-\boldsymbol{I})$, where $\boldsymbol{I}$ is the identity matrix and $\boldsymbol{C}$ the right Cauchy–Green tensor $\boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F} = (\nabla\boldsymbol{u}+\boldsymbol{I})^T(\nabla\boldsymbol{u}+\boldsymbol{I})$. Therefore, we observe that tensor $\boldsymbol{E}$ is expressed with respect to the displacement $\boldsymbol{u}$, which is the unknown displacement field. Moreover, λ and μ are a function of the Young's modulus and of Poisson's ratio that are assumed to be known constants.

If the user would like to change the material model for Neo-Hookean with the strain energy density

$$\psi = \frac{\mu}{2}(I_C - 3) - \mu\ln(J) + \frac{\lambda}{2}\ln(J)^2. \tag{3}$$

It would only require in replacing line 31 with: `psi = (mu / 2) * (Ic - 3) - mu * ln(J) + (lmbda / 2) * (ln(J)) ** 2` in addition to previously defining the corresponding kinematics variables $J = \det(\boldsymbol{F})$ and $I_C = \mathrm{tr}(\boldsymbol{C})$.

```
J = det(F)
I_C = tr(C)
```

Quadrature rule: In listing 1, at line 37, when calculating the total potential energy, it is also possible to choose the

quadrature rule and the degree. In our example, we selected a quadrature degree of 1, triggering by default the Zienkiewicz and Taylor scheme [103] for tetrahedra. Hence, the user could also choose to use a Gauss–Jacobi quadrature of degree 2 [104] by replacing line 37 with:

```
Pi = psi * dx(degree=2, scheme="Gauss-Jacobi")  - inner(B, u) * dx(degree
    =2, scheme="Gauss-Jacobi") - inner(T, u) * ds(degree=2, scheme="Gauss-
    Jacobi")
```

.

2.2 FFCx: from Python code to efficient C kernels

In the particular case of static hyperelastic simulations, we solve the following non-linear system of equation:

$$\boldsymbol{K}(\boldsymbol{u}) \cdot \mathrm{d}\boldsymbol{u} = \boldsymbol{R}(\boldsymbol{u}) - f(\boldsymbol{u}). \tag{4}$$

The $\boldsymbol{R}$ tensor is called the residual vector and is defined as the Gâteaux derivative of the total potential energy Π with respect to change in the displacement $\boldsymbol{u}$ in direction $\boldsymbol{v}$

$$\boldsymbol{R} = \frac{\mathrm{d}\Pi(\boldsymbol{u} + \epsilon \boldsymbol{v})}{\mathrm{d}\epsilon}|_{\epsilon=0}. \tag{5}$$

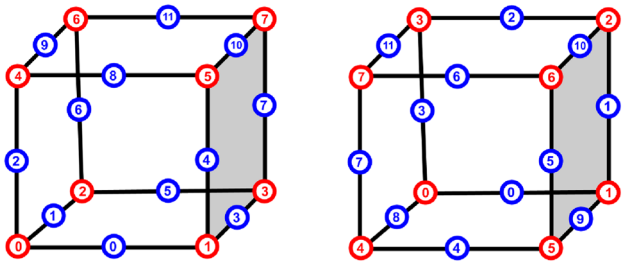
The tensor $\boldsymbol{K}$ is the Jacobian (also called stiffness in the context of mechanics) matrix and corresponds to the derivative of $\boldsymbol{R}$

$$\boldsymbol{K} = \frac{\mathrm{d}\boldsymbol{R}(\boldsymbol{u} + \epsilon \mathrm{d}\boldsymbol{u})}{\mathrm{d}\epsilon}|_{\epsilon=0}. \tag{6}$$

Solving the non-linear system in Eq. 4 can be achieved using the Newton–Raphson algorithm that will iteratively solve a set of linear systems, assuming an initial guess $\boldsymbol{u}_n$

$$\boldsymbol{u}_{n+1} = \boldsymbol{u}_n - d\boldsymbol{u}. \tag{7}$$

The two tensors can be derived symbolically and exported using the UFL syntax with the function `derivative`, as shown at lines 40 and 43 in listing 1. A simple call to `ffcx` will create a.c and.h files containing the code for generating the local $\boldsymbol{R}$ and $\boldsymbol{K}$ tensors.



(a) FFCx/Basix. (b) SOFA.

Fig. 2 Local numbering of element vertices and edges in both FEniCS and SOFA

```
$ ffcx material.py
```

2.3 Integration in SOniCS

In SOFA, the definitions of the residual and stiffness tensors are carried out within a C++ file, `HyperelasticForcefield.cpp`. Each material model is in a separate file. So far, only Saint Venant–Kirchhoff and Neo-Hookean models have been implemented. The users can easily access those functionalities through Python wrappers:

```
node.addObject("SaintVenantKirchhoffMaterial", young_modulus=E,
 poisson_ratio=nu)
node.addObject('HyperelasticForcefield')
```

The `HyperelasticForcefield.cpp` contains several functions, but we are particularly focusing on two of them. The `addForce` and `assemble_stiffness` functions are assembling the global residual and stiffness tensors, respectively. Algorithm 1 details the `addForce` function, while algorithm 2 presents our reimplementation of the procedure. We did not detail the `assemble_stiffness` as it involves the exact same differences between the two implementations.

Algorithm 1 SOFA addForce function. The addForce function is in charge of assembling the global residual vector.

for `element in elements` **do**
  $X \leftarrow$ `element.positions` ▷ return the current positions of the element
  $R_{\text{global}} \leftarrow 0$ ▷ zero the global residual vector of dimension (DOFs ×3)
  $R_{\text{local}} \leftarrow 0$ ▷ zero the local residual vector of dimension (element DOFs ×3)
  for `quadrature in quadratures` **do**
    $\det J \leftarrow \det($`quadrature.nodes`$)$ ▷ return the Jacobian of the quadrature nodes
    $\mathrm{d}N \leftarrow$ `quadrature.nodes.shape_functions_derivatives` ▷ return the derivatives of the shape functions of the quadrature nodes
    $w \leftarrow$ `quadrature.nodes.weights` ▷ return the weights of the quadrature nodes
    $F \leftarrow X^T \cdot \mathrm{d}N$
    $J \leftarrow \det(F)$
    $C \leftarrow F^T \cdot F$
    $S \leftarrow \mathrm{f}(C,$ `MaterialParameters`$)$ ▷ return the second Piola-Kirchhoff depending on the material parameters and kinematics tensors
    for `i in range(0, NumberOfNodesPerElement)` **do**
      $\mathrm{d}x \leftarrow \mathrm{d}N[\mathrm{i}]^T$
      $R_{local}[i] \leftarrow (\det J \cdot w) \cdot F \cdot S \cdot \mathrm{d}x$ ▷ allocate the result in the local residual vector
    end for
  end for
  for `i in range(0, NumberOfNodesPerElement)` **do**
    $R_{\text{global}}[$`global`$(i)] \leftarrow R_{\text{global}}[$`global`$(i)] - R_{\text{local}}[i]$ ▷ i indicates the element node index while `global`(i) denotes the global node index
  end for
end for

Listing 2: Python definition of a hyperealstic forcefield in SOFA using a Saint Venant-Kirchhoff material model.

The structure is similar, but we can still observe a few differences.

- The new implementation of algorithm 2 is more concise and involves less visible tensorial operations, because all

those operations are efficiently hard-coded in the C file provided by FFCx. For example, in algorithm 1, lines 5 to 17 were replaced in the new algorithm 2 by solely line 11.

- Algorithm 2 needs to have access to the initial position of the object and to the displacement vector. This was indeed not needed in the previous implementation, since the modular mechanics plugin takes advantage of writing the deformation gradient only based on the current nodal coordinates $\boldsymbol{x}$

$$\boldsymbol{F} = \boldsymbol{I} + \nabla_{\Omega_0}\boldsymbol{u} = \boldsymbol{I} + \nabla_{\Omega_0}(\boldsymbol{x} - \boldsymbol{x_0}) = \nabla_{\Omega_0}\boldsymbol{x}. \tag{8}$$

∇_{Ω_0} and $\boldsymbol{x_0}$, respectively, denote the gradient and the nodal coordinates in the initial configuration, thus saving one extra vector operation.

Algorithm 2 SOniCS addForce function. The addForce function is in charge of assembling the global residual vector.

1: **for** `element in elements` **do**
2: $X \leftarrow$ `element.positions` ▷ return the current positions of the element
3: $X_0 \leftarrow$ `element.rest_positions` ▷ return the initial positions of the element
4: $B \leftarrow$ `gravity` ▷ return the body forces
5: $T \leftarrow$ `element.forces` ▷ return the traction forces applied on the element
6: $u \leftarrow X - X_0$
7: $R_{\texttt{global}} \leftarrow 0$ ▷ zero the global residual vector of dimension (DOFs ×3)
8: $R_{\texttt{local}} \leftarrow 0$ ▷ zero the local boundary conditions residual vector of dimension (element DOFs ×3)
9: $R^{\texttt{bc}}_{\texttt{local}} \leftarrow 0$ ▷ zero the local residual vector of dimension (element DOFs ×3)
10: `constants` $\leftarrow$ `MaterialParameters` ▷ return the material parameters
11: $R_{\texttt{local}} \leftarrow$ `tabulate_tensor`$(R_{\texttt{local}}, u, B,$ `constants`$, X_0)$
12: $R^{\texttt{bc}}_{\texttt{local}} \leftarrow$ `tabulate_tensor`$(R_{\texttt{local}}, u, T,$ `constants`$, X_0)$
13: **for** `i in range(0, NumberOfNodesPerElement)` **do**
14: $R_{\texttt{global}}[\texttt{global}(i)] \leftarrow R_{\texttt{global}}[\texttt{global}(i)] - (R_{\texttt{local}}[i] + R^{\texttt{bc}}_{\texttt{local}}[i])$ ▷ i indicates the element node index while `global`(i) denotes the global node index
15: **end for**
16: **end for**

- In the SOFA implementation, the boundary conditions and body forces are treated in separate files. In the new implementation of the Forcefield 1, the boundary conditions and body forces are now directly carried out in the Forcefield on lines 11 and 12. It avoids calling another function to loop again through every element of the object, thus speeding up the assembly of the residual vector.

Based on this new implementation, we created a new forcefield `HyperelasticForcefield_FEniCS` as close as possible to the existing syntax of `HyperelasticForcefield`. We also needed to tune the existing material definition to replace unnecessary calculations and allow us to read the corresponding.c file.

```
node.addObject("FEniCS_Material", material="SaintVenantKirchhoff",
 young_modulus=E, poisson_ratio=nu)
node.addObject('HyperelasticForcefield_FEniCS'
```

Listing 3: Python definition of a hyperelastic forcefield in SOniCS using a Saint Venant-Kirchhoff material model.

Listing 3: Python definition of a hyperelastic forcefield in SOniCS using a SaintVenant-Kirchhoff material model.

Finally, the last hurdle was the element definitions. Indeed, SOFA and FEniCS do not use the same vertices, edges, and facets ordering (as shown in Fig. 2). To avoid any conflict with the existing users of SOFA and solve the ordering issue, we proposed rearranging the topology indices, edges, and vertices and creating new elements. It ensured an accurate integration over the elements (especially for quadratic Serendipity integrating over the edges) and preserved an appropriate visualization. Those elements have been interfaced with the existing topology named `CaribouTopology`.

```
node.addObject('CaribouTopology', name='topology', template="Hexahedron
", indices=mesh.cells_dict['hexahedron'])

node.addObject('CaribouTopology', name='topology', template="
Hexahedron_FEniCS", indices=mesh.cells_dict['hexahedron'][:, [4, 5, 0,
1, 7, 6, 3, 2]])
```

Listing 4: Python definition of hexahedron topology in SOFA and SOniCS.

Listing 4: Python definition of hexahedron topology in SOFA and SOniCS.

3 Numerical examples

In this section, we describe three numerical examples used for the validation of our SOniCS implementation. Every simulation described in this section can be reproduced and is available on our GitHub page [1] (https://github.com/Ziemnono/SOniCS_validation). We use the same domain description for each example while varying the boundary conditions and material parameters of each simulation.

Let Ω be a domain represented by a squared-section beam of dimensions $80 \times 15 \times 15\ \text{m}^3$, considered fixed on the right side ($\boldsymbol{u} = 0$ on Γ_D), while Neumann boundary conditions are applied on the left side (Γ_N), as shown in Fig. 3a and 3b.

Ω was discretized using two different geometrical elements using linear and quadratic interpolations. P1 and P2 elements stand for linear or quadratic tetrahedra, while Q1 and Q2 denote linear and quadratic hexahedra.

To solve each hyperelastic formulation described in Eq. 4, we used an identical implementation of the classical Newton

Raphson (NR) solver for SOniCS, SOFA, and FEBio. The solver had the following parameters: a maximum of 25 iterations with a residual and displacement tolerance of 10^{-10}. To compare the running time of the two implementations, we, therefore, introduced the mean NR iteration time. We defined the NR iteration time as the duration for assembling and factorizing the system matrix, solving and propagating the unknown increment, updating and computing the force and displacement residual. After checking that the same number of iterations have been achieved, we averaged the total time over the number of iterations needed for the solver convergence. All calculations were performed using an Intel® Core™ i5-6300HQ CPU @ 2.30GHz × 4 processor with a 16GiB memory and a NV117 / Mesa Intel® HD Graphics 530 (SKL GT2) graphics card.

We evaluated the soundness of the SOniCS solution using SOFA, FEBio, or a manufactured solution as the reference solution and computed the Euclidean relative L^2 error for the displacement and strain fields [54]

$$L^2_u(\boldsymbol{u},\boldsymbol{v}) = \frac{||\boldsymbol{u}-\boldsymbol{v}||_2}{||\boldsymbol{v}||_2}, \tag{9}$$

$$L^2_E(\boldsymbol{u},\boldsymbol{v}) = \frac{||\boldsymbol{E_u}-\boldsymbol{E_v}||_2}{||\boldsymbol{E_v}||_2}, \tag{10}$$

where $\boldsymbol{u}$ and $\boldsymbol{v}$ are the calculated displacement vectors for SOniCS and the reference implementation, respectively, $||\bullet||$ the Euclidean norm, and $\boldsymbol{E_u}$ the Green–Lagrange strain tensor of the displacement $\boldsymbol{u}$.

In this section, we first compare our solution with an analytical one: the manufactured solution. Then, we consider a clamped cantilever beam subject to Neumann boundary conditions and compare its deformation with the SOFA solution. Finally, using the same cantilever beam, we implemented a Mooney–Rivlin model (uncoded in SOFA) using SOniCS and compared the solution with FEBio.

3.1 Manufactured solution

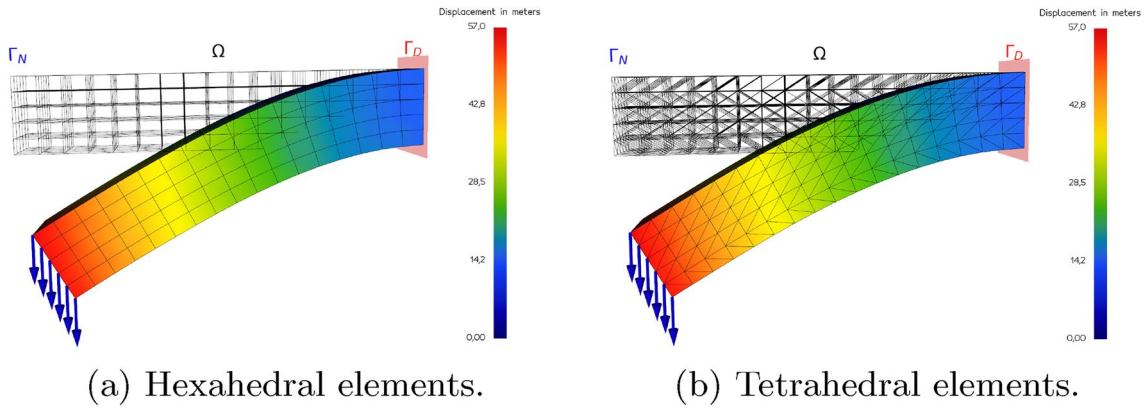
Aiming at code verification, the method of the manufactured solution consists in choosing an exact solution to the problem as an analytical expression [105]. The chosen analytical expression is then inserted into the Partial Differential Equation (PDE) under consideration to find the conditions that lead to this solution. In general, the manufactured chosen solution is expressed in simple primitive functions like sin(), exp(), tanh(), etc. In the context of hyperelastic equations, we considered the following manufactured solution for the displacement:

$$\boldsymbol{u}(x,y,z) = \begin{bmatrix} 10^{-2}\cdot z\cdot e^{x} \\ 10^{-2}\cdot z\cdot e^{y} \\ 10^{-2}\cdot z\cdot e^{z} \end{bmatrix} \text{on } \Omega. \tag{11}$$

Starting from the above-chosen displacement and using continuum mechanics laws, the relative analytical forces are applied as Neumann boundary conditions and deduced as follows:

$$\boldsymbol{F} = \boldsymbol{I_d} + \text{grad}(\boldsymbol{u}), \tag{12}$$

$$\boldsymbol{P} = \frac{\partial W}{\partial \boldsymbol{F}}, \tag{13}$$



(a) Hexahedral elements. (b) Tetrahedral elements.

Fig. 3 Cantilever beam domain discretization and displacement field. Π is a domain represented by a squared-section beam of dimensions $\Pi(\boldsymbol{u}) = \int_\Omega \psi(\boldsymbol{u})\,\mathrm{d}x - \int_\Omega \boldsymbol{B}\cdot\boldsymbol{u}\,\mathrm{d}x - \int_{\partial\Omega} \boldsymbol{T}\cdot\boldsymbol{u}\,\mathrm{d}s,$, considered fixed on the right side (ψ on $\boldsymbol{B}$), while Neumann boundary conditions are applied on the left side ($\boldsymbol{T}$)

$$\boldsymbol{f} = -\boldsymbol{\nabla} \cdot \boldsymbol{P} \, \text{on} \, \Gamma_N, \tag{14}$$

where $\boldsymbol{F}$ is the deformation gradient, $\boldsymbol{I_d}$ is the identity matrix of dimension d, $\boldsymbol{P}$ is the first Piola–Kirchhoff stress tensor, and $\boldsymbol{W}$ is the strain energy density depending on the material model constitutive law. We used a Saint Venant–Kirchhoff material (2) with a Young's Modulus of 3 kPa and a Poisson's ratio of 0.3, and the computation of this solution was performed using Python Sympy package [106].

In this experiment, we generated 7 and 6 discretizations of P1 and P2 elements, respectively, with a decreasing element size in both scenarios. For each discretization, we applied the relative analytical forces deduced from the manufactured solution (11) and used SOniCS Saint Venant–Kirchhoff material implementation with the same parameters as Sympy's to fill the domain. The displacements obtained were compared to the chosen analytical solution in Eq. (11) for each discretization. The results are presented in Fig. 4, and the error metrics are the relative errors presented in Eqs. 9 and 10.

3.2 Benchmark with SOFA

The cantilever beam deflection is a classical mechanical test case, as you can smoothly refine the mesh due to the simplicity of the geometry or modify its boundary conditions to fit real-life experiments. In this context, the beam was still clamped on the right side (natural Dirichlet condition on Γ_D), while Neumann boundary conditions were applied on the left side (Γ_N). To compare our SOniCS implementation, we model the deformation of the beam with two hyperelastic material models: Saint Venant–Kirchhoff and Neo-Hookean. We fixed the mechanical parameters and the Neumann boundary conditions equal to -10 Pa in the y-direction until reaching sufficient large deformations with the same parameters as before: $E = 3$ kPa and $\nu = 0.3$.

This study aims at comparing the finite-element solutions provided by SOniCS and SOFA under the same constraints in terms of computational and running time performances. To do so, we computed the relative L^2 error for the displacement and strain fields ($L^2_u(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$ and $L^2_E(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$)between the SOniCS solution using SOFA as the reference solution.

The results obtained are presented in Tables 1 and 2 for Saint Venant–Kirchhoff and Neo-Hookean materials, respectively.

3.3 Benchmark with FEBio

FEBio is an open-source finite-element package specifically designed for biomechanical applications. It offers modeling scenarios, a wide range of constitutive material models, and boundary conditions relevant to numerous research areas in biomechanics. In this section, FEBio was used to compute the same scenarios as in Sect. 3.2 to evaluate the trustworthiness of SOniCS. A more advanced constitutive material model, Mooney Rivlin, was introduced for this purpose

$$\psi = C_{01}\left(\overline{\mathrm{I}_C} - 3\right) + C_{10}\left(\overline{\mathrm{II}_C} - 3\right) + \frac{K}{2}\ln(J), \tag{15}$$

where C_{01}, C_{10}, and K are the material constants in addition to the modified invariants $\overline{\mathrm{I}_C} = J^{-\frac{2}{3}}\,\mathrm{I}_C$, $\overline{\mathrm{II}_C} = J^{-\frac{4}{3}}\,\mathrm{II}_C$ defined based on the classic invariants $\mathrm{I}_{\boldsymbol{C}} = \mathrm{tr}(\mathbf{C})$, $\mathrm{II}_{\boldsymbol{C}} = \frac{1}{2}\left((\mathrm{tr}(\mathbf{C}))^2 - \mathrm{tr}\left(\mathbf{C}^2\right)\right)$. To obtain sufficiently large deformations, we chose the following material parameters: $C_{01} = 2000$ Pa, $C_{10} = 100$ Pa, and $K = 1000$ Pa.

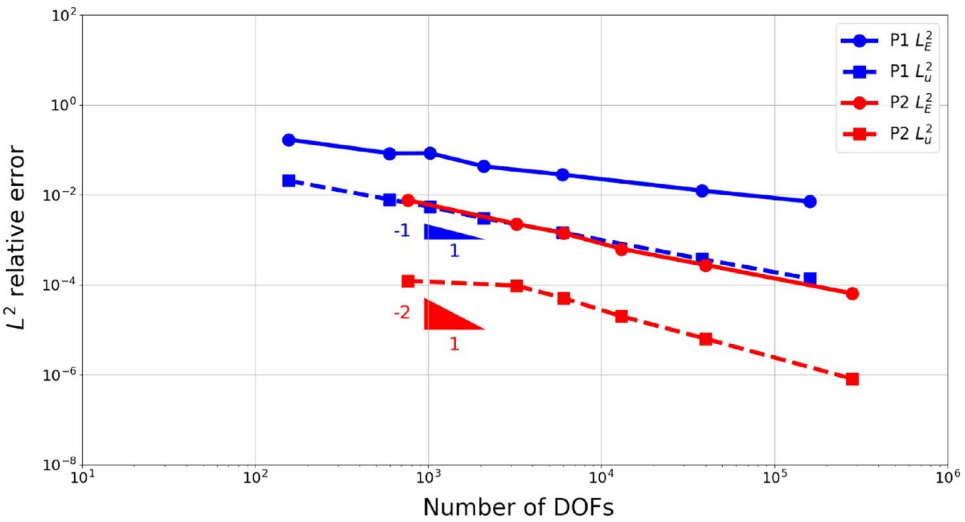


Fig. 4 Plot of the mesh convergence analysis of the manufactured solution. The $\partial\Omega$ errors ($\mathrm{d}s$ for displacement and Ω for strain) between the analytical and the SOniCS simulation are calculated for different number of Degrees of Freedom (DOFs) with fixed parameters ($\mathrm{d}x$ and $\psi = \frac{\lambda}{2}\mathrm{tr}(\boldsymbol{E})^2 + \mu\mathrm{tr}(\boldsymbol{E}^2)$, $=0.3$) for P1 linear tetrahedra (blue) and P2 quadratic tetrahedra (red) elements

Tables 3, 4, 5 show the results obtained for Saint Venant–Kirchhoff, Neo-Hookean, and Mooney–Rivlin material models and considering the four discretizations implemented so far in SOniCS. The error evaluation is still based on the mean relative error defined in Eq. 9 using FEBio as the reference while using a Newton–Raphson solver with the same characteristics in both cases.

4 Holzapfel and Ogden anisotropic material model coupled with haptic simulation

In the context of numerical surgical simulations, robot haptic feedback has been shown to be a consistent tool for drastically improving user interactions and opening up countless applications. Among them, haptic devices have mainly been used as training tools for surgeons. Indeed prior to surgery, under the assumption of known geometry and mechanical properties of the patient's organ, a surgeon would be able to plan and better choose between specific

Table 1 Relative error for displacement ($L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$) and strain ($L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$) defined in Eqs. 9 and 10 and mean Newton–Raphson (NR) iteration time between SOniCS and SOFA for different element geometries and interpolation schemes using Saint Venant–Kirchhoff material model

Saint Venant–Kirchhoff material model					
Element	Number of DOFs	SOniCS mean NR iteration time (s)	SOFA mean NR iteration time (s)	$L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$	$L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$
P1	10,935	0.387	0.438	4.10e−14	2.32e−16
P2	12,705	0.810	0.808	3.49e−14	8.18e−15
Q1	10,935	0.449	0.464	6.19e−13	3.76e−16
Q2	11,772	0.839	0.942	3.81e−13	23.68e−14

P1 and P2 elements stand for linear or quadratic tetrahedra, while Q1 and Q2 denote linear and quadratic hexahedra

Table 2 Relative error for displacement ($L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$) and strain ($L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$) defined in Eqs. 9 and 10 and mean Newton–Raphson (NR) iteration time between SOniCS and SOFA for different element geometries and interpolation schemes using Neo-Hookean material model

Neo-Hookean material model					
Element	Number of DOFs	SOniCS mean NR iteration time (s)	SOFA mean NR iteration time (s)	$L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$	$L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{SOFA}})$
P1	10,935	0.391	0.428	2.17e−14	7.78e−14
P2	12,705	0.826	0.852	1.40e−14	2.18e−20
Q1	10,935	0.471	0.478	4.92e−13	5.77e−16
Q2	11,772	0.826	0.864	1.64e−13	2.17e−14

P1 and P2 elements stand for linear or quadratic tetrahedra, while Q1 and Q2 denote linear and quadratic hexahedra

Table 3 Relative error for displacement ($L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$) and strain ($L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$) defined in Eq. 9 and 10 and mean Newton–Raphson (NR) iteration time between SOniCS and FEBio for different element geometries and interpolation schemes using Saint Venant–Kirchhoff material model

Saint Venant-Kirchhoff material model					
Element	Number of DOFs	SOniCS mean NR iteration time (s)	FEBio mean NR iteration time (s)	$L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$	$L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$
P1	10,935	0.387	0.401	6.01e−10	4.23e−7
P2	12,705	0.810	0.991	0.08	0.117
Q1	10,935	0.449	0.508	4.19e−10	2.96e−6
Q2	11,772	0.839	0.102	0.13	0.189

P1 and P2 elements stand for linear or quadratic tetrahedra, while Q1 and Q2 denote linear and quadratic hexahedra

surgical paths/approaches. In this paper, we used the 3D Systems Touch Haptic Device robot coupled with the SOFA plugin Geomagic to allow interactions between the instrument and the simulations. For this hypothetical simulation, we virtually simulate the contact between a surgical tool and a liver during surgery. The liver was described by an anisotropic Holzapfel Ogden model [107, 108], with an existing FEniCS implementation [52] validated using the manufactured solution. Therefore, to assess the soundness of our implementation, we conducted a mesh convergence analysis on a beam and liver meshes provided in Fig. 5. We prescribed natural Dirichlet boundary conditions on one side and Neumann boundary conditions on the other side to obtain noticeable deformations. Such material could be described using the following strain energy density function:

$$\psi = \psi_{\text{iso}}(\boldsymbol{F}) + \psi_{\text{vol}}(J). \tag{16}$$

Table 4 Relative error for displacement ($L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$) and strain ($L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$) defined in Eqs. 9 and 10 and mean Newton–Raphson (NR) iteration time between SOniCS and FEBio for different element geometries and interpolation schemes using Neo-Hookean material model

Neo-Hookean material model					
Element	Number of DOFs	SOniCS mean NR iteration time (s)	FEBio mean NR iteration time (s)	$L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$	$L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$
P1	10,935	0.391	0.458	6.53e−10	7.33e−7
P2	12,705	0.826	0.101	1.5e−2	3.16e−2
Q1	10,935	0.471	0.523	8.08e−10	1.02e−6
Q2	11,772	0.826	0.908	0.09	0.13

P1 and P2 elements stand for linear or quadratic tetrahedra, while Q1 and Q2 denote linear and quadratic hexahedra

Table 5 Relative error for displacement ($L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$) and strain ($L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$) defined in Eqs. 9 and 10 and mean Newton–Raphson (NR) iteration time between SOniCS and FEBio for different element geometries and interpolation schemes using Mooney–Rivlin material model

Mooney Rivlin material model					
Element	Number of DOFs	SOniCS mean NR iteration time (s)	FEBio mean NR iteration time (s)	$L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$	$L_E^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{FEBio}})$
P1	10,935	0.662	0.804	2.49e−9	6.81-8
P2	12,705	0.102	0.112	9.97e−3	8.28e−2
Q1	10,935	0.818	0.910	10.92	14.71
Q2	11,772	0.101	0.125	4.23e−2	1.229e−1

P1 and P2 elements stand for linear or quadratic tetrahedra, while Q1 and Q2 denote linear and quadratic hexahedra. The large errors obtained for Q1 elements are further discussed in Sect. 5

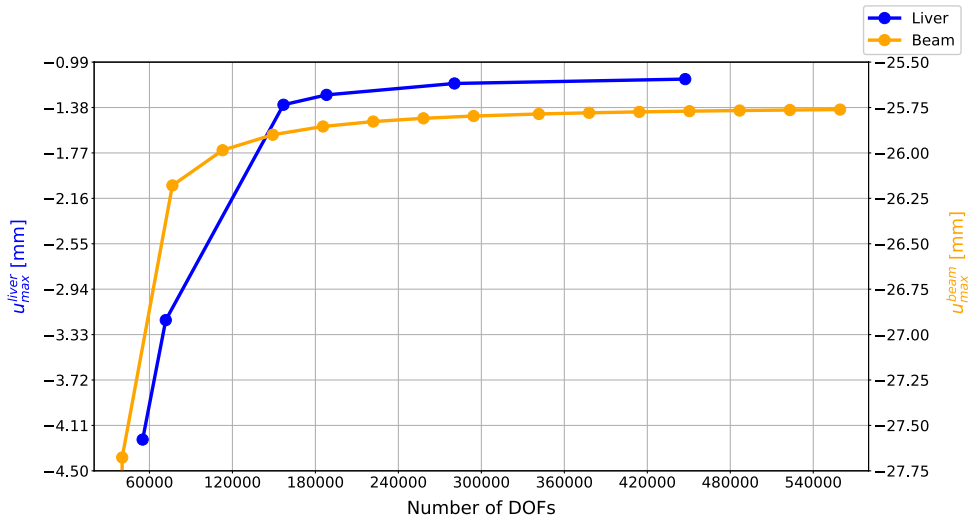


Fig. 5 Plot of the mesh convergence analysis applied to a beam and liver using the Holzapfel and Ogden anisotropic material models with the parameters λ Pa. The maximum displacement of a beam μ and liver $\boldsymbol{E}$ meshes are, respectively, computed in orange and blue for different mesh sizes increasing the number of Degrees of Freedom (DOFs)

ψ_{iso} and ψ_{vol} are the isochoric and volumetric part of the strain energy density function, respectively. The volumetric part can be evaluated as a function of the bulk modulus κ of the material and J

$$\psi_{\text{vol}}(J) = \frac{\kappa}{4}(J^2 - 1 - 2\ln(J)), \tag{17}$$

$$\psi_{\text{iso}}(\boldsymbol{F}) = \frac{a}{2b}\exp\left[b\left(\mathrm{I}_1 - 3\right)\right] + \sum_{i=f,s}\frac{a_i}{2b_i}\exp\left[b_i\left(\mathrm{I}_{4i} - 1\right)^2\right] + \frac{a_{\text{fs}}}{2b_{\text{fs}}}\left(\exp\left[b_{\text{fs}}\mathrm{I}_{8\text{fs}}^2\right] - 1\right), \tag{18}$$

with

$$\mathrm{I}_{4f} = \boldsymbol{f_0}\cdot\boldsymbol{C}\cdot\boldsymbol{f_0},\ \mathrm{I}_{4s} = \boldsymbol{s_0}\cdot\boldsymbol{C}\cdot\boldsymbol{s_0},\ \text{and}\ \mathrm{I}_{8\text{fs}} = \boldsymbol{f_0}\cdot\boldsymbol{C}\cdot\boldsymbol{s_0}. \tag{19}$$

The transversely isotropic behavior can be obtained by removing the parameters a_{fs}, b_{fs}, a_s, and b_s, while the isotropic behavior is obtained by also suppressing the two parameters a_f and b_f. This kind of model is frequently used to model orthotropic materials (e.g., muscle with fibers or tendons). Vectors $\boldsymbol{f_0}$, $\boldsymbol{s_0}$ are the unit base vectors normal to the planes of symmetry. For our application, we selected the following material properties allowing to obtain sufficient deformation of the objects: $\kappa = 10^2$ MPa, $a = 1.10^2$ kPa, $b = 5$ Pa, $a_f = 16$ kPa, $b_f = 12.8$ Pa, $a_s = 18$ kPa, $b_s = 10$ Pa, $a_{fs} = 9$ kPa, $b_{fs} = 12$ Pa.

The instrument is assumed to be a rigid body kinematically constrained by the haptic device position at each time step. The contact forces generated by the collision between the instrument and the liver are calculated using frictional contact and an implicit Euler scheme to solve the dynamic system [109]. Finally, the contact forces are transmitted back to the user's hand through the haptic device. The result is a simulation running at 100 FPS (Frames Per Seconds) on average displayed in Fig. 6. A full video of the interaction with the haptic device is available in the Supplementary Materials. The real-time performance has been obtained using a small number of DOFs (543). A higher number of DOFs could be used once a GPU implementation is available.

5 Discussion

We show that the SOniCS plugin is an efficient implementation of material models for hyperelastic simulations and enables the user to develop an intuitive understanding of the impact of modeling choices on the accuracy and reliability of the predictions. We first demonstrated a convergence study for Saint Venant–Kirchhoff material using P1 and P2 elements with the manufactured solution in Sect. 3.1. Indeed, as expected, by refining the mesh, the L^2 relative errors for strain and displacement fields almost follow the theoretical slopes when increasing the number of DOFs (on a log–log plot), showing the stability of our method.

Then, from Sect. 3.2, two main results are noteworthy from Tables 1 and 2. First, the relative errors of the displacement and strain between SOniCS and SOFA, for both material models, are close to machine precision for P1, P2, Q1, and Q2 discretizations. Second, the last comment concerns the mean NR iteration time. We observe that the SOniCS implementation is slightly faster than SOFA for any elements using Saint Venant–Kirchhoff or Neo-Hookean material model. The reason for this difference is the need for SOFA to compute the shape functions and derivatives, and then calculate the local residual and Jacobian using multiple tensor operations. Conversely, SOniCS has all those operations efficiently hard-coded in C kernels, thus performing faster than SOFA.

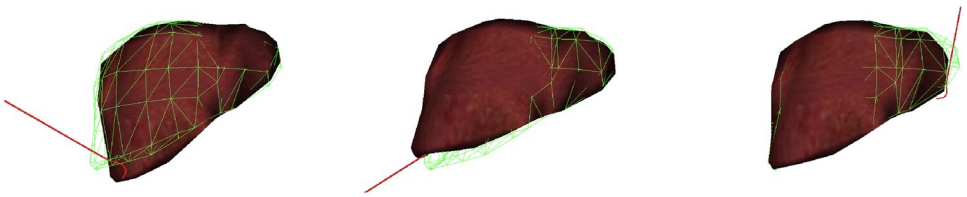


Fig. 6 Three different deformation states of a liver in contact with a surgical tool connected to a haptic device. The surgical tool (in red) is guided by the user through the 3D Systems Touch Haptic Device to deform the liver from the initial configuration (green wire-frame) to a deformed state (textured). The liver is modeled using the Holzapfel Ogden anisotropic material with the following parameters $\frac{1}{2}(\boldsymbol{C} - \boldsymbol{I})$. The maximum displacement of a beam $\boldsymbol{I}$ and liver $\boldsymbol{C}$. In the case of contact detection, the contact forces are transmitted to the user through the haptic device. A video of the simulation is available as supplementary materials

We compared the SOniCS and FEBio simulations for several material models: Saint Venant–Kirchhoff, Neo-Hookean, and Mooney–Rivlin in section 3.3. For P1 elements, the errors are close to machine precision for all three models. For P2 and Q2 elements, the three models display similar errors that still represent a minor error (less than 0.08 and 0.1, respectively) which is of same magnitude between SOFA and FEBio, as well as FEniCS and FEBio. The reasons for those minor errors could be the difference in the implementation of the elements or in the choice of solver parameters. For Q1 elements, we reached an accuracy near machine precision for Saint Venant–Kirchhoff and Neo-Hookean. However, the relative error for displacement rose close to 11 for the Mooney–Rivlin model, while similar behavior is observed for the relative strain errors. To further understand this divergence, we included a third open-source software AceGen [110]. AceGen similarly uses an automatic code generation package for the symbolic generation of new finite elements. The cantilever beam scenario presented in Sect. 3.3 was reproduced using AceGen under the same conditions and with an identical Q1 discretization of the domain. Using the same metric as defined in Eq. 9, the results are the following: $L_u^2(\boldsymbol{u}_{\texttt{SOniCS}}, \boldsymbol{u}_{\texttt{Acegen}}) = 3.33$ and $L_u^2(\boldsymbol{u}_{\texttt{FEBio}}, \boldsymbol{u}_{\texttt{Acegen}}) = 14.19$. Even if SOniCS and Acegen showed similar results, a more in-depth study would be needed to confirm the soundness of our solution and explain the differences between FEBio and SOniCS solutions. Despite using a finer mesh, the error was slightly lower but still noticeable. Eventually, FEBio has shown difficulty in converging with trivial parameter sets or when increasing the number of DOFs. Thus, as shown in Tables 3, 4, and 5, on average, SOniCS is solving the equation system faster than FEBio. Several reasons could explain those differences, such as the number of quadrature points used, the implementation of the Newton–Raphson scheme, or the solver parameters.

Finally, we showed the capabilities of the SOniCS plugin in simulating complex material models, such as the Holzapfel Ogden anisotropic model coupled with a haptic device in Sect. 4. The material model was effortlessly implemented for several elements (P1, P2, Q1, and Q2) without needing any manual derivation or coding. A manufactured solution was used in [52] to validate the FEniCS implementation, while Fig. 5 demonstrates a convergence of the solution for a beam and liver shape using the SOniCS implementation. Therefore, the absence of analytical solutions or experimental data only proves the convergence to a numerical solution that is not necessarily a ground truth. The final result is a real-time simulator functional for surgeons' training or any other biomechanics simulation replicating the behavior of a liver in contact with a surgical tool.

6 Conclusion and outlook

We performed several numerical experiments to develop intuitive understanding of new material models in SOFA using the SOniCS plugin. First, we validated the most common hyperelastic material models: Saint Venant–Kirchhoff and Neo-Hookean using a manufactured solution. Then, utilizing FEBio as a reference, we verified our implementation of a Mooney–Rivlin material model using P1, P2, Q1, and Q2 elements. The final application employed a haptic device to interact with an anisotropic Holzapfel Ogden liver model in real-time.

The study used our SOniCS plugin to generate optimized C code for complex material models compatible with SOFA. On one side, we benefited from FEniCS automatic differentiation and code generation capabilities to bypass the difficulties of deriving and implementing the consistent Jacobian in SOFA. On the user side, we implemented compatible and user-friendly SOFA Forcefields to use the FEniCS C kernels. A SOFA user can now easily define a new material model by specifying its strain energy function, element geometry or family, and the quadrature scheme and degree only in Python. We made the open-source code and all data and test cases available as supplementary material.

In future work, we intend to apply the SOniCS plugin for solving more complex mechanical phenomena. For example, mixed formulations for solving incompressible materials, viscous or plastic effects, and multi-material systems. In this paper, we only utilized the plugin for solving hyperelasticity equations, but it would be interesting to tackle multi-physics problems, such as thermomechanics or magnetomechanics. An interesting future work could be to investigate the effectiveness of the SOniCS plugin for complex material real-time simulations with surrogate and machine-learning models in SOFA. Further, the next step would be to quantify the uncertainty of biomechanical simulations in SOFA via the SOniCS plugin. We only used Lagrangian P1, P2, Q1, and Q2 elements, while more geometries such as prisms could be relevant for the SOFA and FEniCS community. Finally, our validation only focused on the deformation field of the structures, while a more in-depth study would also enable us to validate an analytical stress field.

Some work remains to improve the user-friendliness of our package. Indeed, even if writing the Python file describing the material model and generating the associated C file is mostly effortless and automated, some steps are still manual. Indeed, the `FEniCS_Material` C++ class must have knowledge of every new C file created at compile time. Hence, for the moment, the user still has to manually specify two C++

functions in the code and recompile the whole plugin. Even if this step is manageable, it still requires diving into the C++ code, which can discourage a few users. Meanwhile, to mitigate this effect, actions have been taken by providing detailed documentation and tutorials for this crucial step. In addition, future works aim at improving this stage by directly providing the path of the C file in the Python code to trigger just-in-time compilation for new materials.

Supplementary Information The online version of this article (https://doi.org/10.1007/s00366-023-01877-w) contains supplementary material, which is available to authorized users.

Acknowledgements The authors would like to thank Dr. Michal Habera and Dr. Matthew Scroggs for their help with the quadratic hexahedron Serendipity elements. The authors would also like to acknowledge Marie Amigo for helping with Fig. 1 and Thomas Lavigne for providing the Acegen simulation and reviewing the manuscript.

Funding This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie under Grant Agreement No. 764644. This publication only contains the RAINBOW consortium's views and the Research Executive Agency and the Commission are not responsible for any use that may be made of the information it contains. This publication has been prepared in the framework of the DRIVEN project funded by the European Union's Horizon 2020 Research and Innovation programme under Grant Agreement No. 811099. This project is supported by the "IHU Strasbourg—Institut de chirurgie guidée par l'image" Strasbourg, France. The authors would like to thank the support of the FNR and ANR Grant S-Keloid no. 16399490.

Data Availability The SOniCS plugin source code is available on GitHub as a branch of the Caribou project https://github.com/mimesis-inria/caribou/tree/FeniCS-features and will soon be merged into the main branch https://github.com/mimesis-inria/caribou/tree/master. The reference [1] (https://doi.org/10.6084/m9.figshare.21120118) contains the files used to generate the manufactured solutions, benchmarks with SOFA and FEBio, and the haptic simulation. The latest version is also available on GitHub at https://github.com/Ziemnono/SOniCS_validation.

Declarations

Conflict of interest The authors of this work have no conflicts of interest to disclose.

References

1. Mazier A, El Hadramy S, Brunet J-N, Hale JS, Cotin S, Bordas SPA (2022) Supplementary material for SOniCS: develop intuition on biomechanical systems through interactive error controlled simulations. https://doi.org/10.6084/m9.figshare.21120118
2. Smith M (2009) ABAQUS/Standard User's Manual, Version 6.9. Dassault Systèmes Simulia Corp, United States
3. DeSalvo GJ, Swanson JA (1985) ANSYS engineering analysis system users manual. Swanson analysis systems. Houston, PA
4. Geuzaine C, Remacle J-F (2009) Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Meth Eng 79:1309–1331. https://doi.org/10.1002/nme.2579
5. Ahrens J, Geveci B, Law C (2005) Paraview: an end-user tool for large data visualization. Visualization Handbook
6. Jasak H, Jemcov A, Kingdom U (2007) Openfoam: a c++ library for complex physics simulations. International workshop on coupled methods in numerical dynamics, IUC, pp 1–20
7. Bordas S, Nguyen VP, Dunant C, Nguyen Dang H, Guidoum A (2006) An extended finite element library. Int J Numer Meth Eng 2:1–33
8. Jansari C, Natarajan S, Beex L, Kannan K (2019) Adaptive smoothed stable extended finite element method for weak discontinuities for finite elasticity. Eur J Mech A Solids 78:103824. https://doi.org/10.1016/j.euromechsol.2019.103824
9. Jacquemin T, Bordas SPA (2021) A unified algorithm for the selection of collocation stencils for convex, concave, and singular problems. Int J Numer Meth Eng 122(16):4292–4312. https://doi.org/10.1002/nme.6703
10. Nguyen VP, Rabczuk T, Bordas S, Duflot M (2008) Meshless methods: a review and computer implementation aspects. Math Comput Simul 79(3):763–813. https://doi.org/10.1016/j.matcom.2008.01.003
11. Talebi H, Silani M, Bordas SPA, Kerfriden P, Rabczuk T (2013) A computational library for multiscale modeling of material failure. Comput Mech 53(5):1047–1071. https://doi.org/10.1007/s00466-013-0948-2
12. Sinaie S, Nguyen VP, Thanh Nguyen C, Bordas S (2017) Programming the material point method in Julia. Adv Eng Softw. https://doi.org/10.1016/j.advengsoft.2017.01.008
13. Gibbons CH (1934) History of testing machines for materials. Trans Newcomen Soc 15(1):169–184. https://doi.org/10.1179/tns.1934.011
14. Payan Y, Ohayon J (2017) Preface. In: Payan Y, Ohayon J (eds) Biomechanics of living organs. Translational epigenetics, vol 1. Academic Press, Oxford. https://doi.org/10.1016/B978-0-12-804009-6.10000-8
15. Mihai LA, Budday S, Holzapfel GA, Kuhl E, Goriely A (2017) A family of hyperelastic models for human brain tissue. J Mech Phys Solids 106:60–79. https://doi.org/10.1016/j.jmps.2017.05.015
16. Zhou J, Fung YC (1997) The degree of nonlinearity and anisotropy of blood vessel elasticity. Proc Natl Acad Sci 94(26):14255–14260. https://doi.org/10.1073/pnas.94.26.14255
17. Picinbono G, Delingette H, Ayache N (2003) Non-linear anisotropic elasticity for real-time surgery simulation. Graph Models 65(5):305–321. https://doi.org/10.1016/S1524-0703(03)00045-6. (**Special Issue on SMI 2002**)
18. Flynn C, Taberner A, Nielsen P (2011) Mechanical characterisation of in vivo human skin using a 3d force-sensitive microrobot and finite element analysis. Biomech Model Mechanobiol 10:27–38. https://doi.org/10.1007/s10237-010-0216-8

19. Boyer G, Molimard J, Ben Tkaya M, Zahouani H, Pericoi M, Avril S (2013) Assessment of the in-plane biomechanical properties of human skin using a finite element model updating approach combined with an optical full-field measurement on a new tensile device. J Mech Behav Biomed Mater 27:273–282. https://doi.org/10.1016/j.jmbbm.2013.05.024
20. Elouneg A, Sutula D, Chambert J, Lejeune A, Bordas SPA, Jacquet E (2021) An open-source fenics-based framework for hyperelastic parameter estimation from noisy full-field data: application to heterogeneous soft tissues. Comput. Struct. 255:106620. https://doi.org/10.1016/j.compstruc.2021.106620
21. Martins PALS, Natal Jorge RM, Ferreira AJM (2006) A comparative study of several material models for prediction of hyperelastic properties: application to silicone-rubber and soft tissues. Strain 42(3):135–147. https://doi.org/10.1111/j.1475-1305.2006.00257.x
22. Itskov M (2001) A generalized orthotropic hyperelastic material model with application to incompressible shells. Int J Numer Meth Eng 50(8):1777–1799. https://doi.org/10.1002/nme.86
23. Mihai LA, Chin L, Janmey P, Goriely A (2015) A comparison of hyperelastic constitutive models applicable to brain and fat tissues. J R Soc Interface 12:1–12. https://doi.org/10.1098/rsif.2015.0486
24. Chagnon G, Rebouah M, Favier D (2014) Hyperelastic energy densities for soft biological tissues: a review. J Elast. https://doi.org/10.1007/s10659-014-9508-z
25. Zeraatpisheh M, Bordas SPA, Beex LAA (2021) Bayesian model uncertainty quantification for hyperelastic soft tissue models. Data-Centric Eng 2:9. https://doi.org/10.1017/dce.2021.9
26. Ehlers W, Markert B (2001) A linear viscoelastic biphasic model for soft tissues based on the theory of porous media. J Biomech Eng 123:418–24. https://doi.org/10.1115/1.1388292
27. Haj-Ali RM, Muliana AH (2004) Numerical finite element formulation of the schapery non-linear viscoelastic material model. Int J Numer Meth Eng 59(1):25–45. https://doi.org/10.1002/nme.861
28. Marchesseau S, Heimann T, Chatelin S, Willinger R, Delingette H (2010) Fast porous visco-hyperelastic soft tissue model for surgery simulation: application to liver surgery. Prog Biophys Mol Biol 103(2):185–196. https://doi.org/10.1016/j.pbiomolbio.2010.09.005. (**Special Issue on Biomechanical Modelling of Soft Tissue Motion**)
29. Urcun S, Rohan P-Y, Skalli W, Nassoy P, Bordas SPA, Sciumè G (2021) Digital twinning of cellular capsule technology: emerging outcomes from the perspective of porous media mechanics. PLoS One 16(7):1–30. https://doi.org/10.1371/journal.pone.0254512
30. Simon BR (1992) Multiphase poroelastic finite element models for soft tissue structures. Appl Mech Rev 45(6):191–218. https://doi.org/10.1115/1.3121397
31. Cowin SC (1999) Bone poroelasticity. J Biomech 32(3):217–238. https://doi.org/10.1016/S0021-9290(98)00161-4
32. Stokes I, Chegini S, Ferguson S, Gardner-Morse M, Iatridis J, Laible J (2010) Limitation of finite element analysis of poroelastic behavior of biological tissues undergoing rapid loading. Ann Biomed Eng 38:1780–1788. https://doi.org/10.1007/s10439-010-9938-0
33. Richardson SIH, Gao H, Cox J, Janiczek R, Griffith BE, Berry C, Luo X (2021) A poroelastic immersed finite element framework for modelling cardiac perfusion and fluid-structure interaction. Int J Numer Meth Biomed Eng 37(5):3446. https://doi.org/10.1002/cnm.3446
34. Barrera O (2021) A unified modelling and simulation for coupled anomalous transport in porous media and its finite element implementation. Comput Mech. https://doi.org/10.1007/s00466-021-02067-5
35. Bulle R, Alotta G, Marchiori G, Berni M, Lopomo NF, Zaffagnini S, Bordas S, Barrera O (2021) The human meniscus behaves as a functionally graded fractional porous medium under confined compression conditions. Appl Sci 11:9405. https://doi.org/10.3390/app11209405
36. Lavigne T, Sciumè G, Laporte S, Pillet H, Urcun S, Wheatley B, Rohan P-Y (2022) Société de biomécanique young investigator award 2021: numerical investigation of the time-dependent stress-strain mechanical behaviour of skeletal muscle tissue in the context of pressure ulcer prevention. Clin Biomech 93:105592. https://doi.org/10.1016/j.clinbiomech.2022.105592
37. Han L, Hipwell J, Tanner C, Taylor Z, Mertzanidou T, Cardoso MJ, Ourselin S, Hawkes D (2011) Development of patient-specific biomechanical models for predicting large breast deformation. Phys Med Biol 57:455–472. https://doi.org/10.1088/0031-9155/57/2/455
38. Urcun S, Rohan P-Y, Sciumè G, Bordas S (2021) Cortex tissue relaxation and slow to medium load rates dependency can be captured by a two-phase flow poroelastic model. J Mech Behav Biomed Mater 126:104952. https://doi.org/10.1016/j.jmbbm.2021.104952
39. Tagliabue E, Piccinelli M, Dall'Alba D, Verde J, Pfeiffer M, Marin R, Speidel S, Fiorini P, Cotin S (2021) Intra-operative update of boundary conditions for patient-specific surgical simulation. Med Image Comput Comput Ass Interv MICCAI 2021:373–382
40. Courtecuisse H, Allard J, Kerfriden P, Bordas SPA, Cotin S, Duriez C (2014) Real-time simulation of contact and cutting of heterogeneous soft-tissues. Med Image Anal 18(2):394–410. https://doi.org/10.1016/j.media.2013.11.001
41. MiguezPacheco V, Hench L, Boccaccini A (2014) Bioactive glasses beyond bone and teeth: emerging applications in contact with soft tissues. Acta Biomaterialia. https://doi.org/10.1016/j.actbio.2014.11.004
42. Cotin S, Delingette H, Ayache N (1999) Real-time elastic deformations of soft tissues for surgery simulation. IEEE Trans Visual Comput Graphics 5(1):62–73. https://doi.org/10.1109/2945.764872
43. Lim Y-J, Hu J, Chang C-Y, Tardella N (2006) Soft tissue deformation and cutting simulation for the multimodal surgery training. In: 19th IEEE symposium on computer-based medical systems (CBMS'06), pp 635–640. 19th IEEE symposium on computer-based medical systems
44. Borazjani I (2013) Fluid-structure interaction, immersed boundary-finite element method simulations of bio-prosthetic heart valves. Comput Methods Appl Mech Eng 257:103–116. https://doi.org/10.1016/j.cma.2013.01.010
45. Bianchi D, Monaldo E, Gizzi A, Marino M, Filippi S, Vairo G (2017) A fsi computational framework for vascular physiopathology: a novel flow-tissue multiscale strategy. Med Eng Phys 47:25–37. https://doi.org/10.1016/j.medengphy.2017.06.028
46. Weiss JA, Maker BN, Govindjee S (1996) Finite element implementation of incompressible, transversely isotropic hyperelasticity. Comput Methods Appl Mech Eng 135(1):107–128. https://doi.org/10.1016/0045-7825(96)01035-3
47. Mazier A, Bilger A, Forte A, Peterlik I, Hale J, Bordas S (2022) Inverse deformation analysis: an experimental and numerical assessment using the fenics project. Eng Comput. https://doi.org/10.1007/s00366-021-01597-z
48. Allard J, Courtecuisse H, Faure F (2012) Chapter 21—implicit fem solver on gpu for interactive deformation simulation. In: Hwu WMW (ed) GPU computing gems, Jade. Applications of

GPU computing series. Morgan Kaufmann, Boston, pp 281–294. https://doi.org/10.1016/B978-0-12-385963-1.00021-6
49. Rappel H, Beex L, Hale J, Noels L, Bordas S (2019) A tutorial on Bayesian inference to identify material parameters in solid mechanics. Arch Comput Methods Eng. https://doi.org/10.1007/s11831-018-09311-x
50. Rappel H, Beex LAA, Noels L, Bordas SPA (2019) Identifying elastoplastic parameters with Bayes' theorem considering output error, input error and model uncertainty. Probab Eng Mech 55:28–41. https://doi.org/10.1016/j.probengmech.2018.08.004
51. Hauseux P, Hale JS, Bordas SPA (2017) Accelerating Monte Carlo estimation with derivatives of high-level finite element models. Comput Methods Appl Mech Eng 318:917–936. https://doi.org/10.1016/j.cma.2017.01.041
52. Hauseux P, Hale JS, Cotin S, Bordas SPA (2018) Quantifying the uncertainty in a hyperelastic soft tissue model with stochastic parameters. Appl Math Model 62:86–102. https://doi.org/10.1016/j.apm.2018.04.021
53. Bui HP, Tomar S, Courtecuisse H, Cotin S, Bordas SPA (2018) Real-time error control for surgical simulation. IEEE Trans Biomed Eng 65(3):596–607. https://doi.org/10.1109/TBME.2017.2695587
54. Odot A, Haferssas R, Cotin S (2022) Deepphysics: a physics aware deep learning framework for real-time simulation. Int J Numer Meth Eng 123(10):2381–2398. https://doi.org/10.1002/nme.6943
55. Deshpande S, Lengiewicz J, Bordas SPA (2022) Probabilistic deep learning for real-time large deformation simulations. Comput Methods Appl Mech Eng 398:115307. https://doi.org/10.1016/j.cma.2022.115307
56. Menard M (2011) Game development with unity, 1st edn. Course Technology Press, Boston
57. Sanders A (2016) An introduction to unreal engine 4. A. K. Peters Ltd, Natick
58. Comas O, Taylor Z, Allard J, Ourselin S, Cotin S, Passenger J (2008) Efficient nonlinear fem for soft tissue modelling and its gpu implementation within the open source framework sofa. Springer, New York, pp 28–39. https://doi.org/10.1007/978-3-540-70521-5_4
59. Verschoor M, Lobo D, Otaduy MA (2018) Soft hand simulation for smooth and robust natural interaction. In: 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pp 183–190. https://doi.org/10.1109/VR.2018.8447555. IEEE Conference on Virtual Reality and 3D User Interfaces (VR)
60. Turini G, Condino S, Fontana U, Piazza R, Howard J, Celi S, Positano V, Ferrari M, Ferrari V (2019) Software framework for vr-enabled transcatheter valve implantation in unity, pp 376–384. https://doi.org/10.1007/978-3-030-25965-5_28. International conference on augmented reality, virtual reality and computer graphics
61. Niroomandi S, González D, Alfaro I, Bordeu F, Leygue A, Cueto E, Chinesta F (2013) Real time simulation of biological soft tissues: a pgd approach. Int J Numer Meth Biomed Eng. https://doi.org/10.1002/cnm.2544
62. Wu J, Westermann R, Dick C (2014) Real-time haptic cutting of high-resolution soft tissues. Stud Health Technol Inform 196:469–75. https://doi.org/10.3233/978-1-61499-375-9-469
63. Gilles B, Bousquet G, Faure F, Pai DK (2011) Frame-based elastic models. ACM Trans Graph. https://doi.org/10.1145/1944846.1944855
64. Malgat R, Gilles B, Levin DIW, Nesme M, Faure F (2015) Multifarious hierarchies of mechanical models for artist assigned levels-of-detail. In: Proceedings of the 14th ACM SIGGRAPH/Eurographics symposium on computer animation. SCA '15, New York, NY, USA, pp 27–36. Association for computing machinery. https://doi.org/10.1145/2786784.2786800
65. Guo G, Zou Y, Liu PX (2021) A new rendering algorithm based on multi-space for living soft tissue. Comput Graph 98:242–254. https://doi.org/10.1016/j.cag.2021.06.003
66. Faure F, Duriez C, Delingette H, Allard J, Gilles B, Marchesseau S, Talbot H, Courtecuisse H, Bousquet G, Peterlik I, Cotin S (2012) SOFA: a multi-model framework for interactive physical simulation. In: Payan Y (ed) Soft tissue biomechanical modeling for computer assisted surgery. Studies in mechanobiology, tissue engineering and biomaterials, vol 11. Springer, Berlin, pp 283–321. https://doi.org/10.1007/8415_2012_125
67. Chinesta F, Keunings R, Leygue A (2014) The proper generalized decomposition for advanced numerical simulations. Primer. https://doi.org/10.1007/978-3-319-02865-1
68. Goury O, Amsallem D, Bordas S, Liu W, Kerfriden P (2016) Automatised selection of load paths to construct reduced-order models in computational damage micromechanics: from dissipation-driven random selection to bayesian optimization. Comput Mech. https://doi.org/10.1007/s00466-016-1290-2
69. Goury O, Duriez C (2018) Fast, generic, and reliable control and simulation of soft robots using model order reduction. IEEE Trans Rob 34(6):1565–1576. https://doi.org/10.1109/TRO.2018.2861900
70. Duriez C, Andriot C, Kheddar A (2004) A multi-threaded approach for deformable/rigid contacts with haptic feedback. In: 12th international symposium on haptic interfaces for virtual environment and teleoperator systems, 2004. HAPTICS '04. Proceedings, pp 272–279. https://doi.org/10.1109/HAPTIC.2004.1287206
71. Courtecuisse H, Jung H, Allard J, Duriez C, Lee DY, Cotin S (2010) Gpu-based real-time soft tissue deformation with cutting and haptic feedback. Prog Biophys Mol Biol 103(2):159–168. https://doi.org/10.1016/j.pbiomolbio.2010.09.016. (**Special Issue on Biomechanical Modelling of Soft Tissue Motion**)
72. Courtecuisse H, Allard J, Duriez C, Cotin S (2011) Preconditioner-based contact response and application to cataract surgery. In: Fichtinger G, Martel A, Peters T (eds) Med Image Comput Comput Assisted Interv MICCAI 2011. Springer, Berlin, pp 315–322
73. Alnaes MS, Blechta J, Hake J, Johansson A, Kehlet B, A Logg CR, Ring J, Rognes ME, Wells GN (2015) The FEniCS project version 1.5. Arch Numer Softw. https://doi.org/10.11588/ans.2015.100.20553
74. Lengiewicz J, Habera M, Zilian A, Bordas S (2021) Interfacing acegen and fenics for advanced constitutive models. In: Baratta I, Dokken JS, Richarson C, Scroggs MW (eds) Proceedings of FEniCS 2021, Online, 22–26 March, p 474. https://doi.org/10.6084/m9.figshare.14495463. http://mscroggs.github.io/fenics2021/talks/lengiewicz.html
75. Maas S, Ellis B, Ateshian G, Weiss J (2012) Febio: finite elements for biomechanics. J Biomech Eng 134:011005. https://doi.org/10.1115/1.4005694
76. Duriez C, Dubois F, Kheddar A, Andriot C (2006) Realistic haptic rendering of interacting deformable objects in virtual environments. IEEE Trans Visual Comput Graphics 12(1):36–47. https://doi.org/10.1109/TVCG.2006.13
77. Duriez C (2013) Control of elastic soft robots based on real-time finite element method. In: 2013 IEEE international conference on robotics and automation, pp 3982–3987. https://doi.org/10.1109/ICRA.2013.6631138. 2013 IEEE international conference on robotics and automation

78. Courtecuisse H, Allard J, Duriez C, Cotin S (2010) Asynchronous Preconditioners for Efficient Solving of Non-linear Deformations. In: VRIPHYS—virtual reality interaction and physical simulation. Eurographics Association, Copenhagen, Denmark, pp 59–68. https://doi.org/10.2312/PE/vriphys/vriphys10/059-068. https://hal.inria.fr/hal-00688865
79. Arruda EM, Boyce MC (1993) A three-dimensional constitutive model for the large stretch behavior of rubber elastic materials. J Mech Phys Solids 41(2):389–412. https://doi.org/10.1016/0022-5096(93)90013-6
80. Costa KD, Holmes JW, Mcculloch AD (2001) Modelling cardiac mechanical properties in three dimensions. Philos Trans R Soc Lond Ser A 359(1783):1233–1250. https://doi.org/10.1098/rsta.2001.0828
81. Mooney M (1940) A theory of large elastic deformation. J Appl Phys 11(9):582–592. https://doi.org/10.1063/1.1712836
82. Ogden RW (1972) Large deformation isotropic elasticity—on the correlation of theory and experiment for incompressible rubberlike solids. Proc R Soc Lond A Math Phys Sci 326(1567):565–584. https://doi.org/10.1098/rspa.1972.0026
83. Veronda DR, Westmann RA (1970) Mechanical characterization of skin-finite deformations. J Biomech 3(1):111–124. https://doi.org/10.1016/0021-9290(70)90055-2
84. Brunet J-N (2020) Exploring new numerical methods for the simulation of soft tissue deformations in surgery assistance. Theses, Université de Strasbourg, 4 Rue Blaise Pascal. https://hal.inria.fr/tel-03130643
85. Alnæs MS, Logg A, Ølgaard KB, Rognes ME, Wells GN (2014) Unified form language: a domain-specific language for weak formulations of partial differential equations. ACM Trans Math Softw 40(2):9–1937. https://doi.org/10.1145/2566630. (**Accessed 2015-03-10**)
86. Kirby RC, Logg A (2006) A compiler for variational forms. ACM Trans Math Softw 32(3):417–444. https://doi.org/10.1145/1163641.1163644. (**Accessed 2018-01-31**)
87. Rathgeber F, Ham DA, Mitchell L, Lange M, Luporini F, Mcrae ATT, Bercea G-T, Markall GR, Kelly PHJ (2016) Firedrake: automating the finite element method by composing abstractions. ACM Trans Math Softw (TOMS) 43(3):24–12427. https://doi.org/10.1145/2998441. (**Accessed 2020-01-20**)
88. Bastian P, Blatt M, Dedner A, Dreier N-A, Engwer C, Fritze R, Gräser C, Grüninger C, Kempf D, Klöfkorn R, Ohlberger M, Sander O (2021) The Dune framework: basic concepts and recent developments. Comput Math Appl 81:75–112. https://doi.org/10.1016/j.camwa.2020.06.007. (**Accessed 2022-06-01**)
89. Korelc J (2002) Multi-language and multi-environment generation of nonlinear finite element codes. Eng Comput 18:312–327. https://doi.org/10.1007/s003660200028
90. Ølgaard KB, Wells GN (2020) Applications in solid mechanics. In: Logg A, Mardal K-A, Wells G (eds) Automated solution of differential equations by the finite element method: the FEniCS Book. Lecture notes in computational science and engineering. Springer, Berlin, Heidelberg, pp 505–524. https://doi.org/10.1007/978-3-642-23099-8_26. Accessed 2022-07-04
91. Narayanan H (2012) A computational framework for nonlinear elasticity. In: Logg A, Mardal K-A, Wells G (eds) Automated solution of differential equations by the finite element method. Lecture notes in computational science and engineering. Springer, pp 525–541. https://doi.org/10.1007/978-3-642-23099-8_27 Accessed 2015-03-10
92. Baroli D, Quarteroni A, Ruiz-Baier R (2012) Convergence of a stabilized discontinuous galerkin method for incompressible nonlinear elasticity. Adv Comput Math 39(2):425–443. https://doi.org/10.1007/s10444-012-9286-8
93. Phunpeng V, Baiz PM (2015) Mixed finite element formulations for strain-gradient elasticity problems using the fenics environment. Finite Elem Anal Des 96:23–40. https://doi.org/10.1016/j.finel.2014.11.002
94. Weis JA, Miga MI, Yankeelov TE (2017) Three-dimensional image-based mechanical modeling for predicting the response of breast cancer to neoadjuvant therapy. Comput Methods Appl Mech Eng 314:494–512. https://doi.org/10.1016/j.cma.2016.08.024. (**Special Issue on Biological Systems Dedicated to William S. Klug**)
95. Nguyen-Thanh VM, Zhuang X, Rabczuk T (2019) A deep energy method for finite deformation hyperelasticity. Eur J Mech A/Solids. https://doi.org/10.1016/j.euromechsol.2019.103874
96. Patte C, Genet M, Chapelle D (2022) A quasi-static poromechanical model of the lungs. Biomech Model Mechanobiol 21(2):527–551. https://doi.org/10.1007/s10237-021-01547-0
97. Scroggs MW, Baratta IA, Richardson CN, Wells GN (2022) Basix: a runtime finite element basis evaluation library. submitted to Journal of Open Source Software
98. Kirby RC (2004) Algorithm 839: FIAT, a new paradigm for computing finite element basis functions. ACM Trans Math Softw 30(4):502–516. https://doi.org/10.1145/1039813.1039820. (**Accessed 2018-01-31**)
99. ...Balay S, Abhyankar S, Adams MF, Benson S, Brown J, Brune P, Buschelman K, Constantinescu E, Dalcin L, Dener A, Eijkhout V, Faibussowitsch J, Gropp WD, Hapla V, Isaac T, Jolivet P, Karpeev D, Kaushik D, Knepley MG, Kong F, Kruger S, May DA, McInnes LC, Mills RT, Mitchell L, Munson T, Roman JE, Rupp K, Sanan P, Sarich J, Smith BF, Zampini S, Zhang H, Zhang H, Zhang J (2022) PETSc/TAO users manual. Technical Report ANL-21/39—Revision 3.18, Argonne National Laboratory
100. Rodenberg B, Desai I, Hertrich R, Jaust A, Uekermann B (2021) FEniCS-preCICE: coupling FEniCS to other simulation software. SoftwareX 16:100807. https://doi.org/10.1016/j.softx.2021.100807. (**Accessed 2022-07-05**)
101. Arnold DN, Awanou G (2011) The serendipity family of finite elements. Found Comput Math 11(3):337–344. https://doi.org/10.1007/s10208-011-9087-3. (**Accessed 2022-07-05**)
102. Arbogast T, Tao Z, Wang C (2022) Direct serendipity and mixed finite elements on convex quadrilaterals. Numer Math 150(4):929–974. https://doi.org/10.1007/s00211-022-01274-3. (**Accessed 2022-07-05**)
103. Zienkiewicz OC, Taylor RL, Fox D (2014) The finite element method for solid and structural mechanics, 7th edn. Butterworth-Heinemann, Oxford. https://doi.org/10.1016/B978-1-85617-634-7.00016-8
104. Ralston A, Rabinowitz P (2001) A first course in numerical analysis, 2nd edn. Dover Publications, New York
105. Chamberland É, Fortin A, Fortin M (2010) Comparison of the performance of some finite element discretizations for large deformation elasticity problems. Comput Struct 88(11–12):664–673. https://doi.org/10.1016/j.compstruc.2010.02.007
106. ...Meurer A, Smith CP, Paprocki M, Čertík O, Kirpichev SB, Rocklin M, Kumar A, Ivanov S, Moore JK, Singh S, Rathnayake T, Vig S, Granger BE, Muller RP, Bonazzi F, Gupta H, Vats S, Johansson F, Pedregosa F, Curry MJ, Terrel AR, Roučka V, Saboo A, Fernando I, Kulal S, Cimrman R, Scopatz A (2017) Sympy: symbolic computing in python. PeerJ Comput Sci 3:103. https://doi.org/10.7717/peerj-cs.103
107. Holzapfel GA, Ogden RW (2009) Constitutive modelling of passive myocardium: a structurally based framework for material characterization. Philos Trans R Soc A Math Phys Eng Sci 367(1902):3445–3475. https://doi.org/10.1098/rsta.2009.0091
108. Pezzuto S, Ambrosi D, Quarteroni A (2014) An orthotropic active–strain model for the myocardium mechanics and its numerical approximation. Eur J Mech A Solids 48:83–96. https://doi.org/10.1016/j.euromechsol.2014.03.006

109. Courtecuisse H, Allard J, Kerfriden P, Bordas S, Cotin S, Duriez C (2013) Real-time simulation of contact and cutting of heterogeneous soft-tissues. Med Image Anal 18:394–410. https://doi.org/10.1016/j.media.2013.11.001
110. Korelc J (2022) AceGen/AceFEM website and user manuals. http://symech.fgg.uni-lj.si/ Accessed 20 Jun 2022