Making coupled-field Abaqus user elements simple

Teng Zhang[a,b,*]

[a] *Department of Mechanical and Aerospace Engineering, Syracuse University, Syracuse, NY, 13244, USA*
[b] *BioInspired Syracuse, Syracuse University, Syracuse, NY, 13244, USA*

Abstract

Soft materials, energy storage, and advanced manufacturing couple mechanics with transport, reaction, or damage, so their simulation must solve several interacting fields together. In Abaqus, custom coupled formulations are typically implemented through user elements (UELs), and a coupled UEL must supply the residual, an iteration matrix (ideally the consistent tangent), degree-of-freedom maps, and state updates in exactly the form the solver expects. Deriving and debugging these ingredients, especially the unsymmetric cross-field tangent blocks, separates a model on paper from a working simulation, and an implementation error is hard to distinguish from a modeling choice. We present `abaqus_ufl`, a Python framework in which the user declares the fields and their interpolations and selects and configures supported balance templates for the governing equations. The framework generates a self-contained Fortran UEL and evaluates the required tangent blocks with the established complex-step approximation. Correctness is addressed by layered verification, from Python derivative checks through compiled single-point tests and a lightweight assembly runtime to Abaqus benchmarks, and each demonstration reports the evidence level it actually reached. The examples progress from reproducing an established corrosion element to stabilized, mixed-order, and locally condensed formulations exercised in three-dimensional morphing.

Keywords: Abaqus, user element, complex-step differentiation, verification,

*Corresponding author.
Email address: `tzhang48@syr.edu` (Teng Zhang)

1. Introduction

Many problems in soft materials, fracture, transport, and manufacturing require several interacting fields. Examples include solvent transport and equilibrium swelling in gels [1–4], dissolution and species transport in corrosion [5], and nonlocal damage in elastomers [6]. Field–mechanics coupling also arises in hard-magnetic solids and magnetorheological elastomers [7–9], electroactive elastomers [10], and liquid-crystal elastomers [11, 12]. Related transport–mechanics interactions occur during food drying [13] and in rechargeable batteries, where chemo-mechanical coupling drives heterogeneous electrode damage [14, 15] and, in solid-state cells, electrochemistry, heat, dendrite growth, and fracture may evolve together [16, 17]. These applications differ in physics and scale, but each requires mechanics to be solved with other evolving fields.

Abaqus [18] provides nonlinear solution procedures and a user-element subroutine (UEL) for introducing custom fields. A coupled UEL must nevertheless supply the complete residual, tangent, and state update in the form required by the solver. The fields may use different interpolations and nodal supports, while their tangent contains many cross-derivatives and is generally unsymmetric. The implementation must also map the mathematical variables to Abaqus degree-of-freedom numbers, quadrature data, solution increments, and state variables. Because these choices interact, an incorrect sign or mapping can be difficult to distinguish from an error in the model or analysis setup.

Existing methods address different parts of this implementation problem. Problem-specific formulations and solver extensions have been developed for phase-field fracture [19–21], magnetomechanics [8, 9], coupled geological media [22], hydrogel chemo-mechanics [23], and electromechanical fracture [24]. General tools provide symbolic finite-element code generation [25], executable weak forms [26–29], multiphysics solution environments [30, 31], or constitutive interfaces [32]. A coupled Abaqus UEL still requires a chosen weak form to be

translated into field-specific maps, residual and tangent blocks, local variables, and solver data structures.

We present `abaqus_ufl`, which translates material responses, field declarations, and weak-form contributions into a self-contained Fortran UEL with no Python runtime dependency. Shared metadata define the residual, tangent, interpolation, and state layout, while intermediate response derivatives remain available for focused tests. The contribution is an inspectable translation and verification workflow, not a new constitutive model or solution algorithm. We demonstrate the workflow through an established corrosion update, a stabilized u–ϑ Tet4, mixed-order gel interpolation, and three-dimensional morphing with element-local pressure.

2. The abaqus_ufl framework

The framework separates constitutive physics, discretization metadata, and the Abaqus contract while linking them through shared field and weak-form declarations that drive the residual map, tangent blocks, and Fortran bookkeeping. Because governing equations do not uniquely determine interpolation, stabilization, time integration, or local-variable treatment, the user chooses a formulation and `abaqus_ufl` generates a deterministic, inspectable candidate.

2.1. Material and weak-form layers

The interface separates constitutive physics from element bookkeeping. The first layer is a material object. For a finite-strain solid, the material returns quantities such as the first Piola stress as functions of the deformation gradient and any additional fields. Throughout, $\mathbf{F} = \mathbf{I} + \mathrm{Grad}_X\, \mathbf{u}$, $J = \det \mathbf{F}$, and $\mathbf{C} = \mathbf{F}^{\mathsf{T}}\mathbf{F}$. For a transport or phase-field law, the same object may also return a storage term, a flux, or a local constraint. The user writes these functions in a restricted Python subset using fixed-size tensor operations that mirror the mathematical notation, and the material knows nothing about the mesh or the Abaqus storage layout.

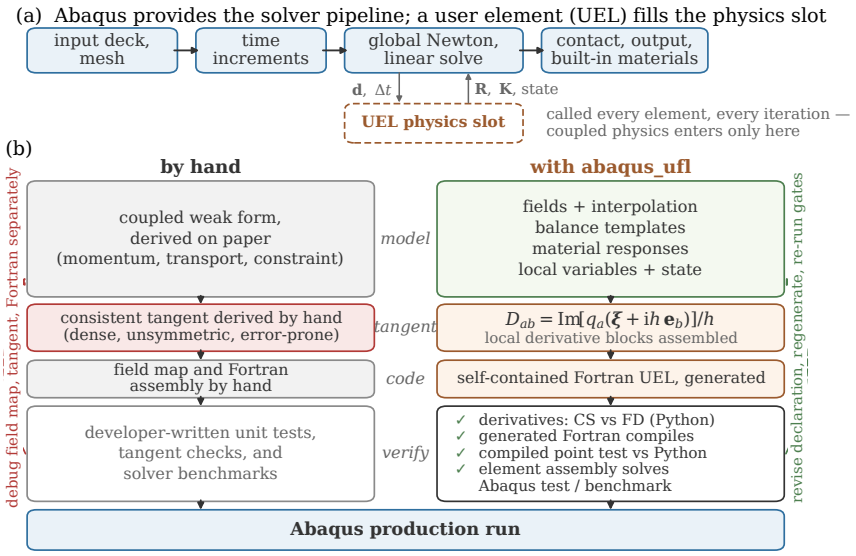

Figure 1: Route from a coupled model to an Abaqus UEL. Abaqus supplies assembly and solution, and the UEL returns `RHS`, `AMATRX`, and `SVARS`. abaqus_ufl derives the field map, derivative blocks, and Fortran from one declaration, so development runs as a closed loop gated by the staged checks.

The second layer is a weak-form object. It declares the solved fields and their interpolation, then selects supported residual templates through method names and signatures. The current templates cover momentum, local constraints, and storage–flux balances. They are not a compiler for arbitrary variational expressions. Each method connects a template to material responses and declares whether a test field enters through its value or gradient, and a coupled element may add a phase field, concentration, chemical potential, pore pressure, or temperature to the displacement field. The generated code then handles shape functions, quadrature, degree-of-freedom ordering, state storage, and the Abaqus UEL interface.

Let $\mathbf{d}_e$ collect the active nodal values of displacement and m additional fields. The generator assembles their residual contributions as

$$\mathbf{R}_e(\mathbf{d}_e) = [\,\mathbf{R}_u^{\mathsf{T}}, \mathbf{R}_1^{\mathsf{T}}, \ldots, \mathbf{R}_m^{\mathsf{T}}\,]^{\mathsf{T}}, \qquad \mathbf{K}_e = \frac{\partial \mathbf{R}_e}{\partial \mathbf{d}_e}. \tag{1}$$

With the residual convention used here, the Abaqus arrays are $\texttt{RHS} = -\mathbf{R}_e$ and $\texttt{AMATRX} = \mathbf{K}_e$. The sign conventions, the supported procedure requests, and the `UNSYMM` declaration required by the unsymmetric coupled blocks are collected in the Supplementary Material. Each field retains its declared interpolation and node support when these blocks are packed into the element arrays.

This is a small Abaqus-targeted interface, not a general variational language such as UFL [26] or a complete finite-element environment such as FEniCS or DOLFINx [27, 28]. Its name nods to the weak-form style popularized by UFL, but the framework neither depends on nor implements UFL. It emits one self-contained Fortran source file without a Python runtime or external finite-element library. The examples in this paper are representative rather than an exhaustive inventory of the framework. Version-specific coverage is maintained with the package documentation and release notes. Figure 2 shows the user-facing scale for the pressure-based gel variant used here.

2.2. Tangent blocks from the declared interface

The framework treats tangent construction first as a mapping problem. The arguments of each material response identify its current-field dependencies, while previous-step fields, stored state, and the time increment remain fixed, and the weak form identifies the test field and its value or gradient operator. Together, these declarations determine the retained local derivative blocks and their positions in the element Jacobian without a separately prescribed block table or Abaqus array indexing.

Once a block has been identified, the generated UEL evaluates its local derivative using the established complex-step approximation [33–36]. For a scalar function,

$$\frac{\partial f}{\partial x} = \frac{\mathrm{Im}[\, f(x + \mathrm{i}\, h)\,]}{h} + \mathcal{O}(h^2). \tag{2}$$

Because Eq. (2) involves no subtractive cancellation, the step can be very small ($h = 10^{-10}$ throughout this work) and the derivative is accurate to working precision on complex-safe paths. Complex-step differentiation is not introduced here as a new algorithm. Its role is to avoid hand differentiation of the declared local blocks, while branches, clipping, and unconverged local iterations require separate treatment and tests.

At each integration point, the generated UEL combines the resulting local derivatives with the declared value and gradient shape-function operators to assemble $\mathbf{K}_e$. The interface-critical choice is not complex-step itself, but the use of the same declarations to identify, differentiate, and assemble each retained coupling block. This numerical route complements symbolic generators such as AceGen, which derive and optimize the element Jacobian symbolically [25]. `abaqus_ufl` instead performs one perturbed evaluation per scalar input component (three for a vector, nine for a 3×3 tensor) plus the unperturbed call, trading some speed for generality and for eliminating hand-derivation errors.

2.3. Code generation

The generator translates the Python model into Fortran routines for material response, residual, tangent, and state updates. It also writes the UEL wrapper

that unpacks nodal fields, reads properties and history, and assembles `RHS`, `AMATRX`, and `SVARS`. Stabilization responses are selected explicitly rather than activated by an element switch. A separate F-bar generator remains limited to single-field Quad4 and Hex8 mechanics [37].

Robustness comes from removing duplicated declarations rather than from assuming generated code is correct. One field layout controls `NDOFEL`, unpacking, interpolation, packing, and the optional input-deck declaration, and stored state is updated only by the unperturbed call. These invariants reduce representational drift but do not verify the formulation. The emitted Fortran favors transparency over optimization, so repeated operations remain visible in Fig. 2, and the user submits one input deck and one Fortran file.

The generator also emits solution structure, not only material code. The element-local pressure route shown in Fig. 2 is the example. There the global unknowns are $\mathbf{x} = [\mathbf{u}, \mu]$, and one constant internal pressure p_e per element is solved at each UEL call from the element constraint $R_p^e(p_e) = 0$ with $R_p^e = \int_{\Omega_e} r_p(\mathbf{F}, p_e, \mu)\, dV$. With $\mathbf{K}_{ab} = \partial \mathbf{R}_a / \partial \mathbf{b}$, eliminating the local pressure increment condenses the element arrays,

$$\mathbf{K}_{xx}^{\mathrm{cond}} = \mathbf{K}_{xx} - \mathbf{K}_{xp}\mathbf{K}_{pp}^{-1}\mathbf{K}_{px}, \qquad \mathbf{R}_x^{\mathrm{cond}} = \mathbf{R}_x - \mathbf{K}_{xp}\mathbf{K}_{pp}^{-1}\mathbf{R}_p, \tag{3}$$

which the generated wrapper returns to Abaqus with the pressure stored in `SVARS(1)`. *Local pressure* thus denotes an internal constraint variable, not an applied load. The local-solve algorithm and its controls are given in the Supplementary Material.

2.4. Layered verification

No single check covers the model, its translation, element assembly, and production solver behavior, so verification proceeds as a ladder. `problem.verify()` compares local complex-step derivatives with centered finite differences at prescribed states. The emitted source must compile, and a thin `f2py` wrapper [38] then executes a generated subroutine at one material point or one element against a Python or finite-difference reference. The same source can be linked

(a) Minimal user declaration, adapted from the repository gel example (complete executable version in the Supplementary Material; illustrative property values).

```python
class PressureGelMaterial(au.Material):
    props = dict(G=1.0, K=100.0, chi=0.1, D=5.0e-9, mu0=0.0,
                 Omega=1.0e-4, Rgas=8.314, theta=298.0, phi0=0.5)
    def stress_PK1(self, F, p, mu):
        return self.G * (F - inv(F).T) + p * inv(F).T
    # pressure_resid, solvent_storage, solvent_flux: Eqs. (8b,d,e)

class PressureGelLocalPressureQuad4(au.WeakForm):
    material = PressureGelMaterial
    ndim = 2
    def define_fields(self):
        self.u = au.VectorField('u', degree=1)
        self.mu = au.ScalarField('mu', degree=1)
        self.p = au.LocalScalar('p', storage='SVARS',
                                condensed=True, initial=0.0)
    def momentum_equation(self, v, F, p, mu):
        return self.material.stress_PK1(F, p, mu)
    # pressure_equation / transport_equation: one call each

generate_uel(problem, 'gel_uel.for', element='Quad4',
             formulation='local_pressure',
             mat_prefix='pressuregel')
```

(b) Generated UEL excerpt. `Rtmp` is the packed $-\mathbf{R}_x$ contribution before the pressure correction.

```
RHS(i,1) = Rtmp(i) + Kxp(i)*Rp_final/K_pp
AMATRX(i,j) = Kxx(i,j) - Kxp(i)*Kpx(j)/K_pp
SUBROUTINE pressuregel_stress_PK1(F, p, mu, props, P_out)
  CALL inv33z(F, INV1)
  CALL transpose33z(INV1, AT2)
  CALL inv33z(F, INV3)
  CALL transpose33z(INV3, AT4)
  ...
     P_out(ii,jj) = props(1)*(F(ii,jj) - AT2(ii,jj))
    & + p*AT4(ii,jj)
```

Figure 2: Declaration-to-element workflow, a minimal declaration adapted from the gel example (complete executable version in Listing S1, property values illustrative). The material supplies local responses, the weak form declares fields, interpolation, and the condensed pressure, and one generator call emits the Fortran. The excerpt shows the Schur terms of Eq. (3) and the emitted stress routine.

into the lightweight `feacheap` runtime, derived from Bower's EN234_FEA teaching code [39], for assembled nonlinear solves before the element reaches Abaqus smoke tests and benchmark reproduction. Each demonstration below states the highest level actually reached. Agreement among Python, generated Fortran, and finite differences is an implementation consistency check, while theory-level validation requires independent evidence such as an analytical solution or a published benchmark.

3. From coupled theories to running Abaqus elements

The four examples each exercise a distinct capability that coupled Abaqus elements need in practice: the corrosion benchmark reproduces a reference implementation whose iteration matrix suppresses the off-diagonal tangent blocks, the stabilized tetrahedron adds a variational-multiscale term through the declared weak form, the gel bilayer requires node-dependent degree-of-freedom maps for its mixed-order interpolation, and the final example condenses one pressure inside each element in a three-dimensional morphing geometry. The constitutive models and solution strategies come from the cited literature. Table S1 in the Supplementary Material summarizes the verification evidence and evidence tier of each demonstration. An example is called a verified benchmark only when an independently constructed quantitative oracle exists, and execution demonstrations are labeled as such.

3.1. Phase-field stress-corrosion cracking

We first consider the phase-field stress-corrosion formulation of Cui, Ma, and Martínez-Pañeda [5]. It couples displacement $\mathbf{u}$, a dissolution phase field ϕ, and concentration c, together with plasticity and history-dependent reaction kinetics. The example reproduces this three-field model and the block-diagonal update of its reference implementation in Abaqus.

With $H(\phi) = \phi^2(3 - 2\phi)$, the implemented balances can be summarized as

$$\nabla \cdot \boldsymbol{\sigma} = \mathbf{0}, \qquad \boldsymbol{\sigma} = [H(\phi) + \kappa_r]\boldsymbol{\sigma}_0, \tag{4a}$$

$$\frac{\dot{\phi}}{\mathcal{L}_n} + \frac{\partial \psi_{\mathrm{ec}}}{\partial \phi} - \nabla \cdot (\alpha \nabla \phi) = 0, \tag{4b}$$

$$\dot{c} + \nabla \cdot \mathbf{j}_c = 0, \qquad \mathbf{j}_c = -D[\nabla c - \Delta c\, H'(\phi)\nabla \phi]. \tag{4c}$$

Here c is normalized concentration, $\boldsymbol{\sigma}_0$ is the J2 return-map stress, $\mathcal{L}_n$ is the kinetic coefficient, and α is the gradient-energy coefficient. Further, $\psi_{\mathrm{ec}} = A[c - c_e(\phi)]^2 + \omega\phi^2(1-\phi)^2$, $c_e = c_l + H(\phi)\Delta c$, and $\Delta c = c_s - c_l$. The phase field takes $\phi = 1$ in metal and $\phi = 0$ in electrolyte. The mechanical coupling enters through the implemented kinetic coefficient, following [5],

$$\begin{aligned} \mathcal{L}_n &= L_0 \left(1 + \frac{\bar{\varepsilon}^p}{\varepsilon_y}\right) \mathrm{e}^{V_m \sigma_h/(RT)}\, f_{\mathrm{rep}}, \qquad \varepsilon_y = \sigma_y / E, \\ f_{\mathrm{rep}} &= \begin{cases} 1, & t_i < t_0, \\ \exp[-k_{\mathrm{rep}}(t_i - t_0)], & t_i \geq t_0, \end{cases} \end{aligned} \tag{5}$$

where σ_h is the damaged hydrostatic stress, $\bar{\varepsilon}^p$ is the equivalent plastic strain, and the repassivation clock t_i resets when the accumulated plastic strain crosses its threshold. The constants V_m, R, and T take the reference implementation's values (Supplementary Material). The benchmark uses a small-strain material class whose damaged J2 stress enters through the element's stress interface (adapter details in the Supplementary Material). Both rates use backward Euler. In the generated element $\mathcal{L}_n$ is lagged from the previous increment, whereas the reference UEL evaluates it from the current update, so the comparison tests a semi-implicit variant of the same balance.

All three fields use the all-node interpolation of a reduced-integration Quad8 element. The comparison in Fig. 3 uses the Cui-style diagonal variant, with $\kappa_r = 10^{-7}$ and a maximum time increment of 0.25 s. It retains the coupled residual while suppressing every off-diagonal block,

$$\widetilde{\mathbf{K}} = \mathrm{diag}(\mathbf{K}_{uu}, \mathbf{K}_{\phi\phi}, \mathbf{K}_{cc}). \tag{6}$$

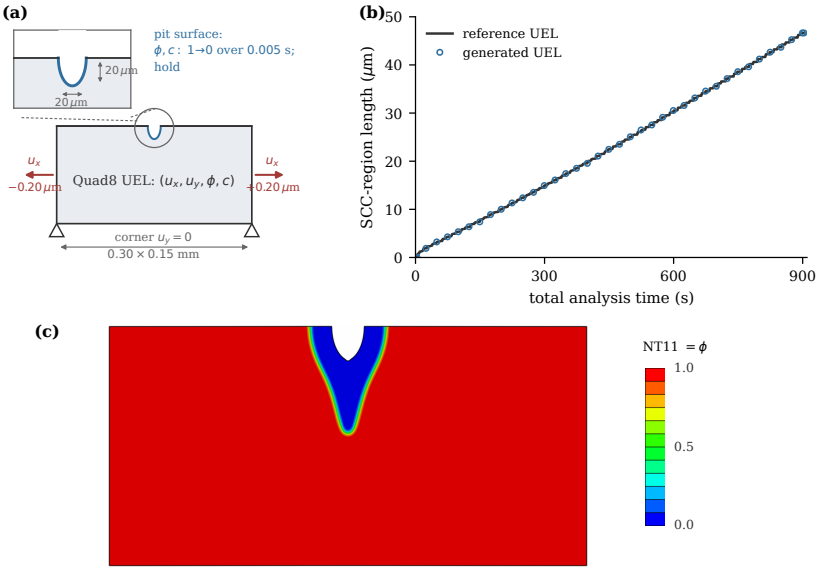

Figure 3: Code-to-code reproduction of the semi-elliptical-pit corrosion benchmark distributed with the Cui reference implementation. (a) Problem setup. (b) SCC-region length (depth at which ϕ crosses 0.5) from the reference UEL's output database and from the generated UEL, both ending at 46.64 μm. Interpolating the generated history to the reference output times gives mean/maximum differences of 0.13/0.63 μm. (c) Final ϕ from the generated UEL.

Within one Abaqus solve this is a block-Jacobi, staggered-style update that changes the iteration matrix but not the residual equations.

The history agreement in Fig. 3(b) supports benchmark reproduction, not a convergence advantage of block suppression. The normalized L_2 difference is 0.64%, with the maximum deviation near $t = 9.5$ s under three documented implementation differences: a boundary-pin adaptation, the lagged kinetic update, and the reference code's suppression of the repassivation clock during the initial setup steps, which the generated element does not replicate. A full-field comparison of the final ϕ and c distributions gives normalized L_2 errors of 1.1×10^{-3} and 1.4×10^{-3}, with the largest local differences near the advancing pit front.

3.2. Stabilized mixed u–ϑ tetrahedron

The second example implements the kinematically stabilized formulation of Scovazzi, Zorrilla, and Rossi [40], in which displacement and an independent volume-ratio field $\vartheta \approx J$ share piecewise-linear Tet4 interpolation. The equal-order pair admits spurious volumetric modes, and the published variational-multiscale remedy adds an h^2-scaled term involving $\nabla\vartheta$, a mesh-dependent stabilization rather than a material length scale. The code stores $\widetilde{\vartheta} = \vartheta - 1$ and reconstructs ϑ in the material routines.

For the block benchmark in Fig. 4, let $\overline{\mathbf{C}} = (\vartheta/J)^{2/3}\mathbf{C}$. The stabilized constitutive quantities and residuals are

$$\mathbf{S} = \mu\mathbf{I} + (\lambda \ln\vartheta - \mu)\overline{\mathbf{C}}^{-1}, \tag{7a}$$

$$\boldsymbol{\mathcal{D}} = (3\lambda + 2\mu - 2\lambda\ln\vartheta)\overline{\mathbf{C}}^{-1}, \tag{7b}$$

$$\mathbf{P}^{\star} = \mathbf{F}\left[\mathbf{S} - \tau_\vartheta(\vartheta - J)\frac{\boldsymbol{\mathcal{D}}}{3\vartheta}\right], \qquad \boldsymbol{\mathcal{G}} = \frac{\tau_u}{3}\frac{J}{\vartheta}\boldsymbol{\mathcal{D}}\nabla_X\vartheta, \tag{7c}$$

$$\mathcal{R}_u(\mathbf{v}) = \int_{\Omega_0} \mathbf{P}^{\star} : \nabla_X\mathbf{v}\, dV - \mathcal{W}_{\text{ext}} = 0, \tag{7d}$$

$$\mathcal{R}_\vartheta(q) = \int_{\Omega_0} [(1-\tau_\vartheta)(\vartheta - J)q + \boldsymbol{\mathcal{G}}\cdot\nabla_X q]\, dV = 0. \tag{7e}$$

Here $\tau_u = c_{\tau u}h_e^2/(2\mu)$ and $\tau_\vartheta = c_{\tau\vartheta}\mu/(\mu + K)$, where λ and μ are the Lamé constants and $K = \lambda + 2\mu/3$. Because the material-response interface does not expose element geometry, $h_e = V_e^{1/3}$ is supplied through 16 element-size property bins in the unstructured Abaqus mesh. The accuracy of this binning is quantified in the Supplementary Material.

The weak-form adapter selects the internal momentum contribution and the ϑ reaction and flux terms in Eq. (7). External loading is supplied through Abaqus. The generated UEL forms seven local derivative blocks for the stress, reaction, and stabilizing flux.

The distorted-tetrahedron affine patch test gives a maximum absolute nodal error below 2.4×10^{-16}. Because the stabilization flux vanishes in an affine state, the element suite additionally verifies deliberately non-affine states with $\nabla_X\vartheta \neq \mathbf{0}$, comparing the active stabilization blocks by complex step against

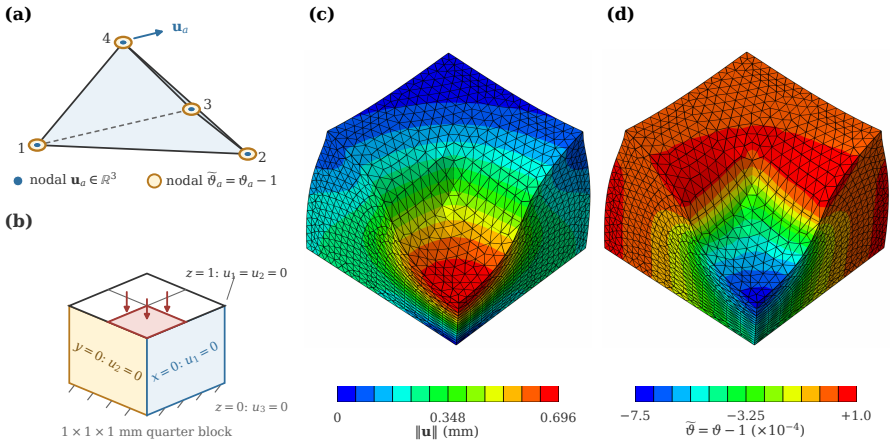

Figure 4: Generated mixed Tet4 applied to the block-compression benchmark of [40]. (a) Four nodes, each carrying three displacement components and $\widetilde{\vartheta} = \vartheta - 1$. (b) The $n = 16$ quarter block with a 0.5×0.5 mm follower-pressure patch ($p_{\text{top}} = 320$ N/mm^2). (c,d) Final displacement magnitude and $\widetilde{\vartheta}$ on the deformed mesh, each panel with its native contour range.

finite differences. The Abaqus/Standard block simulation uses 21,361 generated mixed tetrahedra on a Gmsh mesh with target spacing 1/16 mm. At full follower pressure, its center displacement is $u_3 = -0.696244$ mm, within about 1% of the approximately -0.70 mm value read from the published $n = 16$ unstructured-mesh curve in Figure 12 of [40]. These checks connect element-level verification to a three-dimensional Abaqus boundary-value problem.

3.3. Mixed-order gel element

The gel examples use a pressure reparameterization of the model used by Tao et al. [41], drawing on established coupled gel theory and finite-element formulations [1–4, 23]. Here, $J^e = \exp(p/K)$ introduces a pressure that is treated as either a global field or an element-local variable. The mixed-order element in this section and the condensed element in Sec. 3.4 use the same

balances,

$$\operatorname{Div}\mathbf{P} = \mathbf{0}, \qquad r_p(\mathbf{F}, p, \mu) = 0, \tag{8a}$$

$$\dot{c}_R + \operatorname{Div}\mathbf{j}_R = 0, \tag{8b}$$

$$\mathbf{P} = G(\mathbf{F} - \mathbf{F}^{-\mathsf{T}}) + p\mathbf{F}^{-\mathsf{T}}, \tag{8c}$$

$$r_p = \mu - \mu_0 - RT[\ln(1-\phi) + \phi + \chi\phi^2] + (\phi/\phi_0)\Omega p, \tag{8d}$$

$$\mathbf{j}_R = -M\mathbf{C}^{-1}\operatorname{Grad}\mu, \tag{8e}$$

$$J^e = \exp(p/K), \qquad \phi = \phi_0 J^e/J, \tag{8f}$$

$$c_R = \frac{1-\phi_0}{\Omega} + \frac{\phi_0 - \phi}{\Omega\phi}. \tag{8g}$$

Here G and K are the gel shear and bulk moduli, D is diffusivity, R is the gas constant, T is absolute temperature, Ω is solvent molar volume, χ is the interaction parameter, and μ_0 is the reference chemical potential. Further, J^e is the elastic volume ratio of the polymer network and ϕ is the polymer volume fraction. The corrosion phase field of Sec. 3.1 is not used in this section. The kinematics and the pressure term in r_p derive from the swelling split $J_s = J/J^e = \phi_0 + \Omega c_R$ and the network energy $U(J^e) = \frac{K}{2}(\ln J^e)^2$, as derived in the Supplementary Material. The quantities c_R and $\mathbf{j}_R$ are the referential solvent concentration and flux. The mobility is $M = Dc_R/(RT)$, and r_p is a chemo-elastic constraint containing the Flory–Huggins mixing term. Backward Euler gives $\dot{c}_R \approx [(J/J^e)_{n+1} - (J/J^e)_n]/(\Omega\Delta t)$. The global u–p–μ element enforces $\int_{\Omega_e} q r_p \, dV = 0$ for the pressure test function q.

The mixed-order construction solves three global fields in the order u–p–μ. Displacement and chemical potential use the all-node quadratic-serendipity Quad8 interpolation, while pressure is bilinear and supported only on the four corner nodes. Mixed-order interpolation is classical [42, 43], but similarity to a familiar pair does not establish stability for this three-field finite-strain problem, and a pressure-mode or inf–sup study remains open.

The field choice creates a node-dependent Abaqus map,

$$\begin{aligned} &\text{corner nodes:} && (u_1, u_2, p, \mu) && \mapsto (1, 2, 11, 12), \\ &\text{midside nodes:} && (u_1, u_2, \mu) && \mapsto (1, 2, 12), \end{aligned} \tag{9}$$

which appears verbatim in the generated deck declaration,

```
*User Element, Type=U1, Nodes=8, Coordinates=2,
               Properties=9, Variables=9, Unsymm
1, 2, 11, 12
5, 1, 2, 12
```

The declaration drives interpolation, packing, tangent assembly, and the deck scaffold, and the 28-DOF element passes a two-element chemical-potential-gradient test in Abaqus/Standard.

The demonstration uses a 50-mm-long plane-strain half-bilayer of a 2.5-mm gel layer under a 2.5-mm rubber layer, following the swelling-induced bending example of Chester, Di Leo, and Anand [4]. Conforming meshes share displacement nodes at the naturally impermeable interface. The top-surface gel DOF 12 approaches its swollen value exponentially with a 300 s time constant over the 6 h analysis. The rubber uses quadratic hybrid `CPE8H` elements with $\nu = 0.499$. A negligible-stiffness `CPE8R` copy sharing the UEL nodes ($E = 10^{-20}$ Pa) makes the gel visible, and the colored contours in Fig. 5 are the `LE` output of the physical rubber.

The analysis completed 2160 increments to 21600 s with a final right-edge straightness residual of 2.3×10^{-9} m. The midside-node slider constraint that enforces this straightness, and its sensitivity, are detailed in the Supplementary Material.

Figure 5 thus demonstrates execution and the prescribed boundary kinematics, not quantitative reproduction. Pressure is now global, the interpolation orders have changed, and the rubber is slightly compressible. Response comparison and mixed-field stability remain open.

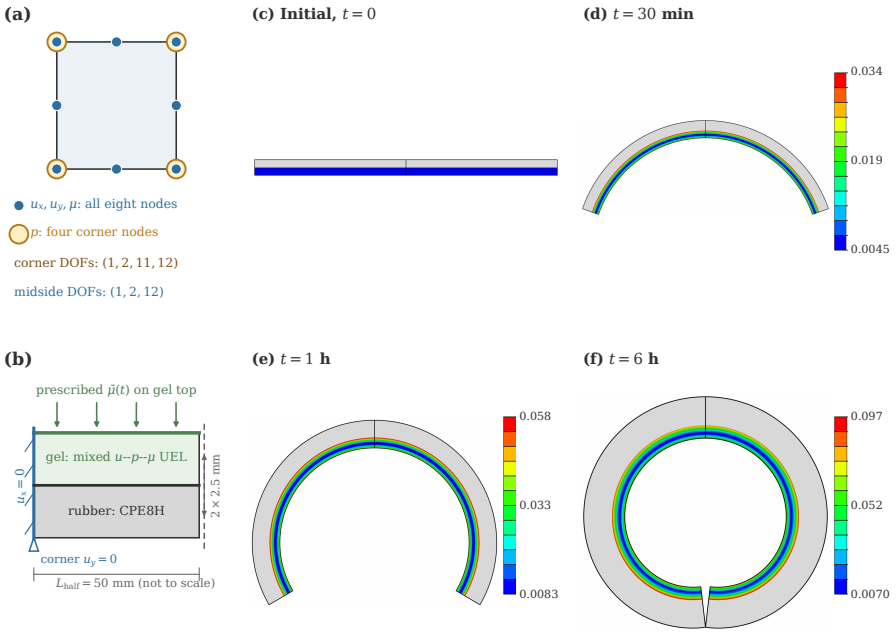

Figure 5: Plane-strain mixed-order swelling of a gel–rubber bilayer. (a) Node-dependent DOF maps: displacement and chemical potential on all eight nodes, bilinear pressure on the corners. (b) Chemical-potential loading, symmetry, a straight-slider right edge meeting hard contact, and an impermeable interface. (c–f) Maximum in-plane principal logarithmic strain of the rubber at $t = 0$, 30 min, 1 h, and 6 h, reflected about the symmetry edge. The gel appears gray through its visualization overlay, and each nonzero panel keeps its own range. The snapshots demonstrate execution, not quantitative reproduction.

3.4. Three-dimensional diffusion-driven morphing

The final example considers the grooved-sheet geometry and solvent-exposure protocol of Tao et al. [41]. Surface grooves create nonuniform uptake through the sheet, and differential swelling drives a three-dimensional change of shape. The simulation solves the balances in Eq. (8) with the Hex8 element-local pressure route of Sec. 2.3, using the full $2 \times 2 \times 2$ integration rule for the element blocks and the pressure constraint. It is an execution demonstration, not a quantitative reproduction of the experiments in [41].

The pressure-based Hex8 source regenerates and compiles, and a one-element `feacheap` solve gives finite polymer fraction and nonzero condensed pressure. A black-box finite-difference test of the compiled element, in which the local pressure is re-solved after every global perturbation, matches the returned condensed tangent to a relative error below 10^{-9} for both the Quad4 and this Hex8 element, verifying the complete condensed Jacobian of Eq. (3) including the implicit pressure sensitivity (Supplementary Material). Figure 6 shows the computed morphing sequence, with the polymer volume fraction evolving toward the swollen state.

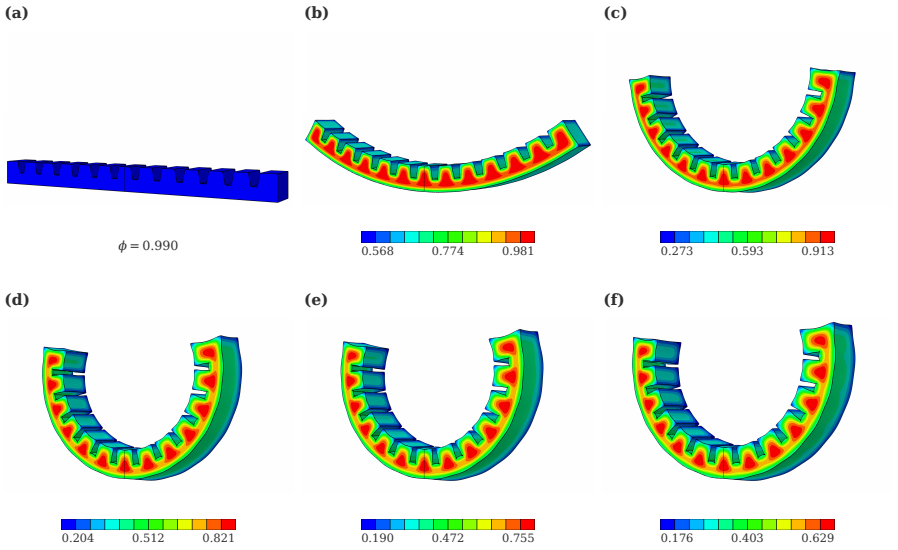

Figure 6: Diffusion-driven morphing with the generated Hex8 local-pressure UEL. Panels (a–f): polymer volume fraction ϕ at six increasing exposure times of the 0–360 s protocol, rendered through the element's visualization mesh. Each panel keeps its own contour range, so colors are not comparable between panels. The sequence demonstrates execution of the condensed formulation.

4. Discussion and conclusion

`abaqus_ufl` connects a compact weak form to the arrays and field maps required by an Abaqus UEL. The four examples isolate selective tangent blocks, weak-form stabilization, node-dependent mixed interpolation, and element-local condensation, and together they show why governing equations alone do not determine a nonlinear UEL. The user chooses the iteration matrix, stabilization, interpolation, and global or local variables, and the framework translates that route into a single inspectable Fortran source.

Complex-step differentiation avoids hand derivatives on complex-safe paths, but it neither verifies the equations nor passes reliably through nonanalytic operations, active-set changes, or unconverged local solves. Correctness still depends on signs, interpolation, history, and Abaqus field maps, and no speed advantage is claimed over optimized analytic tangents. The four demonstrations reach different levels of the verification ladder. For the bilayer, the evidence is limited to generated-code execution and verified boundary kinematics, and quantitative reproduction and mixed-field stability remain open.

The repository accompanying this paper packages the workflow rather than only its outputs: the generators and their templates, a test suite that exercises generated code directly, and a documented example pipeline in which every example must regenerate deterministically, compile, and execute through a compiled or checked runtime. Development runs as a closed loop of declare, generate, verify, and revise, and the recorded evidence of the staged gates keeps each translation step auditable. The framework does not simplify the coupled physics or the formulation choices. It centralizes the repeated tasks of field mapping, derivative construction, array packing, source generation, and verification, so a coupled formulation built from the supported templates enters Abaqus as a short declaration whose path to a production simulation is a sequence of explicit, testable steps.

CRediT authorship contribution statement

Teng Zhang: Conceptualization, Methodology, Software, Validation, Investigation, Visualization, Writing – original draft, Writing – review and editing, Funding acquisition.

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The author thanks Professor Lallit Anand at the Massachusetts Institute of Technology for many inspiring discussions on multiphysics modeling and simulation; Professor Allan Bower at Brown University for his course on computational solid and structural mechanics and for sharing the `feacheap` code; and Professor Kenichi Soga and Yaobin Yang at the University of California, Berkeley, for helpful discussions. The author acknowledges support from the U.S. National Science Foundation (NSF) under Grants CMMI-1847149, OIA-2428643, and CMMI-2517722. All Abaqus simulations reported in this work were performed on the Expanse system at the San Diego Supercomputer Center. Computational support was also provided through the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS), including CloudBank, supported by NSF Awards No. 1925001 and No. 2505560, and Jetstream2, supported by NSF Grant No. 2005506.

Data availability

The `abaqus_ufl` source code and the four paper-example reproduction packages (declarations, generation pipelines, generated subroutines, input decks, reduced reference data, and figure kits, under `paper_examples/`) will be made

publicly available in one tagged, DOI-archived repository at github.com/tengzhang48/abaqus_ufl.

Declaration of generative AI and AI-assisted technologies in the writing process

During preparation of this work, the author used generative AI assistants (Anthropic's Claude, OpenAI's ChatGPT, Google's Gemini, Moonshot's Kimi, DeepSeek, Qwen, and Zhipu's GLM) for discussion of theory, code development and testing, cross-checking derivations and numerical results, and language editing. The author verified the reported results, reviewed and edited the manuscript, and takes full responsibility for its content.

References

[1] W. Hong, X. Zhao, J. Zhou, Z. Suo, A theory of coupled diffusion and large deformation in polymeric gels, Journal of the Mechanics and Physics of Solids 56 (5) (2008) 1779–1793. doi:10.1016/j.jmps.2007.11.010.

[2] W. Hong, Z. Liu, Z. Suo, Inhomogeneous swelling of a gel in equilibrium with a solvent and mechanical load, International Journal of Solids and Structures 46 (17) (2009) 3282–3289. doi:10.1016/j.ijsolstr.2009.04.022.

[3] M. K. Kang, R. Huang, A variational approach and finite element implementation for swelling of polymeric hydrogels under geometric constraints, Journal of Applied Mechanics 77 (6) (2010) 061004. doi:10.1115/1.4001715.

[4] S. A. Chester, C. V. Di Leo, L. Anand, A finite element implementation of a coupled diffusion-deformation theory for elastomeric gels, International Journal of Solids and Structures 52 (2015) 1–18. doi:10.1016/j.ijsolstr.2014.08.015.

[5] C. Cui, R. Ma, E. Martínez-Pañeda, A phase field formulation for dissolution-driven stress corrosion cracking, Journal of the Mechanics and Physics of Solids 147 (2021) 104254. doi:10.1016/j.jmps.2020.104254.

[6] S. M. Mousavi, J. Mulderrig, B. Talamini, N. Bouklas, A chain stretch-based gradient-enhanced model for damage and fracture in elastomers, Computer Methods in Applied Mechanics and Engineering 444 (2025) 118103. doi:10.1016/j.cma.2025.118103.

[7] R. Zhao, Y. Kim, S. A. Chester, P. Sharma, X. Zhao, Mechanics of hard-magnetic soft materials, Journal of the Mechanics and Physics of Solids 124 (2019) 244–263. doi:10.1016/j.jmps.2018.10.008.

[8] C. Dorn, L. Bodelot, K. Danas, Experiments and numerical implementation of a boundary value problem involving a magnetorheological elastomer layer subjected to a nonuniform magnetic field, Journal of Applied Mechanics 88 (7) (2021) 071004. doi:10.1115/1.4050534.

[9] M. Rambausek, D. Mukherjee, K. Danas, A computational framework for magnetically hard and soft viscoelastic magnetorheological elastomers, Computer Methods in Applied Mechanics and Engineering 391 (2022) 114500. doi:10.1016/j.cma.2021.114500.

[10] H. S. Park, T. D. Nguyen, Viscoelastic effects on electromechanical instabilities in dielectric elastomers, Soft Matter 9 (4) (2013) 1031–1042. doi:10.1039/C2SM27375F.

[11] A. S. Kuenstler, Y. Chen, P. Bui, H. Kim, A. DeSimone, L. Jin, R. C. Hayward, Blueprinting photothermal shape-morphing of liquid crystal elastomers, Advanced Materials 32 (2020) 2000609. doi:10.1002/adma.202000609.

[12] Y. Jiang, Z. Xu, R. Xiao, S. Govindjee, T. D. Nguyen, A viscoelastic micro-stretch theory for monodomain nematic liquid crystal elastomers, Journal

of the Mechanics and Physics of Solids 207 (2026) 106412. doi:10.1016/j.jmps.2025.106412.

[13] M. S. Ukidwe, A. K. Datta, C. Koh, S. Tibos, J. Bows, Coupled transport and poromechanics model to understand quality evolution during sequential drying, Chemical Engineering Science 280 (2023) 119010. doi:10.1016/j.ces.2023.119010.

[14] R. Xu, Y. Yang, F. Yin, P. Liu, P. Cloetens, Y. Liu, F. Lin, K. Zhao, Heterogeneous damage in Li-ion batteries: Experimental analysis and theoretical modeling, Journal of the Mechanics and Physics of Solids 129 (2019) 160–183. doi:10.1016/j.jmps.2019.05.003.

[15] L. S. de Vasconcelos, R. Xu, Z. Xu, J. Zhang, N. Sharma, S. R. Shah, J. Han, X. He, X. Wu, H. Sun, S. Hu, M. Perrin, X. Wang, Y. Liu, F. Lin, Y. Cui, K. Zhao, Chemomechanics of rechargeable batteries: Status, theories, and perspectives, Chemical Reviews 122 (15) (2022) 13043–13107. doi:10.1021/acs.chemrev.2c00002.

[16] D. Xue, C. Fincher, R. Fang, B. W. Sheldon, L.-Q. Chen, S. Zhang, Dynamic interplay of dendrite growth and cracking in lithium metal solid-state batteries, Journal of the Mechanics and Physics of Solids 202 (2025) 106197. doi:10.1016/j.jmps.2025.106197.

[17] P. Li, G. Lv, W. Li, H. Fan, Q. Wang, K. Zhou, Fully coupled electro-chemo-thermo-mechanical phase-field fracture modeling for solid-state batteries, International Journal of Solids and Structures 328 (2026) 113831. doi:10.1016/j.ijsolstr.2026.113831.

[18] Dassault Systèmes Simulia Corp., Abaqus analysis user's guide, Dassault Systèmes, version 2022 (2022).

[19] C. Miehe, M. Hofacker, F. Welschinger, A phase field model for rate-independent crack propagation: Robust algorithmic implementation based

on operator splits, Computer Methods in Applied Mechanics and Engineering 199 (45–48) (2010) 2765–2778. doi:10.1016/j.cma.2010.04.011.

[20] G. Molnár, A. Gravouil, 2D and 3D Abaqus implementation of a robust staggered phase-field solution for modeling brittle fracture, Finite Elements in Analysis and Design 130 (2017) 27–38. doi:10.1016/j.finel.2017.03.002.

[21] Y. Navidtehrani, C. Betegón, E. Martínez-Pañeda, A unified Abaqus implementation of the phase field fracture method using only a user material subroutine, Materials 14 (8) (2021) 1913. doi:10.3390/ma14081913.

[22] X. Zhou, Y. Zhang, Implementation and verification of a user-defined element (UEL) for coupled thermal–hydraulic–mechanical–chemical (THMC) processes in saturated geological media, International Journal for Numerical and Analytical Methods in Geomechanics 47 (11) (2023) 2153–2190. doi:10.1002/nag.3556.

[23] B. Datta, T. D. Nguyen, A finite element model and Abaqus user element (UEL) implementation of hydrogel chemo-mechanics, Zenodo, version 1.0.0 (2025). doi:10.5281/zenodo.15725221.

[24] L. Quinteros, E. García-Macías, E. Martínez-Pañeda, Electromechanical phase-field fracture modelling of piezoresistive CNT-based composites, Computer Methods in Applied Mechanics and Engineering 407 (2023) 115941. doi:10.1016/j.cma.2023.115941.

[25] J. Korelc, P. Wriggers, Automation of Finite Element Methods, Springer, 2016. doi:10.1007/978-3-319-39005-5.

[26] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, G. N. Wells, Unified form language: A domain-specific language for weak formulations of partial differential equations, ACM Transactions on Mathematical Software 40 (2) (2014) 1–37. doi:10.1145/2566630.

[27] A. Logg, K.-A. Mardal, G. N. Wells (Eds.), Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book, Vol. 84 of Lecture Notes in Computational Science and Engineering, Springer, 2012. doi:10.1007/978-3-642-23099-8.

[28] I. A. Baratta, J. P. Dean, J. S. Dokken, M. Habera, J. S. Hale, C. N. Richardson, M. E. Rognes, M. W. Scroggs, N. Sime, G. N. Wells, DOLFINx: The next generation FEniCS problem solving environment, Zenodo preprint (2023). doi:10.5281/zenodo.10447666.

[29] F. Rathgeber, D. A. Ham, L. Mitchell, M. Lange, F. Luporini, A. T. T. McRae, G.-T. Bercea, G. R. Markall, P. H. J. Kelly, Firedrake: Automating the finite element method by composing abstractions, ACM Transactions on Mathematical Software 43 (3) (2016) 1–27. doi:10.1145/2998441.

[30] C. J. Permann, D. R. Gaston, D. Andrš, R. W. Carlsen, F. Kong, A. D. Lindsay, J. M. Miller, J. W. Peterson, A. E. Slaughter, R. H. Stogner, R. C. Martineau, MOOSE: Enabling massively parallel multiphysics simulation, SoftwareX 11 (2020) 100430. doi:10.1016/j.softx.2020.100430.

[31] COMSOL AB, COMSOL Multiphysics, https://www.comsol.com, version 6.4 (2025).

[32] T. Helfer, B. Michel, J.-M. Proix, M. Salvo, J. Sercombe, M. Casella, Introducing the open-source mfront code generator: Application to mechanical behaviours and material knowledge management within the PLEIADES fuel element modelling platform, Computers & Mathematics with Applications 70 (5) (2015) 994–1023. doi:10.1016/j.camwa.2015.06.027.

[33] J. N. Lyness, C. B. Moler, Numerical differentiation of analytic functions, SIAM Journal on Numerical Analysis 4 (2) (1967) 202–210. doi:10.1137/0704019.

[34] W. Squire, G. Trapp, Using complex variables to estimate derivatives

of real functions, SIAM Review 40 (1) (1998) 110–112. doi:10.1137/S003614459631241X.

[35] J. R. R. A. Martins, P. Sturdza, J. J. Alonso, The complex-step derivative approximation, ACM Transactions on Mathematical Software 29 (3) (2003) 245–262. doi:10.1145/838250.838251.

[36] R. Kiran, K. Khandelwal, Complex step derivative approximation for numerical evaluation of tangent moduli, Computers & Structures 140 (2014) 1–13. doi:10.1016/j.compstruc.2014.04.009.

[37] E. A. de Souza Neto, D. Perić, M. Dutko, D. R. J. Owen, Design of simple low order finite elements for large strain analysis of nearly incompressible solids, International Journal of Solids and Structures 33 (20–22) (1996) 3277–3296. doi:10.1016/0020-7683(95)00259-6.

[38] P. Peterson, F2PY: A tool for connecting Fortran and Python programs, International Journal of Computational Science and Engineering 4 (4) (2009) 296–305. doi:10.1504/IJCSE.2009.029165.

[39] A. F. Bower, Applied Mechanics of Solids, CRC Press, Boca Raton, 2009. doi:10.1201/9781439802489.

[40] G. Scovazzi, R. Zorrilla, R. Rossi, A kinematically stabilized linear tetrahedral finite element for compressible and nearly incompressible finite elasticity, Computer Methods in Applied Mechanics and Engineering 412 (2023) 116076. doi:10.1016/j.cma.2023.116076.

[41] Y. Tao, Y.-C. Lee, H. Liu, X. Zhang, J. Cui, C. Mondoa, M. Babaei, J. Santillan, G. Wang, D. Luo, D. Liu, H. Yang, Y. Do, L. Sun, W. Wang, T. Zhang, L. Yao, Morphing pasta and beyond, Science Advances 7 (19) (2021) eabf4098. doi:10.1126/sciadv.abf4098.

[42] C. Taylor, P. Hood, A numerical solution of the Navier–Stokes equations using the finite element technique, Computers & Fluids 1 (1) (1973) 73–100. doi:10.1016/0045-7930(73)90027-3.

[43] J. C. Simo, R. L. Taylor, Quasi-incompressible finite elasticity in principal stretches. continuum basis and numerical algorithms, Computer Methods in Applied Mechanics and Engineering 85 (3) (1991) 273–310. doi:10.1016/0045-7825(91)90100-K.