AutoFreeFem: Automatic code generation with FreeFEM++ and LaTex output for shape and topology optimization of non-linear multi-physics problems

Grégoire Allaire, Michael H Gfrerer

AutoFreeFem: Automatic code generation with FreeFEM++ and LaTex output for shape and topology optimization of non-linear multi-physics problems

Grégoire Allaire[1] and Michael H. Gfrerer[1,2*]

[1]CMAP, UMR 7641, Ecole Polytechnique, Institut Polytechnique de Paris, Palaiseau, 91128, France.

[2]Institute of Applied Mechanics, Graz University of Technology, Technikerstraße 4, Graz, 8010, Styria, Austria.

*Corresponding author(s). E-mail(s): gfrerer@tugraz.at;
Contributing authors: gregoire.allaire@polytechnique.fr;

Abstract

For an educational purpose we develop the *Python* package *AutoFreeFem* which generates all ingredients for shape optimization with non-linear multi-physics in *FreeFEM++* and also outputs the expressions for use in LaTeX. As an input, the objective function and the weak form of the problem have to be specified only once. This ensures consistency between the simulation code and its documentation. In particular, *AutoFreeFem* provides the linearization of the state equation, the adjoint problem, the shape derivative, as well as a basic implementation of the level-set based mesh evolution method for shape optimization. For the computation of shape derivatives we utilize the mathematical Lagrangian approach for differentiating PDE-constrained shape functions. Differentiation is done symbolically using *SymPy*. In numerical experiments we verify the accuracy of the computed derivatives. Finally, we showcase the capabilities of *AutoFreeFem* by considering shape optimization of a non-linear diffusion problem, linear and non-linear elasticity problems, a thermo-elasticity problem and a fluid-structure interaction problem.

Keywords: code generation, FreeFEM++, shape derivative, shape optimization

1 Introduction

In order to solve challenging engineering problems, numerical simulation along with shape and topology optimization tools have become an integral part of the design process. Since the computation of linearizations and error-prone shape derivatives for non-linear multi-physics problems are involved, we have developed an educational tool for their automatic code generation. The developed tool outputs always two representations of each expression: one representation for producing a LaTeX documentation and one representation for writing a simulation and shape optimization script in *FreeFEM++*.

1.1 Principles and used software

AutoFreeFem is an open-source *Python* package and can be downloaded at https://gitlab.tugraz.at/autofreefem/autofreefem.

All example files, discussed in this paper, can be found on this repository. It builds on the symbolic calculation capabilities of the open-source *Python* library *SymPy* **?** (see Section 4.1 for details). For the numerical simulation and shape/topology optimization, the popular open-source software *FreeFEM++* **?** is utilized. *FreeFEM++* is designed for the efficient numerical solution of partial differential equations using the finite element method in both two and three dimensions. For the documentation of the problem (input equations, linearization, adjoint problem, shape derivative), the typesetting system LaTeX is used. Thus, the main philosophy of *AutoFreeFem* is to provide an implementation in a *FreeFEM++* script and a documentation in LaTeX from a single source and therefore allows the fast and reliable development of solutions to complicated problems. In order to illustrate this principle, we consider the following elementary example.

Example: divergence of a vector field

Consider the divergence of a vector field which is denoted by the symbol u. Table 1 gives the corresponding outputs for LaTeX and *FreeFEM++*, respectively. In order to distinguish a vector field from a scalar field, it is printed in bold font in LaTeX. On the other hand in *FreeFEM++*, we need to define components, *i.e.* u_x, u_y for a 2d problem and u_x, u_y, u_z in 3d. *AutoFreeFem* works with a versatile unified input that dynamically adapts to various use cases. In the current example, the input takes the form of

```
div(VectorField('u', ...)
```

Here, the class **div** (see Section 4.4) implements the divergence, whereas the class **VectorField** is used to define the vector field **u**. It is worth noting that the vector field, although not fully depicted here for simplicity, requires five input arguments, which give information on the domain of definition and the boundary conditions (see Section 4.2 for details).

Automatic simulation and shape optimization in FreeFEM++

The main class of *AutoFreeFem* is called **Lagrangian** (see Section 4.2). This class has in particular the two methods 'setup_simulation' and 'setup_optimization'. When the first method is called, *AutoFreeFem* first checks if the problem is linear or non-linear (see Section 3.1). In case of a linear problem, the output is a simple *FreeFEM++* script for the simulation of the problem. In case of a non-linear problem, the linearization is computed and used in a Newton's method implemented in the output *FreeFEM++* script. For simulations the input of the Lagrangian are the primary field variables, the corresponding test functions and a variational formulation of the problem (see Section 2.1). For shape optimization problems the method 'setup_optimization' has to be called. Then a linearization in case of a non-linear problem, an adjoint problem, and a shape derivative are computed. Furthermore, a corresponding *FreeFEM++* script is generated (see e.g. Section 2.2). For the numerical solution of shape optimization problems in *FreeFEM++* we employ the level-set based mesh evolution method **?**. To this end we use the aditional open-source libraries *mmg*[1] **?**, *mshdist*[2] **?**, and *advection*[3].

1.2 Relation to the literature

The automatic generation of simulation code and the automatic computation of shape derivatives have been considered in some previous works. As part of the FEniCS Project **?**, the Unified Form Language (UFL) offers a flexible interface for choosing finite element spaces and defining expressions for weak forms in a notation close to mathematical notation **?**. This allows also for the automatic computation of derivatives and therefore the easy treatment of non-linear problems. Based on UFL, the open-source library FEMorph is an automatic shape differentiation toolbox, which can compute first- and second-order shape derivatives **?**. It refactors UFL expressions and applies shape calculus differentiation rules recursively. In **?**, the UFL is extended to shape differentiation using a different strategy. The approach in **?** is based on pullbacks and standard Gateaux derivatives. Furthermore, automated shape derivatives for transient PDEs in FEniCS and Firedrake **?** are presented in **?**. This has been further developed in the software Fireshape **?**. Inspired by the

[1] http://www.mmgtools.org/
[2] https://github.com/ISCDtoolbox/Mshdist
[3] https://github.com/ISCDtoolbox/Advection

LaTeX	*FreeFEM++* (2D)	*FreeFEM++* (3D)
\operatorname{div} \mathbf{u}	dx(ux) + dy(uy)	dx(ux) + dy(uy) + dz(uz)

Table 1: LaTeX and *FreeFEM++* expressions for the divergence of a vector field

FEniCS Project, the finite element software package NGSolve **?** has a flexible interface to *Python*, which allows defining expressions for weak forms in a mathematical notation. In **?** it has been extended for the automatic computation of first- and second-order shape derivatives based on a Lagrangian function, pullbacks, and directional (Gateaux) derivatives. In **?**, NGSolve has been further developed to allow also for the automatic computation of topological derivatives. We mention also the software *cashocs* described in **??**, which offers automated solutions for shape optimization and optimal control.

All works mentioned so far are based on symbolic shape differentiation. In the context of density based methods for topology optimization, an automatic differentiation (AD) tool is presented in **?**. There, practically no difference in the timings of AD and symbolic sensitivities was found. For further references on AD we refer to **?**. In all these references the focus is on the automatic generation of the sensitivity information for use within the computational optimization routine. The main novelty of the present work is to consider, in a pedagogical perspective, the simultaneous generation of LaTeX expressions for the documentation and *FreeFEM++* expressions for numerical optimization.

1.3 Outline of the paper

The next section is a brief tutorial which features two examples: the simulation of a 3d nonlinear fluid flow and the 2d shape optimization of a linearly elastic structure. In Section 3 we present the underlying mathematical theory of *AutoFreeFem*. Section 4 details the implementation. Several numerical examples are discussed in Section 5. Finally, we draw some conclusions from the present work in Section 6.

2 Introductory examples

This section provides two hands-on introductory examples to *AutoFreeFem*. First, the simulation of a viscous fluid flowing through a pipe. Second, the compliance minimization of a cantilever beam.

2.1 3d simulation of a fluid flow

In this example, the fluid flow in a winding pipe is simulated by solving the incompressible Navier-Stokes equations with Taylor-Hood finite elements. In the next subsections, we explain step by step how to solve this problem using *AutoFreeFem*.

2.1.1 Step 1: 3d mesh generation with *FreeFEM++*

For the generation of the computational mesh we use built-in commands of *FreeFEM++* in the file "meshNS3d.txt" (which can be found on the *AutoFreeFem* repository). In particular, we use *border* and *buildmesh* to generate a disk, which is then extruded to a cylinder by *buildlayers*. The final mesh is obtained by a mesh distortion using the command *movemesh*. The chosen 3d geometry of a winding pipe is depicted in Figure 2a. Note that more complicated 3d meshes can be created with another mesh generator and loaded into *FreeFEM++*.

2.1.2 Step 2: Definition of the problem

The input for *AutoFreeFem* for this example is given in Listing 1. The first line imports all modules from the package *AutoFreeFem*. In the lines 4-6 the physical constants ρ (density) and μ (viscosity), as well as a penalty parameter γ are defined using the class **Constant**. Next, in line 10, we use the class **VectorField** to define the fluid velocity $\mathbf{v}$, which is discretized by finite elements of polynomial degree 2 (P2) on the domain *Th*. Furthermore, here Dirichlet boundary conditions (prescribed velocities) on boundaries with labels 1, 3 and 4 are also defined. In order to define the corresponding vector-valued inhomogeneous boundary data, we use the class **BoundaryFunction** in line 9. In line 11 we define a corresponding test function $\delta\mathbf{v}$ to the velocity field $\mathbf{v}$. In lines 13 and 14 the class **ScalarField** is used to define

the fluid pressure p discretized by P1 finite elements and the test function δp. A homogeneous pressure is prescribed at the boundary with label 2. In order to increase the readability in the LaTeX output, we introduce the viscous fluid stress $\boldsymbol{\sigma}_f$ as an **Expression** in lines 16 and 17 (see also Section 4.2). In lines 19-25 a **Lagrangian** object is set up. The first argument is a list of all primary field functions $(\mathbf{v}, p)$, whereas the second argument $(\delta\mathbf{v}, \delta p)$ is a list of all test functions. The third argument is the objective function, which has no meaning in case of a simulation. The fourth argument is the weak formulation of the governing equations. Those are the momentum balance and the continuity equation enhanced by a penalty term. For example, the input in line 20 represents the viscous stress term in the momentum equation and the corresponding LaTeX representation is

$$\int_\Omega (\boldsymbol{\sigma}_f(\mathbf{v}) : \nabla\delta\mathbf{v})\ dx. \tag{1}$$

Note, that the domain integral over Ω is realized by the class **dx** (see Section 4.5 ; not to be confused with the x-derivative notation in *FreeFEM++*), whereas **inner2** (see Section 4.3) implements the double dot product.

Finally, in line 26, we call the method *setup_simulation* of the Lagrangian and specify the file "meshNS3d.txt" for the definition of the finite element mesh.

```
1  from autofreefem import *
2
3  name = 'Navier-Stokes'
4  rho = Constant('\\rho', 1000., 'fluid density',
       'kg/m^3')
5  mu = Constant('\mu', 1.0e-3, 'fluid viscosity',
       'N s/m^2')
6  penalty = Constant('\\gamma', 1.e-9, 'penalty
       term', '1/(Pa s)')
7  Th = Domain('\\Omega','Th')
8  # fluid velocity
9  diri = BoundaryFunction( '0', '0', '5e-2*(r^2-x
       ^2-y^2)*(z<0.05)')
10 v = VectorField('v', 'P2', Th, diri, '1/3/4')
11 testv = VectorField('\delta v', 'P2', Th, '0.',
       'none')
12 # fluid pressure
13 p = ScalarField('p', 'P1', Th, '0.', '2')
14 q = ScalarField('\delta p', 'P1', Th, '0.', '
       none')
15 # viscous fluid stress tensor
16 d = (grad(v) + transpose(grad(v))) / 2
17 sigmaFluidViscous = Expression('\pmb \sigma_f',
       mu * d)
18 #
19 lag = Lagrangian([v, p], [testv, q], 0,
20     dx((inner2(sigmaFluidViscous, grad(testv))),
        Th)  # viscous stress
21     - dx(div(testv) * p, Th)   # pressure
22     + dx(rho * inner(inner(grad(v), v), testv),
       Th)  # convection
23     + dx(penalty * p * q, Th)  # penalty term
24     - dx(div(v) * q, Th), # continuity equation
25     dimensions=3)
26 lag.setup_simulation(name, mesh='meshNS3d.txt')
```

Listing 1: Input file for the Navier-Stokes example (run_navierStokes.py)

2.1.3 Step 3: LaTeX output

The simulation problem defined in Listing 1 is documented using LaTeX. The output is given in Figure 1. The output starts with the user input of the governing equations (here the momentum equation and the continuity equation). Then the abbreviations used, *i.e.* all objects of class **Expression**, are defined (here the viscous fluid stress $\boldsymbol{\sigma}_f$). *AutoFreeFem* automatically detects that the problem is non-linear (due to the here non-linear convective term, see Section 3.1) and computes also a linearization for the use within a Newton method. Thus, the problem for the Newton update is given in the remainder of Figure 1. In addition, the numerical values of the considered physical parameters are supplied in the automatically generated Table 2.

fluid density	ρ	1.00e+03	kg/m^3
fluid viscosity	μ	1.00e-03	$Ns/(m^{2)}$
penalty term	γ	1.00e-09	$1/(Pas)$

Table 2: Automatically generated LaTeX documentation of numerical values of constants defined in Listing 1

2.1.4 Step 4: Simulation with *FreeFEM++*

In addition to the LaTeX output, *AutoFreeFem*, run in the simulation mode (method 'setup_simulation'), produces three ".edp" files:

- run_NavierStokes.edp
- NavierStokesResidual.edp
- NavierStokesNewton.edp

The file "run_NavierStokes.edp" implements a basic solver based on Newton's method for the simulation of the flow problem in *FreeFEM++*. To

The state $[\mathbf{v},\ p]$ is the solution of the non-linear problem

$$\int_\Omega \rho\left(\delta\mathbf{v}\cdot(\nabla\mathbf{v}\cdot\mathbf{v})\right)\,dx - \int_\Omega p\,\mathrm{div}\,\delta\mathbf{v}\,dx + \int_\Omega \left(\boldsymbol{\sigma}_f(\mathbf{v}) : \nabla\delta\mathbf{v}\right)\,dx = 0 \qquad \forall\ \delta\mathbf{v},$$

$$-\int_\Omega \delta p\,\mathrm{div}\,\mathbf{v}\,dx + \int_\Omega \gamma\delta p p\,dx = 0 \qquad \forall\ \delta p,$$

with

$$\boldsymbol{\sigma}_f(\mathbf{v}) = \frac{\left(\nabla\mathbf{v} + \nabla\mathbf{v}^\top\right)\mu}{2}.$$

The Newton update $[\hat{\mathbf{v}},\ \hat{p}]$ at the state $[\mathbf{v},\ p]$ is the solution of

$$\begin{aligned}\int_\Omega \rho\left(\delta\mathbf{v}\cdot((\nabla\hat{\mathbf{v}}\cdot\mathbf{v}) + (\nabla\mathbf{v}\cdot\hat{\mathbf{v}}))\right)\,dx - \int_\Omega \hat{p}\,\mathrm{div}\,\delta\mathbf{v}\,dx + \int_\Omega \left(\partial_{(\mathbf{v},\hat{\mathbf{v}})}\boldsymbol{\sigma}_f(\hat{\mathbf{v}}) : \nabla\delta\mathbf{v}\right)\,dx \\ = -\int_\Omega \rho\left(\delta\mathbf{v}\cdot(\nabla\mathbf{v}\cdot\mathbf{v})\right)\,dx + \int_\Omega p\,\mathrm{div}\,\delta\mathbf{v}\,dx - \int_\Omega \left(\boldsymbol{\sigma}_f(\mathbf{v}) : \nabla\delta\mathbf{v}\right)\,dx \qquad \forall\ \delta\mathbf{v},\end{aligned}$$

$$-\int_\Omega \delta p\,\mathrm{div}\,\hat{\mathbf{v}}\,dx + \int_\Omega \gamma\delta p\hat{p}\,dx = \int_\Omega \delta p\,\mathrm{div}\,\mathbf{v}\,dx - \int_\Omega \gamma\delta p p\,dx \qquad \forall\ \delta p,$$

with

$$\partial_{(\mathbf{v},\hat{\mathbf{v}})}\boldsymbol{\sigma}_f(\hat{\mathbf{v}}) = \frac{\left(\nabla\hat{\mathbf{v}} + \nabla\hat{\mathbf{v}}^\top\right)\mu}{2}.$$

Fig. 1: Automatically generated LaTeX documentation of the problem formulation and linearization of the Navier-Stokes problem defined in Listing 1.

this end, it uses the expressions (varf's) defined in the other two files for evaluating the residual vector and the Jacobian matrix. The latter files can also be used as building blocks for more advanced solvers (*e.g.* preconditioned iterative solvers and/or domain decomposition methods) implemented by the user.

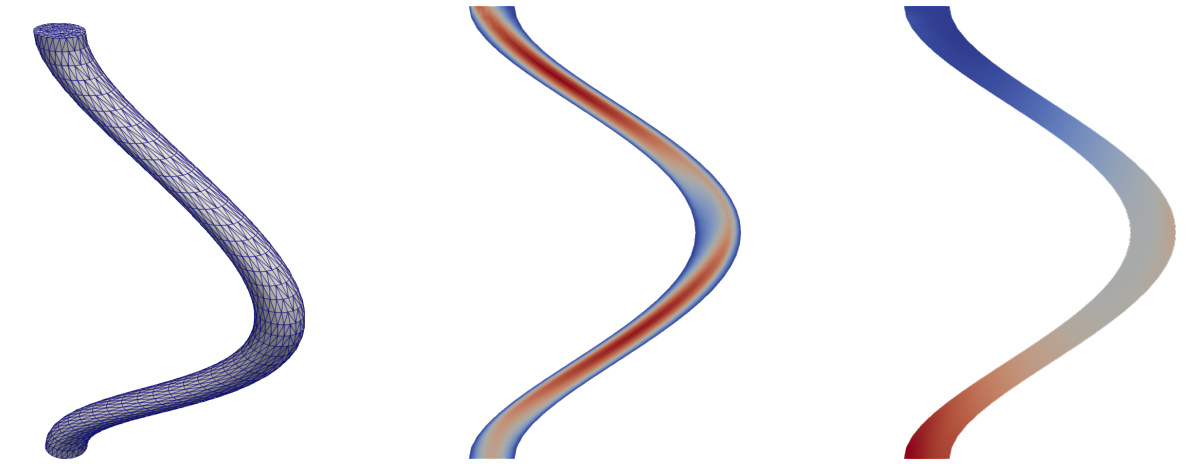
2.1.5 Step 5: Results

Running the file "run_NavierStokes.edp" with *FreeFEM++* produces a ".vtu" file with the simulation results. This file can be viewed with a 3D graphics program such as *paraview*[4]. The computed fluid velocity and pressure are depicted in Figure 2.

2.2 Shape optimization of a cantilever

Our second introductory example is a classical 2d compliance minimization for an elastic cantilever. The working domain is a rectangle of size 2×1 with zero displacement boundary condition on the left side and a vertical load applied on a small portion of length 0.1 at the middle of the right side, denoted by Γ_N, such that the resultant force has unit magnitude. All other boundaries are traction free. The geometry and the boundary conditions are illustrated in Figure 3. There are no body forces.

[4] https://www.paraview.org/



(a) (b) (c)

Fig. 2: Simulation of the fluid flow in a pipe: (a) 3d computational mesh; (b) Distribution of the norm of the fluid velocity over a vertical slice of the domain. Red corresponds to high velocity, blue corresponds to low velocity. On the lower surface the velocity distribution is prescribed as Dirichlet boundary condition. (c) Distribution of the (relative) fluid pressure over a vertical slice of the domain. Red corresponds to high pressure, blue corresponds to low pressure. The pressure on the upper surface is prescribed as Dirichlet boundary condition.

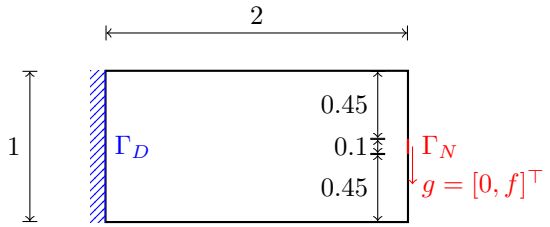


Fig. 3: Geometry and boundary conditions of the elastic cantilever.

2.2.1 Step 1: Mesh

Again, for the generation of the computational mesh, we use built-in commands of *FreeFEM++* in the file meshelasticCantilever.txt.

2.2.2 Step 2: Definition of the problem

```
from autofreefem import *
name = 'linear Elasticity'
bndN = Boundary('\\Gamma_N','ThL','2')
Th = Domain('D','Th',bndN)
# definition of the displacement field and the test function
u = VectorField('u', 'P2', Th, '0.', '4')  # left fixed
testu = VectorField('\delta u', 'P2', Th, '0.', 'none')
# definition of constants
f = Constant('f', -10, 'vertical load component', 'N/m')
nu = Constant('\\nu', .3, 'Poisson\'s ratio', '-')
E = Constant('E', 200., 'Young\'s modulus', 'N/m^2')
Afac = Constant('\ell', 0.25, 'Lagrange multiplier', '-' )
# definition of the material contrast
X = CharacteristicFunction('\\raisebox{\\depth}{$\\chi$}', 1, '1/100')
# definition of expressions
mu = Expression(sp.Symbol('\mu'), E/(2*(1+nu)))
lam = Expression(sp.Symbol('\lambda'), E*nu/((1-2*nu)*(1+nu)))
strain = Expression(sp.Symbol('\pmb \\varepsilon'), (grad(u) + transpose(grad(u))) / 2)
stress = Expression(sp.Symbol('\pmb \\sigma'), X*(lam*tr(strain)*identity() + 2 * mu*strain) )
# definition of the shape optimization problem
J = Expression('J', dx(inner2(stress,strain), Th) + dx(Afac*X, Th))
lag = Lagrangian([u], [testu], J, dx(inner2(stress, grad(testu)), Th) - dsx(f*inner(testu, ey()), Th, 0))
lag.setup_optimization(name, mesh='meshelasticCantilever.txt',
        hmin=0.01, hmax=0.02, diffusion=1./10000., v0=0.002, iterations=82,
        phi='-0.4 - sin(pi * kx * (x+0.5)) * cos(pi * ky * (y))',
        boundaryLabels=['1','4'], fixedLabels=['2'], show_weak_material=False)
```

Listing 2: Input file for the cantilever optimization (run_linearElasticity.py)

The *Python* input for *AutoFreeFem* is given in Listing 2. In lines 1-12 the displacement field, the physical constants and the used expressions are defined. These commands were already used in Section 2.1. Thus, we focus on the following lines which involve new aspects due to the considered shape optimization problem. In line 14 we introduce an object of the class **CharacteristicFunction** in order to distinguish between solid and void material. The first input is a LaTeX symbol (χ) for this function. The second argument represents the relative strength of the solid material (typically 1), whereas the third argument refers to the void material. Here, we use a factor of $1/100$ in order to mimic void by a very weak material. Strictly speaking, the object **CharacteristicFunction**, defined in line 14, is not the characteristic function of a set but rather a "color function", taking two different values (not necessarily 0 and 1) in two sub-domains. In lines 16 to 19 we use the class **Expression** to define the Lamé constants, the strain tensor and the stress tensor. In line 21 the objective function J is defined as a combination of the compliance and a fixed Lagrange multiplier ℓ multiplied by the area of the solid. Next, in line 22 the **Lagrangian** object is set up. The load on the boundary is incorporated in the problem by the corresponding boundary integral using the class **dsx** (see Section 4.5). Finally, in lines 22-26 we call the method *setup_optimization* of the Lagrangian and specify the file "meshelasticCantilever.txt" for the definition of the finite element mesh. Specifically for shape optimization, we also specify a minimal (*hmin*) and maximal (*hmax*) element size for remeshing with *mmg* **?**, a diffusion parameter for the regularization in the shape gradient identification problem **?**, an initial optimization velocity $v0$ used in the advection of the level-set function, a maximal iteration number and an initial level-set function ϕ defining the initial design. Furthermore, we set labels for boundaries where the normal component of the shape gradient should be set to zero (*boundaryLabels*), and labels for boundaries where the full gradient should vanish (*fixedLabels*). Finally, we set the postprocessing option *show˙weak˙material* such that the weak material is not shown in the ".vtu" outputs.

2.2.3 Step 3: LaTeX output

The problem formulation defined in Listing 2 is documented in Appendix A. *AutoFreeFem* automatically detects that the state problem is linear and therefore skips the statement of a superfluous linearization. Furthermore, the numerical values of the considered physical parameters are supplied in the automatically generated Table 3.

Lagrange multiplier	ℓ	0.25	$\frac{N}{m^2}$
Poisson's ratio	ν	0.3	–
Young's modulus	E	200	$\frac{N}{m^2}$
vertical load component	f	-10	$\frac{N}{m^2}$

Table 3: Automatically generated LaTeX documentation of numerical values of constants defined in Listing 2

2.2.4 Step 4: Simulation with *FreeFEM++*

In addition to the LaTeX output, *AutoFreeFem* in the shape optimization mode (method 'setup_optimization') produces six ".edp" files:

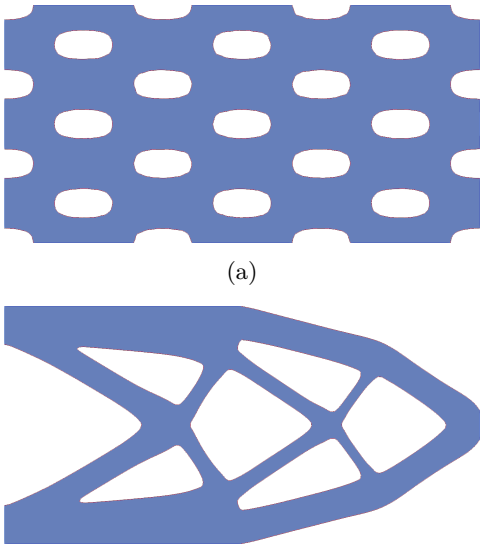
- optimize_linearElasticity.edp
- linearElasticityObjective.edp
- linearElasticityResidual.edp
- linearElasticityNewton.edp
- linearElasticityAdjoint.edp
- linearElasticityShape.edp

The file "optimize_linearElasticity.edp" implements a basic solver for the simulation of the cantilever and a basic variant of the level-set based mesh evolution method introduced in **??**. To this end, it uses the expressions (varf's) defined in the other files for evaluating the objective functional, the residual vector, the stiffness matrix, the resolution of the adjoint problem and finally the shape derivative.

2.2.5 Step 5: Results

Running the file "optimize_linearElasticity.edp" with *FreeFEM++* produces a sequence of 200 ".vtu"-files with the optimization results. The initialization and the optimized design are depicted

in Figure 4. The evolution of the objective function is reported in Figure 5.



(a)

(b)

Fig. 4: Linear elastic cantilever problem: (a) Initialization; (b) Optimized design.

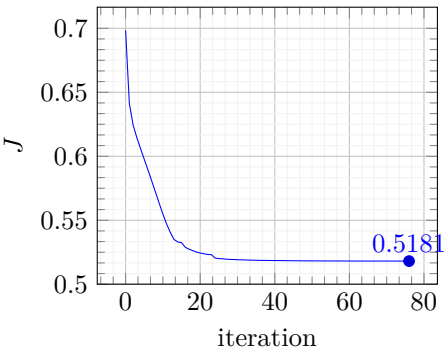


Fig. 5: Convergence history for the linear elastic cantilever problem

3 Theory on optimization problems involving non-linear multi-physics PDE constraints

In this section, we briefly describe the underlying mathematical theory of *AutoFreeFem*. In the present paper, we consider PDE-constrained shape optimization problems of the form:

$$\begin{aligned} &\min_{\Omega\in\mathcal{A}} J(\Omega,u) \\ &\quad \text{subject to } u\in V(\Omega): \\ &\quad R(\Omega,u,v)=0 \quad \text{for all } v\in V(\Omega). \end{aligned} \tag{2}$$

Here, $\mathcal{A}$ is the a set of admissible shapes, $J:\mathcal{A}\times V(\Omega)\to\mathbb{R}$ the objective function and the state u is the solution of the governing non-linear physics incorporated in $R(\Omega,u,v)$ posed on a Hilbert space $V(\Omega)$. For given shape $\Omega\in\mathcal{A}$ let $u(\Omega)$ be the unique solution of the state equation. This allows to introduce the shape functional $\mathcal{J}:\mathcal{A}\to\mathbb{R}$,

$$\mathcal{J}(\Omega)=J(\Omega,u(\Omega)). \tag{3}$$

In Section 3.1 and Section 3.2, we first focus on the classification and resolution of the possibly non-linear state equation $R(\Omega,u,v)=0$. Following this, we describe the theory on computing the shape derivative for single-physics problems in Section 3.3 and for multi-physics problems in Section 3.4.

3.1 Classifying the State Problem: Linear vs. Non-Linear

In a linear state problem, $R(\Omega,u,v)$ contains only terms that are either independent of the solution field u or linearly dependent on u. So we check whether the equation

$$\frac{d^2}{d\tau^2}R(\Omega,\tau u,v)=0 \tag{4}$$

is satisfied or not. Indeed, (4) holds for a linear state problem, but not for a nonlinear problem. In order to implement the automatic evaluation of (4) and subsequent expressions in this section, we use the *SymPy* commands *subs* and *diff* (see Section 4.1). For multi-physics problems

(see Section 3.4) condition (4) generalizes to

$$\frac{d^2}{d\tau^2} R_i(\Omega, \tau u_1, \ldots, \tau u_N, v_i) = 0, \text{ for } i = 1, \ldots, N.$$

3.2 Linearization

The solution of a non-linear state problem is typically computed by Newton's method. Thus, at a discrete level, we should provide the Jacobian matrix after the discretization of $R(\Omega, u, v)$. However, here we compute the expression of the Jacobian matrix at the continuous level. Thus, we follow a *first-differentiate-then-discretize* approach in the present paper, which might not give the same Jacobian matrix obtained by the *first-discretize-then-differentiate* approach. However, for linearizations both approaches typically give the same Jacobian matrix (a typical counterexample is that of plasticity problems, see **?**). Note, that for the shape derivative the result of *first-differentiate-then-discretize* is usually different from that of *first-discretize-then-differentiate*.

In order to illustrate the abstract setting in (2), consider a non-linear diffusion problem with homogeneous boundary conditions on a domain Ω.

Model 1. *Find the solution* $u \in H_0^1(\Omega)$ *such that* $R(\Omega, u, v) = 0$ *for all test functions* $v \in H_0^1(\Omega)$, *where*

$$\begin{aligned} R(\Omega, u, v) &= a(\Omega, u, v) - b(\Omega, v), \\ a(\Omega, u, v) &= \int_\Omega \lambda(\mathbf{X}, u(\mathbf{X}))(\nabla u \cdot \nabla v)\, \mathrm{d}\mathbf{X}, \\ b(\Omega, v) &= \int_\Omega f\, v\, \mathrm{d}\mathbf{X}, \end{aligned} \tag{5}$$

with a spatially varying diffusion coefficient λ, *which might depend on the state* u.

As usual $R(\Omega, u, v)$ is linear with respect to the test function v. In order to linearize $u \mapsto R(\Omega, u, v)$, we introduce its Fréchet derivative as the linear and bounded form $d_F R(\Omega, u, v)$, which satisfies

$$R(\Omega, u + \eta, v) = R(\Omega, u, v) + d_F R(\Omega, u, v) \cdot \eta + o(\eta), \quad \text{with} \quad \lim_{\eta \to 0} \frac{|o(\eta)|}{\|\eta\|} = 0. \tag{6}$$

Then, the linearization LR of R at state u_0 reads **?**

$$LR(\Omega, u_0, v, \eta) = R(\Omega, u_0, v) + d_F R(\Omega, u_0, v) \cdot \eta,$$

where LR is linear in the third and the last argument, but still non-linear in the first two arguments. In Newton's method, the update of a known state u_0 reads $u_0 \leftarrow u_0 + \eta$, where η is the solution to $LR(\Omega, u_0, v, \eta) = 0$, which has to hold for all test functions $v \in V$. In order to practically compute the linearization, it is more convenient to use the notion of Gateaux (or directional) derivative

$$dR(\Omega, u, v, \eta) = \frac{d}{d\tau} R(\Omega, u + \tau\eta, v)\bigg|_{\tau=0}. \tag{7}$$

If R is Fréchet differentiable, then it is Gateaux differentiable too and $d_F R = dR$. Note that for Model 1 we have

$$\begin{aligned} dR(\Omega, u_0, v, \eta) = &\int_\Omega \lambda(\mathbf{X}, u_0)(\nabla\eta \cdot \nabla v)\, \mathrm{d}\mathbf{X} \\ &+ \int_\Omega \frac{\partial\lambda}{\partial u}(\mathbf{X}, u_0)\eta(\nabla u_0 \cdot \nabla v)\, \mathrm{d}\mathbf{X}. \end{aligned}$$

3.3 Shape derivative of single-physics problems

In this section, we briefly review the Lagrangian method for computing the shape derivative of a PDE-constraint objective function. Mathematically rigorous treatments can be found in **???**; here we prefer a pedagogical presentation using notations from continuum mechanics. The notion of shape derivative relies on Hadamard method, which considers variations of the domain $\Omega \subset \mathbb{R}^d$ of the form

$$\Omega_t = (Id + t\mathbf{V})(\Omega) = T_t(\Omega), \tag{8}$$

where $\mathbf{V} : \Omega \to \mathbb{R}^d$ is a vector field and t a scalar perturbation parameter. In the following, we use the notation that $\mathbf{X}$ is a point in the unperturbed domain Ω and $\mathbf{x}$ a point in the perturbed domain Ω_t. Thus, a point $\mathbf{X} \in \Omega$ is therefore mapped to $\mathbf{x} \in \Omega_t$ by

$$\mathbf{x} = T_t(\mathbf{X}) = \mathbf{X} + t\mathbf{V}(\mathbf{X}), \tag{9}$$

and the Jacobian matrix $\mathbf{F}_t : \Omega \to \mathbb{R}^{d\times d}$ reads

$$\mathbf{F}_t(\mathbf{X}) = \nabla T_t(\mathbf{X}) = \mathbf{I} + t\nabla\mathbf{V}(\mathbf{X}), \tag{10}$$

where $\mathbf{I}$ denotes the identity matrix. From a continuum mechanics viewpoint, $\mathbf{X}$ is a Lagrangian coordinate, while $\mathbf{x}$ is an Eulerian coordinate, the vector field $\mathbf{V}$ can be interpreted as a displacement field and (10) defines the associated deformation gradient. For some shape functional $\mathcal{J}(\Omega)$, the shape derivative is defined as

$$DJ(\Omega)(\mathbf{V}) = \lim_{t\to 0} \frac{\mathcal{J}(\Omega_t) - \mathcal{J}(\Omega)}{t}. \tag{11}$$

Remark 1. *It is also possible to define the shape derivative as the Fréchet derivative of the mapping* $\mathbf{V} \mapsto \mathcal{J}((\mathbf{I} + \mathbf{V})(\Omega))$ *in* $\mathbf{V} = \mathbf{0}$ *?.*

In the following we introduce the Eulerian and Lagrangian states, which will be employed in the subsequent derivation of an efficient formula for the computation of the shape derivative (11). For a perturbed domain Ω_t we have the Eulerian state $u^{E,t} \in V(\Omega_t)$, which satisfies the state equation

$$R(\Omega_t, u^{E,t}, v) = 0 \quad \forall v \in V(\Omega_t). \tag{12}$$

In a next step we introduce the Lagrangian state $u^{L,t} \in V(\Omega)$ defined by the pull-back of the Eulerian state $u^{E,t} \in V(\Omega_t)$ to the unperturbed domain Ω

$$u^{L,t}(\mathbf{X}) = u^{E,t} \circ T_t(\mathbf{X}) = u^{E,t}(\mathbf{x}). \tag{13}$$

Conversely, this allows to write

$$u^{E,t}(\mathbf{x}) = u^{L,t} \circ T_t^{-1}(\mathbf{x}) = u^{L,t}(\mathbf{X}). \tag{14}$$

Obviously, for $t = 0$ in (9), we have $\mathbf{x} = \mathbf{X}$ and thus the Lagrangian and the Eulerian states coincide $u^{L,0} = u^{E,0} = u^0$.

In order to illustrate the difference between the Eulerian and Lagrangian frameworks we consider the residual equation of Model 1. The state equation determining $u^{E,t}$ is (12) with

$$R(\Omega_t, u^{E,t}, v) = \int_{\Omega_t} \lambda(\mathbf{x}, u^{E,t}(\mathbf{x}))(\nabla u^{E,t} \cdot \nabla v)\, \mathrm{d}\mathbf{x} - \int_{\Omega_t} f\, v\, \mathrm{d}\mathbf{x}.$$

A Lagrangian formulation is defined as

$$R^L(t, u^{L,t}, v^{L,t}) = R(\Omega_t, u^{L,t}\circ T_t^{-1}, v\circ T_t^{-1}). \tag{15}$$

Using standard rules of integral transformation and $u^{L,t}\circ T_t^{-1}\circ T_t = u^{L,t}$, (15) can be rewritten to

$$R^L(t, u^{L,t}, v^{L,t}) = \int_\Omega \lambda(T_t(\mathbf{X}), u^{L,t}) \left[\nabla(u^{L,t} \circ T_t^{-1}) \cdot \nabla(v^{L,t} \circ T_t^{-1})\right] \circ T_t \det \mathbf{F}_t\, \mathrm{d}\mathbf{X} - \int_\Omega f\, v^{L,t} \det \mathbf{F}_t\, \mathrm{d}\mathbf{X}. \tag{16}$$

Furthermore, using classical transformation rules for gradients (see Section 4.4) the fully Lagrangian setting avoiding the occurrence of T_t^{-1} reads:

$$R^L(t, u^{L,t}, v^{L,t}) = \int_\Omega \lambda(T_t(\mathbf{X}), u^{L,t}) \left(\nabla u^{L,t} \cdot (\mathbf{F}_t^{-1}(\mathbf{X}) \cdot \mathbf{F}_t^{-\top}(\mathbf{X})) \cdot \nabla v^{L,t} \right) \det \mathbf{F}_t \, \mathrm{d}\mathbf{X} - \int_\Omega f \, v^{L,t} \det \mathbf{F}_t \, \mathrm{d}\mathbf{X}. \tag{17}$$

This "pullback of the shape perturbation to the unperturbed domain" will be detailed for all implemented operators in Section 4 .

In order to compute the shape derivative, it is customary **?** to introduce a Lagrangian, in an Eulerian setting, by summing the objective function and the state equations

$$\mathcal{L}(\Omega_t, \varphi, \psi) = J(\Omega_t, \varphi) + R(\Omega_t, \varphi, \psi),$$

where $(\varphi, \psi) \in V(\Omega_t) \times V(\Omega_t)$ are any functions (in the end, φ will be replaced by the state $u(\Omega)$ and ψ by the adjoint state). The Lagrangian allows us to rewrite the numerator in (11) as

$$\mathcal{J}(\Omega_t) - \mathcal{J}(\Omega) = \mathcal{L}(\Omega_t, u^{E,t}, \psi^{E,t}) - \mathcal{L}(\Omega, u^0, \psi^0).$$

However, here the drawback is that $(u^{E,t}, \psi^{E,t}) \in V(\Omega_t) \times V(\Omega_t)$ and $(u^0, \psi^0) \in V(\Omega) \times V(\Omega)$, *i.e.* they are not defined over the same functional space. It turns out that this Eulerian setting is not easily amenable to automatic differentiation, contrary to the Lagrangian setting that we now introduce. Recalling the Lagrangian state (13) we define the Lagrangian $\mathcal{G} : \mathbb{R} \times V(\Omega) \times V(\Omega)$ in a Lagrangian setting by

$$\mathcal{G}(t, \varphi^{L,t}, \psi^{L,t}) = \mathcal{L}(T_t(\Omega), \varphi^{L,t} \circ T_t^{-1}, \psi^{L,t} \circ T_t^{-1}) \tag{18}$$

Now, we have

$$\mathcal{J}(\Omega_t) - \mathcal{J}(\Omega) = \mathcal{G}(t, u^{L,t}, \psi^{L,t}) - \mathcal{G}(0, u^0, \psi^0), \tag{19}$$

which has the advantage that $(u^{L,t}, \psi^{L,t})$ and (u^0, ψ^0) are both defined over the unperturbed domain Ω. Considering (19) in (11), and choosing a test function $\psi^{L,t}$ which is independent of t, yields by the chain rule

$$DJ(\Omega)(\mathbf{V}) = \partial_t \mathcal{G} + \partial_\phi \mathcal{G}[\dot{u}^L], \tag{20}$$

where

$$\partial_t \mathcal{G} = \left(\frac{\partial}{\partial t} \mathcal{G}(t, u^0, \psi^0) \right) \Big|_{t=0},$$
$$\partial_\phi \mathcal{G}[\dot{u}^L] = \left(\frac{d}{d\tau} \mathcal{G}(0, u^0 + \tau \dot{u}^L, \psi^0) \right) \Big|_{\tau=0}.$$

Here, $\dot{u}^L$ is the Lagrangian shape derivative (also called material derivative) of the state. Next we introduce the adjoint state p^0 with the goal to eliminate the Lagrangian shape derivative of the state. To this end, let p^0 be the solution of

$$\left(\frac{d}{d\tau} \mathcal{G}(0, u^0 + \tau v, p^0) \right) \Big|_{\tau=0} = 0 \quad \forall v \in V(\Omega). \tag{21}$$

Then, for $\psi^0 = p^0$ we have in particular $\partial_\phi \mathcal{G}[\dot{u}^L] = 0$, and (20) is reduced to

$$DJ(\Omega)(\mathbf{V}) = \left(\frac{\partial}{\partial t} \mathcal{G}(t, u^0, p^0) \right) \Big|_{t=0}. \tag{22}$$

As explained, e.g., in **?** this shape derivative formula is amenable to automatic differentiation.

3.4 Shape derivative of multi-physics problems

For our multi-physics applications, the objective functionals have the general structure

$$\Omega \mapsto \mathcal{J}(\Omega, u_1(\Omega), \ldots, u_N(\Omega)),$$

where the N scalar or vector-valued fields u_i, $i = 1, \ldots, N$ are the solutions of the respective governing equations $R_i(\Omega, u_1(\Omega), \ldots, u_N(\Omega), v_i) = 0$

for all $v_i \in V_i(\Omega)$. For a perturbed domain Ω_t the perturbed Eulerian states $u_i^{E,t} \in V_i(\Omega_t)$ satisfy

$$R_i(\Omega_t; u_1^{E,t}, \dots, u_N^{E,t}; v_i) = 0 \quad \forall v_i \in V_i(\Omega_t). \tag{23}$$

In accordance with (13) the Lagrangian states $u_i^{L,t} \in V_i(\Omega)$ are defined by

$$u_i^{L,t}(\mathbf{X}) = u_i^{E,t} \circ T_t(\mathbf{X}) = u_i^{E,t}(T_t(\mathbf{X})). \tag{24}$$

The Lagrangian is then defined by summing up the objective function and the state equations

$$\begin{aligned}
&\mathcal{L}(\Omega; \varphi_1, \dots, \varphi_N; \psi_1, \dots, \psi_N) = \mathcal{J}(\Omega; \varphi_1, \dots, \varphi_N) \\
&\mathcal{G}(t; \varphi_1^{L,t}, \dots, \varphi_N^{L,t}, \psi_1^{L,t}, \dots, \psi_N^{L,t}) = \mathcal{L}(T_t(\Omega); \varphi_1^{L,t} \circ T_t^{-1}, \dots, \varphi_N^{L,t} \circ T_t^{-1}; \psi_1^{L,t} \circ T_t^{-1}, \dots, \psi_N^{L,t} \circ T_t^{-1}) \\
&\qquad + \sum_{i=1}^{N} R_i(\Omega; \varphi_1, \dots, \varphi_N; \psi_i).
\end{aligned} \tag{25}$$

for any functions $\varphi_1, \dots, \varphi_N$ and $\psi_1, \dots, \psi_N$. Recalling (8), $\Omega_t = T_t(\Omega)$, the perturbed Lagrangian in a Lagrangian setting is defined by

With (25) the analogous arguments from Section 3.3 can be used to derive the shape derivative formula

$$DJ(\Omega)(\mathbf{V}) = \left(\frac{\partial}{\partial t} \mathcal{G}(t; u_1^0, \dots, u_N^0; p_1^0, \dots, p_N^0) \right) \Big|_{t=0}, \tag{26}$$

where the adjoint solutions $p_1^0, \dots, p_N^0$ are determined by

$$\left(\frac{d}{d\tau} \mathcal{G}(0; u_1^0, \dots, u_i^0 + \tau\, v_i, \dots, u_N^0; p_1^0, \dots, p_N^0) \right) \Big|_{\tau=0} = 0, \tag{27}$$

which have to hold for all test functions $v_i \in V_i(\Omega)$. For (26) to hold it is crucial that the adjoint solutions $p_1^0, \dots, p_N^0$ are determined by (27) in order to kill terms where the material derivative of the state variables show up.

4 Implementation

In this section, we describe some implementation details of *AutoFreeFem*. In view of the theory described in Section 3, the symbolic differentiation of expressions plays an important role. In particular, differentiation with respect to the perturbation parameter t of the Lagrangian $\mathcal{G}$ in (18), as well as the Gateaux derivative for the linearization (7) and the adjoint problem (27) have to be performed. Therefore, *AutoFreeFem* builds upon the *Python* package *SymPy* **?**. Beside the symbolic differentiation, the change of variables in (18) and the LaTeX processing uses and extends standard features of *SymPy*. We give a brief introduction into these topics in *SymPy* in Section 4.1. In Sections 4.2 to 4.7 we describe the implemented classes.

4.1 A brief introduction to differentiation, change of variables and LaTeX processing in *SymPy*

In *SymPy*, symbolic expressions are stored in expression trees. An expression tree is a data structure with a hierarchical form and the properties:

1. Each internal node represents an operator, *e.g.* addition, subtraction, multiplication, division, etc.
2. The operands (numbers and variables) are stored in the leaf nodes.
3. The edges between nodes indicate on which expressions the operators operate.

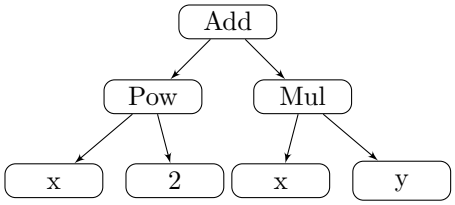
See Figure 6a for a visualization of the expression tree of *expr = x**2 + x*y* in *SymPy*.

Differentiation

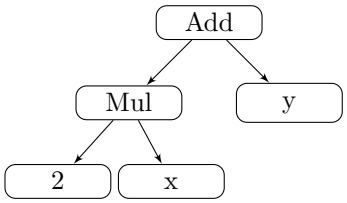
To differentiate this expression *expr* with respect to the variable x the *SymPy* command *diff(expr, x)* is used:

```
1 from sympy import *
```

```
x,y = symbols("x y")
expr = x**2 + x*y
dexpr = diff(expr,x)
print(srepr(expr))
print(srepr(dexpr))
```



(a) Expression tree for $x^2 + xy$



(b) Expression tree for $2x + y$

Fig. 6: Examples of expression trees

Here, the *SymPy* command *srepr* is used to asses the internal tree representation. The above code gives the outputs:

Add(Pow(Symbol('x'), Integer(2)), Mul(Symbol('x'), Symbol('y')))

Add(Mul(Integer(2), Symbol('x')), Symbol('y'))

They correspond to the expression trees in Figure 6 respectively. In order to realize the Gateaux derivative, we use twice the *SymPy* command *subs* and one time the *SymPy* command *diff*. The differentiation of *expr* with respect to x into the direction v is given by:

```
v,tau = symbols("v tau")
expr_vt = expr.subs(x,x+tau*v)
dexpr_vt = diff(expr_vt,tau)
dexpr_v = dexpr_vt.subs(tau,0)
print("expr_vt: ", expr_vt)
print("dexpr_vt: ", dexpr_vt)
print("dexpr_v: ", dexpr_v)
```

The above code gives the outputs:

expr_vt: y(tau*v + x) + (tau*v + x)**2*
*dexpr_vt: v*y + 2*v*(tau*v + x)*
*dexpr_v: 2*v*x + v*y*

The last output is the sought directional derivative $2vx + vy$. We remark that the result can be simplified by using the *SymPy* command *simplify*:

```
print("dexpr_v: ", simplify(dexpr_v))
```

dexpr_v: v(2*x + y)*

Change of variables

In addition to symbolic differentiation, we need to perform a change of variables to obtain the perturbed Lagrangian (18) in a Lagrangian framework. To obtain $\mathcal{G}(t; u_1^{L,t}, \ldots, u_N^{L,t}; p_1^{L,t}, \ldots, p_N^{L,t})$ in an automatic way, we traverse the expression tree and apply to each operator the corresponding change of variable rule such that T_t and T_t^{-1} cancel out. These rules are are non-trivial transformations for differential operators (see Section 4.4) and for integrals (see Section 4.5).

LaTeX and FreeFEM++ output

The special feature of *AutoFreeFem* is that it offers a LaTeX representation of the input and the derived formulas in coordinate independent direct notation and also a representation of them for the use in *FreeFEM++*. Remark that *SymPy* has several built-in options for the output of expressions like the basic string output, a LaTeX output, C code output, and Fortran code output:

```
print("String:", expr)
print("Latex:", latex(expr))
print("C code:", ccode(expr))
print("Fortran code:", fcode(expr))
```

The above code gives the outputs:

*String: x**2 + x*y*
Latex: x^2 + x y
*C code: pow(x, 2) + x*y*
*Fortran code: x**2 + x*y*

We are augmenting these built-in output capabilities in two directions. On the one hand, we are extending the LaTeX processing by introducing coordinate independent direct notation. On the other hand, we introduce the ability to generate code for use in *FreeFEM++*.

Due to the considerations above, each implemented field and operator is a subclass of the *SymPy* Function class and thus uses the same mechanisms as elementary functions in *SymPy*. Additionally, we specify for each class

1. a rule for the generation of LaTeX output,
2. a rule for the generation of *FreeFEM++* output,
3. if necessary the change of variables to obtain the perturbed Lagrangian functional using the Lagrangian states,
4. a rule for computing the derivative,
5. and if possible some algebraic rules to simplify the expressions.

In the following, we describe each implemented class of *AutoFreeFem*. In particular, we give a detailed explanation on how to implement Model 1 (see "nonlinearDiffusion.py" for the full file). To this end, we first import *AutoFreeFem*:

```
1 from autofreefem import *
```

4.2 Fields, Domain, Constants, Expressions and Lagrangian

In this section, two classes of unknown physical fields, such as temperatures, displacements or velocities, are introduced. These classes are summarized in Table 4. The **ScalarField** and the **VectorField** both take five input arguments. The first argument is a string representing the symbol of the unknown field and is used in the LaTeX output and the *FreeFEM++* code. The second argument contains the information how this field should be discretized in *FreeFEM++*. The third argument is the domain on which the field is defined. The fourth and the fifth argument are related to Dirichlet boundary conditions. In particular, the fourth argument specifies a function for the corresponding values of the Dirichlet boundary data. The fifth argument specifies the boundary labels on which Dirichlet boundary conditions are applied. Thus, for Model 1 we make the following definitions:

```
2 symbol = 'u'
3 fespace = 'P1'
4 mesh = Domain('\\Omega', 'Th')
5 dirichlet_boundary_function = '0.'
6 dirichlet_boundary_labels = '4'
7 u = ScalarField(symbol, fespace, mesh,
      dirichlet_boundary_function,
      dirichlet_boundary_labels)
```

Here we use conforming finite elements of polynomial degree 1 (P1) for the field u. Furthermore, we interoperate homogeneous Dirichlet boundary conditions on all boundaries with label 4. The definitions of the computational domain and of the corresponding mesh are done in the class **Domain**. The first argument (here '\\Omega') is the LaTeX expression, whereas within *FreeFEM++* code the second argument (here 'Th') will be used. We proceed by specifying the test function v:

```
8 v = ScalarField('v', fespace, mesh,
      dirichlet_boundary_function, 'none')
```

Note that for the test functions the boundary conditions are inherited form the corresponding unknown fields and therefore the fourth and the fifth argument on line 8 have no effect.

The chosen non-linearity is the diffusion coefficient $\lambda(u) = \lambda_0/(1+u^2)$:

```
9  Lambda0 = Constant('\\lambda_0', 10., 'diffusion
       coefficient','')
10 Lambda = Expression('\\lambda', Lambda0 / (1 + u
       ** 2))
```

In line 9 we first define an object of type **Constant**. It takes four arguments: a symbol, a numerical value, a description text and a string representing the unit. All constants will be automatically gathered and a LaTeX table will be generated for the documentation of the used numerical values (see *e.g.* Table 2 or Table 3). In line 10, we used the class **Expression**, which has mainly the purpose of introducing an abbreviation to achieve a nicely readable LaTeX output. It takes two arguments: a symbol and the actual expression.

Remark 2. *Note that the classes **Domain**, **Constant** and **Expression** are not necessary in order to set up a simulation in FreeFEM++ by AutoFreeFem. Their purpose is to provide the capability to generate nice LaTeX output.*

Next we define the bulk source term f using the class **Constant**:

```
11 f = Constant('f', 1., 'source term','')
```

For the definition of the variational formulation we rely on the classes **grad** (see Section 4.4), **inner** (see Section 4.3), and **dx** (see Section 4.5)

```
12 a = dx(Lambda*inner(grad(u),grad(v)), mesh)
13 b = dx(f*v, mesh)
```

In order to complete the implementation of Model 1, we set up an object of the class **Lagrangian** and call the method 'setup_simulation':

```
13 lag = Lagrangian([u], [v], 0, a - b )
14 lag.setup_simulation('non-linear diffusion')
```

The class **Lagrangian** has four input arguments. The first is a list of all trial fields representing the

description	*AutoFreeFem* input
unknown scalar	ScalarField(symbol, fe-space, mesh, b.c. function, b.c. label)
unknown vector	VectorField(symbol, fe-space, mesh, b.c. function, b.c. label)
domain	Domain(LaTeX symbol, *FreeFEM++* symbol, boundary 1, boundary 2, ...)
constant	Constant(symbol, numerical value, description, unit)
expression/abbriviation	Expression(symbol, formula)
class for computation	Lagrangian(trial fields, test fields, objective, PDE)

Table 4: Implemented classes described in Section 4.2

physical states in the problem, the second input argument is a list of all corresponding test fields. The third argument is the objective function and the last argument is the weak formulation of the PDE constraints.

4.3 Tensor algebra

An overview of the three implemented operators of tensor algebra is given in Table 5. Let $\{\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3\}$ be the standard Cartesian orthonormal basis. In the present paper, a tensor field $T(\mathbf{x})$ of order k assigns to every point $\mathbf{x}$ a tensor of the form $\underbrace{\mathbb{R}^d \otimes \ldots \otimes \mathbb{R}^d}_{\text{k copies}}$. In this way, we identify scalars as tensors of order zero, vectors as tensors of order one and matrices as tensors of order two. In *AutoFreeFem*, the tensor product of two tensors of arbitrary orders k and k', giving rise to a tensor of order $k+k'$, is realized by the class **TensorProduct**. Next, we define the dot product

operator	*AutoFreeFem* input	LaTeX
tensor product	TensorProduct(..., ...)	$(\ldots) \otimes (\ldots)$
dot product	inner(..., ...)	$(\ldots) \cdot (\ldots)$
double dot product	inner2(..., ...)	$(\ldots) : (\ldots)$

Table 5: Implemented operations from tensor algebra. All three operators take two tensor fields as inputs.

(class **inner**) of two tensors as the contraction of these tensors with respect to the last index of the first one, and the first index of the second one. For example, the dot product of a third order tensor $\mathbf{A} = A_{ijl}\mathbf{e}_i \otimes \mathbf{e}_j \otimes \mathbf{e}_l$ and a second order tensor $\mathbf{B} = B_{km}\mathbf{e}_k \otimes \mathbf{e}_m$ gives a third order tensor and reads

$$\begin{aligned}\mathbf{A} \cdot \mathbf{B} &= (A_{ijl}\mathbf{e}_i \otimes \mathbf{e}_j \otimes \mathbf{e}_l) \cdot (B_{km}\mathbf{e}_k \otimes \mathbf{e}_m) \\ &= A_{ijl}B_{lm}\mathbf{e}_i \otimes \mathbf{e}_j \otimes \mathbf{e}_m.\end{aligned}$$

Here, and in the following, the Einstein summation convention applies. Whenever an index occurs twice, we sum over this index, where Latin indices $i, j, \ldots$ take the values $1, 2, 3$. Furthermore, we define the double dot product (class **inner2**) of two tensors as the contraction of these tensors with respect to the last two indices of the first one, and the first two indices of the second one. The contraction is performed on the closest indices first, *e.g.*

$$\begin{aligned}\mathbf{A} : \mathbf{B} &= (A_{ijl}\mathbf{e}_i \otimes \mathbf{e}_j \otimes \mathbf{e}_l) : (B_{km}\mathbf{e}_k \otimes \mathbf{e}_m) \\ &= A_{ijl}B_{lj}\mathbf{e}_i.\end{aligned}$$

As a consequence of these definitions, we have for second order tensors $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, the relation $(\mathbf{A} \cdot \mathbf{B}) : \mathbf{C} = \mathbf{A} : (\mathbf{B} \cdot \mathbf{C})$. The operators in Table 5 commute with the pull back to the unperturbed domain. For the implementation it is also important to note that these operators obey the product rule of differentiation, *i.e.*

$$\begin{aligned}(\mathbf{A} \cdot \mathbf{B})' &= \mathbf{A}' \cdot \mathbf{B} + \mathbf{A} \cdot \mathbf{B}', \\ (\mathbf{A} : \mathbf{B})' &= \mathbf{A}' : \mathbf{B} + \mathbf{A} : \mathbf{B}', \\ (\mathbf{A} \otimes \mathbf{B})' &= \mathbf{A}' \otimes \mathbf{B} + \mathbf{A} \otimes \mathbf{B}',\end{aligned}$$

where $'$ denotes the derivation with respect to a scalar parameter τ.

4.4 Differential operators

An overview of the two implemented differential operators is given in Table 6. The gradient (class **grad**) of some scalar-valued function $f : \mathbb{R}^3 \to \mathbb{R}$

operator	*AutoFreeFem* input	LaTeX
gradient	grad(...)	$\nabla(\dots)$
divergence	div(...)	$\mathrm{div}(\dots)$

Table 6: Implemented differential operators

is defined as

$$\nabla f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial x_i}\mathbf{e}_i \tag{29}$$

with the Cartesian coordinates $\mathbf{x} = (x_1, x_2, x_3)$. We also use the generalization of the gradient for scalar-valued functions (29) to tensor fields. The gradient of a tensor field $\mathbf{A}$ of arbitrary order o is defined by

$$\nabla \mathbf{A}(\mathbf{x}) = \frac{\partial \mathbf{A}(\mathbf{x})}{\partial x_i} \otimes \mathbf{e}_i.$$

Note that $\nabla \mathbf{A}$ is a tensor of order $o+1$. For the gradient, the pullback of the shape perturbation can be obtained by application of the chain rule,

$$(\nabla_x \mathbf{A}(\mathbf{x})) \circ T_t(\mathbf{X}) = \nabla_X \mathbf{A}(T_t(\mathbf{X})) \cdot \mathbf{F}_t^{-1}(\mathbf{X}). \tag{30}$$

The second operator described in this section is the divergence (class **div**). For a tensor field $\mathbf{A}$ of order $o \geq 1$, it is given by

$$\mathrm{div}\, \mathbf{A} = \frac{\partial \mathbf{A}}{\partial x_i} \cdot \mathbf{e}_i = \nabla \mathbf{A}(\mathbf{x}) : \mathbf{I}. \tag{31}$$

Note, that $\mathrm{div}\, \mathbf{A}$ is a tensor of order $o-1$ and that the divergence is not defined for a scalar field. For the divergence the pull back of the shape perturbation is given by

$$\begin{aligned}(\mathrm{div}\, \mathbf{A}(\mathbf{x})) \circ T_t(\mathbf{X}) &= (\nabla_x \mathbf{A}(\mathbf{x})) \circ T_t(\mathbf{X}) : \mathbf{I} \\ &= (\nabla_X \mathbf{A}(T_t(\mathbf{X})) \cdot \mathbf{F}_t^{-1}(\mathbf{X})) : \mathbf{I}.\end{aligned}$$

4.5 Integrals and the normal vector

An overview of the implemented integral operators is given in Table 7. In *AutoFreeFem* domain integrals are understood as integrals over volumes (for 3d problems) or areas (for 2d problems) and are realized by the class **dx**. This class takes two arguments: the function to be integrated and the integration domain. For domain integrals of some

operator	*AutoFreeFem* input	LaTeX
domain integral	dx(f, domain)	$\int_\Omega f\, dx$
surface/line integral	dsx(f, domain, label)	$\int_\Gamma f\, ds_x$
normal vector	SurfaceNormalVector()	$\mathbf{n}$

Table 7: Implemented integral operators and the normal vector

tensor field $\mathbf{A}$, the pullback to the unperturbed domain reads

$$\int_{\Omega_t} \mathbf{A}(\mathbf{x})\, \mathrm{d}\mathbf{x} = \int_\Omega (\mathbf{A} \circ T_t(\mathbf{X}))\, \det \mathbf{F}_t(\mathbf{X})\, \mathrm{d}\mathbf{X}.$$

Boundary integrals (surfaces integrals for 3d problems, line integrals for 2d problems) are realized by the class **dsx**. Let Γ be part of the boundary of the domain Ω (characterized by some label). The pullback of an integral over the perturbed boundary $\Gamma_t = T_t(\Gamma)$ of some function f is given by

$$\int_{\Gamma_t} f(x)\, \mathrm{d}s_{\mathbf{x}} = \int_\Gamma f(T_t(\mathbf{X})) J_\Gamma(\mathbf{X})\, \mathrm{d}s_X, \tag{32}$$

where the Jacobian determinant is

$$J_\Gamma(\mathbf{X}) = \det \mathbf{F}_t(\mathbf{X})\, \|\mathbf{F}_t^{-\top}(\mathbf{X}) \cdot \mathbf{n}(\mathbf{X})\|,$$

with the normal vector $\mathbf{n}(\mathbf{X})$ to Γ at $\mathbf{X}$. In *AutoFreeFem*, the unit exterior normal vector is implemented by the class **SurfaceNormalVector**.

4.6 Matrix functions

An overview of the implemented matrix functions is given in Table 8. They have in common that they take one matrix, *i.e.* a second order tensor, as input argument. The operators in Table 8

operator	*AutoFreeFem* input	LaTeX
matrix transpose	transpose(...)	$(\dots)^\top$
matrix trace	tr(...)	$\mathrm{tr}\,(\dots)$
matrix determinant	determinant(...)	$\det(\dots)$
matrix inverse	inverse(...)	$(\dots)^{-1}$
transpose of inverse	inverse_transpose(...)	$(\dots)^{-\top}$

Table 8: Implemented matrix functions

commute with the pull back to the unperturbed

domain. Furthermore, for the differentiation we have implemented the following rules:

$$
\begin{aligned}
(\mathbf{A}^\top)' &= (\mathbf{A}')^\top, \\
(\operatorname{tr}\mathbf{A})' &= \operatorname{tr}(\mathbf{A}'), \\
\det(\mathbf{A})' &= \det(\mathbf{A}) \operatorname{tr}\left(\mathbf{A}^{-1} \cdot \mathbf{A}'\right), \\
(\mathbf{A}^{-1})' &= -\mathbf{A}^{-1} \cdot \mathbf{A}' \cdot \mathbf{A}^{-1}, \\
(\mathbf{A}^{-\top})' &= -\mathbf{A}^{-\top} \cdot (\mathbf{A}')^\top \cdot \mathbf{A}^{-\top}.
\end{aligned}
$$

4.7 Fixed quantities

An overview of the fixed quantities implemented in *AutoFreeFem* is given in Table 9. The pull back to the unperturbed domain does not alter these quantities and they vanish upon differentiation.

quantity	*AutoFreeFem* input	LaTeX
identity matrix	identity()	$\mathbf{I}$
Cartesian unit vector x-axis	ex()	$\mathbf{e}_x$
Cartesian unit vector y-axis	ey()	$\mathbf{e}_y$
Cartesian unit vector z-axis	ez()	$\mathbf{e}_z$

Table 9: Implemented fixed quantities

5 Non-linear and multi-physics shape optimization examples

This section presents some examples of shape optimization, solved using the automatic code generation capabilities developed in the present paper. In all examples we use the level-set based mesh evolution method introduced in **?**. For a recent tutorial on this method we refer to **?**.

5.1 Verification

For each example, we verified the expressions that we obtained in an automatic way for the shape derivative by looking at the finite difference approximation, as well as the Taylor expansion of the perturbed shape functional. However, the results are shown only for the example of Section 5.2 in Figure 8. For a fixed shape represented by some chosen level-set function and some chosen fixed vector field $\mathbf{V}$ we plot the quantities

$$
e_1(t) = \left| \frac{J(T_t(\Omega)) - J(\Omega)}{t} - DJ(\Omega)(\mathbf{V}) \right|,
$$

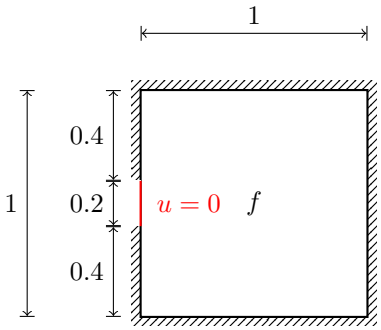


Fig. 7: Geometry and boundary conditions of the diffusion problem.

and

$$
e_2(t) = \left| J(T_t(\Omega)) - J(\Omega) - tDJ(\Omega)(\mathbf{V}) \right|, \quad (34)
$$

for a sequence of decreasing perturbation parameters t. By definition of the shape derivative we have

$$
e_1(t) = \mathcal{O}(t) \quad \text{and} \quad e_2(t) = \mathcal{O}(t^2) \quad \text{as } t \searrow 0.
$$

We remark that in numerical experiments round-off errors are unavoidable. Thus, for $e_1(t)$ we notice a linear decrease in its magnitude with decreasing t when $t > t^*$, where t^* represents a certain threshold. Conversely, $e_1(t)$ tends to increase for $t < t^*$ due to cancellation errors. For $e_2(t)$ we observe a quadratic decrease rate for decreasing t as long as $t > t^*$ and a more or less constant error measure e_2 for $t < t^*$.

5.2 Non-linear diffusion

In this first example, we consider an extension of Model 1 to a two material shape optimization problem. To this end, we additionally consider a compliance objective function and an area penalization with a fixed Lagrange multiplier ℓ. The working domain is the unit square, which is heated by a uniform source of magnitude f. On a small portion of the left side, we assume Dirichlet boundary conditions, whereas all other boundary parts are assumed to be perfectly isolating (see Figure 7). For the non-linear state dependent diffusion coefficient, we assume $\lambda(u) =$

$\chi(x)(1+\alpha u^2)$, where α is a parameter and $\chi(x)$ distinguishes between the two materials. For the material with high conductivity, we have $\chi = 1$, whereas for the material with low conductivity, $\chi = 0.1$. Note that for $\alpha = 0$ the problem becomes linear. A documentation of the formulation of the problem, the adjoint equations and the shape derivative can be obtained by running run_nonlinearDiffusion.py. The numerical values of the considered physical parameters are supplied in Table 10.

Lagrange multiplier	ℓ	100
factor	α	1.00e-02
source	f	-10

Table 10: Numerical values of the physical parameters for the non-linear diffusion problem

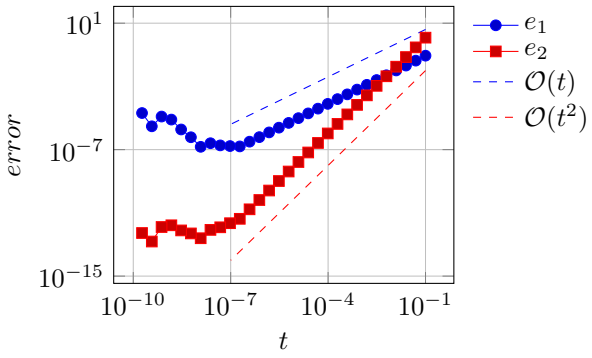
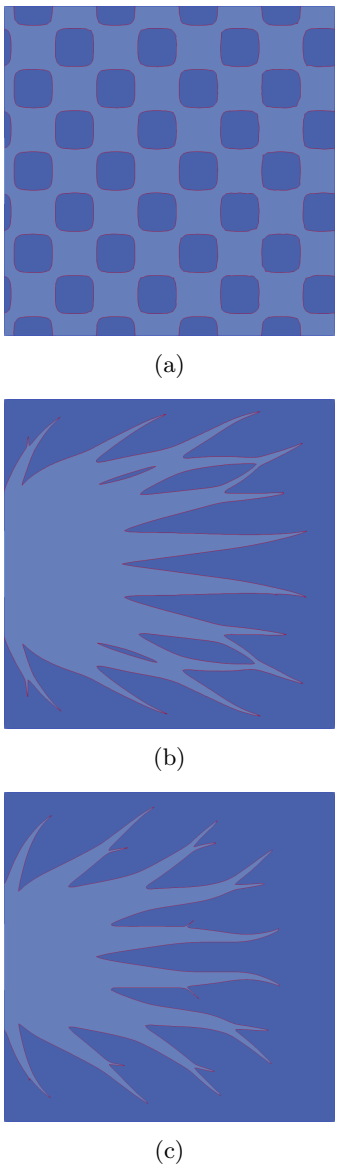


Fig. 8: Results of the verification test for the non-linear diffusion example

The initialization and the corresponding computed optimal designs for a linear model ($\alpha = 0$) and the described non-linear model are displayed in Figure 9. The evolution of the objective function is reported in Figure 10. We observe that for the non-linear model the obtained value of the objective function is lower than for the linear model. This was expected because, in the nonlinear model, the diffusion coefficient is larger than in the linear model.



(a) (b) (c)

Fig. 9: Shape optimization of the diffusion problem: (a) initialization; (b) optimized material distribution for the linear model ($\alpha = 0$); (c) optimized material distribution for the non-linear model ($\alpha = 0.01$)

5.3 Non-linear Elasticity

In this section, we revisit the elasticity cantilever problem discussed in Section 2.2, but now explore both geometrically non-linear and material non-linear behaviors. Again, the working domain is a rectangle of size 2×1, with zero displacement

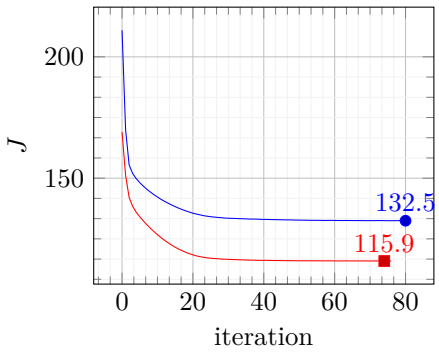


Fig. 10: Convergence history for the diffusion problem: The blue curve shows the evolution of the objective function for the linear model, whereas the red curve for the non-linear model.

boundary condition on the left side and a vertical load applied on a small portion of 0.1 at the middle of the right side denoted by Γ_N such that the resultant force has unit magnitude. All other sides are traction free. The geometry and the boundary conditions are illustrated in Figure 3. There are no body forces. The objective function is analogously to Section 2.2 (see also Section A) the sum of compliance and a fixed Lagrange multiplier ℓ multiplied by the area of the solid,

$$J(\mathbf{u}) = \int_D \ell\chi \, dx + \int_D \mathbf{S}(\mathbf{u}) : \mathbf{E}(\mathbf{u}) \, dx,$$

with the second Piola-Kirchhoff stress tensor $\mathbf{S}$ and the Green-Lagrange strain tensor $\mathbf{E}$.

5.3.1 Non-linear elasticity with Saint Venant-Kirchhoff material

In this section, we consider geometrically non-linear elasticity with the (linear) Saint Venant-Kirchhoff material (see also (**?**, Section 8)). The formulation of the non-linear elasticity problem, the adjoint equations and the shape derivative can be obtained by running run_nonlinearElasticitySaintVernant.py. The numerical values of the considered physical parameters are supplied in Table 3.

As initialization we use the same geometry as for the linear elastic case (see Figure 4a). The computed optimal design is displayed in Figure 11. Due to the non-linear model the design is not symmetric with respect to a horizontal line as it was for the linear model. The evolution of the objective function is reported in Figure 12.

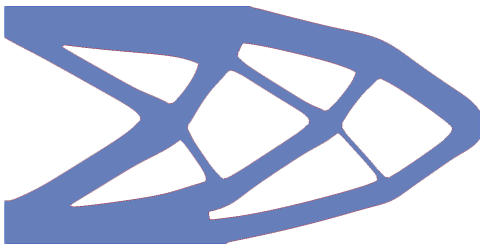


Fig. 11: Optimal design for the non-linear elastic cantilever with Saint Venant-Kirchhoff material.

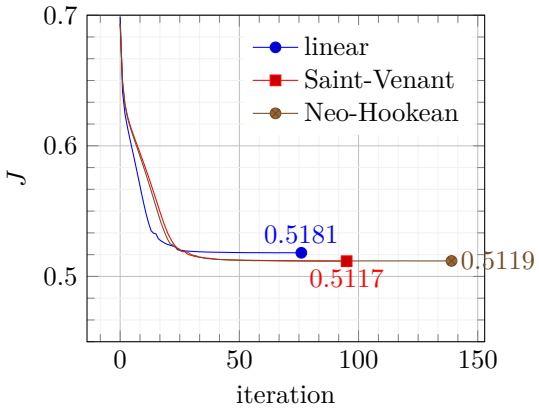


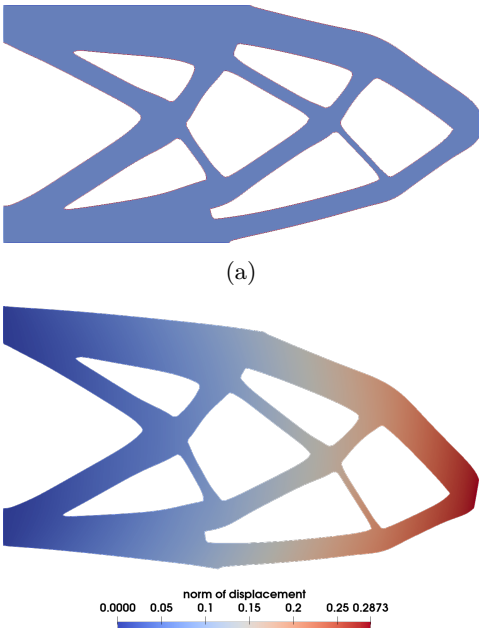
Fig. 12: Convergence history for the cantilever problem with different material laws

5.3.2 Non-linear elasticity with Neo-Hookean material

In this section, we consider now a geometrically and materially non-linear elasticity formulation by resorting to a Neo-Hookean material law **?**. The formulation of the non-linear elasticity problem, the adjoint equations and the shape derivative can be obtained by running run_nonlinearElasticityNeoHookean.py. The numerical values of the considered physical parameters are supplied in Table 3.

As initialization we use the same geometry as for the linear elastic case (see Figure 4a). The computed optimal design is displayed in Figure 13a. Again, due to the non-linear model, the design is not symmetric with respect to a horizontal line

as it was for the linear model. Furthermore, the optimal design differs from the optimal design obtained for the Saint Venant-Kirchhoff material law. In Figure 13b the deformed optimal design is shown. The evolution of the objective function is reported in Figure 12. Although the optimal designs for the different models differ from each other, the obtained values of the objective functions are quite similar.



(a)

(b)

Fig. 13: Non-linear elastic cantilever with Neo-Hookean material: (a) undeformed optimal design; (b) deformed optimal design. The colors indicate the norm of the dispacement.

5.4 Thermo-elasticity

In this example, we optimize a bridge, which is mechanically loaded as well as experiences deformations due to a temperature change. The geometry and the boundary conditions are illustrated in Figure 14. The working domain D is a rectangle of size $4m \times 1m$, with zero displacement boundary condition on the left and right sides Γ_D. A vertical load of constant magnitude q is applied on the top edge Γ_N of the domain. Furthermore, the self-weight (density ρ, gravitational acceleration g) of the bridge is taken into account. The bottom side is traction free. For the thermal part of the problem we prescribe the temperature change T on the lower and the upper edges $\Gamma_T = \Gamma_N \cup \Gamma_1$ and consider three different cases: (a) no temperature change ($g_T(x) = 0$), (b) $T = 30^\circ$ on the upper edge and $T = 0^\circ$ on the lower edge ($g_T(x) = 30y$), and (c) $T = -30^\circ$ on both edges ($g_T(x) = -30$). The state $(T, \mathbf{u}) \in H^1(D) \times [H^1(D)]^2$ is the solution of the classical one-sided coupled thermo-elasticity problem **?**

Poisson's ratio	ν	0.23	–
Young's modulus	E_0	3.20e+07	$\frac{N}{m^2}$
density	ρ	2.50e+03	$\frac{kg}{m^3}$
disp. cost factor	γ	100	$\frac{N}{m^2}$
gravitational acc.	g	9.81	$\frac{m}{s^2}$
material cost factor	ℓ	1	$\frac{N}{m^2}$
penalty parameter	β_T	1.00e+11	$\frac{W}{Km^2}$
penalty parameter	β	1.00e+11	$\frac{N}{m^3}$
ther. conductivity	k_0	1.25	$\frac{W}{Km}$
ther. expansion coeff.	α_T	3.60e-05	$\frac{1}{K}$
vert. load comp.	q	5.00e+03	$\frac{N}{m^2}$

Table 11: Numerical values of the physical and numerical parameters for the thermo-elasticity problem

$$\int_D k\,(\nabla \delta T \cdot \nabla T)\; dx + \int_{\Gamma_T} \beta_T (T - g_T(x)) \delta T \; ds_x = 0, \tag{35}$$

$$\int_{\Gamma_D} \beta\,(\delta \mathbf{u} \cdot \mathbf{u})\; ds_x + \int_D \boldsymbol{\sigma}(\mathbf{u}, T) : \boldsymbol{\epsilon}(\delta \mathbf{u})\, dx$$
$$= \int_{\Gamma_N} q\,(\mathbf{e}_y \cdot \delta \mathbf{u})\; ds_x + \int_D \rho g \chi\,(\mathbf{e}_y \cdot \delta \mathbf{u})\; dx, \tag{36}$$

for all test functions $(\delta T, \delta \mathbf{u}) \in H^1(D) \times [H^1(D)]^2$. We have used the following abbreviations

$$\boldsymbol{\epsilon}(\mathbf{u}) = \frac{\nabla \mathbf{u} + \nabla \mathbf{u}^\top}{2},$$
$$\boldsymbol{\epsilon}_e(\mathbf{u}, T) = \boldsymbol{\epsilon}(\mathbf{u}) - \alpha_T T \mathbf{I},$$

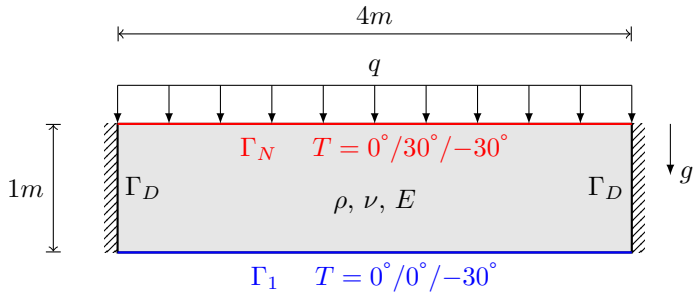


Fig. 14: Geometry and boundary conditions of the thermo-elastic bridge problem.

$$\boldsymbol{\sigma}(\mathbf{u}, T) = \lambda \mathbf{I} \, \mathrm{tr}\,(\boldsymbol{\epsilon}_e(\mathbf{u}, T)) + 2\mu \boldsymbol{\epsilon}_e(\mathbf{u}, T),$$

where α_T is the isotropic thermal expansion coefficient. For the domain occupied by material we have $k = k_0$, $E = E_0$, and $\chi = 1$. Contrary, for the void space we have assumed $k = k_0/1000$, $E = E_0/1000$, and $\chi = 0$. Note that for the imposition of Dirichlet boundary conditions the penalty method is used (penalty parameter β for $\mathbf{u} = \mathbf{0}$ on Γ_D, and β_T for $T = g_T(x)$ on Γ_T). This allows to easily post-process the bearing forces for the evaluation of the objective function.

The numerical values of the considered physical and numerical parameters for concrete material are supplied in Table 11. The objective is to minimize the following three effects:

- the vertical deformation of the upper edge of the domain (displacement cost factor γ),
- the horizontal bearing forces on Γ_D,
- and the material consumption measured as the area (material cost factor ℓ).

The precise objective function to be minimized is

$$J(\mathbf{u}) = -\int_{\Gamma_N} \gamma \, (\mathbf{e}_y \cdot \mathbf{u}) \, ds_x + \int_{\Gamma_D} \beta \, (\mathbf{e}_x \cdot \mathbf{u})^2 \, ds_x + \int_{\Omega} \ell \chi \, dx.$$

The the adjoint equations and the shape derivative can be obtained by running run_thermoElastic.py.

The chosen initial design is depicted in Figure 15a. The optimized designs for the three load cases are visualized in Figures 15b to 15d. The shapes obtained for load cases (a) and (b) exhibit remarkable similarity, while load case (c) yields a significantly different design. In the latter scenario, the optimal design retains the lower horizontal part in the middle of the domain. This phenomenon is due to temperature shrinkage, which creates an uplift force in this particular region counteracting the loading q and the dead load. The evolution of the objective function for the three load cases are reported on Figure 16. We note that the rise in temperature in load case (b) positively impacts the objective function, in contrast to the temperature decrease in load case (c).

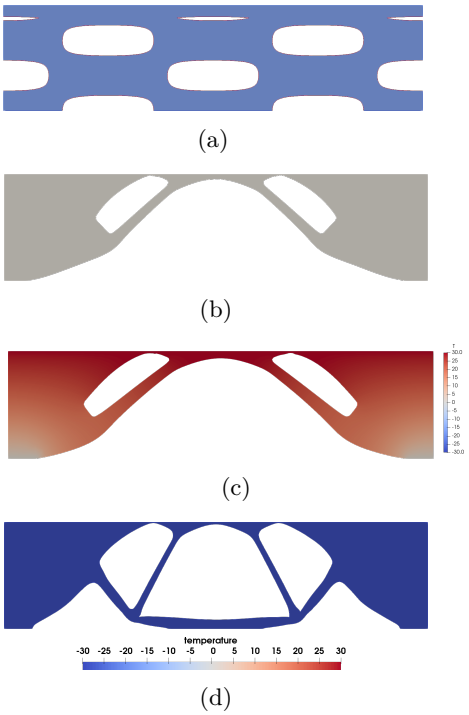


Fig. 15: (a) initialization; (b) optimized design for $T = 0°$; (c) optimized design for $T = 30°$; (d) optimized design for $T = -30°$. The colors in optimized designs indicate the temperature distribution over the structure.

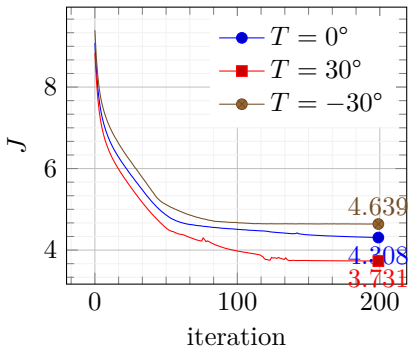


Fig. 16: Convergence history for the thermo-elasticity problem

5.5 Fluid-structure interaction

In this section, we describe a fluid-structure interaction example which is motivated by **??** and **?**. Here, we assume non-linear fluid flow and non-linear elastic deformations by the structure. We use an arbitrary Lagrangian-Euler formulation (ALE) **???** and therefore four unknown fields are sought: the elastic displacement field $\mathbf{u}$, the fluid velocity $\mathbf{v}$, the fluid pressure p, and an extension of the displacement field to the fluid domain $\mathbf{u}_{ext}$. The geometry and boundary conditions of the problem are illustrated in Figure 17. The formulation of the fluid-structure interaction problem, the adjoint equations and the shape derivative can be obtained by running run_FluidStructureInteractionNonlinear.py. Note that the problem is quite complicated and we give only the objective function

$$J = \int_{\Omega_s} \ell \, dx + \int_{\Omega_s} \mathbf{S}(\mathbf{u}) : \mathbf{E}(\mathbf{u}) \, dx, \qquad (37)$$

with the second Piola-Kirchhoff stress tensor $\mathbf{S}$ and the Green-Lagrange strain tensor $\mathbf{E}$. The numerical values of the considered physical parameters are supplied in Table 12. The initial-

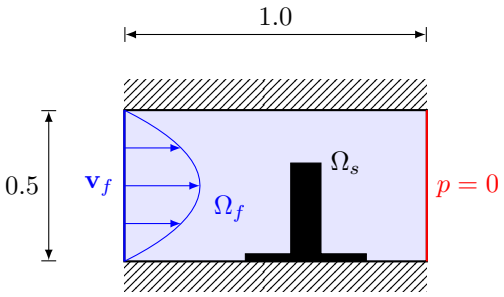


Fig. 17: Geometry and boundary conditions of the fluid-structure interaction problem.

Lagrange multiplier	ℓ	5.00e-03	$\frac{N}{m^2}$
Lamé constant	λ	0.2645	$\frac{N}{m^2}$
Lamé constant	μ	2.38	$\frac{N}{m^2}$
coupling parameter	γ_f	1.00e+08	$\frac{N}{m^3}$
fluid density	ρ	1	$\frac{kg}{m^3}$
fluid viscosity	μ_T	5.00e-03	Pas
penalty parameter	ϵ	1.00e-08	$\frac{1}{Pas}$

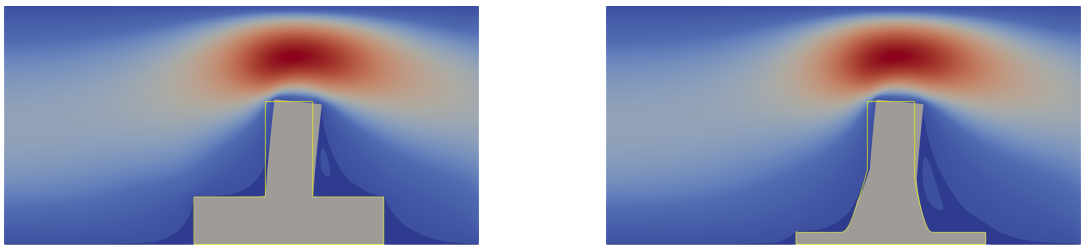
Table 12: Numerical values of the physical parameters for the fluid-structure interaction problem

ization and the optimized material distribution are depicted in Figure 18. The evolution of the objective function is reported in Figure 19.

6 Conclusion

We developed the *Python* package *AutoFreeFem* designed for the automatic generation of simulation code and corresponding problem documentation to facilitate the simulation and optimization of complex non-linear multi-physics problems. A LaTeX component enables users to produce consistent documentations, while the *FreeFEM++* component focuses on the numerical simulation aspect, providing a robust platform for solving partial differential equations. The effectiveness of our approach has been demonstrated through its application to various shape optimization problems.

We believe that this integrated approach offers several significant pedagogical advantages. Firstly,



(a) (b)

Fig. 18: Fluid-structure interaction problem. The deformed structure is shown in grey. The undeformed structure is indicated by the yellow outlines. The colors in the fluid domain represent the norm of the fluid velocity. Red corresponds to high velocity, blue corresponds to low velocity: (a) initialization; (b) optimized design

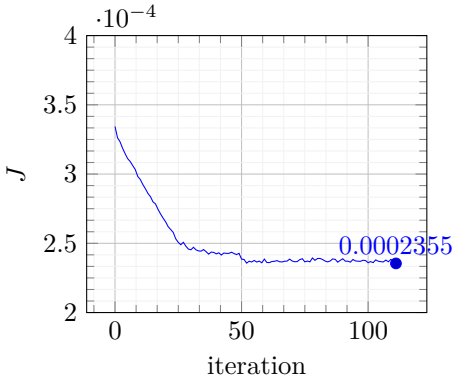


Fig. 19: Convergence history for the fluid-structure problem

it minimizes the risk of discrepancies between the documented theory and the implemented code, as both are derived from the same underlying source. This consistency is crucial for the reproducibility of scientific results. Secondly, the automation of code and documentation generation saves time and reduces the potential for human error, especially for students or beginners in the field.

Supplementary information.

Compliance with ethical standards

Funding

The authors did not receive support from any organization for the submitted work.

Conflict of interest

The authors declare that they have no conflict of interest.

Replication of results

The developed python package is available at https://gitlab.tugraz.at/autofreefem/autofreefem. This allows to reproduce all results of the present paper. All computations were performed using *FreeFEM++* version 4.14.

Appendix A LaTeX documentation of the linear elasticity cantilever problem

All boxed content in the appendix has been automatically generated by *AutoFreeFem*. In order to demonstrate the capabilities of the software, no manual improvements have been made to the output.

Let $\phi(x)$ be the level-set function and

$$\chi(x) = \begin{cases} 1 & \text{if} \quad \phi(x) < 0 \\ 1/100 & \text{if} \quad \phi(x) \geq 0 \end{cases}.$$

The Lagrangian of the linear Elasticity problem is

$$\mathcal{L}([\mathbf{u}], [\delta\mathbf{u}]) = J(\mathbf{u}) - \int_{\Gamma_N} f\,(\mathbf{e}_y \cdot \delta\mathbf{u})\; ds_x + \int_D (\boldsymbol{\sigma}(\mathbf{u}) : \nabla\delta\mathbf{u})\; dx,$$

with

$$\begin{aligned} \lambda &= -\frac{E\nu}{2\nu^2 + \nu - 1}, \\ J(\mathbf{u}) &= \int_D \ell\chi\, dx + \int_D (\boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{u}))\; dx, \\ \boldsymbol{\varepsilon}(\mathbf{u}) &= \frac{\nabla\mathbf{u} + \nabla\mathbf{u}^\top}{2}, \\ \mu &= \frac{E}{2\,(\nu + 1)}, \\ \boldsymbol{\sigma}(\mathbf{u}) &= (\lambda\mathbf{I}\ \mathrm{tr}\,(\boldsymbol{\varepsilon}(\mathbf{u})) + 2\mu\boldsymbol{\varepsilon}(\mathbf{u}))\,\chi. \end{aligned}$$

The state $[\mathbf{u}]$ is the solution of the linear problem

$$-\int_{\Gamma_N} f\,(\mathbf{e}_y \cdot \delta\mathbf{u})\; ds_x + \int_D (\boldsymbol{\sigma}(\mathbf{u}) : \nabla\delta\mathbf{u})\; dx = 0 \qquad \forall\ \delta\mathbf{u}.$$

The adjoint state $[\tilde{\mathbf{u}}]$ to the direct state $[\mathbf{u}]$ is the solution of

$$\partial_{(\mathbf{u},\delta\mathbf{u})} J(\mathbf{u}, \delta\mathbf{u}) + \int_D \left(\partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\sigma}(\delta\mathbf{u}) : \nabla\tilde{\mathbf{u}}\right)\; dx = 0 \qquad \forall\ \delta\mathbf{u},$$

with

$$\begin{aligned} \partial_{(\mathbf{u},\delta\mathbf{u})} J(\mathbf{u}, \delta\mathbf{u}) &= \int_D \left(\partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\sigma}(\delta\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{u})\right) + \left(\boldsymbol{\sigma}(\mathbf{u}) : \partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\varepsilon}(\delta\mathbf{u})\right)\; dx, \\ \partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\sigma}(\delta\mathbf{u}) &= \left(\lambda\mathbf{I}\ \mathrm{tr}\left(\partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\varepsilon}(\delta\mathbf{u})\right) + 2\mu\partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\varepsilon}(\delta\mathbf{u})\right)\chi, \\ \partial_{(\mathbf{u},\delta\mathbf{u})}\boldsymbol{\varepsilon}(\delta\mathbf{u}) &= \frac{\nabla\delta\mathbf{u} + \nabla\delta\mathbf{u}^\top}{2}. \end{aligned}$$

In order to compute the shape derivative, we consider a shape pertubation $\mathbf{x} = T_t(\mathbf{X}) = \mathbf{X} + t\mathbf{V}(X)$ with a suitable velocity field $\mathbf{V}$. The perturbed Lagrangian $\mathcal{G}$ using the Lagrangian state $[\mathbf{u}]^L$ is

given by

$$\mathcal{G}(t,[\mathbf{u}]^L,[\delta\mathbf{u}]^L) = \mathcal{V}(J) - \int_{\Gamma_N} f \det \mathbf{F}(t)\,(\mathbf{e}_y\cdot\delta\mathbf{u})\,\sqrt{((\mathbf{n}\cdot\mathbf{F}^{-1}(t))\cdot(\mathbf{n}\cdot\mathbf{F}^{-1}(t)))}\,ds_x + \int_D \det\mathbf{F}(t)\left(\mathcal{V}(\boldsymbol{\sigma}) : \left(\left(\nabla\delta\mathbf{u}\cdot\mathbf{F}^{-1}(t)\right)\right)\right)\,dx,$$

with

$$\mathcal{V}(\boldsymbol{\sigma}) = (\lambda\mathbf{I}\,\mathrm{tr}\,(\mathcal{V}(\boldsymbol{\varepsilon})) + 2\mu\mathcal{V}(\boldsymbol{\varepsilon}))\,\chi,$$

$$\mathcal{V}(\boldsymbol{\varepsilon}) = \frac{\left(\nabla\mathbf{u}\cdot\mathbf{F}^{-1}(t)\right) + \left(\left(\nabla\mathbf{u}\cdot\mathbf{F}^{-1}(t)\right)\right)^\top}{2},$$

$$\mathcal{V}(J) = \int_D \det\mathbf{F}(t)\,(\mathcal{V}(\boldsymbol{\sigma}) : \mathcal{V}(\boldsymbol{\varepsilon}))\,dx + \int_D \ell\chi\det\mathbf{F}(t)\,dx.$$

For the direct state $[\mathbf{u}]$ and the adjoint state $[\tilde{\mathbf{u}}]$, the volume expression of the shape derivative is given by

$$DJ(\Omega)(\mathbf{V}) = \partial_t J(\mathbf{u},\mathbf{V}) - \int_{\Gamma_N} f\,\mathrm{div}\,\mathbf{V}\,(\mathbf{e}_y\cdot\tilde{\mathbf{u}})\,ds_x + \int_{\Gamma_N} f\,(\mathbf{n}\cdot(\mathbf{n}\cdot\nabla\mathbf{V}))\,(\mathbf{e}_y\cdot\tilde{\mathbf{u}})\,ds_x + \int_D \mathrm{div}\,\mathbf{V}\,(\boldsymbol{\sigma}(\mathbf{u}) : \nabla\tilde{\mathbf{u}}) + (\partial_t\boldsymbol{\sigma}(\mathbf{u},\mathbf{V}) : \nabla\tilde{\mathbf{u}}) - (\boldsymbol{\sigma}(\mathbf{u}) : ((\nabla\tilde{\mathbf{u}}\cdot\nabla\mathbf{V})))\,dx,$$

with

$$\partial_t J(\mathbf{u},\mathbf{V}) = \int_D \ell\chi\,\mathrm{div}\,\mathbf{V}\,dx + \int_D \mathrm{div}\,\mathbf{V}\,(\boldsymbol{\sigma}(\mathbf{u}) : \boldsymbol{\varepsilon}(\mathbf{u})) + (\partial_t\boldsymbol{\sigma}(\mathbf{u},\mathbf{V}) : \boldsymbol{\varepsilon}(\mathbf{u})) + (\boldsymbol{\sigma}(\mathbf{u}) : \partial_t\boldsymbol{\varepsilon}(\mathbf{u},\mathbf{V}))\,dx,$$

$$\partial_t\boldsymbol{\sigma}(\mathbf{u},\mathbf{V}) = (\lambda\mathbf{I}\,\mathrm{tr}\,(\partial_t\boldsymbol{\varepsilon}(\mathbf{u},\mathbf{V})) + 2\mu\partial_t\boldsymbol{\varepsilon}(\mathbf{u},\mathbf{V}))\,\chi,$$

$$\partial_t\boldsymbol{\varepsilon}(\mathbf{u},\mathbf{V}) = -\frac{(\nabla\mathbf{u}\cdot\nabla\mathbf{V}) + ((\nabla\mathbf{u}\cdot\nabla\mathbf{V}))^\top}{2}.$$

References

Meurer, A., Smith, C.P., Paprocki, M., Čertík, O., Kirpichev, S.B., Rocklin, M., Kumar, A., Ivanov, S., Moore, J.K., Singh, S., Rathnayake, T., Vig, S., Granger, B.E., Muller, R.P., Bonazzi, F., Gupta, H., Vats, S., Johansson, F., Pedregosa, F., Curry, M.J., Terrel, A.R., Roučka, v., Saboo, A., Fernando, I., Kulal, S., Cimrman, R., Scopatz, A.: Sympy: symbolic computing in python. PeerJ Computer Science **3**, 103 (2017) https://doi.org/10.7717/peerj-cs.103

Hecht, F.: New development in FreeFem++. Journal of numerical mathematics **20**(3-4), 251–266 (2012) https://doi.org/10.1515/jnum-2012-0013

Allaire, G., Dapogny, C., Frey, P.: Shape optimization with a level set based mesh evolution method. Computer Methods in Applied Mechanics and Engineering **282**, 22–53 (2014) https://doi.org/10.1016/j.cma.2014.08.028

Dapogny, C., Dobrzynski, C., Frey, P.: Three-dimensional adaptive domain remeshing, implicit domain meshing, and applications to free and moving boundary problems. Journal of Computational Physics **262**, 358–378 (2014) https://doi.org/10.1016/j.jcp.2014.01.005

Dapogny, C., Frey, P.: Computation of the signed distance function to a discrete contour on adapted triangulation. Calcolo **49**, 193–219 (2012) https://doi.org/10.1007/s10092-011-0051-z

Alnæs, M., Blechta, J., Hake, J., Johansson, A., Kehlet, B., Logg, A., Richardson, C., Ring, J., Rognes, M.E., Wells, G.N.: The FEniCS project version 1.5. Archive of Numerical Software **3**(100) (2015) https://doi.org/10.11588/ans.2015.100.20553

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

Schmidt, S.: Weak and strong form shape hessians and their automatic generation. Siam journal on scientific computing **40**(2), 210–233 (2018) https://doi.org/10.1137/16m1099972

Ham, D.A., Mitchell, L., Paganini, A., Wechsung, F.: Automated shape differentiation in the unified form language. Structural and multidisciplinary optimization **60**, 1813–1820 (2019) https://doi.org/10.1007/s00158-019-02281-z

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

Dokken, J.S., Mitusch, S.K., Funke, S.W.: Automatic shape derivatives for transient pdes in fenics and firedrake. arXiv preprint arXiv:2001.10058 (2020)

Paganini, A., Wechsung, F.: Fireshape: a shape optimization toolbox for firedrake. Structural and Multidisciplinary Optimization **63**, 2553–2569 (2021) https://doi.org/10.1007/s00158-020-02813-y

Schöberl, J.: C++11 implementation of finite elements in NGSolve. Technical Report 30/2014, Institute for Analysis and Scientific Computing, Vienna University of Technology (2014)

Gangl, P., Sturm, K., Neunteufel, M., Schöberl, J.: Fully and semi-automated shape differentiation in ngsolve. Structural and multidisciplinary optimization **63**, 1579–1607 (2021) https://doi.org/10.1007/s00158-020-02742-w

Gangl, P., Sturm, K.: Automated computation of topological derivatives with application to nonlinear elasticity and reaction–diffusion problems. Computer Methods in Applied Mechanics and Engineering **398**, 115288 (2022) https://doi.org/10.1016/j.cma.2022.115288

Blauth, S.: cashocs: A computational, adjoint-based shape optimization and optimal control software. SoftwareX **13**, 100646 (2021) https://doi.org/10.1016/j.softx.2020.100646

Blauth, S.: Version 2.0 - cashocs: A computational, adjoint-based shape optimization and optimal control software. SoftwareX **24**, 101577 (2023) https://doi.org/10.1016/j.softx.2023.101577

Chandrasekhar, A., Sridhara, S., Suresh, K.: Auto: a framework for automatic differentiation in topology optimization. Structural and Multidisciplinary Optimization **64**(6), 4355–4365 (2021) https://doi.org/10.1007/s00158-021-03025-8

Allaire, G., Dapogny, C., Jouve, F.: Shape and topology optimization. In: Handbook of Numerical Analysis vol. 22, pp. 1–132. North-Holland, Amsterdam (2021). https://doi.org/10.1016/bs.hna.2020.10.004

Wriggers, P.: Nonlinear Finite Element Methods. Springer, Berlin Heidelberg (2008). https://doi.org/10.1007/978-3-540-71001-1

Hinze, M., Pinnau, R., Ulbrich, M., Ulbrich, S.: Optimization with PDE Constraints vol. 23. Springer, Dordrecht (2008). https://doi.org/10.1007/978-1-4020-8839-1

Sturm, K.: Minimax lagrangian approach to the differentiability of nonlinear pde constrained shape functions without saddle point assumption. SIAM Journal on Control and Optimization **53**(4), 2017–2039 (2015) https://doi.org/10.1137/130930807

Henrot, A., Pierre, M.: Shape Variation and Optimization: A Geometrical Analysis vol. 28. EMS tracts in mathematics, European Mathematical Society, Zürich (2018). https://doi.org/10.4171/178

Dapogny, C., Feppon, F.: Shape optimization using a level set based mesh evolution method: an overview and tutorial. Comptes Rendus. Mathématique **361**, 1267–1332 (2023) https://doi.org/10.5802/crmath.498

Allaire, G., Jouve, F., Toader, A.-M.: Structural optimization using sensitivity analysis and a level-set method. Journal of Computational Physics **194**(1), 363–393 (2004) https://doi.org/10.1016/j.jcp.2003.09.032

Nowacki, W.: Thermoelasticity, 2. edn. Pergamon Press, Oxford (1986). https://doi.org/10.1016/C2013-0-03247-1

Yoon, G.H.: Topology optimization for stationary fluid–structure interaction problems using a new monolithic formulation. International Journal for Numerical Methods in Engineering **82**(5), 591–616 (2010) https://doi.org/10.1002/nme.2777

Yoon, G.H.: Stress-based topology optimization method for steady-state fluid–structure interaction problems. Computer Methods in Applied Mechanics and Engineering **278**, 499–523 (2014) https://doi.org/10.1016/j.cma.2014.05.021

Feppon, F., Allaire, G., Bordeu, F., Cortial, J., Dapogny, C.: Shape optimization of a coupled thermal fluid–structure problem in a level set mesh evolution framework. SeMA Journal **76**, 413–458 (2019) https://doi.org/10.1007/s40324-018-00185-4

Le Tallec, P., Mouro, J.: Fluid structure interaction with large structural displacements. Computer Methods in Applied Mechanics and Engineering **190**(24), 3039–3067 (2001) https://doi.org/10.1016/S0045-7825(00)00381-9

Dowell, E.H., Hall, K.C.: Modeling of fluid-structure interaction. Annual review of fluid mechanics **33**(1), 445–490 (2001) https://doi.org/10.1146/annurev.fluid.33.1.445

Hou, G., Wang, J., Layton, A.: Numerical methods for fluid-structure interaction—a review. Communications in Computational Physics **12**(2), 337–377 (2012) https://doi.org/10.4208/cicp.291210.290411s