Technische Universität Graz

Fully and Semi-Automated Shape Differentiation in NGSolve

P. Gangl, K. Sturm, M. Neunteufel, J. Schöberl

Berichte aus dem Institut für Angewandte Mathematik

Technische Universität Graz

Fully and Semi-Automated Shape Differentiation in NGSolve

P. Gangl, K. Sturm, M. Neunteufel, J. Schöberl

Berichte aus dem
Institut für Angewandte Mathematik

Technische Universität Graz
Institut für Angewandte Mathematik
Steyrergasse 30
A 8010 Graz

WWW: `http://www.applied.math.tugraz.at`

Fully and Semi-Automated Shape Differentiation in `NGSolve`

Peter Gangl[*], Kevin Sturm[†] , Michael Neunteufel[‡] and Joachim Schöberl[§]

April 14, 2020

Abstract

In this paper we present a framework for automated shape differentiation in the finite element software `NGSolve`. Our approach combines the mathematical Lagrangian approach for differentiating PDE constrained shape functions with the automated differentiation capabilities of `NGSolve`. The user can decide which degree of automatisation is required and thus allows for either a more custom-like or black-box-like behaviour of the software.

We discuss the automatic generation of first and second order shape derivatives for unconstrained model problems as well as for more realistic problems that are constrained by different types of partial differential equations. We consider linear as well as nonlinear problems and also problems which are posed on surfaces. In numerical experiments we verify the accuracy of the computed derivatives via a Taylor test. Finally we present first and second order shape optimisation algorithms and illustrate them for several numerical optimisation examples ranging from nonlinear elasticity to Maxwell's equations.

Keywords: shape optimisation, automated differentiation, shape Newton method

1 Introduction

Numerical simulation and shape optimisation tools to solve the problems have become an integral part in the design process of many products. Starting out from an initial design, non-parametric shape optimisation techniques based on first and second order shape derivatives can assist in finding shapes of a product which are optimal with respect to a given objective function. Examples include the optimal design of aircrafts [28, 29], optimal inductor design [16], optimisation of microlenses [23], the optimal design of electric motors [10], applications from mechanical engineering [2, 19], multiphysics problems [9] or electrical impedance tomography (EIT) in medical sciences to name only a few [13].

Shape optimisation algorithms are based on the concept of shape derivatives. Let $\mathscr{A} \subset \mathscr{P}(\mathbf{R}^d)$ a set of admissible shapes and $\mathscr{J} : \mathscr{A} \to \mathbf{R}$ a shape function. Given an admissible shape $\Omega \in \mathscr{A}$

---

[*]TU Graz, Steyrergasse 30, 8010 Graz, Austria, E-Mail: gangl(at)math.tugraz.at

[†]TU Wien, Wiedner Hauptstr. 8-10, 1040 Vienna, Austria, E-Mail: kevin.sturm(at)tuwien.ac.at

[‡]TU Wien, Wiedner Hauptstr. 8-10, 1040 Vienna, Austria, E-Mail: michael.neunteufel(at)tuwien.ac.at

[§]TU Wien, Wiedner Hauptstr. 8-10, 1040 Vienna, Austria, E-Mail: joachim.schoeberl(at)tuwien.ac.at

and a sufficiently smooth vector field V, we define the perturbed domain $\Omega_t := (\mathsf{Id} + tV)(\Omega)$ for a small perturbation parameter $t > 0$. The shape derivative is defined as

$$D\mathscr{J}(\Omega)(V) := \left(\frac{d}{dt}\mathscr{J}(\Omega_t)\right)\bigg|_{t=0} = \lim_{t\searrow 0}\frac{\mathscr{J}(\Omega_t) - \mathscr{J}(\Omega)}{t}. \qquad (1.1)$$

In most practically relevant applications, the objective functional depends on the shape of a (sub-)domain via the solution of a partial differential equation (PDE). Thus, one is facing a problem of PDE-constrained shape optimisation of the form

$$\begin{aligned}
&\min_{(\Omega,u)\in\mathscr{A}\times X} J(\Omega, u)\\
&\text{s.t. } (\Omega, u) \in \mathscr{A}\times X : e(\Omega; u, v) = 0 \quad \text{for all } v \in X.
\end{aligned} \qquad (1.2)$$

Here, the second line represents the constraining boundary value problem posed on a Hilbert space X, which we assume to be uniquely solvable for all admissible $\Omega \in \mathscr{A}$. Denoting the unique solution for a given $\Omega \in \mathscr{A}$ by u_Ω, we introduce the notation for the reduced functional

$$\mathscr{J}(\Omega) := J(\Omega, u_\Omega).$$

In order to be able to apply a shape optimisation algorithm to a given problem of this kind, the shape derivative (1.1) has to be computed, see the standard literature [5, 32] or [33] for an overview of different approaches. In the following we focus on computing the so-called volume form of the shape derivative which in a finite element context is known to give a better approximation compared to the boundary form; see [4, 15].

The convergence of shape optimisation algorithms can be speeded up by using second order shape derivatives. Given two sufficiently smooth vector fields V, W and an admissible shape $\Omega \in \mathscr{A}$, let $\Omega_{s,t} := (\mathsf{Id} + sV + tW)(\Omega)$ the perturbed domain. Then, the second order shape derivative is defined as

$$D^2\mathscr{J}(\Omega)(V)(W) := \left(\frac{d^2}{dsdt}\mathscr{J}(\Omega_{s,t})\right)\bigg|_{s,t=0}. \qquad (1.3)$$

Second order information in Newton-type algorithms has been explored in the articles [1, 8, 22, 24, 31]. Since the computation of second order shape derivatives is more involved and error prone, several authors have employed automatic differentiation (AD) tools, see e.g. [27] and [12] for two approaches based on the Unified Form Language (UFL) [3]. In [12], the authors present a fully automated shape differentiation software which uses the transformation properties on the finite element level. In [27] (see also the earlier work [26]) the automated derivatives are computed using UFL. The strategy of [12] and [27] differ in that, for the latter, the software computes an unsymmetric shape Hessian since it involves the term $D\mathscr{J}(\Omega)(\partial VW)$. Optionally the software allows to make the shape Hessian symmetric by requiring $\partial VW = 0$. Let us also mention [7] where automated shape derivatives for transient PDEs in FEniCS and Firedrake are presented.

In this paper we present an alternative framework for AD of PDE constrained problems of type (1.2). There exist several approaches for the rigorous derivation of the shape derivative of PDE-constrained shape functionals, see [34] for an overview. The main idea, however, is always

similar. After transforming the perturbed setting back to the original domain, shape differentiation in the direction of a given vector field reduces to the differentiation with respect to the scalar parameter t which now enters via the corresponding transformation and its gradient. It is shown in [33] that the shape derivative for a nonlinear PDE-constrained shape optimisation problem can be computed as the derivative of the Lagrangian with respect to the perturbation parameter. We will illustrate this systematic procedure for a number of different applications and utilise symbolic differentiation provided by the finite element software package NGSolve [30] to obtain the shape derivative for different classes of PDE-constrained optimisation problems. NGSolve allows for a fast and efficient numerical solution of a large number of different boundary value problems. The aim of this paper is to extend NGSolve by the possibility of semi-automatic and fully automatic shape differentiation and optimisation.

Distinctly from previous approaches we cover the following two points:

- a fully automated setting requiring as input the weak formulation of the constraint and the cost function,

- a semi-automated setting which offers a highly customizable user interface, but requires mathematical background knowledge.

Structure of the paper. In Section 2 we give a brief introduction on how to solve a PDE in NGSolve and present its built-in auto-differentiation capabilities. The introduced syntax will also lay the foundation for the following sections. In Section 3 we present a first unconstrained shape optimisation problem and show how to solve it in NGSolve. For this purpose we show how to compute the first and second order shape derivative in a semi-automated way. Section 4 extends the preceding section by incorporating a PDE constraint. The strategy is illustrated by means of a simple Poisson equation. We also show how to treat the computation of shape derivatives when the PDE is defined on surfaces. While the semi-automated shape differentiation presented in Sections 3 and 4 requires mathematical background knowledge, in Section 5 we show how the shape derivatives can be computed in a fully automated fashion. In the last section of the paper we verify the computed formulas by a Taylor test, discuss optimisation algorithms and present several numerical optimisation examples including nonlinear elasticity, Maxwell's equations and Helmholtz's equation.

2 A brief introduction to NGSolve

In this section, we give a brief overview over the main concepts of the finite element software NGSolve [30]. We first describe the main principles for numerically solving boundary value problems in NGSolve before focusing on its built-in automatic differentiation capabilities. In the subsequent sections of this paper, these ingredients will be combined to implement the shape derivative of unconstrained and PDE-constrained shape optimisation problems in an automated way.

2.1   Solving PDEs with finite elements in `NGSolve`

In this section, we illustrate the syntax of `NGSolve` using the `python` programming language for the Poisson equation with homogeneous Dirichlet conditions as a model problem. We refer the reader to the online documentation

<p align="center">https://ngsolve.org/docu/latest/</p>

for a more detailed description of the many features.

The weak form of the model problem on a domain $\Omega \subset \mathbf{R}^d$ reads

$$\text{Find } u \in H_0^1(\Omega) : \int_\Omega \nabla u \cdot \nabla w \, \mathrm{d}x = \int_\Omega f w \, \mathrm{d}x \quad \text{for all } w \in H_0^1(\Omega). \tag{2.1}$$

We consider a ball of radius $\frac{1}{2}$ in two space dimensions centered at the point $(0.5, 0.5)^\top$, i.e. $\Omega = B((0.5, 0.5)^\top, 0.5)$, and the right hand side is defined by $f(x_1, x_2) = 2x_2(1-x_2)+2x_1(1-x_1)$. We will go through the steps for numerically solving this problem by the finite element method.

We begin by importing the necessary functionalities and setting up a finite element mesh.

```python
from ngsolve import *
from netgen.geom2d import SplineGeometry

geo = SplineGeometry()
geo.AddCircle((0.5,0.5),0.5,bc="circle")

mesh = Mesh(geo.GenerateMesh(maxh=0.2))
mesh.Curve(3)
```

The first line imports all modules from the package `NGSolve`. The second line includes the SplineGeometry function which enables us to define a mesh via a geometric description, in our case a circle centered at $(0.5, 0.5)^\top$ of radius 0.5. Finally the mesh is generated in line 7 and in line 8 we specify that we want to use a curved finite element mesh for a more accurate approximation of the geometry.

Next in line 9 we define an H^1 conforming finite element space of polynomial degree 3 and include Dirichlet boundary conditions on the boundary of the domain $\partial\Omega$ (referenced by the string ``circle'' that we assigned in line 5). On this space we define a trial function `u` in line 11 and a test function `w` in line 12. These are purely symbolic objects which are used to define boundary value problems in weak form.

```python
fes = H1(mesh, order=3, dirichlet="circle")

u = fes.TrialFunction()
w = fes.TestFunction()
```

For a more compact presentation later on, we define a coefficient function `X` which combines the three spatial components:

```python
X = CoefficientFunction((x,y,z))
```

Now, the left and right hand side of problem (2.1) can be conveniently defined as a bilinear or linear form, respectively, on the finite elements space `fes` by the following lines.
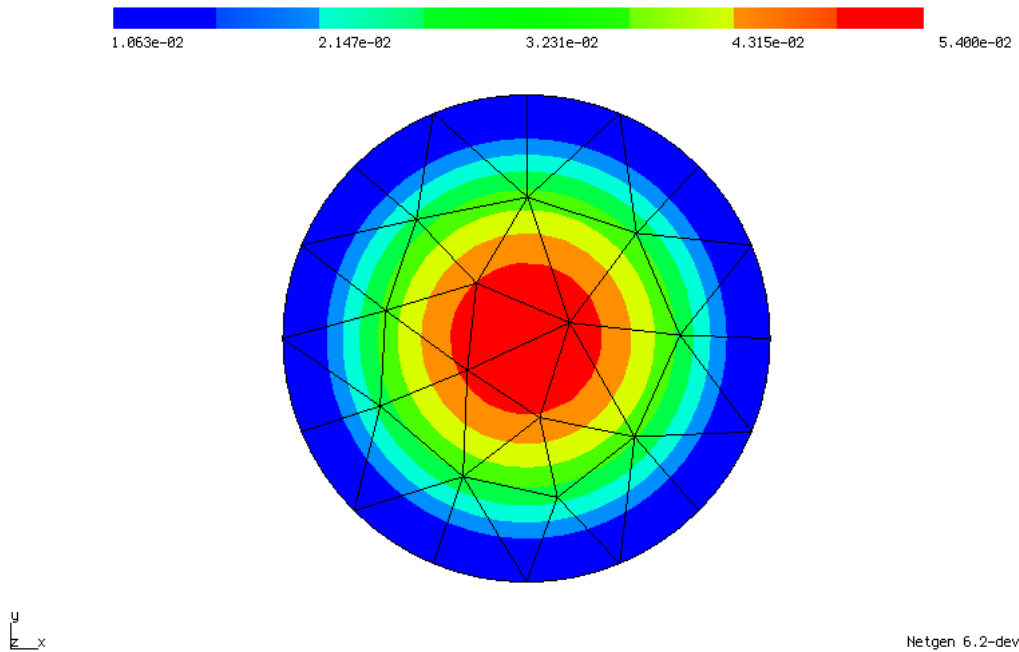
Figure 1: Solution of problem (2.1) by code fragments of Section 2.1 with 29 nodes, 40 (curved) triangular elements and polynomial order 3.

```
14 L = LinearForm ( fes )
15 f1 = (2*X[1]*(1−X[1])+2*X[0]*(1−X[0]))
16 L += f1 * w * dx
17
18 a = BilinearForm ( fes , symmetric=True )
19 a += grad (u) * grad (w) * dx
```

We assemble the system matrix coming from the bilinear form a and the load vector coming from L and solve the corresponding system of linear equations.

```
20 a . Assemble ()
21 L . Assemble ()
22
23 gfu = GridFunction ( fes )
24 gfu . vec . data = a . mat . Inverse ( fes . FreeDofs () , inverse="sparsecholesky ") * L . vec
25
26 Draw ( gfu , mesh , "state" )
```

Here, `gfu` is defined as a `GridFunction` over the finite element space `fes`. The Dirichlet conditions are incorporated into the direct solution of the linear system and the numerical solution is drawn in the graphical user interface. The numerical solution is depicted in Figure 1.

2.2   Automatic Differentiation in `NGSolve`

In `NGSolve`, symbolic expressions are stored in expression trees, see Figure 2 for an example. It is possible to differentiate an expression *expr* with respect to a variable *var* appearing in *expr* into a direction *dir* by the command

$$\texttt{expr.Diff(var, dir)}.$$

Mathematically this line corresponds to the directional derivative of g:=*expr* at $x := var$ in direction $v := dir$, that is,

$$Dg(x)(v). \tag{2.2}$$

When calling the `Diff` command for `expr`, the expression tree of *expr* is gone through node by node, and for each node the corresponding differentiation rules such as product rule or chain rule are applied. When a node represents the variable with respect to which the differentiation is carried out, it is replaced by the direction *dir* of differentiation.

   Figure 2 shows the differentiation of the expression *expr= 2x*x+3y* with respect to *x* into the direction given by *v*:

```
27  v = Parameter(1)
28  expr = 2*x*x+3*y
29  dexpr = expr.Diff(x,v)
30  print(expr)
31  print(dexpr)
```

   The output of `print(expr)` reads

```
coef binary operation '+', real
  coef binary operation '*', real
    coef scale 2, real
      coef coordinate x, real
    coef coordinate x, real
  coef scale 3, real
    coef coordinate y, real
```

which translates to $2x * x + 3y$ and corresponds to the expression tree depicted in Figure 2(a). The output of `print(dexpr)` reads

```
coef binary operation '+', real
  coef binary operation '+', real
    coef binary operation '*', real
      coef scale 2, real
        coef N5ngfem28ParameterCoefficientFunctionE, real
      coef coordinate x, real
    coef binary operation '*', real
      coef scale 2, real
        coef coordinate x, real
      coef N5ngfem28ParameterCoefficientFunctionE, real
  coef scale 3, real
    coef 0, real
```
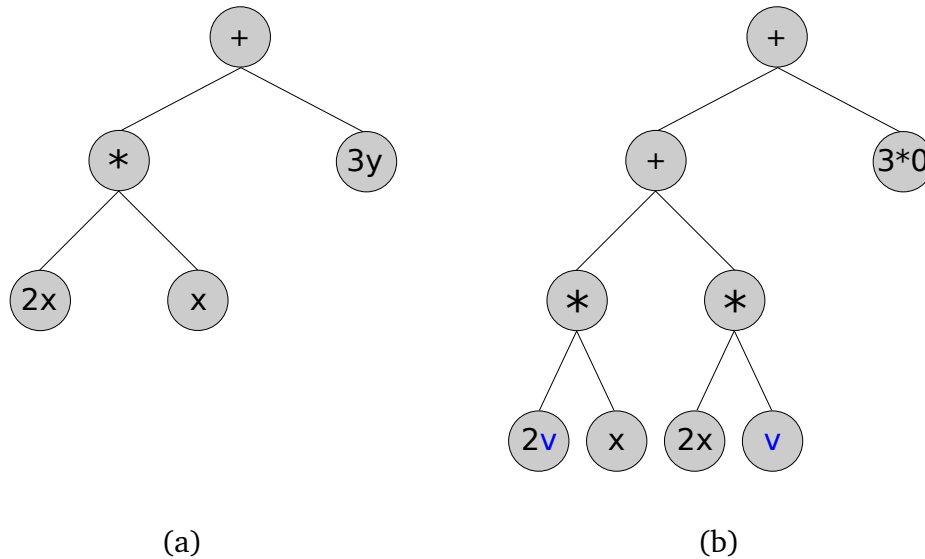
(a)   (b)

Figure 2: Illustration of `Diff` command for example *expr= 2x*x+3y*. (a) Expression tree for *expr*. (b) Expression tree for expression obtained by call of `expr.Diff(x, dir)`.

which translates to $(2v * x + 2x * v) + 3 * 0$ and corresponds to the expression tree depicted in Figure 2(b).

By default NGSolve `GridFunctions` as well as trial and test functions do not depend on the spatial variables x, y, z. While trial and test functions are purely symbolic objects, `GridFunctions` represent functions in the finite element space. The code segments

```
32 u = fes.TrialFunction()    # symbolic object
33 w = fes.TestFunction()      # symbolic object
34 gf = GridFunction(fes)
35 gf.Set(x*x*y)
36
37 print("Diff u wrt x", u.Diff(x))
38 print("Diff w wrt x", w.Diff(x))
39 print("Diff gf wrt x", gf.Diff(x))
```

will give the following output:

```
Diff u wrt x:  ConstantCF, val = 0
Diff w wrt x:  ConstantCF, val = 0
Diff gf wrt x:  ConstantCF, val = 0
```

3   Semi-automatic shape differentiation without constraints

We will illustrate the steps to be taken in order to obtain the shape derivative of a shape function in a semi-automatic way for a simple shape optimisation problem. For $\Omega \subset \mathbf{R}^d$ bounded and open and a continuously differentiable function $f \in C^1(\mathbf{R}^d)$, we consider the shape differentiation of

the shape function

$$\mathscr{J}(\Omega) = \int_{\Omega} f(x)\,\mathrm{d}x. \tag{3.1}$$

Clearly the minimiser of $\mathscr{J}$ over all measurable sets in $\mathbf{R}^d$ is given by $\Omega^* = \{x \in \mathbf{R}^d : f(x) < 0\}$. We also refer to [25] for the computations of first and second order variations of type (3.1) where Ω is a submanifold of $\mathbf{R}^d$.

3.1 First order shape derivative

Given a vector field $V \in C^{0,1}(\mathbf{R}^d)^d$, we define the transformation

$$T_t(x) := (\mathsf{Id} + t\,V)(x), \quad x \in \mathbf{R}^d, \quad t \geq 0.$$

Definition 3.1. The first order shape derivative of a shape function $\mathscr{J}$ at Ω in direction $V \in C^{0,1}(\mathbf{R}^d)^d$ is defined by

$$D\mathscr{J}(\Omega)(V) = \lim_{t \searrow 0} \frac{\mathscr{J}(T_t(\Omega)) - \mathscr{J}(\Omega)}{t}. \tag{3.2}$$

3.1.1 Shape differentiation of unconstrained volume integrals

Using the transformation $y = T_t(x)$ and the notation $F_t := \partial T_t = I + t\partial V$ for the Jacobian of the transformation T_t, we get for $\mathscr{J}$ as in (3.1),

$$\mathscr{J}(\Omega_t) = \int_{\Omega_t} f(x')\,\mathrm{d}x' = \int_{\Omega} (f \circ T_t)(x)\det(F_t(x))\,\mathrm{d}x. \tag{3.3}$$

Now let us explain how to compute the shape derivative of $\mathscr{J}$. Denoting

$$G(T_t, F_t) := \int_{\Omega} (f \circ T_t)(x)\det(F_t(x))\,\mathrm{d}x, \tag{3.4}$$

the chain rule gives (formally)

$$\frac{d}{dt}\mathscr{J}(\Omega_t)\bigg|_{t=0} = \frac{d}{dt}G(T_t, F_t)\bigg|_{t=0} = \left(\frac{dG}{dT_t}\frac{dT_t}{dt} + \frac{dG}{dF_t}\frac{dF_t}{dt}\right)\bigg|_{t=0}.$$

Using that $\frac{dT_t}{dt}(x) = V(x)$ and $\frac{dF_t}{dt}(x) = \partial V(x)$, we get for the shape derivative

$$d\mathscr{J}(\Omega)(V) = \frac{d}{dt}\mathscr{J}(\Omega_t)\bigg|_{t=0} = \left(\frac{dG}{dT_t}V + \frac{dG}{dF_t}\partial V\right)\bigg|_{t=0}.$$

This is the form we use for defining the first order shape derivative in `NGSolve`. Note that a Lipschitz vector field is differentiable almost everywhere and hence $\partial V(x)$ is defined almost everywhere and bounded.

Given the function $f(x_1, x_2) = (x_1 - 0.5)^2/a^2 + (x_2 - 0.5)^2/b^2 - R^2$ with $a = 1.3$, $b = 1/a$ and $R = 0.5$, we implement the transformed cost function (3.3) as follows:

```
40  f = ((X[0]-0.5)/1.3)**2+(1.3*(X[1]-0.5))**2 - 0.5**2
41
42  F = Id(2)                 # symbolic identity matrix
43  G_f = f * Det(F) * dx     # F only acts as a dummy variable
```

Here, we introduce the symbol F and assign to it the value of the identity matrix in line 42. This allows us to differentiate with respect to F. Then we define the function G of (3.4) in line 43. The shape derivative is a bounded linear functional on a space of vector fields. We introduce a vector-valued finite element space VEC and define the object representing the shape derivative dJOmega_f as a linear functional on VEC. In line 48, we differentiate with respect to the spatial variables in the direction given by V. Note that X is the coefficient function we introduced in line 13. In line 49, we deal with the differentiation with respect to F.

```
44  VEC = VectorH1(mesh, order=1, dirichlet="") #vectorial FE space of order 1
45  V = VEC.TestFunction()
46
47  dJOmega_f = LinearForm(VEC)
48  dJOmega_f += G_f.Diff(X, V)
49  dJOmega_f += G_f.Diff(F, grad(V))
```

Remark 3.2. Defining $\xi_t := \det(F_t)$ and using $\frac{d}{dt}\xi_t|_{t=0} = \operatorname{div} V$, it holds

$$\left.\frac{dG}{dF_t}\frac{dF_t}{dt}\right|_{t=0} = \left.\frac{dG}{d\xi_t}\frac{d\xi_t}{dF_t}\frac{dF_t}{dt}\right|_{t=0} = \left.\frac{dG}{d\xi_t}\frac{d\xi_t}{dt}\right|_{t=0} = \left.\frac{dG}{d\xi_t}\operatorname{div} V\right|_{t=0} = \int_\Omega f \operatorname{div} V \, dx.$$

Therefore, we obtain for the first order shape derivative the well-known formula

$$D\mathscr{J}(\Omega)(V) = \int_\Omega \nabla f \cdot V + f \operatorname{div} V \, dx.$$

Finally if Ω is smooth enough (for instance C^1), then the shape derivative is given by

$$D\mathscr{J}(\Omega)(V) = \int_{\partial\Omega} f V \cdot n \, ds, \tag{3.5}$$

where n denotes the outward pointing normal along $\partial\Omega$.

3.1.2 Shape differentiation of unconstrained boundary integrals

For Ω and f as in the previous section we consider

$$\mathscr{J}_{bnd}(\Omega) = \int_{\partial\Omega} f(x) \, dx. \tag{3.6}$$

Then we get

$$\mathscr{J}_{bnd}(\Omega_t) = \int_{\partial\Omega_t} f(x') \, ds_{x'} = \int_{\partial\Omega} (f \circ T_t)(x) \det(F_t(x)) |F_t(x)^{-\top} n(x)| \, ds_x, \tag{3.7}$$

see e.g. [32, Prop. 2.47], with the outer unit normal vector n and $|\cdot|$ denoting the Euclidean norm. Again, the shape derivative can be computed as the total derivative of this expression with respect to the parameter t. In NGSolve, the only difference lies in the necessity to use the trace of the gradient of a test vector field V.

```
50  G_f_bnd = f * Det(F) * Norm( Inv(F).trans*specialcf.normal(2) ) * ds
51
52  dJOmega_f_bnd = LinearForm(VEC)
53  dJOmega_f_bnd += G_f_bnd.Diff(X, V)                              # no trace needed
54  dJOmega_f_bnd += G_f_bnd.Diff(F, grad(V).Trace())               # trace needed
```

3.2   Second order shape derivatives

For second order shape derivatives, we consider perturbations of the form

$$T_{s,t}(x) = (\mathsf{Id} + sV + tW)(x), \quad x \in \mathbf{R}^d,$$

for $s, t \geq 0$ and define $\Omega_{s,t} := T_{s,t}(\Omega)$.

Definition 3.3. The second order shape derivative of a shape function $\mathscr{J}$ at Ω in direction $(V, W) \in C^{0,1}(\mathbf{R}^d)^d \times C^{0,1}(\mathbf{R}^d)^d$ is defined by

$$D^2 \mathscr{J}(\Omega)(V)(W) = \frac{d^2}{ds\,dt} \mathscr{J}(\Omega_{s,t}) \Big|_{s=t=0}. \tag{3.8}$$

Remark 3.4. We remark that if $\mathscr{J}$ is smooth enough, the second order derivative as defined in (3.8) is symmetric by definition:

$$D^2 \mathscr{J}(\Omega)(V)(W) = D^2 \mathscr{J}(\Omega)(W)(V). \tag{3.9}$$

We stress that this derivative is not the same as the shape derivative obtained by repeated shape differentiation, that is, it does not coincide with (see, e.g., [6, Chap. 9, Sec. 6])

$$d^2 \mathscr{J}(\Omega)(V)(W) := \lim_{t \searrow 0} \frac{D \mathscr{J}(T_t^W(\Omega))(V) - D \mathscr{J}(\Omega)(V)}{t} \tag{3.10}$$

which is in general asymmetric. However, in `NGSolve` we compute directly the second derivative as defined in (3.8). However, this derivative is only symmetric if $\partial V W = 0$ since

$$d^2 \mathscr{J}(\Omega)(V)(W) = D^2 \mathscr{J}(\Omega)(V)(W) + D \mathscr{J}(\Omega)(\partial V W). \tag{3.11}$$

In `NGSolve`, when repeating the shape differentiation procedure introduced in Section 3.1, we compute directly the second order shape derivative as defined in (3.8). Here, we exploit that `GridFunctions` are independent of the spatial coordinates, see also Section 2.2.

Similarly to the computations of the first derivative, we use the notation $F_{s,t} := \partial T_{s,t} = I + s\partial V + t\partial W$. For the shape function defined in (3.1) we get

$$\frac{d^2}{ds\,dt} \mathscr{J}(\Omega_{s,t}) \Big|_{s=t=0} = \frac{d^2}{ds\,dt} \int_{\Omega_{s,t}} f(x)\,dx \Big|_{s=t=0}$$

$$= \frac{d^2}{ds\,dt} \int_{\Omega} (f \circ T_{s,t})(x) \det(F_{s,t}(x))\,dx \Big|_{s=t=0}.$$

Again, using the notation $G(T_{s,t}, F_{s,t}) = \int_\Omega (f \circ T_{s,t})(x) \det(F_{s,t}(x)) \, dx$, we get

$$\frac{d^2}{dsdt} \mathcal{J}(\Omega_{s,t}) \bigg|_{s=t=0} = \frac{d^2}{dsdt} G(T_{s,t}, F_{s,t}) \bigg|_{s=t=0} = \frac{d}{ds}\left(\frac{dG}{dT_{s,t}} \frac{dT_{s,t}}{dt} + \frac{dG}{dF_{s,t}} \frac{dF_{s,t}}{dt} \right)\bigg|_{s=t=0}.$$

Using that $\frac{d^2 T_{s,t}}{dsdt} = 0$ and $\frac{d^2 F_{s,t}}{dsdt} = 0$, we get further

$$\frac{d^2}{dsdt} \mathcal{J}(\Omega_{s,t}) \bigg|_{s=t=0} = \frac{d}{ds}\left(\frac{dG}{dT_{s,t}} \right)\frac{dT_{s,t}}{dt} + \frac{d}{ds}\left(\frac{dG}{dF_{s,t}} \right)\frac{dF_{s,t}}{dt}\bigg|_{s=t=0}$$

$$= \left(\frac{d^2 G}{dT_{s,t}^2} \frac{dT_{s,t}}{ds} + \frac{d^2 G}{dF_{s,t}dT_{s,t}} \frac{dF_{s,t}}{ds} \right)\frac{dT_{s,t}}{dt} + \left(\frac{d^2 G}{dT_{s,t}dF_{s,t}} \frac{dT_{s,t}}{ds} + \frac{d^2 G}{dF_{s,t}^2} \frac{dF_{s,t}}{ds} \right)\frac{dF_{s,t}}{dt}\bigg|_{s=t=0}.$$

(3.12)

Formula (3.12) is used for the automatic derivation of the second order shape derivative in NGSolve. Using $\frac{dT_{s,t}}{ds}(x) = V(x)$, $\frac{dT_{s,t}}{dt}(x) = W(x)$ and $\frac{dF_{s,t}}{ds}(x) = \partial V(x)$, $\frac{dF_{s,t}}{dt}(x) = \partial W(x)$, we get

$$\frac{d^2}{dsdt} \mathcal{J}(\Omega_{s,t}) \bigg|_{s=t=0} = \left(\frac{d^2 G}{dT_{s,t}^2} V + \frac{d^2 G}{dF_{s,t}dT_{s,t}} \partial V \right) W + \left(\frac{d^2 G}{dT_{s,t}dF_{s,t}} V + \frac{d^2 G}{dF_{s,t}^2} \partial V \right) \partial W \bigg|_{s=t=0}.$$

(3.13)

Remark 3.5. We remark that the formula (3.13) can be evaluated explicitly and reads

$$D^2 \mathcal{J}(\Omega)(V, W) = \int_\Omega \nabla^2 f V \cdot W + \nabla f \cdot W \operatorname{div} V + \nabla f \cdot V \operatorname{div} W + f \operatorname{div} V \operatorname{div} W - f \partial V^\top : \partial W \, dx.$$

Formula (3.13) can be implemented in NGSolve as follows:

```
55  d2JOmega_f = BilinearForm(VEC)
56  W = VEC.TrialFunction()
57
58  d2JOmega_f+=(G_f.Diff(X, W)+G_f.Diff(F, grad(W))).Diff(X, V)
59  d2JOmega_f+=(G_f.Diff(X, W)+G_f.Diff(F, grad(W))).Diff(F, grad(V))
```

Notice that since W is a trial function it is not affected by the differentiation with respect to X, see Section 2.2. In the same fashion, second order derivatives of boundary integrals of the form (3.6) can be computed.

```
60  d2JOmega_f_bnd = BilinearForm(VEC)
61
62  d2JOmega_f_bnd+=(G_f_bnd.Diff(X, W)+G_f_bnd.Diff(F,grad(W).Trace())).Diff(X, V)
63  d2JOmega_f_bnd+=(G_f_bnd.Diff(X, W)+G_f_bnd.Diff(F,grad(W).Trace())).Diff(F,grad(V
       ).Trace())
```

4 Semi-automatic shape differentiation with PDE constraints

In this section, we describe the automatic derivation of the shape derivative for the following type of equality constrained shape optimisation problems:

$$\min_{(\Omega, u)} J(\Omega, u)$$

(4.1)

subject to $(\Omega, u) \in \mathscr{A} \times X$ solves

$$e(\Omega, u) = 0, \tag{4.2}$$

where $e : \mathscr{A} \times X \to X^*$ with $e(\Omega, \cdot) : X(\Omega) \to X(\Omega)^*$ represents an abstract PDE constraint with $X = \cup_{\Omega \in \mathscr{A}} X(\Omega)$ being the union of Banach spaces $X(\Omega)$ and $\mathscr{A}$ a set of admissible shapes. For any given $\Omega \in \mathscr{A}$ we assume the PDE constraint (4.2) to admit a unique solution which we denote by u_Ω. Moreover, let $\mathscr{J}(\Omega) := J(\Omega, u_\Omega)$ denote the reduced cost functional. One way to eliminate the constraint (4.2) is to introduce a Lagrangian

$$\mathscr{L}(\Omega, u, p) := J(\Omega, u) + \langle e(\Omega, u), p \rangle. \tag{4.3}$$

Now an initial shape Ω is perturbed by a family of transformations T_t, resulting in a new shape $\Omega_t := T_t(\Omega)$. Transforming back to the initial shape Ω leads to the Lagrangian:

$$G(t, u, p) := \mathscr{L}(T_t(\Omega), \Phi_t(u), \Phi_t(p)), \quad u, p \in X(\Omega), \tag{4.4}$$

where $\Phi_t : X(\Omega) \to X(\Omega_t)$ is a bijective mapping. Here the transformation Φ_t depends on the differential operator involved. For instance

- if $X(\Omega) = H_0^1(\Omega)$, then $\Phi_t(u) = u \circ T_t^{-1}$,

- if $X(\Omega) = H(\mathbf{curl}, \Omega)$, then $\Phi_t(u) = \partial T_t^{-\top}(u \circ T_t^{-1})$,

- if $X(\Omega) = H(\mathrm{div}, \Omega)$, then $\Phi_t(u) = \frac{1}{\det(\partial T_t)} \partial T_t(u \circ T_t^{-1})$.

Now the shape differentiability of (4.1)–(4.2) is reduced to proving that (see [34])

$$D\mathscr{J}(\Omega)(V) = \frac{d}{dt} G(t, u^t, 0)|_{t=0} = \partial_t G(0, u, p), \tag{4.5}$$

where $u^t := u_t \circ T_t$ and $u_t \in X(\Omega_t)$ solves $e(\Omega_t, u_t) = 0$. The verification of (4.5) can be accomplished by different methods, see e.g. [34] for an overview. However, we remark that (4.5) holds for a large class of nonlinear PDE constrained shape optimisation; see [33].

The rest of this section is organised as follows: We introduce a model problem, which is the minimisation of a tracking-type cost functional subject to Poisson's equation in Section 4.1. We illustrate how the first and second order shape derivative for this PDE-constrained model problem can be obtained in `NGSolve` in Sections 4.2 and 4.3. Finally, we also briefly discuss the extension to partial differentiation equations on surfaces.

4.1 PDE-constrained model problem

We will illustrate the derivation of the first and second order shape derivative for the minimisation of a tracking-type cost functional subject to Poisson's equation on the unknown domain Ω. Let $d = 2$ or 3, $f, u_d \in H^1(\mathbf{R}^d)$ and $\mathscr{A} \subset \mathscr{P}(\mathbf{R}^d)$ a set of admissible shapes. We consider the problem

$$\min_{(\Omega, u)} J(\Omega, u) = \int_\Omega |u - u_d|^2 \, \mathrm{d}x \tag{4.6a}$$

subject to $(\Omega, u) \in \mathscr{A} \times H_0^1(\Omega)$ solve

$$\langle e(\Omega, u), \psi \rangle := \int_\Omega \nabla u \cdot \nabla \psi \, dx - \int_\Omega f \psi \, dx = 0 \quad \text{for all } \psi \in H_0^1(\Omega). \tag{4.6b}$$

The Lagrangian is given by

$$\mathscr{L}(\Omega, \varphi, \psi) := \int_\Omega |\varphi - u_d|^2 \, dx + \int_\Omega \nabla \varphi \cdot \nabla \psi \, dx - \int_\Omega f \psi \, dx. \tag{4.7}$$

Given an admissible shape Ω, a vector field $V \in C^{0,1}(\mathbf{R}^d)^d$ and $t > 0$ small, let $\Omega_t := (\mathsf{Id} + tV)(\Omega)$ the perturbed domain. Therefore the parametrised Lagrangian is given by

$$G(t, \varphi, \psi) := \mathscr{L}(T_t(\Omega), \varphi \circ T_t^{-1}, \psi \circ T_t^{-1}), \quad \varphi, \psi \in H_0^1(\Omega). \tag{4.8}$$

Changing variables yields

$$G(t, \varphi, \psi) = \int_\Omega |\varphi - u_d^t|^2 \det(F_t) dx + \int_\Omega (F_t^{-\top} \nabla \varphi) \cdot (F_t^{-\top} \nabla \psi) \det(F_t) \, dx - \int_\Omega f^t \psi \det(F_t) \, dx \tag{4.9}$$

$$=: \tilde{G}(T_t, F_t, \varphi, \psi),$$

where $u_d^t = u_d \circ T_t$ and $f^t = f \circ T_t$. Recall that, for a given $\Omega \in \mathscr{A}$, u_Ω denotes the corresponding unique solution to (4.6b) and $\mathscr{J}(\Omega)$ the reduced cost functional, $\mathscr{J}(\Omega) := J(\Omega, u_\Omega)$. Let $u^t \in H_0^1(\Omega)$ the solution of the perturbed state equation brought back to the original domain Ω, that is, $u^t \in H_0^1(\Omega)$ is the unique solution to

$$\partial_\psi G(t, u^t, 0)(\psi) = 0 \quad \text{for all } \psi \in H_0^1(\Omega). \tag{4.10}$$

Note that, for u^t defined by (4.10), it holds $\mathscr{J}(\Omega_t) = G(t, u^t, \psi)$ for all $\psi \in H_0^1(\Omega)$ and therefore also $D\mathscr{J}(\Omega)(V) = \frac{d}{dt} G(t, u^t, \psi)$ for all $\psi \in H_0^1(\Omega)$.

It can easily be shown that (4.5) holds and thus the shape derivative in the direction of a vector field $V \in C^{0,1}(\mathbf{R})^d$ is given by

$$D\mathscr{J}(\Omega)(V) = \partial_t G(0, u, p),$$

where $p \in H_0^1(\Omega)$ denotes the adjoint state and is defined as the unique solution $p \in H_0^1(\Omega)$ to

$$\partial_\varphi G(0, u, p)(\hat{\varphi}) = 0 \quad \text{for all } \hat{\varphi} \in H_0^1(\Omega), \tag{4.11}$$

or explicitly

$$\int_\Omega \nabla \hat{\varphi} \cdot \nabla p \, dx = -2 \int_\Omega (u - u_d) \hat{\varphi} \, dx \quad \text{for all } \hat{\varphi} \in H_0^1(\Omega). \tag{4.12}$$

4.2 First order shape derivative

By the discussion above, the first order shape derivative is given by $\partial_t G(0, u, p)$ with G defined in (4.9) and u and p the unique solutions to the boundary value problems (4.6b) and (4.12), respectively.

Writing $\tilde{G}(T_t, F_t) := \tilde{G}(T_t, F_t, u, p) = G(t, u, p)$ we obtain in analogy to the unconstrained problem

$$D\mathscr{J}(\Omega)(V) = \frac{d}{dt}\mathscr{J}(\Omega_t)\Big|_{t=0} = \left(\frac{dG}{dT_t}V + \frac{dG}{dF_t}\partial V\right)\Big|_{t=0}.$$

We can compute explicitly

$$\frac{dG}{dF_t}\Big|_{t=0}\partial V = \int_\Omega \text{div}(V)(u-u_d)^2 - (\partial V + \partial V^\top)\nabla u \cdot \nabla p - \text{div}(V)\nabla u \cdot \nabla p - f p \, \text{div}(V) \, dx,$$
(4.13)

$$\frac{dG}{dT_t}\Big|_{t=0}V = \int_\Omega -2(u-u_d)\nabla u_d \cdot V - \nabla f \cdot V p \, dx.$$
(4.14)

Now we are in a position to compute the first order shape derivative for the PDE-constrained shape optimisation problem (4.6) in NGSolve. After solving the state equation as shown in Section 2.1, the adjoint equation can be solved as follows.

```
ud = X[0]*(1-X[0])*X[1]*(1-X[1])
def Cost(u):
    return (u-ud)**2 * Det(F) * dx

#solve adjoint equation
gfp = GridFunction(fes)
dCostdu = LinearForm(fes)
dCostdu += Cost(gfu).Diff(gfu, w)
dCostdu.Assemble()
gfp.vec.data = -a.mat.Inverse(fes.FreeDofs(), inverse="sparsecholesky").T * 
    dCostdu.vec

Draw(gfp, mesh, "adjoint")
```

We can now define the Lagrangian (4.9) such that the shape derivative can be obtained by the same procedure as in the unconstrained setting. Note that lines 82–83 coincide with lines 48–49.

```
def Equation(u,w):
    return ((Inv(F).trans * grad(u)) * (Inv(F).trans * grad(w)) - f*w)*Det(F)*dx

G_pde = Cost(gfu) + Equation(gfu, gfp)

dJOmega_pde = LinearForm(VEC)
dJOmega_pde += G_pde.Diff(X, V)
dJOmega_pde += G_pde.Diff(F, grad(V))
```

4.3   Second order shape derivative

Let us introduce the notation

$$G_{V,W}(s,t,\varphi,\psi) = \underbrace{\int_\Omega (F_{s,t}^{-\top}\nabla\varphi)\cdot(F_{s,t}^{-\top}\nabla\psi)\det(F_{s,t})\,\mathrm{d}x - \int_\Omega f\circ T_{s,t}\psi\det(F_{s,t})\,\mathrm{d}x}_{=:\langle E_{V,W}(s,t)\varphi,\psi\rangle} \qquad (4.15)$$

$$+\underbrace{\int_\Omega |\varphi - u_d\circ T_{s,t}|^2\det(F_{s,t})\mathrm{d}x}_{=:J_{V,W}(s,t)}, \qquad (4.16)$$

where $T_{s,t}(x) = x + sV(x) + tW(x)$ and $F_{s,t} := \partial T_{s,t}$. We observe that

$$\mathscr{J}(T_{s,t}(\Omega)) = G_{V,W}(s,t,u^{s,t},p^{s,t}) \qquad (4.17)$$

with $(u^{s,t}, p^{s,t}) \in H_0^1(\Omega)\times H_0^1(\Omega)$ being the solution to

$$\begin{aligned}\partial_p G_{V,W}(s,t,u^{s,t},0)(\varphi) &= 0 \quad \text{for all } \varphi \in H_0^1(\Omega),\\ \partial_u G_{V,W}(s,t,u^{s,t},p^{s,t})(\psi) &= 0 \quad \text{for all } \psi \in H_0^1(\Omega)\end{aligned} \qquad (4.18)$$

for $s, t \geq 0$. In case $t = 0$ we write $u^s := u^{s,t}|_{t=0}$ and $p^s := p^{s,t}|_{t=0}$ and similarly for $t = s = 0$ we write $u := u^{s,t}|_{s=t=0}$ and $p := p^{s,t}|_{s=t=0}$. Therefore, consecutive differentiation of (4.17) first with respect to t at zero and then with respect to s at zero yields

$$\begin{aligned}D^2\mathscr{J}(\Omega)(V)(W) &= \frac{d^2}{dsdt}G_{V,W}(s,t,u^{s,t},p^{s,t})|_{s=t=0} = \frac{d}{ds}\partial_t G_{V,W}(s,0,u^s,p^s)|_{s=0}\\ &= \partial_s\partial_t G_{V,W}(0,0,u,p) + \partial_u\partial_t G_{V,W}(0,0,u,p)(\partial_s u^0) + \partial_p\partial_t G_{V,W}(0,0,u,p)(\partial_s p^0),\end{aligned} \qquad (4.19)$$

where $\partial_s u^0 \in H_0^1(\Omega)$ solves the material derivative equation

$$\partial_u\partial_p G_{V,W}(0,0,u,0)(\psi)(\partial_s u^0) = -\partial_s\partial_p G_{V,W}(0,0,u,0)(\psi) \quad \text{for all } \psi \in H_0^1(\Omega), \qquad (4.20)$$

or equivalently

$$\langle \partial_u E_{V,W}(0,0)(\partial_s u^0),\psi\rangle = -\langle \partial_s E_{V,W}(0,0)u,\psi\rangle \quad \text{for all } \psi \in H_0^1(\Omega). \qquad (4.21)$$

Similarly the function $\partial_s p^0 \in H_0^1(\Omega)$ solves the material derivative equation

$$\partial_p\partial_u G_{V,W}(0,0,u,p)(\psi)(\partial_s p^0) = -\partial_u^2 G_{V,W}(0,0,u,p)(\psi)(\partial_s u^0) - \partial_s\partial_u G_{V,W}(0,0,u,p)(\psi) \qquad (4.22)$$

for all $\psi \in H_0^1(\Omega)$. Formally (4.20) and (4.22) can be written as an operator equation

$$\begin{pmatrix} \partial_u^2 G_{V,W}(0,0,u,p) & \partial_p\partial_u G_{V,W}(0,0,u,p) \\ \partial_u\partial_p G_{V,W}(0,0,u,p) & 0 \end{pmatrix}\begin{pmatrix} \partial_s u^0 \\ \partial_s p^0 \end{pmatrix} = -\begin{pmatrix} \partial_s\partial_u G_{V,W}(0,0,u,p) \\ \partial_s\partial_p G_{V,W}(0,0,u,0) \end{pmatrix}. \qquad (4.23)$$

So to evaluate the second derivative (4.19) in some direction (V,W) we have to solve the system (4.23).

   This is realised in NGSolve by setting up a combined finite element space which we denote by X2. We define trial and test functions as well as grid functions representing the deformation vector fields V and W, which we initialise with some functions.

```
84 X2 = FESpace([ fes , fes ])
85 dsu , dsp = X2 . TrialFunction ()
86 uTest , pTest = X2 . TestFunction ()
87 gfV = GridFunction (VEC)
88 gfW = GridFunction (VEC)
89 gfV . Set ((X[0]∗X[0]∗X[1]∗ exp (X[1]) ,X[1]∗X[1]∗X[0]∗ exp (X[0])))
90 gfW . Set ((X[1]∗X[1]∗X[0]∗ exp (X[0]) ,X[0]∗X[0]∗X[1]∗ exp (X[1])))
```

We define a 2×2 block bilinear form as well as a 2×1 block linear form which will represent the
left and right hand sides of (4.23), respectively. The operator equation in (4.23) can be conve-
niently defined by differentiating the Lagrangian with respect to the corresponding variables.

```
 91 shapeHessLag2 = BilinearForm (X2)
 92 shapeGradLag2 = LinearForm (X2)
 93
 94 shapeHessLag2 += (G_pde. Diff (gfu , uTest )). Diff (gfu , dsu)       #block (1,1)
 95 shapeHessLag2 += (G_pde. Diff (gfu , uTest )). Diff (gfp , dsp)       #block (1 ,2)
 96 shapeHessLag2 += (G_pde. Diff (gfp , pTest )). Diff (gfu , dsu)       #block (2 ,1)
 97
 98 #line 1
 99 shapeGradLag2 += (G_pde. Diff (gfu , uTest )). Diff (F, grad (gfV))
100 shapeGradLag2 += (G_pde. Diff (gfu , uTest )). Diff (X, gfV)
101
102 #line 2
103 shapeGradLag2 += (G_pde. Diff (gfp ,pTest )). Diff (F, grad (gfV))
104 shapeGradLag2 += (G_pde. Diff (gfp ,pTest )). Diff (X, gfV)
```

We can solve this combined system for $\partial_s u^0$ and $\partial_s p^0$ and access and visualise the two compo-
nents in the following way:

```
105 gfCombined2 = GridFunction (X2)
106 shapeHessLag2 . Assemble ()
107 shapeGradLag2 . Assemble ()
108 gfCombined2 . vec . data = shapeHessLag2 . mat . Inverse (X2 . FreeDofs () , inverse = "umfpack
        ") ∗ shapeGradLag2 . vec
109
110 gfdsu = GridFunction ( fes )
111 gfdsp = GridFunction ( fes )
112 gfdsu . vec . data = gfCombined2 . components [ 0 ] . vec
113 gfdsp . vec . data = gfCombined2 . components [ 1 ] . vec
114
115 Draw ( gfdsu , mesh , "dsu")
116 Draw ( gfdsp , mesh , "dsp")
```

In order to obtain the second order shape derivative in the direction given by (V, W), it remains
to evaluate the term (4.19). We define the three terms of (4.19) as bilinear forms, assemble
them and perform vector-matrix-vector multiplications:

```
117 w1 = fes . TrialFunction ()
118 q1 = fes . TrialFunction ()
119
120 shapeHess11 = BilinearForm (VEC)
121 shapeHess11 += (G_pde. Diff (F, grad (W))+G_pde. Diff (X, W)). Diff (F, grad (V))
```

```
122  shapeHess11 += (G_pde.Diff(F,grad(W))+G_pde.Diff(X, W)).Diff(X, V)
123  shapeHess11.Assemble()
124
125  shapeHess12 = BilinearForm(trialspace = fes, testspace = VEC)
126  shapeHess12 += (G_pde.Diff(F, grad(V)) + G_pde.Diff(X, V)).Diff(gfu,w1)
127  shapeHess12.Assemble()
128
129  shapeHess13 = BilinearForm(trialspace = fes, testspace = VEC)
130  shapeHess13 += (G_pde.Diff(F, grad(V)) + G_pde.Diff(X, V)).Diff(gfp,q1)
131  shapeHess13.Assemble()
132
133  av = gfV.vec.CreateVector()
134  av.data = shapeHess11.mat * gfV.vec
135
136  adsu = gfV.vec.CreateVector()
137  adsu.data = shapeHess12.mat * gfdsu.vec
138
139  adsp = gfV.vec.CreateVector()
140  adsp.data = shapeHess13.mat * gfdsp.vec
141
142  d2J = InnerProduct(gfW.vec, av) + InnerProduct(gfW.vec, adsu) + InnerProduct(gfW.
         vec, adsp)
```

4.4 PDEs on surfaces

The automated shape differentiation is not restricted to partial differential equations on domains Ω, but is readily extended to surface PDEs. We consider a two dimensional closed surface $M \subset \mathbf{R}^3$ and denote by n the normal field along M. Let $u_d \in H^1(\mathbf{R}^d)$ be given and define

$$J(M, u) = \int_M |u - u_d|^2 \, \mathrm{d}s, \tag{4.24}$$

where $u \in H^1(M)$ solves the surface equation

$$\int_M \nabla^M u \cdot \nabla^M \varphi + u\varphi \, \mathrm{d}s = \int_M f \varphi \, \mathrm{d}s \quad \text{for all } \varphi \in H^1(M), \tag{4.25}$$

where $\nabla^M \varphi$ denotes the tangential gradient of φ; see [6, p.493, Def.5.1]. We assume that the function $f \in H^1(\mathbf{R}^3)$ is given. The Lagrangian is given by

$$\mathscr{L}(M, \varphi, \psi) := \int_M |\varphi - u_d|^2 \, \mathrm{d}s + \int_M \nabla^M \varphi \cdot \nabla^M \psi + \varphi\psi \, \mathrm{d}s - \int_M f \psi \, \mathrm{d}s. \tag{4.26}$$

As in the previous section we fix an admissible shape M and let $M_t := (\mathsf{Id} + tV)(M)$ be a small perturbation of M by means of a vector field $V \in C^1(\mathbf{R}^d)^d$ for $t > 0$ small. The parametrised Lagrangian is given by

$$G(t, \varphi, \psi) := \mathscr{L}(T_t(M), \varphi \circ T_t^{-1}, \psi \circ T_t^{-1}), \quad \varphi, \psi \in H^1(M). \tag{4.27}$$

Define the density $\omega(F_t) := \det(F_t)|F_t^{-\top} n|$. Changing variables and using

$$(\nabla^{M_t} \varphi) \circ T_t = B(F_t) \nabla^M (\varphi \circ T_t), \quad B(F_t) = \left(I - \frac{F_t^{-\top} n}{|F_t^{-\top} n|} \otimes \frac{F_t^{-\top} n}{|F_t^{-\top} n|} \right) F_t^{-\top} \tag{4.28}$$

yields

$$G(t, \varphi, \psi) = \int_M |\varphi - u_d^t|^2 \, \omega(F_t) \, \mathrm{d}s$$
$$+ \int_M ((B(F_t)\nabla\varphi) \cdot (B(F_t)\nabla\psi) + \varphi\psi) \, \omega(F_t) \, \mathrm{d}s - \int_M f^t \psi \, \omega(F_t) \, \mathrm{d}s, \qquad (4.29)$$

where $u_d^t = u_d \circ T_t$ and $f^t = f \circ T_t$.

Writing $\tilde{G}(T_t, F_t) := G(t, u, p)$ we obtain in analogy to the domain case

$$D\mathscr{J}(\Omega)(V) = \left(\frac{dG}{dT_t} V + \frac{dG}{dF_t} \partial V \right)\Bigg|_{t=0}. \qquad (4.30)$$

We can compute explicitly

$$\frac{dG}{dF_t}\big|_{t=0} V = \int_M \mathrm{div}^M(V)(u - u_d)^2 - (\partial^M V + \partial^M V^\top)\nabla^M u \cdot \nabla^M p$$
$$+ \mathrm{div}^M(V)(\nabla^M u \cdot \nabla^M p + up) - f p \, \mathrm{div}^M(V) \, \mathrm{d}s, \qquad (4.31)$$

$$\frac{dG}{dT_t}\big|_{t=0} \partial V = \int_M -2(u - u_d)\nabla u_d \cdot V - \nabla f \cdot V p \, \mathrm{d}s, \qquad (4.32)$$

where $\partial^M V$ denotes the tangential Jacobian of V and $\mathrm{div}^M(V) := \partial^M V : I$ the tangential divergence, which is defined as the trace of the tangential Jacobian; see [6, p.495].

The implementation is analogous to the previous sections. We will only illustrate first order derivatives here. We first define the geometry of the unit sphere, create a surface mesh and define a finite element space on the surface mesh:

```
143 from netgen.csg import *
144 from netgen.meshing import *
145 from ngsolve.internal import visoptions
146 from ngsolve import *
147
148 geo_surf = CSGeometry()
149 sphere = Sphere(Pnt(0,0,0),1).bc("outer")
150 geo_surf.Add(sphere)
151 mesh_surf = Mesh(geo_surf.GenerateMesh(perfstepsend=MeshingStep.MESHSURFACE,
        optsteps2d=3,maxh=0.2))
152 mesh_surf.Curve(3)
153 fes_surf = H1(mesh_surf,order = 3)
```

Next we define the transformed cost function and partial differential equation needed for setting up the Lagrangian (4.29). Here, we again make use of a symbolic object F to which we assign the identity matrix. We define the tangential determinant ω and the matrix B defined in (4.28) as functions of the deformation gradient F_t.

```
154 X = CoefficientFunction((x,y,z))
155 func = CoefficientFunction(X[0]*X[1]*X[2])
156 F = Id(3)
157 tangDet = Det(F) * Norm( Inv(F).trans*specialcf.normal(3) )
158 Bmat = (Id(3) - 1/Norm(Inv(F).trans*specialcf.normal(3))**2 * OuterProduct(Inv(F).
        trans*specialcf.normal(3), Inv(F).trans*specialcf.normal(3)) ) * Inv(F).trans
```

```
159
160  def Equation_surf(u,w):
161      return ( (Bmat*grad(u).Trace()) * (Bmat*grad(w).Trace()) + u*w − func * w) *
         tangDet * ds
162
163  def Cost_surf(u):
164      return u**2 * tangDet * ds
```

Now we can define the bilinear form and solve the state equation. Here, the right hand side of the equation is included in the bilinear form and the boundary value problem – although linear – is solved by Newton's method (which terminates after only one iteration) for convenience.

```
165  #set up and solve state equation
166  u_surf, w_surf = fes_surf.TnT()
167  a = BilinearForm(fes_surf)
168  a += Equation_surf(u_surf, w_surf)
169  gfu_surf = GridFunction(fes_surf)
170  solvers.Newton(a, gfu_surf, printing = False)
171  Draw(gfu_surf, mesh_surf, "gfu_surf")
```

The adjoint equation is solved as usual:

```
172  #solve adjoint equation
173  lfcost_surf = LinearForm(fes_surf)
174  lfcost_surf += Cost_surf(gfu_surf).Diff(gfu_surf, w_surf)
175  lfcost_surf.Assemble()
176  inva = a.mat.Inverse(fes_surf.FreeDofs(), inverse="sparsecholesky")
177  gfp_surf = GridFunction(fes_surf)
178  gfp_surf.vec.data = −inva.T * lfcost_surf.vec
179  Draw(gfp_surf, mesh_surf, "gfp_surf")
```

The shape derivative is obtained as in the case of PDEs posed on volumes by the evaluation of (4.30):

```
180  G_surf = Cost_surf(gfu_surf) + Equation_surf(gfu_surf, gfp_surf)
181
182  VEC3d = VectorH1(mesh_surf, order=1)
183  V3d = VEC3d.TestFunction()
184  dJOmega_surf = LinearForm(VEC3d)
185  dJOmega_surf += G_surf.Diff(X, V3d) + G_surf.Diff(F, Grad(V3d).Trace())
```

5 Fully automated shape differentiation

In the previous sections we used the automatic differentiation capabilities of NGSolve to alleviate the shape differentiation procedure. However, so far we still had to include some knowledge about the problems at hand. So far, it was necessary to define the objective function or Lagrangian G in the correct way, accounting for the correct transformation rules between perturbed and unperturbed domain. In this section, we will show that also this step can be auto-

mated since all necessary information is already included in the functional setting. The fully automated shape differentiation is incorporated by the command

$$\text{DiffShape}(\ldots).$$

In particular, in the fully automated setting it is enough to set up the cost function or Lagrangian for the unperturbed setting. For a shape function of the type (3.1) we can define the shape derivative of the cost function in the following way:

```
186  G_f_0 = f * dx
187  dJOmega_f_0 = LinearForm(VEC)
188  dJOmega_f_0 += G_f_0.DiffShape(V)
```

Note that there is no term of the form Det(F) showing up in line 186. Here, the transformation of the domain is taken care of automatically. It can be checked that this really gives the same result as dJOmega_f defined in lines 48–49.

```
189  dJOmega_f.Assemble()
190  dJOmega_f_0.Assemble()
191  differenceVec = dJOmega_f.vec.CreateVector()
192  differenceVec.data = dJOmega_f.vec − dJOmega_f_0.vec
193  print("|dJOmega_f − dJOmega_f_0| = ", Norm(differenceVec) )
```

The above code gives the output

```
|dJOmega_f - dJOmega_f_0| = 1.571008573810619e-17
```

which confirms our claim. The same holds true for second order shape derivatives. The lines 58–59 can be replaced by a repeated call of DiffShape(...):

```
194  d2JOmega_f_0 = BilinearForm(VEC)
195  d2JOmega_f_0 += G_f_0.DiffShape(V).DiffShape(W)
```

Again, it can be verified that d2JOmega_f_0 coincides with the previously defined quantity d2JOmega_f. Note that slightly different results may occur due to different integration rules used. This can be cured by enforcing an integration rule of higher order for G_f, i.e. by replacing the symbol dx in the definition of G_f with dx(bonus_intorder=2).

In the more general setting of PDE-constrained shape optimisation, the procedure is very similar. Here the idea exploited in the implementation of the command DiffShape(...) is to just differentiate the general expression (4.4) with respect to the parameter t. The transformations Φ_t appearing in (4.4), which depend on the functional setting of the PDE, are identified automatically from the finite element space from which the corresponding functions originate. The shape derivative of lines 82–83 can be obtained by the following code.

```
196  def Cost_0(u):
197      return (u−ud)**2 * dx
198
199  def Equation_0(u,w):
200      return (grad(u) * grad(w) − f1*w) *dx
201
202  G_pde_0 = Cost_0(gfu) + Equation_0(gfu, gfp)
203
```

```
204  dJOmega_pde_0 = LinearForm (VEC)
205  dJOmega_pde_0 += G_pde_0 . DiffShape (V)
```

Here, `gfu` and `gfp` represent the solutions to the state and adjoint equation, respectively, and must have been computed previously. The bilinear form `shapeHess11` used in Section 4.3 (see lines 121–122) can be obtained similarly:

```
206  shapeHess11_0 = BilinearForm (VEC)
207  shapeHess11_0 += G_pde_0 . DiffShape (W) . DiffShape (V)
```

The same holds true for boundary integrals

```
208  G_f_bnd_0 = f * ds
209  dJOmega_f_bnd_0 = LinearForm (VEC)
210  dJOmega_f_bnd_0 += G_f_bnd_0 . DiffShape (V)
```

and surface PDEs

```
211  def Cost_surf_0 (u):
212      return u**2 * ds
213  def Equation_surf_0 (u,w):
214      return  (grad (u). Trace ()*grad (w). Trace () + u*w − func * w) * ds
215  G_surf_0 = Cost_surf_0 (gfu_surf) + Equation_surf_0 (gfu_surf, gfp_surf)
216  dJOmega_surf_0 = LinearForm (VEC3d)
217  dJOmega_surf_0 += G_surf_0 . DiffShape (V3d)
```

as well as their respective second order derivatives.

Remark 5.1. We remark that the fully automated differentiation using `DiffShape (...)` should be seen to complement the semi-automated shape differentiation techniques introduced in Sections 3 and 4 rather than to replace them. Using the semi-automated differentiation, the user has the possibility to, on the one hand, keep control over the involved terms, and on the other hand also to adjust the shape differentiation to their custom problems which may be non-standard. As an example where the semi-automated differentiation may be beneficial compared to the fully automated differentiation we mention the case of time-dependent PDE constraints considered in a space-time setting when a shape deformation is only desired in the spatial coordinates. Of course, when one is interested in the shape derivative for a more standard problem, the fully automated way appears to be more convenient and less error prone.

6 Numerical Experiments

6.1 Code verification

We verify the expressions that we obtained in a semi-automatical or fully automatical way for the first and second order shape derivatives by looking at the Taylor expansions of the perturbed shape functionals. We illustrate our findings in two examples in $\mathbf{R}^2$. On the one hand, we consider a shape function as introduced in (3.1) with an additional boundary integral as in (3.6), henceforth denoted by $\mathscr{J}_1$; on the other hand, we consider the PDE-constrained shape optimisation problem defined by (4.6), the reduced form of which will be denoted by $\mathscr{J}_2(\Omega)$. More

precisely, we consider

$$\mathscr{J}_1(\Omega) = \int_\Omega f(x)\,\mathrm{d}x + \int_{\partial\Omega} f(x)\,\mathrm{d}s, \tag{6.1}$$

$$\mathscr{J}_2(\Omega) = \int_\Omega |u_\Omega - u_d|^2\,\mathrm{d}x \quad \text{where } u_\Omega \text{ solves (4.6b).} \tag{6.2}$$

In the case of $\mathscr{J}_1$, we used the function $f(x_1,x_2) = \left(0.5 + \sqrt{x_1^2 + x_2^2}\right)^2 \left(0.5 - \sqrt{x_1^2 + x_2^2}\right)^2$ and for $\mathscr{J}_2$, we used $u_d(x_1,x_2) = x_1(1-x_1)x_2(1-x_2)$ and $f(x_1,x_2) = 2x_2(1-x_2) + 2x_1(1-x_1)$ for the function f in the PDE constraint (4.6b).

For the test of the first order shape derivatives $D\mathscr{J}_i(\Omega)(V)$ we choose a fixed shape Ω and a vector field $V \in C^{0,1}(\mathbf{R}^2)^2$ and observe the quantity

$$\delta_1(\mathscr{J}_i, t) := |\mathscr{J}_i((\mathrm{Id} + tV)(\Omega)) - \mathscr{J}_i(\Omega) - t\,D\mathscr{J}_i(\Omega)(V)|, \tag{6.3}$$

for $t \searrow 0$. Likewise, for the second order shape derivative, we consider the remainder

$$\delta_2(\mathscr{J}_i, t) := \left|\mathscr{J}_i((\mathrm{Id} + tV)(\Omega)) - \mathscr{J}_i(\Omega) - t\,D\mathscr{J}_i(\Omega)(V) - \frac{1}{2}t^2 D^2\mathscr{J}_i(\Omega)(V)(V)\right| \tag{6.4}$$

as $t \searrow 0$. By the definition of first and second order shape derivatives, it must hold that

$$\delta_1(\mathscr{J}_i, t) = \mathcal{O}(t^2) \quad \text{and} \quad \delta_2(\mathscr{J}_i, t) = \mathcal{O}(t^3) \quad \text{as } t \searrow 0. \tag{6.5}$$

This behavior can be observed in Figure 3(a) for $\mathscr{J}_1$ and in Figure 3(b) for $\mathscr{J}_2$, where we used $V(x_1,x_2) = (x_1^2 x_2 e^{x_2}, x_2^2 x_1 e^{x_1})$ in both cases.
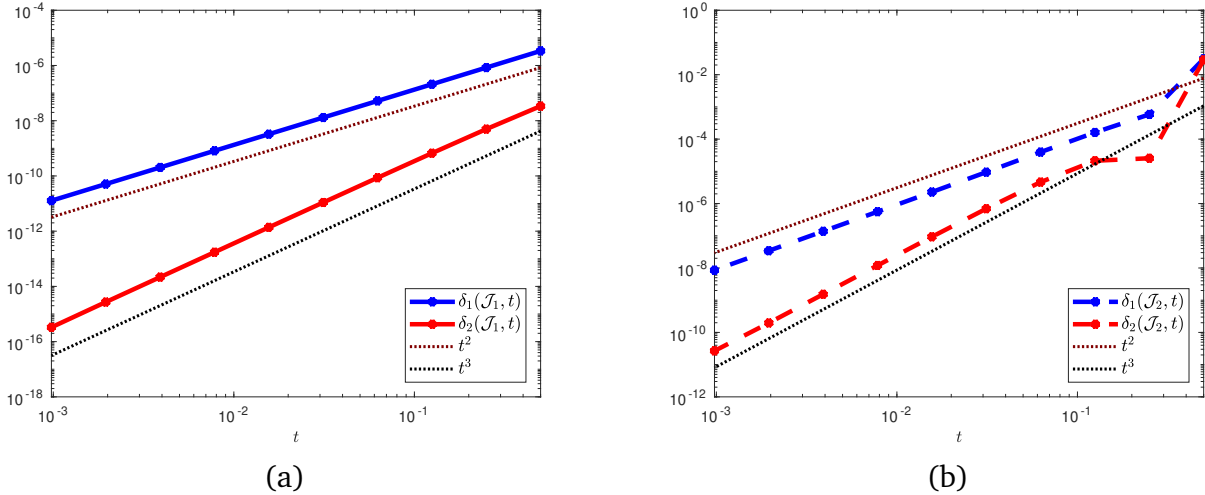

(a)   (b)

Figure 3: Taylor test for functions $\mathscr{J}_1$ and $\mathscr{J}_2$.

The experiments for shape function $\mathscr{J}_1$ was conducted on a mesh consisting of 13662 vertices, 26946 elements and with polynomial order 2 (resulting in 54269 degrees of freedom), and the experiment for $\mathscr{J}_2$ with 95556 vertices and 190062 elements and polynomial degree 1 (95556 degrees of freedom). We conducted these experiments for a number of different problems with different vector fields V, in particular with different PDE constraints and boundary conditions, and obtained similar results in all instances provided a sufficiently fine mesh was used.

6.2 Optimisation algorithms

In this section we discuss how to use optimisation algorithms in conjunction with the automated shape differentiation explained in the previous sections. The starting point of our discussion is a fixed initial shape Ω. Then we consider the mapping

$$V \mapsto g(V) := \mathscr{J}((\mathsf{Id} + V)(\Omega)) \tag{6.6}$$

defined on a suitable space of vector fields $\Theta \subset C^{0,1}(\mathsf{D})^d$. Since the mapping g is defined on an open subset Θ of the Banach space $C^{0,1}(\mathsf{D})^d$ we can employ standard algorithms to minimise g over Θ. The only constraint we must impose is that $\mathsf{Id} + V$ remains invertible, which can be difficult in practice. We observe that for $V, W \in \Theta$ we have

$$\partial g(V)(W) = D\mathscr{J}((\mathsf{Id} + V)(\Omega))(W \circ (\mathsf{Id} + V)^{-1}). \tag{6.7}$$

6.2.1 Gradient computation

The gradient of $\partial g(V)$ in a Hilbert space $H \subset C^{0,1}(\mathsf{D})^d$ is defined by

$$\partial g(V)(W) = (\nabla^H g(V), W)_H \quad \text{for all } W \in H. \tag{6.8}$$

Typical choices for H are

$$H = H_0^1(\mathsf{D})^d, \ (W, V)_H := \int_{\mathsf{D}} \partial W : \partial V + V \cdot W \, \mathrm{d}x, \tag{6.9}$$

$$H = H_0^1(\mathsf{D})^d, \ (W, V)_H := \int_{\mathsf{D}} \varepsilon(W) : \varepsilon(V) + V \cdot W \, \mathrm{d}x, \ \varepsilon(V) := \frac{1}{2}(\partial V + \partial V^\top), \tag{6.10}$$

$$H = H_0^1(\mathsf{D})^d, \ (W, V)_H := \int_{\mathsf{D}} \varepsilon(W) : \varepsilon(V) + V \cdot W + \gamma_{CR} \mathscr{B} V \cdot \mathscr{B} W \, \mathrm{d}x, \text{ where } \gamma_{CR} > 0 \text{ and} \tag{6.11}$$

$$\mathscr{B} := \begin{pmatrix} -\partial_x & \partial_y \\ \partial_y & \partial_x \end{pmatrix}. \tag{6.12}$$

The last choice corresponds to a penalised Cauchy-Riemann gradient and results in a gradient which is approximately conformal and hence preserves good mesh quality. We refer to [17] for a detailed description.

6.2.2 Basic algorithm

Let Ω be an initial shape and let $H \subset C^{0,1}(\mathsf{D})^d$ be a Hilbert space. Then a basic shape optimisation algorithm reads as follows.

Algorithm 1 gradient algorithm

1: **Input:** domain Ω_0, $n = 0$, $N_{max} > 0$, $\epsilon > 0$, $\gamma \geq 0$
2: **Output:** optimal shape Ω^*
3: **while** $n \leq N_{max}$ and $|\nabla \mathcal{J}(\Omega_n)| > \epsilon$ **do**
4:     **if** $\mathcal{J}((\mathsf{Id} - \alpha \nabla \mathcal{J}(\Omega_n))(\Omega_n)) < \mathcal{J}(\Omega_n) - \gamma \alpha |\nabla \mathcal{J}(\Omega_n)|^2$ **then**
5:         $\Omega_{n+1} \leftarrow (\mathsf{Id} - \alpha \nabla \mathcal{J}(\Omega_n))(\Omega_n)$
6:         $n \leftarrow n + 1$
7:         increase α
8:     **else**
9:         reduce α
10:    **end if**
11: **end while**

We present and explain the numerical realisation of Algorithm 1 in `NGSolve` for the case of a PDE-constrained shape optimisation problem in two space dimensions. The simpler case of an unconstrained shape optimisation problem or the case of three space dimensions can be realised by small modifications of the presented code.

First of all, we mention that we realise shape modifications in `NGSolve` by means of deformation vector fields without actually modifying the coordinates of the underlying finite element grid. Recall the vector-valued finite element space VEC over a given mesh as introduced in code line 44. We define a vector-valued `GridFunction` with the name `gfset` which will represent the current shape. We initialise it with some vector-valued coefficient function $V(x_1, x_2) = (x_1^2 x_2, x_2^2 x_1)^\top$ and obtain the deformed shape $(\mathsf{Id} + V)(\Omega)$ by the command `mesh.SetDeformation(gfset)`:

```
218  gfset = GridFunction(VEC)
219  Draw(gfset, mesh, "gfset")
220  SetVisualization (deformation=True)
221  gfset.Set((X[0]*X[0]*X[1],X[1]*X[1]*X[0]))
222  mesh.SetDeformation(gfset)
223  Redraw()
```

Any operation involving the mesh such as integration or assembling of matrices is now carried out for the deformed configuration. The deformation can be unset by the command `mesh.UnsetDeformation()`. Integrating the constant function over the mesh in the perturbed and unperturbed setting,

```
224  print(Integrate(1, mesh))
225  mesh.UnsetDeformation()
226  print(Integrate(1, mesh))
```

gives the output

```
1.7924529046862627
0.7854072970684544
```

respectively.

In the course of the optimisation algorithm the state equation as well as the adjoint equation have to be solved for every new shape. We define the following function, which computes the state and adjoint state for a linear PDE constraint:

```
227  def solvePDE ( ) :
228      a . Assemble ( )
229      L . Assemble ( )
230      dCostdu . Assemble ( )
231
232      inva = a . mat . Inverse ( fes . FreeDofs ( ) ,  inverse=" sparsecholesky " )
233      gfu . vec . data = inva ∗ L . vec
234      gfp . vec . data = −inva . T ∗ dCostdu . vec
```

The shape derivative $\mathtt{dJOmega}$ for some problem at hand can be defined as illustrated in Sections 4.1 and Section 5. Finally, we need to define the shape gradient, which is the solution to a boundary value problem of the form (6.8). We choose the bilinear form defined in (6.11) with $\gamma_{CR} = 10$:

```
235  def eps ( u ) :
236      return 1/2 ∗ ( grad ( u )+grad ( u ) . trans )
237
238  aX = BilinearForm (VEC)
239  W, V = VEC . TnT ( )      # define trial function W and test function V
240
241  aX += InnerProduct ( eps (W) , eps (V) ) ∗dx + InnerProduct (W, V) ∗dx
242  aX += 10 ∗ ( grad (W) [ 1 , 1 ] − grad (W) [ 0 , 0 ] ) ∗ ( grad (V) [ 1 , 1 ] − grad (V) [ 0 , 0 ] ) ∗dx
243  aX += 10 ∗ ( grad (W) [ 1 , 0 ] + grad (W) [ 0 , 1 ] ) ∗ ( grad (V) [ 1 , 0 ] + grad (V) [ 0 , 1 ] ) ∗dx
```

Now we can run Algorithm 1 for problem (4.6):

```
244  alpha = 1
245  alpha_incr_factor = 1.2
246  gamma = 1e−4
247  Nmax = 100
248  epsilon = 1e−7
249
250  isConverged = False
251  gfset . Set ( ( 0 , 0 ) )
252  gfX = GridFunction (VEC)
253  gfsetTemp = GridFunction (VEC)
254
255  solvePDE ( )
256  Jnew = Integrate ( Cost ( gfu ) , mesh )
257  Jold = Jnew
258
259  for k in range (Nmax) :
260      mesh . SetDeformation ( gfset )
261      aX . Assemble ( )
262      dJOmega_pde . Assemble ( )
263      invaX = aX . mat . Inverse (VEC . FreeDofs ( ) , inverse=" sparsecholesky " )
264      gfX . vec . data = invaX ∗ dJOmega_pde . vec
265      currentNormGFX = Norm ( gfX . vec )
266
267      while True :
268          if currentNormGFX < epsilon :
269              isConverged = True
270              break
```

```
271
272          gfsetTemp.vec.data = gfset.vec − alpha ∗ gfX.vec
273          mesh.SetDeformation(gfsetTemp)
274          solvePDE()
275          Jnew = Integrate (Cost(gfu), mesh)
276          mesh.UnsetDeformation()
277          if Jnew < Jold − gamma ∗ alpha ∗ currentNormGFX∗∗2 :
278               Jold = Jnew
279               gfset.vec.data = gfsetTemp.vec
280               alpha ∗= alpha_incr_factor
281               break
282          else:
283               alpha = alpha / 2
284      Redraw(blocking=True)
```

Mesh movement and mesh optimisation   As an alternative to realizing the deformations via `mesh.SetDeformation(...)`, where the underlying mesh is not modified, one could also just move every mesh node in the direction of the given descent vector field by changing its coordinates. This can be realised by invoking the following method:

```
285  def moveNGmesh2D(displ, mesh):
286      for p in mesh.ngmesh.Points():
287          mip = mesh(p[0],p[1])
288          v = displ(mip)
289          p[0] += v[0]
290          p[1] += v[1]
291      mesh.ngmesh.Update()
```

Here, the displacement vector field `displ`, which is of type `GridFunction`, is evaluated for each mesh node and, subsequently, the mesh nodes are updated. At the end of the procedure, the mesh structure needs to be updated, see line 291. Note that the evaluation of `GridFunctions` requires a mapped integration point `mip` of the mesh which is created in line 287.

One advantage of this strategy is that a distorted mesh can easily be repaired by a call of the method `mesh.ngmesh.OptimizeMesh2d()` followed by `mesh.ngmesh.Update()`. Figure 4 shows a distorted mesh and the result of a call of `mesh.ngmesh.OptimizeMesh2d()`.

6.2.3   Newton's method for unconstrained problems

The particular choice $H = H_0^1(D)^d$ and

$$(V, W)_H := D^2 \mathcal{J}(\Omega)(V)(W), \tag{6.13}$$

leads to Newton's method. We refer to [1, 8, 22, 24] where shape Newton methods were used previously and to [14, Chapter 2] and [18, Chapter 5] for Newton's method in an optimal control setting. This bilinear form is only positive semi-definite on $H_0^1(D)^d$ since $D^2 \mathcal{J}(\Omega)(V)(W) = 0$ for V, W with $V = W = 0$ on $\partial\Omega$. Moreover, from the structure theorem for second shape derivatives proved in [21] we know that at a stationary point Ω, that is, $D\mathcal{J}(\Omega)(V) = 0$ for all $V \in C^{0,1}(D)^d$, we have

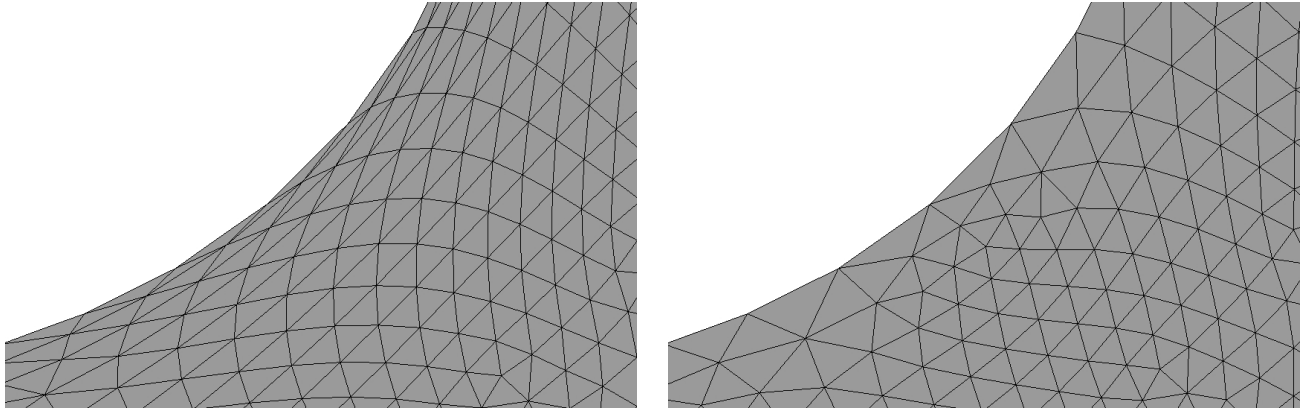$$D^2 \mathcal{J}(\Omega)(V)(W) = \ell_\Omega(V \cdot n, W \cdot n), \tag{6.14}$$

Figure 4: Before and after mesh optimisation by `mesh.ngmesh.OptimizeMesh2d()`.

where $\ell_\Omega : C^0(\partial\Omega) \times C^0(\partial\Omega) \to \mathbf{R}$ is a bilinear function. Hence we also have $D^2 \mathcal{J}(\Omega)(V)(W) = 0$ for all V, W such that $V \cdot n = W \cdot n = 0$. As a result the gradient

$$(\nabla \mathcal{J}(\Omega), V)_H = D\mathcal{J}(\Omega)(V) \quad \text{for all } V \in H_0^0(D)^d \tag{6.15}$$

according to (6.13) is not uniquely determined. To get around this difficulty, the shape Hessian is often regularised by an H^1 term, i.e. (6.13) is replaced by

$$D^2 \mathcal{J}(\Omega)(V)(W) + \delta \int_\Omega \partial V : \partial W + V \cdot W \, \mathrm{d}x, \tag{6.16}$$

see, e.g. [27], which, however, impairs the convergence speed of Newton's method.

Alternative regularisation strategy. Here, we propose the following strategy: We regularise the shape Hessian only on the boundary $\partial\Omega$ and only in tangential direction, i.e., we choose

$$(V, W)_H := D^2 \mathcal{J}(\Omega)(V)(W) + \delta \int_{\partial\Omega} (V \cdot \tau)(W \cdot \tau) \tag{6.17}$$

with a regularisation parameter δ. To exclude the part of the kernel corresponding to interior deformations, we solve the (regularised) Newton equation (6.15) only on the boundary $\partial\Omega$. This is realised by setting Dirichlet boundary conditions for all degrees of freedom except those on the boundary.

```
292  VEC2 = VectorH1(mesh, order=1, dirichlet = "circle") #auxiliary space for boundary
         conditions
293  aX = BilinearForm(VEC)
294  aX += G_f_0.DiffShape(W).DiffShape(V)
295  aX += 100 * InnerProduct(W, specialcf.tangential(2)) * InnerProduct(V, specialcf.
         tangential(2))*ds
296  aX.Assemble()
297  invAX = aX.mat.Inverse(~VEC2.FreeDofs(), inverse="umfpack")
298
299  gfX_bnd = GridFunction(VEC)
300  gfX_bnd.vec.data = invAX * dJOmega_f_0.vec
```

As a result, we get a shape gradient $\nabla \tilde{\mathcal{J}}(\Omega)$ which is nonzero only on the boundary. We extend this vector field to the interior by solving an additional boundary value problem (of linearised elasticity type), where we use the deformation given by $\nabla \tilde{\mathcal{J}}(\Omega)$ as Dirichlet boundary conditions.

```
301  def getExtension(gfX_bnd, freedofs, gfX_ext):
302      u,v = VEC.TnT()
303      aX_ext = BilinearForm(VEC)
304      aX_ext += InnerProduct(grad(u)+grad(u).trans,grad(v))*dx+InnerProduct(u,v)*dx
305
306      gfX_ext.Set(gfX_bnd)
307      aX_ext.Assemble()
308
309      r = gfX_bnd.vec.CreateVector()
310      r.data = (−1) * aX_ext.mat * gfX_ext.vec
311
312      gfX_ext.vec.data += aX_ext.mat.Inverse(freedofs=freedofs) * r
313
314  getExtension(gfX_bnd, VEC2.FreeDofs(), gfX)
315  gfset.Set((0,0))
316  gfset.vec.data = gfset.vec − 1 * gfX.vec
```

The Newton algorithm reads as follows.

Algorithm 2 Newton algorithm

1: **Input:** domain Ω_0, $n = 0$, $N_{max} > 0$, $\epsilon > 0$
2: **Output:** optimal shape Ω^*
3: **while** $n \leq N_{max}$ and $|\nabla \mathcal{J}(\Omega_n)| > \epsilon$ **do**
4:     solve (6.15) to get $\nabla \mathcal{J}(\Omega_n)$
5:     $\Omega_{n+1} \leftarrow (\mathsf{Id} - \nabla \mathcal{J}(\Omega_n))(\Omega_n)$
6:     $n \leftarrow n + 1$
7: **end while**

6.2.4 Newton's method for PDE-constrained problems

We consider the PDE-constrained model problem of Section 4.1 which is subject to the Poisson equation. The unregularised Newton system reads

$$D^2 \mathcal{J}(\Omega)(V)(W) = -D\mathcal{J}(\Omega)(V) \quad \text{for all } V \in H_0^1(\Omega). \tag{6.18}$$

In Subsection 4.3 we discussed how the second order derivative can be evaluated. Recalling that $d\mathcal{J}(\Omega)(V) = \partial_s G_{V,0}(0,0,u,p)$ we see that (4.23) and (4.19) lead to

$$\begin{pmatrix} \partial_s \partial_t G_{V,W}(0,0,u,p) & \partial_u \partial_t G_{V,W}(0,0,u,p) & \partial_p \partial_t G_{V,W}(0,0,u,p) \\ \partial_s \partial_u G_{V,W}(0,0,u,p) & \partial_u^2 G_{V,W}(0,0,u,p) & \partial_p \partial_u G_{V,W}(0,0,u,p) \\ \partial_s \partial_p G_{V,W}(0,0,u,p) & \partial_u \partial_p G_{V,W}(0,0,u,p) & 0 \end{pmatrix} \begin{pmatrix} \tilde{V} \\ \partial_s u^0 \\ \partial_s p^0 \end{pmatrix} = - \begin{pmatrix} \partial_t G_{V,W}(0,0,u,p) \\ 0 \\ 0 \end{pmatrix}. \tag{6.19}$$

The component $\tilde{V}$ then represents the direction which we use for the shape Newton optimisation step. The matrix in (6.19) can be realised in `NGSolve` by using a combined finite element space X3 consisting of three components as follows:

```
317  X3 = FESpace ([VEC, fes, fes ])
318  PHI, u1, p1= X3. TrialFunction ()
319  PSI, uTest1, pTest1 = X3. TestFunction ()
320
321  shapeHessLag3 = BilinearForm (X3)
322  shapeHessLag3 += G_pde_0 . DiffShape (PHI) . DiffShape (PSI)        #block (1,1)
323  shapeHessLag3 += G_pde_0 . DiffShape (PSI) . Diff (gfu , u1)        #block (1,2)
324  shapeHessLag3 += G_pde_0 . DiffShape (PSI) . Diff (gfp , p1)        #block (1,3)
325  shapeHessLag3 += G_pde_0 . Diff (gfu , uTest1) . DiffShape (PHI)    #block (2,1)
326  shapeHessLag3 += (G_pde_0 . Diff (gfu , uTest1)) . Diff (gfu , u1)  #block (2,2)
327  shapeHessLag3 += (G_pde_0 . Diff (gfu , uTest1)) . Diff (gfp , p1)  #block (2,3)
328  shapeHessLag3 += G_pde_0 . Diff (gfp , pTest1) . DiffShape (PHI)    #block (3,1)
329  shapeHessLag3 += (G_pde_0 . Diff (gfp , pTest1)) . Diff (gfu , u1)  #block (3,2)
```

The right hand side of (6.19) can be defined as follows:

```
330  shapeGradLag3 = LinearForm (X3)
331  shapeGradLag3 += (−1) * G_pde_0 . DiffShape (PSI)
```

Recall that the system (6.15) has a nontrivial kernel as discussed in Section 6.2.3. This problem can be circumvented by proceeding like in the unconstrained case. We add a regularisation only on the boundary,

```
332  delta = 1
333  shapeHessLag3 += delta * InnerProduct (PHI, specialcf . tangential (2)) * InnerProduct
         (PSI, specialcf . tangential (2))*ds
```

and exclude the interior degrees of freedom in the first row and column of the 3×3 block system. This can be realised by setting Dirichlet boundary conditions for the interior degrees of freedom, i.e. by dealing with the free degrees of freedom,

```
334  # copy of VEC with Dirichlet boundary conditions on whole boundary:
335  VEC2 = VectorH1 (mesh, order = 1, dirichlet = ".*")
336  freeDofsCombined = BitArray (VEC2. ndof + 2*fes . ndof)
337  for i in range (VEC2. ndof):
338      freeDofsCombined [i] = not VEC2. FreeDofs ()[i]
339  for i in range (fes . ndof):
340      freeDofsCombined [VEC2. ndof+i] = fes . FreeDofs ()[i]
341      freeDofsCombined [VEC2. ndof+fes . ndof+i] = fes . FreeDofs ()[i]
```

and solving the regularised system using these free dofs:

```
342  gfCombined3 = GridFunction (X3)
343  shapeHessLag3 . Assemble ()
344  shapeGradLag3 . Assemble ()
345  gfCombined3 . vec . data = shapeHessLag3 . mat . Inverse (freedofs=freeDofsCombined ,
         inverse="umfpack") * shapeGradLag3 . vec
```

The newton direction is then given as the first of the three components of the obtained solution.

```
346  Vtilde_bnd = GridFunction(VEC)
347  Vtilde = GridFunction(VEC)
348  Vtilde_bnd.vec.data = gfCombined3.components[0].vec
349  getExtension(Vtilde_bnd, VEC2.FreeDofs(), Vtilde)
350
351  gfset.vec.data = gfset.vec + 1 * Vtilde.vec
```

6.3 Numerical shape optimisation of model problems

In this section we will apply the automated shape differentiation and all numerical algorithms introduced in the preceding sections in numerical examples.

6.3.1 A first shape optimisation problem

In this section, we revisit problem (3.1) introduced in Section 3, i.e. the problem of finding a shape Ω such that the cost function $\mathscr{J}(\Omega) = \int_{\Omega} f(x)\,\mathrm{d}x$ is minimised.

First order methods We illustrate our first order methods in a problem which was also considered in [17] and reproduce the results obtained there. We choose the function

$$f(x_1, x_2) = \left(\sqrt{(x_1 - a)^2 + bx_2^2} - 1\right)\left(\sqrt{(x_1 + a)^2 + bx_2^2} - 1\right)$$
$$\cdot \left(\sqrt{bx_1^2 + (x_2 - a)^2} - 1\right)\left(\sqrt{bx_1^2 + (x_2 + a)^2} - 1\right) - \epsilon \qquad (6.20)$$

with $a = \frac{4}{5}$, $b = 2$ and $\epsilon = 0.001$. Recall that the optimal shape is given by $\{(x_1, x_2) \in \mathbf{R}^2 : f(x_1, x_2) < 0\}$ which is depicted in Figure 5 (right). We start our optimisation algorithm with the unit disk, $\Omega^0 = B_1(0)$ as an initial design. Note that the optimal design cannot be reached by means of shape optimisation using boundary perturbations. However, we expect the outer curve of the optimal shape to be reached very closely.

We apply Algorithm 1 with the shape gradient $\nabla \mathscr{J}$ associated to the H^1 inner product (6.9), to the bilinear form of linearised elasticity (6.10) and including the additional Cauchy-Riemann term (6.11). We chose the algorithmic parameters $\gamma = 1e-4$, $\epsilon = 1e-7$, a mesh consisting of 2522 vertices and 4886 elements and a globally continuous vector-valued finite element space VEC of order 3. The results can be seen in Figures 6, 7 and 8, respectively.

Second order method Since Newton's method converges quadratically only in a neighborhood of the optimal solution, we choose a simpler optimal design here. We choose

$$f(x_1, x_2) = \frac{x^2}{a^2} + \frac{y^2}{b^2} - 1 \qquad (6.21)$$

which yields an ellipse with the lengths of the two semi-axes a and b. We choose $a = 1.3$ and $b = 1/a$ and again start the optimisation with the unit disk as initial shape. Figure 9 shows the initial and optimised design after only six iterations of Algorithm 2 with $(\cdot, \cdot)_H$ chosen as
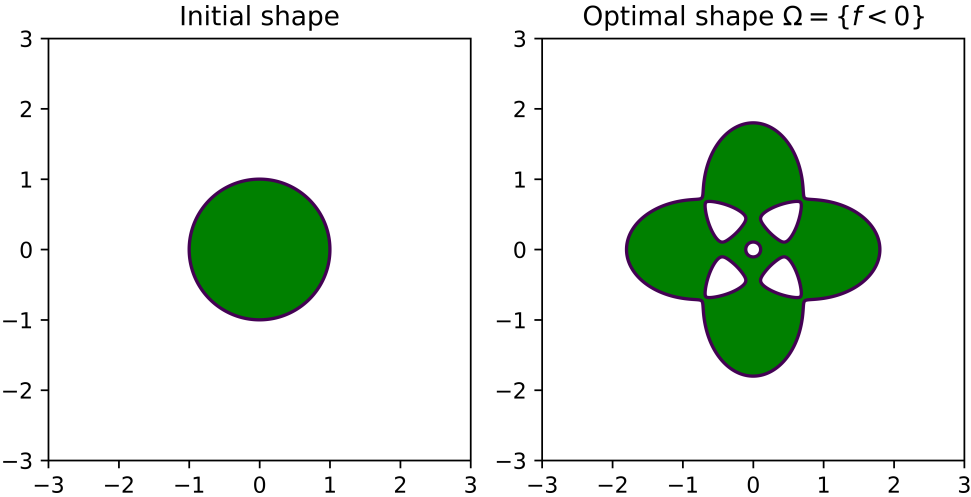
Figure 5: Initial domain Ω_0 and optimal domain Ω^* for problem (3.1) with f chosen according to (6.20).
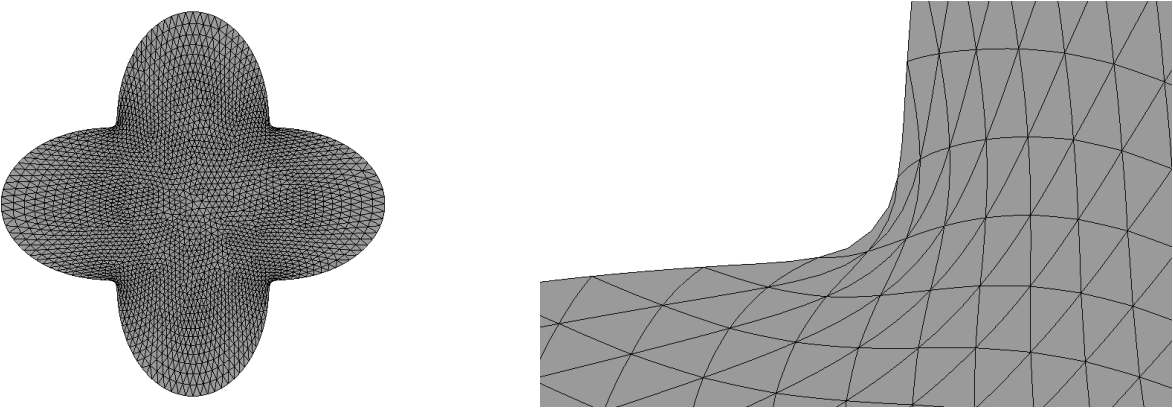

Figure 6: Results of problem (3.1) with f as in (6.20) and the shape gradient associated to the H^1 inner product (6.9).
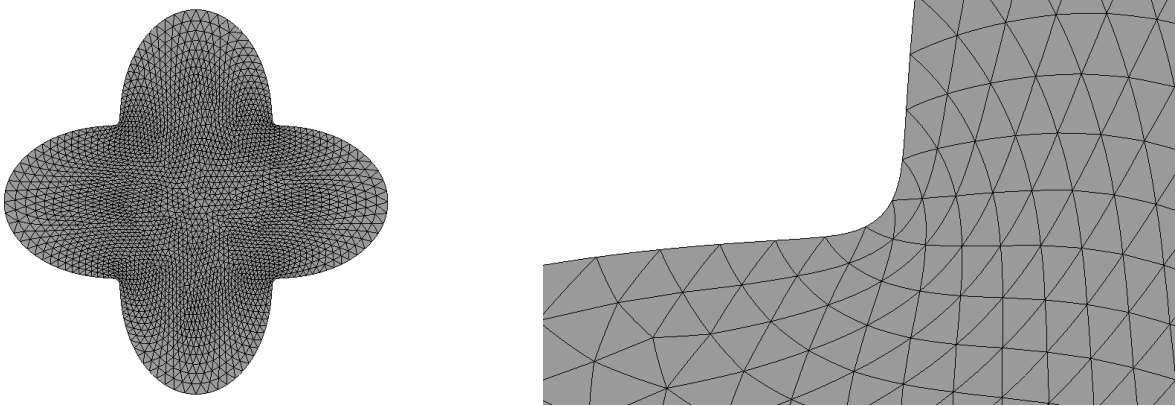
Figure 7: Results of problem (3.1) with f as in (6.20) and the shape gradient associated to the elasticity bilinear form (6.10).
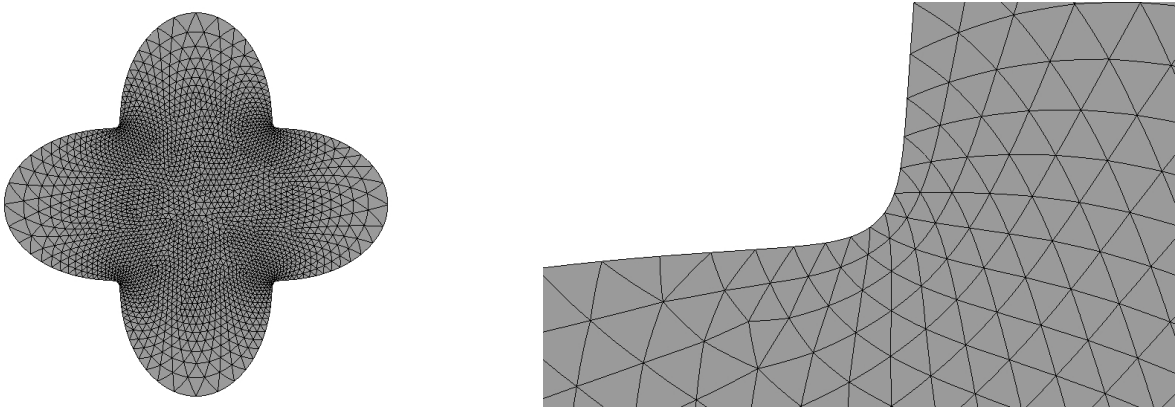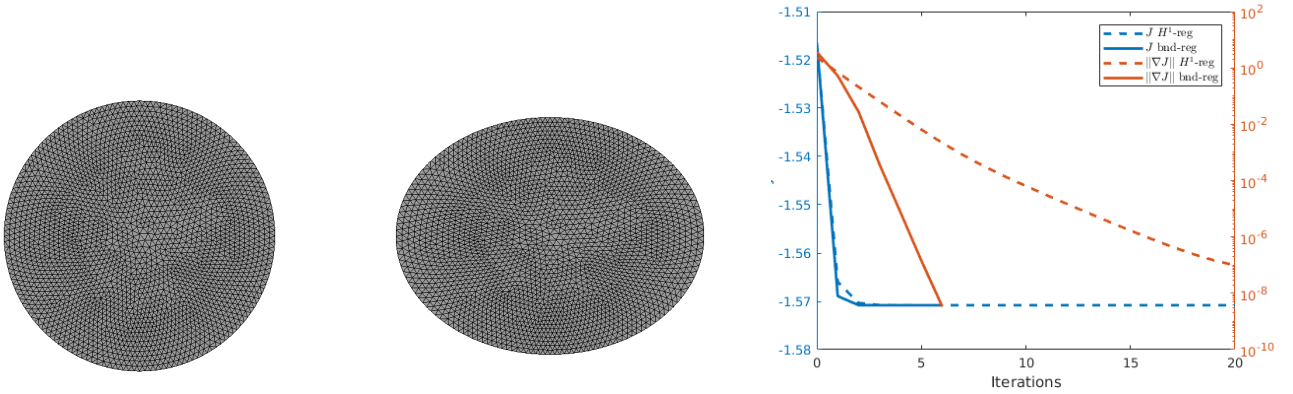

Figure 8: Results of problem (3.1) with f as in (6.20) and the shape gradient associated to the elasticity bilinear form with Cauchy-Riemann term (6.11).

Figure 9: Numerical results for problem (6.17) with f as in (6.21) using second order method. Left: Initial design. Center: Optimised design after six iterations using (6.15)/(6.17). Right: Objective value $\mathscr{J}$ and norm of shape gradient $\|\nabla\mathscr{J}(\Omega)\|$ in the course of second order optimisation using (6.16) with $\delta = 0.5$ and (6.17) with $\delta = 100$.

in (6.17) with $\delta = 100$. A comparison of the convergence histories between the choice (6.17) with $\delta = 100$ and (6.16) with $\delta = 0.5$ is shown in the right picture of Figure 9. In both cases, the parameter δ was chosen empirically to get convergence as fast as possible. The experiments were conducted on a finite element mesh consisting of 2522 nodes and 4886 triangular elements with a finite element space VEC of order 3, with the algorithmic parameter $\epsilon = 10^{-7}$.

6.3.2   Shape optimisation subject to the Poisson equation

In this section, we revisit the model problem introduced in Section 4.1 with $f(x_1, x_2) = 2x_2(1-x_2) + 2x_1(1-x_1)$ and $u_d(x_1, x_2) = x_1(1-x_1)x_2(1-x_2)$. Note that the data is chosen in such a way that, for $\Omega^* = (0, 1)^2$ it holds $\mathscr{J}(\Omega^*) = 0$ and thus Ω^* is a global minimiser of $\mathscr{J}$. We show results obtained by first and second order shape optimisation methods exploiting automated differentiation.

We ran the optimisation algorithm in three versions. On the one hand, we applied a first order method with constant step size $\alpha = 1$. On the other hand, we applied two second order methods with the two different regularisation strategies for the shape Hessian in (6.15) introduced in (6.16) and (6.17). We chose the regularisation parameters δ empirically such that the method performs as well as possible. In the case of (6.16) we chose $\delta = 0.001$ and in the case of (6.17) $\delta = 1$. The experiments were conducted on a finite element mesh consisting of 4886 elements with 2522 vertices and polynomial degree 1. In Figure 10, we can observe the decrease of the objective function as well as of the norm of the shape gradient over 200 iterations for these three algorithmic settings.

Figure 10 shows the initial design as well as the design after 200 iterations of the second order method with regularisation strategy (6.17). Note that the improved design is very close to the global solution $\Omega^* = (0, 1)^2$. The initial design was chosen as the disk of radius $\frac{1}{2}$ centered at the point $\left(\frac{1}{2}, \frac{1}{2}\right)^\top$. The objective value was reduced from $5.297 \cdot 10^{-5}$ to $1.0317 \cdot 10^{-9}$.
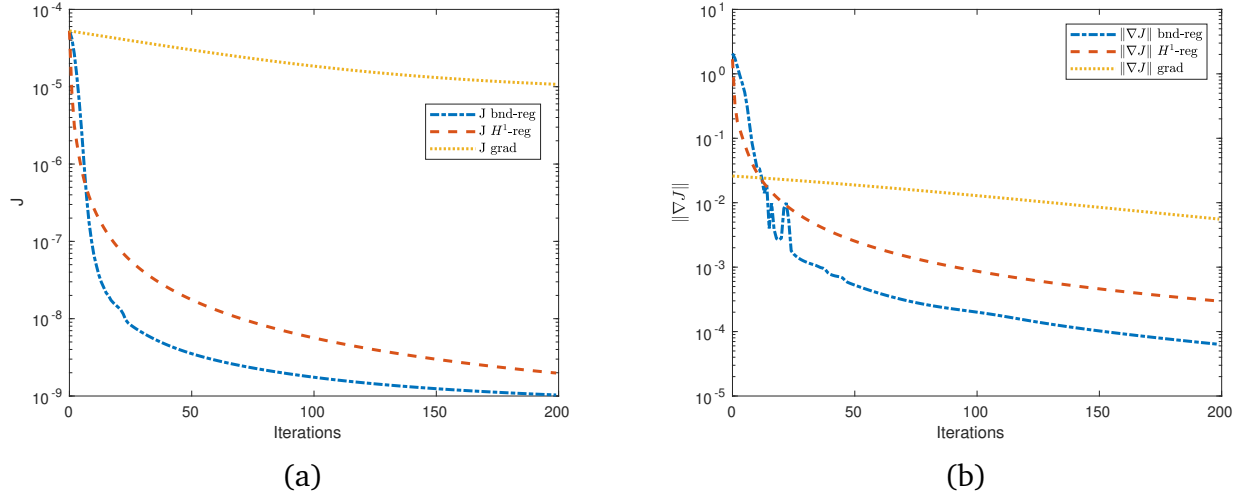
(a) (b)

Figure 10: Convergence behaviour for shape optimisation problem (4.6) with proposed regularisation strategies (6.17) and (6.16) as well as first order method with constant step size $\alpha = 1$. (a) Behaviour of objective function $\mathscr{J}$. (b) Behaviour of norm of shape gradient $\|\nabla \mathscr{J}(\Omega)\|$.
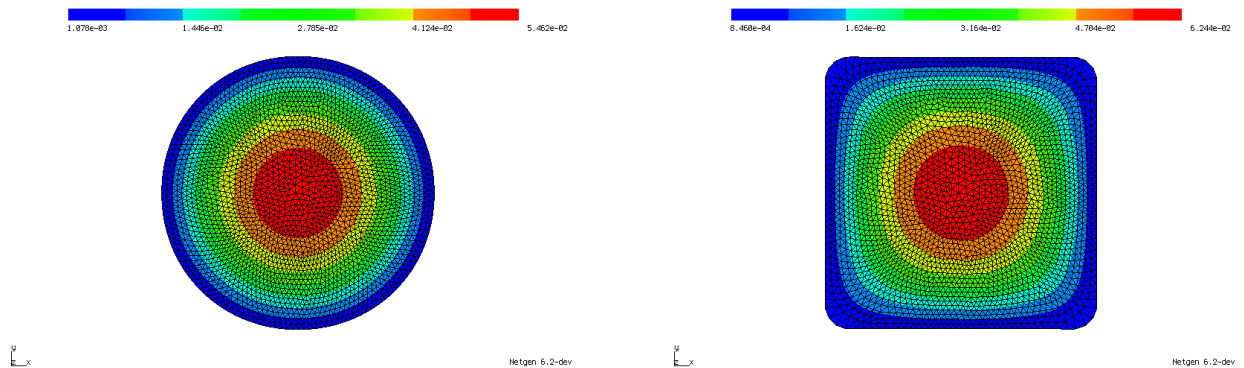

Figure 11: Shape optimisation for problem (4.6). Left: Initial design. Right: Improved design after 200 iterations of second order algorithm with regularisation as proposed in (6.17). Objective value was reduced from $5.297 \cdot 10^{-5}$ to $1.0317 \cdot 10^{-9}$. Color shows solution of constraining PDE (4.6b).

6.3.3   Nonlinear elasticity

Here, we illustrate the applicability of the automated shape differentiation and optimisation in the more realistic and more complicated setting of nonlinear elasticity in two space dimensions using a Saint Venant–Kirchhoff material with Young's modulus $E = 1000$ and Poisson ratio $v = 0.3$. We consider a two-dimensional cantilever which is clamped on the upper and lower left parts of the boundary, $\Gamma_l^1 = \{0\} \times (0.88, 1)$ and $\Gamma_l^2 = \{0\} \times (0, 0.12)$, respectively, and is subject to a surface force $g_N = (0, -100)^\top$ on $\Gamma_r = \{1\} \times (0.45, 0.55)$. The initial geometry with 3 holes is depicted in Figure 12 (a). Let $\Gamma_l := \Gamma_l^1 \cup \Gamma_l^2$ and $H_{\Gamma_l}^1(\Omega)^2$ the subspace of $H^1(\Omega)^2$ with vanishing trace on Γ_l. The displacement $u \in H_{\Gamma_l}^1(\Omega)^2$ under the surface force g_N is given as the solution to the boundary value problem

$$\int_\Omega S(u) : \nabla v \, dx = \int_{\Gamma_r} g_N \cdot v \, ds \tag{6.22}$$

for all $v \in H_{\Gamma_l}^1(\Omega)^2$. Here, $S(u)$ denotes the Saint Venant–Kirchhoff stress tensor

$$S(u) = (I_2 + \nabla u)\left[\lambda \operatorname{Tr}\left(\frac{1}{2}(C(u) - I_2)\right)I_2 + \mu(C(u) - I_2)\right], \tag{6.23}$$

where $C(u) = (I_2 + \nabla u)^\top (I_2 + \nabla u)$ and I_2 is the identity matrix, see also [2, Sec. 8], and λ and μ denote the Lamé constants,

$$\lambda = \frac{E v}{(1 + v)(1 - 2v)}, \qquad \mu = \frac{E}{2(1 + v)}. \tag{6.24}$$

We minimise the functional

$$J(\Omega, u) = \int_\Omega S(u) : \nabla u \, dx + \alpha \int_\Omega 1 \, dx \tag{6.25}$$

with $\alpha = 2.5$ subject to (6.22) which amounts to maximising the structure's stiffness while bounding the allowed amount of material used.

We remark that the well-posedness of (6.22) is not clear, see also the discussion in [2, Sec. 8]. Nevertheless, application of the automated shape differentiation and optimisation yields a significant improvement of the initial design. The highly nonlinear PDE constraint (6.22) is solved by Newton's method. In order to have good starting values, a load stepping strategy is employed, i.e., the load on the right hand side is gradually increased, the PDE is solved and the solution is used as an initial guess for the next load step. This is repeated until the full load is applied. With these ingredients at hand, Algorithm 1 (i.e. code lines 244–284) can be run. We chose the algorithmic parameters `alpha = 0.1` (as an initial value), `alpha_incr_factor = 1` (i.e. no increase), `gamma = 1e-4` and `epsilon = 1e-7`. Moreover, we used (6.9) with an additional Cauchy-Riemann term as in (6.11) with weight $\gamma_{CR} = 10$. The objective value was reduced from 3.125 to 2.635 (volume term from 1.290 to 1.096) in 15 iterations of Algorithm 1. The results were obtained on a mesh consisting of 10614 elements and 5540 vertices using piecewise linear, globally continuous finite elements.

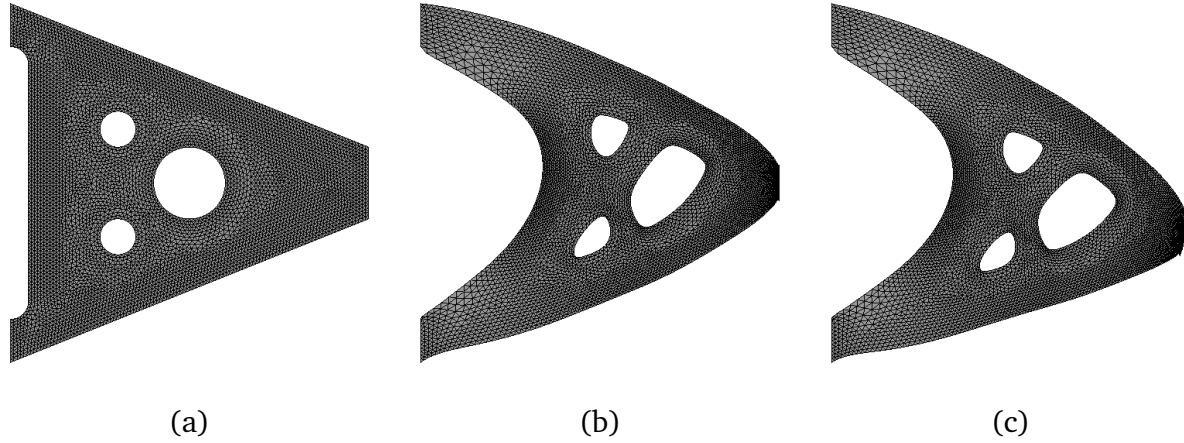|		
(a)	(b)	(c)

Figure 12: Initial and optimised geometry of cantilever under vertical force on right hand side using St. Venant–Kirchhoff model in nonlinear elasticity. (a) Initial geometry. (b) Optimised geometry (reference configuration). (c) Optimised geometry (deformed configuration).

6.3.4 Helmholtz equation

In this section, we consider the problem of finding the optimal shape of a scattering object. More precisely, we consider the minimisation of the functional

$$\int_{\Gamma_r} u\bar{u}\,\mathrm{d}s \tag{6.26}$$

subject to the Helmholtz equation with impedance boundary conditions on the outer boundary,

$$\text{Find } u \in H^1(\Omega, \mathbf{C}): \int_\Omega \left[\nabla u \cdot \nabla \bar{w} - \omega^2 u\bar{w}\right]\mathrm{d}x - i\,\omega \int_\Gamma u\bar{w}\,\mathrm{d}s = \int_\Omega f\,\bar{w} \tag{6.27}$$

for all $w \in H^1(\Omega, \mathbf{C})$. Here, $\bar{w}$ denotes the complex conjugate of a complex-valued function w, ω denotes the wave number, i denotes the complex unit and the function f on the right hand side is chosen as

$$f(x_1, x_2) = 10^3 \cdot e^{-9((x_1-0.2)^2 + (x_2-0.5)^2)}, \tag{6.28}$$

see Figure 13(a). Furthermore $\Omega = B((0.5, 0.5)^\top, 1) \backslash B((0.75, 0.5)^\top, 0.15)$ denotes the domain of interest, $\Gamma = \{(x_1, x_2): x_1^2 + x_2^2 = 1\}$ the outer boundary and $\Gamma_r = \{(x_1, x_2): x_1^2 + x_2^2 = 1, x_1 \geq 0\}$ the right half of the outer boundary. Here, only the inner boundary $\partial\Omega \backslash \Gamma$ is subject to the shape optimisation. Thus, the aim of this model problem is to find a shape of the scattering object such that the waves are reflected away from Γ_r.

Figure 13 (b) and (c) show the initial and final shape of the scattering object, respectively. Figure 14 shows the norm of the state for the initial configuration (circular shape of scattering object) and for the optimised configuration. The objective value was reduced from $3.44 \cdot 10^{-3}$ to $3.31 \cdot 10^{-3}$. The forward simulations were performed using piecewise linear finite elements on a triangular grid consisting of with 34803 degrees of freedom. The optimisation stopped after 12 iterations.
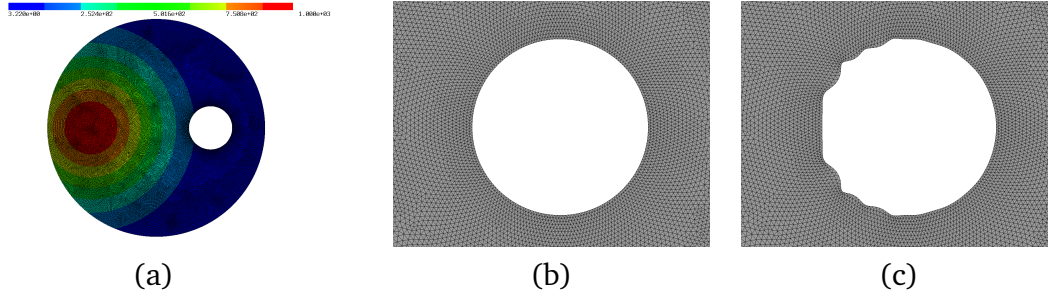
Figure 13: (a) Geometry with right hand side. (b) Initial shape of scatter. (c) Optimised shape of scatter.

Figure 14: (a) Absolute value of state u for initial configuration. (b) Absolute value of state u for optimised configuration.

6.3.5 Application to Electrical Machine

In this section, we consider the setting of three-dimensional nonlinear magnetostatics in $H(\mathbf{curl}, \mathrm{D})$ as it appears in the simulation of electrical machines. Let $\mathrm{D} \subset \mathbf{R}^3$ denote the computational domain, which consists of ferromagnetic material, air regions and permanent magnets. Our aim is to minimise the functional

$$\int_{\Omega_g} |\mathbf{curl}\, u \cdot n - B_d^n|^2 \, \mathrm{d}x, \tag{6.29}$$

where Ω_g denotes the air gap region of the machine, n denotes an extension of the normal vector to the interior of Ω_g, $B_d^n : \Omega_g \to \mathbf{R}^3$ is a given smooth function and $u \in H_0(\mathbf{curl}, \mathrm{D})$ is the solution to the boundary value problem

$$\int_{\mathrm{D}} \nu_\Omega(|\mathbf{curl}\, u|)\, \mathbf{curl}\, u \cdot \mathbf{curl}\, w + \delta u \cdot w \, \mathrm{d}x = \int_{\Omega_m} M \cdot \mathbf{curl}\, w \, \mathrm{d}x \tag{6.30}$$

for all $w \in H_0(\mathbf{curl}, \mathrm{D})$. Here, $\Omega \subset \mathrm{D}$ denotes the union of the ferromagnetic parts of the electrical machine, Ω_m denotes the permanent magnets subdomain and

$$\nu_\Omega = \chi_\Omega(x)\hat{\nu}(|\mathbf{curl}\, u|) + \chi_{\mathrm{D}\backslash\Omega}(x)\nu_0 \tag{6.31}$$

denotes the magnetic reluctivity, which is a nonlinear function $\hat{\nu}$ inside the ferromagnetic regions and equal to a constant ν_0 elsewhere. Further, $\delta > 0$ is a small regularisation parameter and $M : \mathrm{D} \to \mathbf{R}^3$ denotes the magnetisation in the permanent magnets. The nonlinear function $\hat{\nu}$

satisfies a Lipschitz condition and a strong monotonicity condition such that problem (6.30) is well-posed. We refer the reader to [11, Sec. 6] for a more detailed description of the problem and [10] for a 2D version of the same problem.

As mentioned in Section 4, the transformation Φ_t used in (4.4) depends on the differential operator. For the **curl**-operator, we have

$$\Phi_t(u) = \partial T_t^{-\top}(u \circ T_t^{-1}) \quad \text{and} \quad (\mathbf{curl}(\Phi_t(u))) \circ T_t = \frac{1}{\det(\partial T_t)} \partial T_t \, \mathbf{curl}(u), \qquad (6.32)$$

see e.g. [20, Section 3.9]. Thus, the variational equation (6.30) can be defined as follows.

```
352  from math import pi
353  nu0 = 1e7 / (4*pi)
354  delta = 0.1
355
356  F = Id(3)
357  c1 = 1/Det(F) * F
358  c2 = Inv(F).trans
359
360  def EquationIron(u,w):
361      return ( nuIron(Norm(c1*curl(u))) * (c1*curl(u))*(c1*curl(w)) + delta*(c2*u)*(
         c2*w) )*Det(F)*dx("iron")
362
363  def EquationAir(u,w):
364      return (nu0*(c1*curl(u))*(c1*curl(w)) + delta*(c2*u)*(c2*w))*Det(F)*dx("air|
         airgap")
365
366  def EquationMagnets(u,w):
367      return (nu0*(c1*curl(u))*(c1*curl(w)) + delta*(c2*u)*(c2*w)-magn*(c1*curl(w)))
         *Det(F)*dx("magnets")
368
369  def Equation(u,w):
370      return EquationIron(u,w) + EquationAir(u,w) + EquationMagnets(u,w)
```

Here, the computational domain consists of a subdomain representing the ferromagnetic part of the machine (''iron'') and a subdomain comprising the permanent magnets (''magnets''); the union of all air subdomains, including the air gap between rotating and fixed part of the machine, is given by ''air|airgap''. Moreover, nuIron denotes the nonlinear reluctivity function $\hat{v}$ and magn contains the magnetization direction of the permanent magnets. Likewise, the objective function can be defined as follows,

```
371  def Cost(u):
372      return (InnerProduct(c1*curl(u),n2D) - Bd)*Det(F)*dx("airgap")
```

where n2D and Bd represent the extension of the normal vector to the interior of the air gap and the desired curve, respectively. For the definition of all quantities, we refer the reader to the code which was submitted together with this manuscript. The shape differentiation as well as the optimisation loop now works in the same way as in the previous examples. Figure 15 shows the initial design of the motor as well as the optimised design obtained after 11 iterations of Algorithm 1 with $\gamma = 0$. The experiment was conducted using a tetrahedral finite element mesh consisting of 13440 vertices, 57806 elements and Nédélec elements of order 2 (resulting in a
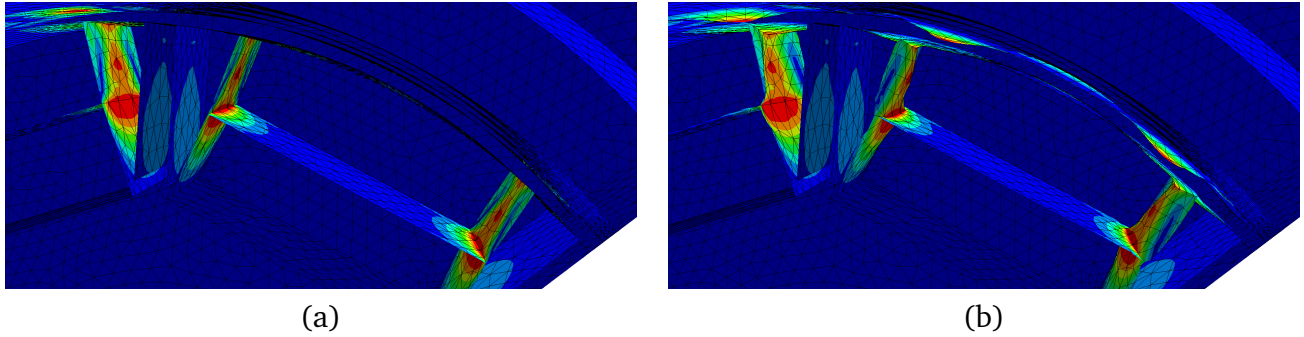
(a)   (b)

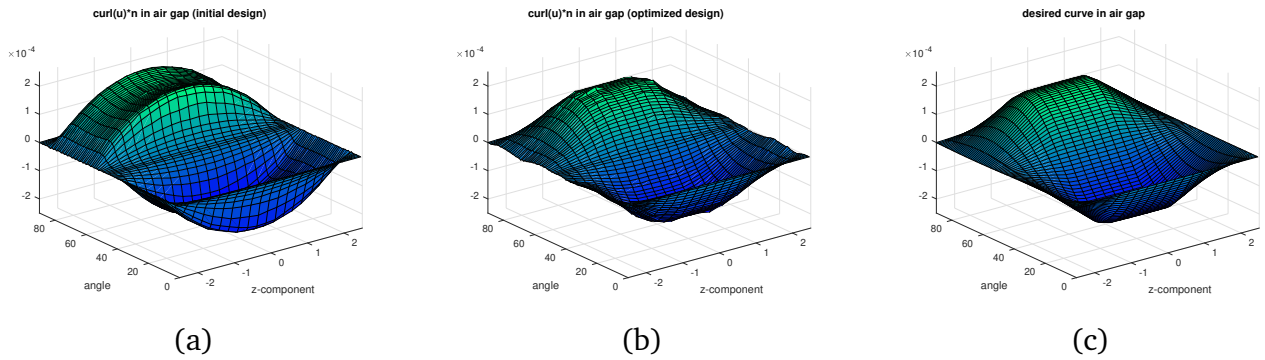Figure 15: (a) Initial design of electrical machine. (b) Optimised design.

(a)   (b)   (c)

Figure 16: Improvement of $\mathbf{curl}\,u \cdot n$ as a function of z and the angle φ for a fixed radius r compared to desired curve B_d^n. (a) $\mathbf{curl}\,u \cdot n$ for initial configuration. (b) $\mathbf{curl}\,u \cdot n$ for optimised configuration after 10 iterations. (c) Desired curve B_d^n in polar coordinates as function of z and the angle φ for a fixed radius r.

total of 323808 degrees of freedom). The objective value was reduced from $2.5944 \cdot 10^{-8}$ to $4.565 \cdot 10^{-10}$ in the course of the first order optimisation algorithm after 11 iterations. It can be seen from Figure 16 that the difference between the quantity $\mathbf{curl}(u) \cdot n$ and the desired curve B_d^n inside the air gap decreases significantly.

6.3.6   Surface PDEs

Finally, we also show the application of a shape optimisation algorithm to a problem constrained by a surface PDE. We solve problem (4.24)–(4.25) with $u_d = 0$, $f(x_1, x_2, x_3) = x_1 x_2 x_3$ and initial shape $M = S^2$ the unit sphere in $\mathbf{R}^3$. We applied a first order algorithm with a line search. Figure 17 shows the initial geometry as well as the decrease of the objective function and of the norm of the shape gradient. The objective value was reduced from $7.08 \cdot 10^{-4}$ to $9.88 \cdot 10^{-9}$. Figure 18 shows the final design which was obtained after 575 iterations from two different perspectives. The experiment was conducted using a surface mesh with 332 vertices and 660 faces and polynomial degree 3 (resulting in 2972 degrees of freedom).

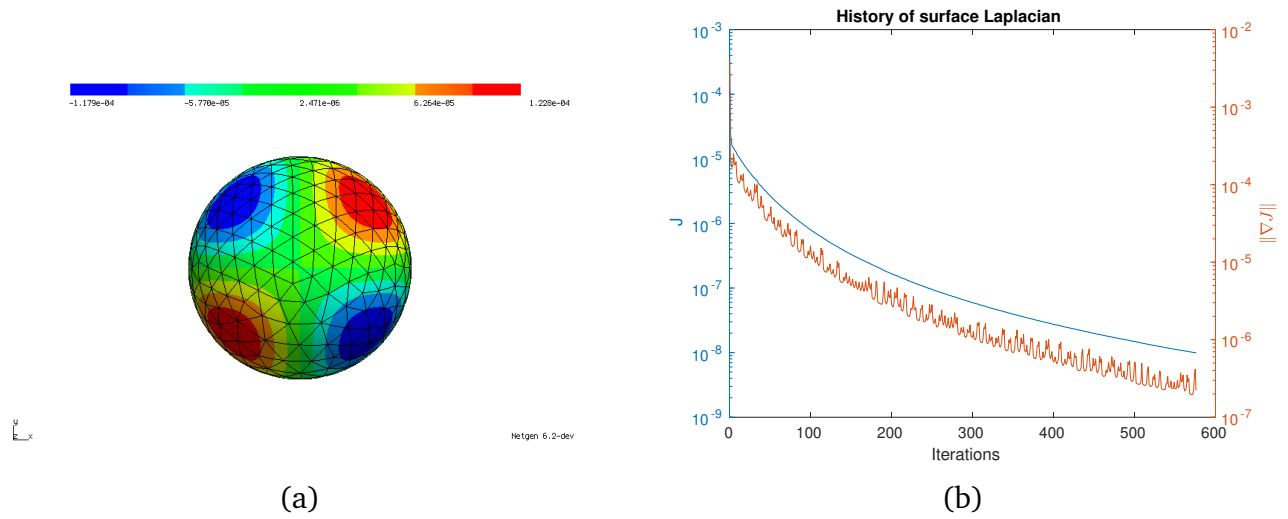(a)   (b)

Figure 17: (a) Initial geometry for shape optimisation with respect to surface PDE (4.24)–(4.25).
(b) History of objective value and norm of shape gradient using a first order algorithm with line
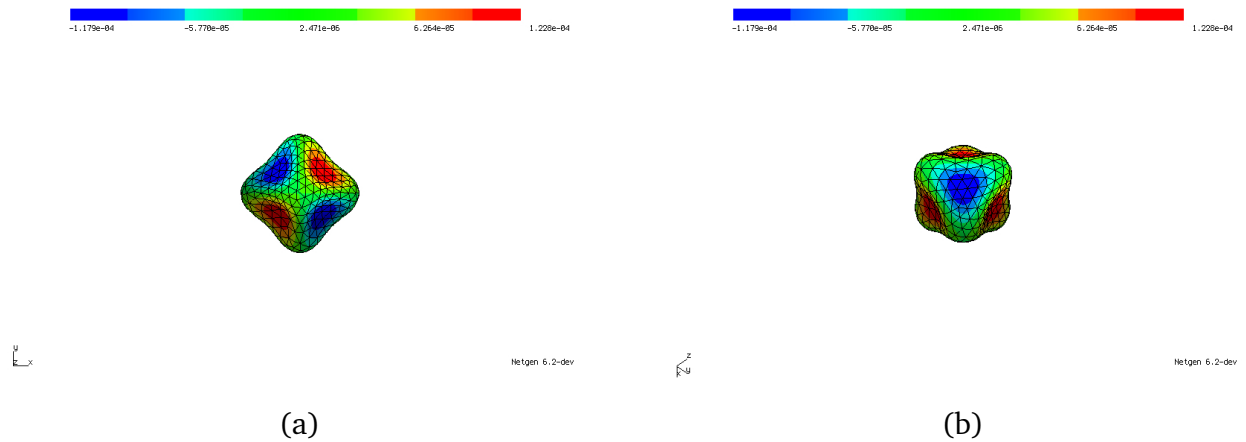search.

(a)   (b)

Figure 18: (a) Final design after 575 iterations. (b) Different view of (a).

Conclusion

We showed how to obtain first and second order shape derivatives for unconstrained and PDE-constrained shape optimization problems in a semi-automatic and fully automatic way in the finite element software package `NGSolve`. We verified the proposed method numerically by Taylor tests and by showing its successful application to several shape optimisation problems. We believe that this intuitive approach can help research scientists working in the field of shape optimisation to further improve numerical methods on the one hand, and product engineers working with `NGSolve` to design devices in an optimal fashion on the other hand.

Acknowledgements

Replication of results

The python scripts which were used for the results presented in this paper are available with the arxiv submission. All computations were performed using NGSolve version V6.2.2004.

References

[1] G. Allaire, E. Cancès, and J. L. Vié. Second-order shape derivatives along normal trajectories, governed by Hamilton-Jacobi equations. *Structural and Multidisciplinary Optimization*, 54(5):1245–1266, 2016.

[2] G. Allaire, F. Jouve, J., and A.-M. Toader. Structural optimization using sensitivity analysis and a level-set method. *Journal of Computational Physics*, 194(1):363 – 393, 2004.

[3] M. S. Alnæs, A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells. Unified form language. *ACM Transactions on Mathematical Software*, 40(2):1–37, 2014.

[4] Martin Berggren. A unified discrete-continuous sensitivity analysis method for shape optimization. In *Applied and numerical partial differential equations*, volume 15 of *Comput. Methods Appl. Sci.*, pages 25–39. Springer, New York, 2010.

[5] M. C. Delfour and J.-P. Zolésio. *Shapes and geometries*, volume 22 of *Advances in Design and Control*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2011. Metrics, analysis, differential calculus, and optimization.

[6] M. C. Delfour and J. P. Zolésio. *Shapes and geometries*. Society for Industrial and Applied Mathematics, 2011.

[7] J. S. Dokken, S. K. Mitusch, and S. W. Funke. Automatic shape derivatives for transient PDEs in FEniCS and Firedrake. *arXiv e-prints*, page arXiv:2001.10058, 2020.

[8] K. Eppler, H. Harbrecht, and R. Schneider. On convergence in elliptic shape optimization. *SIAM Journal on Control and Optimization*, 46(1):61–83, 2007.

[9] F. Feppon, G. Allaire, F. Bordeu, J. Cortial, and C. Dapogny. Shape optimization of a coupled thermal fluid-structure problem in a level set mesh evolution framework. *SeMA*, 76(3):413–458, 2019.

[10] P. Gangl, U. Langer, A. Laurain, H. Meftahi, and K. Sturm. Shape optimization of an electric motor subject to nonlinear magnetostatics. *SIAM Journal on Scientific Computing*, 37(6):B1002–B1025, 2015.

[11] P. Gangl and K. Sturm. Asymptotic analysis and topological derivative for 3D quasi-linear magnetostatics, 2019. arXiv:1908.10775.

[12] D. A. Ham, L. Mitchell, A. Paganini, and . Wechsung, F. Automated shape differentiation in the unified form language. *Structural and Multidisciplinary Optimization*, 60(5):1813–1820, 2019.

[13] M. Hintermüller and A. Laurain. Electrical impedance tomography: from topology to shape. *Control and Cybernetics*, 37(4):913–933, 2008.

[14] M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE constraints*. Springer, New York, 2009.

[15] R. Hiptmair, A. Paganini, and S. Sargheini. Comparison of approximate shape gradients. *BIT*, 55(2):459–485, 2015.

[16] D. Hömberg and J. Sokolowski. Optimal shape design of inductor coils for surface hardening. *SIAM Journal on Control and Optimization*, 42(3):1087–1117, 2003.

[17] J. A. Iglesias, K. Sturm, and F. Wechsung. Two-dimensional shape optimization with nearly conformal transformations. *SIAM Journal on Scientific Computing*, 40(6):A3807–A3830, 2018.

[18] K. Ito and K. Kunisch. *Lagrange multiplier approach to variational problems and applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2008.

[19] A. Laurain. A level set-based structural optimization code using FEniCS. *Structural and Multidisciplinary Optimization*, 58(3):1311–1334, 2018.

[20] P. Monk. *Finite element methods for Maxwell's equations*. Numerical Mathematics and Scientific Computation. Clarendon Press, 2003.

[21] A. Novruzi and M. Pierre. Structure of shape derivatives. *Journal of Evolution Equations*, 2(3):365–382, 2002.

[22] A. Novruzi and J. R. Roche. Newton's method in shape optimisation: A three-dimensional case. *Bit Numerical Mathematics*, 40(1):102–120, 2000.

[23] A. Paganini, S. Sargheini, R. Hiptmair, and Ch. Hafner. Shape optimization of microlenses. *Optics Express*, 23(10):13099, 2015.

[24] A. Paganini and K. Sturm. Weakly normal basis vector fields in RKHS with an application to shape Newton methods. *SIAM Journal on Numerical Analysis*, 57(1):1–26, 2019.

[25] Anton Schiela and Julian Ortiz. Second order directional shape derivatives, March 2017.

[26] S. Schmidt. A two stage CVT / eikonal convection mesh deformation approach for large nodal deformations. *arXiv e-prints*, page arXiv:1411.7663, 2014.

[27] S. Schmidt. Weak and strong form shape Hessians and their automatic generation. *SIAM Journal on Scientific Computing*, 40(2):C210–C233, 2018.

[28] S. Schmidt, C. Ilic, V. Schulz, and N. Gauger. Three-dimensional large-scale aerodynamic shape optimization based on shape calculus. *AIAA Journal*, 51(11):2615–2627, 2013.

[29] S. Schmidt, C. Ilic, V. Schulz, and N. R. Gauger. Airfoil design for compressible inviscid flow based on shape calculus. *Optimization and Engineering*, 12(3):349–369, 2011.

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

[31] V. H. Schulz. A riemannian view on shape optimization. *Foundations of Computational Mathematics*, 14(3):483–501, 2014.

[32] J. Sokołowski and J.-P. Zolésio. *Introduction to shape optimization*, volume 16 of *Springer Series in Computational Mathematics*. Springer-Verlag, Berlin, 1992. Shape sensitivity analysis.

[33] K. Sturm. 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.

[34] K. Sturm. Shape differentiability under non-linear PDE constraints. In *New trends in shape optimization*, volume 166 of *Internat. Ser. Numer. Math.*, pages 271–300. Birkhäuser/Springer, Cham, 2015.

Erschienene Preprints ab Nummer 2018/1

2018/1 U. Langer, M. Schanz, O. Steinbach, W.L. Wendland (eds.): 16th Workshop on Fast Boundary Element Methods in Industrial Applications, Book of Abstracts

2018/2 S. Dohr, J. Zapletal, G. Of, M. Merta, M. Kravcenko: A parallel space-time boundary element method for the heat equation

2018/3 S. Dohr, M. Merta, G. Of, O. Steinbach, J. Zapletal: A parallel solver for a preconditioned space-time boundary element method for the heat equation

2018/4 S. Amstutz, P. Gangl: Toplogical derivative for nonlinear magnetostatic problem

2018/5 O. Steinbach, M. Zank: A Stabilized Space–Time Finite Element Method for the Wave Equation

2018/6 O. Steinbach, H. Yang: A Space-Time Finite Element Method for the Linear Bidomain Equations

2018/7 O. Steinbach, M. Zank: Coercive space-time finite element methods for initial boundary value problems

2018/8 S. Dohr, K. Niino, O. Steinbach: Space-time boundary element methods for the heat equation

2018/8 O. Steinbach, H. Yang: Space-time finite element methods for parabolic evolution equations: Discretization, a posteriori error estimation, adaptivity and solution

2019/1 O. Steinbach (eds.): 15th Austrian Numerical Analysis Day, Book of Abstracts

2019/2 P. Gangl, K. Sturm: A simplified derivation technique of topological derivatives for quasi-linear transmission problems

2019/3 M. Merkel, P. Gangl, S. Schöps: Shape Optimization of Rotating Electric Machines using Isogeometric Analysis and Harmonic Stator-Rotor Coupling

2019/4 P. Gangl and K. Sturm: Asymptotic analysis and topological derivative for 3D quasi-linear magnetostatics

2019/5 M. Holzmann and G. Unger: Boundary integral formulations of eigenvalue problems for elliptic differential operators with singular interactions and their numerical approximation by boundary element methods

2019/6 M. Neumüller, O. Steinbach: Regularization error estimates for distributed control problems in energy spaces

2019/7 U. Langer, M. Schanz, O. Steinbach, W. L. Wendland (eds.): 17th Workshop on Fast Boundary Element Methods in Industrial Applications , Book of Abstracts

2019/8 O. Steinbach, M. Zank: A note on the efficient evaluation of a modified Hilbert transformation

2019/9 P. Gangl: A multi-material topology optimization algorithm based on the topological derivative

2020/1 D. Pacheco, T. Müller, O. Steinbach, G. Brenn: A mixed finite element formulation for generalised Newtonian fluid flows with appropriate natural outflow boundary conditions

2020/2 U. Langer, O. Steinbach, F. Tröltzsch, H. Yang: Unstructured space-time finite element methods for optimal sparse control of parabolic equations

2020/3 D.R.Q. Pacheco, R. Schussnig, O. Steinbach, T.-P. Fries: A fully consistent equal-order finite element method for incompressible flow problems

2020/4 U. Langer, O. Steinbach, F. Tröltzsch, H. Yang: Unstructured space-time finite element methods for optimal control of parabolic equations