

AN OBJECT-ORIENTED LIBRARY FOR SHAPE OPTIMIZATION PROBLEMS GOVERNED BY SYSTEMS OF LINEAR ELLIPTIC PARTIAL DIFFERENTIAL EQUATIONS

Dalibor Lukáš *
dalibor.lukas@vsb.cz

Wolfram Mühlhuber **
wmuehlhu@sfb013.uni-linz.ac.at

Michael Kuhn**
kuhn@sfb013.uni-linz.ac.at

Abstract: An optimization problem arises when looking for optimal parameters of a device under some given requirements on its functionality. There is a common structure in solving shape optimization problems which are governed by linear elliptic partial differential equations (PDE). In the paper, the structure is presented and we discuss the re-use of its components when solving different shape optimization problems. We also present a current implementation of the library which has been built in the *C++* programming language using fast robust solvers for linear elliptic PDE problems discretized by the Finite Element Method. We mainly focus on an incorporation of necessary components, e.g., the mesh generator, the PDE solver and the optimizer. Finally, we give a particular example of shape optimization in magnetostatics solved by using the library.

Keywords: Shape optimization, scientific computing, object-oriented design, magnetostatics

1 Introduction

Optimization problems arise when looking for optimal parameters of devices or systems. The optimality is related to certain requirements on the functionality. Once a mathematical criterion of the optimality is formulated as well as other restrictions (constraints) on the parameters, we can formulate an optimization problem in mathematical terms. One of the constraints usually describes the mathematical model of the device, like Partial Differential Equations (PDE) in mechanics, electromagnetics, fluid dynamics, etc. A special and important class of

* Department of Applied Mathematics, VŠB-TU Ostrava, 17. listopadu, 708 33, Ostrava-Poruba, Czech Rep.

** SFB F013 "Numerical and Symbolic Scientific Computing", University of Linz, Freistädter Straße 313, A-4040 Linz, Austria

optimization problems is called shape optimization. In this case parameters are related to selected interfaces of the device and we are looking for optimal shapes of these interfaces.

Nowadays, the use of scientific computing tools is necessary for the development of industrial products. Thus, optimization plays an important role in the designing process. In this context, the main goal of researchers is to develop a fast and efficient software which is also friendly to those designers not studying mathematics. As there is still a lack of efficient solvers for real-life optimization problems, we find the goal as a challenge for our following research.

The general topics and the mathematical theory on shape optimization are covered in [6] and [16]. In shape optimization, many references can be found in the context of mechanics, e.g., [5], or in fluid dynamics, e.g., [1]. But, there are still only a few papers concerning shape optimization problems governed by the Maxwell equations. Particularly in magnetostatics a numerical solution of a real-life problem, namely the TEAM (Testing Electromagnetic Analysis Methods) problem No. 25, see [17], is given in [2].

1.1 Available Software for Optimization

There are already several commercial software packages for the finite element modelling which also include optimization tools. They provide a friendly user-interface based on the Computer Aided Design technologies. However, the solvers are not fast enough and solving advanced real-life problems takes much too long, even on super-computers.

Some scientific tools for optimization have already been developed. They are usually freely available and they apply the latest research results. The solvers are faster and the advanced real-life problems can be solved when using PCs or workstations. On the other hand, a user has to have more knowledge about the mathematical modelling and the optimization. For instance, the MATLAB Optimization Toolbox [18] provides a very comfortable programming interface on a level of matrix operations but the optimization tools are general and do not supply any special problem class like shape optimization. Let us also mention the ODESSY [7] software. It stands among the freely available academical software for the Computer Aided Design. It provides both an analysis, i.e., a modelling, and a synthesis, e.g., an optimization, based on rigorous scientific methods in mechanical engineering.

1.2 Goal of the Shape Optimization Library

We are still missing efficient tools which are able to re-use components common for the shape optimization problems with different kinds of governing PDE systems covering, e.g., solid mechanics, linear magnetostatics, etc. In our case we base the library on the top of an existing object-oriented code FEPP [9] for solving linear elliptic systems of PDEs. We exploit a high modularity, a robustness, and experience during developing and using the package FEPP. The modularity is provided by several libraries, e.g., for linear algebra, assembling finite element matrices, iterative solvers, multigrid preconditioner, etc. The robustness follows from an abstract setting of the linear elliptic PDE boundary value problem. The abstract setting enables the user to solve a wide class of physical problems while the mathematical structure of the problem is described rather than the physics behind. Finally, the software has been developed for 3 years and it has been used by different research workers, mainly by electrical and mechanical engineers. They applied the software to a variety of real-life problems from mechanics, magnetics and magneto-mechanics, see e.g. [8, 14]. Note that within the same research project

a suitable mesh generator NETGEN [15] has been developed and nowadays there are more than 200 signed licenses all over the world.

The paper is organized as follows. In Sect. 2 we introduce a setting of the shape optimization problem. In Sect. 3 we give a structure, very similar to the structure in [6], for solving optimization problems governed by a linear elliptic PDE problem. Moreover, we distinguish parts specific for shape optimization, e.g., a design-to-mesh mapping, a part calculating sensitivities of the stiffness matrix and of the load vector, a part dealing with the adjoint variable method, etc. We also discuss the re-use of the components and replacing them by alternative components. In Sect. 4 we present a current implementation of the library based on advanced PDE tools. In Sect. 5 we use the library for shape optimization of an electromagnet. The aim of the paper is to present a scientific software tool for shape optimization where the optimization problems are specified with a minimal programming effort.

The aim of our research work is to develop a software tool for shape optimization which will be well-designed, will cover all the fields where linear elliptic systems of PDEs appear, e.g., mechanics, electrostatics, magnetostatics, etc. , will be easily extendable, and mainly, which will solve optimization problems sufficiently fast with only a minimal programming effort necessary for describing the problem itself. Our hope that we will manage the goal is supported by experience with the FEPP and by our promising results [11] when solving shape optimization problems in magnetostatics. Those results are presented briefly in Sect. 5.3.

2 Setting of the Shape Optimization Problem

First we introduce a general optimization problem. Let $\mathbf{p} \in P$ be the parameters to be optimized where P is a feasible set of parameters. Let $\varphi : P \mapsto \mathbb{R}$ be an objective functional which measures the quality of designs and which is to be either minimized or maximized. Without loosing a generality, we will merely consider minimization problems in the sequel. A general optimization problem reads

$$\min_{\mathbf{p} \in P} \varphi(\mathbf{p}) . \quad (1)$$

We will consider shape optimization problems. Here, the design parameters correspond to coordinates of the design boundary nodes and solving a linear elliptic PDE state problem is involved. For a given shape design $\boldsymbol{\alpha}$, an abstract weak formulation of the state problem reads

$$\begin{cases} \text{Find } u \in V : \\ a(u, v) = l(v) \quad \forall v \in V \end{cases} \quad (2)$$

where V is a Hilbert space of functions defined over a domain Ω , $a : V \times V \mapsto \mathbb{R}$ is an elliptic continuous bilinear form and $l : V \mapsto \mathbb{R}$ is a linear continuous functional. We denote the weak formulation (2) by (V, a, l) . In general, all the symbols in (2) depend on $\boldsymbol{\alpha}$.

Now we briefly introduce the FEM concept. We discretize the geometry Ω into finite elements T_h , e.g., a triangulation in the 2-dimensional case or a decomposition into tetrahedra in the case of 3-dimensional domains. By h we denote the discretization parameter, e.g., the length of the longest element edge. Further, let Ω_h denote the discretized domain Ω . Depending on $a(\cdot, \cdot)$ we introduce a finite dimensional subspace $V_h \subset V$ over Ω_h . Then the FEM approximation of (2) reads

$$\begin{cases} \text{Find } u_h \in V_h : \\ a(u_h, v_h) = l(v_h) \quad \forall v_h \in V_h \end{cases} . \quad (3)$$

We denote (3) by (V_h, a, l) . From now on we will merely consider the discretized problem (3). Finally, from (3) we arrive at the following linear algebraic system

$$\mathbf{K}(\boldsymbol{\alpha}) \cdot \mathbf{u}(\boldsymbol{\alpha}) = \mathbf{f}(\boldsymbol{\alpha}) \quad (4)$$

where the so-called stiffness matrix \mathbf{K} is a symmetric and positive definite sparse matrix, \mathbf{f} is the so-called load vector and \mathbf{u} is the solution of (4). Roughly speaking, the stiffness matrix \mathbf{K} represents the discretized bilinear form $a(\cdot, \cdot)$ and the load vector \mathbf{f} represents the discretized linear functional $l(\cdot)$. Again, all the introduced symbols depend on the given shape $\boldsymbol{\alpha}$.

Now we can establish the discretized shape optimization problem. The shape to be optimized is a surface within the geometry Ω_h . After the FEM discretization only the nodes lying on the design surface are considered as the design variables. Let $\boldsymbol{\alpha} \in \mathbb{R}^d$ denote the discretized shape where d is the number of design variables, e.g., coordinates of the control nodes in the FEM discretization T_h . Further let

$$\tilde{\varphi}(\boldsymbol{\alpha}) := \varphi(\boldsymbol{\alpha}, \mathbf{u}(\boldsymbol{\alpha})) \quad (5)$$

be an objective function and let

$$\tilde{\mathbf{g}}(\boldsymbol{\alpha}) \equiv [\tilde{g}_1(\boldsymbol{\alpha}), \dots, \tilde{g}_n(\boldsymbol{\alpha})]^\top := \mathbf{g}(\boldsymbol{\alpha}, \mathbf{u}(\boldsymbol{\alpha})) \equiv [g_1(\boldsymbol{\alpha}, \mathbf{u}), \dots, g_n(\boldsymbol{\alpha}, \mathbf{u})]^\top \quad (6)$$

be a vector constraint function which determines implicitly the feasible set. Then, the discretized shape optimization problem reads as follows :

$$\left\{ \begin{array}{l} \min_{\boldsymbol{\alpha} \in \mathbb{R}^d} \tilde{\varphi}(\boldsymbol{\alpha}) \\ \text{under} \\ \mathbf{K}(\boldsymbol{\alpha}) \cdot \mathbf{u}(\boldsymbol{\alpha}) = \mathbf{f}(\boldsymbol{\alpha}) \\ \mathbf{g}(\boldsymbol{\alpha}, \mathbf{u}(\boldsymbol{\alpha})) \leq \mathbf{0} \end{array} \right. \quad (7)$$

In the sequel we will merely deal with computational aspects rather than with the mathematical background.

3 Structure of the Shape Optimization Library

First we introduce the notation for partial derivatives and gradients of scalar and vector functions. Let $\varphi(\boldsymbol{\alpha}, \mathbf{u}) : \mathbb{R}^d \times \mathbb{R}^m \mapsto \mathbb{R}$ be a scalar function and $\mathbf{g}(\boldsymbol{\alpha}, \mathbf{u}) := [g_1(\boldsymbol{\alpha}, \mathbf{u}), \dots, g_n(\boldsymbol{\alpha}, \mathbf{u})]^\top : \mathbb{R}^d \times \mathbb{R}^m \mapsto \mathbb{R}^n$ be a vector function of the vector arguments $\boldsymbol{\alpha} := (\alpha_1, \dots, \alpha_d)$, $\mathbf{u} := (u_1, \dots, u_m)$. We denote by

$$\partial_{\alpha_i} \varphi(\boldsymbol{\alpha}, \mathbf{u}) := \frac{\partial \varphi}{\partial \alpha_i}(\boldsymbol{\alpha}, \mathbf{u}) \quad , \quad \partial_{\alpha_i} \mathbf{g}(\boldsymbol{\alpha}, \mathbf{u}) := \left[\frac{\partial g_1}{\partial \alpha_i}(\boldsymbol{\alpha}, \mathbf{u}), \dots, \frac{\partial g_n}{\partial \alpha_i}(\boldsymbol{\alpha}, \mathbf{u}) \right]^\top$$

partial derivatives of φ and of \mathbf{g} , respectively, by the scalar variable α_i . Further, let

$$\nabla_{\boldsymbol{\alpha}} \varphi(\boldsymbol{\alpha}, \mathbf{u}) := [\partial_{\alpha_1} \varphi(\boldsymbol{\alpha}, \mathbf{u}), \dots, \partial_{\alpha_d} \varphi(\boldsymbol{\alpha}, \mathbf{u})]^\top \quad , \quad \nabla_{\boldsymbol{\alpha}} \mathbf{g}(\boldsymbol{\alpha}, \mathbf{u}) := [\nabla_{\boldsymbol{\alpha}} g_1(\boldsymbol{\alpha}, \mathbf{u}), \dots, \nabla_{\boldsymbol{\alpha}} g_n(\boldsymbol{\alpha}, \mathbf{u})]$$

denote the partial gradients of φ and of \mathbf{g} , respectively, by the vector variable $\boldsymbol{\alpha}$. Let $\mathbf{K}(\boldsymbol{\alpha}) : \mathbb{R}^d \mapsto \mathbb{R}^{m \times m}$ be a matrix function of the vector argument. We introduce the partial derivative $\partial_{\alpha_i} \mathbf{K}(\boldsymbol{\alpha})$ of the matrix function with respect to the scalar variable in the same way, i.e., component-wise, as in the case of the vector function.

In this section we describe the structure of the solver for shape optimization. We refer to Fig. 1 where the structure is drawn. The functionality of the particular components as well as meanings of the symbols in Fig. 1 will be described in the following subsections. Finally, we will point out the main goal of the library that only a minimal programming effort is required when solving a new shape optimization problem.

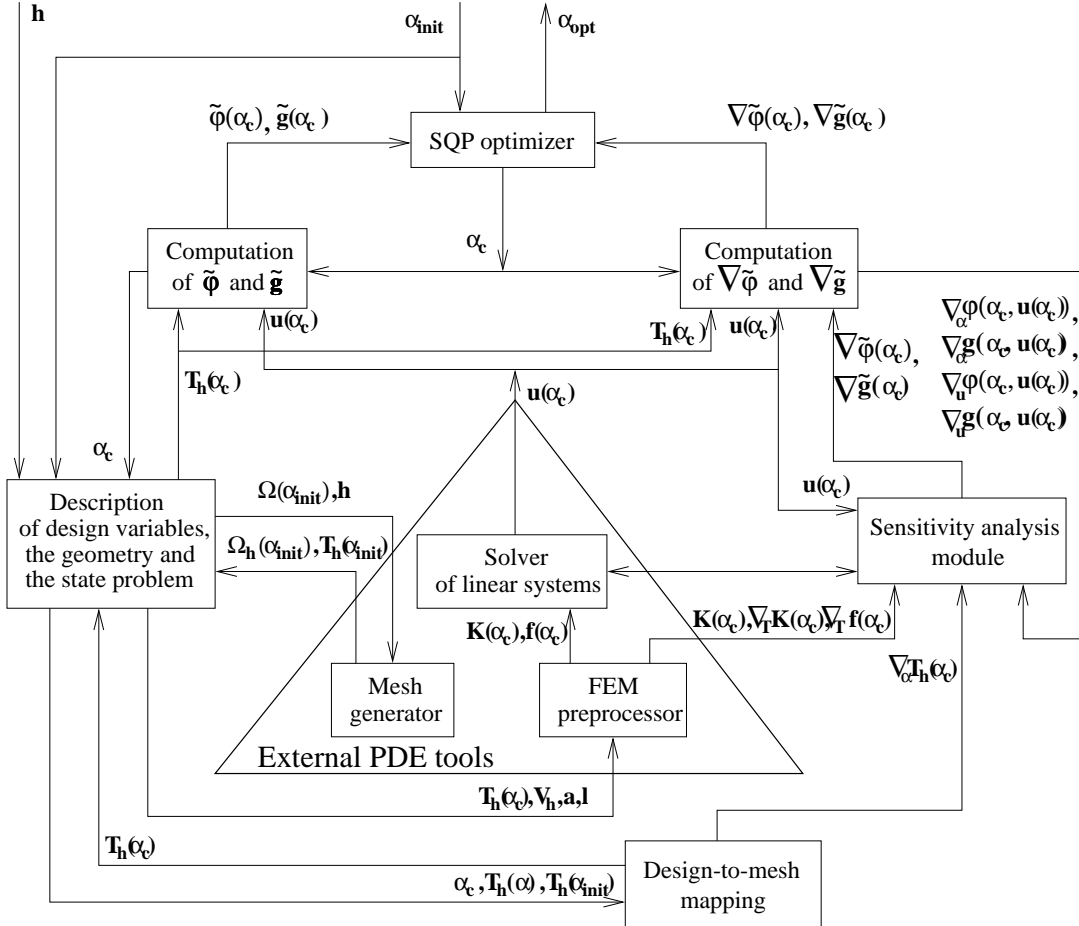


Fig. 1: Structure (data flow diagram) of the library

3.1 The Optimizer

Having introduced the discretized setting (7) of the shape optimization problem, we describe the structure of the solver. The solver is based on the Sequential Quadratic Programming (SQP) with the BFGS update of the Hessian matrix, see e.g. [12] for a detailed description. The SQP method is basically a loop where a Quadratic Programming (QP) subproblem is solved at each iteration. When using the BFGS update, we have to provide only evaluations

of the objective $\tilde{\varphi}(\boldsymbol{\alpha}_c)$ and the constraint $\tilde{\mathbf{g}}(\boldsymbol{\alpha}_c)$ functions at the current design $\boldsymbol{\alpha}_c$ as well as of their gradients $\nabla\tilde{\varphi}(\boldsymbol{\alpha}_c)$ and $\nabla\tilde{\mathbf{g}}(\boldsymbol{\alpha}_c)$. At the very beginning of the algorithm, a mesh generator is called to give a discretization $T_h(\boldsymbol{\alpha}_c)$ for a given mesh parameter h . Then, from the initial design $\boldsymbol{\alpha}_{\text{init}}$ which is feasible, i.e.,

$$\tilde{\mathbf{g}}(\boldsymbol{\alpha}_{\text{init}}) \leq \mathbf{0} \quad , \quad (8)$$

the algorithm proceeds to the optimized design $\boldsymbol{\alpha}_{\text{opt}}$.

3.2 Description of the Problem and Design-to-Mesh Mapping

The module ‘‘Description of the problem’’ accesses the necessary data of the shape optimization problem. On the one hand side, it provides the abstract weak formulation (V_h, a, l) of the state problem. On the other hand, the module provides an evaluation of the design-to-mesh mapping (the mesh deformation) $T_h(\boldsymbol{\alpha}_c)$. The latter needs information about the relation between control nodes and the design variables. In Fig. 1 this information is denoted by $T_h(\boldsymbol{\alpha})$. Moreover, the design-to-mesh mapping has also to provide a differentiation of the mapping with respect to the design variables.

These are the most crucial parts of the library as well as topics of our research. While designing a user-interface describing the geometry and the state problem seems to be rather a technical problem, finding a robust design-to-mesh mapping is quite a difficult task.

3.3 Computation of the Objective and the Constraints

In order to evaluate the objective and the constraint functions at a given design $\boldsymbol{\alpha}_c$, three subproblems have to be solved. First, the functions $\tilde{\varphi}(\boldsymbol{\alpha}_c)$ and $\tilde{\mathbf{g}}(\boldsymbol{\alpha}_c)$ have to be implemented. Moreover, for their evaluation the finite element discretization $T_h(\boldsymbol{\alpha}_c)$ and the solution $\mathbf{u}(\boldsymbol{\alpha}_c)$ of the state problem have to be provided. The procedure is drawn in Alg. 1.

Algorithm 1 Computation of the objective $\tilde{\varphi}$ and the constraints $\tilde{\mathbf{g}}$

Given $\boldsymbol{\alpha}_c$

Call the ‘‘Description module’’: for the design $\boldsymbol{\alpha}_c \rightarrow$ a deformed discretization $T_h(\boldsymbol{\alpha}_c)$

Call the FEM preprocessor: for $T_h(\boldsymbol{\alpha}_c)$ and $(V_h, a, l) \rightarrow$ assembled $\mathbf{K}(\boldsymbol{\alpha}_c)$ and $\mathbf{f}(\boldsymbol{\alpha}_c)$

Call the ‘‘Solver’’: Solve $\mathbf{K}(\boldsymbol{\alpha}_c) \cdot \mathbf{u}(\boldsymbol{\alpha}_c) = \mathbf{f}(\boldsymbol{\alpha}_c) \rightarrow \mathbf{u}(\boldsymbol{\alpha}_c)$

$\tilde{\varphi}(\boldsymbol{\alpha}_c) := \varphi(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$

$\tilde{\mathbf{g}}(\boldsymbol{\alpha}_c) := \mathbf{g}(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$

3.4 Sensitivity Analysis Module

This module is the main part of the library. It provides analytical evaluations of the gradients. First we introduce the used symbols. We formally denote sensitivities of the stiffness matrix by $\nabla_{T_h} \mathbf{K}(\boldsymbol{\alpha}_c) := \bigcup_j \{\partial_{x_j} \mathbf{K}(\boldsymbol{\alpha}_c), \partial_{y_j} \mathbf{K}(\boldsymbol{\alpha}_c), \partial_{z_j} \mathbf{K}(\boldsymbol{\alpha}_c)\}$ and of the load vector by $\nabla_{T_h} \mathbf{f}(\boldsymbol{\alpha}_c) := \bigcup_j \{\partial_{x_j} \mathbf{f}(\boldsymbol{\alpha}_c), \partial_{y_j} \mathbf{f}(\boldsymbol{\alpha}_c), \partial_{z_j} \mathbf{f}(\boldsymbol{\alpha}_c)\}$ where the triple (x_j, y_j, z_j) stands for coordinates of the node j within the discretization $T_h(\boldsymbol{\alpha}_c)$. Further, we formally denote the derivative of the

design-to-mesh mapping by $\nabla_{\alpha} T_h(\alpha_c) := \bigcup_j \{\nabla_{\alpha} x_j(\alpha_c), \nabla_{\alpha} y_j(\alpha_c), \nabla_{\alpha} z_j(\alpha_c)\}$. Finally, we introduce the following vector functions

$$\tilde{\Psi}(\alpha) := [\tilde{\varphi}(\alpha), \tilde{g}_1(\alpha), \dots, \tilde{g}_n(\alpha)]^T, \quad (9)$$

$$\Psi(\alpha, \mathbf{u}(\alpha)) := [\varphi(\alpha, \mathbf{u}(\alpha)), g_1(\alpha, \mathbf{u}(\alpha)), \dots, g_n(\alpha, \mathbf{u}(\alpha))]^T. \quad (10)$$

Recall that n stands for the number of constraints.

Alg. 2 describes the function of the sensitivity analysis module in terms of the introduced notation. Note that in the implementation of Alg. 2 the matrix-vector products, e.g., $\partial_{\alpha_i} \mathbf{K}(\alpha_c) \cdot \mathbf{u}(\alpha_c)$, are assembled rather than the matrices $\partial_{\alpha_i} \mathbf{K}(\alpha_c)$ themselves.

Algorithm 2 Sensitivity analysis module

Given $\alpha_c, \mathbf{u}(\alpha_c), \nabla_{\alpha} \varphi(\alpha_c, \mathbf{u}(\alpha_c)), \nabla_{\mathbf{u}} \varphi(\alpha_c, \mathbf{u}(\alpha_c)), \nabla_{\alpha} \mathbf{g}(\alpha_c, \mathbf{u}(\alpha_c)), \nabla_{\mathbf{u}} \mathbf{g}(\alpha_c, \mathbf{u}(\alpha_c))$
Call the FEM preprocessor: Access the stiffness matrix $\rightarrow \mathbf{K}(\alpha_c)$
Call the FEM preprocessor: Assemble sensitivities $\rightarrow \nabla_{T_h} \mathbf{K}(\alpha_c), \nabla_{T_h} \mathbf{f}(\alpha_c)$
Evaluate derivative of the design-to-mesh mapping $\rightarrow \nabla_{\alpha} T_h(\alpha_c)$
for $i := 1, \dots, d$ **do**
 $\partial_{\alpha_i} \mathbf{K}(\alpha_c) := \sum_j (\partial_{x_j} \mathbf{K}(\alpha_c) \cdot \partial_{\alpha_i} x_j(\alpha_c) + \partial_{y_j} \mathbf{K}(\alpha_c) \cdot \partial_{\alpha_i} y_j(\alpha_c) + \partial_{z_j} \mathbf{K}(\alpha_c) \cdot \partial_{\alpha_i} z_j(\alpha_c))$
 $\partial_{\alpha_i} \mathbf{f}(\alpha_c) := \sum_j (\partial_{x_j} \mathbf{f}(\alpha_c) \cdot \partial_{\alpha_i} x_j(\alpha_c) + \partial_{y_j} \mathbf{f}(\alpha_c) \cdot \partial_{\alpha_i} y_j(\alpha_c) + \partial_{z_j} \mathbf{f}(\alpha_c) \cdot \partial_{\alpha_i} z_j(\alpha_c))$
end for
if $d < n + 1$ **then**
 / The direct method */*
 for $i := 1, \dots, d$ **do**
 Call the ‘‘Solver’’: Solve $\mathbf{K}(\alpha_c) \cdot \partial_{\alpha_i} \mathbf{u}(\alpha_c) = \partial_{\alpha_i} \mathbf{f}(\alpha_c) - \partial_{\alpha_i} \mathbf{K}(\alpha_c) \cdot \mathbf{u}(\alpha_c) \rightarrow \partial_{\alpha_i} \mathbf{u}(\alpha_c)$
 $\partial_{\alpha_i} \tilde{\Psi}(\alpha_c) := (\partial_{\alpha_i} \mathbf{u}(\alpha_c))^T \cdot \nabla_{\mathbf{u}} \Psi(\alpha_c, \mathbf{u}(\alpha_c)) + \partial_{\alpha_i} \Psi(\alpha_c, \mathbf{u}(\alpha_c))$
 end for
else
 / The adjoint variable method */*
 $\mathbf{A}(\alpha_c, \mathbf{u}(\alpha_c)) := [\partial_{\alpha_1} \mathbf{K}(\alpha_c) \cdot \mathbf{u}(\alpha_c), \dots, \partial_{\alpha_d} \mathbf{K}(\alpha_c) \cdot \mathbf{u}(\alpha_c)]$
 for $i := 1, \dots, n + 1$ **do**
 Call the ‘‘Solver’’: Solve $\mathbf{K}(\alpha_c) \cdot \lambda(\alpha_c, \mathbf{u}(\alpha_c)) = \nabla_{\mathbf{u}} \Psi_i(\alpha_c, \mathbf{u}(\alpha_c)) \rightarrow \lambda(\alpha_c, \mathbf{u}(\alpha_c))$
 $\nabla \tilde{\Psi}_i(\alpha_c) := (\nabla \mathbf{f}(\alpha_c) - \mathbf{A}(\alpha_c, \mathbf{u}(\alpha_c)))^T \cdot \lambda(\alpha_c, \mathbf{u}(\alpha_c)) + \nabla_{\alpha} \Psi_i(\alpha_c, \mathbf{u}(\alpha_c))$
 end for
end if
 $\nabla \tilde{\varphi}(\alpha_c) := \nabla \tilde{\Psi}_1(\alpha_c)$
 $\nabla \tilde{\mathbf{g}}(\alpha_c) := [\nabla \tilde{\Psi}_2(\alpha_c), \dots, \nabla \tilde{\Psi}_{n+1}(\alpha_c)]$

Finally, we discuss advantages and drawbacks of the method. A very common ‘‘black box’’ method for calculating gradients is numerical differentiation. But, for more design variables this method takes much computational time. Our results given in Sect. 5.3 show that using a method of the analytical sensitivity analysis is much more efficient. The only drawback of the analytical sensitivity analysis methods is more programming effort concerning sensitivities. However, once we implement sensitivities for supported elliptic operators, we can use them independently from the shape optimization problem. Similarly, when a more general design-to-mesh mapping is available as well as its derivative, we can re-use the mapping for

different problems. The only part which has to be supplied by the user is calculating the partial gradients, i.e., $\nabla_{\alpha}\varphi(\alpha_c, \mathbf{u}(\alpha_c))$, $\nabla_{\mathbf{u}}\varphi(\alpha_c, \mathbf{u}(\alpha_c))$, $\nabla_{\alpha}\mathbf{g}(\alpha_c, \mathbf{u}(\alpha_c))$ and $\nabla_{\mathbf{u}}\mathbf{g}(\alpha_c, \mathbf{u}(\alpha_c))$. Practically this means that we differentiate the routines which evaluate the objective and the constraints. Thus, in the future we plan to use an automatic differentiation module [3] rather than a problem dependent module with hand-coded gradients. With these properties, our library becomes fast and robust. The only parts which will always have to be user-specified deal with implementing the objective and the constraint functions, and with describing the state problem.

3.5 Computation of the Gradients

Alg. 3 describes the evaluations of the gradients of the objective and of the constraint functions. This module evaluates the partial gradients mentioned in Sect. 3.4 and calls the sensitivity analysis module for the evaluation itself.

Algorithm 3 Computation of the gradients $\nabla\tilde{\varphi}$ and $\nabla\tilde{\mathbf{g}}$

Given α_c
 Call the “Description module”: for the design $\alpha_c \rightarrow$ a deformed discretization $T_h(\alpha_c)$
 Call the FEM preprocessor: for $T_h(\alpha_c)$ and $(V_h, a, l) \rightarrow$ assembled $\mathbf{K}(\alpha_c)$ and $\mathbf{f}(\alpha_c)$
 Call the “Solver”: Solve $\mathbf{K}(\alpha_c) \cdot \mathbf{u}(\alpha_c) = \mathbf{f}(\alpha_c) \rightarrow \mathbf{u}(\alpha_c)$
 Evaluate $\nabla_{\alpha}\varphi(\alpha_c, \mathbf{u}(\alpha_c))$, $\nabla_{\mathbf{u}}\varphi(\alpha_c, \mathbf{u}(\alpha_c))$, $\nabla_{\alpha}\mathbf{g}(\alpha_c, \mathbf{u}(\alpha_c))$, $\nabla_{\mathbf{u}}\mathbf{g}(\alpha_c, \mathbf{u}(\alpha_c))$
 Call the sensitivity analysis module $\rightarrow \nabla\tilde{\varphi}(\alpha_c)$, $\nabla\tilde{\mathbf{g}}(\alpha_c)$

3.6 External PDE Tools

There are three external modules in Fig. 1. The performance of the library strongly depends on the performances of these modules. There are two base components, namely a mesh generator and an FEM preprocessor, which deals with the preprocessing the optimization problem.

The mesh generator is called once at the beginning but it is the crucial module considering the quality of the mesh and the possibility to mesh complicated real-life geometries as well as to provide various refinements.

The FEM preprocessor is called several times during a run of the optimizer. That is why a fast assembling of stiffness matrices and load vectors is required. Moreover, the FEM preprocessor should be able to deal with various linear elliptic problems.

On top of these external modules, a solver for linear algebraic systems is built. This is the main processing module. It should provide a fast solution of linear systems with a sparse symmetric positive definite matrix for different right hand side vectors.

4 Implementation Based on Advanced PDE Tools

4.1 The PDE Tools NETGEN, FEPP and PEBBLES

Throughout this paper we have considered optimization problems governed by PDEs. The latter are treated by finite element models and are thus represented by a system of equations.

Even 3D problems with moderate geometries lead easily to system with several 10^5 unknowns taking into account appropriate accuracy requirements. Handling the geometries, generation of the finite element meshes, generation of the systems and the fast solution of the large-scale systems require advanced software tools. Nowadays linear system with 10^5 unknowns can be solved within minutes on PCs using appropriate preconditioned iterative solvers. Multigrid methods (geometric and algebraic) have been shown to be very efficient [9, 14] and robust and can be applied to a variety of problem classes, e.g., linear elasticity and magnetics.

NETGEN [15] provides a mesh generator for 3D geometries. Input data can be supplied in CSG (constructive solid geometry) or STL (surface triangulation) or STEP AP 203 (standard for the exchange of product model data) format, i.e., an interface to commercial tools is available. NETGEN generates tetrahedral meshes suited for finite element calculations. Moreover, NETGEN provides projection methods, i.e., near-boundary points can be projected to those boundaries which are exactly represented internally.

Furthermore, flexible PDE tools are required. FEPP [9] provides enough flexibility together with state-of-the-art methods for this purpose. So matrix generation routines are available for scalar potential, elasticity and magnetic field problems using Lagrange-type and Nédélec-type finite elements, respectively. Even boundary element methods are available which are useful especially for modelling exterior field problems [10]. The components can be chosen from a toolbox using a meta-language. However, the main strength of FEPP is the availability of fast parallel solvers for the resulting system of equations [4]. These solvers are based on multigrid methods. Typically the CPU time required by these methods is proportional to the size of the system being solved. This is a major advantage over all classical direct and iterative solvers. In the geometric case, these methods can be combined with adaptive mesh refinement. A mesh hierarchy is explicitly required by the solver. On the other hand, the matrix equation itself is sufficient for constructing the preconditioner in the case of algebraic multigrid solvers. The tool PEBBLES [13] provides these kind of solvers together with an easy to use interface to existing FE-codes.

4.2 Making Use of the PDE Tools in the Library

We build our shape optimization library on the top of the PDE tools NETGEN, FEPP and PEBBLES. Even if we have developed the library only for 4 months, we have already exploited the features provided by NETGEN and the flexibility of FEPP. Namely, we have made use of the CSG model in NETGEN and of the flexible state problem description in FEPP. NETGEN provides an easy and general description of the geometry together with accessing the interfaces and the mesh nodes which is especially of use in the shape optimization. In FEPP we describe the state problem by the meta-language by means of components of the bilinear and of the linear form in the weak formulation (3). The latter brings that we do not need to specify any physical notion as long as linear elliptic PDEs are considered. Moreover, the modularity of the code provides an easy implementation of derivatives of the bilinear and of the linear forms with respect to the nodal coordinates, i.e., for the sensitivities of the stiffness matrix and of the load vector.

However, the PDE tools provide a more powerful functionality than we have used so far, mainly, considering PEBBLES and the fast parallel solvers. So far, we have dealt with 2D shape optimization problems in magnetostatics which are still only of several 10^3 unknowns. However, even these shape optimization problems are very advanced. Now we deal with 3D problems in magnetostatics and making use of the fast solvers is straightforward. Moreover, the main topic of our scientific research concerns on hierarchical shape optimal design methods,

thus, making use of adaptive mesh refinement and of mesh hierarchies seems to be promising. In this context we can see a big potential of the library and we also hope that it will become of use for engineers and designers.

4.3 The Current Implementation

We have implemented the structure in Fig. 1 as a library of the package FEPP. This has been done in the *C++* programming language.

Corresponding to Sect. 3.1, there is a general part which implements the SQP optimizer. Out of the classes, there are, e.g., *RangeQPOptimizer* which solves the QP problem and *ReportSQPOptimizer* which solves the SQP problem. Moreover, abstract classes like *Function_C0*, *Function_C1* and *VectorFunction_C1* for continuous, differentiable and differentiable vector functions, respectively, are provided. They are used for defining the objective and the constraint functions.

Considering the “Description module” in Sect. 3.2, we use NETGEN for describing and discretizing the geometry and we use the meta-language in FEPP for describing the state problem. Moreover, we have implemented another command (option) which specifies indices of the shape design boundaries. Coordinates of the nodes lying on these boundaries are taken as the design variables. Considering the design-to-mesh mapping we implemented an abstract class *MeshDeformer* providing an interface for the design-to-shape and/or the shape-to-mesh mapping. We also implemented particular design-to-shape and shape-to-mesh mappings which have been used for the example given in Sect. 5. We currently work on more general mappings of that kind.

Corresponding to Sect. 3.3 and to Sect. 3.5, we implemented an abstract class *StateDepVecFunction_C1* which supplies evaluation of the function $\tilde{\Psi}(\boldsymbol{\alpha})$ as well as its gradient. Components of this vector function are the objective and the constraint functions which are dependent on the solution of the state problem. The user has to overload the class and re-implement only the methods evaluating the functions $\varphi(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$, $\mathbf{g}(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$ and their partial gradients $\nabla_{\boldsymbol{\alpha}}\varphi(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$, $\nabla_{\mathbf{u}}\varphi(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$, $\nabla_{\boldsymbol{\alpha}}\mathbf{g}(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$, $\nabla_{\mathbf{u}}\mathbf{g}(\boldsymbol{\alpha}_c, \mathbf{u}(\boldsymbol{\alpha}_c))$. Note that hand-coding the gradients will be replaced by an automatic differentiation module later.

Finally, the sensitivity analysis module was already completed. There is a class *SensitivityAnalyzer* which implements efficiently both the direct and the adjoint variable method. We have also extended FEPP to provide the sensitivities of the bilinear and the linear forms for supported kinds of finite elements. At the moment sensitivities for the Laplace bilinear form and for the Lagrange elements are implemented.

5 Example

5.1 The Physical Problem

Let us consider an electromagnet of the so-called Maltese Cross geometry, see Fig. 2. This is used for generating a homogeneous magnetic field. The electromagnet is applied for measurements of magneto-optical effects in a cubic crystal by means of polarization of rays. The device has been developed by the team around Prof. J. Pištora at Department of Physics at VŠB-TU Ostrava. The optimization aims at the optimal shapes of the pole heads in order to minimize inhomogeneities of the field.

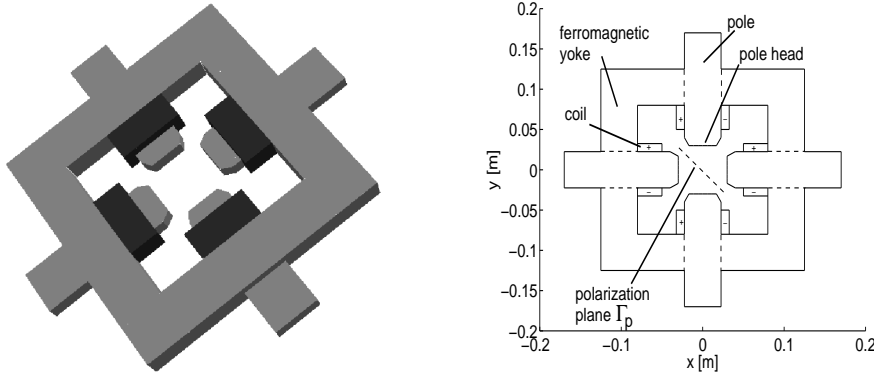


Fig. 2: The Maltese Cross and its cross section

First we will describe the physical problem. The device consists of a ferromagnetic yoke, 4 poles, and 4 windings. A sample of a magneto-optical material is placed to the center. Rays having a defined polarization come to the center and reflect from the sample. A polarization of the reflected rays is different from the defined original one. Components of the reflected polarization vector are measured. The magnetic field in the center is to be as homogeneous as possible. We can change the magnetization direction by switching the polarity of two currents.

5.2 The Shape Optimization Problem

First we briefly introduce the underlying linear magnetostatic problem. Let $\overline{\Omega}$ be a computational domain. Let \mathbf{B} denote the magnetic induction, \mathbf{H} be the magnetic strength density, and \mathbf{J} be the current density. Then the Maxwell equations formally read

$$\left. \begin{array}{l} \mathbf{rot}(\mathbf{H}) = \mathbf{J} \\ \mathbf{B} = \mu\mathbf{H} \\ \mathbf{div}(\mathbf{B}) = 0 \end{array} \right\} \text{ in } \Omega \quad (11)$$

with the corresponding interface conditions and mixed boundary conditions. We introduce the magnetic vector potential \mathbf{A} as follows :

$$\mathbf{rot}(\mathbf{A}) = \mathbf{B} . \quad (12)$$

This state problem is discretized by the FEM. Looking at a typical cross-section, we obtain reduced 2-dimensional Poisson's problem, see [11] for details.

Now we introduce the shape optimization problem in a continuous setting. Let Γ_p be the polarization plane, see Fig. 2. Let α be a function being the graph of the shape of the pole head. Since it is possible to generate the homogeneous magnetic field in different directions, the geometry of the considered electromagnets is always symmetric. That implies the same and symmetric shapes of all the pole heads. Independently from the geometry of the electromagnet,

the following continuous shape optimization problem in 2D (3D) is considered :

$$\left\{ \begin{array}{l} \min_{\boldsymbol{\alpha} \in F} \tilde{\varphi}(\boldsymbol{\alpha}) \\ \text{under} \\ \mathbf{B}_{\boldsymbol{\alpha}} \text{ solves (11)} \\ \mathbf{B}_{\min}^{\text{avg}} \leq \|\mathbf{B}_{\boldsymbol{\alpha}}^{\text{avg}}\| \\ \alpha_1 \leq \boldsymbol{\alpha} \leq \alpha_u \\ -\frac{1}{\rho_{\min}} \leq \frac{\partial_{x_i, x_i}^2 \boldsymbol{\alpha}}{[1+(\partial_{x_i} \boldsymbol{\alpha})^2]^3} \leq \frac{1}{\rho_{\min}} \text{ for } i = 1, 2 \end{array} \right. \quad (13)$$

where

$$\tilde{\varphi}(\boldsymbol{\alpha}) := \frac{1}{\text{meas}(\Gamma_p) \cdot \|\mathbf{B}_{\boldsymbol{\alpha}}^{\text{avg}}\|^2} \cdot \int_{\Gamma_p} \|\mathbf{B}_{\boldsymbol{\alpha}}(\mathbf{r}) - \mathbf{B}_{\boldsymbol{\alpha}}^{\text{avg}}\|^2 ds ,$$

$$\mathbf{B}_{\boldsymbol{\alpha}}^{\text{avg}} := \frac{1}{\text{meas}(\Gamma_p)} \cdot \int_{\Gamma_p} \mathbf{B}_{\boldsymbol{\alpha}}(\mathbf{r}) ds ,$$

$$F := \{ \boldsymbol{\alpha} \in C^{0,1} \mid \boldsymbol{\alpha} \text{ is symmetric} \} .$$

Note that the first inequality constraint in (13) prescribes a minimal magnetic field $\mathbf{B}_{\min}^{\text{avg}}$ necessary for the polarization effect and the last constraint is a regularity constraint on the minimal curvature radius ρ_{\min} of the shape $\boldsymbol{\alpha}$. By $\mathbf{B}_{\boldsymbol{\alpha}}(\mathbf{r})$ we denote a magnetic flux density for the given design $\boldsymbol{\alpha}$ at the given point \mathbf{r} .

5.3 Numerical Results

At the end we give some results for the Maltese Cross electromagnet which were calculated by using the library. First, a comparison of the adjoint method with the numerical differentiation method is presented in Fig. 3. We point out the last column in the table where CPU times show remarkable differences between the methods, even, for a small number of design variables. Note that for both the methods we employed a hierarchical optimization strategy which is another point of great importance within our research. This briefly means that several discretized optimization problems are solved sequentially such that they approximate the problem finer at higher levels and the optimized design is used as the initial design at the next level.

Even if shape optimization tools for 3D are still not tuned enough, we can present a first 3D result, see Fig. 4. Note that an optimized 2D coarse design was produced and we improved almost 10 times the objective value. At Department of Physics, VŠB-TU Ostrava, measurements were done afterwards. They showed a very good correspondence with the computations and even a better improvement in terms of the objective value. We refer to [11] for more details.

6 Conclusions

We introduced the structure of a library for solving shape optimization problems governed by PDEs. Within our following work we will complete the implementation of the designed shape optimization library. The user-interface is still not friendly enough, thus, we will tune the interface to the mesh generator NETGEN for the purposes of shape optimization. So far, we have used the library for the problems in magnetostatics only. We will implement the sensitivities for the other elliptic operators and use the library in solid mechanics, as well. We

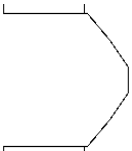
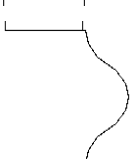
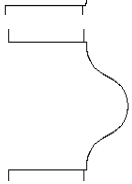
optimized designs	# of des. variables	# of unknowns	# of SQP iters.	CPU time [s]
	3	482	7	8.45
			vers. 6	vers. 17.34
	6	1469	19	52.59
			vers. 18	vers. 126.33
	11	3331	26	117.80
			vers. 24	vers. 350.68

Fig. 3: The adjoint variable method versus numerical differentiation

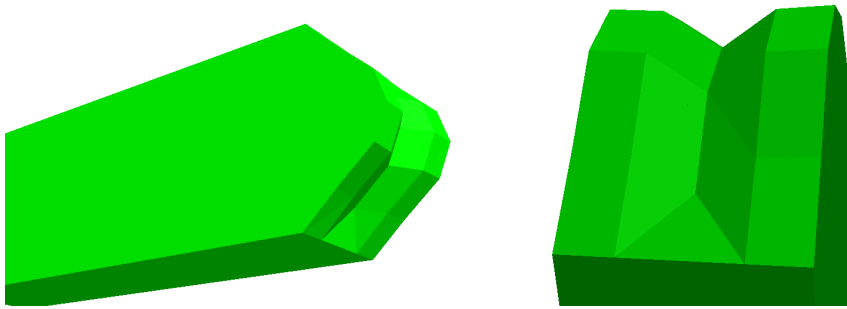


Fig. 4: A 3D optimized pole head

will also incorporate an automatic differentiation module instead of the hand-coded module calculating the partial gradients. However, the main topic of our research will be finding a more general shape-to-mesh mapping. All these improvements will satisfy our main goal, namely, minimizing the effort which is necessary for describing a new shape optimization problem. Note that we also study the hierarchical optimization methods and we will exploit the fast multigrid solvers in shape optimization.

Acknowledgements: *This work has been done during a 1 year stay of the first author at the SFB “Numerical and Symbolic Computing” in Linz. The research has been supported by the Austrian Science Fund FWF within the SFB “Numerical and Symbolic Computing” under the grant SFB F013 and by the Czech Ministry of Education under the research project CEZ J:17/98:272400019.*

Bibliography

1. E. Arian and V. N. Vatsa. A preconditioning method for shape optimization governed by the Euler equations. *INTJCF*, 12(1):17–27, 1999.
2. R. B. Brandtstätter, W. Ring, Ch. Magele, and K. R. Richter. Shape design with great geometrical deformations using continuously moving finite element nodes. *IEEE Transactions on Magnetism*, 34(5):2877–2880, September 1998.
3. A. Griewank. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*, volume 19 of *Frontiers in Applied Mathematics*. SIAM, Philadelphia, 2000.
4. G. Haase, M. Kuhn, and U. Langer. Parallel multigrid 3d maxwell solvers. *Parallel Comp.*, 2001. Accepted.
5. G. Haase and E. H. Lindner. Advanced solving techniques in optimization of machine component. *Computer Assisted Mechanics and Engineering Sciences*, 6:337–343, 1999.
6. J. Haslinger and P. Neittaanmäki. *Finite Element Approximation for Optimal Shape Design: Theory and Applications*. John Wiley & Sons Ltd., Chichester, 1988.
7. Denmark Institute of Mechanical Engineering, Aalborg University. *The Optimum DESign SYSTEM*. <http://www.ime.auc.dk/afd3/odessy/manuals/index.htm>.
8. M. Kaltenbacher, S. Reitzinger, and J. Schöberl. Algebraic multigrid for solving 3d nonlinear electrostatic and magnetostatic field problems. In *IEEE Transactions on Magnetism*, editor, *Selected Papers from the 12th Conference of the Computation of Electromagnetic Fields (COMPUMAG '99)*, volume 36, pages 1561–1564, July 2000.
9. M. Kuhn, U. Langer, and J. Schöberl. Scientific computing tools for 3d magnetic field problems. *The Mathematics of Finite Elements and Applications (MAFELAP X)*, pages 239–259, 2000.
10. M. Kuhn and O. Steinbach. FEM–BEM coupling for 3D exterior magnetic field problems. In T. Tiihonen and P. Neittaanmäki, editor, *ENUMATH 99 - Proceedings of the 3rd European Conference on Numerical Mathematics and Advanced Applications, Jyväskylä, Finland, July 26-30, 1999*, pages 180–187, Singapore, 2000. World Scientific.
11. D. Lukáš. Shape optimization of homogeneous electromagnets. In *Proceedings of SCEE 2000*, Lecture Notes in Computational Science and Engineering. Springer, 2001. To appear.
12. J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research. Springer, 1999.
13. S. Reitzinger. PEBBLES – user's guide. SFB "Numerical and Symbolic Scientific Computing", <http://www.sfb013.uni-linz.ac.at>.
14. M. Schinnerl, J. Schöberl, M. Kaltenbacher, and R. Lerch. Multigrid methods for the fast numerical simulation of coupled magnetomechanical systems. *ZAMM*, 1999. To appear.
15. J. Schöberl. NETGEN - An advancing front 2D/3D-mesh generator based on abstract rules. *Comput. Visual. Sci.*, pages 41–52, 1997.
16. J. Sokolowski and J.-P. Zolesio. *Introduction to Shape Optimization*. Number 16 in Springer Series in Computational Mathematics. Springer, Berlin, 1992.
17. N. Takahashi. Optimization of die press model. In *Proceedings of the TEAM Workshop in the Sixth Round*, Okayama, Japan, March 1996.
18. The MathWorks, Inc. *MATLAB Optimization Toolbox User Manual*, 1993.

Reviewer: Prof. Ing. Jaromír Pištorá, CSc., Institute of Physics, VŠB-TU Ostrava