Abstract

This contribution contains an approach to the optimisation of slender compressed structures which leads to designs being robust against geometrical imperfections. In the developed optimisation model, the shape of the structure depends not only on the optimisation variables but also explicitly on a set of imperfection variables. The model of geometrical imperfections is based on the concept of random fields while for the uncertainty analysis a convex model is employed. The implementation of the presented method takes place in a software framework consisting of distributed software components communicating via CORBA. Each of the components is responsible for a certain aspect of the overall problem.

Keywords: structural optimisation, geometrical imperfections, random fields, convex model, distributed components, CORBA.

1 Introduction

Optimisation of slender structures which are subjected to compressive stresses requires careful attention to geometrical imperfections. In particular, it is crucial to pay attention to the fact that the influence of a given imperfection shape strongly depends on the structural design. Hence, it is not possible to prescribe an imperfection shape at the beginning and to keep it unchanged during the optimisation process.

In structural optimisation, the original engineering design task is expressed as a mathematical optimisation problem. In this contribution, the focus is on the scalar valued continuous optimisation problem

\[ \min_x f_0(x) \text{ s.t. } f_i(x) \leq 0, \ i = 1, \ldots, m \]  

(1)

where x is the n-dimensional design vector, \( f_0 \) is the objective function and \( f_1, \ldots, f_m \)
are constraints. In the following, the \( m + 1 \) functions \( f_0, \ldots, f_m \) will be collectively called problem functions. These problem functions are set up by means of an optimisation model which has to incorporate all relevant design criteria.

In this contribution, an optimisation model for slender structures subjected to compressive loading is presented. In distinction to previous work like [1] or [2], the uncertain nature of geometrical imperfections is explicitly taken into account. Hereby, the approach is similar to e.g. [3], [4] or [5] but the scope of application is extended to complex structures from the field of civil engineering requiring advanced concepts of geometrical modelling.

The optimisation model consists of several interacting sub-models. The geometry model describes the “ideal” shape of the structure in terms of the design vector \( x \in \mathbb{R}^n \). Possible deviations from the “ideal” shape are introduced by the imperfection model which takes the imperfection vector \( y \in \mathbb{R}^q \) as input. Together, these two sub-models constitute the model of the imperfect geometry providing the structural variables \( z \in \mathbb{R}^N \). Taking the structural variables as input, the analysis model computes the structural responses (displacements, stresses) \( u \in \mathbb{R}^M \). In order to capture the stability behaviour correctly, a geometrical nonlinear analysis using a Newton-Raphson scheme is necessary. Despite the uncertainties present in the model, unambiguous statements about the overall structural performance have to be made. This is the purpose of the uncertainty model. Finally, the numerical values of the problem functions \( f_0(x), \ldots, f_m(x) \) are computed on the basis of the output of the uncertainty model. Figure 1 shows the overall optimisation model along with its connection to an optimisation algorithm.

![Diagram](image)

**Figure 1:** Schematic view on the optimisation model
Structural optimisation, at least for non academic problems, introduces a considerable amount of complexity and thus poses high demands on the employed software design. Since large portions of the problem domain are rather independent (think of mathematical optimisation and finite element analysis linked only by structural optimisation) the component oriented programming paradigm seems very promising. Basically, the idea behind component-oriented programming is to build the application of interchangeable and reliable components much like in the field of electronics or mechanics. Also, a legacy application can, when adequately wrapped, be used as a software component. When an application is built of components located on different computers, the term distributed component architecture is used. In this context, the vendor independent Common Object Request Broker Architecture (CORBA) [6] standard is widely established. It defines a Interface Definition Language (IDL) as well as the Internet Inter Orb Protocol (IIOP) such that CORBA components can be used transparently over the network, implemented in almost any programming language and run on almost any platform.

The software framework for structural optimisation presented in this paper comprises several server components (mostly implemented in C++) being glued together by a client application. Since the computational intense tasks are performed inside the server components, the client application is conveniently implemented in Java without seriously affecting performance.

In the following section, the software framework in which the implementation of the optimisation model takes place is described. The following next three sections discuss the theoretical background of the individual sub-models, except the structural model where standard nonlinear finite element techniques are used. Additionally, at the end of each section, some notes about the implementation of the sub model will be given. Finally, an application example illustrates how the software and the optimisation model are used to optimise an engineering structure.

2 Distributed Software Components

The individual software components to be presented are based on object-oriented design principles and rely heavily on the corresponding abstraction principles. Throughout the text, the Unified Modelling Language UML [7] is used to describe the design of the components. It is assumed that the reader is already familiar with object-oriented software design as well as with the UML.

According to the problem domains identifiable in Figure 1, the following three software components are introduced: (i) an optimisation component providing various optimisation algorithms, (ii) a optimisation model component providing the elements needed for the modelling part and (iii) a finite element analysis component for the simulation of structural behaviour. Each of these components is realized as a CORBA server and equipped with a client component for user interaction. The user interface – either graphical or command line driven (both variants have been implemented) – is
composed of the individual client components. Figure 2 shows the overall architecture of the framework in a UML component diagram.

![Component Diagram](image)

Figure 2: Component diagram of the framework’s architecture

Because IIOP is used for inter-object communication, each server component may be located on a different computer. Typically, the server components would located on one or more powerful compute server whereas the user interface runs on a desktop computer [8].

### 2.1 Component for Mathematical Optimisation

Nowadays, a wide variety of optimisation algorithms have been developed, but experience as well as theoretical investigations [9] show that there is no “best” method solving each optimisation problem fast and reliable. For practical structural optimisation it is moreover necessary to have a set of optimisation methods at hand and to be able to switch easily between them. Therefore, an optimisation component has been created [10] which fulfils the above requirement by providing the optimisation methods and packages listed in Table 1.

<table>
<thead>
<tr>
<th>Name</th>
<th>Method(s)</th>
<th>Author</th>
<th>Impl. language</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOT</td>
<td>FR/BFGS/SLP/MMFD/SQP</td>
<td>Vanderplaats Inc. [12]</td>
<td>Fortran</td>
</tr>
<tr>
<td>EVOL</td>
<td>Evolutionary strategy</td>
<td>Grill [13]</td>
<td>C++</td>
</tr>
<tr>
<td>FSQP</td>
<td>Feasible SQP</td>
<td>Lawrence et al. [14]</td>
<td>C</td>
</tr>
<tr>
<td>NLPQL</td>
<td>SQP</td>
<td>Schittkowski [15]</td>
<td>Fortran</td>
</tr>
<tr>
<td>SCP</td>
<td>CONLIN/MMA/SCP</td>
<td>Zillober [16]</td>
<td>Fortran</td>
</tr>
</tbody>
</table>

Table 1: Provided optimisation algorithms and packages

The design of the optimisation component shall be elucidated by analysing the individual steps performed before, during and after an optimisation run. In the beginning,
the optimiser gets the actual optimisation problem passed and an appropriate optimisation method is chosen. Hereupon the optimisation process can be started, which is basically an iterative procedure. Inside the optimisation loop mainly two things happen: At first, the problem functions and, if needed, their derivatives are evaluated. In the case that derivatives are needed but can not be computed analytically, they are numerically approximated by finite differences. Secondly, the optimisation algorithm, according to its internal logic, generates the next, improved design vector based upon the previous function evaluations. These two steps are repeated until some termination criterion is met. Since for most practically relevant structural optimisation problems this is a time consuming procedure, the engineer using the system must be provided with information about the optimisation progress in order to be able to cancel the optimisation process when things go wrong. Finally, when the optimisation run has been terminated, results and the iteration history can be reviewed.

Out of the mentioned tasks, obviously only the generation of the improved design vector is specific to a particular optimisation algorithm. This makes it possible to implement a large portion of functionality in a framework-like manner; individual optimisation algorithms then are plugged into this framework with little effort. In the software model (Figure 3), each of the depicted classes is responsible for a part of the process outlined above. For the sake of clarity, the class diagram contains only a representative subset of the methods provided by the individual classes.

![Class diagram of the optimisation component](image-url)

Figure 3: Class diagram of the optimisation component
2.2 Optimisation Model Component

The basic idea of the optimisation model component is to decompose the overall computation performed inside the optimisation model into elementary data processing tasks and to represent these by so-called process objects. A process object can represent an elementary operation like the addition of two vectors as well as a complex finite element simulation; it must only be able to transform numerical input into numerical output. As an example, Figure 4 schematically shows a process object performing a finite element analysis. Having a set of process objects at hand, a wide variety of optimisation models can be set up by adequately linking together the right process objects in a network. In this, the presented approach is similar to how complex electronic devices (like e.g. computers) are built of prefabricated elements.

![Figure 4: Finite element analysis process object](image)

In the object-oriented software model of this concept, the properties all types of process objects have in common are encapsulated in an abstract base class. Primarily, this class declares an abstract method which invokes the computation of the process object’s output data. Each concrete process object implements this method according to its specific purpose. Furthermore, it should be possible to access the input data and the output data of a process object. For this purpose, so-called port objects are introduced. According to the different schemes data is usually organised in (single numbers, lists of numbers, coordinates or lists of coordinates), different types of port objects exist. Any process object creates the port objects it provides when it is instantiated. Since the port objects are identified by self-explanatory strings, a process object can be asked for its input and output ports during runtime. By that, a semantic layer for the access to input and output data is introduced which allows for a high level of flexibility in the implementation and in the use of process objects.
Having introduced the unified interface for process objects, it becomes possible to arbitrarily link them together in a network. For that purpose so called connection objects are employed. A connection object holds a reference to the port object it belongs to and has a method returning the current value of the referred quantity. Since the way how the referred quantity is stored by the connection object, several specialisations for the different type of output port objects are necessary.

In order to connect the network of process objects to the optimisation component, an optimisation problem adapter is introduced which implements the optimisation problem interface (see Figure 3). This adapter consists of a design evaluation object which holds references to several connections, each of which represents one of the problem function $f_i$ and a design object providing the current values of the design variables.

Whenever the numerical values of the problem functions are needed, each process object must compute its output values whereas a correct execution sequence must be ensured. This task is delegated to a network object class being the base class for process objects as well as for connections. It holds a list of its predecessors and has a method to invoke the execution of its preliminary subnet.

Figure 5 shows the class diagram of the optimisation model component. Various process objects are introduced in the next sections while a network of process objects for a specific design problem is given in section Section 6.
2.3 Finite Element Analysis Component

For structural analysis, two different software components have been developed. The first one is an object-oriented interface to the procedural public domain software FElt [17]. The second interacts with the commercial finite element system ANSYS. Both components are incorporated in the optimisation model component by process objects which provide access to the input and output data of the corresponding finite element model. For further details, the reader is referred to [18].

3 Geometry Description

In the geometry model, the shape of the structure is described hierarchically by several higher level geometrical elements. This allows for an efficient control of the overall shape using only a few key parameters. Also, this simplifies the integration of a model of geometrical imperfections.

The starting point for the geometry model is the definition of a so called basis element which represents the overall shape of the structure. This basis element is a curve for one-dimensional structures, like curved beams or arches, and a surface for two-dimensional structures, like shells or grid shells. This basis element is represented by means of Non Uniform Rational B-Splines (NURBS), which are nowadays widely used in computer aided geometric design. Along with the imperfection model the imperfect basis element is defined being the reference frame for the definition of a so called relative geometry. Finally, the relative geometry defines the coordinates of the finite element nodes. The described scheme is illustrated in Figure 6 and applied to one-dimensional structures in the section.

3.1 One-dimensional Structures

The basis element of one-dimensional structures is the system curve

\[ \mathbf{r}^p(t) = \begin{bmatrix} r_1^p(t) \\ r_2^p(t) \\ r_3^p(t) \end{bmatrix}^T \]

which is marked as perfect by the superscript \( p \). Here, \( r^p \) is a NURBS. The control points and weights of the NURBS are linked to the corresponding entries of the design.
vector \mathbf{x}. For the sake of a compact notation, this dependency is not included in (2). A trihedron having the base vectors \( \mathbf{e}_1^p(t), \mathbf{e}_2^p(t) \) and \( \mathbf{e}_3^p(t) \) and following the perfect basis element is defined. In the case of one-dimensional structures, the imperfection shape depends only on the curve parameter \( t \) and has two components:

\[
\mathbf{h}(t) = [h_1(t), h_2(t)]^T, \quad (3)
\]

Again, the dependency of \( h(t) \) from \( x \) and \( y \) is not explicitly noted. In accordance to Figure 7(a), the imperfect system curve yields to

\[
\mathbf{r}^i(t) = \mathbf{r}^p(t) + h_1(t) \cdot \mathbf{e}_2^p(t) + h_2(t) \cdot \mathbf{e}_3^p(t). \quad (4)
\]

Figure 7: Geometry model for one-dimensional structures

Then again, a following trihedron is attached to the imperfect system curve. This trihedron serves as a reference frame for the relative geometry where the remaining dimensions of the system are again described by NURBS. The final results are the curves on which the nodes of the finite element model are located. Figure 7(b) exemplifies the relative geometry for a girder having three chords. Given the function \( h_s : \mathbb{R} \rightarrow \mathbb{R} \) which is the height and the function \( b_s : \mathbb{R} \rightarrow \mathbb{R} \) representing the width, the curve of the top chord

\[
\mathbf{g}_1(t) = \mathbf{r}^i(t) + h_s(t) \cdot \mathbf{e}_2^i(t) \quad (5)
\]

and the curves of the lower chords

\[
\begin{align*}
\mathbf{g}_2(t) &= \mathbf{r}^i(t) - h_s(t) \cdot \mathbf{e}_2^i(t) + b_s(t) \cdot \mathbf{e}_3^i, \\
\mathbf{g}_3(t) &= \mathbf{r}^i(t) - h_s(t) \cdot \mathbf{e}_2^i(t) - b_s(t) \cdot \mathbf{e}_3^i
\end{align*} \quad (6)
\]

are easily obtained. The concept described here for one-dimensional structures can be extended without much effort to two-dimensional structures like grid shells (see [18]).


3.2 Process Objects for Geometry Description

Besides the imperfection shape presented in the next section, the outlined geometry model requires a process object representing a NURBS and some additional process objects for the manipulation of coordinates. The class diagram in Figure 8 shows these

![Class diagram of process objects for geometry description](image)

Figure 8: Class diagram of process objects for geometry description

process objects whereas the process object for subtracting coordinates is not explicitly listed. Since runtime information is not included a class diagram (port objects are created dynamically, see Section 2.2), the input ports and the output ports of a process object are provided as additional information.

4 Geometrical Imperfections

Geometrical imperfections can be interpreted as a phenomenon of random nature. Nevertheless, the fluctuations at two different points are not completely independent, rather they are coupled stochastically over their distance.

For the modelling of such spatially varying random quantities the concept of random fields [19] is well suited. An application of random fields to geometrical imperfections is given in [20].

4.1 Scalar Random Fields

Within this paper, the emphasis lies on scalar valued random fields, but an extension to vectorial random fields is straightforward. A scalar random field \( H \) assigns to each point \( x \), within the considered domain \( D \subseteq \mathbb{R}^3 \), a random number \( H(x) \). A realization \( h \) of the random field \( H \) is a function which assigns a number \( h(x) \) to each point \( x \). Applying this concept, the properties of a random field can be stated.

4.1.1 Properties of Random Fields

In each point, a scalar random field has a distribution function

\[
F^H(h(x)) = P[H(x) \leq h(x)].
\]
Moreover, the mean value function

\[ m^H(x) = \mathbb{E}[H(x)] \]  

(8)
gives the expected value of the random field at each point while the covariance function

\[ B^H(x_1, x_2) = \text{Cov}[H(x_1), H(x_2)] \]  

(9)
captures the stochastic coupling of the fluctuations of two arbitrary points.

When random fields are employed to model real world phenomena, the properties introduced above have to be adequately identified. Usually, geometrical imperfections do not show a preferred direction and an accumulation around the perfect geometry has to be expected. Therefore, in this work the random field is Gaussian, having a mean value of zero. In addition a homogenous correlation structure is used. This means that the value of the covariance function depends only on the relative position of two points. The covariance function thus depends on a scalar parameter \( \xi = \xi(x_1, x_2) \) which reflects the metric of the considered domain. Using these assumptions and an inverse exponential correlation function, the covariance function writes as

\[ B^H(\xi) = \sigma^2 e^{-\xi/b} \]  

(10)

where \( \sigma \) is the standard deviation and \( b \) is called correlation length.

4.2 Discretisation

For the implementation in a computer program, the above continuous formulation has to be discretised. For that, the random field is considered at discrete points. The original random field is then represented by a vector of random numbers

\[ Z = [H(x_1), \ldots, H(x_p)]^T \]  

(11)
in which \( p \) is the number of discretisation points. The covariance matrix of \( Z \) is computed by means of the covariance function (9) and is

\[ B^Z = \begin{bmatrix} B^H(x_1, x_1) & \cdots & B^H(x_1, x_p) \\ \vdots & \ddots & \vdots \\ B^H(x_p, x_1) & \cdots & B^H(x_p, x_p) \end{bmatrix}. \]  

(12)

Because of the symmetry of the covariance function, the covariance matrix is symmetric and, as can be easily shown, positive definite.

4.3 Transformation in the Uncorrelated Space

Since the covariance function (10) takes positive values for each \( x \in D \), the covariance matrix \( B^Z \) has only non zero entries such that the random vector \( Z \) is fully
correlated. For the treatment of the random field in the uncertainty model, it is necessary to represent the random vector $Z$ in terms of a new random vector $Y$ which has more convenient properties.

Therefore, a set of base vectors $a_1, \ldots, a_q$ which are collected in the matrix $A \in \mathbb{R}^{p-r \times q}$ is introduced. The random vector $Z$ is then represented by the linear combination

$$Z = a_1 Y_1 + \cdots + a_n Y_n = AY.$$ \hfill (13)

The covariance matrices of $Z$ and $Y$ are connected by the set of base vectors $A$ through the relation

$$B^Z = AB^Y A^T$$ \hfill (14)

which can be found by inserting the linear combination (13) in the well known definition of covariance.

The next step is to choose the matrix $A$. This is done by applying a spectral decomposition on $B^Z$. Since $B^Z$ is positive definite, it can be written as

$$B^Z = Q \sqrt{\Lambda} I \sqrt{\Lambda} Q^T.$$ \hfill (15)

where $Q$ is the matrix formed by the eigenvectors $q_i$ of $B^Z$ and $\Lambda$ is a diagonal matrix containing the corresponding eigenvalues. The comparison of (14) and (15) yields that, if

$$A = Q \sqrt{\Lambda},$$ \hfill (16)

then $B^Y$ is the identity matrix $I$. Thus, by employing the basis (16), the imperfection shape is represented by a linear combination of deterministic base vectors scaled by their probabilistic weights and uncorrelated uniformly distributed random numbers.

Figure 9: Base vectors of a one dimensional random field

Figure 9 shows the first five base vectors for a one dimensional random field of length one with $\sigma = 1$ and $b = 0.5$. The left picture corresponds to an unconstrained random field while the right picture shows the base vectors of a random field having a vanishing variability at both ends.
4.4 Random Field Process Object

The process object implementing a one dimensional random field having a Gaussian distribution function and a triangular or inverse exponential correlation function is depicted in (Figure 10). As can be seen from Equation (4), the imperfect basis element is defined in the reference frame following the perfect basis element. Therefore, the random field process object has input ports for the points on the perfect base element and the corresponding base vectors of the following trihedron. Analogously, output ports comprise the points on the imperfect system line and again the base vectors. The imperfection shape itself is controlled by the imperfection variables $y$ which are managed by two additional input port objects; one for each of the directions $e_1^p$ and $e_2^p$.

In order to set the properties of the random field mentioned above, the random field process object provides the corresponding methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setCorrelationType</td>
<td>Set correlation type</td>
</tr>
<tr>
<td>setCorrelationLength</td>
<td>Set correlation length</td>
</tr>
<tr>
<td>setVariance</td>
<td>Set variance</td>
</tr>
<tr>
<td>setNumPts()</td>
<td>Set number of points</td>
</tr>
<tr>
<td>setNumCoeffs()</td>
<td>Set number of coefficients</td>
</tr>
<tr>
<td>getQualityMeasure()</td>
<td>Get quality measure</td>
</tr>
</tbody>
</table>

Figure 10: Class diagram of the process object for random fields

5 Modelling of the Uncertainty

The model of the geometrical imperfections presented introduces a substantial amount of uncertainty in the structure. Following Elishakoff [21], there exist mainly three approaches to handle uncertainty in structural analysis: (i) the classical probabilistic approach in which one aims for failure probabilities, (ii) the option to represent uncertain parameters as fuzzy sets and (iii) convex models of uncertainty. In addition, the concept of fuzzy-randomness introduced recently [22] is an approach which includes both probabilistic and fuzzy concepts.

Since the model of the geometrical imperfections has a probabilistic foundation, the fuzzy approach is not necessary. The probabilistic approach has the disadvantage of being potentially numerically expensive and requiring extensive knowledge of the stochastic properties of the uncertain parameters. Especially the latter argument is essential, because for the imperfections of complex structures it is difficult to acquire sufficient knowledge about probability distributions. For this reasons, a convex model of geometrical imperfections based upon the above imperfection model is presented.
5.1 Convex Modelling

The idea of convex models, which goes back to Ben-Haim and Elishakoff [23], is based upon the assumption of unknown but bounded uncertain parameters. For that, a subset of possible parameters $S_y$ is defined which has the property of being convex.

The authoritative structural response $v_i(x)$ is then the worst structural response for all uncertain parameters contained in the set $S_y$. This condition is formulated as the optimisation problem

$$v_i(x) = \max_y u_i(x, y) \text{ with } y \in S_y$$

(17)

which can be solved using existing optimisation software. Since (17) denotes the search for the worst structural response, convex modelling is often associated with the term anti-optimisation. The overall procedure, therefore, consists of two nested loops: an external optimisation loop to optimise the overall structural system and an internal loop in which the worst imperfection shape for the required response quantity is determined.

The next question is, how to choose the convex set $S_y$ such that the original probabilistic properties of the random vector $Y$ are appropriately captured. Since the entries of $Y$ are normally distributed random numbers, the convex set has to be invariant against rotations of the coordinate system. This requirement is fulfilled by the set

$$S_y = \{ y \in \mathbb{R}^q \mid \|y\| \leq \epsilon \},$$

(18)

which represents a $q$-dimensional ball. The radius $\epsilon$ of the ball is therefore the parameter governing this convex model. The basis for the parameter $\epsilon$ is the characteristic amplitude of the imperfection shape $h_{sp}$. According to [24], the value $h_{sp}$ is chosen as the 95% fractile of all possible imperfection shapes and thus can be determined using the Gaussian distribution function. Taking into account the number of imperfection variables $q$, the radius yields to $\epsilon = \sqrt{q} h_{sp}$.

5.2 Formulation as Unconstrained Problem

Because of the specific structure of the convex set (18) it is possible to transform the constrained optimisation problem (17) into an unconstrained one. This is carried out by means of a projection. Using the new vector of imperfection variables

$$\tilde{y} = [\tilde{y}_1, \ldots, \tilde{y}_{q+1}]^T$$

(19)

the projection $\psi : \mathbb{R}^{q+1} \to \mathbb{R}^q$ with

$$\psi(\tilde{y}) = \frac{\epsilon}{\|\tilde{y}\|} \cdot [\tilde{y}_1, \ldots, \tilde{y}_{q+1}]^T$$

(20)

is defined. The domain of $\psi$ is exactly $S_y$ and

$$\psi(\tilde{y}) \in S_y \quad \forall \quad \tilde{y} \in \mathbb{R}^{q+1}$$

(21)
holds. In Eq. (19), the entry $\hat{y}_{q+1}$ can be interpreted as a slack variable, which allows
the representation of interior points of $S_y$. Using the relation (21) and $\hat{u}(x, \hat{y}) = u(x, \psi(\hat{y}))$ the constrained anti-optimisation problem (17) can be written as

$$v_i(x) = \max_{\hat{y}} \hat{u}(x, \hat{y}) \quad \text{with} \quad \hat{y} \in \mathbb{R}^{q+1},$$

which represents an unconstrained problem.

### 5.3 Derivatives

The optimisation algorithms employed in this work need the derivatives of the problems functions with respect to the design variables. Consequently, the derivatives of the worst structural responses (22) have to be computed.

Based upon the condition (22), a functional dependency amongst $\hat{y}$ and $x$ is formulated:

$$\hat{y} = \hat{y}(x)$$

and the relation

$$v_i(x) = \hat{u}_i(x, \hat{y}(x))$$

can be stated. Using the chain rule of differentiation, the partial derivatives of (24) yields

$$\frac{\partial v_i}{\partial x_j}(x) = \frac{\partial \hat{u}_i}{\partial x_j}(x, \hat{y}(x))$$

$$+ (\nabla_\hat{y} \hat{u}_i(x, \hat{y}(x)))^T \frac{\partial \hat{y}}{\partial x_j}(x).$$

On the other hand, the gradient

$$\nabla_\hat{y} \hat{u}(x, \hat{y}) = 0$$

must vanish due to the necessary condition for the solution of problem (22). Therefore, using $\hat{y}^*$, the result

$$\frac{\partial v_i}{\partial x_j}(x) = \frac{\partial \hat{u}_i}{\partial x_j}(x, \hat{y}^*)$$

is obtained. This is crucial for two reasons: (i) the convex modelling, as performed here, does not influence the $C^1$ continuity of the problem functions and (ii) the anti-optimisation problem does not have to be solved in the perturbation steps of the numerical sensitivity analysis.

### 5.4 Process Objects for Uncertainty Modelling

Two process objects are required by the above uncertainty model: one for the projection of the uncertainty vector into $S_y$ and one for anti-optimisation. The input of the
projection process object are the \( q+1 \) uncertainty variables for which the projected uncertainty vector is computed. In addition, the number of uncertainty variables and the radius of the \( q \)-dimensional ball can be set. Based upon the convex model constituted by the random field process object in conjunction with the projection process object, the anti-optimisation process object determines the worst combination of imperfection variables for a given structural response quantity. For that, it takes the response quantities depending on the imperfection shape as input and provides the imperfection variables and the worst response quantities as output. Because the anti-optimisation does not have to be carried out during sensitivity analysis (see Section 5.3), the anti-optimisation process object needs to know the master optimiser responsible for the overall optimisation. Since it serves as an optimiser as well as defining an optimisation problem, it aggregates the corresponding classes (see Figure 11).

### 6 Application Example

In this section, the optimisation model described above is applied exemplarily to the arched truss girder in Figure 12 which is subjected to a constant area load on the roof area. The material for the structure is steel, the modulus of elasticity is \( E = 2.1 \cdot 10^5 \) N/mm\(^2\) and the density \( \rho = 7850 \) kg/m\(^3\). The structural model consists of 77 nodes
and 209 truss elements having a circular cross section (tubes). The weight of the initial structure is 41022 kg.

Geometrical imperfections are modelled by means of a random field. Since the structure is spatial, the random field implies two components each of which is described by ten imperfection variables. The random field has an inverse-exponential correlation function and a standard deviation of $\sigma = 0.2$ m and a correlation length of $b = 600$ m. In the convex model, the radius of radius of the ball is $\epsilon = 8.85$. Within the geometry model, the structure is described by 21 optimisation variables. Five optimisation variables refer to the wall thicknesses of the cross sections and 16 variables describe the geometry of the structure. As the objective function, the strain energy stored in the deformed system is used. The optimisation is subject to two constraints, (i) the weight of the structure must not exceed 41,300 kg and (ii) the total height of the system is restricted to 40 m. Figure 13 contains the network of process objects establishing the optimisation model.

Figure 13: Network of process objects
The optimisation is carried out using an SQP algorithm [15]. A comparison of the initial and the optimised structure is given in Table 2. Of special interest is the comparison of the strain energy for the perfect configuration (Perf.) and the worst imperfection shape (Imp.). Obviously, the increase in strain energy is much smaller for the optimised structure compared to the initial design. This is visible also in the optimisation history shown in Figure 14 where the lower line belongs to the search for the worst imperfection shape carried out in each iteration step of the main optimisation loop. The layout of the optimised structure is shown in Figure 15.

<table>
<thead>
<tr>
<th></th>
<th>Initial Design</th>
<th>Best Design</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(kNm)</td>
<td>(%)</td>
</tr>
<tr>
<td>Perf.</td>
<td>2134.58</td>
<td>100.00</td>
</tr>
<tr>
<td>Imp.</td>
<td>2412.40</td>
<td>113.02</td>
</tr>
<tr>
<td>Diff.</td>
<td>277.82</td>
<td>13.02</td>
</tr>
</tbody>
</table>

Table 2: Objective Function

Figure 14: Optimisation history

Figure 15: Optimised Structure
7 Conclusion

In this paper, an approach to the optimisation of structural systems with uncertainties induced by geometrical imperfections has been presented. The optimisation model hereby comprises a NURBS based geometry description, random fields for the modelling of geometrical imperfections, a nonlinear finite element analysis and a convex model of uncertainty.

As an instance of a complex engineering structure an arched girder has been optimised. The results achieved show that the proposed approach yields a structural design which is significantly less imperfection-sensitive than the initial design.

References


