Structural and Multidisciplinary Optimization (2021) 63:1633–1651
https://doi.org/10.1007/s00158-020-02821-y

RESEARCH PAPER

Relaxed gradient projection algorithm for constrained node-based shape optimization

Ihar Antonau[1] · **Majid Hojjat**[2] · **Kai-Uwe Bletzinger**[1]

Received: 4 August 2020 / Revised: 13 November 2020 / Accepted: 9 December 2020 / Published online: 10 February 2021

Abstract

In node-based shape optimization, there are a vast amount of design parameters, and the objectives, as well as the physical constraints, are non-linear in state and design. Robust optimization algorithms are required. The methods of feasible directions are widely used in practical optimization problems and know to be quite robust. A subclass of these methods is the gradient projection method. It is an active-set method, it can be used with equality and non-equality constraints, and it has gained significant popularity for its intuitive implementation. One significant issue around efficiency is that the algorithm may suffer from zigzagging behavior while it follows non-linear design boundaries. In this work, we propose a modification to Rosen's gradient projection algorithm. It includes the efficient techniques to damp the zigzagging behavior of the original algorithm while following the non-linear design boundaries, thus improving the performance of the method.

Keywords Gradient-based constrained optimization · Shape optimization · Vertex Morphing · Rosen's gradient projection algorithm · Node-based shape parametrization

1 Introduction

The aim of this paper is to propose a modified algorithm for constrained node-based shape optimization. It has good potential to improve the objective function by finding a new design through the modification of the shape of the initial model. In our paper, we are interested in iterative optimization methods, where a continuous evolution of the design produced. Shape optimization is successfully used in many fields of application: aerospace engineering (Kroll et al. 2007; Kenway et al. 2014; Palacios et al. 2012), automotive industry (Najian Asl et al. 2017; Hojjat et al. 2014), structural mechanics (Chen et al. 2019; Haftka and Grandhi 1986; Firl and Bletzinger 2012), fluid-structure interaction (FSI) (Hojjat et al. 2010; Heners et al. 2017), etc.

General, constrained shape-optimization problems can be formulated as follows:

$$\begin{aligned} &\textbf{minimize}: f(\boldsymbol{x}) \\ &\textbf{design variables}: \boldsymbol{x} \\ \text{s.t.:} &g_j(\boldsymbol{x}) \leq 0, \text{ where } j = 1..n_g \\ &h_k(\boldsymbol{x}) = 0, \text{ where } k = 1..n_h \end{aligned} \tag{1}$$

where $f(\boldsymbol{x})$ is the objective function, $\boldsymbol{x}$ is the vector of design parameters, $g_j(\boldsymbol{x})$ are inequality constraints, and $h_k(\boldsymbol{x})$ are equality constraints. An important step in optimization is the choice of the design variables. In the optimization of the shape (and topology), there are two main types of shape parametrization: explicit and implicit. Implicit parametrization can be presented, for instance, by the free-form deformation (FFD) approach (Sieger et al. 2012) or a level-set method (Wang and Luo 2020; Luo et al. 2008). Alternatively, in the explicit parametrization, such as Vertex Morphing (Bletzinger 2017; Hojjat et al. 2014) or CAD-based parametrization (Xu et al. 2014; Agarwal et al. 2018; Hardee et al. 1999), the representation of the geometry is directly used as a design parameter field. In this work, we are using Vertex Morphing parametrization.

Responsible Editor: Ming Zhou

✉ Ihar Antonau
ihar.antonau@tum.de

[1] Structural Analysis, Technical University Munich, Munich, Germany

[2] BMW Group, Munich, Germany

The main advantage of the Vertex Morphing is no additional optimization model is needed. The analysis model is used directly, where the coordinates of the surface nodes are the design parameters. Isogeometric parametrization (Ummidivarapu and Voruganti 2017; Ummidivarapu et al. 2020) is a good alternative to the Vertex Morphing. Both methods have similarities, and the difference is typically in the number of design variables. Vertex Morphing uses surface nodes of the FE model as a design parameters; therefore, there is a large number of variables. That allows us to find new unknown solutions by changing the parametrization settings. On the other hand, with Vertex Morphing, it is challenging to apply boundaries and geometrical constraints to the design parameters (Najian Asl et al. 2017). The interested reader can find more details about Vertex Morphing and form-finding in Bletzinger (2017), Baumgärtner et al. (2016), Hojjat et al. (2014).

Solving industrial problems is state of the art. The main focuses groups researching shape optimization problems are developing industrial applications, deriving sensitivity analysis w.r.t shape design variables, and finding new designs of the models. In most cases, they use well-established optimization algorithms, such as steepest descent, gradient projection, augmented Lagrangian, or trust-region algorithms. Nonetheless, classical algorithms may suffer from poor efficiency due to the specific properties of the problems. For instance, the active-set methods may suffer from the zigzagging phenomenon (Fletcher 2013; Sun and Yuan 2006) because constraints repeatedly enter and leave the active set. Therefore, it results in slow convergence of the method (Gallagher and Zienkiewicz 1977). Typical properties of the node-based shape optimization problem are:

- A large number of design variables. In the Vertex Morphing practice, the "usual" number is around $10e5 - 10e6$. That makes solving the optimization problem not straightforward;
- The objectives, as well as the physical constraints, are non-linear in state and design;
- The sensitivity analysis for different objective or constraint functions cannot always be solved analytically; thus, they are solved with a tolerance;
- The sensitivities of the different responses have to be scaled due to the different physical units. Scaling may mean that information regarding the size of the raw sensitivities is lost;
- Calculation of the $f(\boldsymbol{x}), \nabla f(\boldsymbol{x}), g(\boldsymbol{x}), \nabla g(\boldsymbol{x})$ is computationally expensive. Doing physical analysis may take up $\approx$ 50–80% of the one optimization iterations computational time;
- Algorithms such as gradient projection that require extra calculations of the response functions to calculate the correction step precisely (we discuss this in the details in the Sections 2.1 and 2.2). This can be numerically expensive or may require additional assumptions and simplifications.
- Line search techniques can be numerically expensive or non-accurate for highly non-linear functions. In practice, a constant step size may be preferred.

In this work, we propose a relaxed gradient projection method. The method is a modification of the classical Rosen's gradient projection algorithm (Rosen 1960, 1961). In this context, "relaxed" means that constraints can be in the transient stage between active and non-active. The relaxation and correction factors mildly control the relaxation and violation of the constraints. In the proposed method, we introduce the buffer (critical) zone to calculate the relaxation factor and the correction term violated constraints. As a result, the algorithm has efficient techniques to damp zigzagging behavior when it follows the design boundaries and has stable performance.

The paper is structured as follows: First, the Rosen's gradient projection algorithm is reviewed as the reference method. Then the proposed algorithm and its simplified version are described. The next section describes the numerical experiments and shows a detailed analysis of the performance of the proposed and reference methods. Finally, conclusions are drawn from the work.

2 Rosen's gradient projection algorithm

This section describes the Rosen's gradient projection algorithm (GP), its advantages, and disadvantages in the context of shape optimization problems. The method is used as the reference algorithm in our studies.

2.1 Gradient projection method

The gradient projection algorithm calculates feasible search direction by projecting the steepest descent direction into the tangent subspace to the active constraints. Detailed description can be found in the article by Rosen (1960) and Du et al. (1990). We will describe the way to calculate the projected search direction for optimization problems with linear constraints by following Haftka and Kamat (1990). The problem can be formulated as follows:

$$\begin{aligned} \textbf{minimize} : f(\boldsymbol{x}) &= \sum_i \omega_i f_i(\boldsymbol{x}), \text{ where } i = 1..n_f \\ \text{s.t.} : g_j(\boldsymbol{x}) &= \boldsymbol{a}_j \boldsymbol{x} - b_j <= 0, \; j = 1, ..., n_g \\ h_k(\boldsymbol{x}) &= \boldsymbol{a}_k \boldsymbol{x} - b_k = 0, k = 1, ..., n_h \end{aligned} \tag{2}$$

If we select only the r active constraints, we can define an n by r matrix N, such that the columns of this matrix are the

gradients of active constraints. The basic assumption of the gradient projection method is that $\boldsymbol{x}$ lies on the tangential subspace to the boundary of the active constraints. If our solutions $\boldsymbol{x}^{(i)}$ and $\boldsymbol{x}^{(i+1)}$ at the iteration i and $i+1$ satisfy the constraints, then the constraints can be rewritten as:

$$N^T \boldsymbol{s} = 0 \tag{3}$$

where $\boldsymbol{s}$ is a search direction. If we want to project the steepest descent direction $-\nabla f$ on the tangent subspace of the active set of constraints, we can redefine problem (2) as follows:

$$\begin{aligned} &\textbf{minimize} : \boldsymbol{s}^T \nabla f \\ &\quad \text{s.t.} : N^T \boldsymbol{s} = 0, \\ &\qquad \text{and } \boldsymbol{s}^T \boldsymbol{s} = 1 \end{aligned} \tag{4}$$

where the second condition bounds the solution. The Lagrangian function is:

$$L(\boldsymbol{s}, \boldsymbol{\lambda}, \mu) = \boldsymbol{s}^T \nabla f - \boldsymbol{s}^T \boldsymbol{N} \lambda - 2\mu(\boldsymbol{s}^T \boldsymbol{s} - 1) \tag{5}$$

The condition for L to be stationary is

$$\frac{\partial L}{\partial \boldsymbol{s}} = \nabla f - \boldsymbol{N}\lambda - 2\mu \boldsymbol{s} = 0 \tag{6}$$

We can find the Lagrangian multiplier λ by multiplying (6) with N^T and using condition from (3):

$$\lambda = (\boldsymbol{N}^T \boldsymbol{N})^{-1} \boldsymbol{N}^T \nabla f \tag{7}$$

and the feasible search direction s:

$$\boldsymbol{s} = \frac{1}{2\mu}[\boldsymbol{I} - \boldsymbol{N}(\boldsymbol{N}^T \boldsymbol{N})^{-1} \boldsymbol{N}^T] \nabla f \tag{8}$$

In Najian Asl et al. (2017) and Haftka and Kamat (1990), the authors observe that the factor $\frac{1}{2\mu}$ does not play an important role in the determination of search direction because it scales the vector and does not change its direction. The final search direction to minimize the objective function can be changed with sign factor "-".

To find the Lagrangian multiplier λ in (7), the linear system of equation of size $r \times r$ needs to be solved. Depending on the number of active constraints r and design variables n, the constraint matrix N can be sparse and large. The condition number of such a system can be large; therefore, special attention should be paid to the choice of an efficient and robust linear solver. The reader may refer to Najian Asl et al. (2017) for more details on solving (7). After finding the feasible search direction, new shape $\boldsymbol{x}_{i+1}$ can be found. A line-search can be used to find the step size $\alpha^{(i)}$ that sufficiently reduces the objective function or a constant step size can be used. Design update can be found as follows:

$$\boldsymbol{x}^{(i+1)} = \boldsymbol{x}^{(i)} + \alpha^{(i)} \boldsymbol{s} \tag{9}$$

2.2 Reduced gradient projection algorithm

Rosen's work (1961) provides an extension to the gradient projection algorithm to handle non-linear constraints. The main idea is to calculate the correction (restoring) move that can bring violated constraints back into the feasible domain. To calculate the restoring move, we linearize the constraint:

$$g_j \approx g_j(\boldsymbol{x}^{(i)}) + \nabla g_j^T (\bar{\boldsymbol{x}}^{(i)} - \boldsymbol{x}^{(i)}) \tag{10}$$

Using the linearized equation of the constraints, we can find the correction move:

$$\begin{aligned} \bar{\boldsymbol{x}}^{(i+1)} - \tilde{\boldsymbol{x}}^{(i+1)} &= -\boldsymbol{N}(\boldsymbol{N}^T \boldsymbol{N})^{-1} \boldsymbol{g_a} \\ g_{a,j} &= g_j(\tilde{\boldsymbol{x}}^{(i+1)}) \end{aligned} \tag{11}$$

where $\tilde{\boldsymbol{x}}^{(i+1)}$ is a new design after minimizing in the tangential direction, (8), $\bar{\boldsymbol{x}}^{(i+1)}$ is the corrected design, and $\boldsymbol{g_a}$ is a vector which contains the violations of the active constraints. Equation (11) is based on the linear approximation and therefore should be repeated several times, until $\boldsymbol{g_a}$ is sufficiently small. In addition, the matrix N should be re-evaluated for each point, which means all physical solvers should be deployed again in order to undertaken a sensitivity analysis for active constraints. In the industrial case, the deployment of physical solvers can be very time consuming. Hence, it can be computationally expensive to calculate a correction move several times, or even just once. To reduce computation cost, one can use the violation of the constraint g_j at the beginning of the iteration, $g_{a,j} = g_j(\boldsymbol{x}^{(i)})$. With this assumption, the correction move might not bring the $\boldsymbol{x}_{i+1}$ back on the design boundary. Therefore, in one or more optimization iterations, the active constraint would become non-active ($g_j(\boldsymbol{x}^{(i+1)}) < 0$). In this case, the algorithm would perform a steepest descent step, like other feasible direction methods (Vanderplaats 2007), and violate the constraint again in the next iteration. That leads to zigzagging behavior of the algorithm, and the reduction of its performance (Fletcher 2013; Sun and Yuan 2006).

2.3 Numerical example

Figure 1 gives the typical diagram with the zigzagging behavior of the gradient projection algorithm with a constant step size. After some initial iterations, the non-linear design boundary is reached and overshot. On the next iteration, the algorithm calculates the projected direction and applies the correction move to bring the solution back

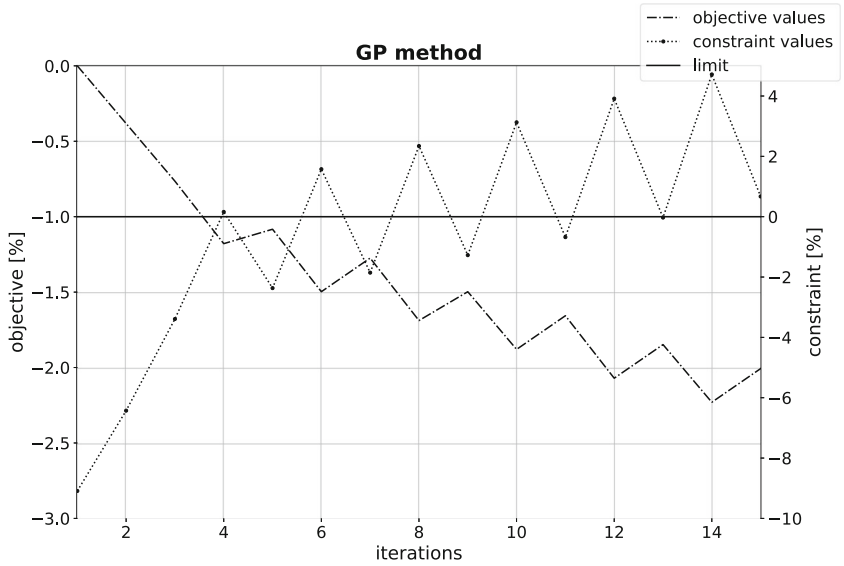


Fig. 1 Zigzagging behavior of GP method

to the feasible side. This leads to the zigzagging of the objective and constraint values.

3 Relaxed gradient projection algorithm

To overcome issues with gradient projection methods in our optimization problems, we introduce the proposed method, a relaxed gradient projection algorithm (RGP). It incorporates techniques for damping the zigzagging behavior of the algorithm, while following non-linear active constraints. This section describes the RGP method and its simplified version (SRGP).

3.1 Buffer (critical) zone

The gradient projection algorithm has an issue with switching on and off the constraint while following the design boundary. To avoid switching, we introduce the buffer (critical) zone. The buffer (critical) zone is the region where the constraint is considered as active. Inside this zone, we calculate the buffer coefficient $\omega_j^{(i)}$, which defines how "strongly" the constraint should be considered. If the constraint value has not reached the limit value, the $\omega_j^{(i)}$ coefficient makes the constraint "weaker." On the other hand, if the constraint value is on the limit or has violated the limit, the $\omega_j^{(i)}$ coefficient makes the constraint fully active. It smoothly varies from zero to two, where "zero" means that constraint is non-active, and "one" means that the constraint value has reached its limit value. If the buffer coefficient is more than one, the constraint is violated, and the correction part should be applied. We use linear distribution through the buffer zone for the buffer coefficient. Non-linear distribution is non-applicable because the algorithm varies the buffer coefficient in a non-linear way, and it reduces the stability of the method. Based on the size of buffer and its central position, one can calculate the buffer coefficient $\omega_j^{(i)}$ for inequality constraints ($g_j(\boldsymbol{x}_i) \leq 0$):

$$
\begin{aligned}
LBV_j^{(i)} &= CBV_j^{(i)} - BS_j^{(i)} \\
\omega_j^{(i)} &= \frac{g_j(\boldsymbol{x}^{(i)}) - LBV_j^{(i)}}{BS_j^{(i)}}
\end{aligned}
\tag{12}
$$

or for equality constraints ($h_j(\boldsymbol{x}_i) = 0$):

$$
\omega_j^{(i)} = 1 + \frac{abs[g_j(\boldsymbol{x}^{(i)}) - LV_j]}{BS_j^{(i)}} \tag{13}
$$

where $LBV_j^{(i)}$ is a value "lower buffer value," $BS_j^{(i)}$ is a value "buffer size," $CBV_j^{(i)}$ is a value "central buffer value," $g_j(\boldsymbol{x}^{(i)})$ is a constraint value, and LV_j is a limit value. All values are calculated for the jth constraint at the ith iteration.

With (12) and (13), CBV_j and BS_j should first be defined. Initially, $CBV_j^{(i)}$ can be set to be the same as the corresponding constraint limit value. Finding suitable $BS_j^{(i)}$ requires the use of historical information. In the first step, BS_j can be initialized as some small value, for instance $1e-12$ or 1% of the constraint limit value. Starting from the second iteration, we can calculate the maximum change in the constraint value $\Delta g_j^{(i)}$ during the optimization process. By the multiplying maximum change by the buffer size factor BSF, we can estimate the $BS_j^{(i)}$:

$$
\begin{aligned}
BS_j^{(i)} &= BSF \cdot \max_k(\Delta g_j(\boldsymbol{x}^{(k)})) \\
\Delta g_j^{(i)} &= abs(g_j(\boldsymbol{x}^{(i)}) - g_j(\boldsymbol{x}^{(i-1)}))
\end{aligned}
\tag{14}
$$

In general, the buffer factor should be more than one ($BSF > 1.0$) and can be changed during the optimization

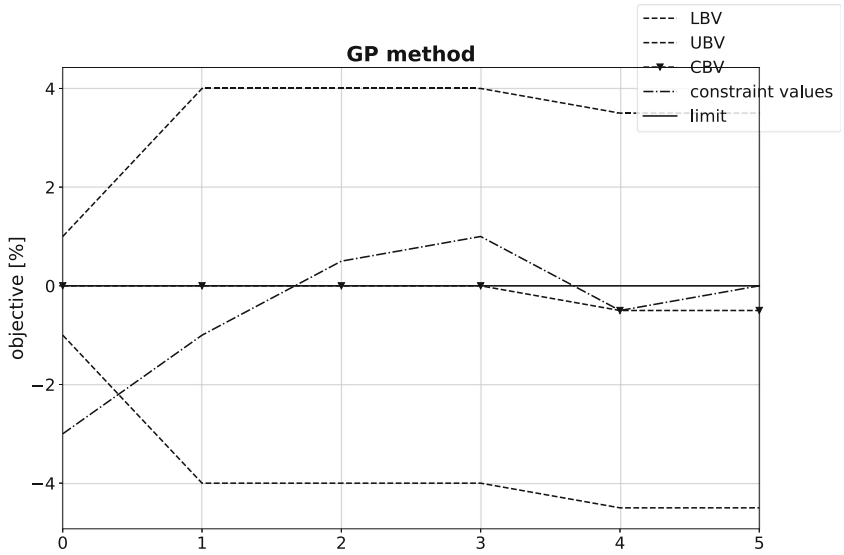


Fig. 2 Buffer zone around constraint limit

process via the buffer adaptation functions. Initially, in our examples, the buffer factor *BSF* is set to 2 because then the algorithm has at least one optimization iteration inside the buffer zone before the constraint value reaches its limit. To sum up, the buffer zone controls the active set of constraints through the constraint's value. In Fig. 2, there is a graph to demonstrate the typical buffer zone around the constraint value.

3.2 Search direction

The RGP algorithm inherits from Rosen's gradient projection algorithm the projection part of the feasible direction, and can rotate it to the direction of the active constraints gradients. The buffer coefficient $\omega_j^{(i)}$ is divided into two components. The first part is relaxation, whereby the constraint is "relaxed" when it is in the feasible domain. The $\omega_j^{r,(i)}$ relaxation coefficient is calculated as follows:

$$\omega_j^{r,(i)} = \begin{cases} \omega_j^{(i)}, & \text{if } \omega_j^{(i)} \leq 1.0 \\ 1, & \text{if } \omega_j^{(i)} > 1.0 \end{cases} \tag{15}$$

If the constraint is equality, the relaxation coefficient is always equal to one, $\omega_j^{r,(i)} = 1.0$. The second component, the $\omega_j^{c,(i)}$ correction coefficient is:

$$\omega_j^{c,(i)} = \begin{cases} BSF_{init}(\omega_j^{(i)} - 1), & \text{if } 1.0 < \omega_j^{(i)} < \omega^{max} \\ 0, & \text{if } \omega_j^{(i)} \leq 1.0 \\ BSF_{init}\omega^{max}, & \text{if } \omega_j^{(i)} \geq \omega^{max} \end{cases} \tag{16}$$

where the factor BSF_{init} is the initial buffer size factor, and ω^{max} is the maximum correction coefficient. If the problem starts from an infeasible domain, the correction coefficient can be very high and may cause numerical issues. The $\omega^{max} = 2$ limits the correction coefficient to the values inside the buffer zone and can work in the most cases. Nonetheless, if the problem starts well inside the infeasible domain, the ω^{max} can be increased to a relatively large value, for instance 10 or 100. If we combine the relaxation and correction components, we can define the search direction:

$$\begin{aligned} \boldsymbol{p}^{(i)} &= -[\boldsymbol{I} - \boldsymbol{N}\boldsymbol{\omega}^{r,(i)}(\boldsymbol{N}^T\boldsymbol{N})^{-1}\boldsymbol{N}^T]\nabla f^{(i)} \\ \hat{\boldsymbol{s}}^{(i)} &= \boldsymbol{p}^{(i)} - \boldsymbol{N}\boldsymbol{\omega}^{c,(i)} \\ \boldsymbol{s}^{(i)} &= \frac{\hat{\boldsymbol{s}}^{(i)}}{||\hat{\boldsymbol{s}}^{(i)}||_{max}} \end{aligned} \tag{17}$$

where $\boldsymbol{\omega}^{r,(i)}$ is an r by r diagonal matrix; $\omega_j^{r,(i)}$ is placed in the main diagonal; $\boldsymbol{\omega}^{c,(i)}$ is a vector with size r, which consist of $\omega_j^{c,(i)}$ buffer coefficients; and $\boldsymbol{p}^{(i)}$ is the relaxed projected direction. All vectors, $\nabla f(\boldsymbol{x}^{(i)})$ and $\nabla g_j(\boldsymbol{x}^{(i)})$ are scaled using max norm ($\nabla f \leftarrow \frac{\nabla f}{||\nabla f||_{max}}, \nabla g_j \leftarrow \frac{\nabla g_j}{||\nabla g_j||_{max}}$). The first equation in (17) calculates the relaxed projected direction $\boldsymbol{p}^{(i)}$, which is similar to the (8). The $\boldsymbol{\omega}^{r,(i)}$ relaxation coefficient can be understood as a factor to control how strongly the steepest direction should be turned into the projected direction. The second equation in (17), the correction equation, is different to the correction move from (11). In contrast to the correction move (11), the correction part rotates the projected direction to point towards the feasible domain, instead of calculating the design update. If the violation of the constraint is higher, the rotation is more significant. If there are several violated constraints and the $\omega_j^{c,(i)}$ correction coefficients are large, the final search direction can have high norm. To avoid it, the third equation normalizes (bounds) the search direction.

Figure 3 shows the possible search directions, calculated using the RGP method. As it is shown, if the constraint value is not inside the buffer zone ($\omega_j^{(i)} = 0$), the search direction

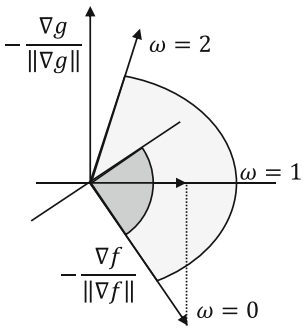


Fig. 3 Possible search direction inside buffer zone, calculated using RGP method

is the same as the steepest descent. If the $\omega_j^{(i)} = 1$, the search direction is the projection of the steepest direction onto the tangential subspace of the active constraint (8). If the $\omega_j^{(i)} > 1$, the search direction is rotated towards the feasible domain. The dark gray sector shows the search directions, which can improve the objective functions.

3.3 Buffer adaptation functions

The performance of the proposed algorithm strongly depends on the buffer size, because the buffer size is used to calculate the feasible search direction. There is a chance that the initial size and central position of the buffer zone may become non-optimal during the optimization process. Therefore, the algorithm requires functions in order to correct the buffer size in two cases: the zigzagging behavior around the constraint limits for the case where constraint violation increases.

In the first case, if the zigzagging for the constraint is detected, the buffer size factor is increased by the following rule:

$$BSF_j^{(i+1)} = BSF_j^{(i)} + abs(\omega_j^{(i)} - \omega_j^{(i-1)}) \cdot \text{factor} \tag{18}$$

where factor is a positive number that scales the update of the buffer size factor. In all numerical examples, factor is set to one. With an increase in the buffer size factor $BSF_j^{(i+1)}$, the buffer size $BS_j^{(i+1)}$ is increased respectively. If we calculate the constraint value $g_j(\boldsymbol{x}^{(i+1)})$ and its respective correction coefficient $\omega_j^{c,(i)}$ using previous and new buffer sizes $BS_j^{(i)}$, $BS_j^{(i+1)}$, it can be seen that the correction coefficient $\omega_j^{c,(i)}$ is smaller with the new buffer size factor than with the previous one. In contrast, the relaxation coefficient $\omega_j^{r,(i)}$ is greater with the new buffer size. Therefore, the value of the constraint will be changed less from iteration to iteration. Alternatively, the buffer size factor can be modified by various rules. For instance, buffer size factor can be doubled, if zigzagging is detected.

The zigzagging behavior can be detected in different ways. One of them is comparing n number of constraint values with the constraint limit value. If the constraint values are sequentially greater and lower than the limit value, the zigzagging around limit is detected. Alternatively, the sign of the result of the multiplication of the difference in constraint values $\Delta g_j^{(i)}$ at n previous iterations can be checked. If we consider 4 previous iterations, the zigzagging criteria is:

$$\begin{cases} \Delta g_j^{(i)} \cdot \Delta g_j^{(i-1)} < 0 \\ \Delta g_j^{(i-1)} \cdot \Delta g_j^{(i-2)} < 0 \\ \Delta g_j^{(i)} = g_j(\boldsymbol{x}^{(i)}) - g_j(\boldsymbol{x}^{(i-1)}) \end{cases} \tag{19}$$

Besides the zigzagging behavior, the constraint value can move away from a limit value in the infeasible direction. The condition to detect this in the case of inequality constraints $g_j \leq 0$ is:

$$\begin{cases} g_j(x^{(i)}) > 0 \\ g_j(x^{(i-1)}) > 0 \\ \Delta g_j^{(i)} \geq 0 \\ \Delta g_j^{(i)} = g_j(\boldsymbol{x}^{(i)}) - g_j(\boldsymbol{x}^{(i-1)}) \end{cases} \tag{20}$$

To bring the design back into feasible domain, the algorithm moves the buffer center in the direction of the feasible domain. That makes the buffer zone non-symmetric around the limit value and increases the $w_j^{(i)}$ buffer coefficient. This function can be understood as moving the real boundaries deeper inside the feasible domain. The new buffer center CBV can thus be found as:

$$CBV_j^{i+1} = CBV_j^{i+1} - (g_j(\boldsymbol{x}^{(i-1)}) - LV_j) \tag{21}$$

With similar schema (20, 21), the buffer center CBV can be restored back to the limit value, in case the constraint correction factor is too strong. In case of the equality constraint, the condition (20) can be extended as follows:

$$\begin{cases} h_j(x^{(i)}) > 0 \\ h_j(x^{(i-1)}) > 0 \\ \Delta h_j^{(i)} \geq 0 \end{cases} \text{ or } \begin{cases} h_j(x^{(i)}) < 0 \\ h_j(x^{(i-1)}) < 0 \\ \Delta h_j^{(i)} \leq 0 \end{cases} \tag{22}$$

Numerical example Figure 4 shows a typical diagram of the smooth application of the active constraint with a constant step size. In the first 3 iterations, the buffer zone was adjusted to refer to the history of the constraint values. In contrast to Rosen's gradient projection, there is no zigzagging behavior along the constraint limit and no jumps in the objective function values.

3.4 Simplified relaxed gradient projection algorithm

In addition to the relaxed gradient projection method from the previous section, there is a simplified version of the proposed algorithm (SRGP). The SRGP method

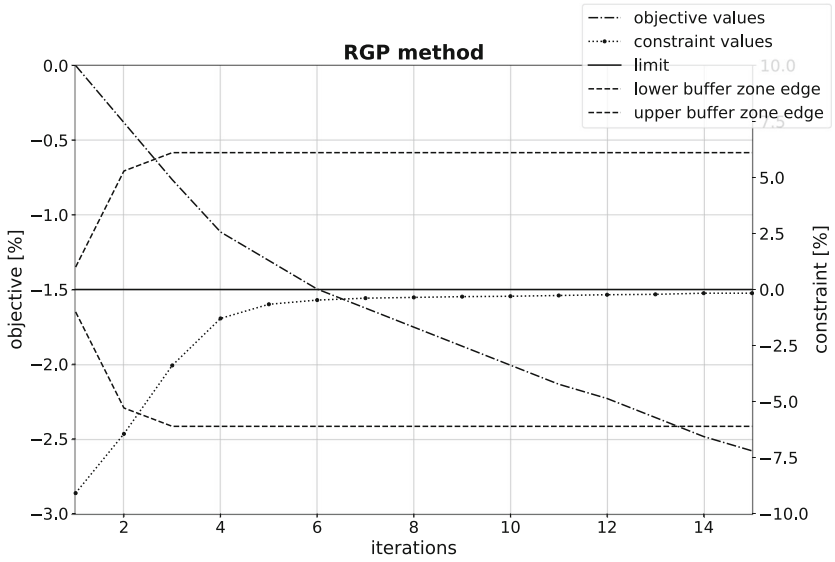


Fig. 4 Smooth adding constraint into the active set using RGP method

does not solve the linear system of equations (7). Instead, it subtracts weighted constraint gradients from negative objective gradient. The weights are calculated in the same way as the buffer coefficient $\omega_j^{(i)}$, (12). In case of a single constraint, the feasible direction is:

$$\boldsymbol{s} = -(1-\omega^{(i)})\nabla f - \omega^{(i)}\nabla g \tag{23}$$

where the $\omega_j^{(i)}$ is the buffer coefficient. In contrast to RGP method, it differs in the range [0; 1] and can be calculated as:

$$\omega^{(i)} = \frac{g(\boldsymbol{x}_i) - LBV}{2 \cdot BS} \tag{24}$$

In the case of multiple constraints, the search direction can be found as follows:

$$\begin{aligned} \boldsymbol{s} &= -(1-\omega^{(i)})\nabla f - \boldsymbol{N}\boldsymbol{\omega}^{(i)}, \\ \omega^{(i)} &= \max_j(\omega_j^{(i)}), \text{ or} \\ \omega^{(i)} &= \sum_j^n(\omega_j^{(i)})/n \end{aligned} \tag{25}$$

where $\boldsymbol{\omega}^{(i)}$ is a vector with size r, which consist of the buffer coefficients $\omega_j^{(i)}$. The weight $\omega^{(i)}$ of the objective gradients can be defined in several ways. It can be the maximum or the average of the $\boldsymbol{\omega}_j^{(i)}$ buffer coefficients.

The simplified relaxed gradient projection method uses the same buffer zone (3.1) and buffer-adaptation functions (Section 3.3) as the RGP method to damp the zigzagging. Unlike the RGP method, SRGP does not calculate the projection direction. Therefore, it cannot follow the linear constraints. In general, the simplified method oscillates constraint and objective values more, and requires more iterations to damp the zigzagging behavior. The advantages of the simplified method are the lack of the need to solve the Lagrangian multipliers (7); that it can work with a large number of constraints; and that it is easy to implement in the optimization framework. All vectors, $\nabla f(\boldsymbol{x}^{(i)})$ and $\nabla g_j(\boldsymbol{x}^{(i)})$, are scaled using max norm. Figure 5 shows the possible directions calculated via the SRGP method (light gray sector). In contrast to the RGP method, the light gray sector is larger and the direction, which is calculated when the constraint value is on the limit ($\omega_j^{(i)} = 0.5$), points towards the feasible domain. Above all, the simplified relaxed gradient projection method can be effectively used in different optimization problems.

Numerical example In Fig. 6, one can see the typical diagram of the smoothing applied on the active constraint with constant step size. In the first 3 iteration, the method adjusts the buffer zone by referring to the history of the constraint values, like in the RGP case. There is a small violation of the constraint while following it. Therefore, the buffer zone slightly changes through iterations. The performance is similar to the RGP case in this simple example.

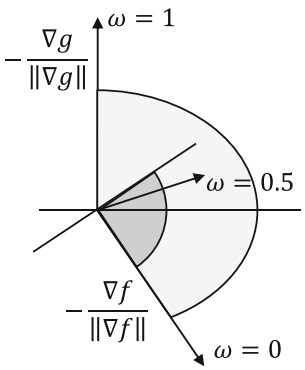


Fig. 5 Possible search direction inside buffer zone, calculated by SRGP method

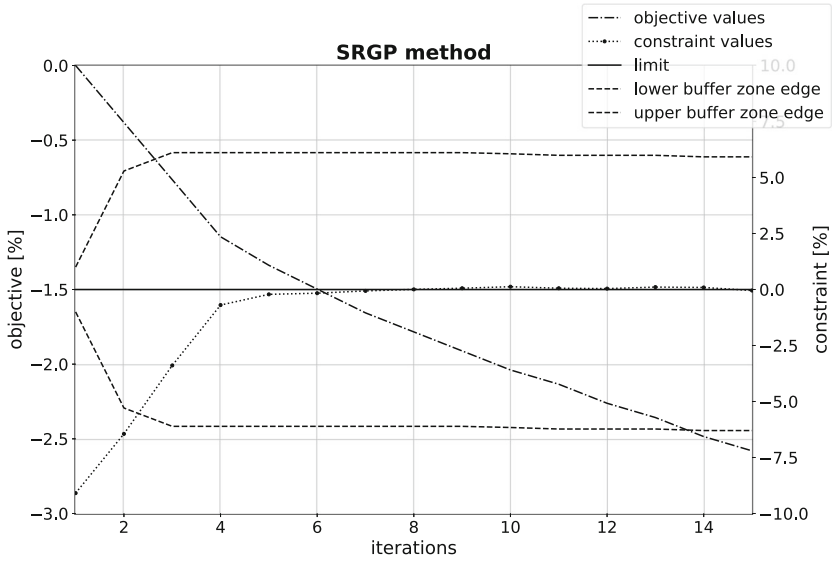


Fig. 6 Smooth adding constraint into the active set with simplified RGP method

3.5 Optimization algorithm

The Algorithm 1 "Relaxed Gradient Projection" is written as the simplified pseudo-code to highlight important steps and to show their order. An interested reader can see more details in the provided python script (see additional materials).

Algorithm 1 Relaxed gradient projection.

$i \leftarrow 0$
// Optimization Loop
for $i = 1, 2, \ldots$ **do**
  Calculate $f(\boldsymbol{x}^{(i)})$, $g_j(\boldsymbol{x}^{(i)})$
  UpdateBufferZones() // (14)
  CalculateBufferCoefficient() // (12, 13)
  ApplyBufferAdaptiveFunctions() //(18, 21)
  ComputeGradients() $\nabla f, \nabla g_j$
  NormalizeSensitivities() $\dfrac{\nabla f}{||\nabla f||_{max}}, \dfrac{\nabla g}{||\nabla g||_{max}}$
  ComputeSearchDirection() $(\boldsymbol{s})^{(i)}$ // (17)
  UpdateDesignParameters() $(\boldsymbol{x})^{(i+1)} \leftarrow (\boldsymbol{x})^{(i)} + \alpha(\boldsymbol{s})^{(i)}$
  CheckConvergence()
  $i \leftarrow i + 1$
end for

4 Numerical examples

To demonstrate the performance of the proposed method, several optimization problems are solved. The main focus in the practical problems is to compare the gradient projection method with its modified "relaxed" version. Results of the simplified relaxed gradient method are shown, but they are not in the main focus of discussions. All methods are implemented in the optimization framework "ShapeModule" (BMW Group). Altair Optistruct^{TM} software is used to solve the structural primal and adjoint analysis of the numerical models.

4.1 Analytical optimization problems

Solution of the test examples for nonlinear programming codes from Hock and Schittkowski (1981) is solved. Description, the reference solutions, and RGP solutions are shown in Tables 1, 2, and 3. All problems are solved with the constant step size. No scaling for the sensitivities is used. The results show that the RGP method is not efficient in solving analytical problems. The main reason for that is a constant step size. After several iterations, the parameter update is extremely low. Using line search techniques can improve the performance of the method. In practical cases, rather than accurately converging to the minima, it is needed to find sufficient improvement. For instance, in the Problem # 43, after 50 iterations, all constraints are satisfied and the objective value is $f(x) = -42.75$.

4.2 Structural optimization problem

Our first solved optimization problem is the typical mass reduction of the model, while the compliance of the model should satisfy the given limit value. The experiment shows the difference in the performance of the gradient projection and relaxed gradient projection when constraints are activated during the optimization process.

Case description In Fig. 10, the geometry of the model can be seen from 2 different sides. It is the fixture for soft-top attachment in the BMW i8 Roadster. The blue color indicates the parts of the model that are damped, which means the optimization algorithm cannot move them.

Table 1 Example 1

Problem	#2
Classification	PBR-T1-2
Number of variables	$n = 2$
Number of constraints	$m = 1$
Objective function	
	$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$
Constrained function	
	$1.5 \leq x_2$
Start	
	$x_0 = (-2, 1)$
	$f(x_0) = 909$
Reference solution	
	$x^* = (2a\ \cos(\frac{1}{3}\ \arccos(1/b)), 1.5)$
	$a = (598/1200)^{0.5}$
	$b = 400a^3$
	$f(x^*) = 0.0504261879$
RGP solution	
	$x^* = (1.22437007, 1.49999902)$
	$f(x^*) = 0.05042600898328847$
	$g(x^*) = 9.765624997548628e - 07$
	$n_{\text{iter}} = 261$
RGP settings	
step size	$5e - 4$

Table 2 Example 2

Problem	#22
Classification	QQR-T1-6
Number of variables	$n = 2$
Number of constraints	$m = 2$
Objective function	
	$f(x) = (x_1 - 2)^2 + (x_2 - 1)^2$
Constrained function	
	$-x_1 - x_2 + 2 \geq 0$
	$-x_1^2 + x_2 \geq 0$
Start	
	$x_0 = (2, 2)$
	$f(x_0) = 1$
Reference solution	
	$x^* = (1.0, 1.0)$
	$f(x^*) = 1.0$
RGP solution	
	$x^* = (0.99999962,\ 1.00000126)$
	$f(x^*) = 1.0000007616095747$
	$g_1(x^*) = -8.769388442075865e - 07$
	$g_2(x^*) = 2.0193504708387877e - 06$
	$n_{\text{iter}} = 102$
RGP settings	
step size	$5e - 2$

The red parts are design variables. There is a transition zone between blue and red parts, where the model can be modified to maintain continuity between damped and design parts. The optimization problem can be formulated as follows:

$$\begin{aligned} & \textbf{minimize} : \text{mass}(\boldsymbol{x}) \\ \text{s.t.} : \ & \text{compliance}_1(\boldsymbol{x}) \leq \text{compliance}_1(\boldsymbol{x}^{(0)}) * 1.1 \\ & \text{compliance}_2(\boldsymbol{x}) \leq \text{compliance}_2(\boldsymbol{x}^{(0)}) * 1.1 \end{aligned} \tag{26}$$

where $\text{mass}(\boldsymbol{x})$ is the mass response function, $\boldsymbol{x}$ is a vector of the design parameters, $\boldsymbol{x}^{(0)}$ is the initial state, and $\text{compliance}_1(\boldsymbol{x})$ and $\text{compliance}_2(\boldsymbol{x})$ are the compliance responses according to the different load cases. The load case 1 is applied in y-direction and load case 2 in z-direction. Both loads are applied on the upper blue zone while the bottom blue part of the FE model is fixed (Fig. 10a). In the case of node-based parametrization, the surface nodes are used as design variables. The size of the $\boldsymbol{x}$ is 145,326. Our stopping criteria is a maximum number of iterations (50). Further optimization steps are not useful, as the mesh quality is not acceptable. The maximum shape update magnitude is constant and equals 0.15 mm.

Optimization method settings Table 4 shows the settings we have used for the methods. The correction coefficient scales the restoring move (11) of the violated constraint. As the shape update is limited, the coefficient helps to balance the impact of the violated constraint with respect to other responses to the final shape update. The parameters are tuned in a way that violated constraints can be corrected in one step. With smaller coefficients, the violations are initially smaller, but after some iterations, method diverges because it can not handle constraints anymore.

Results Figure 7 gives the graphs with the compared objective and constraint values through optimization iterations. The gradient projection algorithm detects the violated constraint $\text{compliance}_2(\boldsymbol{x})$ at iteration 6, adds it to the active set of the constraints, and applies the correction move to bring the solution back into the feasible domain. At iteration, 7, the solution is feasible, and the constraint is removed from the active set of constraints. Hence, the algorithm does not consider it. Therefore, the constraint value is again violated in the next iteration. After this point, marked zigzagging behavior can be seen for objective and constraint values. In contrast, a relaxed gradient projected algorithm has no zigzagging behavior after iteration 6 or 7. If we compare constraint values precisely, in Fig. 7b, c, one can see that up until iteration 4, both algorithms calculate the same shape update because both methods perform the steepest descent step. At iteration 5, the RGP method detects the constraint

Table 3 Example 3

Problem	#43
Classification	QQR-T1-11
Number of variables	$n = 4$
Number of constraints	$m = 3$
Objective function	
	$f(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2$
	$-5x_1 - 5x_2 - 21x_3 + 7x_4$
Constrained function	
	$8 - x_1^2 - x_2^2 - x_3^2 - x_4^2 -$
	$x_1 + x_2 - x_3 + x_4 \geq 0$
	$10 - x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2$
	$+x_1 + x_4 \geq 0$
	$5 - 2x_1^2 - x_2^2 - x_3^2$
	$-2x_1 + x_2 + x_4 \geq 0$
Start	
	$x_0 = (0, 0, 0, 0)$
	$f(x_0) = 0$
Reference solution	
	$x^* = (0.0, 1.0, 2.0, -1.0)$
	$f(x^*) = -44$
RGP solution	
	$x^* = (1.31159684e-07, 1.0,$
	$2.0, -9.99999586e-01)$
	$f(x^*) = -43.999999009762654$
	$g_1(x^*) = 9.520910907445668e-07$
	$g_2(x^*) = 1.0000019930022122$
	$g_3(x^*) = 1.9072899259953147e-08$
	$n_{\text{iter}} = 2766$
RGP settings	
step size	$5e-2$

compliance$_2(\boldsymbol{x})$ as active and adds it to the active set of constraints with the relaxation coefficient. At iteration 9, RGP adds constraint compliance$_1(\boldsymbol{x})$ in the same way. Therefore, the RGP method smoothly and in advance adds constraints to the active set. Still, both constraints were violated during the optimization process, but the amount of the violation is lower than in GP. For instance, after 20 iterations, the constraint compliance$_2(\boldsymbol{x})$ starts to zigzag around the limit, but the RGP method is able to damp the oscillations by using the adaptation function (21) (see Fig. 8). While constraint compliance$_2(\boldsymbol{x})$ is oscillating, the objective function is still improving with nearly the same speed as in the previous iterations. With regard to the results, the objective function is improved faster using the RGP method compared to GP (Fig. 9). During $5 - 50$ optimization iterations, while both optimization algorithms are traveling along the design boundaries, the GP method is able to improve our objective function by 9.5% and RGP by 14.4%. Hence, the RGP method improves the objective function 1.7 times faster than GP in this case. SRGP method sees nearly the same improvement of the objective function as RGP, but the oscillations of the constraint value around the limits are greater. In Fig. 10b, the comparison between initial and optimum design (RGP method) can be seen.

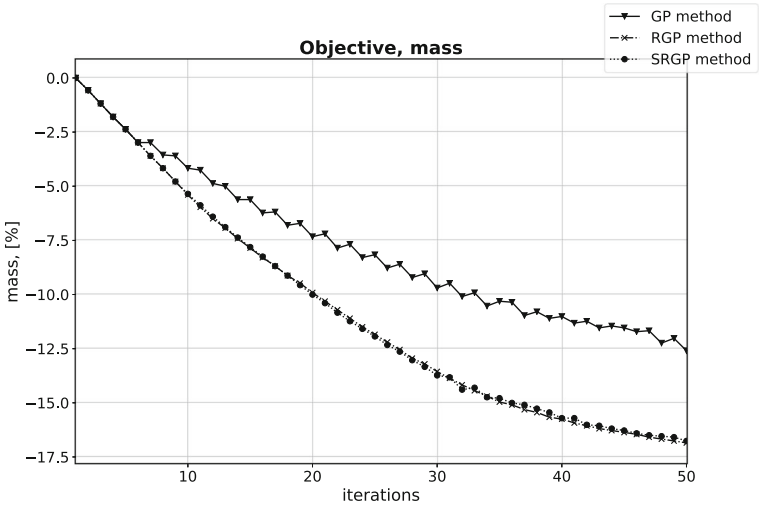
Table 4 Optimitation method settings

Gradient projection	
Step size	0.15
Compliance constraint corr. coeff.	1.0
Compliance constraint corr. coeff.	1.0
RGP and SRGP	
Step size	0.15
Buffer scale factor eq. (18)	1
Initial BSF	2.0

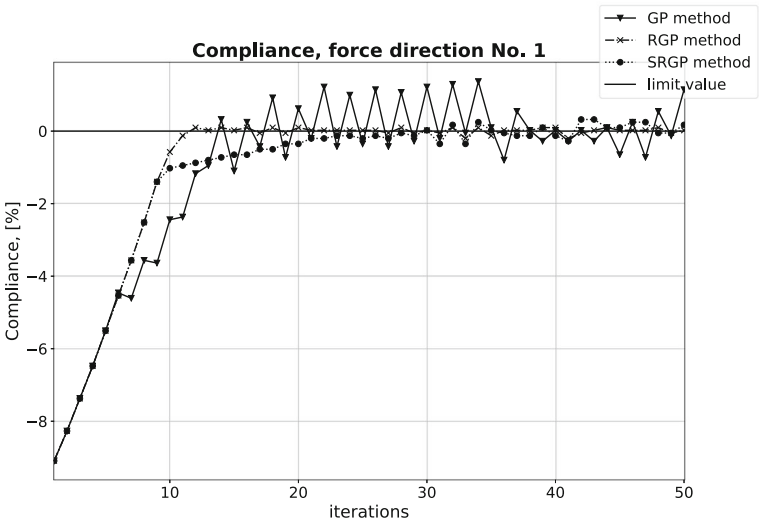
In Fig. 11, there is a comparison in the shape updates between GP and RGP methods. At iteration 1, both methods perform the steepest descent step; therefore, the shape updates are similar in both cases. The difference occurs at iteration 5. The GP method does not detect the constraint compliance$_2(\boldsymbol{x})$ and continues performing steepest descent step. RGP already detects the constraint and calculates "constrained" shape update. At iterations 6–8, the GP method strongly modifies the shape updates: at iteration 6, there is a shape update with a correction move, at iteration 7, there is the steepest descent update, and at iteration 8, there is again a shape update with a correction move. The behavior continues during the whole optimization process. In contrast, the RGP method smoothly modifies the shape updates during iterations 5–8 and beyond. The RGP method tries to find the feasible search direction that will not significantly oscillate the constraints. Therefore, there are fewer oscillations compare to GP.

Figure 12 shows the difference in the final shapes. There is no big difference between SRGP and RGP generated shapes. The shape, generated by the GP method, has less modification of the design due to problems with zig zagging. All methods has similar pattern of the update, therefore all of them converging to similar local optima. The reason for that is that all methods are trying to follow similar optimization path along the active sets of the constraints. If there are no active constraints, all method use steepest descent direction. If the initial design or the parametrization will be changed, new final design can be found.

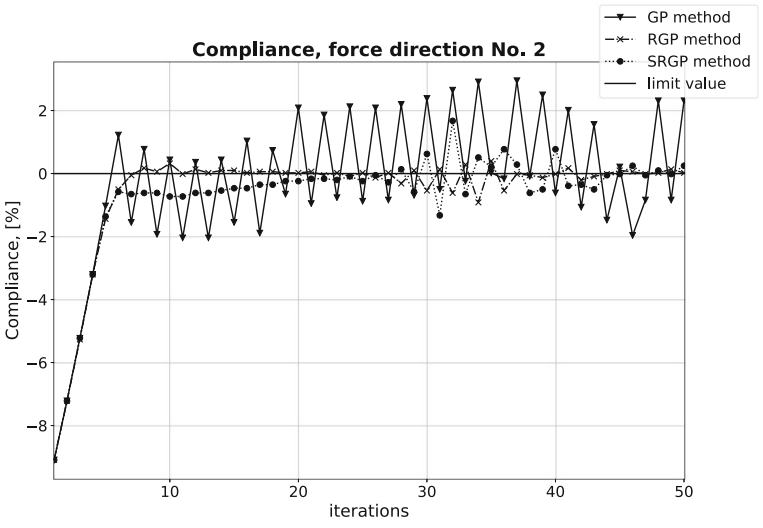
Conclusion The RGP method demonstrates good performance in this case, compared to the GP method. It was able to smoothly activate the constraints as they approached the limit values. If the zigzagging of the constraint is detected,



(a)



(b)



(c)

Fig. 7 GP vs RGP vs SRGP method comparison. **a** Objective values. **b** Constraint compliance1(x) . **c** Constraint compliance2(x)

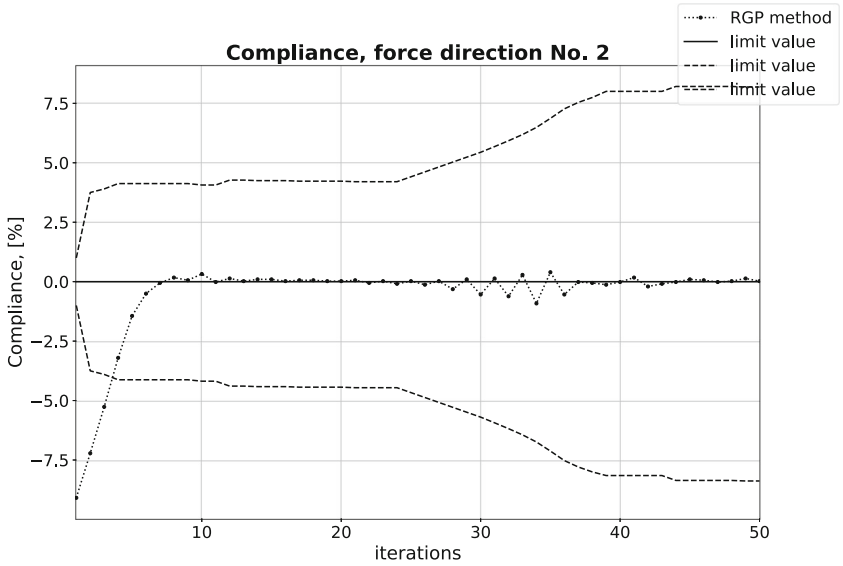


Fig. 8 RGP method, constraint compliance2($\boldsymbol{x}$) with buffer zone

the method is able to damp them, while the objective function keeps nearly the same ratio of improvement. The SRGP method exhibits similar performance to the RGP method, although the constraint violations are greater in number.

4.3 Structural optimization problem with geometrical constraint

In the second numerical example, the structural optimization problem with a geometrical constraint should be solved. In practical cases, it is common to apply geometrical constraints because the designed part exists within a given space so as not to collide with neighboring parts; therefore, the shape boundaries should be fixed. Details of the packaging constraint can be found in Najian Asl et al. (2017).

Case description The objective function is to reduce the mass of the initial model with respect to two constraints: the model should be inside a packaging box (geometrical constraint) and the maximum displacement of any point should be below the given limit. The problem can be defined as follows:

$$\begin{aligned} &\textbf{minimize: } \text{mass}(\boldsymbol{x}) \\ &\quad \textbf{s.t. } d_i \le d^{crit} \\ &\quad\quad x_i \in \boldsymbol{V^c} \end{aligned} \tag{27}$$

where $\boldsymbol{x}$ is a vector of the design parameters, mass($\boldsymbol{x}$) is a mass response, d_i is the displacement at the point i, and $\boldsymbol{V^c}$ is the packaging constraint. The initial configuration can be seen in Fig. 13a, where the red zones are the design nodes and the blue zones are damped and cannot be moved. Figure 13b and c show which parts initially violate the geometrical constraint and should be corrected during the optimization process. The stopping criteria are the maximum number of optimization iterations (100) or the lack of sufficient improvement in the objective function (less than 0.1% in the last 10 iterations). The maximum shape update magnitude is constant and equals to 0.5 mm.

Optimization method settings Table 5 shows the settings we have used for the methods. The correction coefficients were tuned in a way that packaging constraint is less dominant during the optimization process.

Results In Fig. 9, there is a comparison of the objective and constraint values. Initial design is infeasible with respect to packaging constraint; therefore, all methods perform their first iterations to correct the packaging constraint. On graph Fig. 9a, it can be seen that the RGP method improves the objective function much faster than the other two methods. In the case of SRGP, the method performs the "steepest descent steps" in the direction of the violated geometrical constraint ($\boldsymbol{s} = 0 \cdot \nabla f - 1 \cdot \boldsymbol{N}$). The GP method has calculated a correction move; hence, most of the shape update is performed to correct the constraint rather than improve the objective function. In the RGP method, the correction term is limited by the correction coefficient $\omega^{max} = 2$ (see Section 3.2); therefore, the correction is not as high as for the other, and the objective function is improved faster. When the packaging constraint value is corrected, the speed of objective improvement increases.

Figure 14d provides a comparison of the initial and final design of the model. The initial design is in transparent blue, and the optimized design is in white. Visible are the zones where the mass was reduced and the zone where the shape was modified to satisfy the packaging constraint. All methods converged towards a similar optimized design,

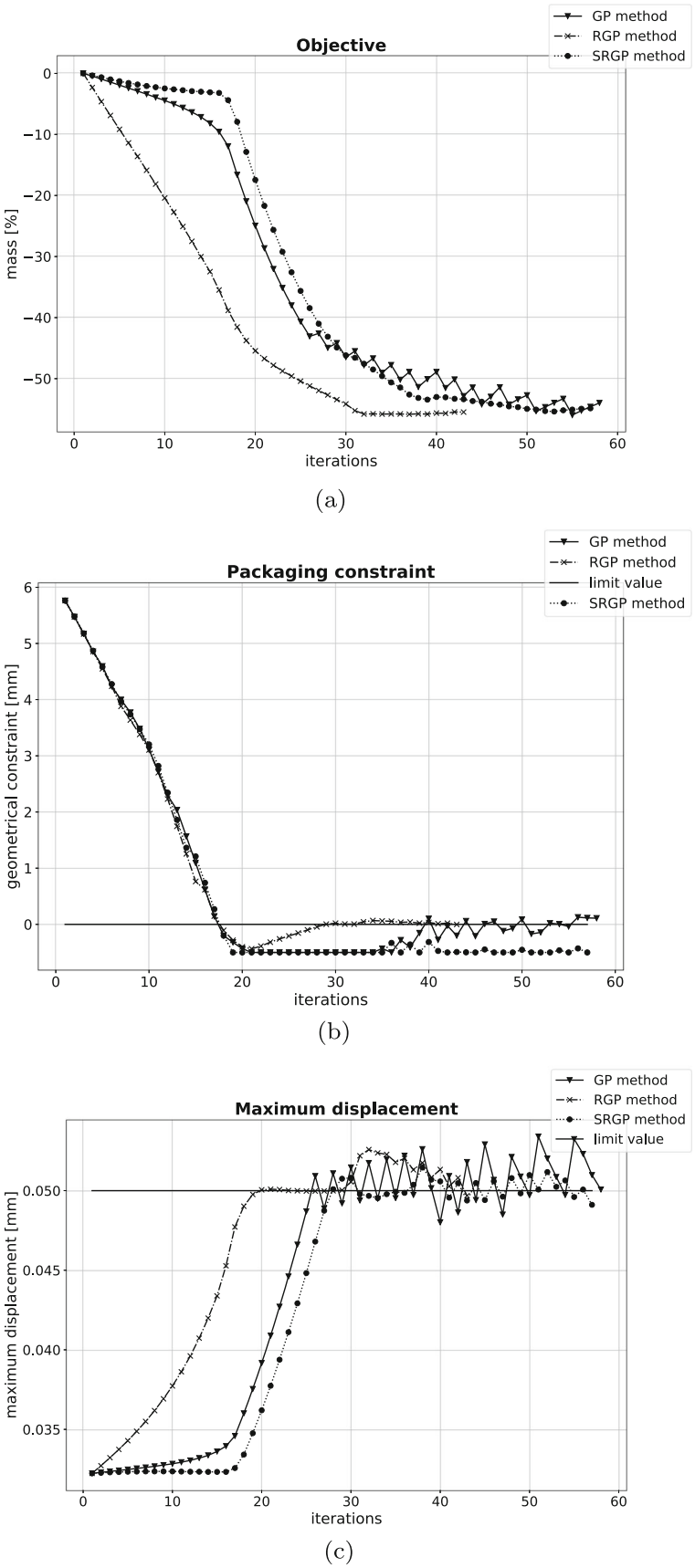
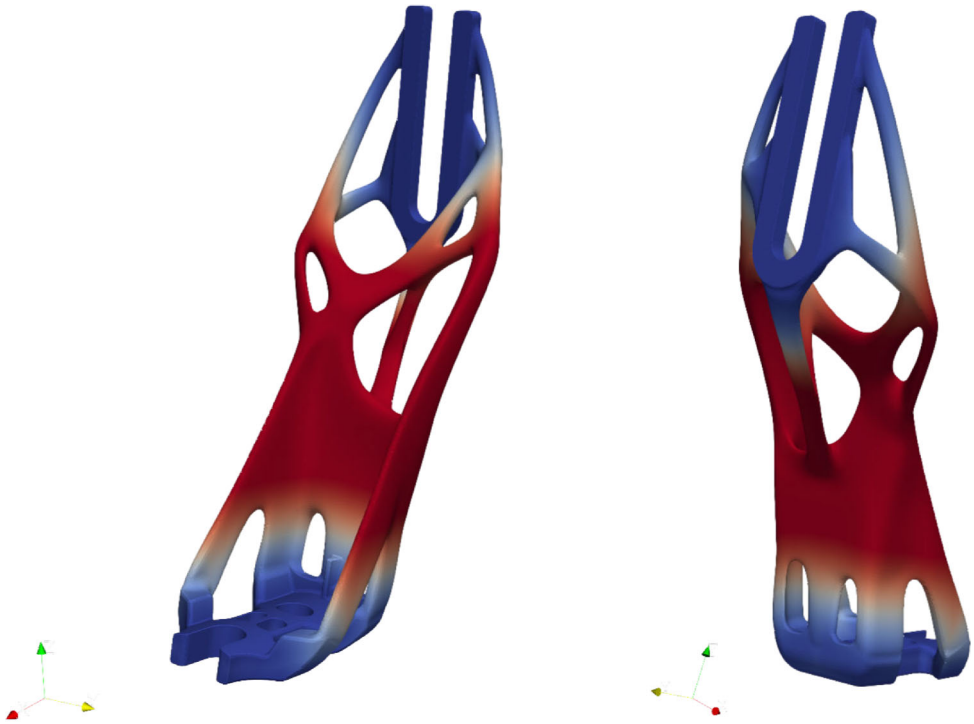
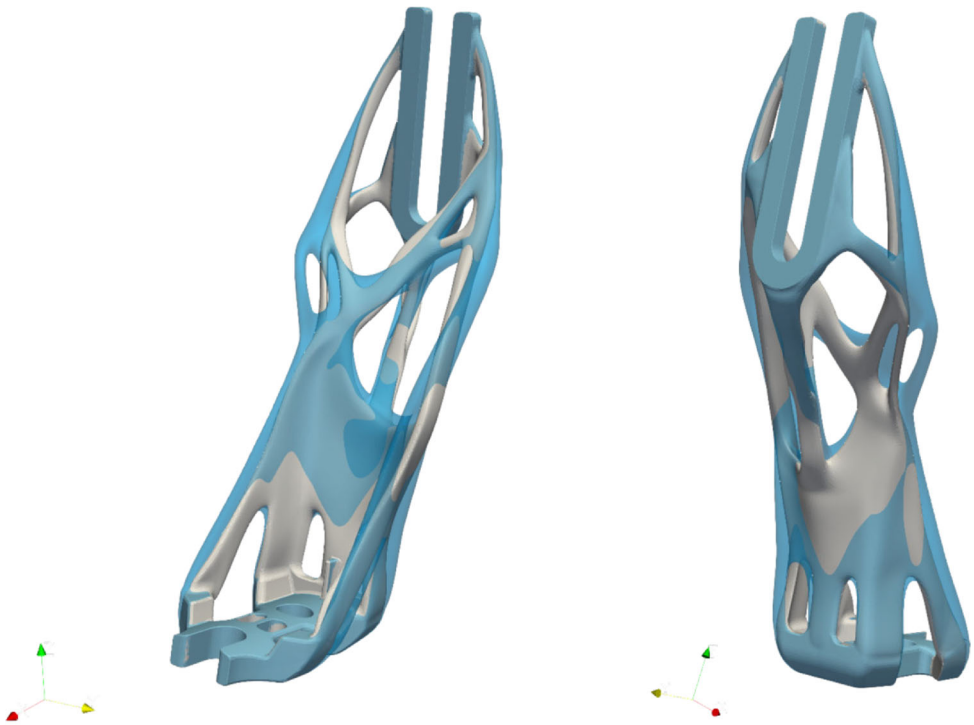


Fig. 9 GP vs RGP vs SRGP methods compared. **a** Objective values. **b** Constraint compliance1(x). **c** Constraint compliance2(x)



(a)



(b)

Fig. 10 Optimization model. **a** Geometry of the optimization model, red design part, blue damped part. **b** Initial design (transparent blue), optimized design (white) via RGP method

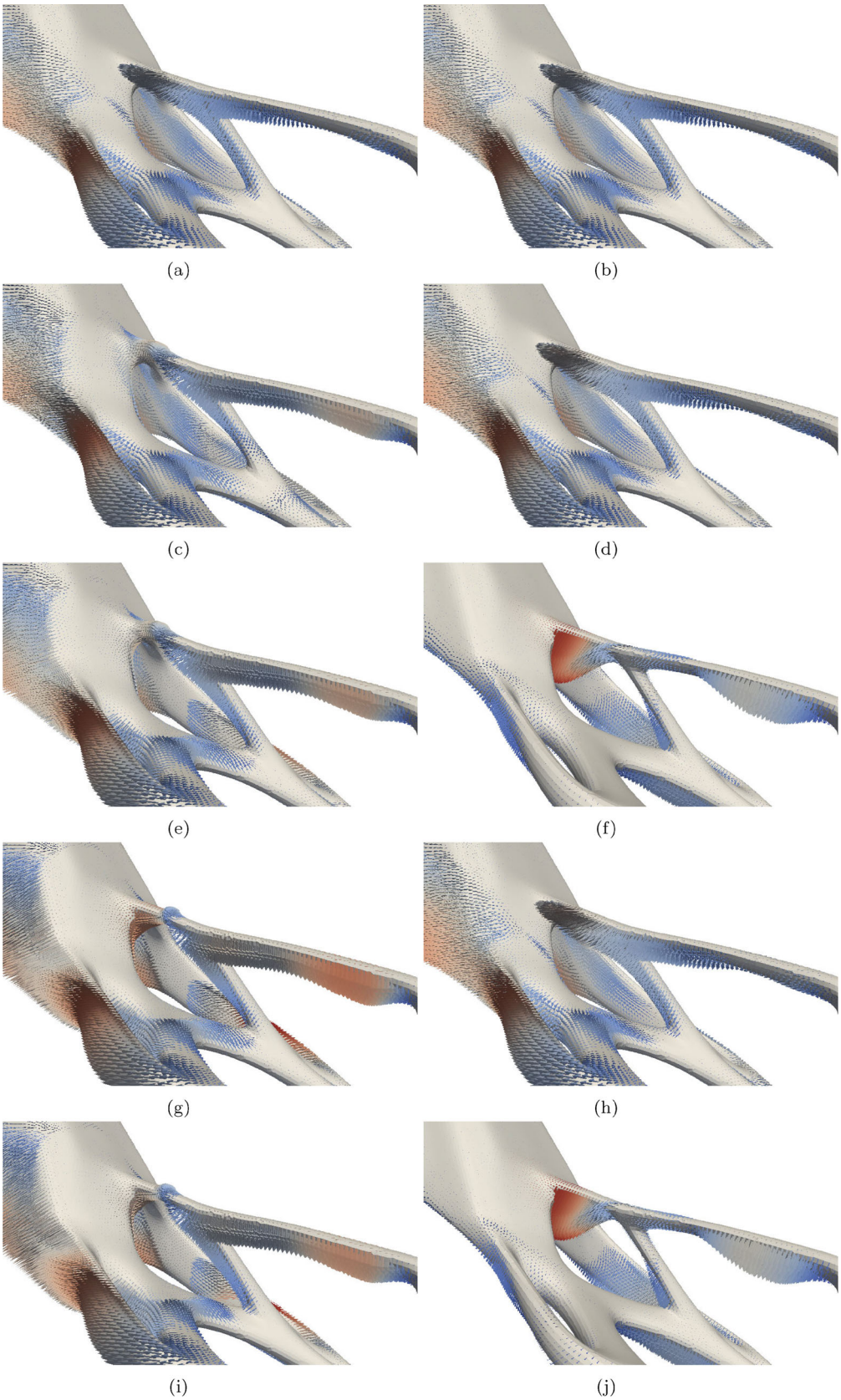


Fig. 11 Comparison of the shape update (scaled), calculated by GP and RGP methods. **a** Iteration 1, RGP. **b** Iteration 1, GP. **c** Iteration 5, RGP. **d** Iteration 5, GP. **e** Iteration 6, RGP. **f** Iteration 6, GP. **g** Iteration 7, RGP. **h** Iteration 7, GP. **i** Iteration 8, RGP. **j** Iteration 8, GP

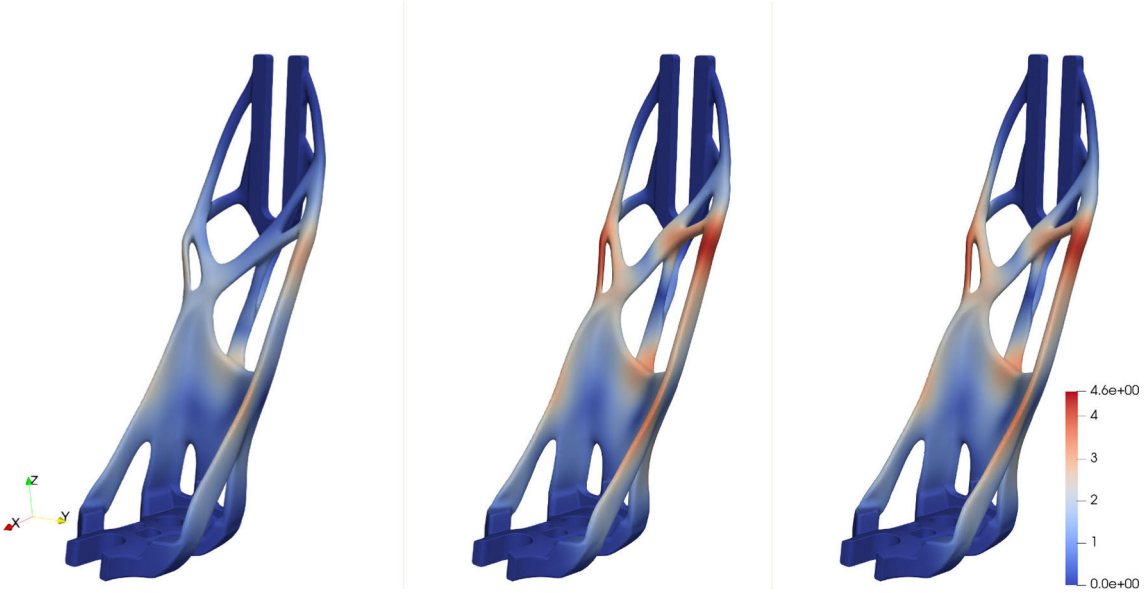


Fig. 12 Comparison of the optimized designs. Data filed shows absolute update in (mm). From left to right: GP, RGP, SRGP

(see Fig. 14a–c). The optimization process was stopped by the convergence criteria because there was no more sufficient improvement in the objective function, and the constraints are satisfied within the given tolerance.

An important difference in performance between the RGP method and others is that in this case the RGP method is much more stable and is able to maintain its geometrical constraint very accurately. There is a violation in the maximum displacement constraint at the 28th iteration, but the method is able to correct the constraint. In case of the SRGP method, there are more oscillations of the constraint values, but the violations are lower. The method was not as efficient as RGP due to the initially slow improvement of the objective. When the geometrical constraint was corrected, the method performed well, and in a stable manner. The GP method displayed similar issues with zigzagging as in the previous examples. Nonetheless, the GP method finds the solution with the same number of iterations as the SRGP method. It is important to note that the computational time for one iteration for each method is similar because there is the same number of calls for analysis and they take up the most time.

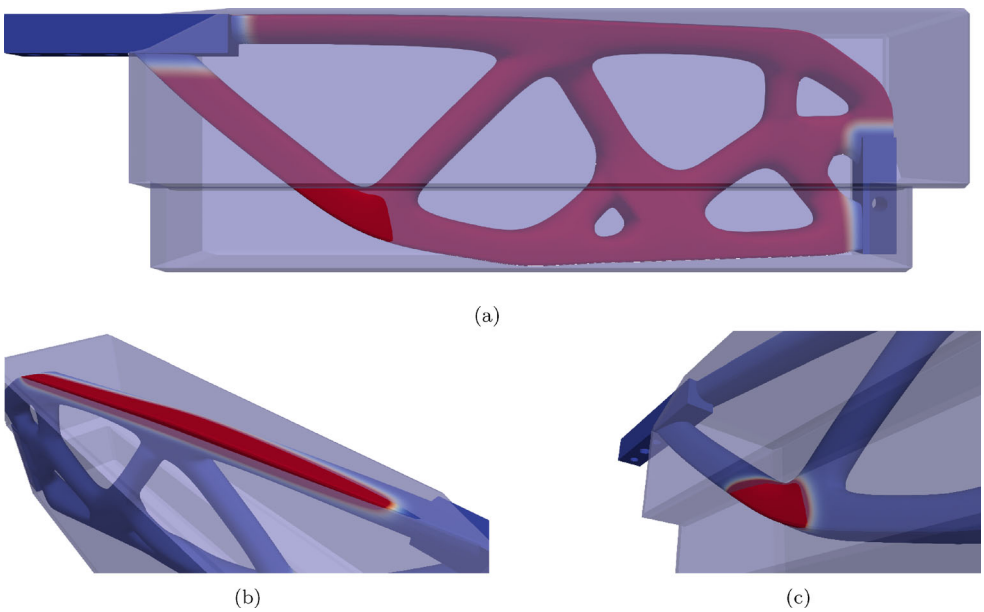


Fig. 13 Optimization model, packaging constraint(light purple). **a** Design model, red design part, blue damped part. **b** Geometrical constraint violation of the initial design, upper side (red). **c** Geometrical constraint violation of the initial design, lower side

Table 5 Optimitation method settings

Gradient projection	
Step size	0.5
Maximum displacement constraint corr. coeff.	1.0
Packaging constraint corr. coeff.	0.05
RGP and SRGP	
Step size	0.5
Buffer scale factor (18)	1
Initial BSF	2.0

4.4 Computational time

Table 6 shows the comparison of the computational time needed to solve the optimization problem from the Section 4.3. Intel Xeon(R) CPU E5-1650 v4 with 6 cores was used in cooperation with 64 GiB RAM. 137 s is needed to solve primal and adjoint solutions and less than 0.7 s to find shape update with constant step size. Rest of the time is needed to compute parametrization, mesh update, and save the output files.

Table 6 Computation time

Method	GP	RGP	SRGP
Aver. time, 1 Iter, s	247	250	248
Full time, s	14326	10750	14136

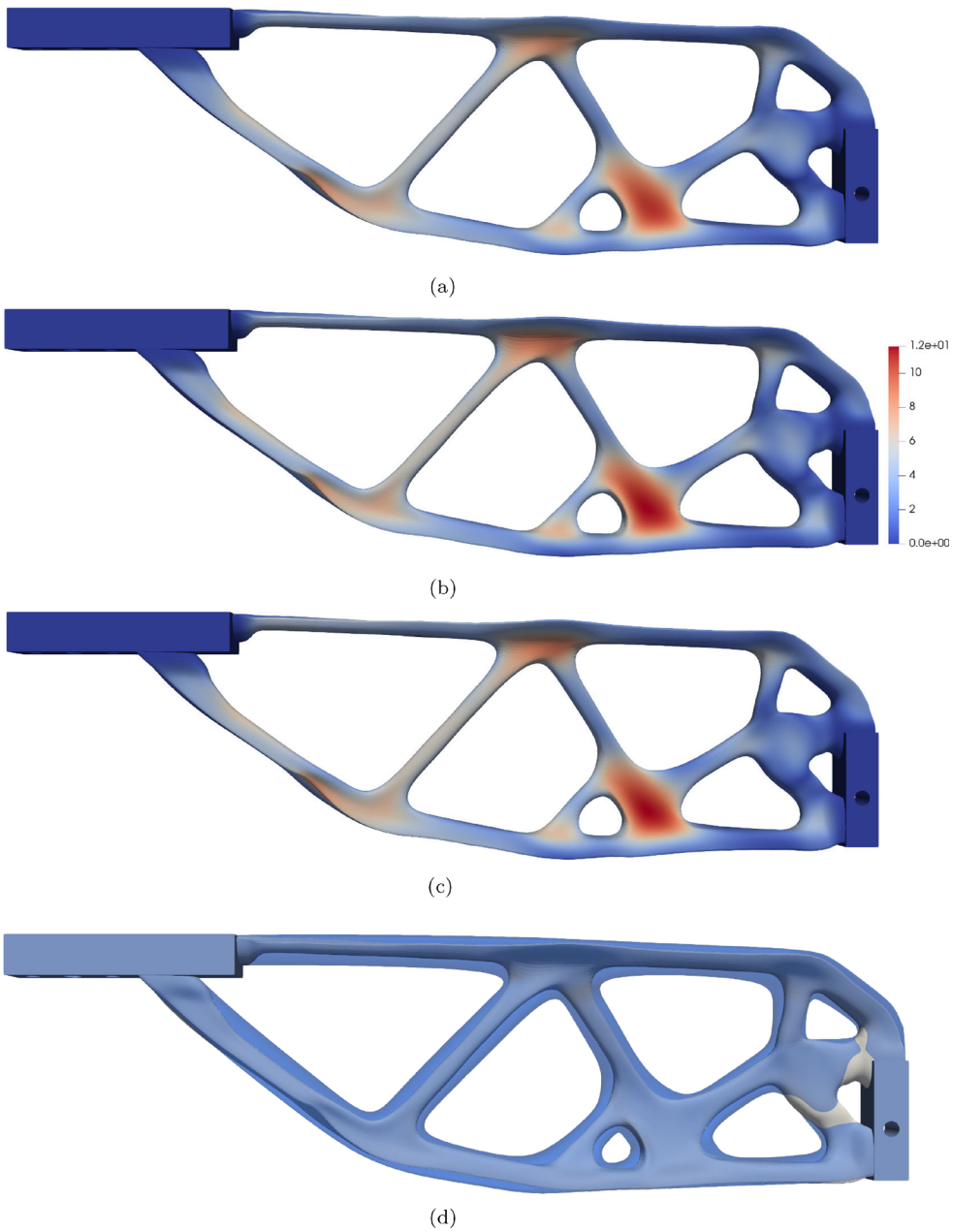


Fig. 14 Comparison of the optimized designs. Data filed shows absolute update in (mm). **a** Gradient Projection method method **b** Relaxed Gradient Projection method **c** Simplied Relaxed Gradient Projection method **d** Initial design (transparent blue), optimized design (white) via RGP method

5 Conclusions

In this paper, the relaxed gradient projection method is introduced. The proposed modifications to Rosen's gradient projections show good speed up in the rate of the objective improvement, and in avoiding marked zigzagging behavior. The proposed method can activate the constraint in advance before the limit value is reached, and it has techniques to reduce the zigzagging behavior while following the constraint boundaries. It does not require accurate parameter set up; therefore, it is easier and stable in daily practice. Further research can be done to find efficient line-search techniques that can be efficiently used in the practical applications, and computing more accurately the correction part of the search direction. In conclusion, we see the relaxed gradient projection algorithm as being one of the group of feasible direction methods. We do not claim that the proposed method is the best option for shape optimization problems in general. The proposed algorithm should be considered as a good alternative to other successful optimization methods, such as inner-point (Chen et al. 2019) or trust-region algorithms (Yuan 1999).

Supplementary Information The online version contains supplementary material available at 10.1007/s00158-020-02821-y.

Acknowledgments Open Access funding enabled and organized by Projekt DEAL. The authors wish to thank the ShapeModule project (BMW Group) for the support. Thanks are also due to S. Stahl M.Sc. (Design for Additive Manufacturing, BMW Group) for providing numerical models.

Funding This paper is based on a part of the research sponsored by the BMW group.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

Replication of results The proposed method is implemented in the optimization framework "ShapeModule" (BMW Group, shapemodule@bmw.de). The code has no public access, as well as shown models. Following suggestions from Haftka et al. (2019), to improve the replication of results, the proposed method is implemented in the python script (see additional materials). The script has all significant steps of the relaxed gradient projection method, and it solves a constrained quadratic problem. In the script, all methods have references to the corresponding equations. Additionally, the constant parameter update and scaling of the sensitivities are applied. An interested reader can contact the corresponding author for the respective explanations.

References

Agarwal D, Robinson T, Armstrong C, Kapellos C (2018) Enhancing cad-based shape optimisation by automatically updating the cad model's parameterisation. Struct Multidiscip Optim 59:11. https://doi.org/10.1007/s00158-018-2152-7

Baumgärtner D, Viti A, Dumont A, Carrier G, Bletzinger K-U (2016) Comparison and combination of experience-based parametrization with vertex morphing in aerodynamic shape optimization of a forward-swept wing aircraft 06. https://doi.org/10.2514/6.2016-3368

Bletzinger K-U (2017) Shape optimization, pp 1–42. ISBN 9781119176817. https://doi.org/10.1002/9781119176817.ecm2109

Chen L, Bletzinger K-U, Geiser A, Wüchner R (2019) A modified search direction method for inequality constrained optimization problems using the singular-value decomposition of normalized response gradients. Struct Multidiscip Optim 06:1–19. https://doi.org/10.1007/s00158-019-02320-9

Du D, Wu F, Zhang X (1990) On rosen's gradient projection methods. Ann Oper Res 24:9–28, 12. https://doi.org/10.1007/BF02216813

Firl M, Bletzinger K-U (2012) Shape optimization of thin walled structures governed by geometrically nonlinear mechanics. Comput Methods Appl Mech Eng 237–240:107–117, 09. https://doi.org/10.1016/j.cma.2012.05.016

Fletcher R (2013) General linearly constrained optimization, chap 11. Wiley, New York, pp 259–276. https://doi.org/10.1002/9781118723203.ch11. ISBN 9781118723203

Gallagher R, Zienkiewicz O (1977) Optimum structural design—theory and applications 01

Haftka RT, Grandhi RV (1986) Structural shape optimization—a survey. Comput Methods Appl Mech Eng 57(1):91–106. https://doi.org/10.1016/0045-7825(86)90072-1. ISSN 0045-7825

Haftka R, Kamat M (1990) Element of structural optimisation, vol 1 01. https://doi.org/10.1007/978-94-015-7862-2

Haftka RT, Zhou M, Queipo NV (2019) Replication of results. Struct Multidiscip Optim 60(2):405–409. https://doi.org/10.1007/s00158-019-02298-4. ISSN 1615-1488

Hardee E, Chang K-H, Tu J, Choi K, Grindeanu I, Yu X (1999) A cad-based design parameterization for shape optimization of elastic solids. Adv Eng Softw 30:185–199, 03. https://doi.org/10.1016/S0965-9978(98)00065-9

Heners J, Radtke L, Hinze M, Düster A (2017) Adjoint shape optimization for fluid-structure interaction of ducted flows. Comput Mech 61:259–276, 08. https://doi.org/10.1007/s00466-017-1465-5

Hock W, Schittkowski K (1981) Test examples for nonlinear programming codes, vol 187. Springer, Berlin. https://doi.org/10.1007/978-3-642-48320-2

Hojjat M, Stavropoulou E, Gallinger T, Israel U, Wüchner R, Bletzinger K-U (2010) Fluid-structure interaction in the context of shape optimization and computational wind engineering. 73: 351–381, 09, Fluid Structure Interaction II, Springer, Berlin Heidelberg

Hojjat M, Stavropoulou E, Bletzinger K-U (2014) The vertex morphing method for node-based shape optimization. Comput Methods Appl Mech Eng 268:494–513, 01. https://doi.org/10.1016/j.cma.2013.10.015

Kenway G, Kennedy G, Martins J (2014) Aerostructural optimization of the common research model configuration, 06. ISBN 978-1-62410-283-7. https://doi.org/10.2514/6.2014-3274

Kroll N, Gauger N, Brezillon J, Dwight R, Vollmer D, Becker K, Barnewitz H, Schulz V, Hazra S (2007) Flow simulation and shape optimization for aircraft design. J Comput Appl Math 203:397–411, 06. https://doi.org/10.1016/j.cam.2006.04.012

Luo Z, Wang M, Wang S, Wei P (2008) A level set-based parameterization method for structural shape and topology optimization. Int J Numer Methods Eng 76:1–26, 10. https://doi.org/10.1002/nme.2092

Najian Asl R, Shayegan S, Geiser A, Hojjat M, Bletzinger K-U (2017) A consistent formulation for imposing packaging constraints in shape optimization using vertex morphing parametrization. Struct Multidiscip Optim 56:1–13, 10. https://doi.org/10.1007/s00158-017-1819-9

Palacios F, Economon T, Alonso J (2012) Optimal shape design for open rotor blades 06. https://doi.org/10.2514/6.2012-3018

Rosen J (1960) The gradient projection method for nonlinear programming. Part i. Linear constraints. J Soc Ind Appl Math 8:03. https://doi.org/10.1137/0108011

Rosen J (1961) The gradient projection method for nonlinear programming: part ii. SIAM J Appl Math - SIAMAM 9:01

Sieger D, Menzel S, Botsch M (2012) A comprehensive comparison of shape deformation methods in evolutionary design optimization

Sun W, Yuan Y-X (2006) Optimization theory and methods. Nonlinear Program 1:01. https://doi.org/10.1007/b106451

Ummidivarapu V, Voruganti H (2017) Shape optimisation of two-dimensional structures using isogeometric analysis. Int J Eng Syst Model Simul 9:169, 01. https://doi.org/10.1504/IJESMS.2017.085080

Ummidivarapu V, Voruganti H, Khajah T, Bordas S (2020) Isogeometric shape optimization of an acoustic horn using the teaching-learning-based optimization (tlbo) algorithm. Comput Aided Geom Des 80:101881, 05. https://doi.org/10.1016/j.cagd.2020.101881

Vanderplaats G (2007) Multidiscipline design optimization. Vanderplaats Research & Development Inc

Wang M, Luo Z (2020) Shape and topology optimization for compliant mechanisms using level set-based parameterization method, pp 18–21, 05

Xu S, Jahn W, Mueller J-D (2014) Cad-based shape optimisation with cfd using a discrete adjoint. Int J Numer Methods Fluids 74:01. https://doi.org/10.1002/fld.3844

Yuan Y-X (1999) A review of trust region algorithms for optimization. In: ICM99: proceedings of the fourth international congress on industrial and applied mathematics 09