THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Numerical Folding of Airbags Based on Optimization and Origami

Christoffer Cromvik

CHALMERS | GÖTEBORG UNIVERSITY

Department of Mathematical Sciences
Chalmers University of Technology and Göteborg University
Göteborg, Sweden 2007

Numerical Folding of Airbags Based on Optimization and Origami
Christoffer Cromvik

Licentiate Thesis
ISSN 1652-9715
NO 2007:12
Department of Mathematical Sciences
Division of Mathematics
Chalmers University of Technology and Göteborg University
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31 772 1000

Printed in Göteborg, Sweden 2007

Numerical Folding of Airbags Based on Optimization and Origami

Christoffer Cromvik

Department of Mathematical Sciences
Chalmers University of Technology
Göteborg University

Abstract

We present an algorithm for folding three-dimensional airbags. The algorithm is based on nonlinear optimization and Origami mathematics.

The airbag is folded to fit into its compartment. A numerical simulation of the inflation requires an accurate geometric representation of the folded airbag. However, the geometry is often specified in the inflated three-dimensional form, and a flat folded geometry must be computed.

Our algorithm starts by approximating the geometry of the inflated airbag by a quasi-cylindrical polyhedron. Origami mathematics is used to compute a crease pattern for folding the polyhedron flat. The crease pattern is computed with the intention of being fairly simple and to resemble the actual creases on the real airbag.

The computation of the crease pattern is followed by a computation of the folding. This is based on solving an optimization problem in which the optimum is a flat folded model. We use a Sequential Quadratic Programming method which is designed for large-scale problems.

Finally, the flat airbag is further folded or rolled into its final shape (without using Origami).

We test the algorithm on a passenger airbag.

Acknowledgments

First of all, I wish to thank my supervisors Kenneth Eriksson, Bengt Pipkorn and Stig Larsson, for their support and patience over the years. Without them, I would probably not have been given the opportunity to be a Ph. D. student in Mathematics.

Autoliv Research has financially supported this work, and I am very grateful for this. At Autoliv Research, I wish to thank Krystoffer Mroz and Magnus Eriksson for helping me with airbag folding and other engineering skills.

In the Department of Mathematical Sciences, I wish to thank Thomas Ericsson, who has helped me a lot with numerical linear algebra and computer implementations. I also wish to thank my colleagues and friends for creating a pleasant working environment. In particular, I wish to thank Niklas Eriksen, David Heintz, Karin Kraft, Peter Lindroth, Ali Mesforush, Anna Nyström, Michael Patriksson, and Fardin Saedpanah.

Finally, I wish to thank my fiancée Julia.

This licentiate thesis consists of the following papers:

Paper I: *Numerical folding of airbags based on optimization and Origami*

Paper II: *A low-storage sequential quadratic programming method*

Paper III: *Airbag folding based on Origami mathematics (together with K. Eriksson)*

NUMERICAL FOLDING OF AIRBAGS BASED ON OPTIMIZATION AND ORIGAMI

CHRISTOFFER CROMVIK

Abstract. We present an algorithm for folding three-dimensional airbags. The algorithm is based on nonlinear optimization and Origami mathematics.

The airbag is folded to fit into its compartment. A numerical simulation of the inflation requires an accurate geometric representation of the folded airbag. However, the geometry is often specified in the inflated three-dimensional form, and a flat folded geometry must be computed.

Our algorithm starts by approximating the geometry of the inflated airbag by a quasi-cylindrical polyhedron. Origami mathematics is used to compute a crease pattern for folding the polyhedron flat. The crease pattern is computed with the intention of being fairly simple and to resemble the actual creases on the real airbag.

The computation of the crease pattern is followed by a computation of the folding. This is based on solving an optimization problem in which the optimum is a flat folded model. Finally, the flat airbag is further folded or rolled into its final shape (without using Origami).

We test the algorithm on a passenger airbag.

1. Introduction

One of the earliest references to airbags in automobiles was the patent by John W. Hetrick in 1953. The idea of using airbags was not entirely new, since it had been used in some airplanes before. The first airbags were made publically available in some selected vehicle models in the 1970s. Airbags were initially designed to be substitutes for seat belts, but in the 1980s car designers abandoned that idea and considered the airbag as a supplementary restraint system.

The airbag system mainly consists of three parts: the airbag itself, the inflator unit and the crash sensor or diagnostic unit. The crash sensor activates the inflation of the airbag depending on a set of parameters which indicates a crash. One parameter is the deceleration of the vehicle. The inflator has a pyrotechnic charge consisting of a mixture of chemical compounds. An igniter starts a reaction which generates an expansion of gas consisting of mostly nitrogen. The airbag is made of nylon, which has a light weight and is very resistant to stretch.

Computer simulations of crash tests are considered a standard tool for evaluating the safety of a car. Simulations can be done early in the design phase for a new car, and it is relatively inexpensive compared to real crash tests. The computer models are often very complex and requires massive amount of computational time. Both the behavior of the outer structure and the interior of a car need to be examined to evaluate the crashworthiness. The interior of a car includes crash test dummies and the restraint systems.

Simulating a crash where the crash test dummy hits an expanding airbag is a challenge to the industry. This situation is called out-of-position (OOP), reflecting that the airbag is not designed for occupants that are sitting too close or for some other reason hit the airbag before it is fully inflated.

The difficulty with an OOP situation compared to an in-position situation is that the inflation of the folded airbag is much more important. It has to be realistically simulated, since it affects the impact of the dummy. Attaining a realistic simulation means starting with a correct geometry

of the folded airbag and simulating the inflation with correct gas dynamics. Several commercial software packages exist that can simulate the inflation process of an airbag, e.g., the explicit Finite Element (FE) code `LS-DYNA` [11]. However, these require an accurate description of the initial geometry.

Different airbags are folded by different methods and with different numbers and types of foldings. The airbags are often folded by both machines and humans according to a folding scheme. Still, the creases are not entirely deterministically positioned. It is very difficult to control the placement of smaller creases. The folding schemes all assume that the airbag initially lies flat and stretched in some direction. In this position, different foldings are executed until the dimension of the folded airbag is small enough so that it fits in the airbag compartment. The foldings can be a combination of simple folds, but also roll folds. This work aims at developing an algorithm for computing an accurate geometry of the flat folded airbag. Some preprocessors to `LS-DYNA`, e.g., `EASi-FOLDER` [6] and `OASYS-PRIMER` [3] contain software for folding a (nearly) flat FE airbag mesh. They are capable of executing the type of foldings that are normally used in production on flat airbags, e.g., roll-fold, z-fold. However, they are not accurate when folding an airbag from its three-dimensional shape to a flat airbag.

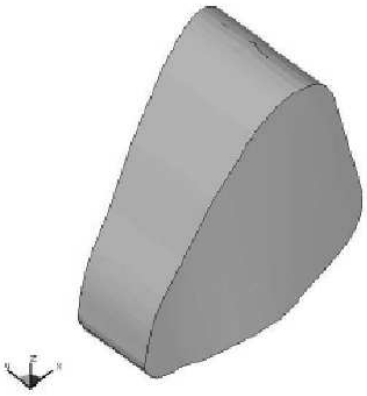

FIGURE 1. A CAD model of a passenger airbag.

Some airbag models have a simple geometry, e.g., the driver airbag which is made of two flat circular layers sewn together at the boundary. It is essentially two-dimensional. Passenger airbags are often more complicated. They are made of several layers sewn together to form a three-dimensional shape, with no trivial flat two-dimensional representation. See Figure 1 for an example of the geometry of a passenger airbag.

In this work we compute the geometry of the flat folded airbag in two steps. First a crease pattern is computed on a polyhedral approximation of the airbag. Then, a nonlinear optimization problem is formed and solved. The optimal solution gives the flat geometry. The accuracy of the computed approximation is measured by comparing its area to the area of the inflated model.

2. Crease Pattern

There is a strong connection between paper folding and airbag folding. Both topics deal with the folding of thin materials. For paper folding, through a series of foldings, a three-dimensional object is created from a flat sheet of paper. In the case of airbag folding, the three dimensional

shape is given, and we want to compute a crease pattern for flattening. For this task we use the mathematical theory from Origami.

Origami is the art of paper folding and has its origin in Japan. The word Origami comes from the two words, *oru* which means "to fold" and *kami* which means paper [2]. Although Origami is an ancient art, the interest from a mathematical point of view has increased during the last century. Specifically, the area of Computational Origami began. Computational Origami provides a connection between theory and algorithms. One of the pioneers in this field was D. A. Huffman.

An example of Computational Origami is the work of R. J. Lang. He is the author of the computer program TreeMaker [10] which lets you compute the crease pattern on a sheet of paper to fold very complex geometries.

Origami has also found its way to industrial applications. Just to mention a few, Lang discovered the application of Origami to airbag folding, in cooperation with the company EASi [8]. You and Kuribayashi considered the application of Origami for a stent used in medicine [19].

Origami is usually connected to folding a square piece of paper into a three dimensional object. For airbag folding (or polyhedron folding), the problem may seem reversed. Given a three dimensional shape, find the flat shape. However, the theory of Origami can be applied, and the key is to apply the theory to each face of the polyhedron.

From now on we assume that the airbag is a polyhedron. We begin by considering a special shape which we call quasi-cylinder. See Figure 2 for an example of a quasi-cylinder.

Definition 2.1. *A quasi-cylindrical polyhedron is a closed cut-off cylinder with a polygonal cross-section. By gables we refer to the two cross-sections, and the mantle is the surface joining the two gables.*

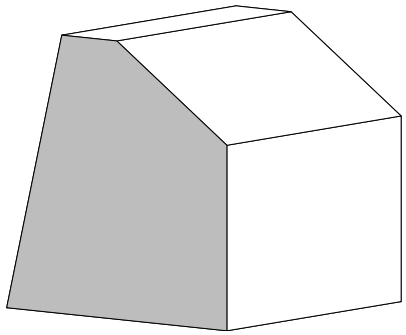

FIGURE 2. A quasi-cylinder with a pentagonal gable in gray color.

2.1. **Origami Molecules.** In an Origami design, certain crease patterns keep reappearing in different parts of the design. They consist of a polygon with a characteristic crease pattern. If the polygon is a triangle, the crease pattern is always the same, but for polygons of higher order there are a number of different crease patterns. The fact that these polygons reappear several times in the design, makes us think of an Origami design as made up of such polygons. This

is why they are called "Origami Molecules". A set of polygons, each with an individual crease pattern, form a global crease pattern which is the Origami design.

The molecules have certain properties. When folded along the crease pattern, the projection of its folded form onto a plane forms a stick figure, also called a tree. All the edges of the polygon fall on a single line after the folding. The latter property is what enables a construction of the origami design by joining molecules. Also, two molecules sharing an edge, must only have creases that cross the edge perpendicularly.

The key to understanding how Origami molecules can be used on polyhedra comes from the fact that if a polygonal side of a polyhedron is divided by non-intersecting diagonals, each sub-polygon can form a molecule, and hence can be treated almost individually. When each molecule is folded, all the edges fall on a straight line, and since each neighboring molecule has a common edge, all edges of the molecules fall on a line.

The problem that remains is how to join the crease patterns of the molecules. This is easy if the polyhedron is a quasi-cylinderical polyhedron with parallel gables. In that case the molecules are formed identically on both gables and the crease patterns are joined by creases along the mantle. This enables a collapse of the gables. The use of diagonals can be interpreted as slicing the polyhedron.

The simplest molecule is the crease pattern for a triangle, called the "Rabbit Ear Molecule", see Figure 3. The crease pattern consists of three creases along the bisectors of the triangle, forming the ridge in the figure, and an additional set of creases from the point of intersection to the edges of the triangle. Each additional crease crosses the edge perpendicularly. In the figure, three additional creases are present, but only one is needed for the molecule to be flattened.

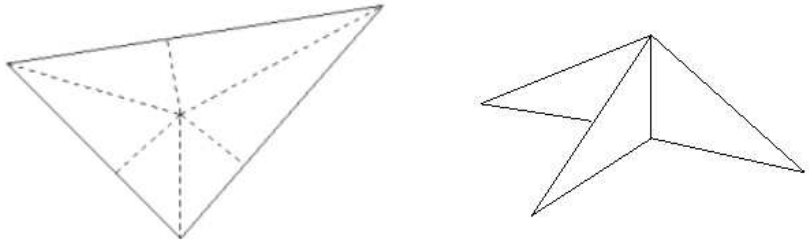

FIGURE 3. The Rabbit Ear Molecule. The figure to the left shows the unfolded molecule. Dashed lines are creases and the solid lines mark the edges.

To create a crease pattern for a general polygon, there are at least two algorithms: the Straight Skeleton [1] and the Universal Molecule [9]. For a convex polygon, the Straight Skeleton is equal to the medial axis [13] which is constructed by using the angular bisectors.

Here we present a new algorithm, the Skew Skeleton, for computing a crease pattern of a quasi-cylinder. It is based on the Straight Skeleton.

If the gables of the quasi-cylinder are parallel, and the mantle is perpendicular to the gables, then the Straight Skeleton can be applied almost directly to form a crease pattern. However, if the quasi-cylinder is skew, then the crease pattern is no longer derived from bisectors as in the Straight Skeleton.

In [5] the crease patterns for some polyhedra are described.

2.2. **Algorithm for Quasi-Cylinders.** This subsection describes the algorithm for computing the crease pattern for a quasi-cylindrical polyhedron.

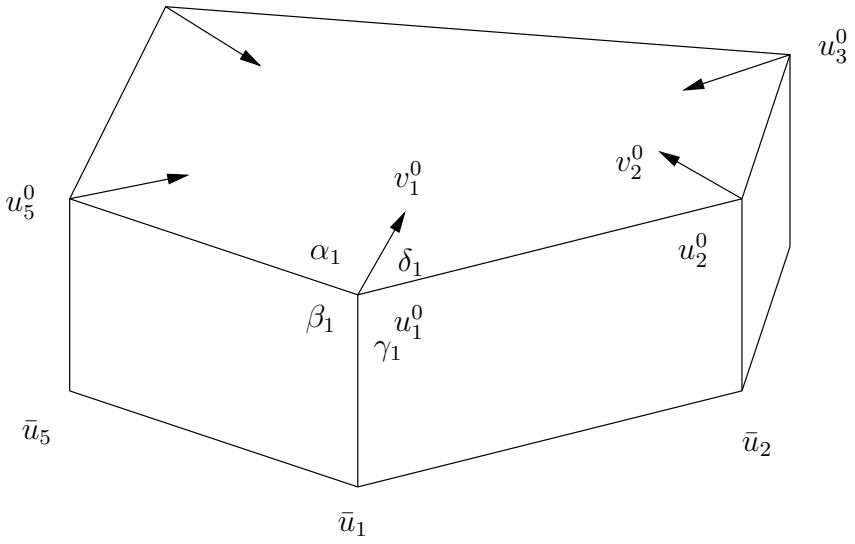

FIGURE 4. The vertices of the top face, the gable, are denoted u^0. The vertices of the opposite gable are denote $\bar{u}$. The inset vectors are v, and around each vertex u^0 there are four angles α, β, γ, δ.

We assume that the gable is convex. Let $\{u_i^0\}_{i=1}^n$ be vertices of one gable oriented counter clockwise and let $\{\bar{u}_i\}_{i=1}^n$ be the vertices of the other gable oriented so that u_i^0 and $\bar{u}_i$ are connected by an edge on the mantle, see Figure 4. Also, let $u_0^0 = u_n^0$ and $u_{n+1}^0 = u_1^0$. To each vertex u_i^0 there is an inset vector v_i^0 constrained to lie in the gable. The inset vectors form the crease pattern. Let α_i, β_i, γ_i and δ_i be the angles around vertex u_i^0, i.e.

$$\cos\alpha_i = \frac{v_i^0 \cdot (u_{i-1}^0 - u_i^0)}{\|v_i^0\|\|u_{i-1}^0 - u_i^0\|} \qquad \cos\beta_i = \frac{(u_{i-1}^0 - u_i^0)\cdot(u_i^0 - \bar{u}_i^0)}{\|u_{i-1}^0 - u_i^0\|\|u_i^0 - \bar{u}_i^0)\|}$$

$$\cos\gamma_i = \frac{(u_i^0 - \bar{u}_i^0)\cdot(u_{i+1}^0 - u_i^0)}{\|u_i^0 - \bar{u}_i^0\|\|u_{i+1}^0 - u_i^0\|} \qquad \cos\delta_i = \frac{(u_{i+1}^0 - u_i^0)\cdot v_i^0}{\|u_{i+1}^0 - u_i^0\|\|v_i^0\|}$$

Each inset vector v_i^0 is constructed such that

$$\alpha_i + \beta_i = \gamma_i + \delta_i. \tag{1}$$

Let the point of intersection of a pair of inset vectors be $w_i^0 = u_i^0 + s_i v_i^0 = u_{i+1}^0 + s_{i+1} v_{i+1}^0$, with $s_i, s_{i+1} \geq 0$. Also, let $\bar{w}_i^0$ be the orthogonal projection of w_i^0 onto the line segment (u_i^0, u_{i+1}^0), and let $h_i^0 = \|w_i^0 - \bar{w}_i^0\|$. To the left in Figure 5, the distance h is marked with a dashed line.

The algorithm for computing the crease pattern, which we call the Skew Skeleton, terminates in a finite number of steps. In each iteration, a pair of inset vectors intersect, and a new inset vector is formed. The crease pattern consists of the inset vectors $\{v_i^k\}$, and a set of additional creases. If a pair of inset vectors intersect, a crease is drawn from the point of intersection to the edge of the gable. The direction of the crease must be such that the angles around the intersection of the crease and the edge of the gable fulfill the equivalent to equation (1). In the triangle to the left in Figure 3, the bisectors correspond to the inset vectors, and the dashed lines are the additional creases. In this case, all three inset vectors intersect at one point.

When two inset vectors v_i^k, v_{i+1}^k intersect, a new inset vector can be computed by extending the edges of the gable, see Figure 6.

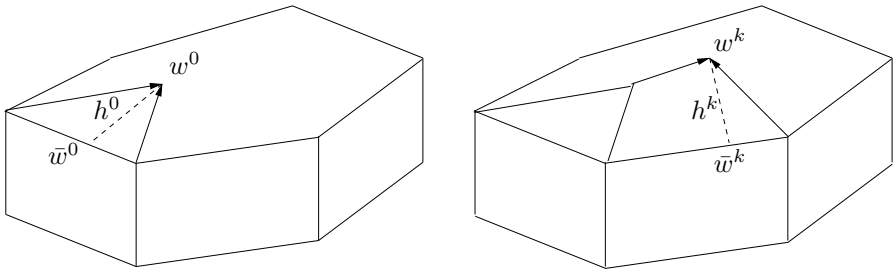

FIGURE 5. The arrows are the inset vectors. The left figure shows the first intersection, and the right shows the second intersection. The distance h is shown as a dashed line.

When only two inset vectors remains, a crease is drawn to connect them.

The crease pattern is computed identically on both gables. For each pair of additional creases, one on each gable, a crease is drawn over the mantle to connect the pair.

Algorithm Skew Skeleton
Input: Vertices $\{u_i^0\}_{i=1}^n$ and $\{\bar{u}_i\}_{i=1}^n$.
Output: List of creases C.

Set $C = \emptyset$.
for $k = 0, \ldots, n-3$
    for $i = 1, \ldots, n-k$
        Compute inset vector v_i^k.
        Compute distance h_i^k from the point of intersection
          line segment (edge of gable).
    end
    $j = \text{argmin}_i \{h_i^k\}$.
    Compute lengths s_j, s_{j+1} for intersection of v_j^k and v_{j+1}^k.
    Set $u_j^{k+1} = w_j^k = u_j^k + s_j v_j^k$.
    Add creases (u_j^k, w_j^k) and (u_{j+1}^k, w_j^k) to C.
    Add crease $(\bar{w}_j^k, w_j^k)$ to C.
    Set $u_l^{k+1} = u_l^k$ for $l \neq \{j, j+1\}$.
    Renumber the nodes u_l^{k+1}, $l = 1, \ldots, n-k$.
end
Add crease (u_1^{n-1}, u_2^{n-1}) to C.

It must be noted that the use of the distance h to determine which pair of inset vectors intersect is not entirely satisfactory. In general the inset vectors v_i^k should be extended as far as possible before they intersect any other inset vector v_i^l, $l \geq k$. However, since all the other inset vectors depend on which inset vectors are joined previously, this forms a rather tricky criterion. The algorithm Skew Skeleton does a good job in most cases.

We have assumed that the gable is convex. If this is not the case, a technique called slicing can be used. Slicing can split the gables into convex parts, and we can treat each part separately. Slicing will add additional creases along the slice, which may be beneficial if the computed crease

pattern is supposed to approximate a "physical" one which has a crease in that position. If slicing is used, the Skew Skeleton algorithm is applied to each part of the gable.

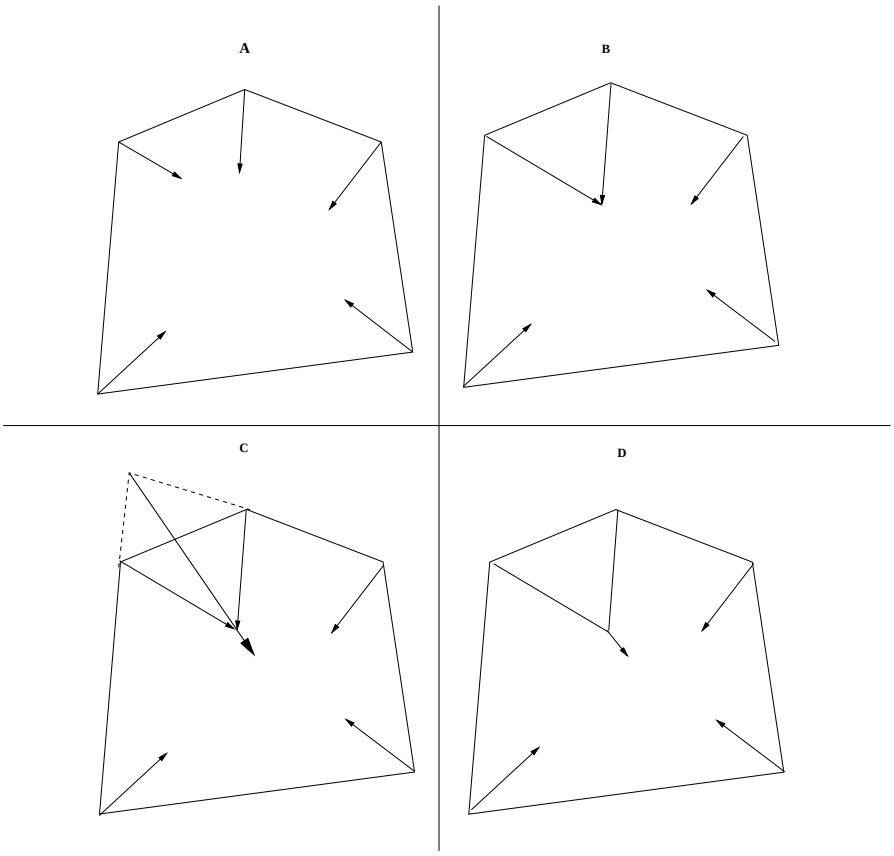

FIGURE 6. The figure shows the process of joining inset vectors in the algorithm for computing the crease pattern. A: The arrows are the inset vectors. One for each vertex. Their directions depend on the geometry of the polyhedron (not just the polygonal face). B: Two inset vectors meet before the others. C: A new inset vector is formed from the two joined inset vectors. D: The process of joining inset vectors are repeated, this time with one inset vector less.

3. Folding

Given a crease pattern for the polyhedron, the folding problem is only partially solved. To actually compute the geometry of the flat folded airbag, we need an algorithm which folds the object according to the crease pattern.

For airbags, there are various alternatives for simulating the folding process. This is specially due to the fact that the problem is artificial in the sense that it needs not be realistic, e.g., there is no need to introduce the concept of time. The objective is to create a flat geometry which is physically possible, not to fold it in a realistic way, although those two objectives may be coupled.

Our algorithm for folding the polyhedron is based on solving an optimization problem. A program is formulated such that the optimal solution represents a flat geometry. The target function, to be minimized, is a sum of rotational spring potentials, one spring over each crease. The minimal value of a spring potential is found when a fold is completed. The constraints are

formulated in order to conserve a physically correct representation of the polyhedron. This means conserving the shape and area and also avoiding self-intersections of the faces of the polyhedron.

To arrive at a suitable model, we examine the problem stepwise through a couple of examples. Each one is designed to present the problems and possibilities with the optimization approach.

3.1. **Example 1.** In a first example, we want to simulate the folding action without an unnecessarily complex model. We consider an open part of a box involving only a few creases. The creases are not generated using the crease algorithm described previously.

Physically, we know an optimum exists, and we can also imagine how to fold it. The object consists of 7 connected patches with $n = 11$ vertices, see Figure 7. The target function is the sum

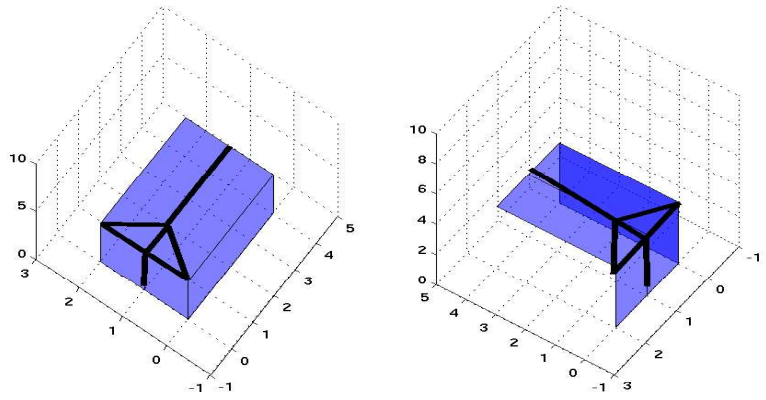

FIGURE 7. The object in Example 1 from two view points. The object is an open part of a box. The creases are marked with thick lines.

of $n_c = 7$ artificial rotational spring potentials, one over each crease. They are computed using the scalar product of the (normalized) normals n_i^1, n_i^2 of the two neighboring patches joined by a crease $i = 1, \ldots, n_c$. The scalar product is 1 when the two patches are parallel, and -1 when the fold is completed.

The constraints are chosen to conserve the edge lengths l_i of the edges $i = 1, \ldots, n_e$, where $n_e = 17$. The vertices of edge i are denoted x_i^1 and x_i^2. Let $x = (x_x^1, x_y^1, x_z^1, \ldots, x_x^n, x_y^n, x_z^n)$ store the coordinates of the vertices. The optimization problem can be formulated as,

$$\begin{aligned} \min_x f(x) &= \sum_{i=1}^{n_c} n_i^1 \cdot n_i^2 \\ \text{subject to} \quad & \|x_i^1 - x_i^2\|_2^2 - l_i^2 = 0, \ i = 1, \ldots, n_e, \end{aligned} \tag{2}$$

where $f : \mathbf{R}^{3n} \to \mathbf{R}$. The optimization problem is solved using the subroutine `fmincon` from the Matlab Optimization Toolbox [12]. It is an implementation of a medium-scale SQP method which maintains a dense quasi-Newton approximation of the Hessian. Figure 8 shows a few iteration snapshots.

The optimization subroutine terminated after 25 iterations. The minimum of the target function is $f^* = -7$. In Figure 9 the difference between the function value and the optimal value is plotted for each iteration.

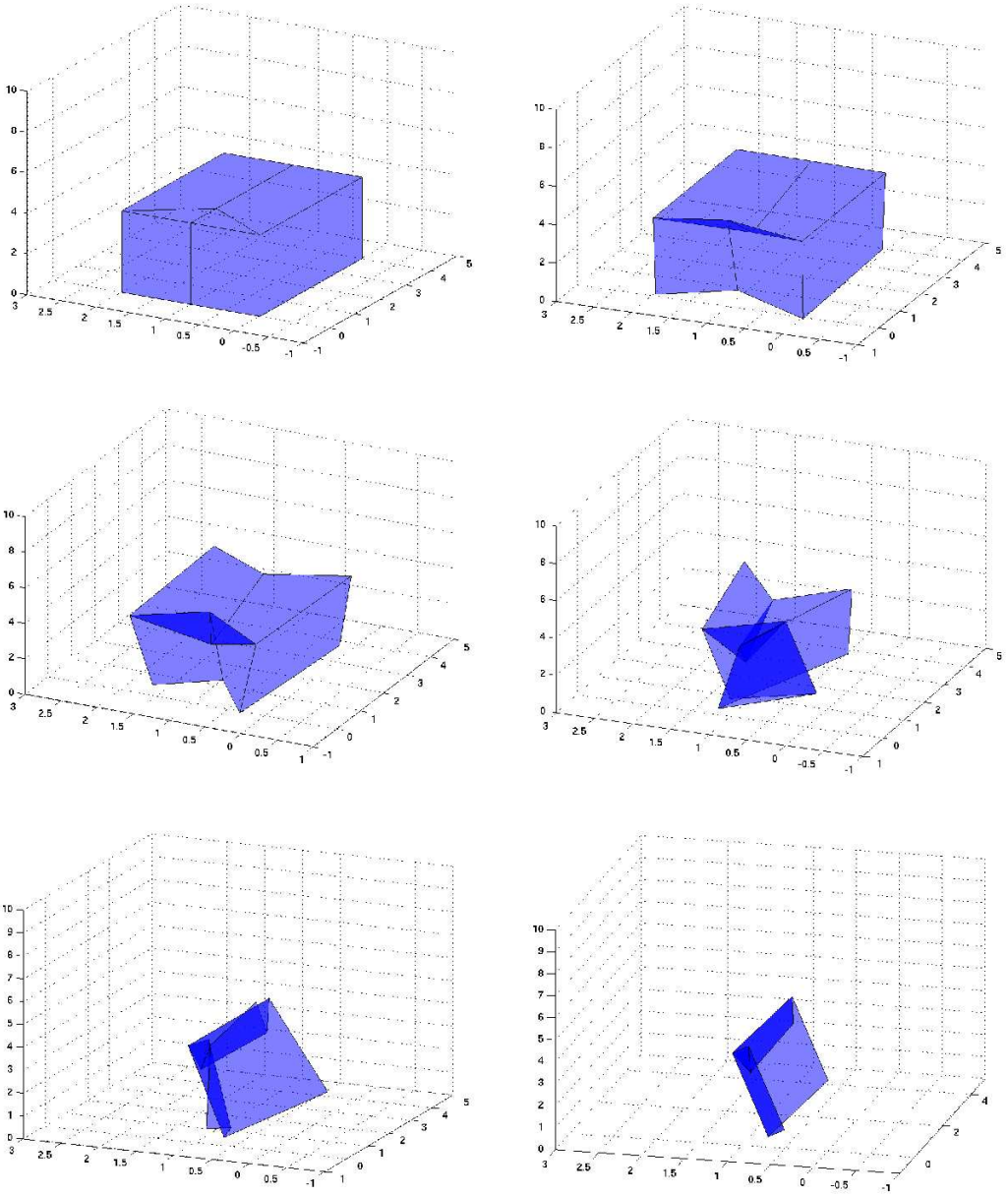

FIGURE 8. A folding of Example 1 simulated by solving an optimization program. From upper left to right: iteration 0, iteration 1, iteration 2, iteration 3, iteration 4 and iteration 20.

The example shows that a target function based on the scalar products of the normals for each crease succeeds in executing the folds at least in this example. This indicates that it could be a possible choice for the application of airbag folding.

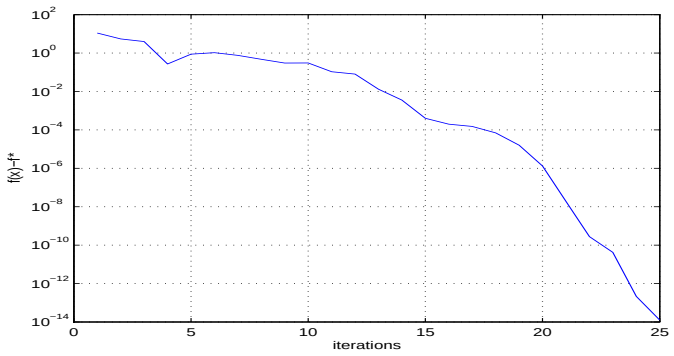

FIGURE 9. The difference between the value of the current iterate $f(x)$ and the optimal value f^* from Example 1.

3.2. **Example 2.** In a second example, again a box is to be folded. This will take us closer to the actual application, airbag folding. The crease pattern is computed using the algorithm discussed in the previous section. The first example demonstrated the use of a rotational spring potential as the driving mechanism for the folding. This time we are interested in other practical considerations. For one thing, we are now folding a complete object, in the sense that all creases are connected, and we have no "free end". Also, for a realistic application, we want to fold the object without any surface penetration. This example does not consider any contact checking, but we have prepared for this by using triangulated surfaces. Dividing the surfaces of the polyhedron into smaller triangles will enable a more accurate contact checking and also a more flexible folding process. Surface intersection will most definitely obstruct the folding, and therefore it is important that the surface is allowed to be somewhat flexible.

This example also uses the triangles to define the surface area constraint.

The crease pattern consists of $n_C = 32$ creases. It splits the faces of the box into smaller polygons, called patches, see Figure 10.

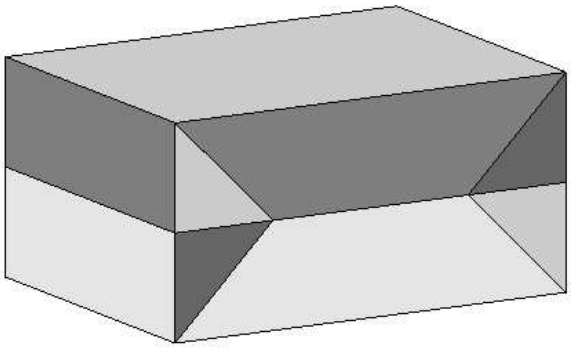

FIGURE 10. The box in Example 2. The crease pattern consists of 32 creases. The polygons are called patches.

Each patch is triangulated. The resulting optimization program is based on (2), with the modification that the normal is computed as the average of the normals of the triangles of the patch. Some of the nodes in the mesh are fixed. As in Example 1, let the coordinates of the vertices of the mesh $\{x^i\}_{i=1}^n$ be stored in $x = (x_x^1, x_y^1, x_z^1, \ldots, x_x^n, x_y^n, x_z^n)$. Let x_i^1 and x_i^2 be the vertices of edge i, $i = 1, \ldots, n_e$. Then the optimization problem is

$$\begin{aligned} \min_x f(x) &= \sum_{i=1}^{n_C} a_i \, n_i^1 \cdot n_i^2 \\ \text{subject to} \quad & h_i(x) = \|x_i^1 - x_i^2\|_2^2 - l_i^2 = 0, \; i = 1, \ldots, n_e \\ & x_j = 0, \quad j \in J. \end{aligned} \tag{3}$$

The last constraint fixes a set J of node coordinates: node 1 is fixed to the origin, node 19 is fixed to the xz-plane, and node 41 is fixed along the x-axis, see Figure 11. This will fix a reference position for the box, and will not interfere with the folding. The constant a_i is equal to 1 or -1. It is used to control the individual creases.

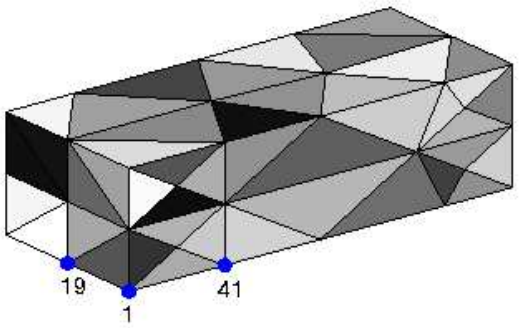

FIGURE 11. The meshed box in folding Example 2. The nodes 1, 19, and 41 are marked.

After 97 iterations the optimization routine `fmincon` stopped when the the number of function evaluations exceeded a given threshold.

A completely folded box should have an optimal value $f^* = -32$. In Figure 13, the function value is plotted for each iteration. As is shown, most of the progress is halted after just 3 iterations. In Figure 12, the meshed box is shown after 3 iterations and 97 iterations. A reason for the slow progress may be found in the properties of the Jacobian H of the constraints. The Jacobian for this example is a square matrix of size 138. The quotient of the smallest and the largest singular values of H is

$$\frac{\sigma_{\min}(H)}{\sigma_{\max}(H)} \approx 2.20 \times 10^{-3},$$

indicating it is non-singular. Since the SQP-method in `fmincon` uses H to find a direction p such that $Hp = 0$, naturally the iteration progress is slow. This example was constructed, but it still shows that this problem can occur and must be dealt with. A conclusion drawn from this, is that it may be better to preserve the edge lengths by including the constraints as penalties, instead of regular equality constraints.

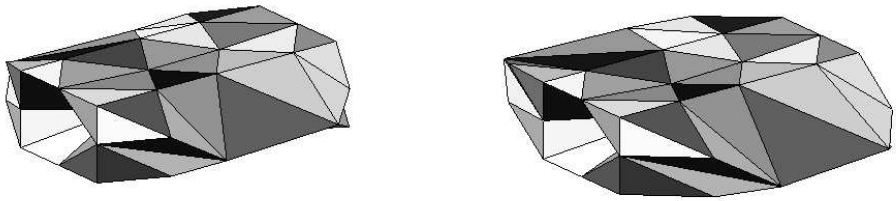

FIGURE 12. The meshed box in folding Example 2. To the left the box after 3 iterations using `fmincon`, and to the right the box after 97 iterations.

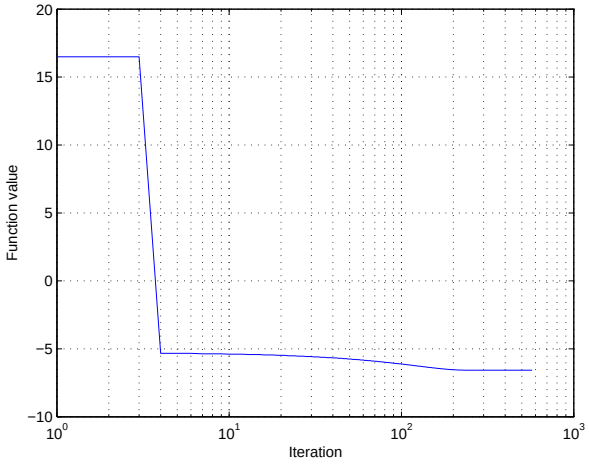

FIGURE 13. The function value progress for Folding example 2.

3.3. **Folding Model.** With motivation drawn from previous examples, we will formulate an optimization problem which will be used for folding the polyhedral airbags. One issue not addressed before is the problem of surface penetration. Most contact checking routines only report if and where a contact has occurred. In an optimization environment, such a constraint works poorly, since it is not continuous. Instead, we seek an alternative continuous constraint. One possibility is to check the distance to contact, and then require positive distance. A function reporting also negative "distances" is preferred. It indicates how far the penetration has gone. We construct this by using tetrahedra to fill the interior of the polyhedron, and then check that all surface points are outside of all the tetrahedra (except the ones it belongs to).

The crease pattern over a polyhedron induces a subdivision of its surface of polygons called patches. In addition, the patches are triangulated, and the interior of the polyhedron is meshed with tetrahedra. Let the nodes of the mesh be $\{x^i\}_{i=1}^n$, and let the indices of the surface nodes be I_S. Let the tetrahedra be $\{K_i\}_{i=1}^{n_K}$ and set $I_K = \{1, \ldots, n_K\}$. Let the four indices of the nodes of tetrahedron k be $V_k(i)$, $i = 1, \ldots, 4$. The edges of the triangular faces are denoted $\{E_i\}_{i=1}^{n_E}$, and the indices of the two nodes of edge e are $W_e(i)$, $i = 1, 2$.

Denote the creases $\{C_i\}_{i=1}^{n_C}$. The spring potential over each crease C_i is computed using the scalar product of the normalized normals, n_i^1,n_i^2, of the two neighboring patches. The normals point outward from the polyhedron.

The folding process of a polyhedron with n nodes (surface and interior mesh nodes) is formulated as the following nonlinear program with $f : \mathbf{R}^{3n} \to \mathbf{R}$,

$$\min_x f(x) \tag{4}$$

$$\begin{aligned} f(x) &= f_1(x) + f_2(x) + f_3(x) \\ &= k_m \sum_{k=1}^{n_K} \left(\sum_{1 \le i < j \le 4}^{4} \|x^{V_k(i)} - x^{V_k(j)}\| - d_{V_k(i),V_k(j)} \right)^2 \\ &\quad + \sum_{i=1}^{n_C} a_i \, n_i^1 \cdot n_i^2 + k_p \sum_{i=1}^{n_E} \left(\|x^{W_i(1)} - x^{W_i(2)}\| - l_{W_i} \right)^2, \end{aligned}$$

subject to

$$\begin{aligned} \mathrm{vol}(K_i) &\ge \varepsilon_1, \qquad i = 1, \ldots, n_K, \\ \mathrm{dist}(x^i, K_j) &\ge \varepsilon_2, \qquad i \in I_S,\ j \in I_K \setminus p_i, \end{aligned}$$

where d_{ij} is the original distance between node x^i and x^j, l_i is the original length of edge i and k_m, k_p are penalty parameters. The constant a_i is equal to 1 or -1. The first constraint function, $\mathrm{vol}(K_i)$, is the signed volume of the tetrahedron K_i. The second constraint, $\mathrm{dist}(x^i, K_j)$, is the distance from a surface node x^i to a tetrahedron K_j, and p_i are the tetrahedron indices connected to node x^i. Finally, ε_1 and ε_2 are small positive constants.

The target function f is composed of three parts: f_1 is a penalty function which strives to keep the tetrahedral mesh as uniform as possible, f_2 is the virtual spring potential which drives the folding, and f_3 is a penalty function which keeps the edges of the triangles stiff. The last one is used to maintain the shape and surface area of the patches.

4. Numerical results

The purpose of this section is to establish some numerical results concerning the folding mechanism. Before applying the folding to airbags, we want to investigate the effect of different parameters as well as different crease patterns to the folding performance.

The crease patterns are computed using a `Matlab` GUI described in the next section. The mesh was created using `TetGen`[1] [17]. The computation time for computing the crease pattern is in seconds.

The optimization problem was solved using an SQP algorithm described in [4]. If not stated otherwise, the following parameters are used in the folding, see (4): $k_m = 10^{-6}$, $\epsilon_1 = 10^{-8}$, and $\epsilon_2 = 0$. If the distance between a node and a tetrahedron is less than 10^{-2}, the pair is included as a constraint for the QP.

Since there are no guarantees that the optimization routine will terminate at a global minimum for these folding problems it might be questionable to draw conclusions about the foldability of a particular model. The optimization routine is a local method, which means that it does not actively tries to find a global minimum, only a local one. However, when the optimization terminates without finding a minimum this is (if not stated otherwise) due to the fact that no

[1]The command line interface was used with the parameters 'qMa0.1C'. The maximum size of the tetrahedra were constrained to be less than 0.1.

descent direction is found. This indicates that it cannot find a possible way through, without severely violating feasibility. The conclusion that we draw from this is that we cannot find a continuous map from the unfolded object to the flat folded object.

4.1. **Edge Penalty.** In a first setup, the folding algorithm was applied to a $4 \times 3 \times 2$ box, with a crease pattern identical to Example 2. The data for the mesh output is shown in Table 1.

TABLE 1. The mesh statistics for the box.

Nodes	Tetrahedra	Faces	Creases
159	510	260	32

Different values of the penalty parameter k_p for the edge lengths are compared to the number of iterations required for folding. Also, the surface area of the flat object were compared to the initial surface area. The result is shown in Figure 14 and Table 2. The optimization routine stopped if the objective value was within 0.1 of the optimal value $f^* = -32$. N/A indicates that the optimization routine did not converge.

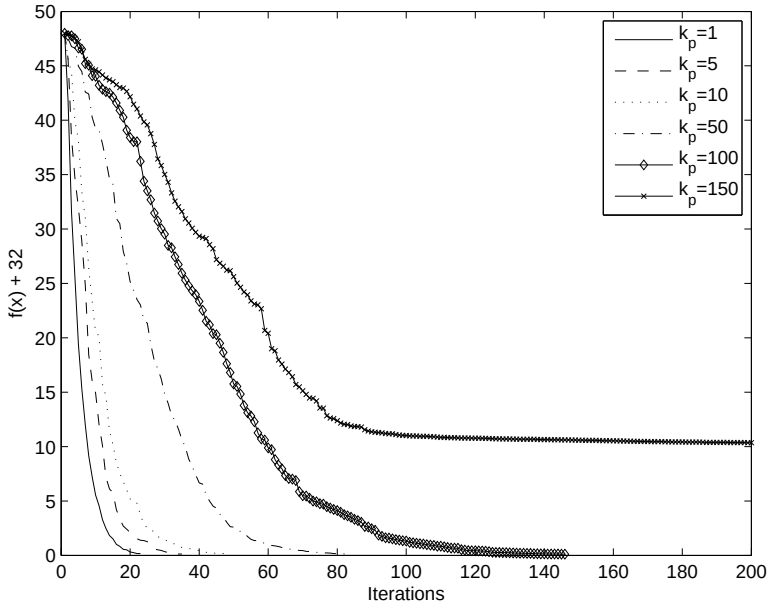

FIGURE 14. Iteration progress for different values of the penalty parameter k_p, see (4).

TABLE 2. The table shows the number of iterations required to fold the box for different values of the penalty parameter k_p. Also, the relative change in surface area is reported.

k_p	No. Iterations	$\lvert A - A_0\rvert / A_0$
1	22	4×10^{-3}
5	34	1×10^{-3}
10	48	9×10^{-4}
50	83	7×10^{-4}
100	145	2×10^{-4}
150	N/A	N/A

The results indicate that there may be an upper limit to the value of k_p for which a complete fold is attained. In contrast, a low value of k_p will result in greater area loss.

4.2. **Skewness.** In this setup, the box from the previous subsection was tilted symmetrically such that the gables become skew. We let the parameter α denote the skewness of the polyhedron, see Figure 4.2.

The polyhedron is a quasi-cylindrical polyhedron with quadrilateral gables, and the crease pattern algorithm described previously is applicable. This numerical test shows the performance of the folding algorithm as a function of the skewness angle. Since a change of geometry will result in a different mesh, the effect of a change in the skewness angle cannot be fully isolated when it comes to the folding performance.

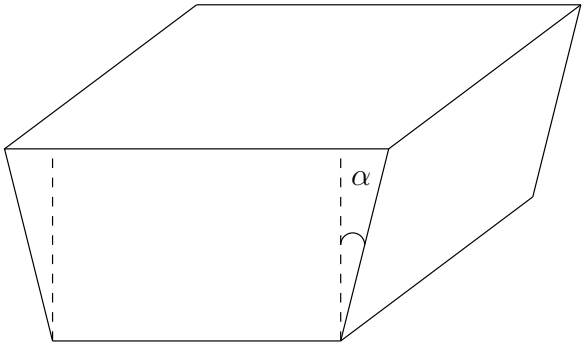

FIGURE 15. The titled box with a skewness angle α. The base of the box is still 4×3. The dotted lines are 2 units long. The gables are the right and left face.

The skewness angle α and the number of iterations required are displayed in the Table 3. As before, N/A means that the optimization routine did not converge.

TABLE 3. The table shows the number of iterations required for folding depending on the skewness angle. Also, the mesh statistics are shown.

Skewness Angle (^{o})	Nodes	Tetrahedra	Faces	No. Iterations
0	159	510	260	48
10	257	761	442	60
20	199	660	324	56
30	231	755	374	59
40	238	792	382	N/A
60	262	875	424	N/A

The conclusions drawn here is that our folding method is not well suited to handle very skew objects. The crease pattern is correct, but the poor performance is probably due to the mesh quality. For large values of α, some elements in the mesh are proportionally small and may obstruct the folding.

4.3. **Crease Pattern.** In this subsection, we study the effect of different crease patterns to a particular model A. The model is quasi-cylindrical with gables consisting of 8 vertices. Four different crease patterns are examined, see Figure 16. They are created using non-crossing diagonals which split the gables into sub-polygons. The edge penalty parameter was set to default, $k_p = 10$.

The result of the folding is shown in Table 4. For each crease pattern the mesh statistics and the number of creases are compared to the number of iterations for convergence. Also, the relative difference in area between the initial unfolded model and the flat model is reported.

It is difficult to draw any conclusion of the impact of using different crease patterns for the same model. In this test, it would appear that it has no significant effect on the folding performance.

5. Application to airbag folding

In the previous section, some of the properties of the folding algorithm were tested. We are now interested in examining its application to airbag folding, since this is our objective for developing algorithms for crease patterns and folding.

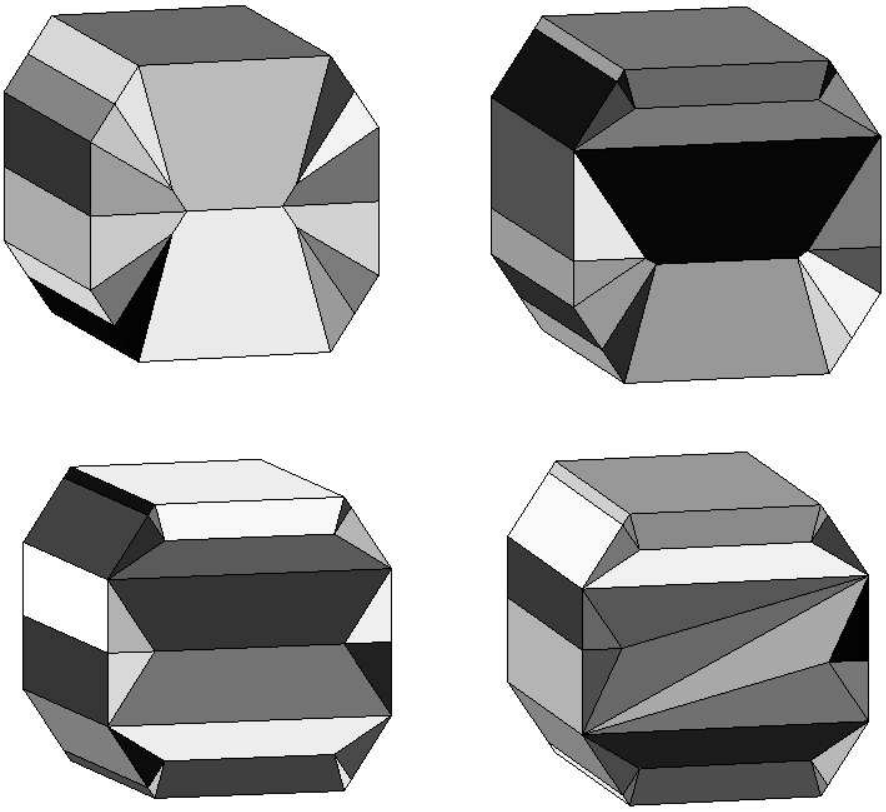

FIGURE 16. Model A with different crease patterns. Upper left is A_1 (crease pattern number 1), upper right is A_2, lower left is A_3 and lower right is A_4.

TABLE 4. The table shows the number of iterations required for the different crease patterns. Also, the mesh statistics as well as the area loss is shown.

Crease pattern No.	Nodes	Tetrahedra	Faces	Creases	No. Iterations	$\lvert A - A_0\rvert / A_0$
1	496	2003	646	80	108	7.1×10^{-4}
2	627	2422	870	84	103	6.6×10^{-5}
3	579	2223	804	88	93	1.2×10^{-3}
4	988	3561	1520	100	114	2.2×10^{-4}

The airbag model which is considered is a three-dimensional passenger airbag provided by Autoliv Research. It has nearly the shape of a quasi-cylindrical polyhedron, although its surface is more curved. The geometry of the airbag is described in CAD-format, and therefore the first step is to acquire a quasi-cylindrical approximation. It is difficult to list the properties the approximation must have in order to be a good approximation. Two simple measures are surface area and "inflated" volume. However these do not describe the shape of the airbag precisely. In our case, an approximation was made manually, so that the result was satisfactory when it comes

to the shape. The area and volume were measured: the approximation differed about 0.5% in area and volume. The polyhedron approximation with the crease pattern that was used can be found in Figure 17.

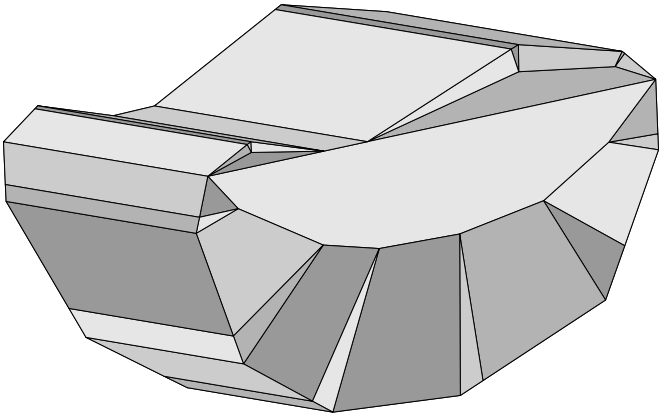

FIGURE 17. Polyhedral approximation of an airbag model together with a computed crease pattern.

Slicing was used twice to create the crease pattern. Two upper "bumps" were sliced, creating a partition of the gables into three parts. The crease pattern was computed on each part. The airbag geometry is quasi-cylindrical, and the upper part is wider than the lower part, which creates a skewness that was discussed in the previous section.

In production, the airbag is flattened in a simple manner. The airbag is just laid out flat, and stretched in some direction. To resemble this flattening, we did not use any forced creases (besides the two that split the gables into convex parts).

A few iteration snapshots of the airbag folding are shown in Figure 18. The end result was acquired after about 200 iterations. The surface area of the flat folded polyhedron was found to be within 0.5% of the surface area of the unfolded polyhedron.

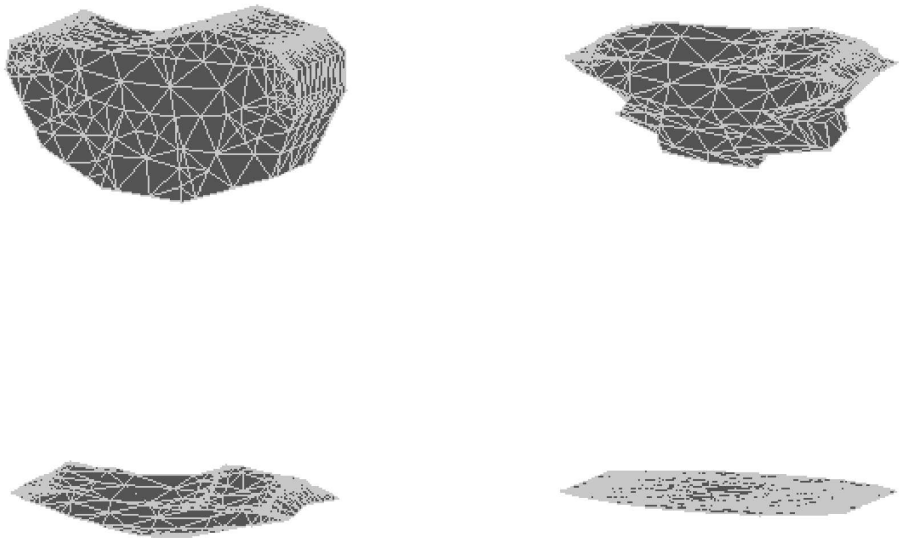

FIGURE 18. The figures show iteration snapshots from the folding of the polyhedron approximation from Figure 17. The upper left shows the unfolded polyhedron, the upper right: 40 iterations, the lower left: 60 iterations, and the lower right: 200 iterations.

As an effect of contact prevention, the number of constraints for each subproblem (QP) in the optimization routine increases, see Figure 19.

6. Software

Both the crease pattern generation and folding procedure are implemented in software. The subroutines for computing the crease pattern are written in `Matlab`. They are organized around a Graphical User Interface (GUI), see Figure 20, which is also capable of displaying the folding result and exporting files in `LS-DYNA` format.

The triangular and tetrahedral mesh are generated using `TetGen` [17].

The optimization method is described in [4]. It is implemented in the program `foldopt` and written in FORTRAN 90. All the linear algebra is performed using high performance `BLAS`[2] and `LAPACK`[3] for efficiency. In particular, `ACML`[4] has been used. The solution of the augmented

[2]**B**asic **L**inear **A**lgebra **S**ubroutines implemented in Fortran 77. See `http://www.netlib.org`

[3]**L**inear **A**lgebra **PACK**age implemented in Fortran 77. See `http://www.netlib.org.`

[4]**A**MD **C**ore **M**ath **L**ibrary. See `http://developer.amd.com/acml.aspx`

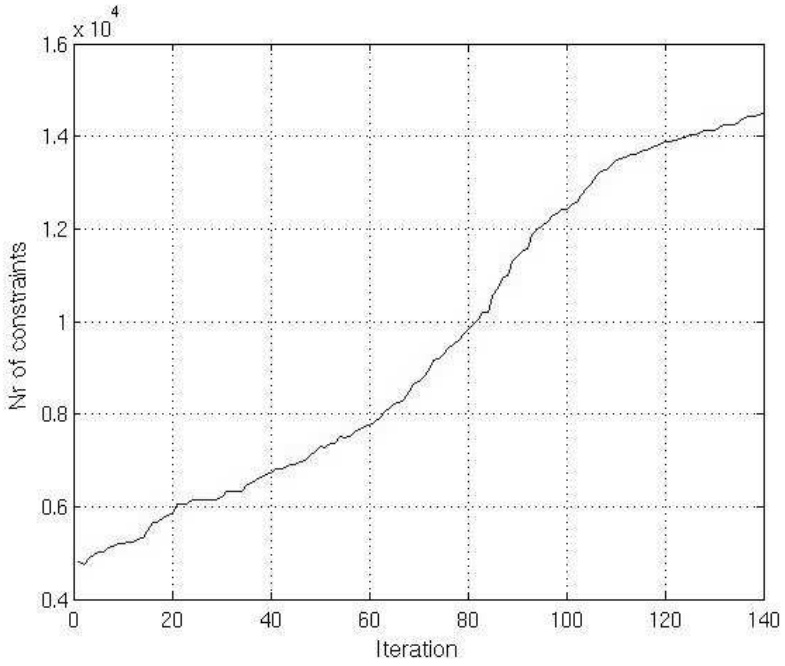

FIGURE 19. The number of constraints included in the optimization subproblem for each iteration.

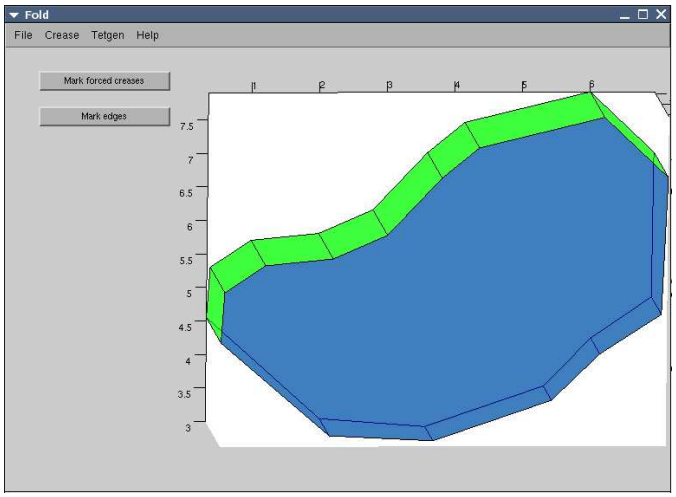

FIGURE 20. Graphical User Interface.

system is obtained using a shared-memory multiprocessing parallel direct sparse solver `PARDISO 3.0` [18, 15, 16, 14].

The optimization routine `foldopt` is shared-memory parallel, using `OpenMP`[5] directives. Finite differences are used to compute the derivatives. Estimations of proper interval lengths can be found in [7]. Both the computation of the numerical derivatives of the gradient of the target function and the Jacobian of the constraints are threaded to run in parallel on a shared-memory processor.

Communication between the Matlab GUI and `foldopt` is done through the use of ASCII text files. The output from the GUI and `TetGen` is sent to `foldopt`. The files store the mesh, pointers from creases to adjacent patches, and data indicating if the normals of the adjacent patches should point in the same or opposite direction. The output, node coordinates and connectivity, from `foldopt` can be imported to the GUI and in `OpenDX`.

In `foldopt`, all the penalty parameters in (4) are user-settable. Feasibility and progress of the iteration are used as the termination criteria for the nonlinear program. A maximal number of iterations can also be set. For each QP, primal and dual feasibility as well as duality gap are used as termination criteria.

References

[1] O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gärtner, *A novel type of skeleton for polygons*, J. Universal Computer Science **1** (1995), 752–761.

[2] E. M. Andersen, *Paperfolding*, `http://www.paperfolding.com`.

[3] Arup, *Oasys-primer*, `http://www.arup.com`.

[4] C. Cromvik, *A low-storage sequential quadratic programming method*, Preprint, Chalmers University of Technology, 2007.

[5] C. Cromvik and K. Eriksson, *Airbag folding based on origami mathematics*, Preprint, Chalmers University of Technology, 2006.

[6] ESI-group, *Easi-folder*, `http://www.esi-group.com`.

[7] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Elsevier Academic Press, 1986.

[8] R. J. Lang, *Airbag folding*, `http//www.langorigami.com`.

[9] ———, *A computational algorithm for origami design*, Annual Symposium on Computational Geometry, 1996.

[10] ———, *Treemaker 4.0: A program for origami design*, 1998, `http://origami.kvl.nl/programs/TreeMaker/trmkr40.pdf`.

[11] Livermore Software Technology Corp., *LS-DYNA*, `http://www.lstc.com`.

[12] The Mathworks, Inc., *Optimization Toolbox User's Guide*, third ed., 2005.

[13] J. O'Rourke, *Computational Geometry in C*, second ed., Cambridge University Press, Cambridge, 1998.

[14] O. Schenk and K. Gärtner, *Efficient sparse LU factorization with left-right looking strategy on shared memory multiprocessors*, BIT **40** (2000), 158–176.

[15] ———, *On fast factorization pivoting methods for sparse symmetric indefinite systems*, Technical report, Department fo Computer Science, University of Basel, 2004.

[16] ———, *Solving unsymmetric sparse systems of linear equations with PARDISO*, J. Future Generation Computer Systems **20** (2004), 475–487.

[17] H. Si, *Tetgen a quality tetrahedral mesh generator and three dimensional Delaunay triangulator version 1.3 user's manual*, Technical report, Weierstrass-Institut für Angewandte Analysis und Stochastik, 2004.

[18] University of Basel, *PARDISO 3.0*, `http://www.computational.unibas.ch/cs/scicomp/software/pardiso/`.

[19] Z. You and K. Kuribayashi, *A novel origami stent*, Summer Bioengineering Conference, 2003, `http://www.tulane.edu/ sbc2003/`.

Department of Mathematical Sciences, Chalmers University of Technology, SE–412 96 Göteborg, Sweden

Department of Mathematical Sciences, Göteborg University, SE–412 96 Göteborg, Sweden
E-mail address: `christoffer.cromvik@chalmers.se`

[5]OpenMP is a specification for a set of compiler directives, library routines, and environment variables that can be used to specify shared memory parallelism in Fortran and C/C++ programs. See www.openmp.org

A LOW-STORAGE SEQUENTIAL QUADRATIC PROGRAMMING METHOD

CHRISTOFFER CROMVIK

Abstract. A Sequential Quadratic Programming method is formulated which only requires access to the first order derivatives of the objective function and the constraints. It is a low storage method based on the L-BFGS update of the Hessian. Convergence is proved under certain assumptions, and some numerical results are provided.

1. Introduction

We consider the nonlinear program of the following form:

$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & h(x) = 0 \\ & g(x) \geq 0, \end{aligned} \tag{1}$$

where $f : \mathbf{R}^n \to \mathbf{R}$, $h : \mathbf{R}^n \to \mathbf{R}^p$ and $g : \mathbf{R}^n \to \mathbf{R}^m$.

The algorithm presented is a Sequential Quadratic Programming (SQP) method designed for large-scale problems. Several SQP methods with successful implementations have been developed, but the earliest reference is in the work of Wilson [20] in 1963. Boggs and Tolle [1] provides a survey of the early and recent developments of various SQP-methods.

SQP has the advantage that only one main optimization problem needs to be solved in contrast to penalty based algorithms. SQP does not require feasible iteration points which is advantageous for nonlinear constraints, and the method usually experiences fast convergence near the solution due to its derivation from Newton's method. Practically, SQP appear to be particularly favorable when the constraints are highly nonlinear.

SQP methods have been successfully applied to small and medium scale problems ($n < 1000$), however it has not been thoroughly examined for large-scale problems. One of the reasons for this is that when inequality constraints are present either a direct strategy for determining the active constraints must be used or a Quadratic Problem (QP) needs to be solved at each major iteration. The QP is often the most time consuming part of an SQP code.

Early works with SQP methods were developed to use secant updates, e.g., the BFGS-update, for Hessian approximations. Recent developments though have been targeted to use the exact Hessian to achieve faster convergence. In some cases the Hessian is available or it can be computed sufficiently fast by using algorithmic differentiation (AD), see [7].

In the algorithm presented here, we will not use the explicit Hessian. Instead we will use an approximation using secant-updates. This is justified for large problems where the Hessian is dense. With an approximation also comes the possibility of using a positive definite matrix. This will lower the computational burden, since the QP solved at each iteration will be a convex problem.

2. Algorithm

The SQP method presented in what follows is an inequality constrained QP (IQP) method. This means that a quadratic subproblem is solved at each major iteration and all the constraints

from (1) are included in the QP. The alternative method to IQP is equality constrained QP (EQP) method. A QP with only equality constraints is much easier to solve, but it is not clear how to decide which constraints should be included as equalities at each iteration.

In the following subsections, the SQP method is described. In short, at each iteration a QP is solved, and the solution of this subproblem will give a step in the primal and the dual variables. Using a line search, the variables are updated.

2.1. **Local SQP.** We present a local SQP method where "local" indicates that it requires starting points to be sufficiently close to the minimum to converge. In Subsection 2.4, a merit function is formulated which enables convergence from remote starting points.

The Lagrangian function for problem (1) is

$$L(x, \lambda_h, \lambda_g) = f(x) - \lambda_h^{\mathrm{T}} h(x) - \lambda_g^{\mathrm{T}} g(x)$$

where $\lambda_h \in \mathbf{R}^p$ and $\lambda_g \in \mathbf{R}^m$ are the Lagrange multipliers, and $h(x) = [h_1(x), \dots, h_p(x)]^{\mathrm{T}}$, $g(x) = [g_1(x), \dots, g_m(x)]^{\mathrm{T}}$. Let $I(x^*) = \{j : g_j(x^*) = 0\}$ be the set of active (inequality) constraints at the optimal solution x^*.

Definition 1. *Given the optimal point x^* and the active set $I(x^*)$, the linear independence constraint qualification (LICQ) holds if the set of active constraint gradients*

$$\{\nabla h_i(x^*), \nabla g_j(x^*), i = 1, \dots, p,\ j \in I(x^*)\}$$

is linearly independent.

If LICQ holds at the optimal solution, then the first order necessary optimality conditions, also known as the KKT-conditions, state that there exists multipliers $[\lambda_h^*, \lambda_g^*]$ such that

$$\begin{aligned} &\nabla_x L(x^*, \lambda_h^*, \lambda_g^*) = 0 \\ &(\lambda_g^*)_j g_j(x^*) = 0, \quad j = 1, \dots, m \end{aligned} \tag{2}$$

for an optimal solution x^*. A local Sequential Quadratic Programming method (SQP) can be derived from Newton's method applied to the system of equations

$$F(x, \lambda_h, \lambda_g) = \begin{bmatrix} \nabla_x L(x, \lambda_h, \lambda_g) \\ h_i(x) \\ g_j(x) \end{bmatrix} = 0 \tag{3}$$

with $i = 1, \dots, p$, $j \in I(x^*)$. Note that, usually, the active constraints at the optimal solution, $I(x^*)$, are not known.

Let A denote the Jacobian of the active constraints $I(x^*)$, i.e.,

$$A(x) = [\nabla h_i(x), \nabla g_j(x)]^{\mathrm{T}}, \quad i = 1, \dots, p,\ j \in I(x^*).$$

Also, let

$$c = [h_i(x), g_j(x)]^{\mathrm{T}}, \quad i = 1, \dots, p,\ j \in I(x^*)$$

and

$$\lambda = [(\lambda_h)_i, (\lambda_g)_j]^{\mathrm{T}}, \quad i = 1, \dots, p,\ j \in I(x^*).$$

Let x_k be the solution at iteration k, and $A_k = A(x_k)$, $c_k = c(x_k)$, $\nabla f_k = \nabla f(x_k)$. Then, a Newton direction to (3) with $[x_{k+1}, \lambda_{k+1}] = [x_k, \lambda_k] + [p_k, q_k]$ is given by

$$\begin{bmatrix} W_k & -A_k^{\mathrm{T}} \\ A_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ p_\lambda \end{bmatrix} = \begin{bmatrix} -\nabla f_k + A^{\mathrm{T}} \lambda_k \\ -c_k \end{bmatrix} \tag{4}$$

where $W_k = \nabla_x^2 L(x_k, \lambda_k)$ is the Hessian of the Lagrangian. Setting $\lambda_{k+1} = \lambda_k + q_k$ in equation (4) gives

$$\begin{bmatrix} W_k & -A_k^{\mathrm{T}} \\ A_k & 0 \end{bmatrix} \begin{bmatrix} p_k \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} -\nabla f_k \\ -c_k \end{bmatrix} \tag{5}$$

The Newton direction is well defined if the Jacobian in (4) (the KKT-matrix) is nonsingular. This holds if the constraint Jacobian A_k has full row rank, and if $p^{\mathrm{T}}W_k p > 0$ for all $p \neq 0$ such that $A_k p = 0$.

The solution (p_k, λ_{k+1}) to equation (5) is equal to the primal and dual optimal solution of the following Quadratic Problem (QP):

$$\begin{aligned} \min_p \quad & \tfrac{1}{2} p^{\mathrm{T}} W_k p + \nabla f_k^{\mathrm{T}} p \\ \text{subject to} \quad & A_k p + c_k = 0 \end{aligned} \tag{6}$$

Instead of estimating the active set $I(x^*)$, we include all the constraints in the QP:

$$\begin{aligned} \min_p \quad & \tfrac{1}{2} p^{\mathrm{T}} W_k p + \nabla f_k^{\mathrm{T}} p \\ \text{subject to} \quad & H_k p + h_k = 0 \\ & G_k p + g_k \geq 0 \end{aligned} \tag{7}$$

where $H_k = [\nabla h_1^{\mathrm{T}}(x_k), \ldots, \nabla h_p^{\mathrm{T}}(x_k)]^{\mathrm{T}}$, $G_k = [\nabla g_1^{\mathrm{T}}(x_k), \ldots, \nabla g_m^{\mathrm{T}}(x_k)]^{\mathrm{T}}$.

As was mentioned in the introduction, we will use an approximation B_k of the Hessian W_k. The approximation is constructed from secant updates and it is positive definite.

2.2. **Quadratic Problem.** In each SQP iteration problem, the QP in (7) is solved. From now on, we will drop the subscript k, and we will only treat inequality constraints. Setting $B = W_k$, $d = \nabla f_k$, $A = G_k$, and $b = g_k$, we get

$$\begin{aligned} \min_p \quad & \tfrac{1}{2} p^{\mathrm{T}} B p + d^{\mathrm{T}} p \\ \text{subject to} \quad & Ap + b \geq 0 \end{aligned} \tag{8}$$

where $A \in \mathbf{R}^{m \times n}$, $b \in \mathbf{R}^m$. We use a primal-dual interior-point solver to solve (8). Since the approximated Hessian B is chosen to be positive definite, (8) is a convex program. As with SQP, the method can be derived from Newton's method applied to the first order optimality conditions for (8). Introducing slack variables $y = A\,x + b$, and Lagrange multipliers λ, the conditions are

$$\begin{aligned} B\,x - A^T\,\lambda + d &= 0 \\ A\,x - y + b &= 0 \\ y_i\,\lambda_i &= 0, \quad i = 1, ..., m \\ y &\geq 0 \\ \lambda &\geq 0 \end{aligned} \tag{9}$$

The first equation is for Lagrangian stationarity, the second for primal feasibility, and the third one for complementarity slackness. The optimality conditions can be formulated as a constrained nonlinear system of equations

$$F(x, \lambda, y) = \begin{bmatrix} B\,x - A^T\,\lambda + d \\ A\,x - y + b \\ Y\,\Lambda\,e \end{bmatrix} = 0, \quad \lambda, y \geq 0 \tag{10}$$

where

$$\Lambda = \mathrm{diag}(\lambda_1, ..., \lambda_m), \quad Y = \mathrm{diag}(y_1, ..., y_m), \quad e = [1, ..., 1]^T$$

The idea behind the primal-dual interior-point algorithm is to solve (10) with an augmented right hand side. Instead of solving $F = 0$, the solution is obtained by solving

$$F(x_\tau, \lambda_\tau, y_\tau) = \begin{bmatrix} 0 \\ 0 \\ \tau\, e \end{bmatrix}, \quad \lambda_\tau, y_\tau > 0 \tag{11}$$

where $\tau \geq 0$. The solution $(x_\tau, \lambda_\tau, y_\tau)$ is said to be on a central path. When $\tau \to 0$, complementarity will hold for the solution, and it will be optimum. Let μ be a measure of the complementarity condition $(y_i \lambda_i = 0)$ in (9),

$$\mu = \frac{y^T \lambda}{m}$$

and introduce $\sigma \in (0, 1)$, such that $\tau = \sigma\mu$. The parameter σ is called a centering parameter. At each iteration, we set the right hand side set to

$$F = \begin{bmatrix} 0 \\ 0 \\ \sigma\, \mu\, e \end{bmatrix}, \tag{12}$$

and solve the system using a modification of Newton's method. The Newton step is found by solving the system of equations

$$\begin{bmatrix} B & -A^T & 0 \\ A & 0 & -I \\ 0 & Y & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta y \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_b \\ -r_\sigma \end{bmatrix} \tag{13}$$

with

$$r_d = B\, x - A^T\, \lambda + d, \quad r_b = A\, x - y + b, \quad r_\sigma = \Lambda Y\, e - \sigma\, \mu\, e. \tag{14}$$

If $\sigma = 1$, the step is called centering, because if (12) is solved exactly, $\lambda_i y_i = \mu$. When $\sigma \to 0$, the iterates are forced along a path to optimality. The next iterate is found by a line search

$$(x^{k+1}, \lambda^{k+1}, y^{k+1}) = (x^k, \lambda^k, y^k) + \theta^k\, (\Delta x^k, \Delta\lambda^k, \Delta y^k) \tag{15}$$

with θ^k chosen such that the iterates remain in the interior of the domain, i.e., $\lambda^k, y^k > 0$.

The following theorem shows convergence under the very strong assumption that the step length θ^k is bounded from below. The proof is based on the proof for linear problems, see [18].

Theorem 2.1. *Suppose there is a real number $t > 0$ and an integer K such that for all $k \leq K$,*

$$t \leq \theta^k \leq 1$$

Then after $k \leq K$ iterations, the following holds:

$$\begin{aligned} \|r_d^k\| &\leq (1 - t)^k \|r_d^0\|, \\ \|r_b^k\| &\leq (1 - t)^k \|r_b^0\|, \\ \gamma^k &\leq (1 - \bar{t})^k \gamma^0 \end{aligned}$$

where

$$\begin{aligned} \gamma^k &= (\lambda^k)^T y^k \\ \bar{t} &= (1 - \sigma + \sigma t) t \leq 1 \end{aligned}$$

Proof. From (14), (15), and (13) we have

$$\begin{aligned} r_d^{k+1} &= Bx^{k+1} - A^{\mathrm{T}}\lambda^{k+1} + d \\ &= Bx^k - A^{\mathrm{T}}\lambda^k + d + \theta^k(B\Delta x^k - A^{\mathrm{T}}\Delta\lambda^k) \\ &= r_d^k - \theta^k r_d^k = (1-\theta^k) r_d^k \\ r_b^{k+1} &= Ax^{k+1} - y^{k+1} - b \\ &= Ax^k - y^k - b + \theta^k(A\Delta x^k - \Delta y^k) = (1-\theta) r_b^k \end{aligned}$$

Also

$$\begin{aligned} \gamma^{k+1} &= (\lambda^k + \theta^k\Delta\lambda^k)^{\mathrm{T}}(y^k + \theta\Delta y^k) \\ &= (\lambda^k)^{\mathrm{T}} y^k + \theta^k((\lambda^k)^{\mathrm{T}}\Delta y^k + (\Delta\lambda^k)^{\mathrm{T}} y^k) + (\theta^k)^2(\Delta\lambda^k)^{\mathrm{T}}\Delta y^k \end{aligned}$$

Multiplying the third equation in (13) by e^{T}, we get

$$(\lambda^k)^{\mathrm{T}}\Delta y^k + (\Delta\lambda^k)^{\mathrm{T}} y^k = \sigma\mu m - (\lambda^k)^{\mathrm{T}} y^k$$

and from (13) and (14),

$$\begin{aligned} (\Delta\lambda^k)^{\mathrm{T}}\Delta y^k &= (\Delta\lambda^k)^{\mathrm{T}}(\Lambda^k)^{-1}(-r_\sigma^k - Y^k\Delta\lambda^k) \\ &= (\Delta\lambda^k)^{\mathrm{T}}(\Lambda^k)^{-1}(-\Lambda^k Y^k e + \sigma\mu e - Y^k\lambda^k) \\ &= (\Delta\lambda^k)^{\mathrm{T}} Y^k e + \sigma\mu m - (\Delta\lambda^k)^{\mathrm{T}}(\Lambda^k)^{-1} Y^k\Delta\lambda^k \\ &= \sigma\mu m - 2(\Delta\lambda^k)^{\mathrm{T}}(\Lambda^k)^{-1} Y^k\Delta\lambda^k \le \sigma\mu m \end{aligned}$$

where the last inequality follows since y^k and λ^k are positive. Summing up

$$\gamma^{k+1} = (\lambda^k)^{\mathrm{T}} y^k + \theta(\sigma\mu m - \lambda^k)^{\mathrm{T}} y^k + (\theta^k)^2\sigma\mu m = (1 - (1 - \sigma + \sigma\theta^k)\theta^k)\gamma^k$$

Using the bound on θ^k, we get

$$\begin{aligned} \|r_d^k\| &\le (1-t)\|r_d^{k-1}\| \le \cdots \le (1-t)^k\|r_d^0\| \\ \|r_b^k\| &\le (1-t)\|r_b^{k-1}\| \le \cdots \le (1-t)^k\|r_b^0\| \\ \gamma^k &\le (1-\hat{t})\gamma^{k-1} \le \cdots \le (1-\hat{t})^k\gamma^0 \end{aligned}$$

□

To accelerate convergence, we use an idea taken from [12], where an algorithm for solving LPs is formulated. It is based on a predictor-corrector step. At each iteration, two steps are carried out, instead of just the one for (13): one predictor step which tries to reduce the complementarity by setting σ low and one corrector step which moves to the center of the domain by setting σ high. The reason for using a corrector step is that the next iterate will be at a more favorable position for the next iteration. We want $\mu \to 0$, however if $\lambda_i y_i = 0$ for some i, this might interfere with the progress in the next iteration.

The predictor step is computed by setting $\sigma = 0$ in (13),

$$r_\sigma = \Lambda Y e$$

The step length parameter θ_p is chosen such that $\lambda, y > 0$. A measure of the complementarity condition after the predictor step is

$$\mu_p = (\lambda + \theta_p\Delta\lambda)^T (y + \theta_p\Delta y)/m$$

If this value is low compared to μ, centering is hardly needed and hence σ should be set small. Vice versa, if μ_p is not low compared to μ, σ is set high, and a centering step is taken. For example the choice used in [12]

$$\sigma = \left(\frac{\mu_p}{\mu}\right)^3$$

has the desired effect.

A corrector step is obtained by using the information from the predictor step and expanding the expression for μ which gives $r_\sigma = \Delta Y \Delta\Lambda\, e$.

A complete step is obtained by combining the predictor, the corrector and the centering step and solve

$$\begin{bmatrix} B & -A^T & 0 \\ A & 0 & -I \\ 0 & Y & \Lambda \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \\ \Delta y \end{bmatrix} = \begin{bmatrix} -r_d \\ -r_b \\ -r_\sigma \end{bmatrix} \tag{16}$$

with $r_\sigma = -Y\Lambda\, e + \Delta Y \Delta\Lambda\, e + \sigma\mu e$.

The matrix in the linear system (16) can be reduced to a 2 by 2 block matrix C by eliminating Δy. The resulting system is called the augmented system,

$$\begin{bmatrix} -B & A^T \\ A & \Lambda^{-1}Y \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta\lambda \end{bmatrix} = \begin{bmatrix} r_d \\ -r_b - \Lambda^{-1} r_\sigma \end{bmatrix} \tag{17}$$

with $\Delta y = \Lambda^{-1}\left(-r_\sigma - Y\Delta\lambda\right)$. The system can be reduced one step further by eliminating $\Delta\lambda$,

$$\left(B + A^T \Lambda Y^{-1} A\right) \Delta x = -r_d + A^T \Lambda Y^{-1}\left(-r_b - \Lambda^{-1} r_\sigma\right) \tag{18}$$

with $\Delta\lambda = \Lambda Y^{-1}\left(-r_b - \Lambda^{-1} r_\sigma - A\Delta x\right)$. Although (18) is of smaller size than (17) and also positive definite, (17) may be preferable if sparsity is exploited. Adding the matrix $A^T\Lambda Y^{-1}A$ to the Hessian could destroy the sparse structure. The matrix in the augmented system, C, is symmetric and also quasi-definite [17]. A quasi-definite matrix T can be written as

$$T = \begin{bmatrix} -E & V^{\mathrm{T}} \\ V & F \end{bmatrix} \tag{19}$$

where E and F are positive definite matrices. A Cholesky factorization $T = LDL^{\mathrm{T}}$ could be used for this system since T is symmetric. The matrix L is lower triangular, and D is block-diagonal, with blocks of size 1 or 2. Since both E and D are positive definite, no pivoting is required for stability during the factorization phase. This means that we could do a static reordering before the factorization phase to reduce the fill-in in L.

2.3. **Inconsistent QP.** Even for well conditioned problems, the QP in (7) can be infeasible. This is due to the linearization of the constraints. A step can still be computed by solving the augmented QP

$$\begin{aligned} \min_{p,t} \quad & \frac{1}{2} p^{\mathrm{T}} B_k p + \nabla f_k p + M t^2 \\ \text{subject to} \quad & H_k p + h_k = 0 \\ & G_k p + g_k + te \geq 0 \end{aligned} \tag{20}$$

where $t \in \mathbf{R}$ is an additional variable, and M is a big number.

2.4. **Merit Function.** The SQP method formulated previously is only locally convergent. When the starting point is far from the solution, a merit function can be used. It should determine if a potential next iterate is acceptable given the objective function and the constraints. We use a non-differentiable l_1-penalty function. It is an exact penalty function, meaning that for a sufficiently large penalty parameter $\rho \geq \bar{\rho}$, an optimal point of the penalty function is equal to an optimal point of the constrained problem. The l_1-penalty function for the problem with both equality and inequality constraints is

$$\begin{aligned} \Phi_1(x,\rho) &= f(x) + \rho\left(\sum_{i=1}^{p}|h_i(x)| - \sum_{i=1}^{m}\min\{0, g_j(x)\}\right) \\ &= f(x) + \rho\left(\|h(x)\|_1 + \|g^+(x)\|_1\right) \end{aligned} \tag{21}$$

Although, the l_1- penalty function is not differentiable everywhere, it has a directional derivative, see, e.g., Boggs [1],

$$D(\Phi_1(x,\rho);d) = \nabla f(x)^{\mathrm{T}} d - \rho\left(\|h(x)\|_1 + \|g^+(x)\|_1\right)$$

An Armijo line search criterion together with backtracking is used to determine if the the next iterate is accepted. The line search may not fulfill Wolfe's condition, but it can be corrected by using a damped update of the quasi-Newton update (Damped BFGS), see [14] or [13].

Damped BFGS

Let $s_k = x_{k+1} - x_k$, $y_k = \nabla_x L(x_{k+1}, \lambda_{k+1}) - \nabla_x L(x_k, \lambda_{k+1})$
Set $r_k = \theta_k y_k + (1-\theta_k) B_k s_k$,
where

$$\theta_k = \begin{cases} 1 & \text{if } s_k^{\mathrm{T}} y_k \geq 0.2\, s_k^{\mathrm{T}} B_k s_k \\ (0.8\, s_k^{\mathrm{T}} B_k s_k)/(s_k^{\mathrm{T}} B_k s_k - s_k^{\mathrm{T}} y_k) & \text{if } s_k^{\mathrm{T}} y_k < 0.2\, s_k^{\mathrm{T}} B_k s_k \end{cases}$$

Update B_k using r_k instead of y_k.

A consequence of using an exact penalty function in a SQP method, is that steps that make good progress towards the solution may be rejected by the merit function. This phenomenon is called the Maratos effect, [11]. It can be avoided by using a second order correction step, [4], which reduces the constraint violation, or a watchdog technique, [3], which accepts some iterates to increase in the merit function. We use a second order correction step described in [13]. At each iteration, if the unit step p_k is not accepted, a correction w_k is added such that

$$A_k w_k + c(x_k + p_k) = 0 \tag{22}$$

where A is the Jacobian of the active constraints c. Since (22) is an under-determined system, we choose the minimum-norm solution.

2.5. **Large-Scale.** For problems with a large number of variables or constraints, solving the systems of equations could be time consuming. If the Hessian of the Lagrangian and the Jacobian of the constraints are sparse, a linear algebra solver which exploits the structure should be used. If the Hessian is not sparse, and the problem is large-scale, a low-storage approximation could be used. A popular method for solving large unconstrained problems is low-storage quasi-Newton methods, e.g., the limited memory BFGS (L-BFGS) [10]. Instead of forming the Hessian explicitly, a small number, r, of secant vectors are stored and used to update the approximation of the Hessian. However the SQP method needs an explicit expression for the Hessian approximation in (17).

It turns out that the L-BFGS update of the Hessian can be written in a compact form [2]. Let B_k be the approximation of the Hessian of a function f at iteration k. Assuming $k \geq r$, then

$$\begin{aligned} B_k =& B_0 + \begin{bmatrix} B_0 S_k & Y_k \end{bmatrix} \begin{bmatrix} S_k^T B_0 S_k & L_k \\ L_k^T & -D_k \end{bmatrix}^{-1} \begin{bmatrix} S_k^T B_0 \\ Y_k^T \end{bmatrix} \\ =& B_0 + M\, B^{-1} M^T \end{aligned} \tag{23}$$

where

$$\begin{aligned} S_k &= \begin{bmatrix} s_{k-r} & \dots & s_{k-1} \end{bmatrix}, \quad s_i = x_{i+1} - x_i \\ Y_k &= \begin{bmatrix} y_{k-r} & \dots & y_{k-1} \end{bmatrix}, \quad y_i = \nabla f(x_{i+1}) - \nabla f(x_i) \\ (L_k)_{ij} &= \begin{cases} s_{k-m+i-1}^T y_{k-m+j-1}, & i > j \\ 0 \end{cases} \end{aligned}$$

and B_0 is the initial Hessian approximation. Usually, $B_0 = \xi I$ with some positive constant ξ. The solution to the augmented system can be obtained by using the Sherman-Morrison-Woodbury formula for the inverse

$$\begin{aligned} C^{-1} =& \Big(\underbrace{\begin{bmatrix} -\xi I & A^T \\ A & \Lambda^{-1} Y \end{bmatrix}}_{=K} + \underbrace{\begin{bmatrix} M \\ 0 \end{bmatrix}}_{=U} \underbrace{\begin{bmatrix} B^{-1} M^T & 0 \end{bmatrix}}_{V^T} \Big)^{-1} \\ =& K^{-1} - K^{-1} U (I + V^T K^{-1} U)^{-1} V^T K^{-1} \end{aligned} \tag{24}$$

If $K = LDL^{\mathrm{T}}$ is available, the factorization of $(I + V^T K^{-1} U)$ is inexpensive, since it is a $2r \times 2r$ matrix. The main cost is factorizing K which is now sparse given sparse constraints. This idea is used in the interior-point algorithm found in [19].

For problems with very many inequality constraints the algorithm could be computationally expensive even with a sparse Hessian and sparse constraints. However, since the SQP method linearizes the problem and solves a QP at each iteration, only the near active constraints at the current iterate need to be chosen to be included in the QP instead of all constraints. This avoids a potentially expensive QP and reduces the number of evaluations of the constraints.

2.6. **Algorithm.** Summing up, we formulate the SQP algorithm below.

L-BFGS SQP

Set $\eta \in (0, 0.5)$. Choose initial point x_0.
Evaluate f_0, ∇f_0, g_0, G_0.
Initialize $B_0 = I$.
for $k = 0, 1, \ldots$
  if termination criterion fulfilled
    stop
  Compute p_k and $(\lambda_{k+1}, \mu_{k+1})$ by solving (7).
  Set ρ_k such that p_k is a descent direction for Φ_1.
  if $\Phi_1(x_k + p_k, \rho_k) \leq \Phi_1(x_k, \rho_k) + \eta D\Phi_1(x_k, \rho_k)$
    $x_{k+1} = x_k + p_k$
  else
    Compute second order correction w_k using (22).
    if $\Phi_1(x_k + p_k + w_k, \rho_k) \leq \Phi_1(x_k, \rho_k) + \eta D\Phi_1(x_k, \rho_k)$
      $x_{k+1} = x_k + p_k + w_k$
    else
      Set $\alpha_k = 1$.
      while $\Phi_1(x_k + \alpha_k p_k, \rho_k) > \Phi_1(x_k, \rho_k) + \eta\alpha_k D\Phi_1(x_k, \rho_k)$
        Set $\alpha_k = \alpha_k/2$
      end
      Set $x_{k+1} = x_k + \alpha_k p_k$
  Compute the Jacobian of the active constraints G_{k+1} at x_{k+1}.
  Add $s_k = \alpha_k p_k$ and $y_k = \nabla_x L(x_{k+1}, \lambda_{k+1}, \mu_{k+1}) - \nabla_x L(x_k, \lambda_{k+1}, \mu_{k+1})$
  to the damped L-BFGS representation. $B_k \to B_{k+1}$.
end

2.7. **Convergence.** For a full BFGS approximation with a damped update we can get superlinear convergence, see Powell [16]. For L-BFGS with a damped update we would expect linear convergence, as is the case for unconstrained problems. A proof of this can be constructed from Boggs [1] and Kelley [9].

For the remainder we make the following assumptions.

Assumption 1.

(a) The sequence $\{B_k\} \in \mathbf{R}^{n\times n}$ is uniformly positive definite. In particular there exists a β_1 such that $p^T B_k p \geq \beta_1 \|p\|^2$ for all k.
(b) The sequence $\{B_k\}$ is uniformly bounded, i.e. there exists a β_2 such that $\|B_k\| \leq \beta_2$.
(c) The sequence $\{B_k^{-1}\}$ exists and is uniformly bounded, i.e., there exists a β_3 such that $\|B_k^{-1}\| \leq \beta_3$.
(d) The constraints are of equality type $c_k \in \mathbf{R}^m$ with Jacobian $A_k \in \mathbf{R}^{n\times m}$.
(e) The Jacobian of the constraints A_k has linearly independent rows.

Under the assumptions presented, the solution to (4) with W_k replaced by an approximation B_k is

$$q_k = \left[A_k B_k^{-1} A_k^{\mathrm{T}}\right]^{-1} \left[c_k - A_k B_k^{-1} \nabla_x L(x_k, \lambda_k)\right] \tag{25}$$

and

$$p_k = -B_k^{-1} \nabla_x L(x_k, \lambda_{k+1}) \tag{26}$$

The optimal dual variables λ^* are given by

$$\lambda^* = -\left[A^* K (A^*)^{\mathrm{T}}\right]^{-1} A^* K \nabla f^* \tag{27}$$

for any nonsingular matrix K that is positive definite on the null space of A^*. After a step q_k, the dual iterate is

$$\begin{aligned} \lambda_{k+1} &= \left[A_k B_k^{-1} A_k^{\mathrm{T}}\right]^{-1} \left[c_k - A_k B_k^{-1} \nabla f_k\right] \\ &= T(x_k, B_k) \end{aligned} \tag{28}$$

With $K = B_k^{-1}$ in (27) gives

$$\lambda^* = T(x^*, B_k) \tag{29}$$

and therefore

$$\begin{aligned} \lambda_{k+1} - \lambda^* =& T(x_k, B_k) - T(x^*, B_k) \\ =& \left[A_k B_k^{-1} A_k^{\mathrm{T}}\right]^{-1} A_k B_k^{-1} \left(B_k - L^*\right)(x_k - x^*) + w_k \end{aligned} \tag{30}$$

where $w_k \leq \kappa \|x_k - x^*\|^2$ for a constant κ. For the convergence in x we have

$$\begin{aligned} x_{k+1} - x^* =& x_k - x^* - B_k^{-1} \left[\nabla_x L(x_k, \lambda_{k+1}) - \nabla_x L(x^*, \lambda^*)\right] \\ =& B_k^{-1} \left[(B_k - L^*)(x_k - x^*) - A_k^{\mathrm{T}}(\lambda_{k+1} - \lambda^*)\right] + O\left(\|x_k - x^*\|^2\right) \\ =& B_k^{-1} V_k (B_k - L^*)(x_k - x^*) + O\left(\|x_k - x^*\|^2\right) \end{aligned} \tag{31}$$

where

$$V_k = I - A_k^{\mathrm{T}} \left[A_k B_k^{-1} A_k^{\mathrm{T}}\right]^{-1} A_k B_k^{-1}$$

Defining the projection matrix

$$P_k = I - A_k^{\mathrm{T}} \left[A_k A_k^{\mathrm{T}}\right]^{-1} A_k$$

we have

$$V_k P_k = V_k$$

and hence

$$\|x_{k+1} - x^*\| \leq \|B_k^{-1}\| \|V_k\| \|P_k (B_k - L^*)(x_k - x^*)\| + O\left(\|x_k - x^*\|^2\right) \tag{32}$$

For a SQP method with full quasi-Newton updates, this can be used to prove super-linear convergence. For L-BFGS, we cannot expect super-linear convergence. However the update has the bounded deterioration property, [6], which can be used to prove linear convergence. The proof of the bounded deterioration property follows from [9] and [6].

Assumption 2. *The Hessian of the Lagrangian $L(x^*, \lambda^*)$ is positive definite.*

Theorem 2.2. *Let Assumptions 1 and 2 hold. Let $E_k = B_k^{-1} - \nabla^2_{xx} L^*$ denote the error of the Hessian approximation and assume $\|E_0\| \leq \delta_0$. Let $e_k = x_k - x^*$ and $h_k = \lambda_k - \lambda^*$. Assume that no damping is required, i.e., $\theta_k = 1$ for all k. Also, assume $\|x_0 - x^*\| < \delta_0$. Then*

$$\|E_{k+1}\| \leq \|E_k\| + \rho\left(\|e_k\| + \|e_{k+1}\| + \|h_{k+1}\|\right) \tag{33}$$

where ρ is a small constant. Furthermore there is a γ such that

$$\|E_k\| \leq \gamma \quad \forall k$$

and hence the method has linear local convergence.

Proof. By assumption 2, we can assume $\nabla^2_{xx} L^* = I$. Let us first consider the full BFGS when each update is used in the approximation. Remembering that $s_k = x_{k+1} - x_k$, $y_k = \nabla_x L(x_{k+1}, \lambda_{k+1}) - \nabla_x L(x_k, \lambda_{k+1})$, the BFGS update can be written as

$$B_{k+1}^{-1} = \left(I - \frac{s_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k}\right) B_k^{-1} \left(I - \frac{y_k s_k^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k}\right) + \frac{s_k s_k^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k} \tag{34}$$

First, y_k is

$$y_k = \int_0^1 \nabla^2_{xx} L(x_k + t s_k, \lambda_{k+1}) s_k \, dt = s_k + \int_0^1 \left(\nabla^2_{xx} L(x_k + t s_k, \lambda_{k+1}) - I\right) s_k \, dt = s_k + \Lambda_1 s_k \tag{35}$$

where

$$\Lambda_1 = \int_0^1 \left(\nabla^2_{xx} L(x_k + t s_k, \lambda_{k+1}) - I\right) \, dt \tag{36}$$

and $\|\Lambda_1\| \leq k_1(\|e_k\| + \|e_{k+1}\| + \|h_{k+1}\|)$, where $h_{k+1} = \lambda_{k+1} - \lambda^*$ and k_1 is a constant. Using this, we get

$$\frac{s_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k} = \frac{s_k s_k^{\mathrm{T}} + s_k \left(\Lambda_1 s_k\right)^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k + \left(\Lambda_1 s_k\right)^{\mathrm{T}} s_k} = \frac{s_k s_k^{\mathrm{T}}}{s_k^{\mathrm{T}} s_k} - F_k \tag{37}$$

where

$$F_k = \frac{s_k s_k^{\mathrm{T}} s_k^{\mathrm{T}} \Lambda_1 s_k / (s_k^{\mathrm{T}} s_k) - s_k s_k^{\mathrm{T}} \Lambda_1}{s_k^{\mathrm{T}} s_k + s_k^{\mathrm{T}} \Lambda_1 s_k}$$

and

$$\|F_k\| \leq m_1 \|\Lambda_1\| \leq m_1 k_1 \left(\|e_k\| + \|e_{k+1}\| + \|h_{k+1}\|\right)$$

for a constant m_1.

Furthermore

$$\frac{s_k s_k^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k} = \frac{s_k \left(y_k - \Lambda_1 s_k\right)^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k} = \frac{s_k y_k^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k} - G_k$$

where

$$G_k = \frac{s_k \left(\Lambda_1 s_k\right)^{\mathrm{T}}}{y_k^{\mathrm{T}} s_k}$$

and

$$\|G_k\| \leq m_2 \|\Lambda_1\| \leq m_2 k_1 \left(\|e_k\| + \|e_{k+1}\| + \|h_{k+1}\|\right)$$

with a constant m_2. With $w_k = s_k / \|s_k\|$, the error in the Hessian can be written as

$$\begin{aligned} E_{k+1} =& B_{k+1}^{-1} - I \\ =& \left(I - w_k w_k^{\mathrm{T}} + F_k\right) B_k^{-1} \left(I - w_k w_k^{\mathrm{T}} + F_k^{\mathrm{T}}\right) + w_k w_k^{\mathrm{T}} - F_k - G_k - I \\ =& \left(I - w_k w_k^{\mathrm{T}}\right) \left(E_k + I\right) \left(I - w_k w_k^{\mathrm{T}}\right) + w_k w_k^{\mathrm{T}} + H_k - I \\ =& \left(I - w_k w_k^{\mathrm{T}}\right) E_k \left(I - w_k w_k^{\mathrm{T}}\right) + H_k \end{aligned} \tag{38}$$

where

$$H_k = -F_k - G_k + F_k B_k^{-1} \left(I - w_k w_k^{\mathrm{T}} + F_k^{\mathrm{T}}\right) + \left(I - w_k w_k^{\mathrm{T}}\right) B_k^{-1} F_k^{\mathrm{T}}$$

We prove the theorem inductively. Assume $\|E_0\| \leq \delta_0$, $\|e_0\| \leq \delta_0$, $\|h_0\| \leq \delta_0$, where δ_0 is chosen small enough such that by (32) and (30),

$$\|e_1\| \leq \sigma \|e_0\|$$
$$\|h_1\| \leq \sigma \|e_0\|$$

with $0 \leq \sigma < 1$. Let also

$$\delta_0 \left(1 + \frac{\rho(1+2\sigma)}{1-\sigma}\right) < \gamma$$

We consider the first t iterations, where t is the number of pairs of secant vectors stored for the L-BFGS approximation. In this case the L-BFGS update is the same as the full BFGS, and we have

$$\|H_0\| \leq \rho\left(\|e_0\| + \|e_1\| + \|h_1\|\right)$$

for a constant ρ, and

$$\begin{aligned}\|E_1\| &\leq \|E_0\| + \rho\left(\|e_1\| + \|e_0\| + \|h_1\|\right) \\ &\leq \|E_0\| + \rho(1+2\sigma)\|e_0\| \\ &\leq \delta(1 + \rho(1+2\sigma)) < \gamma\end{aligned}$$

Now assume $\|E_{j-1}\| \leq \gamma$ and

$$\begin{aligned}\|e_j\| &\leq \sigma\|e_{j-1}\| \\ \|h_j\| &\leq \sigma\|e_{j-1}\| \quad \forall j \leq t\end{aligned}$$

Then

$$\|H_{j-1}\| \leq \rho\left(\|e_{j-1}\| + \|e_j\| + \|h_j\|\right)$$

and

$$\begin{aligned}\|E_j\| &\leq \|E_{j-1}\| + \rho(\|e_j\| + \|e_{j-1}\| + \|h_j\|) \\ &\leq \|E_{j-1}\| + \rho(1+2\sigma)\|e_{j-1}\| \\ &\leq \|E_{j-1}\| + \rho(1+2\sigma)\sigma^{j-1}\|e_0\| \\ &\leq \|E_{j-1}\| + \rho(1+2\sigma)\sigma^{j-1}\delta_0 \\ &\leq \|E_0\| + \delta_0\rho(1+2\sigma)\sum_{i=1}^{j}\sigma^{i-1} \\ &\leq \delta_0\left(1 + \rho(1+2\sigma)\frac{1-\sigma^j}{1-\sigma}\right) < \gamma\end{aligned}$$

Next we consider the case of $\|E_l\|$ when $l > t$. In this case, the Hessian approximation from L-BFGS is not equal to the full BFGS update. The L-BFGS updates an initial approximation B_0 at each iteration. We assume B_0 is constant for all iterations k. The approximation B_{k+1} is not only generated from the current B_k, but with secant pairs s_j, y_j which are generated by previous approximations B_j, $j \leq k$. This means that we have

$$\begin{aligned}&\|E_{l-j}\| \leq \gamma \\ &\|e_{l-j+1}\| \leq \sigma\|e_{l-j}\| \\ &\|h_{l-j+1}\| \leq \sigma\|e_{l-j}\|, \quad j = 1, \ldots, t-1\end{aligned}$$

If we assume that at each iteration, the initial approximation $B_0 = B_{l-t+1}$ is chosen such that

$$\|E_{l-t+1}\| \leq \delta_0$$

then the same analysis as for the first t iterations can be applied to prove that

$$\|E_l\| \leq \gamma$$

and

$$\|E_{l+1}\| \leq \|E_l\| + \rho\left(\|e_l\| + \|e_{l+1}\| + \|h_{l+1}\|\right)$$

□

Using the merit function, convergence from remote starting points can be shown, see, e.g., Boggs [1].

3. Numerical Results

The first set of test problems are taken from the Hock and Schittkowski collection of test problems [8]. The set contains nonlinear constrained test problems of smaller sizes.

The second set of test problems are taken from the application of airbag folding [5].

The performance of the L-BFGS approximation is compared to full BFGS. The tests should not be considered as real benchmarks for the software. The problems are small-scale, and the algorithm was primarily designed for large-scale problems. The full BFGS is expected to perform better, but the question if L-BFGS is comparable in speed. For larger problems, full BFGS, is not an option, due to the dense Hessian approximation. The performance of an SQP method with full BFGS approximation has been evaluated several times, see, e.g., Powell [16]. However, L-BFGS in the context of SQP, has not been thoroughly tested.

For all test problems, the optimization routine terminates if certain criteria are fulfilled. Feasibility, first order optimality, and descent is checked. When the minimum value is known, it is also checked.

Test 1. The test problems are taken from the Hock and Schittkowski collection [8]. The low-storage SQP method is compared to the same method with a full BFGS update and also compared to `fmincon` from the `MATLAB` Optimization Toolbox. The result is shown in Table 1.

Problem	n	m	L-BFGS SQP	BFGS SQP	`fmincon`
hs26	3	1	11	15	5
hs100	7	4	35	10	25
hs118	15	60	16	11	13

Table 1. The number of iterations required for the Hock and Schittkowski collection. n is number of variables, and m is the number of constraints (equality and inequality). The L-BFGS SQP used 5 secant vectors.

Test 2. The test problems are from the application of folding, see [5]. In short, we are given a crease pattern over a polyhedron, and we want to fold the polyhedron by minimizing the potential of the rotational springs. Constraints are set to conserve the edge lengths.

The crease pattern divides the the faces of the polyhedron into smaller polygons, called patches. Each patch has been triangulated. A rotational spring connected to each crease. Their potentials are computed using the scalar product of the (normed) normals n_i^1, n_i^2 of the two neighboring patches joined by a crease i, $i = 1, \ldots, n_c$. The coordinates of the mesh are $\{x^i\}_{i=1}^n$. Let x_i^1 and x_i^2 be the vertices of edge i, $i = 1, \ldots, n_e$. Then the optimization problem is:

$$
\begin{aligned}
\min_x f(x) &= \min_x \sum_{i=1}^{n_c} n_i^1 \cdot n_i^2 \\
\text{subject to} \quad & g_{2i-1}(x) = \|x_i^1 - x_i^2\|_2^2 \leq (1+\epsilon) l_i^2, \; i = 1, \ldots, n_e \\
& g_{2i}(x) = -\|x_i^1 - x_i^2\|_2^2 \leq -(1+\epsilon) l_i^2, \; i = 1, \ldots, n_e.
\end{aligned} \tag{39}
$$

where $\varepsilon > 0$ and l_i is the original length of edge i.

Three folding problems are tested, `fold3, fold4, fold5`. Table 2 shows a comparison of the algorithms.

The method used in the subroutine `fmincon` is similar to the reference method "BFGS SQP", but another merit function is used, see [15]. Instead of one penalty parameter a vector is used

Problem	n	m	L-BFGS 3 SQP	L-BFGS 5 SQP	L-BFGS 7 SQP	BFGS SQP	`fmincon`
fold3	30	64	31	29	28	20	10
fold4	48	104	25	10	10	7	11
fold5	66	144	50	47	61	39	64

TABLE 2. The number of iterations required for the two folding problems. n is number of variables, and m is the number of inequality constraints. The number after L-BFGS is the number of secant vectors.

with one component for each constraint. In practice, this is sometimes preferable, however there are no convergence proofs for this variant.

REFERENCES

[1] P. T. Boggs and J. W. Tolle, *Sequential quadratic programming*, Acta numerica, Cambridge Univ. Press, Cambridge, 1995, pp. 1–51.

[2] R. Byrd, R. Schnabel, and J. Nocedal, *Representations of Quasi-Newton matrices and their use in limited memory methods*, Math. Programming **63** (1994), 129–156.

[3] R. M. Chamberlain, M. J. D. Powell, C. Lemarechal, and H. C. Pedersen, *The watchdog technique for forcing convergence in algorithms for constrained optimization*, Math. Programming Stud. (1982), 1–17.

[4] T. F. Coleman and A. R. Conn, *Nonlinear programming via an exact penalty function: asymptotic analysis*, Math. Programming **24** (1982), 123–136.

[5] C. Cromvik, *Airbag folding based on optimization and origami*, Preprint, Chalmers University of Technology, 2007.

[6] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical methods for unconstrained optimization and nonlinear equations*, Classics in Applied Mathematics, vol. 16, Society for Industrial and Applied Mathematics (SIAM), 1996, Corrected reprint of the 1983 original.

[7] A. Griewank, *Evaluating derivatives*, Frontiers in Applied Mathematics, vol. 19, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2000, Principles and techniques of algorithmic differentiation.

[8] W. Hock and K. Schittkowski, *Test examples for nonlinear programming codes*, Lecture Notes in Economics and Mathematical Systems, vol. 187, Springer-Verlag, Berlin, 1981.

[9] C. T. Kelley, *Iterative methods for optimization*, Frontiers in Applied Mathematics, vol. 18, Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 1999.

[10] D.C. Liu and J. Nocedal, *On the limited memory method for large scale optimization*, Math. Programming **45** (1989), 503–528.

[11] N. Maratos, *Exact penalty function algorithms for finite-dimensional and control optimization problems*, Ph.D. thesis, University of London, 1978.

[12] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM J. Optim. **2** (1992), 575–601.

[13] J. Nocedal and S. J. Wright, *Numerical Optimization*, Springer Series in Operations Research, Springer-Verlag, New York, 1999.

[14] M. J. D. Powell, *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, Nonlinear programming, 3 (Proc. Sympos., Special Interest Group Math. Programming, Univ. Wisconsin, Madison, Wis.), vol. 3, Academic Press, 1978, pp. 27–63.

[15] ———, *A fast algorithm for nonlinearly constrained optimization calculations*, Numerical analysis (Proc. 7th Biennial Conf., Univ. Dundee, Dundee, 1977), Springer, Berlin, 1978, pp. 144–157. Lecture Notes in Math., Vol. 630.

[16] ———, *On the rate of convergence of variable metric algorithms for unconstrained optimization*, Proceedings of the International Congress of Mathematicians, Vol. 1, 2 (Warsaw, 1983) (Warsaw), PWN, 1984, pp. 1525–1539.

[17] R. J. Vanderbei, *Symmetric quasi-definite matrices*, SIAM J. Optim. **5** (1995), 100–113.

[18] ———, *Linear Programming*, second ed., International Series in Operations Research & Management Science, 37, Kluwer Academic Publishers, Boston, MA, 2001, Foundations and extensions.

[19] R. A. Waltz, J. L. Morales, J. Nocedal, and D. Orban, *An interior algorithm for nonlinear optimization that combines line search and trust region steps*, Math. Program. **107** (2006), no. 3, Ser. A, 391–408.

[20] R. B. Wilson, *A simplicial algorithm for concave programming*, Ph.D. thesis, Harvard University, 1963.

Department of Mathematical Sciences, Chalmers University of Technology, SE–412 96 Göteborg, Sweden

Department of Mathematical Sciences, Göteborg University, SE–412 96 Göteborg, Sweden
E-mail address: christoffer.cromvik@chalmers.se

AIRBAG FOLDING BASED ON ORIGAMI MATHEMATICS

CHRISTOFFER CROMVIK AND KENNETH ERIKSSON

Abstract. A new algorithm for folding three-dimensional airbags is presented. The method is based on Origami mathematics combined with nonlinear optimization.

The airbag is folded to fit into its compartment. Simulating an inflation therefore requires an accurate geometric representation of the folded airbag. However, the geometry is often specified in the inflated three-dimensional form, and finding a computer model of the folded airbag is a non-trivial task. The quality of a model is usually measured by the difference in area between the folded and the inflated airbags.

The method presented here starts by approximating the geometry of the inflated airbag by a quasi-cylindrical polyhedron. Origami mathematics is used to compute a crease pattern for folding the polyhedron flat. The crease pattern is computed with the intention of being fairly simple and to resemble the actual creases on the real airbag.

The computation of the crease pattern is followed by a computation of the folding. This is based on solving an optimization problem in which the optimum is a flat folded model. Finally, the flat airbag is further folded or rolled into its final shape (without using Origami).

The method has been successfully applied to various models of passenger airbags, providing more realistic geometric data for airbag inflation simulations.

1. Introduction

Simulating a crash when the crash test dummy hits the airbag while it is still expanding remains a challenge to the industry. This situation is called out-of-position (OOP), reflecting that the airbag was not designed for occupants that are sitting too close or for some other reason hit the airbag before it is fully inflated.

The difficulty with an OOP situation compared to an in-position situation is that the inflation of the folded airbag is much more important. It has to be realistically computed, since it affects the impact of the dummy. Attaining a realistic simulation means starting with a correct geometry of the folded airbag and simulating the inflation with correct gas dynamics. Several commercial software packages exist that can simulate the inflation process of an airbag, e.g., the explicit Finite Element (FE) code `LS-DYNA` [5].

This work aims at developing an algorithm for computing an accurate geometry of the flat folded airbag. Different airbags are folded by different methods and with different numbers and types of foldings. The airbags are often folded by both machines and humans according to a folding scheme. Still, the creases are not entirely deterministically positioned. It is very difficult to control the placement of smaller creases. The folding schemes all assume that the airbag lies flat and stretched in some direction. In this position, different foldings are executed until the dimension of the folded airbag is small enough so that it fits into the airbag compartment. The foldings can be a combination of simple folds, but also roll folds.

Some preprocessors to `LS-DYNA`, e.g., `EASi-FOLDER` [4] and `OASYS-PRIMER` [1] contain software for folding a (nearly) flat FE airbag mesh. They are capable of executing the type of foldings that are normally used in production on flat airbags, e.g., roll-fold, z-fold. However, they are not as accurate when folding an airbag in its three dimensional shape to a flat airbag.

Some airbag models have a simple construction, e.g., the driver model which is made of two circular layers sewn together. It is essentially two-dimensional. Passenger airbags are often more

complicated. They are made of several layers sewn together in a three dimensional shape, with no trivial two-dimensional representation. See Figure 1 for an example.

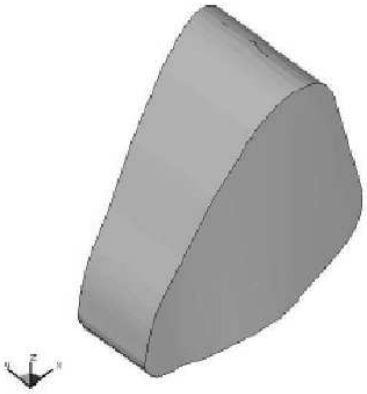

FIGURE 1. A CAD model of a passenger airbag.

In the present work, the computation of the geometry of the flat folded airbag is organized into two steps. First a crease pattern is computed on a polyhedral approximation of the airbag. Second, a nonlinear optimization problem is formed and solved for the purpose of finding the flat geometry. The accuracy of the computed approximation is measured by comparing its area to the area of the inflated model.

2. Crease Pattern

A crease pattern is first designed for a tetrahedron. We present a series of proofs for different types of polyhedra. The proofs are constructive, and their results can be used to design a crease pattern for our application.

Flat foldability, meaning that the polyhedron can be flattened using a fixed crease pattern, is achieved by cutting along the crease lines, folding the resulting object, and then gluing the cut-up faces back according to the correct connections.

Theorem 2.1. *The tetrahedron can be folded flat.*

Proof. The proof is organized in a sequence of figures shown in Figure 7, each visualizing the cutting and folding. Consider the tetrahedron with vertices A, B, C, D as in the figure. Cut up the triangle BCD of the tetrahedron, with straight cuts from a point E on the face, to the three vertices B, C, D, respectively, as in the figure.

Then open up the tetrahedron by rotating the triangular patches BDE, BCE, and CDE around the axes BD, BC, and CD, respectively, until these triangles become parts of the three planes through ABD, ABC, and ACD, respectively, as in the figure.

Cut the quadrilateral surface with vertices A, B, E', D along a straight cut from E' to A, and then rotate the resulting triangular faces ABE' and $AE'D$ around the axes AB and AD, respectively, until these faces become parts of the two planes ABC and ACD, respectively, as in the figure.

We choose the point E such that the edge BE' after rotation coinsides with BE'' and DE' with DE'''. The condition for this is that $\angle ABD + \angle DBE = \angle ABC + \angle CBE$ and $\angle ADB + \angle BDE = \angle ADC + \angle CDE$.

Using this, we may now (partly) restore the surface of the tetrahedron by joining the surfaces ABE'' and $ABE''C$ along the edge BE'', and the surfaces ADE''' and $ADE'''C$ along the edge DE'''.

Finally we rotate the (partly double layered) surface $ADE'''C$ around the axis AC until it coincides with the plane through A,B and C as in the figure. To conclude the proof of the flat foldability of the tetrahedron we now note that the point E''' after rotation coincides with E''. We may therefore now completely restore the topology of the original tetrahedron by joining the edges AE'' and AE''' (after rotation) and the edges CE'' and CE''' (after rotation). □

Note that the proof is based on cutting and gluing. It does not reveal if there is a continuous deformation to a flat shape.

Remark 2.1. *Concerning the line AE' we remark that the angles $\angle BAE'$ and $\angle DAE'$ satisfy $\angle BAE' + \angle DAE' = \angle BAD$ and $\angle BAC - \angle BAE' = \angle CAD - \angle DAE'$, as in the figure, and are thus independent of the plane BCD. We further note that we may also consider rotating the triangles BDE, BCE and CDE in the opposite direction, again until they become parts of the planes ABD, ABC and ACD, respectively, as in figure. We now choose the point E so that $\angle ABD - \angle DBE = \angle ABC - \angle CBE$ and $\angle ADB - \angle BDE = \angle ADC - \angle CDE$. Continuing from the figure we may then again make a straight cut from E' to A (partly double layered). Again, when we now rotate around the axes AB and AD as before the (rotated) point E' will coincide with E'' and E''' respectively, and we can partly restore the tetrahedron by joining along the edges. Finally, after rotation around AC we may completely restore the topology of the surface of the tetrahedron by joining along the edges. Concerning the crease line from A to E'' we note that again the angles $\angle BAE'$ and $\angle DAE'$ must satisfy the same equations $\angle BAE' + \angle DAE' = \angle BAD$ and $\angle BAC - \angle BAE' = \angle CAD - \angle DAE'$ as before and therefore must be the same as above. We therefore conclude that this crease line is independent of both direction of rotation of the triangles BCE, BDE and CDE, and of the position and orientation of the plane BCD (as long as the angles at A are unchanged).*

We now proceed by cutting the tetrahedron by a plane, see Figure 2. We call the cut-off tetrahedron a prism type polyhedron.

Theorem 2.2. *The prism type polyhedron can be folded flat.*

Proof. Consider a tetrahedron $ABCD$ with the crease pattern from the proof of Theorem 2.1. Cut the tetrahedron with a plane, see Figure 2. In the cut, insert two additional triangular surfaces, such that the two cutoff parts are closed, but not separated. The "smaller" cutoff part is a tetrahedron, and the "bigger" part is a prism type polyhedron. Let the vertices of the smaller tetrahedron be a, b, c, d, where $A = a$, b lies on the edge AB, c on AC and d on AD.

Remark 2.1 shows that the crease line from A to E', see Figure 7, is independent of how the inserted triangular face of the "smaller" tetrahedron is folded. Let it be folded to the interior of the "smaller" tetrahedron. This means that a crease pattern can be constructed which will coincide with the crease pattern of the original tetrahedron, i.e., the crease line which is constructed by drawing a straight line from a to e' will coincide with the crease line that was created from the line segment from A to E' in the proof of Theorem 2.1.

Now, make an identical copy of the crease pattern on the inserted triangular face belonging to the prism. Folding the original tetrahedron with its inserted triangular faces is possible by the construction of the crease pattern. Let the two polyhedra be separated by moving the tetrahedron in the plane. By the foldability of the tetrahedron, both the smaller tetrahedron and the prism can be folded flat. □

Next, we cut the prism type polyhedron by a plane, see Figure 3. We call the cut-off prism a box type polyhedron.

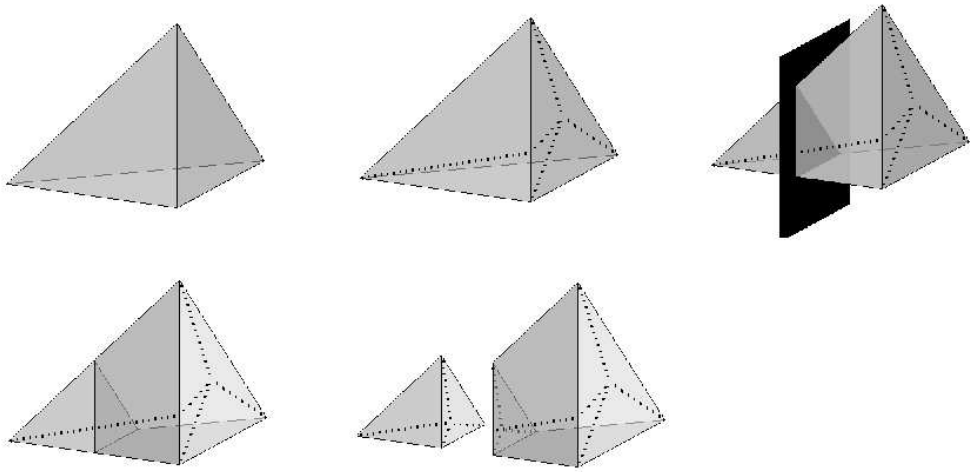

FIGURE 2. A tetrahedron is cut, and in the cut two additional interior triangular faces are created. Identical crease patterns are created on both interior faces, and the tetrahedron is separated into two parts: a smaller tetrahedron and a prism. The flat foldability of the prism follows from the foldability of the tetrahedron.

Theorem 2.3. *The box type polyhedron can be folded flat.*

Proof. Let the prism from the cut-off tetrahedron, with its crease pattern, be cut by a plane, see Figure 3. In the cut insert one additional quadrilateral surface which is only connected to the prism by its four vertices. Along the inserted surface put a crease line γ. Its position is only determined by the position of the upper and lower face of the prism. When the prism (with its cut) and the additional inserted surface are folded, there will be a gap along the sides of the prism, see Figure 4. Let the crease line on the side of the original prism be called ξ. Also, let the point where the crease γ meets ξ unfolded be called p_1, see Figure 4. The gap can be closed by forming two triangles: from a point p, see Figure 4, somewhere along ξ, to the intersection where ξ meets the inserted surface p_2, to B respectively C.

Note that the lengths Cp_1 and Cp_2 are the same, as well as the lengths Bp_1 and Bp_2, and the length Cp is shared by both the gap and the new triangles. Let C_1 and C_2 be positioned according to Figure 4. If the point p is chosen such that $\angle C_1Cp_1 + \angle p_2Cp = \angle C_1CC_2 + \angle C_2Cp$, then the new triangles are an identical match to the gap. By Theorem 2.2, the prism is foldable, so the full construction is foldable, and since the cut does not influence its foldability, and its gap is filled, therefore the box type polyhedron is flat foldable. □

In the proof of Theorem 2.3, a prism was cut off the polyhedron. The process of cutting off a prism can be repeated to create other types of polyhedra.

Definition 2.1. *A quasi-cylindrical polyhedron is a closed cut-off cylinder with a polygonal cross-section.*

Theorem 2.4. *Convex quasi-cylindrical polyhedra are flat foldable.*

Proof. This follows by the proof of Theorem 2.3. In each step, cut off a prism from the polyhedron, until the result forms the given shape.

□

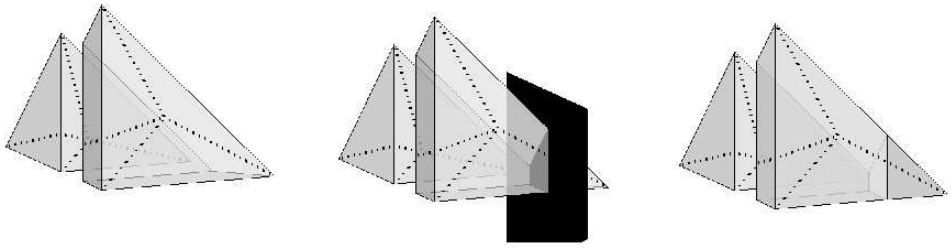

FIGURE 3. The prism from Figure 2 is cut, and in the cut, an additional interior quadrilateral surface is created. The flat foldability of the box type polyhedron follows from the foldability of the prism and the tetrahedron.

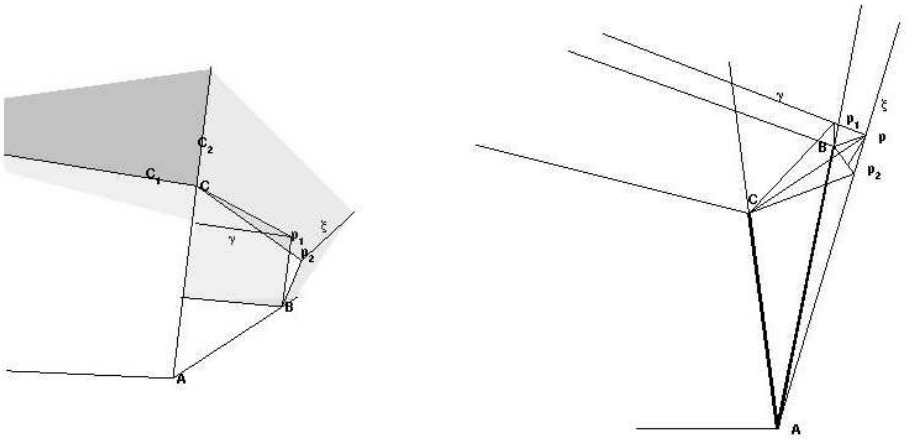

FIGURE 4. The left figure shows the gap around the inserted additional surface from the cut. The right figure shows the same object from above.

Airbags are usually quasi-cylindrical. There are cases, e.g. non-convex polyhedra, for which the technique for generating a crease pattern does not work. These situations might be avoided by slicing the polyhedron, and computing a crease pattern for each part.

Theorem 2.4 provides an algorithm for designing a crease pattern. Given a quasi-cylindrical polyhedron, we can extend it gradually using prisms until it reaches the shape of a tetrahedron. In each step, we apply the theory for flat foldability, creating a working crease pattern.

3. FOLDING

For airbags, there are various alternatives for simulating the folding process. This is specially due to the fact that the problem is artificial in the sense that the folding need not be realistic, e.g., there is no need to introduce the concept of time. The objective is to create a flat geometry which is physically correct, not to fold it in a realistic way.

Our algorithm for folding the polyhedron is based on solving an optimization problem. A program is formulated such that the optimal solution represents a flat geometry. The target

function, to be minimized, is a sum of rotational spring potentials, one spring over each crease. The minimal value of a spring potential is found when a fold is completed. The constraints are formulated in order to conserve a physically correct representation of the polyhedron, which means conserving the area and avoiding any self-intersections of the faces of the polyhedron.

The crease pattern over a polyhedron induces a subdivision of polygons called patches. In addition, the patches are triangulated, and the interior of the polyhedron is meshed with tetrahedra. Let the nodes of the mesh be $\{x^i\}_{i=1}^n$, and let the indices of the surface nodes be I_S. Let the tetrahedra be $\{K_i\}_{i=1}^{n_K}$ and set $I_K = \{1, \ldots, n_K\}$. Let the four indices of the nodes of tetrahedron k be $V_k(i)$, $i = 1, \ldots, 4$. The edges of the triangular faces are denoted $\{E_i\}_{i=1}^{n_E}$, and the indices of the two nodes of edge e are $W_e(i)$, $i = 1, 2$.

Denote the creases $\{C_i\}_{i=1}^{n_C}$. The spring potential over each crease C_i is computed using the scalar product of the normals, n_i^1,n_i^2, of the two neighbouring patches. The normals point outward from the polyhedron, and the scalar product is 1 when the two patches are parallel, and -1 when the fold is completed.

The folding process of a polyhedron with n nodes (surface and interior mesh nodes) is formulated as the following nonlinear program with $f : \mathbf{R}^{3n} \to \mathbf{R}$,

$$\min_x f(x)$$

$$\begin{aligned} f(x) &= f_1(x) + f_2(x) + f_3(x) \\ &= k_m \sum_{k=1}^{n_K} \left(\sum_{i=1}^{4} \sum_{j=i+1}^{4} \|x^{V_k(i)} - x^{V_k(j)}\| - d_{V_k(i),V_k(j)} \right)^2 \\ &\quad + \sum_{i=1}^{n_C} n_i^1 \cdot n_i^2 + k_p \sum_{i=1}^{n_E} \left(\|x^{W_i(1)} - x^{W_i(2)}\| - l_{W_i} \right)^2, \end{aligned}$$

subject to

$$\begin{aligned} \mathrm{vol}(K_i) &\geq \varepsilon_1, \qquad i = 1, \ldots, n_K, \\ \mathrm{dist}(x^i, K_j) &\geq \varepsilon_2, \qquad i \in I_S,\ j \in I_K \setminus p_i, \end{aligned}$$

where d_{ij} is the original distance between node x^i and x^j, l_i is the original length of edge i and k_m, k_p are penalty parameters. The first constraint function is $\mathrm{vol}(K_i)$ which is the signed volume of the tetrahedron K_i. The second constraint is $\mathrm{dist}(x^i, K_j)$, which is the distance from a surface node x^i to a tetrahedron K_j, and p_i are the tetrahedron indices connected to node x^i. Finally, ε_1 and ε_2 are small positive constants.

The target function f is composed of three parts. f_1 is a penalty function which strives to keep the tetrahedral mesh uniform. f_2 is the virtual spring potential which drives the folding. f_3 is a penalty function which keeps the edges of the triangles stiff. This is used to maintain the shape and surface area of the patches.

4. Numerical Example

In section 2, a theory for computing a crease pattern was discussed. To demonstrate its practical use, and also to demonstrate the folding algorithm, a numerical experiment is presented. From a CAD-drawing, an airbag shaped polyhedron was constructed. The surface area of the approximation differs about 0.5% to the original area. An in-house optimization solver was used to solve the optimization problem in section 3. It is a Fortran 90 implementation of a low-storage Quasi-Newton SQP method [6, 3, 2], that can handle a few thousand variables and constraints.

The crease pattern was generated by slicing off two upper "bumps", see Figure 5, from the airbag approximation. The crease pattern for these parts were computed separately from the rest of the polyhedron, and the complete crease pattern was formed by joining the parts.

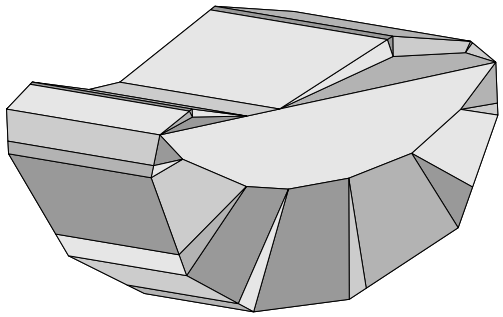

FIGURE 5. Polyhedral approximation of an airbag model together with a computed crease pattern.

The polyhedron approximation with its crease pattern was meshed using `TetGen` [7]. The visual result (solution) from the optimization progress is shown in Figure 6 for different iteration snapshots.

It was found that the surface area of the flat folded polyhedron was within 0.5% of the surface area of the unfolded polyhedron.

Acknowledgement. The authors would like to thank Prof S. Larsson, Dr B. Pipkorn and K. Mroz for valuable advice.

REFERENCES

[1] Arup, *Oasys-primer*, `http://www.arup.com`.
[2] D.P. Bertsekas, *Nonlinear programming*, Athena Scientific, 1999.
[3] R. Byrd, R. Schnabel, and J. Nocedal, *Representations of quasi-newton matrices and their use in limited memory methods*, Mathematical Programming **63** (1994), 129–156.
[4] ESI-group, *Easi-folder*, `http://www.esi-group.com`.
[5] Livermore Software Technology Corp., *Ls-dyna*, `http://www.lstc.com`.
[6] J. Nocedal and S. J. Wright, *Numerical optimization*, Springer-Verlag New York , Inc., 1999.
[7] Hang Si, *Tetgen a quality tetrahedral mesh generator and three dimensional Delaunay triangulator version 1.3 user's manual*, Technical report, Weierstrass-Institut für Angewandte Analysis und Stochastik, 2004.

DEPARTMENT OF MATHEMATICAL SCIENCES, CHALMERS UNIVERSITY OF TECHNOLOGY, SE–412 96 GÖTEBORG, SWEDEN

DEPARTMENT OF MATHEMATICAL SCIENCES, GÖTEBORG UNIVERSITY, SE–412 96 GÖTEBORG, SWEDEN
E-mail address: `christoffer.cromvik@chalmers.se`

DEPARTMENT OF MATHEMATICAL SCIENCES, CHALMERS UNIVERSITY OF TECHNOLOGY, SE–412 96 GÖTEBORG, SWEDEN

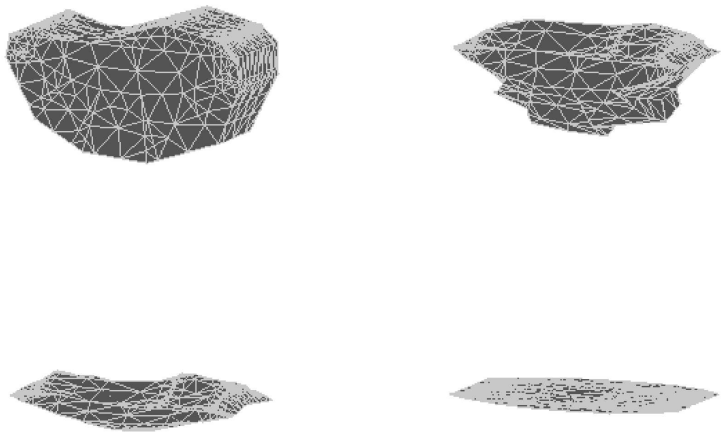

FIGURE 6. The figures show iteration snapshots from the folding of the polyhedron approximation from Figure 5. The upper left shows the unfolded polyhedron, the upper right: 40 iterations, the lower left: 60 iterations, and the lower right: 200 iterations.

DEPARTMENT OF MATHEMATICAL SCIENCES, GÖTEBORG UNIVERSITY, SE–412 96 GÖTEBORG, SWEDEN
E-mail address: kenneth.eriksson@hv.se

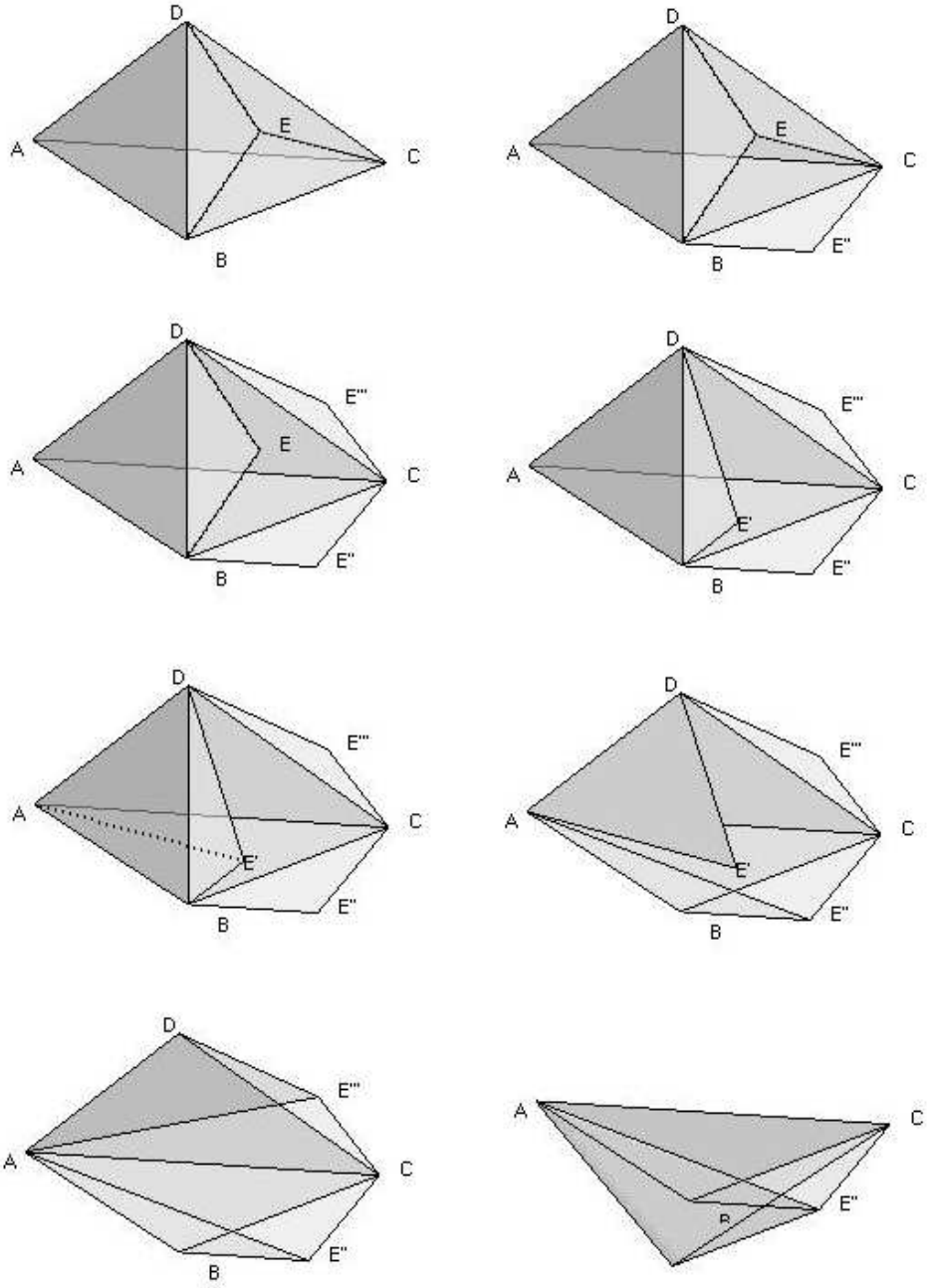

FIGURE 7. Supporting figure for the proof of Theorem 2.1. The proof follows the figures from left to right beginning at the top.