Structural optimization using sensitivity analysis and a level-set method, in Scilab and Matlab

Anton Mario Bongio Karrman, Caltech
Grégoire Allaire, Ecole Polytechnique

October 13, 2009

Abstract

A common problem in mechanical structure design is to optimize the shape and topology of an elastic structure given certain boundary conditions; the optimal structure usually performs some desired function and is lightweight but strong. Shape optimization is usually quite straightforward unlike topology optimization, which requires advanced techniques because topology variation usually causes discontinuous changes in the physical properties of structures. A shape and topology optimization algorithm developed in recent years uses the classical shape derivative paired with the level set method in order to find the optimal shape and topology of elastic structures given certain loads and conditions. In this project, using finite element analysis along with numerical discretization schemes that solve the Hamilton-Jacobi level set equation, we implement this algorithm by applying it to several test cases and present the code in Scilab.

1 Introduction

The purpose of the code presented in this paper is to make freely distributable a user-friendly two-dimensional topology optimization code using methods developed by Gregoire Allaire and his topology optimization team at the Center of Applied Mathematics at Ecole Polytechnique in 2002 [1].

The program comes with several test cases as well as the means to easily develop new problems by differing the boundary conditions, surface loads, volume forces, initialization, along with several other parameters. Optimization and improvement of the code is encouraged considering it is principally for educational use.

2 The topology optimization problem

2.1 The setting of the problem

For this project, the goal is to find an optimal linearly elastic structure. Optimal implies it is light but at the same time very strong and rigid; mathematically, our goal is to minimize the weight while maximizing the total amount of work the forces acting on the structure do at the microscale; this work is called the compliance.

We assume that our structure is composed of some linear isotropic elastic material that occupies the open bounded set $\Omega \subset \mathbb{R}^2$ in some working domain D that is a rectangular bounded open set containing all admissible shapes Ω. The boundary of Ω is $\partial\Omega = \Gamma_N \cup \Gamma_D$ where Γ_N and Γ_D have Neumann and Dirichlet boundary conditions, respectively. The vector-valued volume forces are given by f while the surface loads are g. The first step in the optimization step is to solve for u, the displacement field in the linearized elasticity system

$$\begin{cases} -\mathrm{div}(Ae(u)) = f & \text{in } \Omega \\ u = 0 & \text{on } \Gamma_D \\ (Ae(u))n = g & \text{on } \Gamma_N \end{cases} \tag{1}$$

where A is defined from Hooke's law by

$$A\xi = 2\mu\xi + \lambda(tr(\xi))\mathrm{Id} \tag{2}$$

and $e(u)$ is the strain tensor given by

$$e(u) = \frac{1}{2}(\nabla u + (\nabla u)^t) \tag{3}$$

For the objective function, we use the compliance, a classical choice in elastic structure optimization literature, in order to minimize the total work done by the structure:

$$J_1(\Omega) = \int_\Omega f \cdot u \, \mathrm{d}x + \int_{\Gamma_N} g \cdot u \, \mathrm{d}s = \int_\Omega Ae(u) \cdot e(u) \, \mathrm{d}x \tag{4}$$

For some fixed volume V, we define the set of all admissible shapes as

$$\mathcal{U}_{ad} = \{\Omega \subset D \text{ such that } |\Omega| = V\} \tag{5}$$

and our optimization problem becomes

$$\inf_{\Omega \in \mathcal{U}_{ad}} J_1(\Omega) \tag{6}$$

Referring the reader to [1], we know that this problem does not however have a solution, therefore we must impose other geometrical or topological constraints; in our program, we introduce control with volume and perimeter Lagrange multipliers, ℓ and ℓ' respectively, and end up with the well posed problem

$$\inf_{\Omega \in \mathcal{U}_{ad}} J_1(\Omega) + \ell|\Omega| + \ell'|\partial\Omega| \tag{7}$$

2.2 The shape derivative

We apply a gradient method in order to calculate (7) by using the classical shape derivative. For a much more detailed derivation we refer the reader to the work of Allaire et al. ([3] and [7]) and to Sokołowski's work on shape sensitivity analysis [8]. After simplification, we have

$$J'(\Omega)(\theta) = \int_{\Gamma_0} (\ell + \ell' H - Ae(u) \cdot e(u))\theta \cdot n \, \mathrm{d}s \tag{8}$$

where Γ_0 is the boundary that varies during the optimization process and H is the mean curvature of $\partial\Omega$ defined by $H = \mathrm{div}(n)$.

2.3 Shape representation using the level-set method

To track the boundary and shape of our structure, we represent it with a level set equation defined by:

$$\begin{cases} \psi(x) = 0 & x \in \partial\Omega \cap D \\ \psi(x) < 0 & x \in \Omega \\ \psi(x) > 0 & x \in (D \backslash \overline{\Omega}) \end{cases} \tag{9}$$

The level-set method provides convenient ways to evaluate key variables in our code; using finite differences because our mesh is uniformly rectangular, we can easily calculate the structure shape's normal $n = \nabla\psi/|\nabla\psi|$ as well as the curvature $H = \nabla \cdot n = \nabla \cdot (\psi/|\nabla\psi|)$.

For the optimization algorithm, we introduce the time variable t. Because our boundary at any time t is given by $\psi(t, x(t)) = 0$ for $x(t) \in \partial\Omega$, we get by derivation:

$$\frac{\partial\psi}{\partial t} + \dot{x}(t) \cdot \nabla\psi = \frac{\partial\psi}{\partial t} + Vn \cdot \nabla\psi = \frac{\partial\psi}{\partial t} + V|\nabla\psi| = 0 \tag{10}$$

After initializing ψ with some guess (usually holes spread throughout D), we use our shape derivative from earlier and let $V = -(\ell + \ell' H - Ae(u) \cdot e(u))$ to solve (10) a given number of times with a time step Δt small enough such that, after a given number of iterations, the objective function decreases. In our program, we use for the reinitialization and solution of ψ an upwind discretization scheme that is second order in space. For further details concerning the level-set method we refer the reader to the work of Sethian [5] and Osher [6].

2.4 The optimization algorithm

The steps for the actual algorithm are as follows:

1. The level-set function ψ is intialized with some initial guess.

2. Iterating for $k \geq 0$ until convergence to an optimal structure:

   (a) Solve for u_k and then use it to define the field V_k with which we solve the level-set equation.

   (b) Solve $\frac{\partial \psi}{\partial t} + V_k|\nabla \psi| = 0$ with Δt_k such that $J(\Omega_{k+1}) \leq J(\Omega_k)$.

3. While running some variable amount of iterations of 2.b per each finite element analysis, we minimize errors by reinitializing the level-set function to make sure that it does not get too flat or too steep at and near the border of the structure; this is done about every five iterations of 2.b and is especially important for finer meshes.

3 The implementation in Scilab and Matlab

The following sections describe our program and help to explain it in more depth.

3.1 The files

To run this algorithm, the user needs three principal files. The first, functions.sci, holds all of the functions used in the main program and therefore must be loaded before any of the other files. The second file to load is that which contains all the specific information describing our optimization problem. There are several example files of this type provided and each has a name corresponding to whatever problem it solves. For instance, one of these files is named bridge.sce; it contains all the parameters and specific settings needed in order to optimize a classical bridge shape. The final file to load is optalg.sce; this file is the actual optimization algorithm that uses both our specific setting file along with the needed functions in order to find our optimal structure.

3.2 The parameters and settings for specific problems

Most of the parameters and settings are self-explanatory, however a few need a bit of clarification.

For each distinct problem we first define our working domain size and resolution according to what would fit our problem. Although having a one to one aspect ratio for the resolution is not necessary, it makes the computation more accurate. Next we decide how many and what size holes with which we want to initialize our structure.

After this, we set our Lagrangian multipliers so that they give us an acceptable shape. If we want a lighter stucture, we increase $lagV = \ell$ whereas if we want a smaller perimeter, we increase $lagP = \ell'$, and vice versa.

Otherwise, there is a collection of miscellaneous variables that adjusts how the optimization process works. Sometimes we want to do many optimization iterations in order to be certain that our objective function converges, in which case we would set *entire* to a rather large value; however, doing this is often unnecessary. Also, depending on the setup of our problem, we can increase the variable *allow* so that the objective function has more local freedom to increase in case convergence to a globally optimal shape follows a rather significant topology change.

Our finite element matrices are defined globally as sparse matrices to improve computational and memory efficiency. In the code, we define the setting of the elasticity problem with the forces acting on the structure, the fixed parts of the structure, and the passive parts of the working domain.

We define the forces acting on the structure with the matrix $forceMatrix$. It works in the following manner: each row corresponds to a different force. Each force has four components: the first two are the fractions along the x and y axes of the working domain that give us the location of the force (the origin is the top left corner). The third element is a binary value that gives a force in the horizontal direction if it is 0 and in the vertical direction if it is 1. The fourth value is a scalar quantity that indicates the strength and direction of the force; negative values indicate the force is in the downward or leftward direction whereas positive values give forces in the upward or rightward direction. Examples of this can be found later in the paper when we present test cases.

We give parts of our working domain fixed Dirichlet conditions using the $fixeddofs$ matrix, which is broken down into a vector for our finite element routine. This vector lists all the pairs of x and y degrees of freedom for all the nodes and each corresponds to an element in the finite element mesh that spans our working domain according to Figure 1.

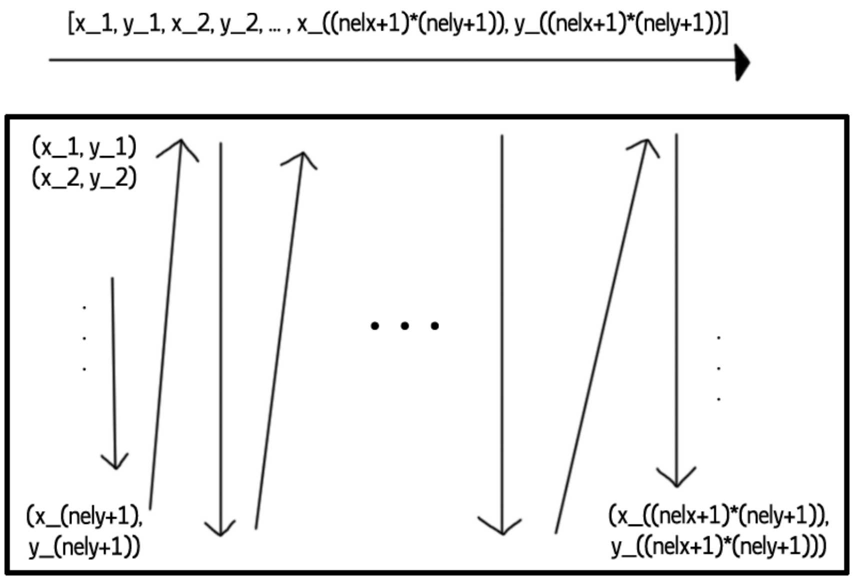


Figure 1: The way the finite element vector corresponds to the actual node locations with their degrees of freedom

Finally, we allow the user to define a passive part of our working domain. This is useful if we want to manipulate the rectangular shape of the domain or even if we want to just add certain unchanging holes or structure elements to our problem. An example of this is given with the L-beam test case.

3.3 Intialization and re-initialization

Initialization is an attempt to try and guess an optimal topology for our final structure, therefore it is often specific to the structure we are trying to optimize. This part of the algorithm takes place in the second file that gives the specific setting and parameters for the problem. Normally, we just set $\psi(x, y) = -.1$ for coordinates (x, y) that are included as part of our structure and $\psi = .1$ for holes that we do not want to include.

For accuracy, we must ensure that ψ is not too steep or too flat at its zero level-set; to do this, every five or so iterations while solving our convection equation we re-initialize our mesh by solving the following:

$$\begin{cases} \frac{\partial \psi}{\partial t} + \operatorname{sign}(\psi_0)(|\nabla\psi| - 1) = 0 & \text{in } D \times \mathbb{R}^+ \\ \psi(t = 0, x) = \psi_0(x) & \text{in } D \end{cases} \tag{11}$$

The function that solves for this signed distance function is referred to as $mesh00$ and takes the level-set function defined across the nodes as its argument along with $RIiter$, which is the number of times we solve (11). Initially we want to reinitialize the mesh to its limit, however later on, while solving the level-set equation for instance, it is only necessary to reinitialize with a few iterations because we only need accuracy on and near the boundary of our structure; hence $RIiterinit = 50$ whereas $RIiter = 10$.

3.4 Finite element analysis

In order to get our velocity field which we use to solve (10) we must use a finite element routine for our rectangular mesh. Our first step is to translate our level set function into a function that represents a material with a density of 1 for all values in Ω and some value eps for all values in $D\backslash\overline{\Omega}$. We then use a finite element routine borrowed from Ole Sigmund's Matlab shape optimization program [2]; taking advantage of Cholesky factorization of sparse matrices, we can solve for the displacement field (given by the vector U) and thus calculate a function across our finite elements that presents the unintegrated compliance; we translate this from $FEAeueu$ to $lvlAeueu$ because the latter fits the level-set mesh, which unlike the former, is defined at each node rather than at each element.

3.5 Using the compliance to solve the convection equation

Depending on which Lagrange multipliers we decide to include, we let $V = -(\ell + \ell' H - Ae(u) \cdot e(u))$ and then solve the Hamilton-Jacobi equation; it is during this part that we solve (11) every five iterations.

3.6 Checking the objective function

The goal of our optimization problem is to minimize our objective function, and it can be proven that if we are solving (10) with a V derived from shape derivative of the compliance, then the objective function will decrease as long as the time step is small enough. Therefore, after we do the finite element analysis and advect the level set, we analyze the new shape using the finite element routine; if the objective function decreases, we move on to the next main iteration but if not, we solve the level-set equation with fewer iterations ($HJiter$), which is similar to decreasing Δt_k except is faster but less accurate.

Ulimately, the objective function should reach a limit. We know that we have an optimal structure according to the specific initialization and parameters when the velocity function V equals 0 on the border of the structure.

4 Numerical examples

In this section we propose several test cases to better familiarize the reader with the program.

4.1 Short cantilever

This is a very basic case; the conditions imposed on our structure are described by Figure 2.

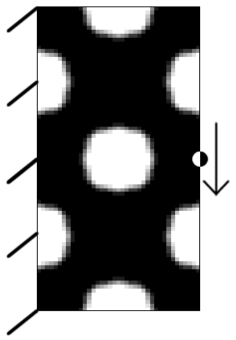


Figure 2: Initialization and boundary conditions for the short canitlever

Our working domain D is a 1 by 2 rectangle with Dirichlet conditions on the entire left border and a downward force applied in the middle of the right border. We define the force and Dirichlet conditions with (12) and (13), respectively.

$$forceMatrix = [1\ .5\ 1\ \text{-}1] \tag{12}$$

$$fixeddofs = [1,\ 2,\ \ldots,2(nely+1)-1,\ 2(nely+1)] \tag{13}$$

We initialize a mesh of 40 by 80 elements and run 20 iterations, noticing convergence quite early on during the optimization process:

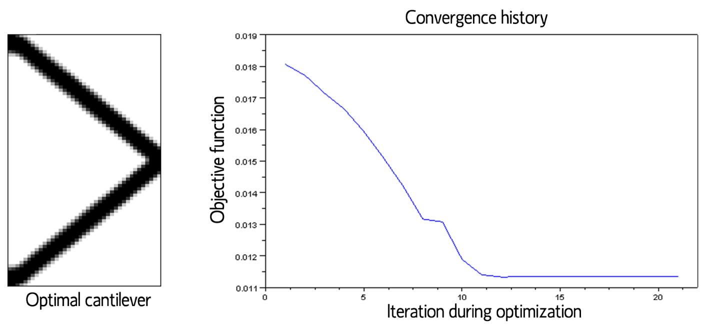


Figure 3: Convergence to the optimal cantilever

4.2 Long cantilever

This next example is the same problem as before except we switch the dimensions of our cantilever and use an intialization with more holes (see Figure 4).

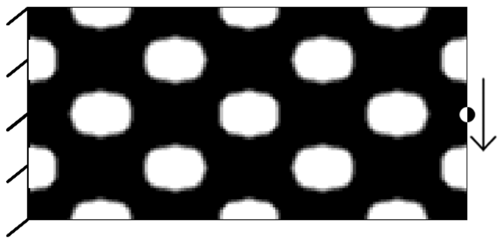


Figure 4: Initialization and boundary conditions for the long canitlever

We use 160 by 80 finite elements and a weight Lagrangian equal to .005. Figure 5 shows a shape that is a bit more intricate than that of the short cantilever.

4.3 Cantilever with distributed forces

For this example, rather than applying one force in the downward direction at the center of the right boundary, we distribute two forces: one in the upward direction in the upper right corner and one in the downward direction in the lower right corner. We use a mesh of 60 by 60 finite elements and a weight Lagrangian equal to .04. Our intial setup of the problem given by Figure 6 wheras our converged shape is given in Figure 7.

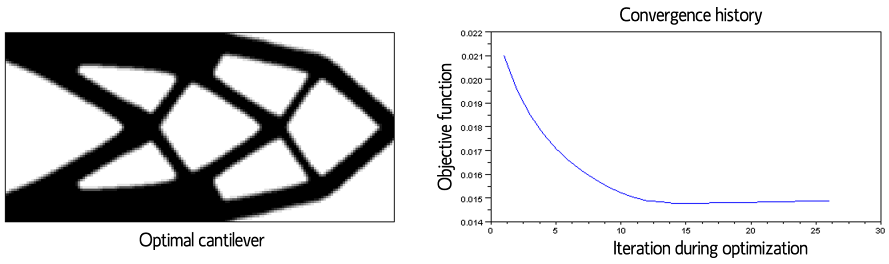


Figure 5: Convergence to the optimal cantilever

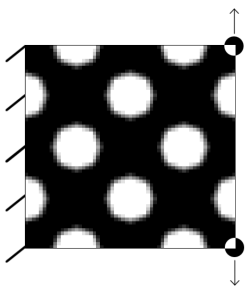


Figure 6: The intial setup of the cantilever with two distributed surface loads

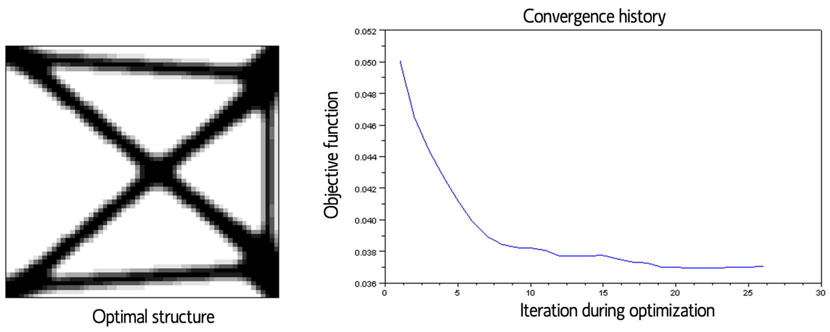


Figure 7: The optimal structure with two distributed surface loads

Table 1: More examples for the c function (the first three apply to the bridge)

c(0,1,0)	bottom left corner, x direction
c(0,1,1)	bottom left corner, y direction
c(1,1,1)	bottom right corner, y direction
c(0,0,0)	top left corner, x direction
c(.5,0,0)	half way across top border, x direction
c(1,1,0)	bottom right corner , x direction
c(1,.5,1)	half way across right border, y direction

4.4 Bridge

This example is much different: our force is applied midway across the bottom border and we fix the x and y freedom of the bottom left corner and the y freedom of the bottom y corner. The way we fix these nodes is by using the function named c (c stands for conditions because this function lets us easily find a freedom direction of a node given a fraction across the x and y axes, with the top left corner being the origin). So if we wanted for example to fix a node from moving in the y direction in the bottom left corner, we would take $c(0, 1, 1)$. The first two numbers indicate we have moved 0 percent across the x axis and 100 percent across the y axis, and since we started at the origin defined as the upper lefthand corner, this leaves us in the bottom lefthand corner. The third number 1 indicates we are fixing the y direction freedom of the node (0 would indicate we would want to fix the x direction freedom). So if we put multiple instances of $c(xfrac, yfrac, direction)$ in the $fixeddofs$ vector, the corresponding Dirichlet conditions are applied. Refer to Table 1 for an explanation via several more examples (the first two examples in the table correspond to the limits imposed on the bottom left corner of the bridge whereas the third gives the limits on the bottom right corner).

Our initialization is shown in Figure 8. However, as it is also an intial guess and we want a bottom base for our bridge, we initialize a strip at the bottom as part of our structure.

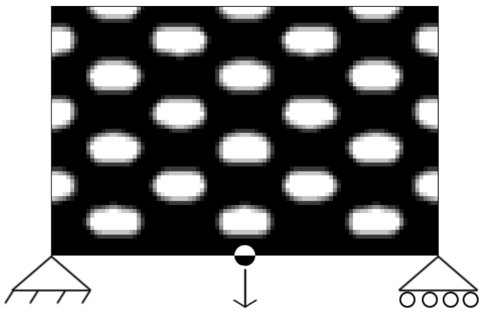


Figure 8: The intialization and boundary conditions for the bridge problem

With 100 by 60 finite elements in a working domain that is 2 by 1.2 (we make it a bit greater than 1.2 so that the top border of the working domain does not interfere with the structure as it makes a semi-circle), we run 35 iterations for the optimization of the bridge (Figure 9).

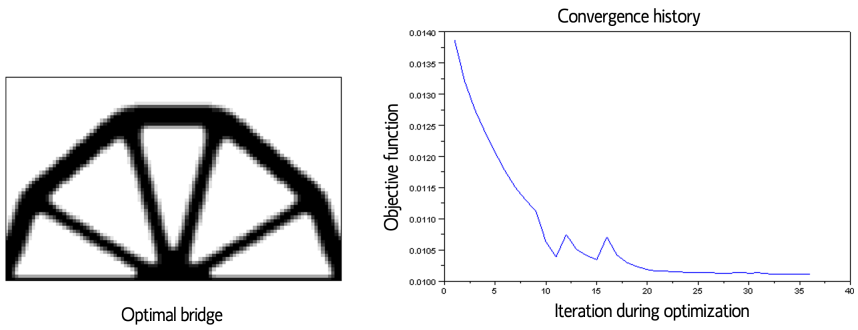


Figure 9: Convergence for the optimal bridge

4.5 L-beam

We initialize our domain and define the problem according to Figure 10.

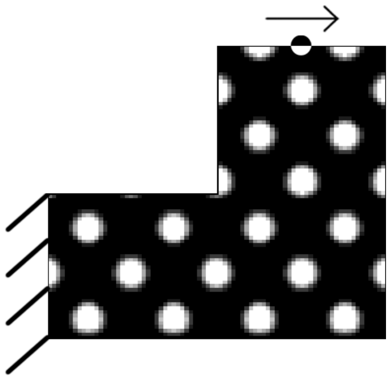


Figure 10: The initial setup for the L-beam

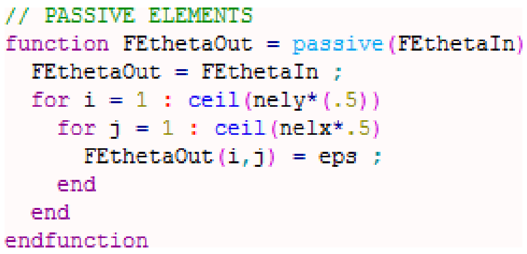
First, we use a section of code in the L beam.sce file that defines our passive function such that our working domain is changed from a square to an L shape (see Figure 11).

Then, we fix the bottom half of the left border with Dirichlet conditions:

$$fixeddofs = [(nely + 1),\ ...\ ,\ 2(nely + 1)] \tag{14}$$

Finally, we apply a rightward force 3/4 of the way across the top border with

$$forceMatrix = [.75\ 0\ 0\ 1] \tag{15}$$



```
// PASSIVE ELEMENTS
function FEthetaOut = passive(FEthetaIn)
  FEthetaOut = FEthetaIn ;
  for i = 1 : ceil(nely*(.5))
    for j = 1 : ceil(nelx*.5)
      FEthetaOut(i,j) = eps ;
    end
  end
endfunction
```

Figure 11: We change the working domain by setting a ”passive” region that is constantly empty with density equal to *eps*.

We run 35 iterations with weight and perimeter Lagrangians equal to .015 and .0001, respectively.

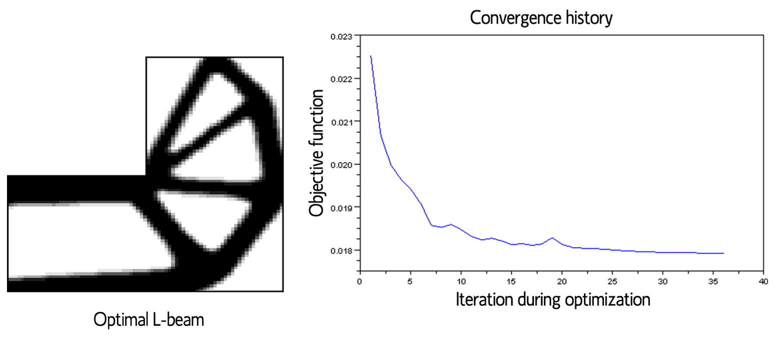


Figure 12: The convergence of the L-beam

5 Conclusion

Topology optimization methods used in the past include truss designs and homogenization, each with their own advantages and disadvantages. In comparison to such methods, defining our structure with a level set function and then pairing the level set method with the shape derivative of the compliance function proves to be advantageous for three main reasons:

1. it allows drastic topology changes throughout.
2. CPU cost is not very large.
3. with a good initialization, it is as efficient as the homogenization method (*Allaire et al.* [1]).

Future plans to improve the efficiency of this algorithm have already been successfully implemented by others and include the introduction of nucleation mechanisms. For our program, optimization of the finite element routine could improve computation time.

In order to generalize this algorithm, we could include the general adjoint state of u; this way, we could optimize the structure of compliant mechanisms. Also, a graphic interface could easily be added considering the fact that the conditions applied to the structure are quite easily implemented by using organized matrices and functions. Originally, the ultimate goal of this project was to develop the optimization algorithm in the free and easily distributable numeric computation package Scilab; however, we have made trivial changes to the code and present it in Matlab as well.

References

[1] Allaire G., Jouve F., Toader A.-M., Structural optimization using sensitivity analysis and a level set method, *J. Comp. Phys.*, Vol 194/1, 363–393 (2004).

[2] Sigmund O., A 99 line topology optimization code in Matlab, *Struct. Multidisc. Optim.*, Vol 21, pp.120-127 (2001).

[3] Dousset C., Allaire G. Calcul de formes optimales, *Rapport de Stage*, (2005).

[4] Osher S., Fedkiw R., *Level set methods and dynamic implicit surfaces,* Applied Mathematical Sciences, 153, Springer-Verlag, New York (2003).

[5] Sethian J.A., *Level-Set Methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision and materials science,* Cambridge University Press (1999).

[6] Osher S., Fedkiw R., *Level set methods and dynamic implicit surfaces,* Applied Mathematical Sciences, 153, Springer-Verlag, New York (2003).

[7] G. Allaire, *Conception optimale de structures,* Collection: Mathmatiques et Applications, Vol. 58, Springer (2007).

[8] Sokołowski J., Zolesio J.P., *Introduction to shape optimization: shape sensitivity analysis, Springer Series in Computational Mathematics, Vol. 10, Springer, Berlin,* (1992).

[9] Scilab, a scientific software developed by INRIA and ENPC, freely donwloadable at
`http://www.scilab.org`