GPTO: Topology Optimization using Geometry Projection

Hollis Smith and Julián A. Norato
Department of Mechanical Engineering
University of Connecticut
Version 1.0.1, August, 2021
Version 1.0.0, December, 2019

Abstract

This document provides instructions to install and run an example of GPTO, a Matlab code for the topology optimization of 2D and 3D structures with bars using geometry projection. The document also provides an explanation of the options in the master input file.

The objective of this manual is to provide instructions on how to prepare input files for a run of GPTO by modifying the input files of the default example problem that comes packaged with the code. For a detailed description of the geometry projection formulation, as well as of the capabilities of the code, please refer to the accompanying journal paper.

Contents

1 Installation

1.1 What's new

Version 1.0.1

- The Matlab implementation of the method of moving asymptotes (MMA) [1, 2, 3] is now included in the distribution of GPTO, thanks to Prof. Svanberg kindly making it freely available via a GNU General Public License version 3. Note that we have replaced two lines of `subsolv.m` that improve efficiency.

1.2 License

This program is freely distributed under a Creative Commons CC-BY-NC 4.0 license, which means it is free for non-commercial use and that appropriate credit must be given. The program is not open source in the sense that a repository to which users can contribute to further development of the code is not available. Nevertheless, the authors welcome any suggestions users may have for future improvements.

This software is provided by the contributors "as-is" with no explicit or implied warranty of any kind. In no event shall the University of Connecticut or the contributors be held liable for damages incurred by the use of this software.

Matlab is registered trademark of The MathWorks, Inc.

NVIDIA is a registered trademark of NVIDIA Corporation in the U.S. and/or other countries.

Gmsh© [4] is open source software licensed under the GNU General Public License (GPL).

VTK is an open-source toolkit licensed under the BSD license.

As mentioned in the accompanying journal paper, the finite element analysis follows closely the code and efficient techniques presented in [5].

1.3 System requirements

The program requires Matlab to be installed in the system. If you are planning to use Matlab's `fmincon`, then the installation must have the Optimization Toolbox. if you are planning to use GPUs for the solution of the system of linear equations, the installation must have the Parallel Computing Toolbox installed.

This code has been tested using versions R2018b and R2019b of Matlab, and in the Red Hat Enterprise Linux 7.4, Ubuntu 19.10, macOS Mojave and Catalina, and Windows 10 operative systems.

To use GPUs to solve the finite element analysis, the system must have an NVIDIA GPGPU card.

Approximately 40 Mb of disk space are required to install the program.

1.4 Installation and testing

To install the code, simply download it and uncompress it, making sure the folder structure is preserved. To test it, change the working folder in MATLAB to the root folder where `GPTO.m` is located in your installation and run it.

To perform the optimization using the MATLAB code for MMA, you must copy the files `mmasub.m`, `subsolv.m` and `kktcheck.m` in the `optimization` folder of the GPTO installation.

The default problem uses MATLAB's `fmincon` routine as optimizer, and so it should run even if you do not have MMA. This problem consists of the design of a 2D cantilever beam with eight bars in the initial design. If the code runs successfully, it should converge and produce a convergence plot at the end of the optimization.

2 Tutorial – 2D MBB beam

As described in the accompanying journal paper, the program has quite a few input parameters related to mesh generation, boundary conditions, finite element analysis, geometry projection, optimization and output. All of these parameters must be specified as inputs in several MATLAB input files. Therefore, the easiest way to setup a new problem is to copy the input files for one of the examples available in the installation and modifying them where necessary. This is the approach we follow in this tutorial, where we copy the input files for the default 2D cantilever beam problem to create a new problem, which consists of the extensively studied 2D Messerschmitt-Bölkow-Blohm (MBB) beam.

The dimensions, displacement boundary conditions (BCs) and load for the MBB problem are shown in Fig. 1. The magnitude of the load is $F = 0.1$. The problem is symmetric with respect to the centerline (only the right-hand side is shown in the figure). The volume fraction constraint for this problem is $v_f^* = 0.45$.

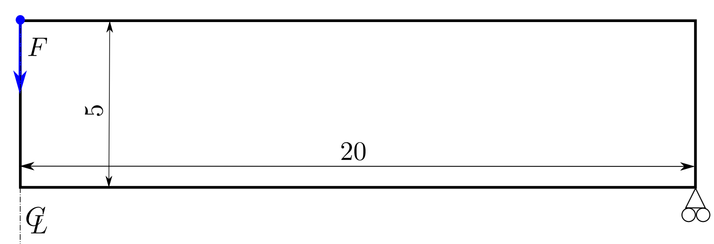

Figure 1: MBB beam problem

2.1 Copying the input files

The first step in the tutorial is to copy the input files for the default 2D cantilever problem to another folder. You can copy these files anywhere, but a suggestion to keep different models organized is to create a folder inside the `/input_files` folder. **1)** For this tutorial, create a folder named `mbb2d` inside the `/input_files` folder. **2)** Next, copy the following files:

inputs_cantilever2d.m
initial_cantilever2d_geometry.m
setup_bcs_cantilever2d.m

from the folder /input_files/cantilever2d to the /input_files/mbb2d folder, and **3)** rename them as

inputs_mbbd2.m
initial_mbbd2_geometry.m
setup_bcs_mbbd2.m

Note that you can really change these names to anything you want (since you anyway have to modify these names in the input files as well), but for simplicity we will follow the naming strategy we suggested above. These three files correspond to the master input file, the description of bars in the initial design, and the boundary conditions for the analysis, respectively.

2.2 Renaming the functions

For each one of the three files you copied, you have to modify the name of the corresponding MATLAB function in the first line so that the name of the function matches the name of the file (this is a MATLAB requirement, so that it knows in what file to find the function when invoked somewhere else).

- In inputs_mbb2d.m, change
  function inputs_cantilever2d()
  to
  function inputs_mbb2d()
- In initial_mbb2d_geometry.m, change
  function inputs_cantilever2d_geometry()
  to
  function initial_mbb2d_geometry()
- In setup_bcs_mbb2d.m, change
  function setup_bcs_cantilever2d()
  to
  function setup_bcs_mbb2d()

2.3 Invoking the master input file from the main script

Next, we need to make sure the main code invokes the master input file for this example. To do this, remove (or comment out) the line

run('input_files/cantilever2d/inputs_cantilever2d.m')

in the file get_inputs.m in the root folder of the installation and add the line

run('input_files/mbb2d/inputs_mbb2d.m').

2.4 Modifying the master input file

We will now modify the master input file inputs_mbb2d.m. For this problem, as with the default cantilever problem, we will let the code automatically generate the mesh.

1) To update the dimensions of the design region and use a mesh with square elements of size 1, change the following lines:

```
FE.mesh_input.box_dimensions = [20 5];
FE.mesh_input.elements_per_side = [200 50];
```

2) We now need to tell the master input file the location of the file with the BCs for this problem (which we will modify later):

```
FE.mesh_input.bcs_file = ...
'input_files/mbb2d/setup_bcs_mbb2d.m';
```

Note that the line breaks above are to used to fit the column width of this manual and are not needed in the MATLAB code.

3) We also need to indicate in the master input file the location of the file with the specification of the bars that make up the initial design (which we will also modify later):

```
GEOM.initial_design.path = ...
'input_files/mbb2d/initial_mbb2d_geometry.m';
```

4) For this problem, we will use bars of a fixed width $r_b = 0.25$. We will also use the MMA optimizer. As detailed in the accompanying journal paper, in this case we need to assign lower and upper bounds to the bar radii that are slightly different:

```
GEOM.min_bar_radius = 0.2499;
GEOM.max_bar_radius = 0.2501;
```

5) To impose the volume fraction constraint of $v_f^* = 0.45$, modify the following line:

```
OPT.functions.constraint_limit = [0.45];
```

6) For this problem we will use the MMA optimizer; if you do not have it, you can leave the optimizer as `fmincon`. In that case, you can set both `GEOM.min_bar_radius` and `GEOM.max_bar_radius` to 0.25. To switch to the MMA optimizer, modify the line:

```
OPT.options.optimizer = 'mma';
```

7) Finally, we will use a looser stopping criterion for the relative change in the design variables:

```
OPT.options.step_tol = 1e-2
```

2.5 Specifying boundary conditions

The displacement boundary conditions and loading are specified in `input_files/mbb2d/setup_bcs_mbb2d.m`.

1) The first step is to retrieve the node sets at the locations where we have supports and the load, namely the left-top corner, the bottom-right corner, and the left edge. The program has a convenient routine to retrieve these node sets for cuboid- and rectangular-shaped design regions. Once retrieved, these sets are stored in the global `FE` structure. Replace/ add the corresponding lines in the script:

```
compute_predefined_node_sets({'TL_pt', 'BR_pt', 'L_edge'});
TL_pt = FE.node_set.TL_pt;
BR_pt  = FE.node_set.BR_pt;
L_edge = FE.node_set.L_edge;
```

2) We start by assigning the applied force. The direction of the load is 2 (vertical) and the net magnitude is given a negative sign to indicate it is applied downwards. This load is applied on the top-left corner. Replace/ add the following lines after the previous section:

```
net_mag = -0.1;
load_dir = 2;
load_region = TL_pt;
load_mag = net_mag/length(load_region);
```

The code then builds an array (`load_mat`) with three columns: the first column has the ID of the node where the load is applied; the second has the direction, and the third has the magnitude. Do not modify the code that builds this array.

3) The displacement BCs are slightly more complicated than those in the cantilever beam problem, because we have two regions (the left edge and the bottom-right corner) that have different BCs. The simplest strategy is to define the these two BCs separately and subsequently combine them. To do this, replace/ add the following lines in the script:

```
% Symmetry boundary condition on left-hand side edge
disp_region1 = L_edge;
disp_dirs1 = ones(1, length(disp_region1));
disp_mag1 = zeros(1, length(disp_region1));
% Vertical roller on bottom-right point
disp_region2 = BR_pt;
disp_dirs2 = [2];
disp_mag2 = [0];
% Combine displacement BC regions
disp_region = [disp_region1 disp_region2];
disp_dirs = [disp_dirs1 disp_dirs2];
disp_mag = [disp_mag1 disp_mag2];
```

Note that the symmetry BC on the left edge corresponds to a zero displacement in the 1-direction, while the roller on the bottom-right corner corresponds to a zero displacement in the 2-direction.

As before, the code builds an array (`disp_mat`) similar to that for the loads. Do not modify the code that builds this array.

2.6 Specifying the initial design

For this problem, we will use the initial design shown in Fig. 2, which consists of 32 bars. In this problem, the bars are 'floating', i.e., they do not share endpoints, and therefore there are 62 separate endpoints. The size variable α_b for each bar in the initial design is set to 0.5, and the bar radius r_b to 0.25.

1) Replace the point matrix in `initial_mbb2d_geometry.m` with:

```
point_matrix = ...
[
1   0.25 4.75
2   4.75 4.75
3   0.25 2.75
4   4.75 4.75
5   0.25 4.75
6   4.75 2.75
7   0.25 2.75
8   4.75 2.75
9   0.25 2.25
10  4.75 2.25
11  0.25 0.25
12  4.75 2.25
13  0.25 2.25
14  4.75 0.25
15  0.25 0.25
16  4.75 0.25
17  5.25 4.75
18  9.75 4.75
19  5.25 2.75
20  9.75 4.75
21  5.25 4.75
22  9.75 2.75
23  5.25 2.75
24  9.75 2.75
25  5.25 2.25
26  9.75 2.25
27  5.25 0.25
28  9.75 2.25
29  5.25 2.25
30  9.75 0.25
31  5.25 0.25
32  9.75 0.25
33  10.25 4.75
34  14.75 4.75
35  10.25 2.75
36  14.75 4.75
37  10.25 4.75
38  14.75 2.75
39  10.25 2.75
40  14.75 2.75
41  10.25 2.25
42  14.75 2.25
43  10.25 0.25
44  14.75 2.25
45  10.25 2.25
46  14.75 0.25
47  10.25 0.25
48  14.75 0.25
49  15.25 4.75
50  19.75 4.75
```

```
51  15.25 2.75
52  19.75 4.75
53  15.25 4.75
54  19.75 2.75
55  15.25 2.75
56  19.75 2.75
57  15.25 2.25
58  19.75 2.25
59  15.25 0.25
60  19.75 2.25
61  15.25 2.25
62  19.75 0.25
63  15.25 0.25
64  19.75 0.25
];
```

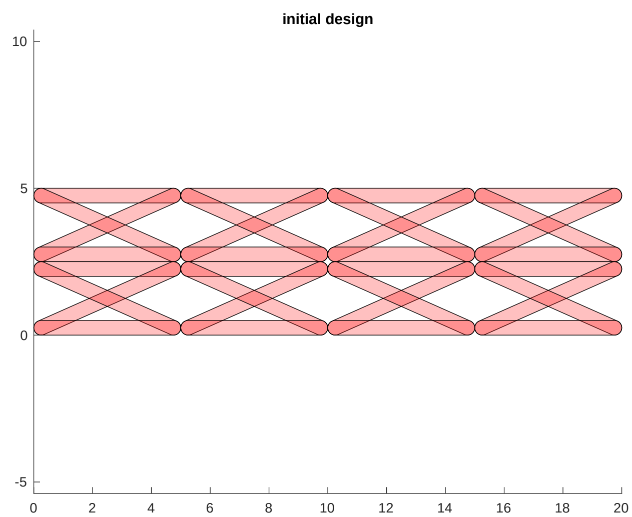

Figure 2: Initial design for MBB beam.

The first column in the array contains an integer index for each point, and the second and third columns correspond to the x- and y-coordinates.

2) Now replace the bar matrix:

```
bar_matrix = ...
[1 , 1 , 2 , 0.5, 0.25
2 , 3 , 4 , 0.5, 0.25
3 , 5 , 6 , 0.5, 0.25
4 , 7 , 8 , 0.5, 0.25
5 , 9 , 10, 0.5, 0.25
6 , 11, 12, 0.5, 0.25
```

```
7 , 13, 14, 0.5, 0.25
8 , 15, 16, 0.5, 0.25
9 , 17 , 18 , 0.5, 0.25
10 , 19 , 20 , 0.5, 0.25
11 , 21 , 22 , 0.5, 0.25
12 , 23 , 24 , 0.5, 0.25
13 , 25 , 26, 0.5, 0.25
14 , 27, 28, 0.5, 0.25
15 , 29, 30, 0.5, 0.25
16 , 31, 32, 0.5, 0.25
17 , 33 , 34 , 0.5, 0.25
18 , 35 , 36 , 0.5, 0.25
19 , 37 , 38 , 0.5, 0.25
20 , 39 , 40 , 0.5, 0.25
21 , 41 , 42, 0.5, 0.25
22 , 43, 44, 0.5, 0.25
23 , 45, 46, 0.5, 0.25
24 , 47, 48, 0.5, 0.25
25 , 49 , 50 , 0.5, 0.25
26 , 51 , 52 , 0.5, 0.25
27 , 53 , 54 , 0.5, 0.25
28 , 55 , 56 , 0.5, 0.25
29 , 57 , 58, 0.5, 0.25
30 , 59, 60, 0.5, 0.25
31 , 61, 62, 0.5, 0.25
32 , 63, 64, 0.5, 0.25];
```

The first column contains an integer index for each bar; the second and third columns are the integer IDs of the endpoints of the medial axis; the fourth column is the value of the size variable of the bar in the initial design; and the fifth column is the value of the width of the bar in the initial design.

Note that, as detailed in the accompanying journal paper, it is important to make sure that the length of the bars in the initial design is not zero, which is simply accomplished by making sure that the coordinates of the two endpoints of the axis of the bar do not coincide.

2.7 Running the optimization

To run the optimization, simply execute the `GPTO.m` script. Upon convergence, the design may look similar to the one shown in Fig. 3. The actual design you obtain may be different depending on the type and version of the optimizer (including the version of MATLAB), and it may take a different number of iterations to converge.

3 Parameters in the master input file

This section provides a listing of the meaning and options of all the parameters in the master input file. The equation numbers indicated for some parameters correspond to the most relevant expressions in the accompanying journal paper.

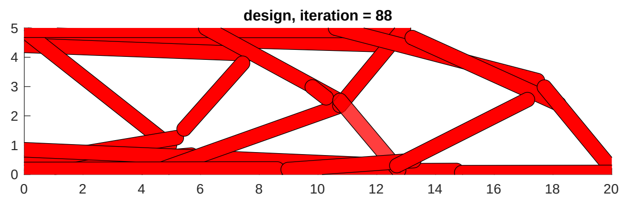

Figure 3: Optimal design for MBB beam.

3.1 Plotting

`plot_cond` : Options: `true` or `false`
Turns on or off plotting of figures during the optimization.

3.2 Finite element mesh

`FE.mesh_input.type` : Options: `'generate'`, `'read-home-made'`, `'read-gmsh'`
Determines the type of mesh:
`'generate'`: the code automatically generates a mesh for rectangular (2D) and cuboid (3D) regions. If this option is chosen, the option `FE.mesh_input.elements_per_side` must be specified.
`'read-home-made'`: loads a mesh from a Matlab .mat file previously created using the `makemesh` function available in the code. If this option is chosen, the option `FE.mesh_input.mesh_filename` must be specified.
`'read-gmsh'`: Read a quadrilateral or hexahedral mesh generated by Gmsh and exported to the Matlab format. For this code, we tested version 4.0.6 of Gmsh (but it is possible that earlier versions work; if so, try at your own risk). If this option is chosen, the option `FE.mesh_input.gmsh_filename` must be specified.

Several screencasts are available with Gmsh tutorials, including tutorials to create quadrilateral and hexahedral meshes.

`FE.mesh_input.box_dimensions` : `[ux uy uz]`
A $1 \times d$ array, where $d = 2, 3$ for 2D and 3D problems, respectively, that contains the dimensions of a rectangular (or cuboid) design region

in the x (`ux`), y (`uy`) and z (`uz`) directions.

`FE.mesh_input.elements_per_side` : `[nx ny nz]`
A $1 \times d$ array, where $d = 2, 3$ for 2D and 3D problems, respectively, that contains the number of elements to be generated in the x (`nx`), y (`ny`) and z (`nz`) directions.

`FE.mesh_input.mesh_filename` : A string with the path and name of a MATLAB `.mat` file with a mesh previously generated with the `makemesh` utility in the code.

`FE.mesh_input.gmsh_filename` : A string with the path and name of a file with a quadrilateral or hexahedral mesh created with Gmsh and exported to MATLAB.

`FE.mesh_input.bcs_file` : A string with the path and name of the MATLAB `.m` script that contains the specification of the displacement BCs and loads.

3.3 Material properties

`FE.material.E` : Elasticity modulus of the material.

`FE.material.nu` : Poisson's ratio of the material.

→ Eq. (5) `FE.material.rho_min` : Lower bound on combined density to prevent an ill-posed analysis.

FE.material.nu_void : Poisson's ratio of the lower bound material.

3.4 Initial design

`GEOM.initial_design.restart` : Options: `true` or `false`
If `false`, start the optimization from a MATLAB `.m` file with a specification of the initial design (e.g., as done in Section 2.6).
If `true`, starts optimization from a design previously saved to a `.mat` file.

`GEOM.initial_design.path` : A string with the path and name of a file containing the bars that make up the initial design.
If `GEOM.initial_design.restart = false`, this file must correspond to a `.m` file that builds the points and bars matrices, such as the one discussed in Section 2.6.
If `GEOM.initial_design.restart = true`, this file must correspond to a a MATLAB `.mat` file with a design previously generated with the code. Note that during the optimization, the code saves the current design at every iteration into such a `.mat` file.

→ Eq. (19) `GEOM.min_bar_radius` `GEOM.max_bar_radius` : The lower and upper bounds on the bar radius r_b for *all* bars. If you want to impose a fixed radius simply make the two bounds equal if using `fmincon` as an optimizer; or make them only slightly different if using MMA as an optimizer (cf. Section 2.4)

3.5 Finite element analysis

`FE.analysis.solver.type` : Options: `'direct'`, `'iterative'`
If `'direct'`, the code uses the direct solver (it simply uses MATLAB's left division).
If `'iterative'`, the code uses MATLAB's preconditioned conjugate gradient (PCG) method (`pcg`) for the solution. If this option is

chosen, you must also specify the `FE.analysis.solver.tol` and `FE.analysis.solver.maxit` options.

`FE.analysis.solver.tol` : Convergence tolerance on the relative residual norm for PCG.

`FE.analysis.solver.maxit` : Maximum number of iterations for PCG.

`FE.analysis.solver.use_gpu` : Options: `true` or `false`
If `true`, and if the system has an NVIDIA GPGPU card, solves the system of linear equations using GPUs. In this case, the code uses a Jacobi preconditioner for PCG.
If `false`, use the system CPUs for the iterative solution. In this case, the code uses an incomplete Cholesky preconditioner for PCG.

3.6 Optimization problem

`OPT.functions.objective` : A string with the name of the objective function. This is a string corresponding to the name given to the function in the list of functions to be computed (see the `init_optimization.m` script.) For this educational code, this parameter is `'compliance'`.

`OPT.functions.constraints` : A Matlab cell array (enclosed in curly braces, '{}') with the names of the functions that are constraints in the optimization. These names correspond to the names given the functions in the list of functions to be computed (see the `init_optimization.m` script.). For this educational code, this parameter is {`'volume fraction'`}.

`OPT.functions.constraint_limit` : A one-dimensional array with the values of the constraint limits for inequality constraints. As customary in nonlinear programming, all inequality constraints should be of the form '$\leq$'.

3.7 Geometry projection

`OPT.parameters.elem_r` :
$\rightarrow$ Eq. (2)
Sample window radius r_e. By default, this parameter is commented out, and this radius is automatically computed by the code as the radius of the circle (in 2D) or sphere (in 3D) that circumscribes the element. If uncommented, the value you provide will overwrite the default, and in it must be given as an absolute dimension. If a scalar is given, then the same sample window radius will be used for all elements. If a vector is passed, it should be of dimensions $n_e \times 1$, where n_e is the total number of elements.

`OPT.parameters.smooth_max_scheme` : Options: `'mod_p_norm'`, `'mod_p-mean'`, `'KS'`, `'KS_under'`
$\rightarrow$ Eq. (5)
Smooth maximum function to combine multiple effective densities into single combined density. These functions are defined in the `smooth_max.m` script.

`OPT.parameters.smooth_max_param` : Parameter for the smooth maximum function.

`OPT.parameters.penalization_scheme` : Options: `'SIMP'`, `'RAMP'`
$\rightarrow$ Eq. (4)
Penalization function μ to compute effective density.

`OPT.parameters.penalization_param` : Parameter for the penalization function.

3.8 Optimization and output

`OPT.options.optimizer` : Options: `'fmincon-active-set'`, `'mma'`
Optimization algorithm. If `'fmincon-active-set'`, the Optimization Toolbox in MATLAB should be installed. If `'mma'`, the MMA MATLAB files must be copied to the
`optimization` folder in the installation.

`OPT.options.write_to_vtk` : Options: `'none'`, `'last'`, `'all'`
If `'none'`, do not write any Visualization Toolkit (VTK) output.
If `'last'`, write the mesh and combined density field of the design to a `.vtk` file for the design of the last optimization iteration.
If `'all'`, write a `.vtk` file for every optimization in the optimization. This files can be used for visualization with many tools that can process `.vtk` files.

`OPT.options.vtk_output_path` : A string with the path of the folder where output files should be written to.

`OPT.options.dv_scaling` : Options: `true` or `false`
→ Eq. (32)
If `true`, scale design variables so that they all lie within the $[0, 1]$ interval.

`OPT.options.move_limit` : Move limit on design update at each iteration.
→ Eq. (33)

`OPT.options.max_iter` : Maximum number of iterations for the optimization.

`OPT.options.step_tol` : Stopping criterion for the relative change in design variables, computed as the 2-norm of the difference between the vectors of design variables for two consecutive iterations.

`OPT.options.kkt_tol` : Stopping criterion for optimality , measured as the 2-norm of the Karush-Kuh-Tucker (KKT) optimality conditions.

3.9 Finite difference check of sensitivities

`OPT.make_fd_check` : Options: `true` or `false`
If `true`, perform forward finite difference check of analytical sensitivities.

`OPT.fd_step_size` : Size of the finite difference perturbation.

`OPT.check_cost_sens` : Options: `true` or `false`
If `true`, perform finite difference check of the sensitivities of the cost (objective) function.

`OPT.check_cons_sens` : Options: `true` or `false`
If `true`, perform finite difference check of the sensitivities of the cost (objective) function.

References

[1] Krister Svanberg. The method of moving asymptotes—a new method for structural optimization. *International Journal for Numerical Methods in Engineering*, 24(2):359–373, 1987.

[2] Krister Svanberg. A class of globally convergent optimization methods based on conservative convex separable approximations. *SIAM Journal on Optimization*, 12(2):555–573, 2002.

[3] Krister Svanberg. Mma and gcmma, versions september 2007. *Optimization and Systems Theory*, page 104, 2007.

[4] Christophe Geuzaine and Jean-François Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International journal for numerical methods in engineering*, 79(11):1309–1331, 2009.

[5] Erik Andreassen, Anders Clausen, Mattias Schevenels, Boyan S Lazarov, and Ole Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011.