AN EFFICIENT AND COMPACT MATLAB IMPLEMENTATION OF TOPOLOGY OPTIMIZATION: APPLICATION TO COMPLIANT MECHANISM

Anderson Pereira[a], Ivan F. M. Menezes[a], Cameron Talischi[b] and Glaucio H. Paulino[b]

[a]*Tecgraf (Group of Technology in Computer Graphics), Pontifical Catholic University of Rio de Janeiro (PUC–Rio), Rua Marquês de São Vicente, 225, 22453-900, Rio de Janeiro, RJ, Brazil, {anderson,ivan}@tecgraf.puc-rio.br, http://www.tecgraf.puc-rio.br*

[b]*Department of Civil and Environmental Engineering, University of Illinois at Urbana-Champaign, Newmark Laboratory, 205 North Mathews Avenue, Urbana, IL, 61801, U.S.A. {ktalisch,paulino}@uiuc.edu, http://cee.illinois.edu*

Keywords: Topology optimization, Unstructured meshes, Polygonal finite elements, Compliant mechanism, MATLAB software

Abstract. This paper presents an effective MATLAB implementation of a general topology optimization method for compliant mechanism synthesis of statically loaded structures. Our implementation is based on the educational framework `PolyTop` (Talischi et al., 2011b), which is easily extended to handle compliant mechanism design. The main features of `PolyTop` are preserved, including a general finite element module using polygons, which are superior to conventional linear triangles and quads in topology optimization as they are not susceptible to checkerboard patterns. The MATLAB code is explained in detail and benchmark numerical examples are presented to illustrate the capabilities of the code. Examples of mechanism synthesis are presented. Moreover, `PolyTop` offers room for further exploration of finite elements and topology optimization formulations both for research and for practical engineering applications.

1 INTRODUCTION

Since the publication of the popular "99 line" code (Sigmund, 2001) sharing and publication of educational software has become a tradition in the topology optimization community. For example, Allaire and Pantz (2006) presented a structural optimization code based on FreeFem++, Liu et al. (2005) introduced a coupled level set method using the FEMLAB package and Challis (2010) presented a discrete level-set MATLAB code very much in the spirit of the "99 line" code. Suresh (2010) developed a 199 line code for Pareto-optimal tracing with the aid of topological derivatives. The recent "88 line" code (Andreassen et al., 2011), an improved version of the popular "99 line" code, was developed to achieve higher efficiency. Recently, `PolyMesher` (Talischi et al., 2011a) and `PolyTop` (Talischi et al., 2011b) complemented the technical literature by means of a MATLAB implementation of topology optimization that, among other features, introduces a general framework for finite element discretization and analysis. As demonstrated in reference (Talischi et al., 2011b), `PolyTop` outperforms the "88 line" MATLAB code (Andreassen et al., 2011).

In this work, we present the implementation of compliant mechanism problem in the `PolyMesher`/`PolyTop` framework. The remainder of this paper is organized as follows: in the next two sections, we review the main concepts of `PolyMesher` and `PolyTop`. We explain the MATLAB implementation of this algorithm in section 4 and present numerical examples in section 5. Conclusions, potential extensions and generalizations of the code are addressed in section 6.

2 MESH GENERATOR: `PolyMesher`

`PolyMesher` is a simple and robust MATLAB code for polygonal mesh generation. The main ingredients of `PolyMesher` are the implicit representation of the domain and the use of Centroidal Voronoi diagrams for its discretization. The implicit description offer great flexibility to construct a relatively large class of domains with algebraic expressions. A discretization of the domain is constructed from a Centroidal Voronoi tessellation (CVT) that incorporates an approximation to its boundary. This approximation is obtained by including the set of reflections of the seeds (Bolander and Saito, 1998; Yip et al., 2005). Additionally, the Lloyd's method is used to establish a uniform (optimal) distribution of seeds and thus a high quality mesh (Talischi et al., 2010).

2.1 Use of `PolyMesher`

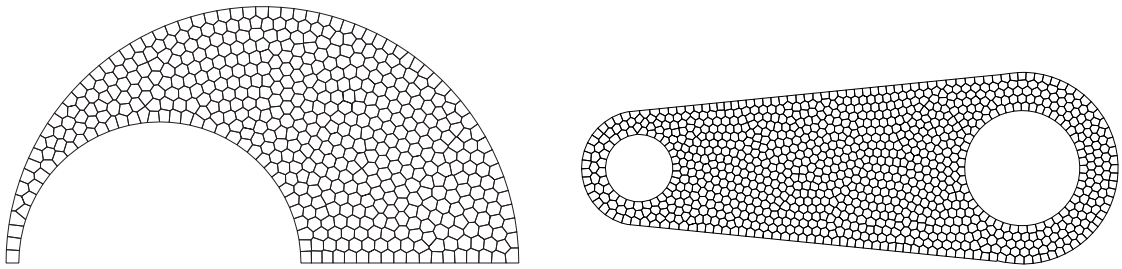
To generate a mesh using `PolyMesher`, the user needs to define a `Domain` function that defines the meshing domain. The `Domain` function is passed to the `PolyMesher` providing access to information about domain geometry (i.e., the sign distance function), the bounding box and the boundary conditions. The details of this function are discussed in Talischi et al. (2011a).

The call to `PolyMesher` is as follows:

```
[Node,Element,Supp,Load,P] = PolyMesher(@Domain,NElem,MaxIter,P)
```

where `Domain` is a MATLAB function defining the domain, `NElem` is the desired number of elements in the mesh, `MaxIter` specifies the maximum number of Lloyd's iterations and `P` is an optional argument where the user can input an initial set of seeds.

Figure 1 shows sample meshes generated using `PolyMesher`. The `Domain` functions for these examples, geometries and the library of distance functions are provided as supplementary material in Talischi et al. (2011a).



(a) Horn geometry. (b) Wrench geometry.

Figure 1: Sample meshes generated using `PolyMesher` (Talischi et al., 2011a).

3 TOPOLOGY OPTIMIZATION: `PolyTop`

`PolyTop` is an efficient MATLAB code for structural topology optimization that includes a general finite element routine based on isoparametric polygonal elements. The code also features a modular structure in which the analysis routine and the optimization algorithm are separated from the specific choice of topology optimization formulation. Within this framework, the finite element and sensitivity analysis routines contain no information related to the formulation and thus can be extended, developed and modified independently.

3.1 MATLAB files

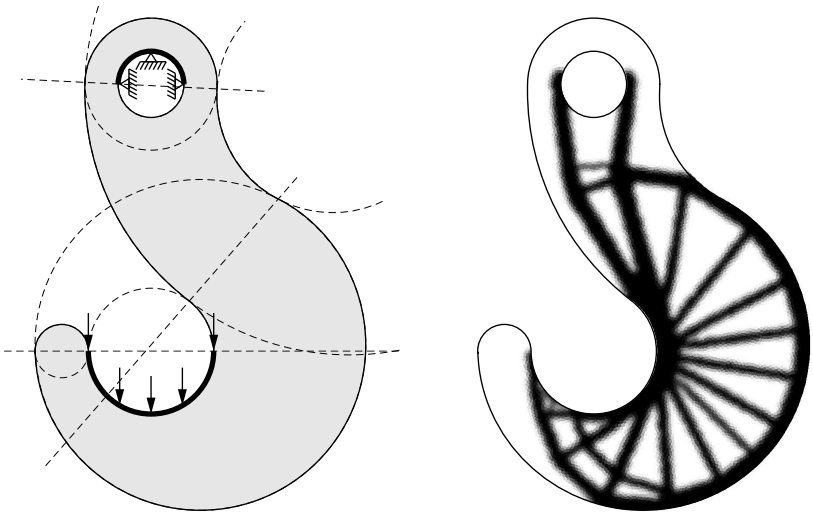
To run `PolyTop`, the user needs two main files. The first, `PolyTop.m` is the kernel of the code that contains the optimizer and analysis routines, including the FE routine and functions responsible for computing cost functional and their sensitivities. The second, `PolyScript.m`, a MATLAB script that calls the kernel, holds all the parameters related to topology optimization that link design variables with the analysis parameters (e.g. filter matrix, material interpolation functions), as well as the finite element model (e.g. the mesh, load and support boundary conditions for the state equation).

In the implementation of `PolyScript`, the auxiliary functions `PolyMesher` and `PolyFilter` are called to generate the finite element mesh and construct the linear filtering matrix. Another auxiliary function `MatIntFnc.m`, responsible for the material interpolation, is passed to the kernel via the `opt.MatIntFnc` field.

All files listed above are provided as supplementary material in Talischi et al. (2011b). Figure 2 shows the final topology for a non-trivial geometry using `PolyTop`.

4 MATLAB IMPLEMENTATION OF COMPLIANT MECHANISM PROBLEM

In the previous sections we briefly described the `PolyMesher` and `PolyTop` implementation. The aim now is to explain how to extend those codes to solve compliant mechanism problems. The original prototype for `PolyTop` is written to solve the compliance minimization problem but specific functions are designed to compute the objective and constraint functions. This functional decoupling easily allows to extend to compliant mechanism problems without major changes in the original code.



(a) Hook geometry. (b) Hook final topology.

Figure 2: Sample compliance problems with non-trivial geometry using `PolyTop` (Talischi et al., 2011b).

4.1 Input data and `PolyScript`

All the input and internal parameters of the code are collected in two MATLAB struct arrays. One struct, called `fem`, contains all the FE-related parameters while the other, `opt`, has the variables pertaining to the topology optimization formulation and optimizer. For the compliant mechanism, new fields are added to the original `fem` structure (see Table 1) while the `opt` remains the same (see Table 2). Note that some of the `fem` fields are populated inside the `PolyTop` kernel unless they are already specified. Also the user has access to all the model parameters since these structures reside in the MATLAB workspace.

4.2 Compliant Mechanism and `PolyTop`

The objective function of the optimization problem for the minimum compliance is given by:

$$f = \mathbf{F}^T\mathbf{U} \tag{1}$$

while for the compliant mechanism, we have:

$$f = \mathbf{L}^T\mathbf{U} \tag{2}$$

where $\mathbf{F}$ and $\mathbf{U}$ are the global force and displacement vectors and $\mathbf{L}$ is a vector composed of zeros except the degree of the output position which is one.

For a discretized linear state equation, $\mathbf{KU} = \mathbf{F}$ where $\mathbf{K}$ is the stiffness matrix, the sensitivity of the objective function with respect to a state variable $\mathbf{E}$ is given by:

$$\frac{\partial f}{\partial \mathbf{E}} = -\boldsymbol{\lambda}^T \frac{\partial \mathbf{K}}{\partial \mathbf{E}}\mathbf{U} \tag{3}$$

where $\boldsymbol{\lambda}$ is the solution to the adjoint load problem

$$\mathbf{K}\boldsymbol{\lambda} = \frac{\partial f}{\partial \mathbf{U}} \tag{4}$$

Table 1: List of fields in the `fem` structure. The fields marked with the superscript †, if empty, are populated inside `PolyTop`.

Field	Definition
`fem.NNode`	Number of nodes
`fem.NElem`	Number of elements
`fem.Node`	[`NNode` × 2] array of nodes
`fem.Element`	[`NElement` × Var] cell array of elements
`fem.Supp`	[`NSupp` × 3] Array of supports
`fem.Load`	[`NLoad` × 3] Array of loads
`fem.Nu0`	Poisson's ratio of solid material
`fem.E0`	Young's modulus of solid material
`fem.Reg`	Tag for regular meshes
`fem.ElemNDof`†	Array showing number of DOFs of elements
`fem.ShapeFnc`†	Cell array with tabulated shape functions and weights
`fem.k`†	Array of local stiffness matrix entries
`fem.i`†	Index array for sparse assembly of `fem.k`
`fem.j`†	Index array for sparse assembly of `fem.k`
`fem.e`†	Array of element IDs corresponding to `fem.k`
`fem.ElemArea`†	Array of element areas
`fem.F`†	Global load vector
`fem.FreeDofs`†	Array of free degrees of freedom
Added fields	
`fem.Spring`	[`NSpring` × 3] Array of springs
`fem.DOut`	Target node
`fem.DofDOut`†	Target DOF
`fem.s`†	Array of global springs entries

For the compliance minimization problem, Eq. 1, the right hand side of Eq. 4 is $\partial f/\partial \mathbf{U} = \mathbf{F}$ leading to a self-adjoint problem where the solution of the adjoint equation is $\boldsymbol{\lambda} = \mathbf{U}$. In this case, the sensitivity of the objective function is given by:

$$\frac{\partial f}{\partial \mathbf{E}} = -\mathbf{U}^T \frac{\partial \mathbf{K}}{\partial \mathbf{E}} \mathbf{U} \tag{5}$$

On the other hand, for the compliant mechanism problem, Eq. 2, the right hand side of Eq. 4 is $\partial f/\partial \mathbf{U} = \mathbf{L}$ where for the solution of the adjoint equation we need to solve another linear system of equations:

$$\mathbf{K}\boldsymbol{\lambda} = \mathbf{L} \tag{6}$$

In order to computer the sensitivities efficiently, `PolyTop` stores the element stiffness matrix using the index vectors `fem.i`, `fem.j` and `fem.k`. For compliance minimization we need to compute $-\sum \mathbf{U}_i \left(\mathbf{k}_l\right)_{ij} \mathbf{U}_j$ where the sum is taken over all DOFs i and j of element Ω_l. This requires summing the block of `-U(fem.i).*fem.k.*U(fem.j)` that corresponds to element l. This computation is carried out using `cumsum`, the cumulative sum function in MATLAB. For compliant mechanism we compute $-\sum \boldsymbol{\lambda}_i \left(\mathbf{k}_l\right)_{ij} \mathbf{U}_j$, where the adjoint solution $\boldsymbol{\lambda}$ is stored in the second column of $\mathbf{U}$ vector so the sensitivities are computed summing the block of `-U(fem.i,1).*fem.k.*U(fem.j,2)`. The incomplete code for solving compliant mechanism problem is presented in the Appendix A. The complete code can be assembled by adding the differences in Appendix A with the original `PolyTop` code.

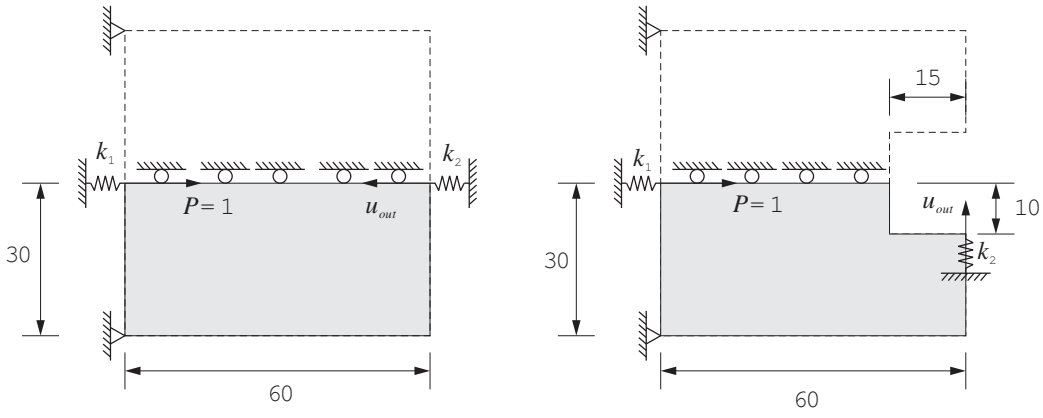
Table 2: List of fields in the `opt` structure.

Field	Definition
`opt.zMin`	Lower bound for design variables
`opt.zMax`	Upper bound for design variables
`opt.zIni`	Initial design variables
`opt.MatIntFnc`	Handle to material interpolation fnc.
`opt.P`	Matrix that maps design to element variables
`opt.VolFrac`	Specified volume fraction cosntraint
`opt.Tol`	Convergence tolerance on design variables
`opt.MaxIter`	Max. number of optimization iterations
`opt.OCMove`	Allowable move step in OC update scheme
`opt.OCEta`	Exponent used in OC update scheme

5 NUMERICAL EXAMPLES

In this section, we present numerical results for benchmark compliant mechanism problems to demonstrate the versatility of the code. For all results, the Ersatz parameter ϵ was set to 10^{-4} and the Young's modulus and Poisson's ratio of the solid phase were taken to be $E_0 = 1$ and $\nu = 0.3$, respectively. Also the maximum tolerance for the change in design was taken to be 1%. The Solid Isotropic Material with Penalization (SIMP) model (Bendsøe, 1989; Rozvany et al., 1992; Rozvany, 2009) was adopted with continuation on the penalty parameter p in the following manner: the value of p was increased from 1 to 4 using increments of size 0.5 and for each value of p, a maximum of 150 iterations was performed (by setting `opt.MaxIter=150`). For the optimization algorithm, `UpdateScheme` function, we used move limit $M = 0.05$ (by setting `opt.OCMove=0.05`), and numerical damping parameter $\eta = 0.3$ (by setting `opt.OCEta=0.3`). Also, for stable convergence of the OC update, positive sensitivities were replaced by a small positive number

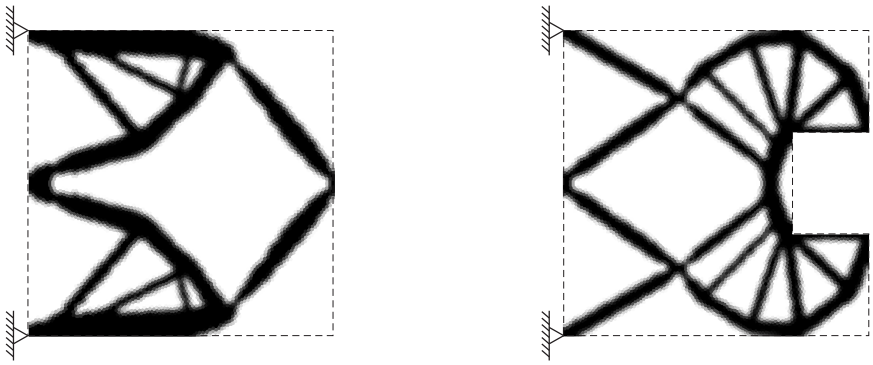
The examples presented are the force inverter and gripper compliant mechanism design problems, as shown in Fig. 3, Due to the symmetry of the problem, only half of the domain is considered in the optimization algorithm (shaded areas in Fig. 3). The spring constants k_1 and k_2 are 0.1. For each problem, a mesh of $5,000$ polygonal elements was generated using `PolyMesher` (Talischi et al., 2011a).



(a) Inverter mechanism. (b) Gripper mechanism.

Figure 3: Compliant mechanism problems.

The final result shown in Fig. 4 was obtained using a linear filter of radius 1.2.



(a) Inverter mechanism. (b) Gripper mechanism.

Figure 4: Final topologies.

6 CONCLUSIONS

An efficient implementation for compliant mechanism synthesis of statically loaded structures was presented using the recently developed educational framework `PolyTop` (Talischi et al., 2011b). `PolyTop` was easily extended to handle compliant mechanism design by replacing 10 existing lines and adding 7 new ones. Numerical examples were presented to illustrate the capabilities of the code.

We would like to point out that `PolyTop` offers room for further exploration of finite elements and topology optimization formulations both for research and for practical engineering applications. We hope that the community can make use of `PolyTop` in ways that we cannot anticipate.

Acknowledgments

The first two authors acknowledge the financial support provided by Tecgraf (Group of Technology in Computer Graphics), PUC-Rio, Rio de Janeiro, Brazil. The last two acknowledge the support by the Department of Energy Computational Science Graduate Fellowship Program of the Office of Science and National Nuclear Security Administration in the Department of Energy under contract DE-FG02-97ER25308.

A APPENDIX: MATLAB CODE FOR COMPLIANT MECHANISM SYNTHESIS

The MATLAB code for compliance minimization `PolyTop` described in Talischi et al. (2011b) can be changed into a code for mechanism synthesis by replacing 10 existing lines and adding 7 new ones. These consist of change in the finite element analysis functions (to account for the input/output springs as well as solving the adjoint problem), the objective function (changing compliance to compliant mechanism), and the update scheme (changes needed to stabilize OC for the compliant mechanism problem). Instead of listing the whole program we just show a list of changes. To facilitate the comparison, we have renamed the modified kernel `PolyTopM.m`. The list of changes is obtained by comparing `PolyTop.m` with `PolyTopM.m` using the UNIX command "diff PolyTop.m PolyTopM.m". This results in output where "<"

means lines in `PolyTop.m` and ">" means lines in `PolyTopM.m`. In the following we briefly discuss the changes.

First we rename the code from "PolyTop" to "PolyTopM"

```
7c7
< function [z,V,fem] = PolyTop(fem,opt)
---
> function [z,V,fem] = PolyTopM(fem,opt)
```

The objective function was changed to a target output displacement and the sensitivities depend on the solution to the adjoint load case (second column of the displacement matrix **U**)

```
29,30c29,30
< f = dot(fem.F,U);
< temp = cumsum(-U(fem.i).*fem.k.*U(fem.j));
---
> f = U(fem.DofDOut,1);
> temp = cumsum(U(fem.i,1).*fem.k.*U(fem.j,2));
```

The bi-sectioning part of the `UpdateScheme` was changed as

```
51c51
< while l2-l1 > 1e-4
---
> while (l2-l1)/(l2+l1) > 1e-4 && l2>1e-40
```

Also, for stabilizing convergence, positive sensitivities were replaced by a small positive number (Bendsøe and Sigmund, 2003)

```
53c53
<    B = -(dfdz./dgdz)/lmid;
---
>    B = max(1e-10,-(dfdz./dgdz)/lmid);
```

The external force vector was allocated for the real and the adjoint load cases and the external springs were considered

```
81,83c81,90
<    fem.F = zeros(2*fem.NNode,1);  %external load vector
<    fem.F(2*fem.Load(1:NLoad,1)-1) = fem.Load(1:NLoad,2);  %x-crdnt
<    fem.F(2*fem.Load(1:NLoad,1))   = fem.Load(1:NLoad,3);  %y-crdnt
---
>    fem.F = zeros(2*fem.NNode,2);  %external load vector
>    fem.F(2*fem.Load(1:NLoad,1)-1,1) = fem.Load(1:NLoad,2);  %x-crdnt
>    fem.F(2*fem.Load(1:NLoad,1),1)   = fem.Load(1:NLoad,3);  %y-crdnt
>    fem.DofDOut = 2*fem.DOut(1)-2+fem.DOut(2);
>    fem.F(fem.DofDOut,2) = -1;
>    NSpring = size(fem.Spring,1);
>    s = zeros(2*fem.NNode,2);  %spring vector
>    s(2*fem.Spring(1:NSpring,1)-1) = fem.Spring(1:NSpring,2);  %x-crdnt
>    s(2*fem.Spring(1:NSpring,1))   = fem.Spring(1:NSpring,3);  %y-crdnt
>    fem.s = spdiags(s(:),0,2*fem.NNode,2*fem.NNode);
```

The assembly of the global stiffness matrix was changed for adding the external springs

```
91c98
< K = sparse(fem.i,fem.j,E(fem.e).*fem.k);
---
> K = sparse(fem.i,fem.j,E(fem.e).*fem.k) + fem.s;
```

Finally, we allocated the displacement vector for the real and the adjoint load cases

```
93c100
< U = zeros(2*fem.NNode,1);
---
> U = zeros(2*fem.NNode,2);
```

REFERENCES

Allaire G. and Pantz O. Structural optimization with FreeFem++. *Struct Multidisc Optim*, 32(3):173–181, 2006.

Andreassen E., Clausen A., Schevenels M., Lazarov B.S., and Sigmund O. Efficient topology optimization in MATLAB using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, 2011. doi:10.1007/s00158-010-0594-7.

Bendsøe M. and Sigmund O. *Topology Optimization Theory, Methods and Applications*. Berlin: Springer, 2003.

Bendsøe M.P. Optimal shape design as a material distribution problem. *Structural and Multidisciplinary Optimization*, 1(4):193–202, 1989.

Bolander J.E. and Saito S. Fracture analyses using spring networks with random geometry. *Eng Fract Mech*, 61(5-6):569–591, 1998.

Challis V.J. A discrete level-set topology optimization code written in matlab. *Struct Multidisc Optim*, 41(3):453–464, 2010.

Liu Z., Korvink J., and Huang R. Structure topology optimization: fully coupled level set method via FEMLAB. *Struct Multidisc Optim*, 29:407–417, 2005.

Rozvany G. A critical review of established methods of structural topology optimization. *Structural and Multidisciplinary Optimization*, 37:217–237, 2009. ISSN 1615-147X. 10.1007/s00158-007-0217-0.

Rozvany G.I.N., Zhou M., and Birker T. Generalized shape optimization without homogenization. *Structural and Multidisciplinary Optimization*, 4:250–252, 1992. ISSN 1615-147X. 10.1007/BF01742754.

Sigmund O. A 99 line topology optimization code written in Matlab. *Struct Multidisc Optim*, 21(2):120–127, 2001.

Suresh K. A 199-line matlab code for Pareto-optimal tracing in topology optimization. *Struct Multidisc Optim*, 42(5):665–679, 2010.

Talischi C., Paulino G.H., Pereira A., and Menezes I.F.M. Polygonal finite elements for topology optimization: A unifying paradigm. *Int J Numer Meth Eng*, 82(6):671–698, 2010.

Talischi C., Paulino G.H., Pereira A., and Menezes I.F.M. PolyMesher: A general-purpose mesh generator for polygonal elements written in Matlab. *Structural and Multidisciplinary Optimization*, 2011a. In Press, doi:10.1007/s00158-011-0706-z.

Talischi C., Paulino G.H., Pereira A., and Menezes I.F.M. PolyTop: A Matlab implementation of a general topology optimization framework using unstructured polygonal finite element meshes. *Structural and Multidisciplinary Optimization*, 2011b. In Press, doi:10.1007/s00158-011-0696-x.

Yip M., Mohle J., and Bolander J. Automated modeling of three-dimensional structural components using irregular lattices. *Comput-Aided Civ Inf*, 20(6):393–407, 2005.