TopOpt.jl

An efficient and high-performance topology optimization package in the Julia programming language

Mohamed Tarek Mohamed

Wold Congress of Structural and Multidisciplinary Optimization 13

May 20, 2019

Introduction

About me

- Second year PhD candidate at University of New South Wales, Canberra, Australia
- Background in mechanical and industrial engineering
- Research interest in topology optimization
  - Continuation methods and parameter interactions
  - Local stress-constrained optimization
  - Large-scale buckling constrained optimization
- Active open source developer in the Julia community
- Former Google Summer of Code student and current mentor
- GitHub: https://github.com/mohamed82008

About me

Contributions:

Linear algebra	Machine learning
• IterativeSolvers.jl • Preconditioners.jl • AlgebraicMultigrid.jl	• Turing.jl • Bijectors.jl
Parallelism	**Visualization**
• KissThreading.jl	• VTKDataTypes.jl • VTKDataIO.jl

Why Julia?

- Open source
- Friendly syntax similar to Matlab and Python
- Can be as fast as C and Fortran
- Excellent linear algebra support
- Excellent mathematical optimization ecosystem
  - JuMP.jl and MathOptInterface.jl
  - Optim.jl and LineSearches.jl
  - Many more

What is TopOpt.jl?

Topology optimization

There are many families of topology optimization problems characterized by:

- Decision variables
- Objective(s)
- Constraint(s)
- Mechanical system and materials
- Boundary conditions

TopOpt.jl

- Open source topology optimization program - MIT licensed
- Written with efficiency in mind
- 100% in Julia
- Friendly user interface
- Extensible design
- Ambitious goals
  - End-to-end topology optimization
  - State-of-the-art algorithms
  - Efficient and scalable implementations of algorithms (multiple GPUs, distributed computing, etc.)

Features and examples

Problem definition

- Input problem specs: mesh, material, boundary conditions

```
problem = InpStiffness("example.inp",
↪   keep_load_cells = true)
```

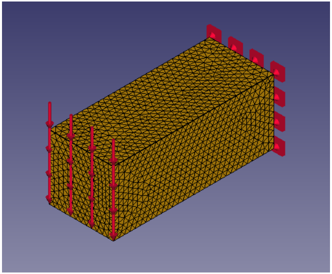


Figure: Problem definition in FreeCAD.

Finite element solver

- Choose x_{min}

```
xmin = 1e-4
```

- Choose the penalty function and parameter

```
penalty = PowerPenalty(1.0)
penalty = RationalPenalty(0.0) [7]
penalty = SinhPenalty(1.0) [3]
```

$$\boldsymbol{K} = \sum_e P(\rho_e; p)\boldsymbol{K}_e$$

Finite element solver

- Create a finite element solver
  - Cholesky-based linear system solver

    ```
    solver = FEASolver(Displacement, Direct,
    problem, xmin = xmin, penalty = penalty)
    ```
  - Assembly-based conjugate gradient (CG) linear system solver

    ```
    solver = FEASolver(Displacement, CG,
    Assembly, problem, xmin = xmin, penalty =
    penalty)
    ```
  - Assembly-free/matrix-free CG linear system solver

    ```
    solver = FEASolver(Displacement, CG,
    MatrixFree, problem, xmin = xmin, penalty =
    penalty)
    ```
- Planned extension: distributed finite element analysis

Finite element solver

- Other keyword arguments
  - `quad_order`: Gaussian quadrature order
  - `conv`: convergence criteria of the CG algorithm, e.g. `conv = EnergyCriteria()` [1]
  - `cg_max_iter`: maximum number of iterations in the CG algorithm
  - tol: tolerance of the CG algorithm

Objective and constraint functions

- Create a function
  - Compliance function with chequerboard sensitivity filter [5]

    ```
    compfunc = ComplianceFunction(problem,
    solver, filtering = true, rmin = 30.0)
    ```
  - Volume fraction

    ```
    volfunc = VolumeFunction(problem, solver)
    ```
  - WIP extension: aggregated stress violation functions
  - Planned extension: density interpolation filter, and buckling support

Objective and constraint functions

- Create the objective and constraints
  - Minimization objective

    ```
    obj = Objective(compfunc)
    ```

  - Constraint $volfunc(x) \leq 0.3$

    ```
    constr = Constraint(volfunc, 0.3)
    ```

  - Multiple constraints

    ```
    constr = (Constraint(...), Constraint(...))
    constr = [Constraint(...), Constraint(...)]
    ```

  - Planned extension: block constraints, semidefinite constraints and multi-objective support

Mathematical programming

- Method of moving asymptotes [8, 9]
  - `optimizer = MMAOptimizer(obj, constr, MMA87(), ConjugateGradient())`
  - `MMA87()`/`MMA02()`: method of moving asymptotes [8]/[9]
  - A log-barrier approach is used to handle the box constraints of the dual
  - Planned extension: Ipopt, NLopt, augmented Lagrangian solver and multi-objective support

Topology optimization

- Solid isotropic material with penalization (SIMP) [2]

  `simp = SIMP(optimizer, 3.0)` (penalty is 3.0)

- Bi-directional evolutionary structural optimization (BESO) [5]

  `beso = BESO(obj, constr; p = 3.0, maxiter = 200, tol = 0.0001, er = 0.02)`

- Genetic evolutionary structural optimization [6, 10]

  `geso = GESO(obj, constr; p = 3.0, maxiter = 1000, tol = 0.0001, Pcmin = 0.6, Pcmax = 1.0, Pmmin = 0.5, Pmmax = 1.0, string_length = 4)`

- Planned extension: level set methods

Topology optimization

- Continuation SIMP
  - Easy constructor: 40 penalty steps with power penalty from $p = 1$ to $p = 5$, i.e. 41 subproblems.

    ```
    cont_simp = ContinuationSIMP(simp, 40)
    ```
  - Almost all the SIMP and MMA options can be changed by any of the following continuation schemes.
    - Power continuation: $f(i) = a \times i^b + c$
    - Exponential continuation: $f(i) = a \times e^{b \times i} + c$
    - Logarithmic continuation: $f(i) = a \times log(b \times i) + c$
  - The penalty parameter of the rational penalty function can be additionally changed using the continuation scheme in [7].

    ```
    p_gen = Continuation(RationalPenalty(0.0),
    steps = 40, xmin = xmin)
    ```

Topology optimization

- Rational penalty and decreasing tolerance continuation SIMP

```
steps = 40

# Decreasing tolerance generator
maxtol, mintol = 0.1, 0.001
b = log(mintol / maxtol) / steps
a = maxtol / exp(b)
tol_gen = ExponentialContinuation(a, b, 0.0,
↪   steps+1, mintol)

# Default options for the MMA algorithm
default_mma_options = MMA.Options(maxiter=1000)

# MMA options generator
mma_options_gen = TopOpt.MMAOptionsGen(steps =
↪   steps, initial_options = default_mma_options,
↪   kkttol_gen = tol_gen)
```

Topology optimization

- Rational penalty and decreasing tolerance continuation SIMP

```
# Penalty generator
p_gen = Continuation(RationalPenalty(0.0), steps =
↪  steps, xmin = xmin)

# Continuation SIMP options
csimp_options = TopOpt.CSIMPOptions(steps = steps,
↪  options_gen = mma_options_gen, p_gen = p_gen)

# Instance of ContinuationSIMP
cont_simp = ContinuationSIMP(simp, steps,
↪  csimp_options)
```

Running the algorithm

- Set an initial design

```
x0 = ones(length(solver.vars))
```

- Run the algorithm

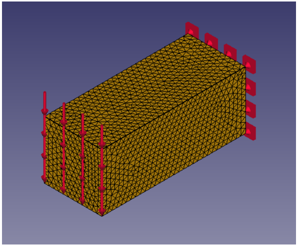
```
result = simp(x0)
result = beso(x0)
result = geso(x0)
result = cont_simp(x0)
```

- Shared result fields
  - Final topology: `result.topology`
  - Number of objective and constraint evaluations: `result.fevals`
  - Final objective value: `result.objval`

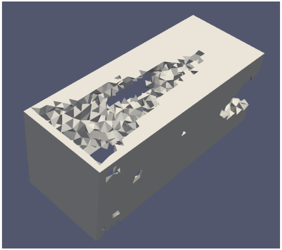
Output

- Output topology to vtu file

```
TopOpt.save_mesh(filename, problem,
↪   result.topology)
```



(a) FreeCAD



(b) Paraview

Figure: SIMP, power penalty, $p = 3$.

Usability issues

- Make GPU support optional
  - TopOpt.jl uses CUDA, doesn't work for AMD systems
  - Setting up CUDA on NVIDIA systems is not trivial
  - Most of the package doesn't require a GPU
- Improve README
- Add more tests and documentation

Conclusion

Get involved

- Download and try TopOpt.jl
- Open issues with bug reports, feature requests, and/or questions
- Read and contribute to the source code
- Send me an email (m.mohamed@student.adfa.edu.au / mohamed82008@gmail.com) to collaborate
  - Interesting applications or use cases
  - New algorithms or enhancements

Further Readings I

[1] Oded Amir, Mathias Stolpe, and Ole Sigmund. Efficient use of iterative solvers in nested topology optimization. *Structural and Multidisciplinary Optimization*, 42(1):55–72, 2010.

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

[3] T. E. Bruns. A reevaluation of the SIMP method with filtering and an alternative formulation for solid-void topology optimization. *Structural and Multidisciplinary Optimization*, 30(6):428–436, 2005.

[4] William W. Hager and Hongchao Zhang. Algorithm 851: CG_DESCENT, a conjugate gradient method with guaranteed descent. *ACM Transactions on Mathematical Software (TOMS)*, 32(1):113–137, 2006.

[5] Xiaodong Huang and Yi Min Xie. A further review of ESO type methods for topology optimization. *Structural and Multidisciplinary Optimization*, 41(5):671–683, 2010.

Further Readings II

[6] Xia Liu, Wei-Jian Yi, Q.S. Li, and Pu-Sheng Shen. Genetic evolutionary structural optimization. *Journal of Constructional Steel Research*, 64(3):305–311, 2008.

[7] M. Stolpe and K. Svanberg. An alternative interpolation scheme for minimum compliance topology optimization. *Structural and Multidisciplinary Optimization*, 22(2):116–124, 2001.

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

[9] Krister Svanberg. A Class of Globally Convergent Optimization Methods Based on Conservative Convex Separable Approximations. *SIAM Journal on Optimization*, 12(2):555–573, 2002.

[10] Z. H. Zuo, Y. M. Xie, and X. Huang. Combining genetic algorithms with BESO for topology optimization. *Structural and Multidisciplinary Optimization*, 38(5):511–523, 2009.

Questions?