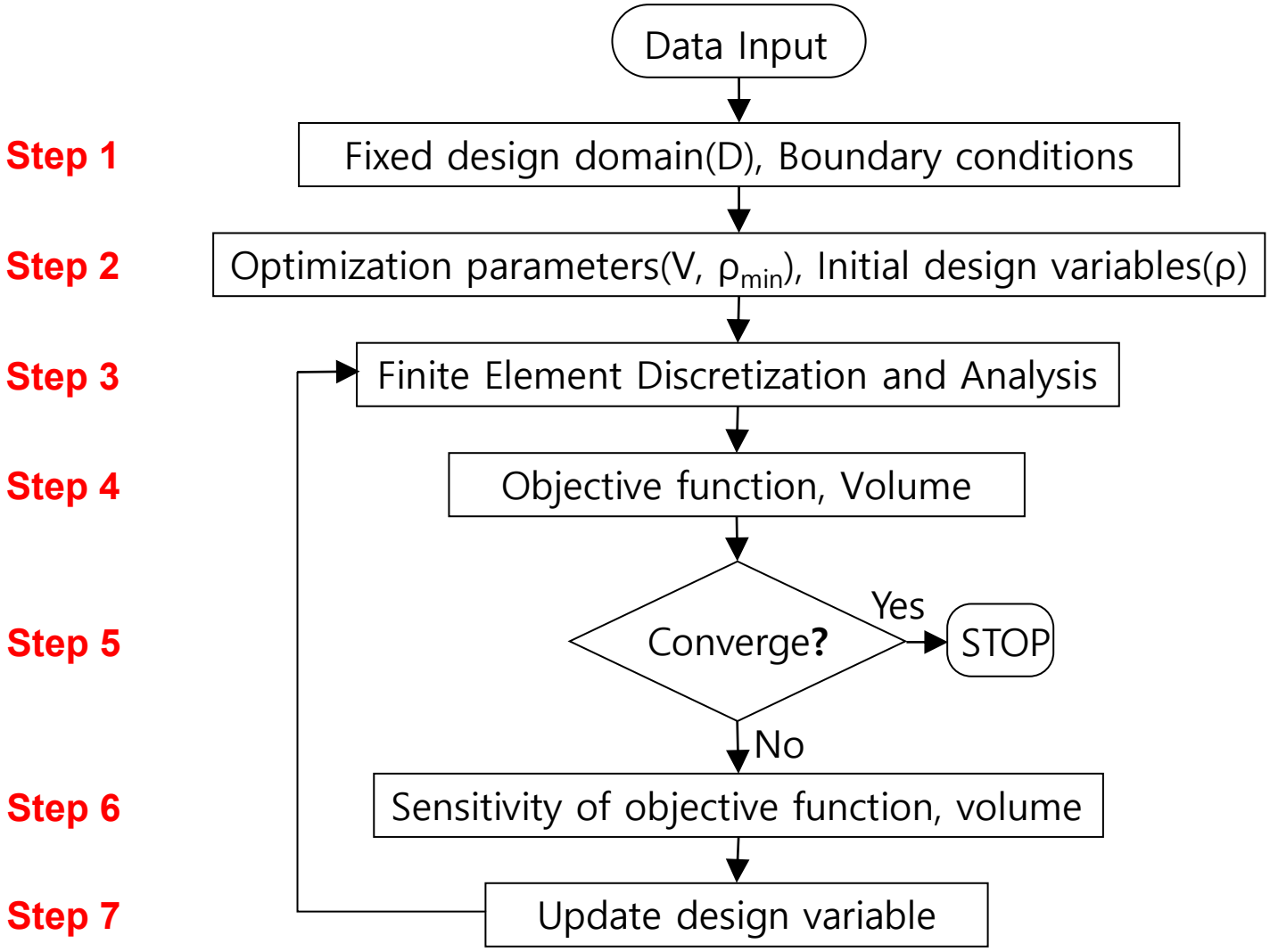
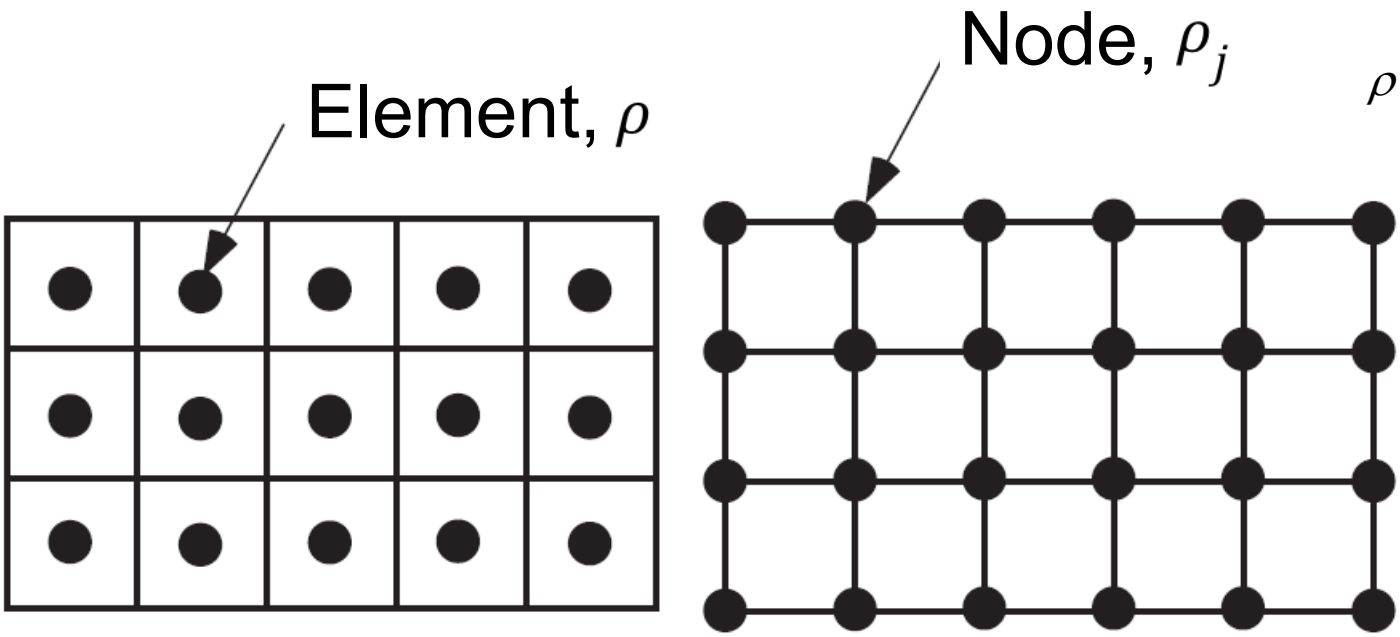
Topology Optimization: Implementation

- Optimization Process
- Algorithms
- Some Issues
  - Checkerboard pattern
  - Gray scale
  - Mesh dependency
- Matlab codes
  - 99 line (2001): top
  - 88 line (2011): top88
  - New 99 line (2020): top99neo, top3D125

Optimization Process



Step 2: Design Variables



- C^{-1}
- Checkerboard pattern

- C^{0}
- No checkerboard pattern
- Easy to extract the shape
- Gray scale

Step 3~5

Step 3: Finite Element Analysis

$$\mathbf{KU} = \mathbf{F}$$

$$\begin{cases} \mathbf{K}_e = \int_{\Omega_e} \mathbf{B}^T \mathbf{E} \mathbf{B} d\Omega \\ \mathbf{F}_e = \int_{\Omega_e} \mathbf{N}^T \mathbf{b} d\Omega + \int_{\Gamma_e} \mathbf{N}^T \mathbf{t} d\Gamma \\ \mathbf{B} = \partial \mathbf{N} \end{cases}$$

Step 4: Objective Function and Volume

$$F(\mathbf{U}) = -\frac{1}{2} l_{\text{m.c.}} = -\frac{1}{2} \mathbf{F}^T \mathbf{U} = -\frac{1}{2} \mathbf{U}^T \mathbf{K} \mathbf{U}$$

$$\begin{cases} \text{element: } \Omega = \sum_{e=1}^{n} \rho_e \Omega_e \\ \text{node: } \Omega = \int_D \rho d\Omega \end{cases}$$

1st order element → overestimate shear stress → checkerboard pattern
Need filtering scheme

Step 5: Convergence Check

$$\left\| F(\mathbf{U})^{(k+1)} - F(\mathbf{U})^{(k)} \right\| \le \varepsilon$$

Step 6: Sensitivity Analysis

Obejctive function

$$F(\mathbf{U}) = -\frac{1}{2} l_{\text{m.c.}} = -\frac{1}{2}\mathbf{F}^T\mathbf{U} = -\frac{1}{2}\left\{\mathbf{F}^T\mathbf{U} - \mathbf{U}^{*T}\left(\mathbf{K}\mathbf{U} - \mathbf{F}\right)\right\}$$

$$\frac{\partial F(\mathbf{U})}{\partial \rho_i} = -\frac{1}{2}\left\{\mathbf{F}^T\frac{\partial \mathbf{U}}{\partial \rho_i} - \mathbf{U}^{*T}\left(\frac{\partial \mathbf{K}}{\partial \rho_i}\mathbf{U} + \mathbf{K}\frac{\partial \mathbf{U}}{\partial \rho_i}\right)\right\} = -\frac{1}{2}\left\{\left(\mathbf{F}^T - \mathbf{U}^{*T}\mathbf{K}\right)\frac{\partial \mathbf{U}}{\partial \rho_i} - \mathbf{U}^{*T}\frac{\partial \mathbf{K}}{\partial \rho_i}\mathbf{U}\right\}$$

$$\mathbf{F}^T - \mathbf{U}^{*T}\mathbf{K} = \mathbf{0} \Leftrightarrow \mathbf{U}^* = \mathbf{U}\left(\text{self-adjoint}\right)$$

$$\rightarrow \frac{\partial F(\mathbf{U})}{\partial \rho_i} = \frac{1}{2}\mathbf{U}^{*T}\frac{\partial \mathbf{K}}{\partial \rho_i}\mathbf{U} \quad \text{where} \begin{cases} \text{element: } \dfrac{\partial \mathbf{K}}{\partial \rho_i} = \dfrac{\partial \mathbf{K}_i}{\partial \rho_i} = \displaystyle\int_{\Omega_i} \mathbf{B}^T\frac{\partial \mathbf{E}}{\partial \rho_i}\mathbf{B}\,d\Omega \\ \text{node: } \dfrac{\partial \mathbf{K}}{\partial \rho_i} = ? \end{cases}$$

Volume

$$\begin{cases} \text{element: } \dfrac{\partial \Omega}{\partial \rho_i} = \Omega_i \\ \text{node: } \dfrac{\partial \Omega}{\partial \rho_i} = ? \end{cases}$$

No need to solve the adjoint equation
Need U only related to ρ_i

Step 7: Updating Scheme

$$\begin{aligned} &\underset{\mathbf{x}}{\text{minimize}} \quad f(\mathbf{x}) \\ &\text{subject to} \quad g(\mathbf{x}) \le 0 \\ &\qquad\qquad x_i^L \le x_i \le x_i^U \quad (i = 1, \ldots, n) \end{aligned}$$

$$L = f(\mathbf{x}) + \Lambda g(\mathbf{x}) + \sum_{i=1}^{n} \left\{ \lambda_{-i} \left(-x_i + x_i^L \right) + \lambda_{+i} \left(x_i - x_i^U \right) \right\}$$

$$\begin{cases} \dfrac{\partial L}{\partial x_i} = \dfrac{\partial f(\mathbf{x})}{\partial x_i} + \Lambda \dfrac{\partial g(\mathbf{x})}{\partial x_i} - \lambda_{-i} + \lambda_{+i} = 0 \quad (i = 1, \ldots, n) \\ \dfrac{\partial L}{\partial \Lambda} = g(\mathbf{x}) = 0 \\ \dfrac{\partial L}{\partial \lambda_{-i}} = -x_i + x_i^L \le 0 \quad (i = 1, \ldots, n) \\ \dfrac{\partial L}{\partial \lambda_{+i}} = x_i - x_i^U \le 0 \quad (i = 1, \ldots, n) \\ \Lambda g(\mathbf{x}) = 0 \\ \lambda_{-i} \left(-x_i + x_i^L \right) = 0 \quad (i = 1, \ldots, n) \\ \lambda_{+i} \left(x_i - x_i^U \right) = 0 \quad (i = 1, \ldots, n) \\ \Lambda \ge 0, \ \lambda_{-i} \ge 0, \ \lambda_{+i} \ge 0 \quad (i = 1, \ldots, n) \end{cases}$$

I. OC: Optimality Criteria Method

$$\frac{\dfrac{\partial f(\mathbf{x})}{\partial x_i}}{-\Lambda\dfrac{\partial g(\mathbf{x})}{\partial x_i}} = 1 \quad (i = 1,\ldots,n) \rightarrow x_i^{(k+1)} = x_i^{(k)}\left[\frac{\partial f(\mathbf{x})/\partial x_i^{(k)}}{-\Lambda^{(k)}\partial g(\mathbf{x})/\partial x_i^{(k)}}\right]^{\eta} = x_i^{(k)}\left(A_i^{(k)}\right)^{\eta}$$

under the assumption that side constraints are inactive (possibility to be out of range)

$$x_i^{(k+1)} = \mathrm{Min}\left\{\mathrm{Max}\left\{x_i^{L^{(k)}}, x_i^{(k)}\left(A_i^{(k)}\right)^{\eta}\right\}, x_i^{U^{(k)}}\right\} = \begin{cases} x_i^{L^{(k)}} & \left(x_i^{(k)}\left(A_i^{(k)}\right)^{\eta} \le x_i^{L^{(k)}}\right) \\ x_i^{(k)}\left(A_i^{(k)}\right)^{\eta} & \left(x_i^{L^{(k)}} < x_i^{(k)}\left(A_i^{(k)}\right)^{\eta} < x_i^{U^{(k)}}\right) \\ x_i^{U^{(k)}} & \left(x_i^{(k)}\left(A_i^{(k)}\right)^{\eta} \ge x_i^{U^{(k)}}\right) \end{cases}$$

where $\begin{cases} x_i^{L^{(k)}} = \mathrm{Max}\left\{(1-\varsigma)x_i^{(k)}, x_i^{L^{(k)}}\right\} \\ x_i^{U^{(k)}} = \mathrm{Min}\left\{(1+\varsigma)x_i^{(k)}, x_i^{U^{(k)}}\right\} \end{cases}$

Good convergence for stiffness maximization Problems:
Λ = 0 (inactive volume constraint)
Positive sensitivity of objective

II. SLP: Sequential Linear Programming

- Linear approximation of functions at given point

$$\begin{cases} f(\mathbf{x}) \approx f(\mathbf{x}^0) + \sum_{i=1}^{n} \left.\frac{\partial f}{\partial x_i}\right|_{x_i = x_i^0} \left(x_i - x_i^0\right) \\ g(\mathbf{x}) \approx g(\mathbf{x}^0) + \sum_{i=1}^{n} \left.\frac{\partial g}{\partial x_i}\right|_{x_i = x_i^0} \left(x_i - x_i^0\right) \end{cases}$$

- Solve the LP problem within the move limit

$$(1-\varsigma)\, x_i \le x_i \le (1+\varsigma)\, x_i$$

III. Sequential Convex Programming (1)

- Introduction of the intermediate variable: $y_i = y_i(x_i)$
- Linear approximation at given point

$$\begin{cases} f(\mathbf{x}) \approx f(\mathbf{x}^0) + \sum_{i=1}^{n} \dfrac{\partial f}{\partial x_i} \dfrac{dx_i}{dy_i}\bigg|_{x_i = x_i^0} \left[y_i(x_i) - y_i(x_i^0) \right] \\ g(\mathbf{x}) \approx g(\mathbf{x}^0) + \sum_{i=1}^{n} \dfrac{\partial g}{\partial x_i} \dfrac{dx_i}{dy_i}\bigg|_{x_i = x_i^0} \left[y_i(x_i) - y_i(x_i^0) \right] \end{cases}$$

- Change the intermediate variable depending on the sign of sensitivity

$$f(\mathbf{x}) \approx f(\mathbf{x}^0) + \sum_{+} \frac{\partial f}{\partial x_i} \frac{dx_i}{dy_i^+}\bigg|_{x_i = x_i^0} \left(y_i^+(x_i) - y_i^+(x_i^0) \right) + \sum_{-} \frac{\partial f}{\partial x_i} \frac{dx_i}{dy_i^-}\bigg|_{x_i = x_i^0} \left(y_i^-(x_i) - y_i^-(x_i^0) \right)$$

III. Sequential Convex Programming (2)

- Convex approximation → separable → dual problem
  - CONLIN（Convex Linearization)

$$y_i = \begin{cases} y_i^+ = x_i & \text{if } \partial f / \partial x_i > 0 \\ y_i^- = 1/x_i & \text{if } \partial f / \partial x_i < 0 \end{cases} \quad \text{for } i=1,...,n$$

  - MMA(Method of Moving Asymptotes)

$$y_i = \begin{cases} y_i^+ = 1/(U_i - x_i) & \text{if } \partial f / \partial x_i > 0 \\ y_i^- = 1/(x_i - L_i) & \text{if } \partial f / \partial x_i < 0 \end{cases} \quad \text{for } i=1,...,n$$

$$L_i \to -\infty \text{ and } U_i \to +\infty : \text{ linear approximation}$$

$$L_i \to 0 \text{ and } U_i \to +\infty : \text{ CONLIN}$$

III. Sequential Convex Programming (3)

$$\frac{\partial L}{\partial x_i} = \left(f\left(\mathbf{x}^0\right) + \sum_{i=1}^{n} \frac{\partial f}{\partial x_i}\frac{dx_i}{dy_i}\bigg|_{x_i=x_i^0} \left[y_i\left(x_i\right) - y_i\left(x_i^0\right) \right] \right) + \Lambda \left(g\left(\mathbf{x}^0\right) + \sum_{i=1}^{n} \frac{\partial g}{\partial x_i}\frac{dx_i}{dy_i}\bigg|_{x_i=x_i^0} \left[y_i\left(x_i\right) - y_i\left(x_i^0\right) \right] \right)$$

$$+ \sum_{i=1}^{n} \left[\lambda_{-i}\left(-x_i + x_i^L\right) + \lambda_{+i}\left(x_i - x_i^U\right) \right]$$

$$= \left[f\left(\mathbf{x}^0\right) + \Lambda g\left(\mathbf{x}^0\right) \right] + \sum_{i=1}^{n} \left\{ \frac{\partial f}{\partial x_i}\frac{dx_i}{dy_i}\bigg|_{x_i=x_i^0} \left[y_i\left(x_i\right) - y_i\left(x_i^0\right) \right] + \Lambda \frac{\partial g}{\partial x_i}\frac{dx_i}{dy_i}\bigg|_{x_i=x_i^0} \left[y_i\left(x_i\right) - y_i\left(x_i^0\right) \right] \right\}$$

$$+ \sum_{i=1}^{n} \left[\lambda_{-i}\left(-x_i + x_i^L\right) + \lambda_{+i}\left(x_i - x_i^U\right) \right]$$

$$\underset{\mathbf{x}}{\text{minimize}} \quad l_i = \frac{\partial f}{\partial x_i}\frac{dx_i}{dy_i}\bigg|_{x_i=x_i^0} \left[y_i\left(x_i\right) - y_i\left(x_i^0\right) \right] + \Lambda \frac{\partial g}{\partial x_i}\frac{dx_i}{dy_i}\bigg|_{x_i=x_i^0} \left[y_i\left(x_i\right) - y_i\left(x_i^0\right) \right]$$

$$\text{subject to} \quad x_i^L \le x_i \le x_i^U \quad \left(i = 1, \ldots, n\right)$$

Dual Methods

- LP
  - Dual variables = Lagrange multipliers

$$\left.\begin{array}{l} < primal > \\ \min\ \boldsymbol{c}^T \boldsymbol{x} \\ \text{s.t.}\quad \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \geq 0 \\ \qquad \boldsymbol{x} \geq 0 \end{array}\right\} \leftrightarrow \left\{\begin{array}{l} < dual > \\ \min\ \boldsymbol{\lambda}^T \boldsymbol{b} \\ \text{s.t.}\quad \boldsymbol{\lambda}^T \boldsymbol{A} - \boldsymbol{c} \geq 0 \\ \qquad \boldsymbol{\lambda} \geq 0 \end{array}\right.$$

- NLP
  - Falk's dual formulation
  - Condition: convex optimization problem, functions to be twice differentiable, $\partial^2 L(\boldsymbol{x}, \boldsymbol{\lambda}) / \partial x^2$ to be nonsingula r @ $\boldsymbol{x}^*$

$$\left.\begin{array}{l} < primal > \\ \min\ f(\boldsymbol{x}) \\ \text{s.t.}\quad g_j(\boldsymbol{x}) \geq 0 \quad j = 1, \ldots, n_g \end{array}\right\} \leftrightarrow \left\{\begin{array}{l} < dual > \\ \max\ L_m(\boldsymbol{\lambda}) \\ \text{s.t.}\quad \lambda_j \geq 0 \quad j = 1, \ldots, n_g \end{array}\right.$$

Dual Methods: Separable Problems

– Both objective function and constraints are separable

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} f_i(x_i)$$

$$g_j(\boldsymbol{x}) = \sum_{i=1}^{n} g_{ji}(x_i) \quad j = 1, \ldots, n_g$$

– Benefit in the dual formulation: series of one-dimensional minimizations

  • Discrete design variables

$$x_i \in X_i = \{d_{i1}, \ldots\}, \quad i = 1, \ldots, n$$

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{i=1}^{n} \left[f_i(x_i) - \sum_{j=1}^{n_g} \lambda_j g_{ji}(x_i) \right] = \sum_{i=1}^{n} L_i(x_i, \boldsymbol{\lambda})$$

$$L_m(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \min_{x_i \in X_i} L_i(x_i) \rightarrow L_i(d_{ij}) = L_i(d_{i(j+1)}) : \text{boundaries in } \lambda\text{-space}$$

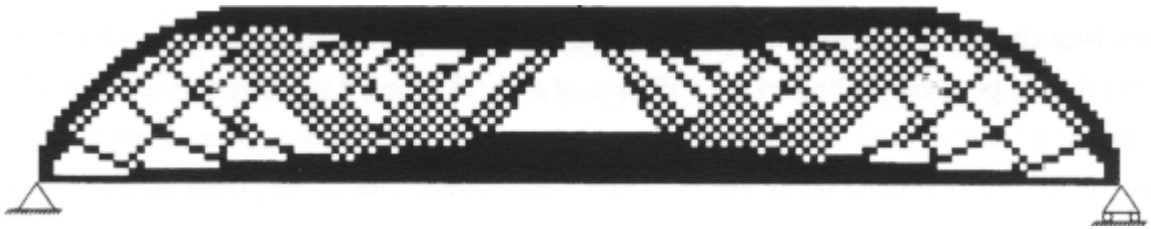
Summary of Optimization Methods

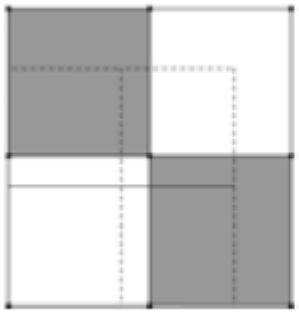
- Optimality Criteria Method (OC)
  - Appropriate for the problem where functions are monotone such as stiffness maximization
  - Difficult for the problem with stationary points
- Sequential Linear Programming (SLP)
  - Applicable to any kind of objective function
  - Convergence problem?
- Sequential Convex Programming
  - Applicable to any kind of objective function
  - Good convergence for stiffness maximization
  - Many parameters to set (may depend on the problem): MMA

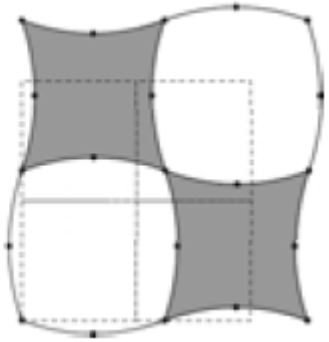
Difficulties in Topology Optimization

- Numerical instability
  - Checkerboard pattern
  - Gray scale
  - Mesh dependency
  - Complex sub-structure
- Reasons for numerical instability
  - Independent and constant design variable within a finite element
  - Characteristics of finite elements
- Method to overcome
  - Filtering, perimeter control
    - Difficult to determine parameters
  - Improvement of finite elements
    - High order element, non-conforming element

Checkerboard Problem (1)



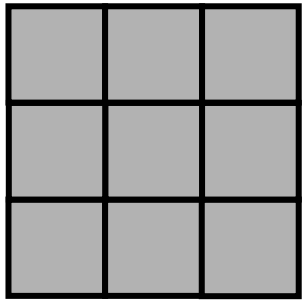




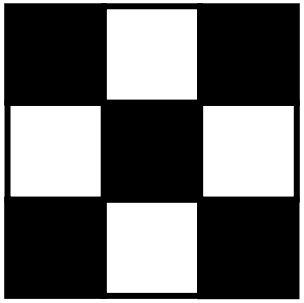
- Remedy
  - Higher order elements
  - Non-conforming elements
  - Nodal projection
  - Filtering
  - Constraints
- References
  - Diaz and Sigmund (1995)
  - Jog and Haber (1996)

Checkerboard Problem (2)

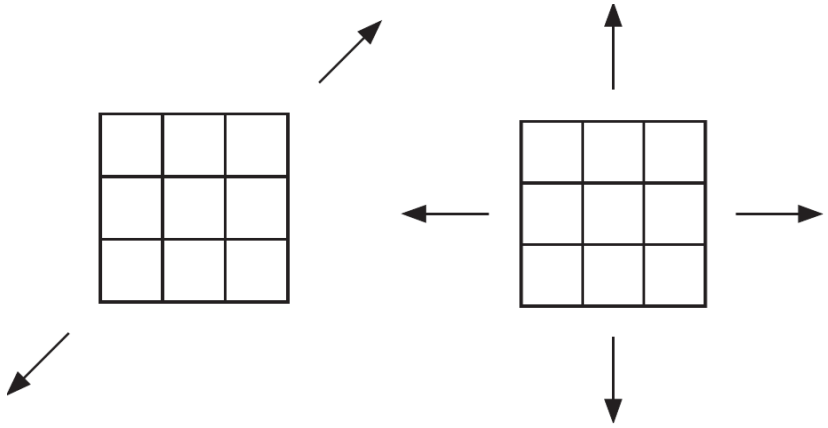
- Comparison of eigenvalues of stiffness matrix $K_e\phi = \lambda\phi$



$\rho = (5\times 1.0 + 4\times 0.001)/9$

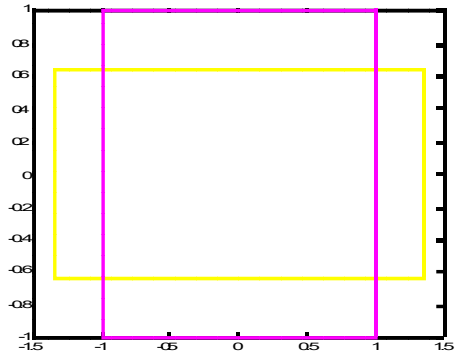


$E = 1.0,\ \nu = 0.3$

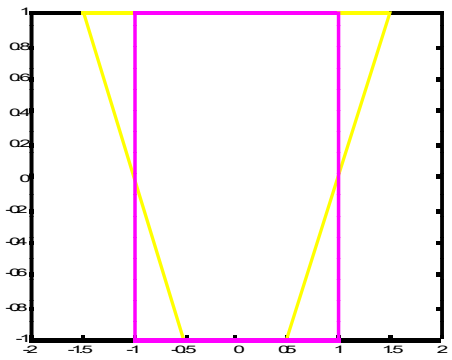


Eigenvalue (non-zero)	Uniform distribution	Checkerboard
Axial deformation	0.1230	0.0008
x-directional bending	0.1602	0.001
y-directional bending	0.1605	0.001
Shear deformation	0.2478	**0.3034**
Volumetric deformation	0.3109	**0.3711**

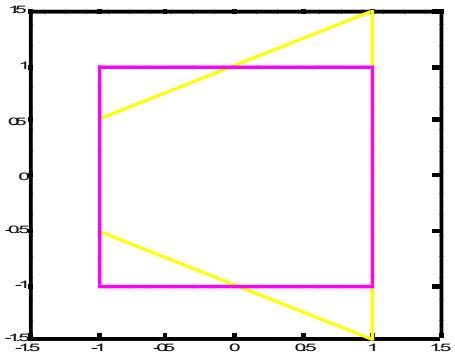
Eigenmode of Stiffness Matrix



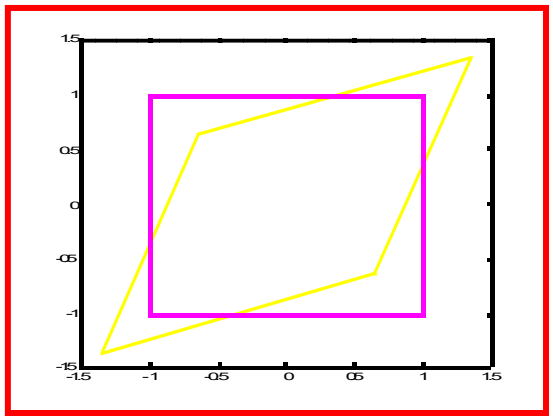
Axial deformation



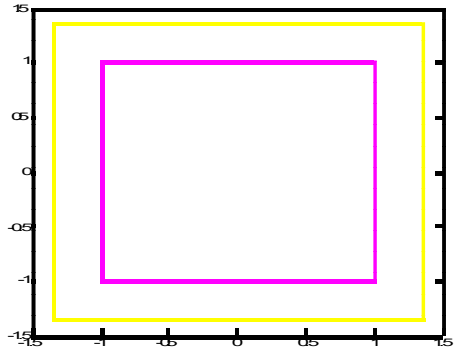
x-directional bending



y-directional bending



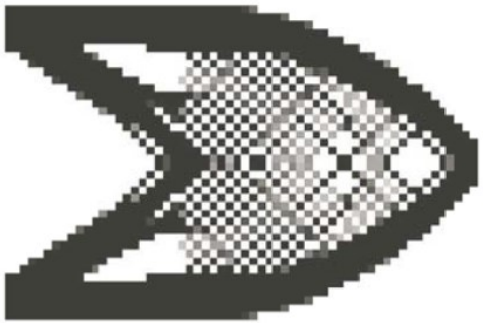
Shear deformation

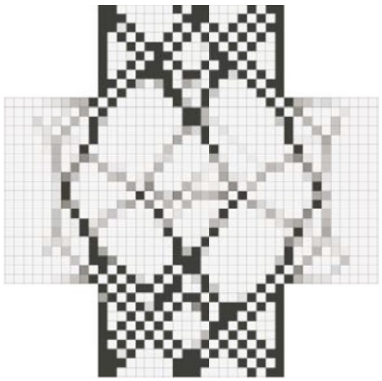


Volumetric deformation

Method to Prevent Checkerboard Pattern

- Finite element: shear stiffness < other stiffnesses
  - Selective reduced integration (SRI)
  - nine node element
- Design variable: node
- Filtering





Filtering Method (1)

- Basically employed to eliminate the mesh dependency → can eliminate checkerboard, too

$$\chi_\Omega \mathbf{E} \approx \big((\rho * K)(\mathbf{x})\big)^p \mathbf{E}$$

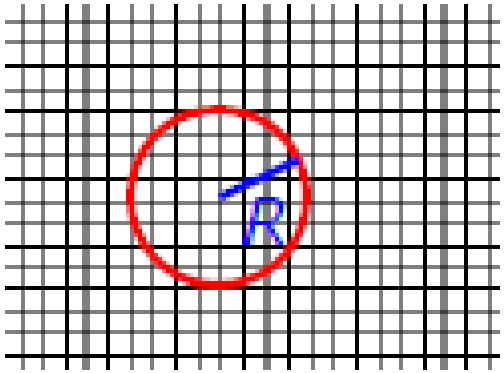
$$(\rho * K)(\mathbf{x}) = \int_\Omega \rho(\mathbf{y})\, K(\mathbf{x}-\mathbf{y}) dy$$

$$K(\mathbf{x}) = \begin{cases} 1 - \dfrac{\|\mathbf{x}\|}{r} & \textbf{if } \|\mathbf{x}\| < r \\ 0 & \textbf{otherwise} \end{cases}$$

- Weight sum of the value of nearby element when constructing stiffness matrix
- Worse convergence
- Unclear boundaries of optimal structure

Filtering Methods (2)

neighborhood:
$N_e = \{i | \|x_i - x_e\| \leq R\}$
where R is the filter radius
$H(x_i) = R - \|\boldsymbol{x}_j - \boldsymbol{x}_i\|$



sensitivity filtering: $\dfrac{\widetilde{\partial c}}{\partial \rho_e} = \dfrac{\sum_{i \in N_e} H(x_i)\rho_i \dfrac{\partial c}{\partial \rho_i}}{\rho_e \sum_{i \in N_e} H(x_i)}$

Sigmund (1994, 1996)

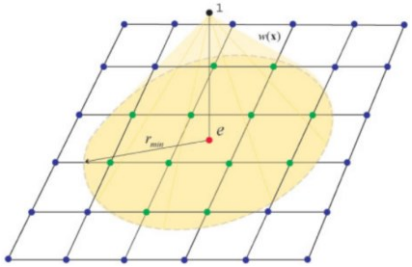
density filtering: $E_e(\rho) = \tilde{\rho}_e{}^p E_0$ where $\tilde{\rho}_e = \dfrac{\sum_{i \in N_e} H(x_i) A_i \rho_i}{\sum_{i \in N_e} H(x_i) A_i}$

Bruns and Tortorelli (2001), Bourdin (2001)

Filtering Methods (3)

sensitivity filter

$$\frac{\widetilde{\partial f}}{\partial \rho_i} = \frac{\sum_{j\in N_i} w(\boldsymbol{x}_j)\rho_j \frac{\partial f}{\partial \rho_i}/v_j}{\frac{\rho_i}{v_i}\sum_{j\in N_i} w(\boldsymbol{x}_j)} \quad \text{where } w(\boldsymbol{x}_j) = \begin{cases} R - \|\boldsymbol{x}_j - \boldsymbol{x}_i\| \\ exp\left[-\frac{1}{2}\left(\frac{\|\boldsymbol{x}_j - \boldsymbol{x}_i\|}{\sigma}\right)^2\right] \\ 1 \end{cases}$$



density filter

$$\widetilde{\rho_i} = \frac{\sum_{j\in N_i} w(x_j)v_j\rho_j}{\sum_{j\in N_i} w(x_j)v_j} \rightarrow \widetilde{\rho_i} = \frac{\sum_{j\in N_i} w(x_j)\widetilde{w}(\rho_j)v_j\rho_j}{\sum_{j\in N_i} w(x_j)\widetilde{w}(\rho_j)v_j} \text{ (Bi - lateral)}$$

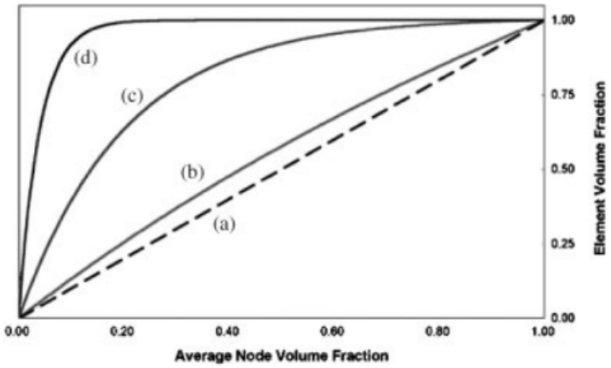
$$where\ \widetilde{w}(\rho_j) = exp\left[-\frac{1}{2}\left(\frac{\rho_j - \rho_i}{\sigma_r}\right)^2\right]$$

density filter with a Heaviside step function: 0/1 solutions

$$\overline{\rho_i} = \begin{cases} 1 - exp(-\beta\widetilde{\rho_i}) + \widetilde{\rho_i}exp(-\beta) \\ exp(1 - \beta\widetilde{\rho_i}) - \ (1 - \widetilde{\rho_i})exp(-\beta) \\ \frac{tanh(\beta\eta) + tanh[\beta(\widetilde{\rho_i} - \eta)]}{tanh(\beta\eta) + tanh[\beta(1 - \eta)]} \end{cases}$$

$$\rho \quad \rightarrow \quad \tilde{\rho}(\rho) \quad \rightarrow \overline{\rho_i}\left(\tilde{\rho}(\rho)\right)$$

$$\left\{\begin{matrix}\text{design} \\ \text{variable}\end{matrix}\right\} \rightarrow \left\{\begin{matrix}\text{filtered} \\ \text{density}\end{matrix}\right\} \rightarrow \{\text{projection}\}$$



The regularized Heaviside step function for various magnitudes of β: (a) $\beta = 0$ (linear); (b) $\beta = 1$; (c) $\beta = 5$; and (d) $\beta = 25$.

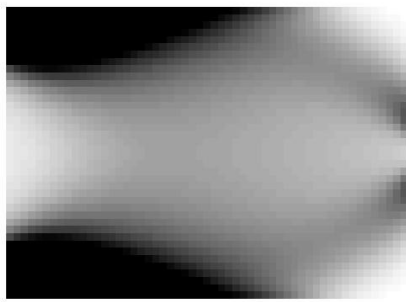
Gray Scale

- Layered microstructure
- Dynamic problem: eigenfrequency
- 3D problem
- Fundamentally difficult to avoid
  - physically meaningful, but engineering?

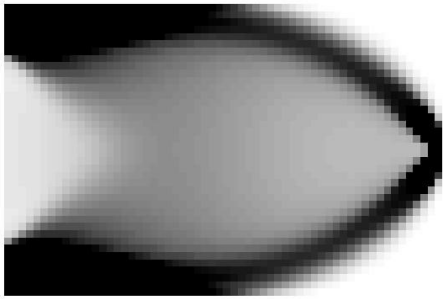
- Remedy
  - Change the microstructure
  - Increase the penalization parameter
  - Good(?) initial design
  - Mapping with Heaviside function
  - Paradigm change: level set method

Effect of Penalization Parameter (1)

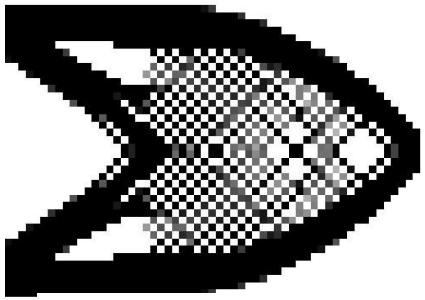
- Design variable: element
- No filtering vs Filtering



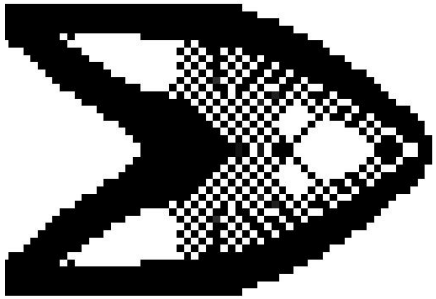
(a) $p = 0.5$



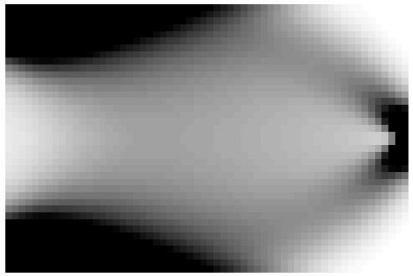
(b) $p = 1.0$

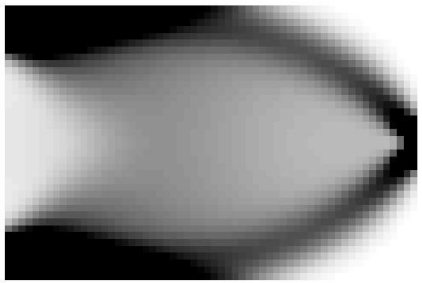


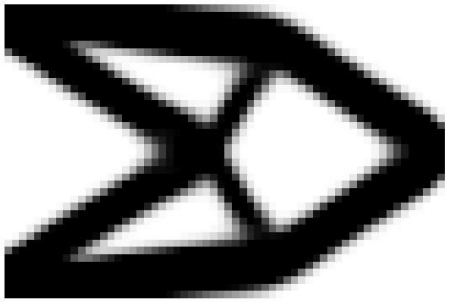
(c) $p = 2.0$

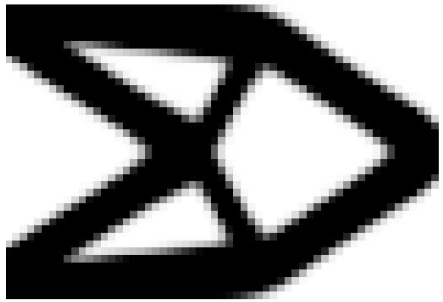


(d) $p = 3.0$



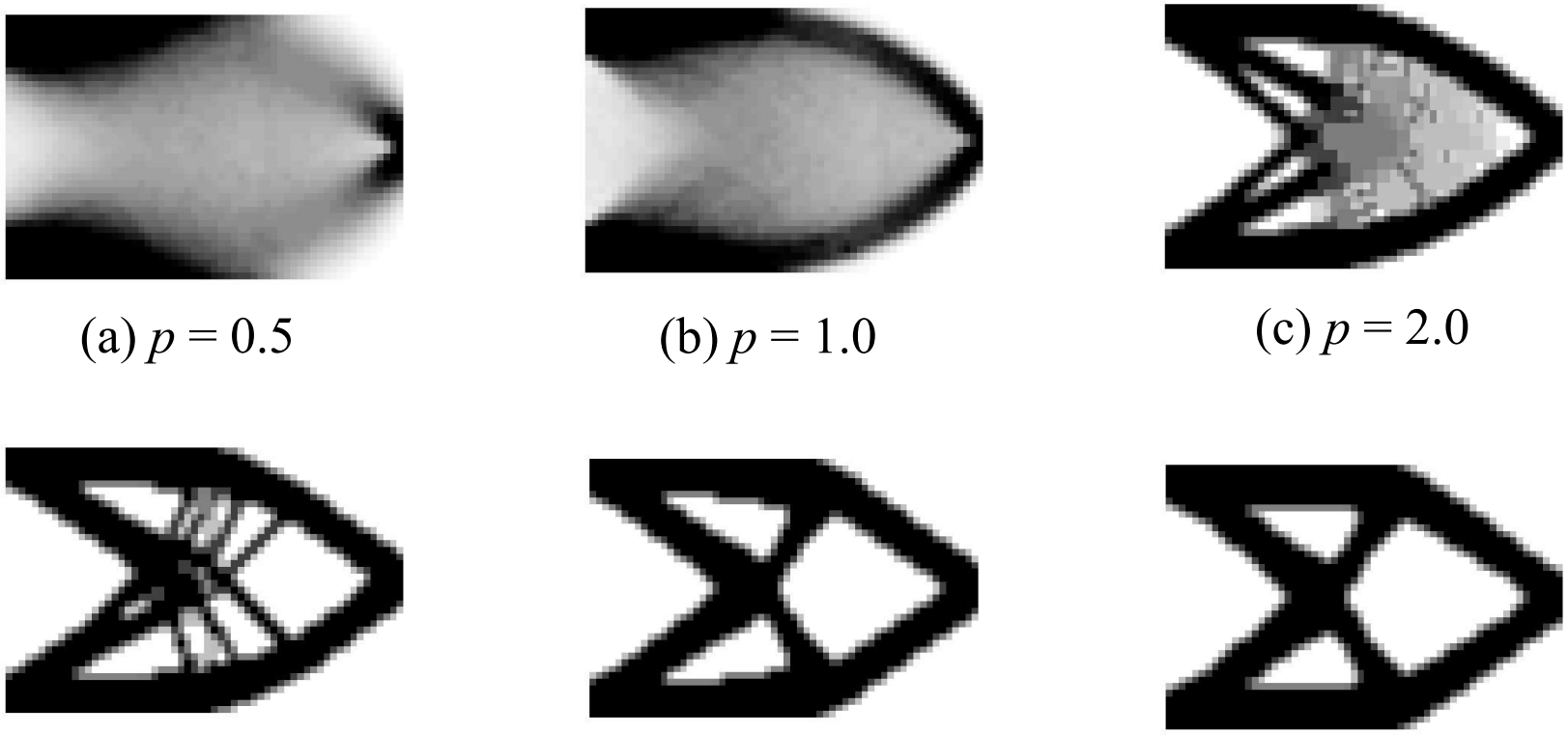






Effect of Penalization Parameter (2)

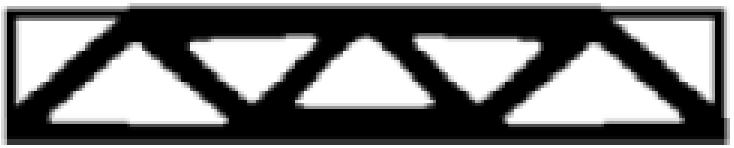
- Design variable: node
- No filtering



(a) $p = 0.5$ (b) $p = 1.0$ (c) $p = 2.0$

(d) $p = 3.0$ (e) $p = 4.0$ (f) $p = 5.0$

Mesh Dependency: Existence of Solutions

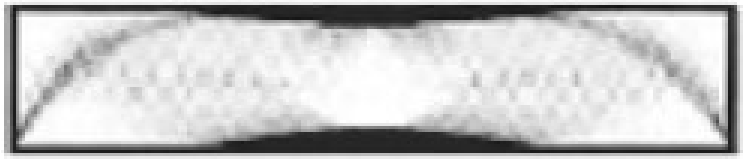
Good 

Better 

More details:
better structure

Better yet 

but more
expensive
to produce

Best 

structure with an optimal use of a composite

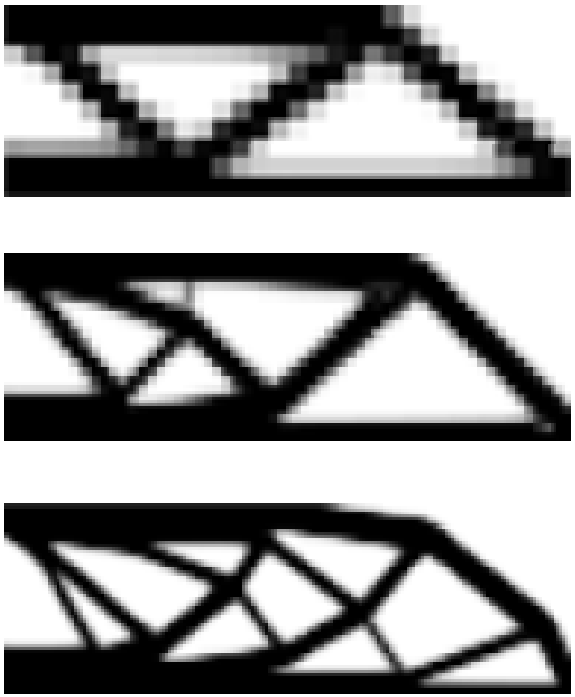
Mesh Dependency / Complex Sub-structure

60 x 40

120 x 80

150 x 100

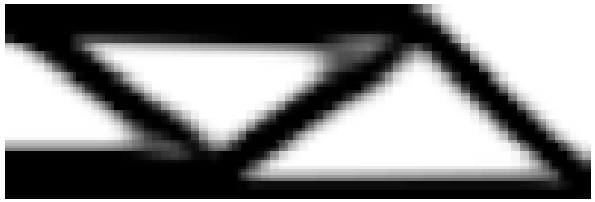
Elimination of Mesh Dependency



Mesh-dependency

Mesh refinement

Mesh refinement



Mesh-independency

perimeter control

$$P = \int_{\Omega \setminus \Gamma} |\nabla \rho| \, d\Omega + \int_{\Gamma} |\langle \rho \rangle| \, d\Gamma \leq P^*$$

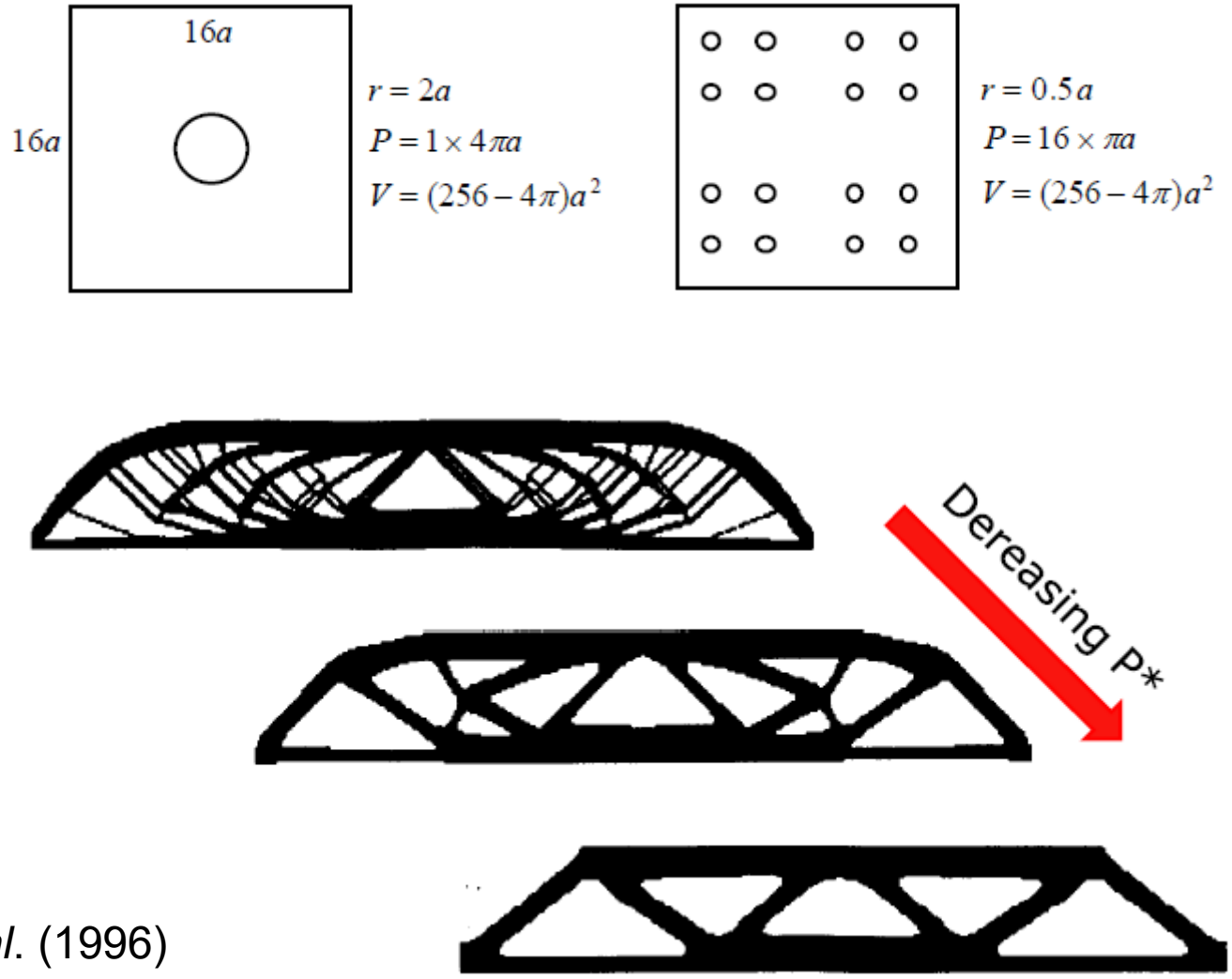
gradient control

$$\left| \frac{\partial \rho}{\partial x_i} \right| \leq C \quad i = 1, d$$

low−pass filtering

$$\tilde{\rho} = H(\rho), \frac{\partial \tilde{c}}{\partial \rho_e} = H\left(\frac{\partial c}{\partial \rho_e} \right)$$

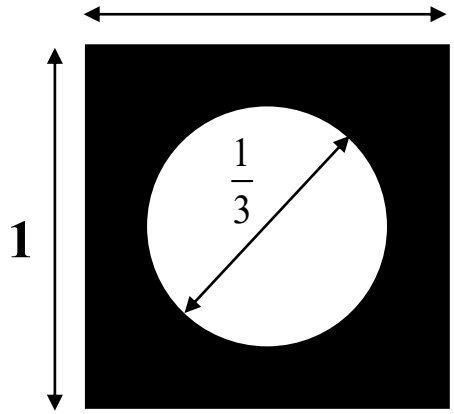
Perimeter Control



Haber *et al*. (1996)

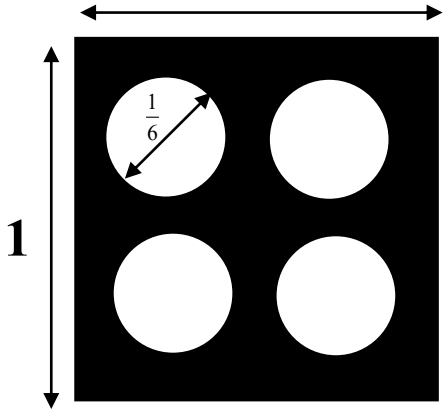
Perimeter Control (1)

- Perimeter = total length of the lines that form a shape
- Simple or complex structure by controlling the value of *P*: small ~ large



$$V = 1 - \pi\left(\frac{1}{3}\right)^2 \approx 0.65$$

$$P = 2\pi\frac{1}{3} \approx 2.09$$



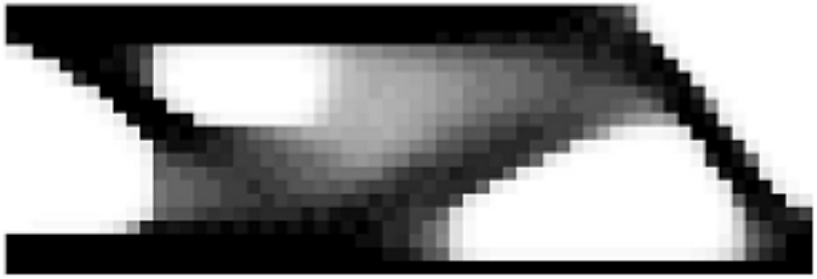
$$V = 1 - 4\pi\left(\frac{1}{6}\right)^2 \approx 0.65$$

$$P = 4 \cdot 2\pi\frac{1}{6} \approx 4.19$$

- In practice, total variation of density is used
  - Small P* → gray scale

$$TV(\rho) = \int_\Omega \|\nabla\rho\| dx \le P^* \quad \text{where} \quad P = \sum_{k=1}^{n} l_k \left(\sqrt{\left(\rho_i - \rho_j\right)^2 + \varepsilon^2} - \varepsilon \right)$$

Perimeter Control (2)

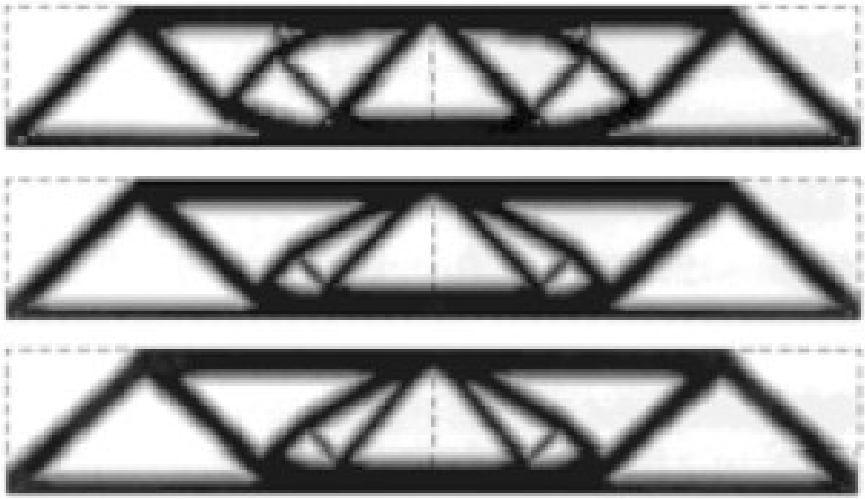


(a) $T_{\max} = 300$

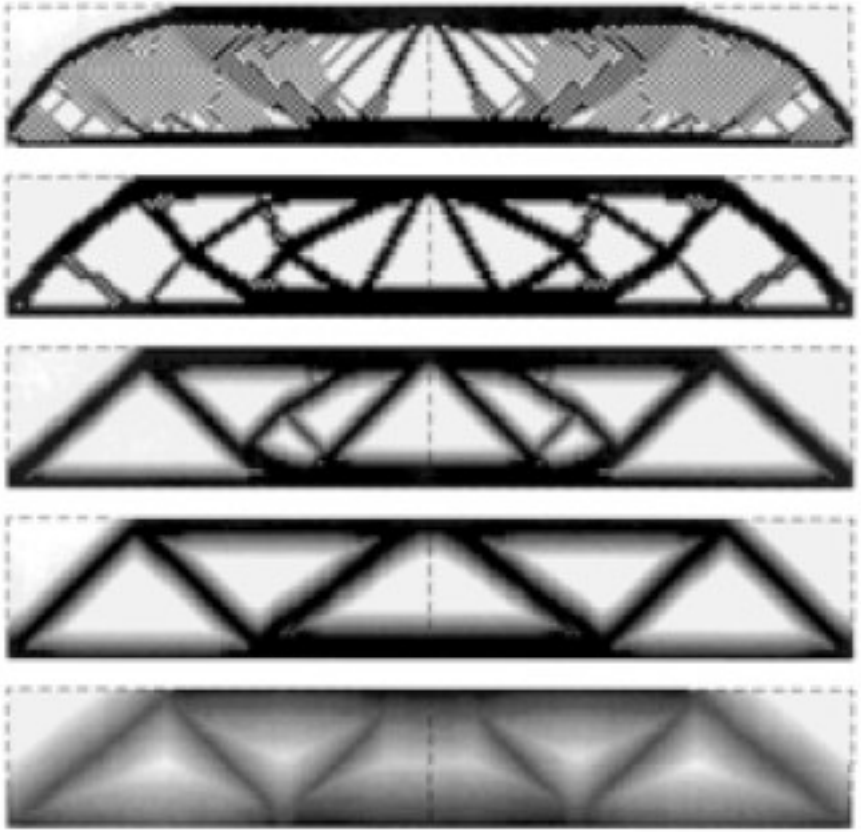
(b) $T_{\max} = 400$

(c) $T_{\max} = 500$

Gradient Control



Refining the mesh



Tightening the slope constraint

Niordson (1983) Petersson and Sigmund (1998)

99 Lines of Matlab Code (2001)

Educational article

Struct Multidisc Optim 21, 120–127

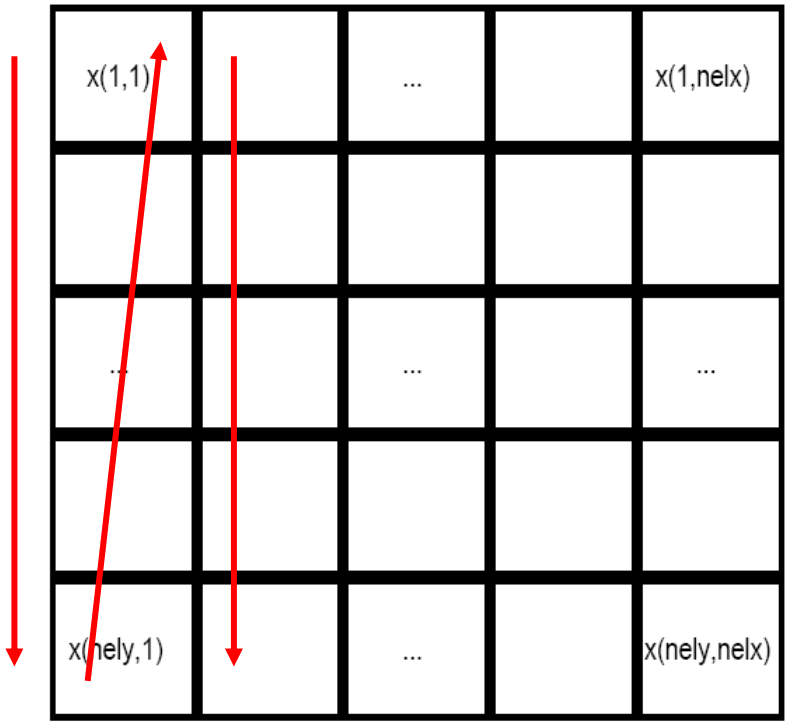
A 99 line topology optimization code written in Matlab

O. Sigmund

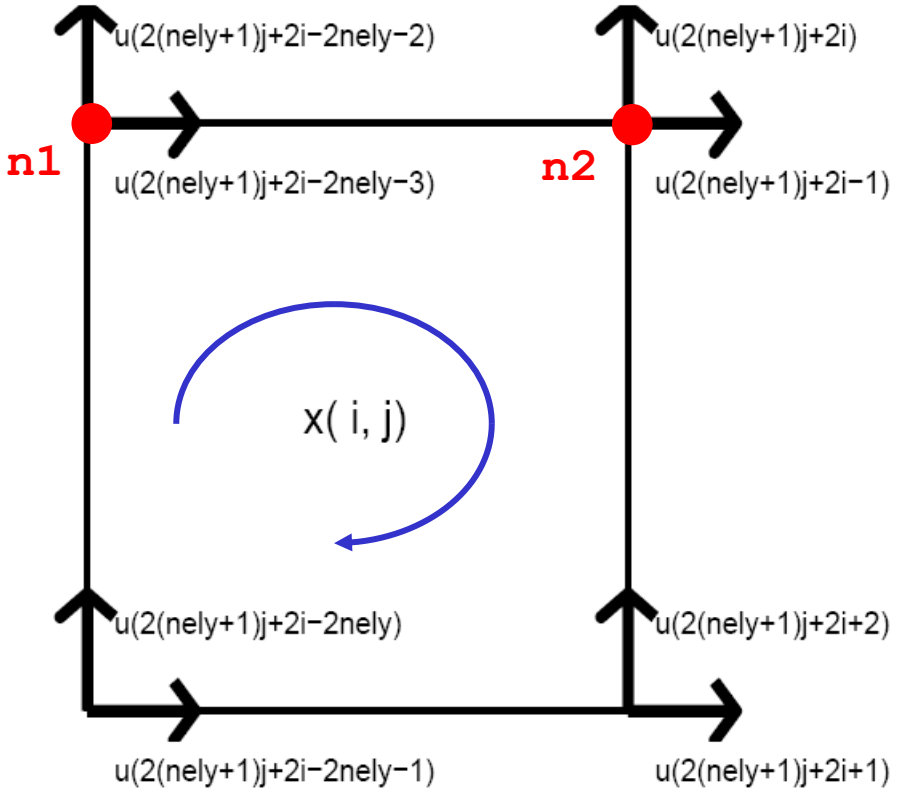
```
top(nelx,nely,volfrac,penal,rmin)
```

- nelx and nely: number of elements in the horizontal and vertical directions
- volfrac: volume fraction
- penal: penalization power
- rmin: filter size(divided by element size)

- Index order of the density matrix



- Displacements of a single finite element



Main program (lines 1–36)

```
 1 %%% A 99 LINE TOPOLOGY OPTIMIZATION CODE BY OLE
   SIGMUND, OCTOBER 1999 %%%
 2 function top(nelx,nely,volfrac,penal,rmin);
 3 % INITIALIZE
 4 x(1:nely,1:nelx) = volfrac;
 5 loop = 0;
 6 change = 1.;
 7 % START ITERATION
 8 while change > 0.01
 9   loop = loop + 1;
10   xold = x;
11 % FE-ANALYSIS
12   [U]=FE(nelx,nely,x,penal);
13 % OBJECTIVE FUNCTION AND SENSITIVITY ANALYSIS
14   [KE] = lk;
15   c = 0.;
16   for ely = 1:nely
17     for elx = 1:nelx
18       n1 = (nely+1)*(elx-1)+ely;
19       n2 = (nely+1)* elx   +ely;
20       Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;
              2*n2+2; 2*n1+1;2*n1+2],1);
21       c = c + x(ely,elx)^penal*Ue'*KE*Ue;
22       dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*
                       Ue'*KE*Ue;
23     end
24   end
25 % FILTERING OF SENSITIVITIES
26   [dc]   = check(nelx,nely,rmin,x,dc);
27 % DESIGN UPDATE BY THE OPTIMALITY CRITERIA METHOD
28   [x]    = OC(nelx,nely,x,volfrac,dc);
29 % PRINT RESULTS
30   change = max(max(abs(x-xold)));
31   disp([' It.: ' sprintf('%4i',loop) ' Obj.: '
          sprintf('%10.4f',c) ...
32        ' Vol.: ' sprintf('%6.3f',sum(sum(x))/
            (nelx*nely)) ...
33         ' ch.: ' sprintf('%6.3f',change )])
34 % PLOT DENSITIES
35   colormap(gray); imagesc(-x); axis equal; axis
     tight; axis off;pause(1e-6);
36 end
```

$$\left.\begin{aligned} \min_{\mathbf{x}}: \quad & c(\mathbf{x}) = \mathbf{U}^T\mathbf{K}\mathbf{U} = \sum_{e=1}^{N} (x_e)^p \, \mathbf{u}_e^T \, \mathbf{k}_0 \, \mathbf{u}_e \\ \text{subject to}: \quad & \frac{V(\mathbf{x})}{V_0} = f \\ & : \quad \mathbf{K}\mathbf{U} = \mathbf{F} \\ & : \quad \mathbf{0} < \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{1} \end{aligned}\right\}$$

$$\frac{\partial c}{\partial x_e} = -p(x_e)^{p-1} \, \mathbf{u}_e^T \, \mathbf{k}_0 \, \mathbf{u}_e$$

Optimality criteria based optimizer (lines 37–48)

```
37 %%%%%%%%% OPTIMALITY CRITERIA UPDATE %%%%%%%%%
38 function [xnew]=OC(nelx,nely,x,volfrac,dc)
39 l1 = 0; l2 = 100000; move = 0.2;
40 while (l2-l1 > 1e-4)
41   lmid = 0.5*(l2+l1);
42   xnew = max(0.001,max(x-move,min(1.,min(x+move,x.
           *sqrt(-dc./lmid)))));
43   if sum(sum(xnew)) - volfrac*nelx*nely > 0;
44     l1 = lmid;
45   else
46     l2 = lmid;
47   end
48 end
```

bi-sectioning algorithm to find the Lagrangian multiplier

Heuristic design updates:

$$x_e^{\text{new}} = \begin{cases} \max(x_{\min}, x_e - m) & \text{if} \quad x_e B_e^{\eta} \leq \max(x_{\min}, x_e - m), \\ x_e B_e^{\eta} & \text{if} \quad \max(x_{\min}, x_e - m) < x_e B_e^{\eta} < \min(1, x_e + m), \\ \min(1, x_e + m) & \text{if} \quad \min(1, x_e + m) \leq x_e B_e^{\eta}, \end{cases}$$

$$B_e = \frac{-\frac{\partial c}{\partial x_e}}{\lambda \frac{\partial V}{\partial x_e}}$$

→ Adjusted until volume constraint fulfilled

Mesh-independency filtering (lines 49–64)

```
49 %%%%%%%%%% MESH-INDEPENDENCY FILTER %%%%%%%%%%
50 function [dcn]=check(nelx,nely,rmin,x,dc)
51 dcn=zeros(nely,nelx);
52 for i = 1:nelx
53   for j = 1:nely
54     sum=0.0;
55     for k = max(i-round(rmin),1):
           min(i+round(rmin),nelx)
56       for l = max(j-round(rmin),1):
         min(j+round(rmin), nely)
57         fac = rmin-sqrt((i-k)^2+(j-l)^2);
58         sum = sum+max(0,fac);
59         dcn(j,i) = dcn(j,i) + max(0,fac)*x(l,k)
           *dc(l,k);
60       end
61     end
62     dcn(j,i) = dcn(j,i)/(x(j,i)*sum);
63   end
64 end
```

nearest integers

$$\widehat{\frac{\partial c}{\partial x_e}} = \frac{1}{x_e \sum_{f=1}^{N} \hat{H}_f} \sum_{f=1}^{N} \hat{H}_f \, x_f \, \frac{\partial c}{\partial x_f}$$

$$\hat{H}_f = r_{\min} - \operatorname{dist}(e, f)\,,$$

$$\{f \in N \mid \operatorname{dist}(e, f) \leq r_{\min}\}, \quad e = 1, \ldots, N$$

Finite element code (lines 65–99)

```
65 %%%%%%%%%% FE-ANALYSIS %%%%%%%%%%%%
66 function [U]=FE(nelx,nely,x,penal)
67 [KE] = lk;
68 K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*
       (nely+1));
69 F = sparse(2*(nely+1)*(nelx+1),1); U =
       sparse(2*(nely+1)*(nelx+1),1);
70 for ely = 1:nely
71   for elx = 1:nelx
72     n1 = (nely+1)*(elx-1)+ely;
73     n2 = (nely+1)* elx   +ely;
74     edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1;
              2*n2+2;2*n1+1; 2*n1+2];
75     K(edof,edof) = K(edof,edof) +
              x(ely,elx)^penal*KE;
76   end
77 end
78 % DEFINE LOADS AND SUPPORTS (HALF MBB-BEAM)
79 F(2,1) = -1;
80 fixeddofs   = union([1:2:2*(nely+1)],
              [2*(nelx+1)*(nely+1)]);
81 alldofs     = [1:2*(nely+1)*(nelx+1)];
82 freedofs    = setdiff(alldofs,fixeddofs);
83 % SOLVING
84 U(freedofs,:) = K(freedofs,freedofs) \
                   F(freedofs,:);
85 U(fixeddofs,:)= 0;
```

```
86 %%%%%%%%%% ELEMENT STIFFNESS MATRIX %%%%%%%
87 function [KE]=lk
88 E = 1.;
89 nu = 0.3;
90 k=[ 1/2-nu/6  1/8+nu/8 -1/4-nu/12 -1/8+3*nu/8 ...
91    -1/4+nu/12 -1/8-nu/8  nu/6       1/8-3*nu/8];
92 KE = E/(1-nu^2)*
     [ k(1) k(2) k(3) k(4) k(5) k(6) k(7) k(8)
93     k(2) k(1) k(8) k(7) k(6) k(5) k(4) k(3)
94     k(3) k(8) k(1) k(6) k(7) k(4) k(5) k(2)
95     k(4) k(7) k(6) k(1) k(8) k(3) k(2) k(5)
96     k(5) k(6) k(7) k(8) k(1) k(2) k(3) k(4)
97     k(6) k(5) k(4) k(3) k(2) k(1) k(8) k(7)
98     k(7) k(4) k(5) k(2) k(3) k(8) k(1) k(6)
99     k(8) k(3) k(2) k(5) k(4) k(7) k(6) k(1)];
```

alldofs - fixeddofs

Plotting Displacements

Plotting displacements

Insert the following lines in your program instead of the current plotting line:

```
% colormap(gray); imagesc(-x); axis equal; axis tight; axis off;pause(1e-6);
colormap(gray); axis equal;
for ely = 1:nely
  for elx = 1:nelx
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx +ely;
    Ue = 0.005*U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
    ly = ely-1; lx = elx-1;
    xx = [Ue(1,1)+lx Ue(3,1)+lx+1 Ue(5,1)+lx+1 Ue(7,1)+lx ]';
    yy = [-Ue(2,1)-ly -Ue(4,1)-ly -Ue(6,1)-ly-1 -Ue(8,1)-ly-1]';
    patch(xx,yy,-x(ely,elx))
  end
end
drawnow; clf;
```

A Matlab threshold code

The Matlab script shown below is intended as a post-processing step that converts a grey scale design obtained with the 99-line code (Sigmund 2001a) to a discrete design satisfying the volume fraction constraint.

In the script, the total volume includes the volume taken up by low-density elements. If the discrete approach does not include low-density elements, the third line above can simply be changed to

```
vt = floor(volfrac*nelx*nely);
```

'For the more compact 88-line code (Andreassen et al. 2011) the FE-part of above script should be substituted with the following

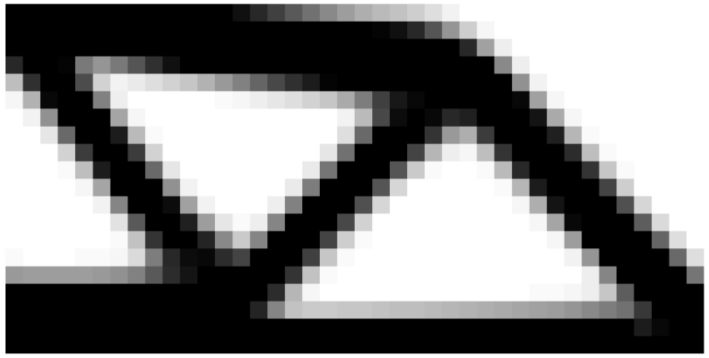
```
%% FE-ANALYSIS
sK = reshape(KE(:)*(0.001+xd(:)'*(1-0.001)),64*nelx*nely,1);
K = sparse(iK,jK,sK); K = (K+K')/2;
U(freedofs) = K(freedofs,freedofs)\F(freedofs);
ce = reshape(sum((U(edofMat)*KE).*U(edofMat),2),nely,nelx);
cd = sum(sum((Emin+xd.^penal*(E0-0.001)).*ce))
```

```
% THRESHOLD DESIGN SATISFYING VOLUME CONSTRAINT
[Y,I]=sort(x(:),'descend');
vt = floor(((volfrac-0.001)*nelx*nely)/(1-0.001));
xd(I(1:vt))=1;
xd(I(vt+1:end))=0.001;
xd=reshape(xd,nely,nelx);
imagesc(-xd); axis equal; axis off;
% FE-ANALYSIS
[U]=FE(nelx,nely,xd,penal);
% CALCULATE DISCRETE OBJECTIVE FUNCTION
cd = 0.;
for ely = 1:nely
 for elx = 1:nelx
  n1 = (nely+1)*(elx-1)+ely;
  n2 = (nely+1)* elx   +ely;
  Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2;
          2*n2+1;2*n2+2; 2*n1+1;2*n1+2],1);
  cd = cd + xd(ely,elx)^penal*Ue'*KE*Ue;
 end
end
disp(['  Discrete Obj.: ' sprintf('%10.4f',cd) ...
 ' Vol.: ' sprintf('%6.3f',sum(xd(:))/(nelx*nely))])
```

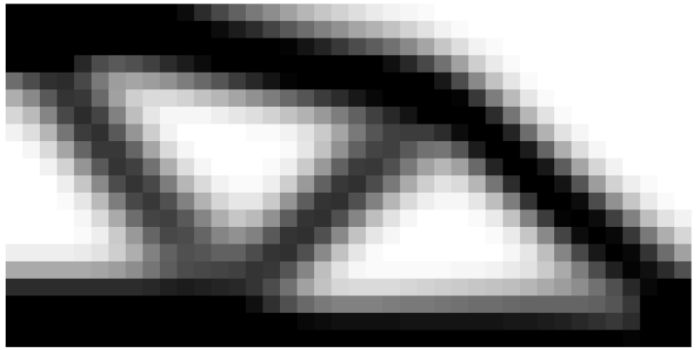
Checkerboard Pattern (1)

- top(40,20,0.5,3,1.0)
  - Obj: 80.4086

- top(40,20,0.5,3,1.5)
  - Obj: 82.7562

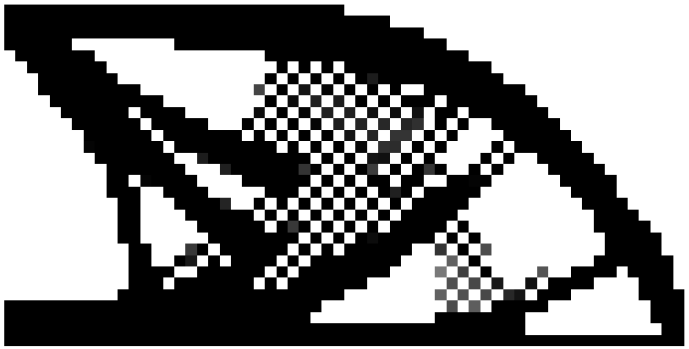


- top(40,20,0.5,3,3.0)
  - Obj: 99.1929

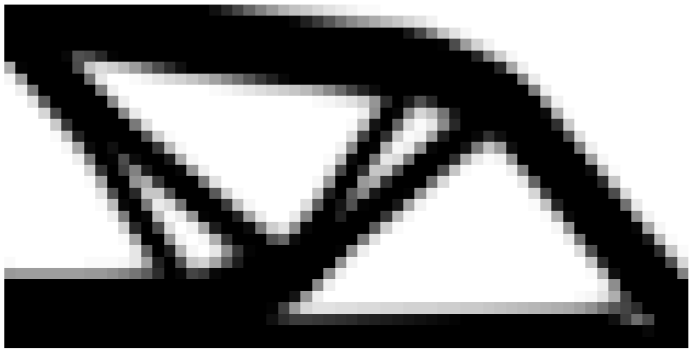


Checkerboard Pattern (2)

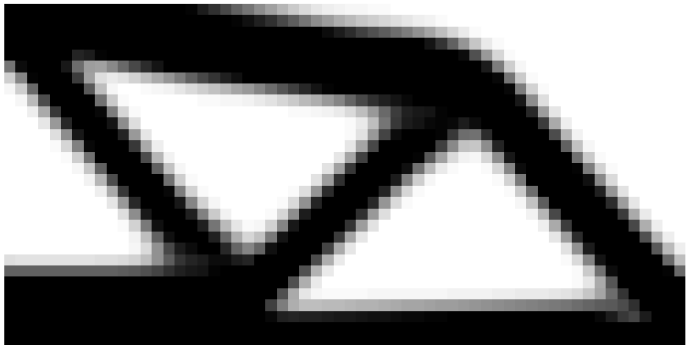
- top(60,30,0.5,3,1.0)
  - Obj: 83.0834



- top(60,30,0.5,3,1.5)
  - Obj: 81.3491



- top(60,30,0.5,3,2.25)
  - Obj: 83.5963

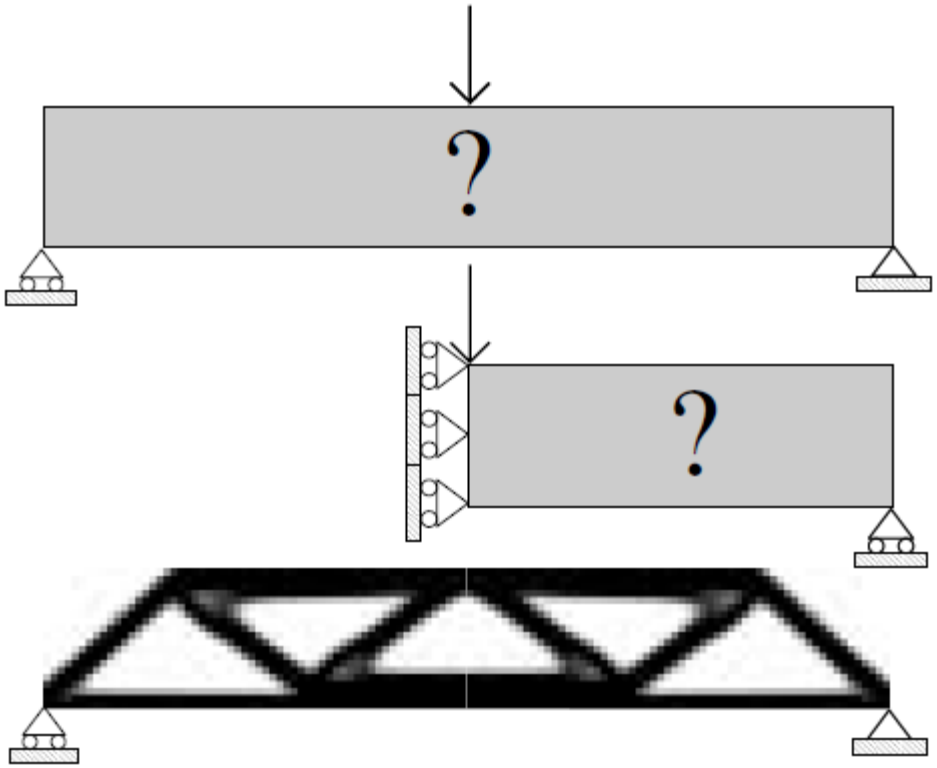


Extensions

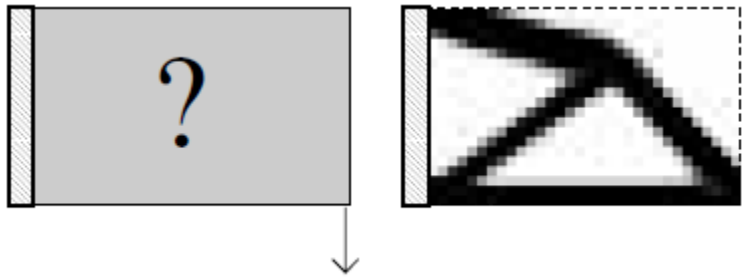
- Investigations
  - Filter size: rmin
  - Penalization power: penal

- Other boundary conditions
- Multiple load cases
- Passive elements
- Alternative optimizer
  - Optimality Criteria (OC) methods
  - Sequential Linear Programming (SLP) methods
  - Method of Moving Asymptotes (MMA)

Other boundary conditions

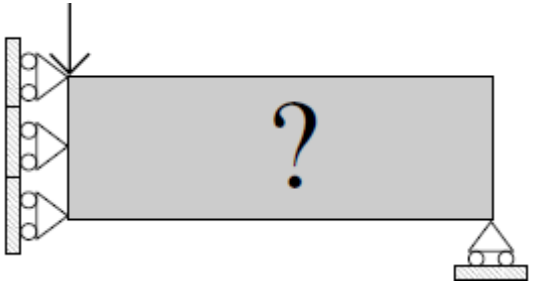


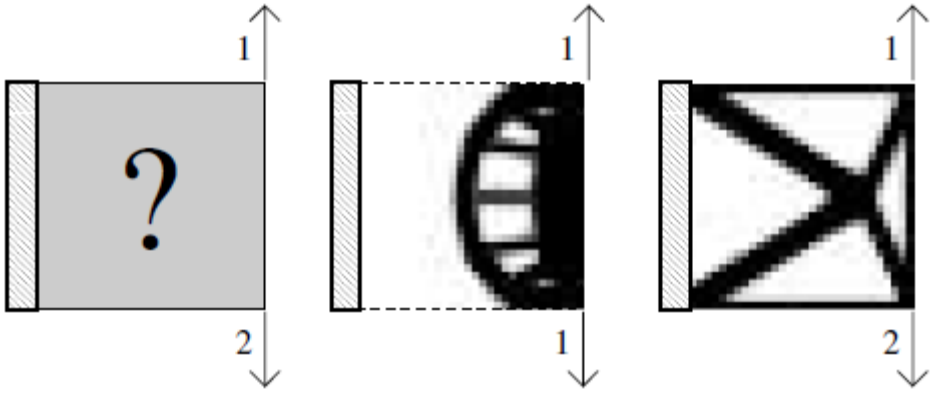

```
79 F(2,1) = -1;
80 fixeddofs   = union([1:2:2*(nely+1)],
                 [2*(nelx+1)*(nely+1)]);
```




```
79 F(2*(nelx+1)*(nely+1),1) = -1;
80 fixeddofs   = [1:2*(nely+1)];
```

Multiple load cases






```
69 F = sparse(2*(nely+1)*(nelx+1),1); U =
       sparse(2*(nely+1)*(nelx+1),1);
79 F(2,1) = -1;
```

```
69 F = sparse(2*(nely+1)*(nelx+1),2);
   U = sparse(2*(nely+1)*(nelx+1),2);
79 F(2*(nelx+1)*(nely+1),1) = -1.;
             F(2*(nelx)*(nely+1)+2,2) = 1.;
```

```
20      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2; 2*n2+1;
             2*n2+2; 2*n1+1;2*n1+2],1);
21      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
22      dc(ely,elx) = -penal*x(ely,elx)^(penal-1)*
                  Ue'*KE*Ue;
```

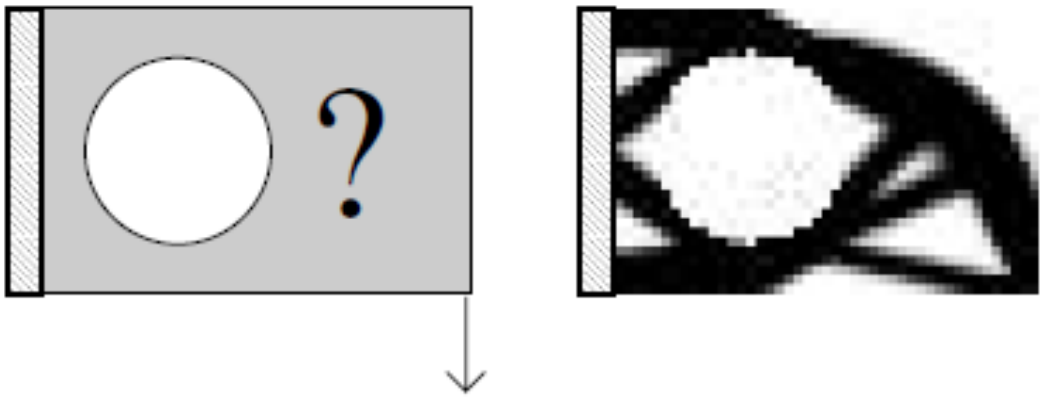
```
19b   dc(ely,elx) = 0.;
19c   for i = 1:2
20      Ue = U([2*n1-1;2*n1; 2*n2-1;2*n2;
             2*n2+1;2*n2+2;2*n1+1;2*n1+2],i);
21      c = c + x(ely,elx)^penal*Ue'*KE*Ue;
22      dc(ely,elx) = dc(ely,elx) -
           penal*x(ely,elx)^(penal-1)*Ue'*KE*Ue;
22b   end
```

Passive elements

```
42  xnew = max(0.001,max(x-move,min(1.,min(x+move,x.
          *sqrt(-dc./lmid)))));

42b   xnew(find(passive)) = 0.001;

      for ely = 1:nely
        for elx = 1:nelx
          if sqrt((ely-nely/2.)^2+(elx-nelx/3.)^2) <
                                                nely/3.
            passive(ely,elx) = 1;
            x(ely,elx) = 0.001;
          else
            passive(ely,elx) = 0;
          end
        end
      end
```



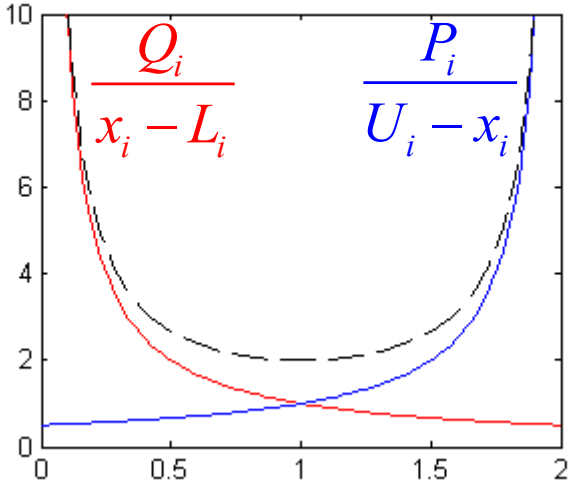
Mathematical Programming (1)

- Solve smooth, non-linear optimization problems
  - Sequence of simpler approximate subproblems
  - SLP, SQP
- Method of Moving Asymptotes (MMA) ← CONLIN
  - Convex and separable approximations
  - Sensitivity information at a current iteration point as well as some iteration history
  - Subproblem solution: dual method, interior point (primal-dual) algorithm

$$F(\boldsymbol{x}) \approx F(\boldsymbol{x}^0) + \sum_{i=1}^{n} \left(\frac{r_i}{U_i - x_i} + \frac{s_i}{x_i - L_i} \right)$$

$$\begin{cases} \dfrac{\partial F}{\partial x_i}(\boldsymbol{x}^0) > 0 \rightarrow r_i = (U_i - x_i^0)^2 \dfrac{\partial F}{\partial x_i}(\boldsymbol{x}^0) \text{ and } s_i = 0 \\ \dfrac{\partial F}{\partial x_i}(\boldsymbol{x}^0) < 0 \rightarrow r_i = 0 \text{ and } s_i = -(x_i^0 - L_i)^2 \dfrac{\partial F}{\partial x_i}(\boldsymbol{x}^0) \end{cases}$$

U_i, L_i : vertical asymptotes for the approximation F to control the range



Mathematical Programming (2)

- MMA approximation of the compliance

$$\min_{\rho_e} c\left(\rho_e\right) \xrightarrow{\frac{\partial c}{\partial \rho_e}\left(\rho^K\right)<0} \min_{\rho_e}\left\{c\left(\rho^K\right)-\sum_{e=1}^{N}\frac{\left(\rho_e^K-L_e\right)^2}{\rho_e-L_e}\frac{\partial c}{\partial \rho_e}\left(\rho^K\right)\right\}$$

- Solve this problem by dual method ↔ OC
  - (1) minimize the Lagrange functional ↔ update scheme
  - (2) maximize the resulting functional with respect to Λ ↔ adjust the value to satisfy the volume constraint

$$\text{for } \rho_{\min} \le \rho_e \le 1,\ e=1,\ldots,N$$

$$L=c\left(\rho^K\right)-\sum_{e=1}^{N}\frac{\left(\rho_e^K-L_e\right)^2}{\rho_e-L_e}\frac{\partial c}{\partial \rho_e}\left(\rho^K\right)+\Lambda\left(\sum_{e=1}^{N}v_e\rho_e-V\right)$$

$$L_e=0 \text{ and } \zeta=\infty \rightarrow \text{ optimality criteria update scheme}$$

Updating Design Variables (1)

$$\min_x f_0(x)$$
$$\text{s.t. } f_i(x) \le 0, \quad i = 1,\ldots,m$$
$$x \in \chi = \left\{ x \in \Re^n, x_j^{\min} \le x_j \le x_j^{\max} \right\}$$

[SLP subproblem at iteration k]

$$\min_x f_0(x^k) + \nabla f_0(x^k)^T (x - x^k)$$
$$\text{s.t. } f_i(x^k) + \nabla f_i(x^k)^T (x - x^k) \le 0, \quad i = 1,\ldots,m$$
$$l_j^k \le x_j - x_j^k \le u_j^k, \quad j = 1,\ldots,n$$
$$x \in \chi = \left\{ x \in \Re^n, x_j^{\min} \le x_j \le x_j^{\max} \right\}$$

[SQP subproblem at iteration k]

$$\min_x f_0(x^k) + \nabla f_0(x^k)^T (x - x^k) + \frac{1}{2}(x - x^k)^T H(x^k)(x - x^k)$$
$$\text{s.t. } f_i(x^k) + \nabla f_i(x^k)^T (x - x^k) \le 0, \quad i = 1,\ldots,m$$
$$x \in \chi = \left\{ x \in \Re^n, x_j^{\min} \le x_j \le x_j^{\max} \right\}$$

Updating Design Variables (2)

[Convex linearization (CONLIN) subproblem at iteration k]

$$\min_x f_0^{C,k}(x)$$
$$\text{s.t. } f_i^{C,k}(x) \le 0,\ i = 1, \dots, m$$
$$x \in \chi = \left\{ x \in \Re^n, x_j^{min} \le x_j \le x_j^{max} \right\}$$

where

$$f_i^{C,k}(x) \doteq f_i\left(x^k\right) + \sum_{j \in \Omega_+} f_{ij}^{L,k}(x) + \sum_{j \in \Omega_-} f_{ij}^{R,k}(x)$$

$$\Omega_+ \doteq \left\{ j : \frac{\partial f_i\left(x^k\right)}{\partial x_j} > 0 \right\} \text{ and } \Omega_- \doteq \left\{ j : \frac{\partial f_i\left(x^k\right)}{\partial x_j} \le 0 \right\}$$

$$f_{ij}^{L,k}(x) = \frac{\partial f_i\left(x^k\right)}{\partial x_j}\left(x_j - x_j^k\right) \text{ and } f_{ij}^{R,k}(x) = \frac{\partial f_i\left(x^k\right)}{\partial x_j}\frac{x_j^k\left(x_j - x_j^k\right)}{x_j}$$

Updating Design Variables (3)

[Method of Moving Asymptotes (MMA) subproblem at iteration k]

$$\min_x f_0^{M,k}(x)$$

$$\textbf{s.t. } f_i^{M,k}(x) \le 0, \qquad i = 1, \dots, m$$

$$\alpha_j^k \le x_j \le \beta_j^k, \qquad j = 1, \dots, n$$

$$x \in \chi = \left\{ x \in \Re^n, x_j^{min} \le x_j \le x_j^{max} \right\}$$

where

$$f_i^{M,k}(x) \doteq f_i\left(x^k\right) - \sum_{j=1}^{n} \left(\frac{p_{ij}^k}{U_j^k - x_j^k} + \frac{q_{ij}^k}{x_j^k - L_j^k} \right) + \sum_{j=1}^{n} \left(\frac{p_{ij}^k}{U_j^k - x_j} + \frac{q_{ij}^k}{x_j - L_j^k} \right)$$

$$p_{ij}^k = \begin{cases} \left(U_j^k - x_j^k\right)^2 \dfrac{\partial f_i\left(x^k\right)}{\partial x_j} & \text{if} \dfrac{\partial f_i\left(x^k\right)}{\partial x_j} > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$q_{ij}^k = \begin{cases} 0 & \text{if} \dfrac{\partial f_i\left(x^k\right)}{\partial x_j} \ge 0 \\ -\left(x_j^k - L_j^k\right)^2 \dfrac{\partial f_i\left(x^k\right)}{\partial x_j} & \text{otherwise} \end{cases}$$

Svanberg's MMA codes are used to solve the following

$$\left.\begin{aligned} \min_{\boldsymbol{x},\boldsymbol{y},z} \; &: \; f_0(\boldsymbol{x}) + a_0\, z + \sum_{i=1}^{m}(c_i y_i + \frac{1}{2} d_i y_i^2) \\ \text{subject to} \; &: \; f_i(\boldsymbol{x}) - a_i\, z - y_i \le 0, \qquad i = 1,\ldots,m \\ &: \; x_j^{min} \le x_j \le x_j^{max}, \qquad j = 1,\ldots,n \\ &: \; y_i \ge 0, \qquad i = 1,\ldots,m \\ &: \; z \ge 0 \end{aligned}\right\},$$

Our problem is

$$\left.\begin{aligned} \min_{\boldsymbol{x}} \; &: \; f_0(\boldsymbol{x}) \\ \text{subject to} \; &: \; f_i(\boldsymbol{x}) \le 0, \qquad i = 1,\ldots,m \\ &: \; x_j^{min} \le x_j \le x_j^{max}, \qquad j = 1,\ldots,n \end{aligned}\right\},$$

Svanberg's suggestion for our problem

$$a_0 = 1,\; a_i = 0,\; c_i = 1000,\; d = 0.$$

The MMA call is

```
function [xmma,ymma,zmma,lam,xsi,eta,mu,zet,s,low,upp] = ...
    mmasub(m,n,iter,xval,xmin,xmax,xold1,xold2, ...
    f0val,df0dx,df0dx2,fval,dfdx,dfdx2,low,upp,a0,a,c,d);
```

Hints:

- Check the definition of the MMA variables in the mmasub.m file.
- Be careful of the difference between row and colume vectors.
- Normalize constraints and objective function. i.e. use $V(x)/V_0-1\leq 0$ instead of $V(x)\leq V_0$.

Alternative ‘88 line Matlab code’ (2011)

- for experienced FE/Matlab people
- Usage: `top88(nelx,nely,volfrac,penal,rmin,ft)`

Struct Multidisc Optim (2011) 43:1–16
DOI 10.1007/s00158-010-0594-7

EDUCATIONAL ARTICLE

Efficient topology optimization in MATLAB using 88 lines of code

Erik Andreassen · Anders Clausen · Mattias Schevenels ·
Boyan S. Lazarov · Ole Sigmund

88 line vs. 99 line

- Advantages
  - Dramatically faster for large problems (speed quick-fix for 99-line code will be provided)
  - Access to a density filter in addition to the sensitivity filter
    - ft =1: sensitivity filter, spatial smoothing of sensitivities to avoid checkerboards and mesh dependency – mathematically inconsistent but works well in practice
    - ft =2: density filter, spatial smoothing of design variables + consistent update of sensitivities
- Disadvantages
  - More difficult to read
  - More difficult to modify

99-Line Code Speed Quick-Fix

- Main: assembly of sparse matrix

```
I = zeros(nelx*nely*64,1); J = zeros(nelx*nely*64,1); X = zeros(nelx*nely*64,1);
F = sparse(2*(nely+1)*(nelx+1),1); U = zeros(2*(nely+1)*(nelx+1),1);
ntriplets = 0;
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx +ely;
    edof = [2*n1-1 2*n1 2*n2-1 2*n2 2*n2+1 2*n2+2 2*n1+1 2*n1+2];
    xval = x(ely,elx)^penal;
    for krow = 1:8
      for kcol = 1:8
        ntriplets = ntriplets+1;
        I(ntriplets) = edof(krow);
        J(ntriplets) = edof(kcol);
        X(ntriplets) = xval*KE(krow,kcol);
      end
    end
  end
end
K = sparse(I,J,X,2*(nelx+1)*(nely+1),2*(nelx+1)*(nely+1));
```

```
K = sparse(2*(nelx+1)*(nely+1), 2*(nelx+1)*(nely+1));
F = sparse(2*(nely+1)*(nelx+1),1); U = zeros(2*(nely+1)*(nelx+1),1);
for elx = 1:nelx
  for ely = 1:nely
    n1 = (nely+1)*(elx-1)+ely;
    n2 = (nely+1)* elx +ely;
    edof = [2*n1-1; 2*n1; 2*n2-1; 2*n2; 2*n2+1; 2*n2+2; 2*n1+1; 2*n1+2];
    K(edof,edof) = K(edof,edof) + x(ely,elx)^penal*KE;
  end
end
```

New 99 line Matlab code (2020)

Structural and Multidisciplinary Optimization
https://doi.org/10.1007/s00158-020-02629-w

EDUCATIONAL PAPER

A new generation 99 line Matlab code for compliance topology optimization and its extension to 3D

Federico Ferrari[1] · Ole Sigmund[1]

Received: 18 February 2020 / Revised: 2 May 2020 / Accepted: 10 May 2020


```
top99neo(nelx,nely,volfrac,penal,rmin,ft,ftBC,eta,beta,move,maxit)
```

```
top3D125(nelx,nely,nelz,volfrac,penal,rmin,ft,ftBC,eta,beta,move,maxit)
```

Educational computer programs for structural topology optimization

Name and authors	Environment	Method	Assumption
`99-line` (Sigmund 2001)	MATLAB	SIMP	Linear
`SOFT-KILL` (Huang and Xie 2010)	MATLAB	ESO	Linear
`dLSM` (Challis 2010)	MATLAB	LSM	Linear
`PolyTop` (Talischi et al. 2012)	MATLAB	SIMP	Linear
`169-line 3D` (Liu and Tovar 2014)	MATLAB	SIMP	Linear
`115-line` (Tavakoli and Mohseni 2014)	MATLAB	SIMP	Linear
`PYTHON 3D` (Zuo and Xie 2015)	PYTHON and ABAQUS	ESO	Linear
`MMC188` (Zhang et al. 2016)	MATLAB	MMC	Linear
`SERA` (Loyola et al. 2018)	MATLAB	ESO	Linear
`FEniCS` (Laurain 2018)	FEniCS	LSM	Linear
`88-line` (Wei et al. 2018)	MATLAB	LSM	Linear
`213-line` (Chen et al. 2019)	MATLAB and ANSYS	SIMP	Nonlinear
`185-line` (Laurain 2018)	FEniCS	LSM	Linear
`AC# 3D` (Lagaros et al. 2019)	SAP2000	SIMP	Linear
`128-line` (Liang and Cheng 2020)	MATLAB	SIMP	Linear
`108-line` (Kim et al. 2020)	FREEFEM	LSM	Linear
`62-line` (Yaghmaei et al. 2020)	MATLAB	LSM	Linear