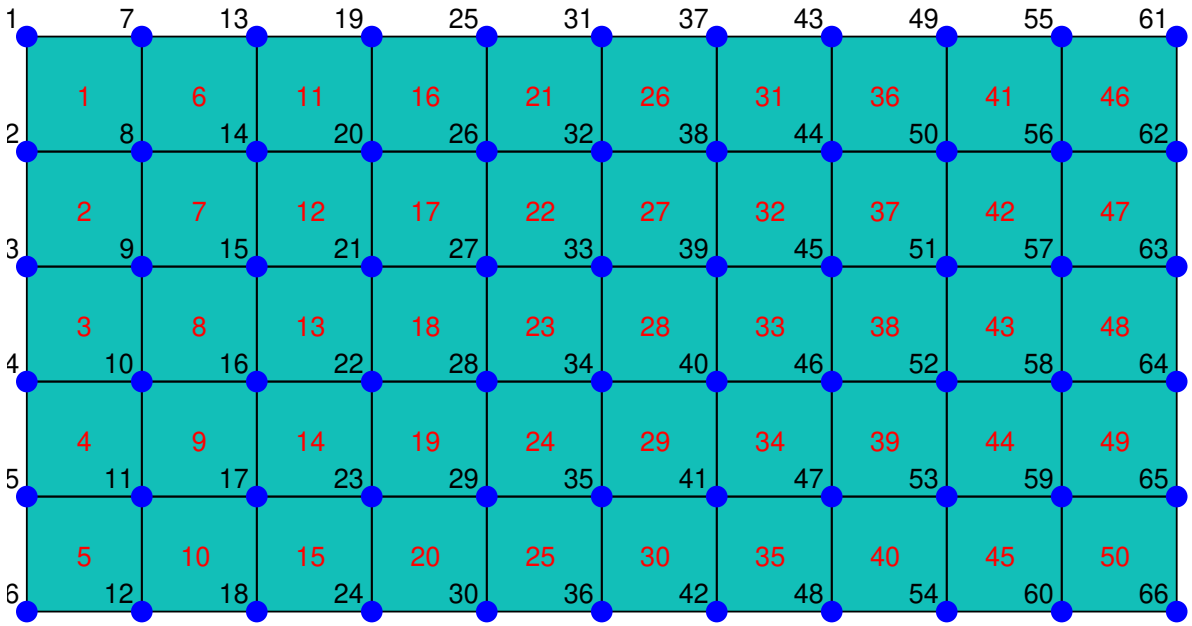
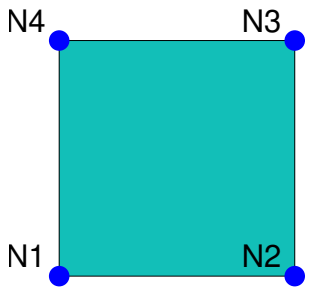
Topology optimization V: Continuum topology optimization exercises

Department of Mechanics
Faculty of Civil Engineering
Czech Technical University in Prague

1. Initialize problem
   - FEM problem: discretization, boundary conditions, material properties
   - Optimization problem: objective, constraints
   - Prepare filter
2. Starting point $\boldsymbol{\rho}$ (e.g., $\boldsymbol{\rho} = \mathbf{1}\frac{\overline{V}}{\sum_{i=1}^{n_\mathrm{e}} v_i}$)
3. While not converged (usually $\|\boldsymbol{\rho}\|_\infty < 0.01$) or exceeded iteration limit
   - Solve the state problem
   - Evaluate the sensitivities
   - Filtering
   - (Projections)
   - Update the design variables

```
nelemx = 10;              % number of elements in x direction (img. width)
nelemy = 5;               % number of elements in y direction (img. height)
filterRadius = 2;         % number of elements for the filter
filterType = 'density';% filter type. Either density/sensitivity
```






```
nodes.x = kron(0:nelemx,ones(1,nelemy+1)); % x coordinates of nodes
nodes.y = kron(ones(1,nelemx+1),0:nelemy); % y coordinates of nodes
elements.nodes4 = 1:(nelemy+1)*nelemx;
elements.nodes4(nelemy+1:nelemy+1:end) = [];
elements.nodes3 = elements.nodes4+nelemy+1;
elements.nodes2 = elements.nodes3+1;
elements.nodes1 = elements.nodes2-nelemy-1;
```

```
E0 = 1.0;                  % Young modulus of the solid phase
Emin = 1e-9;               % Young modulus of the void phase
nu = 0.3;                  % Poisson ratio
penalizationFactor = 3;    % for SIMP parametrization
volumeFraction = 0.5;      % upper bound for the material volume
```

- Modified SIMP interpolation

$$E_i(\rho_i) = E_{\min} + \rho_i^p (E_0 - E_{\min})$$

- Isotropic material, plane stress problem

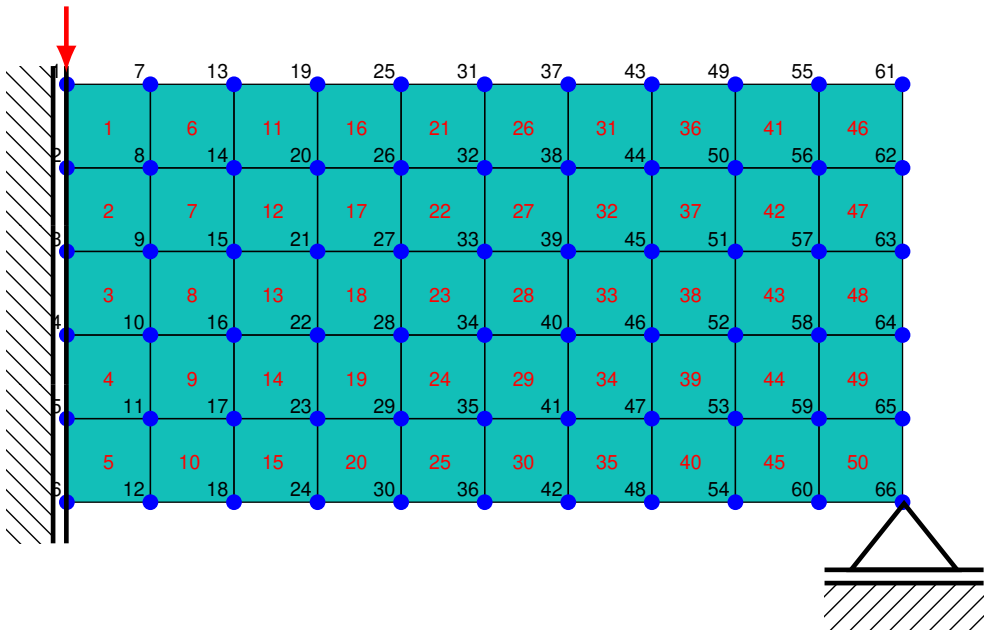
$$\mathbf{D}_i(\rho_i) = \frac{E_i(\rho_i)}{1-\nu^2} \begin{pmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{pmatrix}$$

- Element stiffness matrix (bilinear square quadrilaterals)

$$\mathbf{K}_i(\rho_i) = E_i(\rho_i)\mathbf{K}_0$$

- Kinematic and static boundary conditions
- In kinematic b.c., we assume only zero prescribed values

```
% node indices with restricted x-displacements
kinematic.x.nodes = 1:nelemy+1;
% node indices with restricted y-displacements
kinematic.y.nodes = (nelemy+1)*(nelemx+1);
loads.x.nodes = [];      % node indices with x-direction forces
loads.x.value = [];      % size of the x-direction forces
loads.y.nodes = 1;       % node indices with y-direction forces
loads.y.value = 1;       % size of the y-direction forces
```



- Kinematic and static boundary conditions
- In kinematic b.c., we assume only zero prescribed values

```
% node indices with restricted x-displacements
kinematic.x.nodes = 1:nelemy+1;
% node indices with restricted y-displacements
kinematic.y.nodes = (nelemy+1)*(nelemx+1);
loads.x.nodes = [];      % node indices with x-direction forces
loads.x.value = [];      % size of the x-direction forces
loads.y.nodes = 1;       % node indices with y-direction forces
loads.y.value = 1;       % size of the y-direction forces
```

- Mark degrees of freedom by `true`
- Order: $u_{1,x}, u_{1,y}, u_{2,x}, u_{2,y}, \ldots, u_{n_{\text{nodes}},y}$

```
dofs = true(nnodes*2,1);                % no kinematic boundary conditions
dofs(kinematic.x.nodes*2-1) = false; % mark prevented movement in x-dir
dofs(kinematic.y.nodes*2) = false;   % mark prevented movement in y-dir
```

- Prepare data for FEM

```
% initial guess: uniform mass distribution
densities = ones(nelemy,nelemx)*volumeFraction;
densitiesPhysical = densities;      % physical field after filtering
compliance = Inf;                   % currently unknown compliance value
elementStiffnessMatrix = ...        % unitary element stiffness matrix
elementsDOFs = ...                  % matrix of element DOFs
stiffnessMatrixRows = ...           % how to assemble stiffness matrix
stiffnessMatrixColumns = ...        % how to assemble stiffness matrix
% constant force vector
forceVector = sparse([loads.x.nodes*2-1; loads.y.nodes*2;], 1, ...
                     [loads.x.value; loads.y.value], nnodes*2, 1);
displacements = zeros(size(dofs)); % displacement field
```

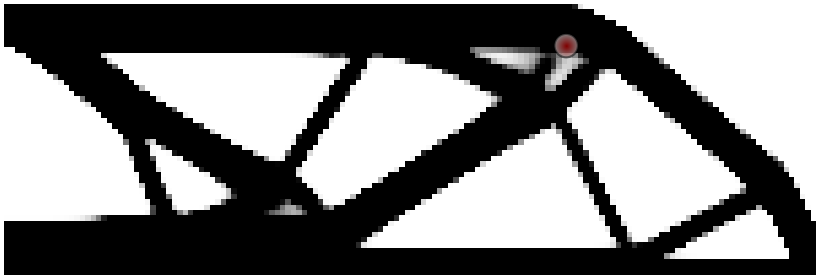
- Prepare filtering matrices

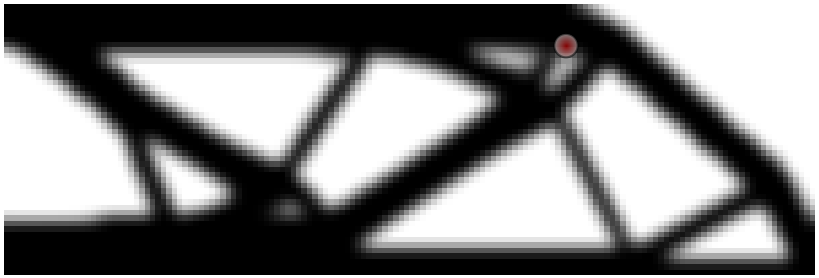
```
filterMatrix = sparse(filterMatrixRow, filterMatrixColumn, ...
                                       filterMatrixValue);
filterMatrixScaling = sum(filterMatrix, 2);
```

- Regularization of the density field by a filter with radius r

$$\tilde{\rho}_i = \frac{\sum_{j=1}^{n_\mathrm{e}} w(\mathbf{x}_j) v_j \rho_j}{\sum_{j=1}^{n_\mathrm{e}} w(\mathbf{x}_j) v_j}, \text{ where } w_j(\mathbf{x}_j) = \max\{r - \|\mathbf{x}_j - \mathbf{x}_i\|_2, 0\}$$

$$\frac{\partial f}{\partial \rho_i} = \sum_{j=1}^{n_\mathrm{e}} \frac{\partial f}{\partial \tilde{\rho}_j} \frac{\partial \tilde{\rho}_j}{\partial \rho_j}, \text{ where } \frac{\partial \tilde{\rho}_j}{\partial \rho_j} = \frac{w(\mathbf{x}_j) v_j}{\sum_{k=1}^{n_\mathrm{e}} w(\mathbf{x}_k) v_k}$$





While not converged (usually $\|\boldsymbol{\rho}\|_\infty < 0.01$) or exceeded iteration limit

- Solve the state problem
- Evaluate the sensitivities
- Filtering
- Update the design variables (OC)

```
change = Inf;       % Infinity norm of the change of design variables
iter = 0;           % Iteration number
limitChange = 1e-2;% Change for convergance
% Repeat until we reach the desired accuracy
while (change>limitChange) && (iter<500)
   ...
   change = max(abs(densitiesNew(:) - densities(:)));
   densities = densitiesNew;
end
```

- Solution to the state problem

```
% Stiffness matrix values for the assembly
stiffnessMatrixValues = reshape(elementStiffnessMatrix(:) * ...
            (Emin + densitiesPhysical(:)' .^ penalizationFactor * ...
            (E0 - Emin)), 64 * nelemx * nelemy, 1);
% Assembly
stiffnessMatrix = sparse(stiffnessMatrixRows, ...
                         stiffnessMatrixColumns, stiffnessMatrixValues);
% Ensure that the stiffness matrix is numerically symmetric
stiffnessMatrix = (stiffnessMatrix + stiffnessMatrix') / 2;
% Solve the FE problem
displacements(dofs) = stiffnessMatrix(dofs, dofs) \ forceVector(dofs);
% Compute compliance
compliance = forceVector(dofs)'*displacements(dofs);
```

- Compute sensitivities

```
% u'*K0*u for all the elements
energyElements = reshape(sum((displacements(elementsDOFs) * ...
     elementStiffnessMatrix) .* displacements(elementsDOFs), 2), ...
     nelemy, nelemx);
% Sensitivity of the compliance function
sensitivity = -penalizationFactor * (E0 - Emin) * ...
     densitiesPhysical .^ (penalizationFactor - 1) .* energyElements;
% Sensitivity of the volume constraint
sensitivityVolume = ones(nelemy, nelemx);
```

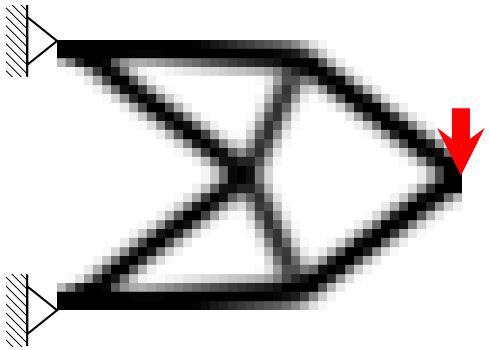
- Filtering

```
% Sensitivity filter
if strcmp(filterType,'sensitivity')
   sensitivity(:) = filterMatrix * (densities(:) .* sensitivity(:)) ./...
          filterMatrixScaling ./ max(1e-3, densities(:));
% Density filter
elseif strcmp(filterType,'density')
   sensitivity(:) = filterMatrix*(sensitivity(:)./filterMatrixScaling);
   sensitivityVolume(:) = filterMatrix * (sensitivityVolume(:) ./...
          filterMatrixScaling);
else
   error('Unknown filter type.');
end
```

- OC method

```
muLower = 0; muUpper = 1e9; dampingCoefficient = 0.5; moveLimit = 0.2;
while (muUpper-muLower)/(muLower+muUpper) > 1e-3
   muMiddle = 0.5*(muLower+muUpper);
   densitiesNew = ... % compute new densities
   if strcmp(filterType,'density')
      densitiesPhysical(:) = (filterMatrix * ...
         densitiesNew(:)) ./ filterMatrixScaling;
   end
   if sum(densitiesPhysical(:)) > (volumeFraction * nelemx * nelemy),...
      muLower = muMiddle; else, muUpper = muMiddle; end
end
```

1. Change boundary conditions to the cantilever beam optimization



2. Investigate the effect of mesh refinement
3. Investigate the effect of relative volume fraction
4. Investigate the effect of different filter types and radii
5. Investigate the effect of the penalization coefficient value
6. Change the objective function to optimization of compliant mechanisms, optimize a force inverter (recall previous lecture for boundary conditions)
   - $k_{\text{in}} = 1$, $k_{\text{out}} = 0.05$, $b_i = \max\{10^{-10}, -\frac{\mathrm{d}f}{\mu \mathrm{d}V}\}$
   - Change `dampingCoefficient` to 0.3
7. (Implement Heaviside projection)
8. (Replace OC with MMA)
9. (Dynamic compliance optimization)

- E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund, Efficient topology optimization in MATLAB using 88 lines of code, *Structural and Multidisciplinary Optimization*, 43(1): 1–16, 2010, doi: 10.1007/s00158-010-0594-7
- M. P. Bendsøe and O. Sigmund, *Topology Optimization*. Springer Berlin Heidelberg, 2004, doi: 10.1007/978-3-662-05086-6