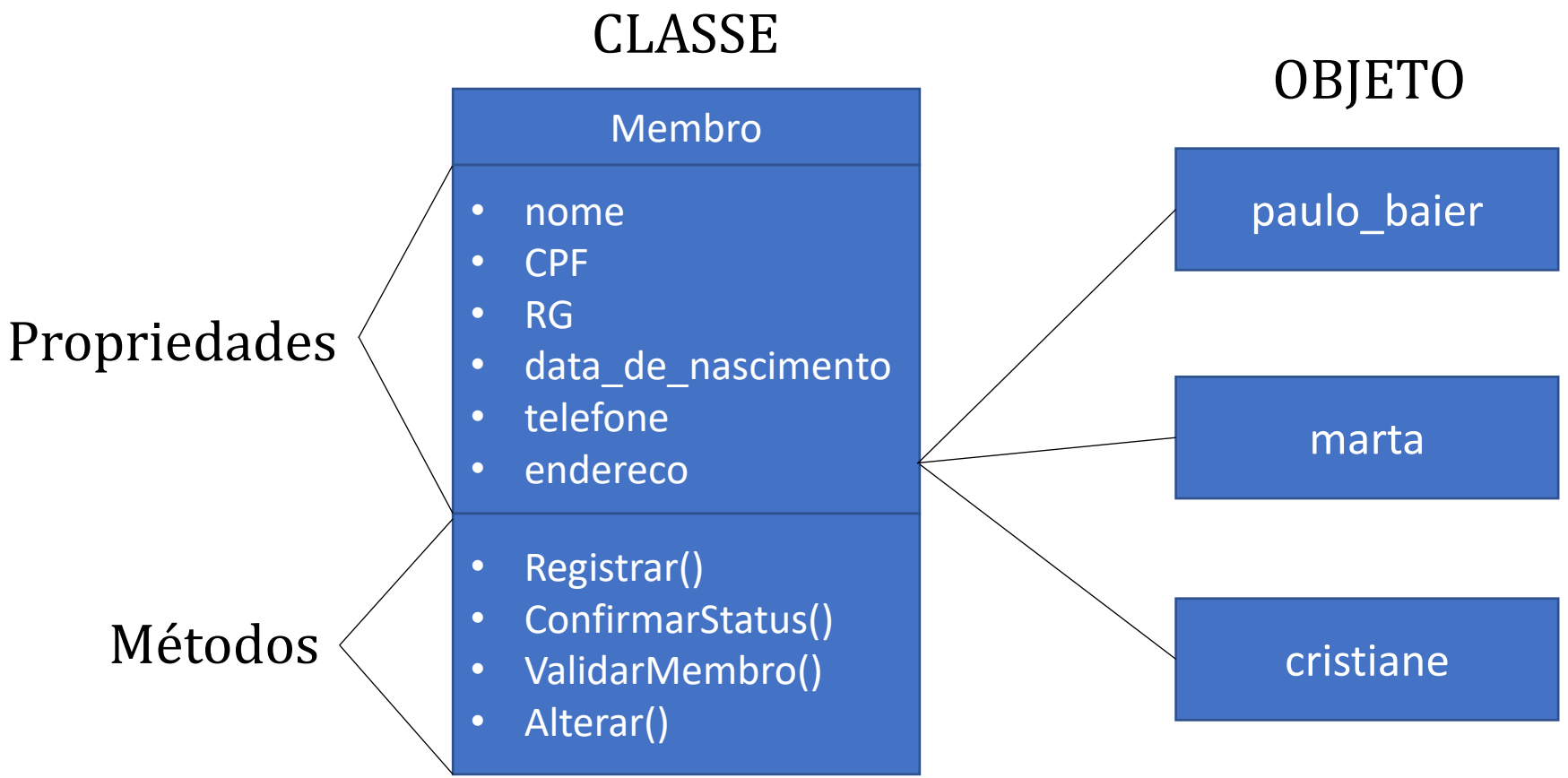
PMR5250 – Método de Otimização Topológica Aplicada ao Projeto Mecânico

Aula 2b – MEF - Implementação

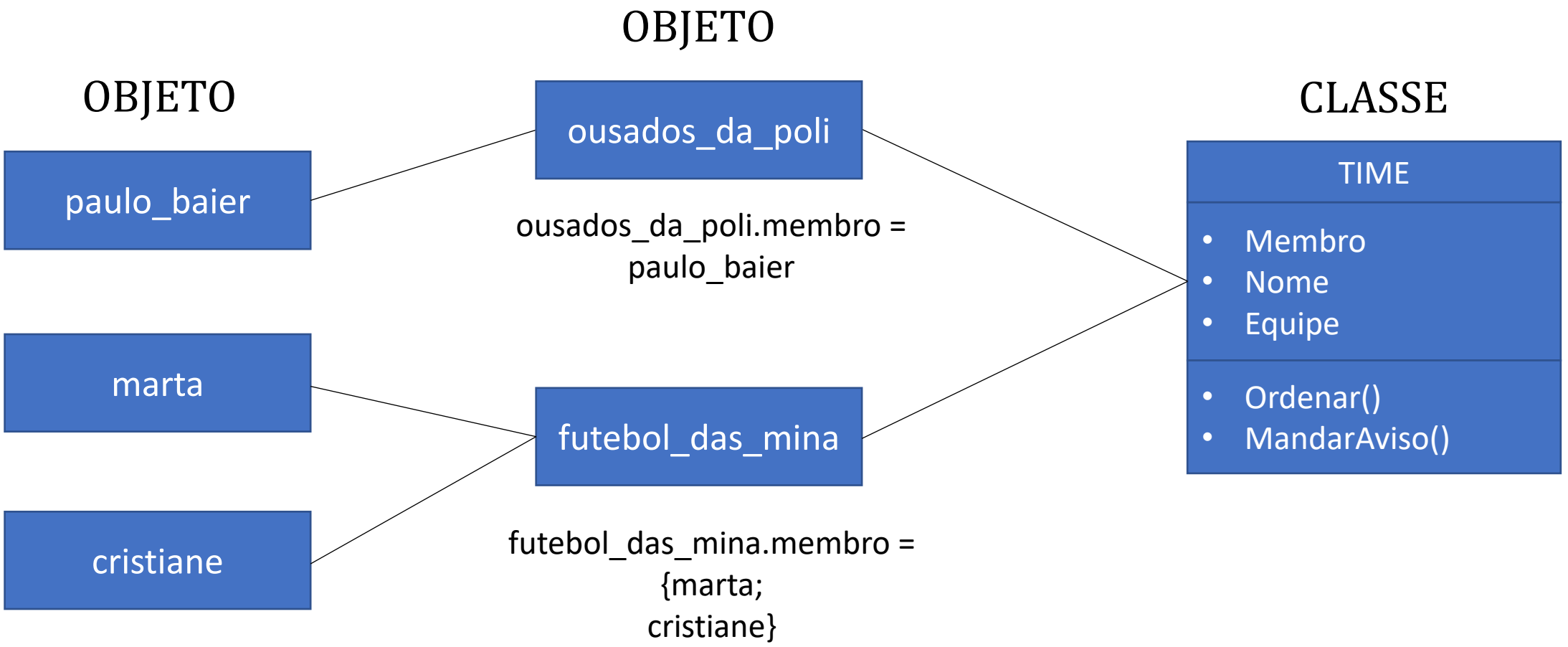
Prof. Dr. Emílio C. N. Silva
Prof. Dr. Renato Picelli

Departamento de Engenharia Naval e Oceânica
Escola Politécnica da Universidade de São Paulo

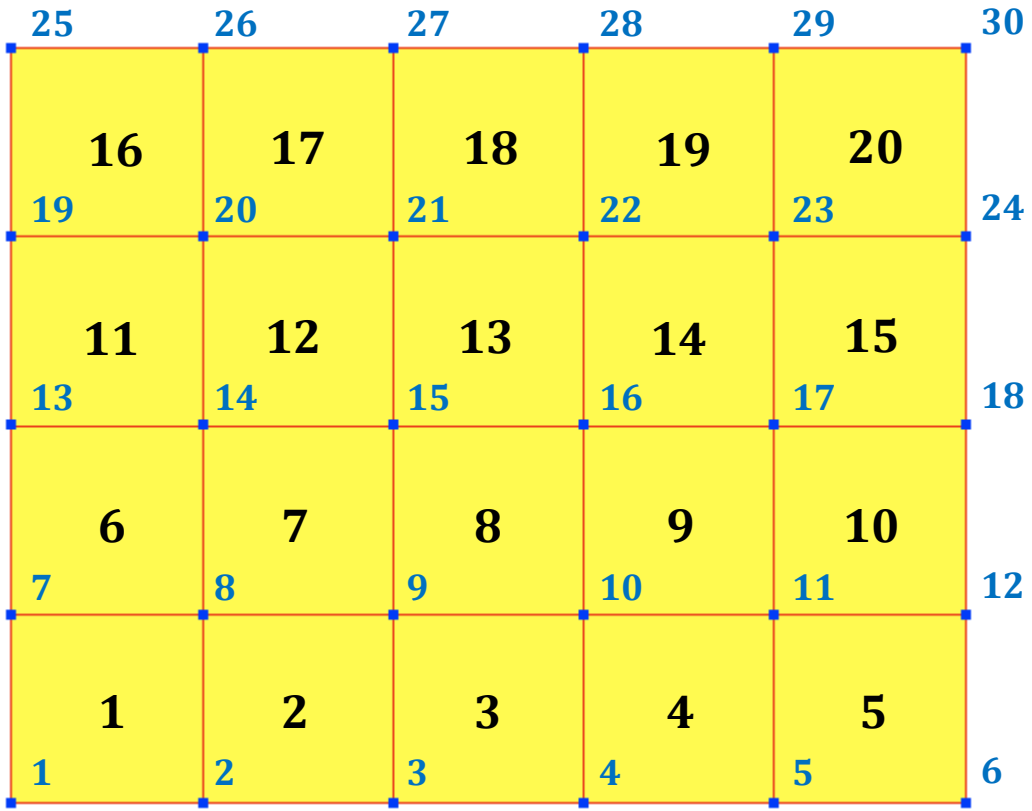
Programação orientada a objetos (POO)



Programação orientada a objetos (POO)



Malha (numeração)



Classe “Mesh”

```
% Mesh size info
Lx, Ly, Lz, nelx, nely, nelz

% Nodal coordinates (lines = nodes, columns = coordinates)
% [  coordinate_X1, coordinate_Y1, coordinate_Z1,
%    coordinate_X2, coordinate_Y2, coordinate_Z2,
%    ...
%    coordinate_Xn, coordinate_Yn, coordinate_Zn  ];
coordinates

% Element incidence - connectivity
 (lines = elements, columns = element type and connected to node)
% [  element_1_type, node_1, node_2, node_3, node_4,
%    element_2_type, node_1, node_2, node_3, node_4,
%    ...
%    element_e_type, node_1, node_2, node_3, node_4  ];
incidence
```

PNV USP

Classe “Mesh”

```
% Dirichlet boundary conditions
% [  node, DOF type, value  ];
dirichlet_boundary

% Neumann boundary conditions
% [  node, DOF type, value  ];
neumann_boundary
```

PNV USP

Classe “Mesh”

```
% Centroids coordinates (lines = elements, columns = coordinates)
% [  centroid_X1, centroid_Y1
%    centroid_X2, centroid_Y2
%    ...
%    centroid_Xn, centroid_Yn  ]; NOT READY FOR 3D YET!
centroids

% Nodal incidence - connectivity (lines = nodes, columns = element connected)
% [  node_1_number, element_1, element_2, element_3, element_4,
%    node_2_number, element_1, element_2, element_3, element_4,
%    ...
%    node_n_number, element_1, element_2, element_3, element_4  ];
nodal_connectivity
```

PNV USP

Código base

PNV USP

Classe “Elasticity”

```
%% Properties
properties

    % Finite element mesh
    mesh

    % Material properties
    E           % Young's modulus
    nu          % Poisson's ratio
    rho         % Solid material density
    th = 1.0; % thickness
```

PNV USP

Classe “Elasticity”

```
% ID matrix (relation node-DOF)
% (lines = DOF's, columns = nodes)
ID
number_of_equations

% Equation matrices
F % forces
K % stiffness
M % mass
U % displacements (state variable)
```

PNV USP

Matrix de Identificação (ID)

```
% Creating ID matrix (relation node-DOF)
% Initializing ID matrix: dofs = (u_x, u_y)
fea.ID = ones(2,size(fea.mesh.coordinates,1));
fea.ID(1,:) = 1+2*[0:(size(fea.mesh.coordinates,1)-1)];
fea.ID(2,:) = 2+2*[0:(size(fea.mesh.coordinates,1)-1)];

% Number of equations
fea.number_of_equations = fea.ID(2,end);
```

PNV USP

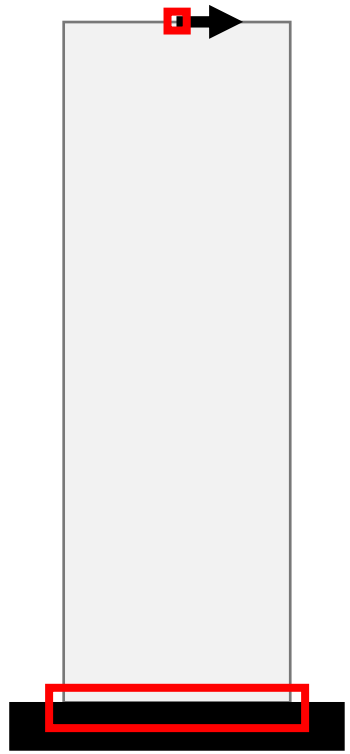
Parâmetros de entrada

```
% Definir dimensões da malha
Lx = 20;
Ly = 60;
% Número de elementos na malha (em x e y)
nelx = 20;
nely = 60;

% Propriedades do material
E = 1.0;    % Módulo de elasticidade
nu = 0.3;   % Coeficiente de Poisson
rho = 1.0; % Material density
```

PNV USP

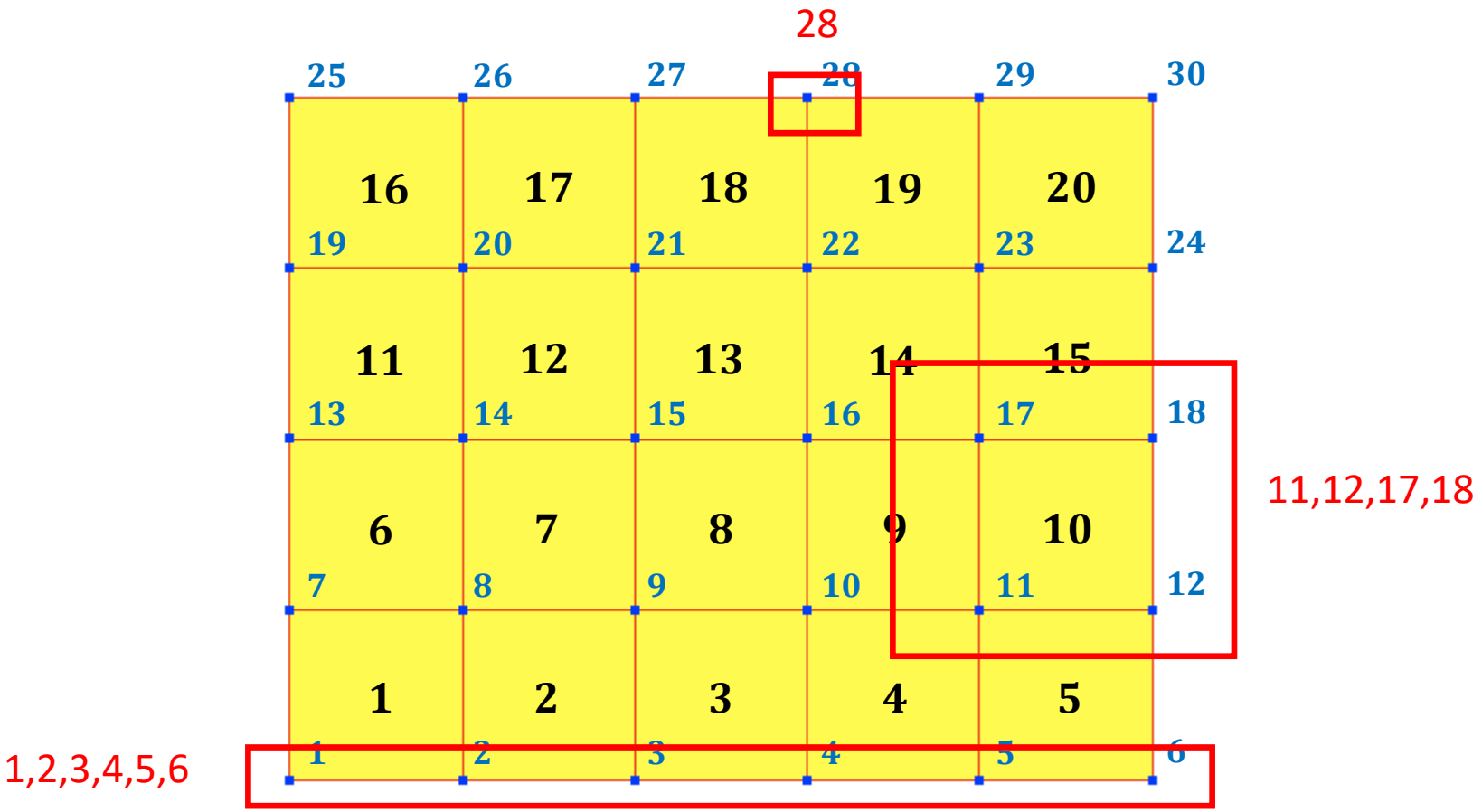
Condições de contorno

```
% Adicionar condições de contorno de Dirichlet
dirichlet.point_1 = [-0.001, -0.001];
dirichlet.point_2 = [mesh.Lx*1.001, 0.001];
dirichlet.dofs = [1, 2];
dirichlet.value = 0.0;
mesh = AddDirichletBC(mesh,dirichlet);

% Adicionar condições de contorno de Neumann
neumann.point_1 = [mesh.Lx*0.4999, mesh.Ly*0.9999];
neumann.point_2 = [mesh.Lx*0.5001, mesh.Ly*1.0001];
neumann.dofs = [1];
neumann.value = 1.0;
mesh = AddNeumannBC(mesh,neumann);
```



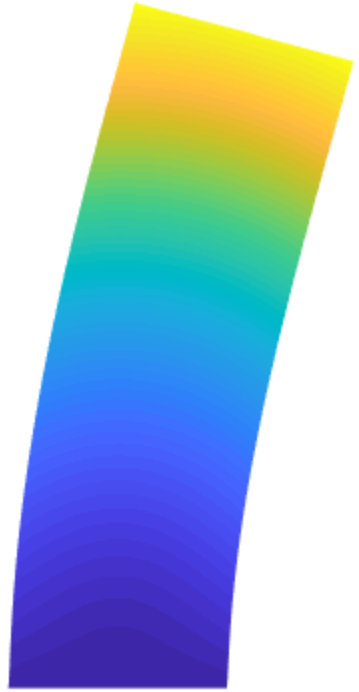
PNV USP

Seleção de nó



Condições de contorno, *solver* e *plot*

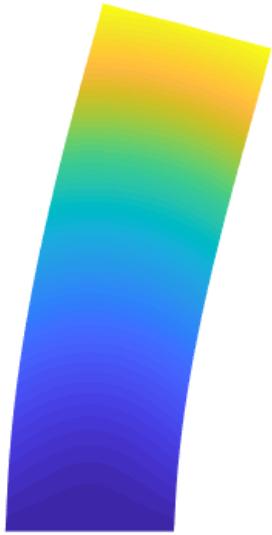
```
% Assembly do vetor de forças
fea = AssemblePointLoads(fea);

% Assembly da matriz de rigidez
densities = ones(length(fea.mesh.incidence),1);
fea = AssembleStructuralK(fea,densities);

% Resolver equação de equilíbrio
fea = SolveStaticStructuralFEA(fea);

% Plotar deslocamentos da estrutura
PlotStructuralDisplacements(fea,1e-1)
```



PNV USP

Solução da equação de equilíbrio

Impor condições de contorno de Neumann ($f(x,y) = f_0$)

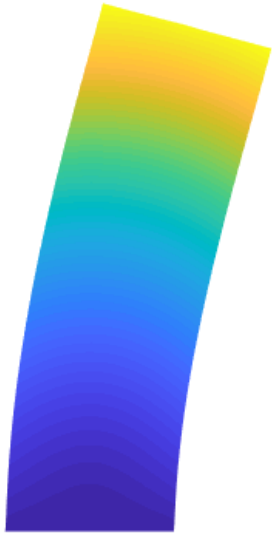


```
% Load vector assembly
self.F = sparse(zeros(2,1));
self.F(self.number_of_equations) = 0;
% Applied forces
for i = 1:size(self.mesh.neumann_boundary,1)
if (self.mesh.neumann_boundary(i,1) ~= 0)
    self.F(self.ID(self.mesh.neumann_boundary(i,2),self.mesh.neumann_boundary(i,1)),1) = ...
              ... self.mesh.neumann_boundary(i,3);
end
end
```

PNV USP

Solução da equação de equilíbrio

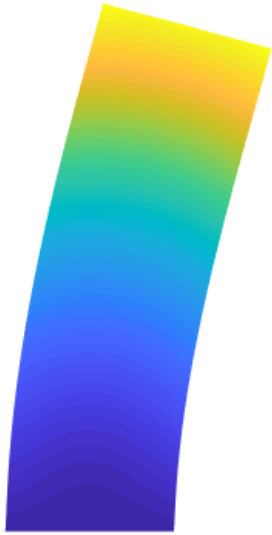
Montagem da matriz de rigidez global:



```
% Assembly
self.K = sparse(I,J,Kg);
```

Solução da equação de equilíbrio

Identificação das condições de contorno de Dirichlet ($u = 0$)



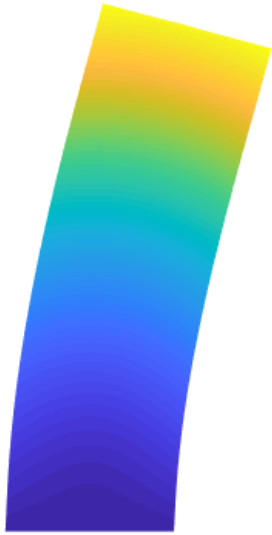
```
% Identifying active DOF's
dofn = [1:self.number_of_equations]';
% Blocking DOF's with dirichlet boundary conditions = 0
for i = 1:size(self.mesh.dirichlet_boundary,1)
    if (self.mesh.dirichlet_boundary(i,1) ~= 0) && (self.mesh.dirichlet_boundary(i,3) == 0)
       dofn(self.ID(self.mesh.dirichlet_boundary(i,2),self.mesh.dirichlet_boundary(i,1))) = 0;
    end
end
% Active DOF's
dofa = nonzeros(dofn);
```

PNV USP

Solução da equação de equilíbrio

Solução do sistema linear:

```
% Matlab linear solver
self.U(dofa) = self.K(dofa,dofa)\self.F(dofa);
```



Plotando deslocamentos estruturais

```
function PlotStructuralDisplacements(self,scale)

    disp([' '])
    disp(['          Plotting structural displacements.'])

    hold on

    % Initializing deformed coordinates and displacements
    Dcoord = self.mesh.coordinates;
    Ux = zeros(size(self.mesh.coordinates,1),1);
    Uy = zeros(size(self.mesh.coordinates,1),1);
```

PNV USP

Plotando deslocamentos estruturais

```
% Displacement vector magnitude
Gu = sqrt( Ux.^2 + Uy.^2 );

for i = 1:size(self.mesh.incidence,1)

    inci = self.mesh.incidence;

    % Plotting displacements
    x = [ Dcoord(inci(i,2),1) Dcoord(inci(i,3),1) ...
                                ... Dcoord(inci(i,4),1) Dcoord(inci(i,5),1) ];
    y = [ Dcoord(inci(i,2),2) Dcoord(inci(i,3),2) ...
                                ... Dcoord(inci(i,4),2) Dcoord(inci(i,5),2) ];
    c = [ Gu(inci(i,2)) Gu(inci(i,3)) Gu(inci(i,4)) Gu(inci(i,5)) ];
    fill(x,y,c,'LineStyle','none')

end
```

PNV USP

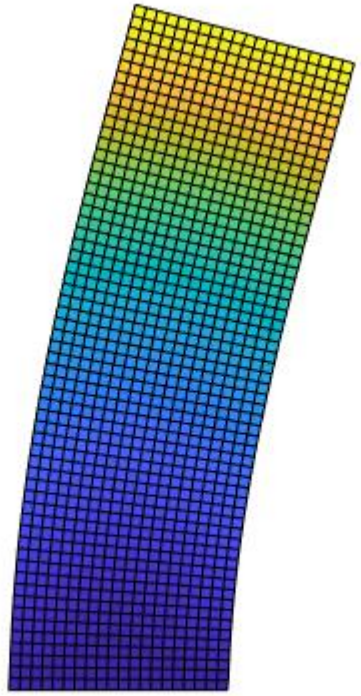
Plotando deslocamentos estruturais

Remover ...

```
fill(x,y,c,'LineStyle','none')
```

... e usar:

```
fill(x,y,c)
```



Tensão de von Mises

- O conceito de máxima tensão permitida é introduzido.

$$\sigma_e \leq \sigma_{\text{allowed}}$$

σ_e → Tensão equivalente

- O critério de tensão de von Mises (Richard von Mises, 1883-1953) é baseado na máxima energia de distorção.

$$\sigma_e = \sqrt{\sigma_x^2 + \sigma_y^2 - \sigma_x \sigma_y + 3\tau_{xy}^2}$$ → tensão de von Mises

PNV USP

Tensão de von Mises

- Equação

$$\sigma^{vm} = \sqrt{\sigma_x^2 + \sigma_y^2 - \sigma_x \sigma_y + 3\tau_{xy}^2}$$

- MEF

$$\boldsymbol{\sigma} = \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{Bmatrix} = \mathbf{DBu}$$

PNV USP

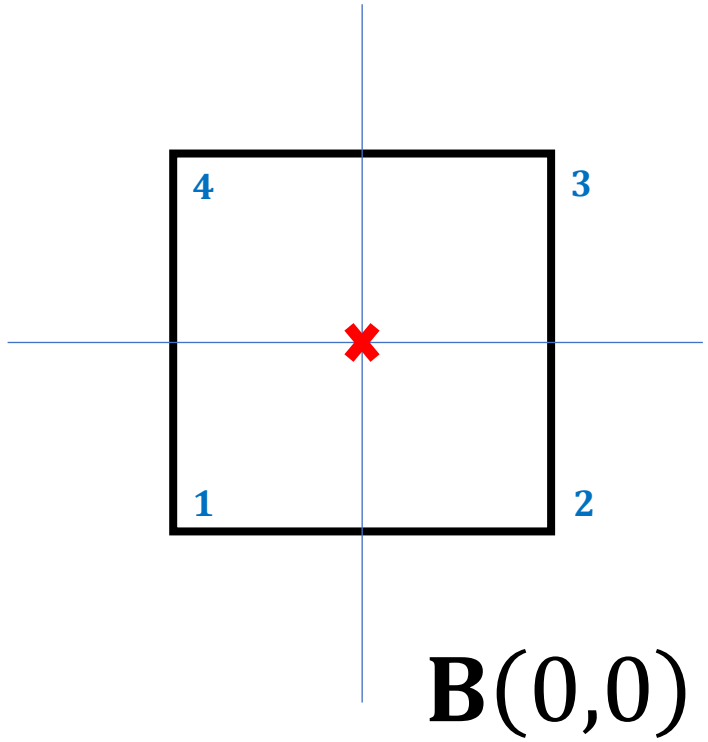
Tensão de von Mises

- Equação

$$\sigma^{vm} = \sqrt{(\mathbf{DBu})^T \mathbf{V} (\mathbf{DBu})}$$

- Matriz de Voigt

$$\mathbf{V} = \begin{bmatrix} 1 & -0.5 & 0 \\ -0.5 & 1 & 0 \\ 0 & 0 & 3 \end{bmatrix}$$



Tensão de von Mises

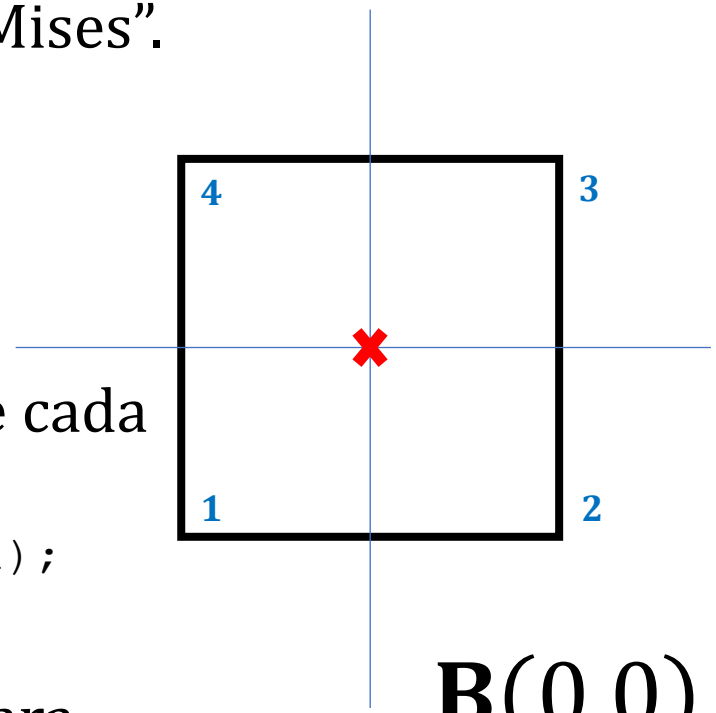
- Guia para implementar o código:
  - Criar uma “propriedade” em “Elasticity” chamada “vonMises”.
  - Criar uma nova função chamada “ComputeStresses”.

```
self = ComputeStresses(self,stress_case)
```

  - Alocar o vetor para armazenar a tensão de von Mises de cada elemento.

```
self.vonMises = zeros(length(self.mesh.incidence),1);
```

  - Loop nos elementos.
  - Copiar o conteúdo interno de “ComputeStructuralKe” para obter **DB** no centróides.



PNV USP

Tensão de von Mises

- Selecione os deslocamentos estruturais.

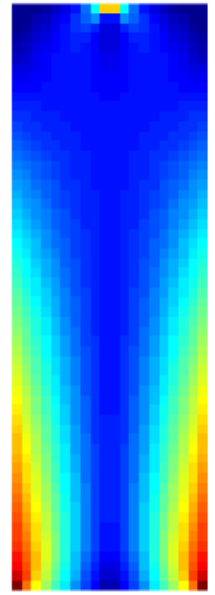
```
% Element nodes
no1 = self.mesh.incidence(i,2);
no2 = self.mesh.incidence(i,3);
no3 = self.mesh.incidence(i,4);
no4 = self.mesh.incidence(i,5);

% Vector with element DOF's
loc = [self.ID(1,no1), self.ID(2,no1), self.ID(1,no2), self.ID(2,no2),...
       self.ID(1,no3), self.ID(2,no3), self.ID(1,no4), self.ID(2,no4)];

% Element's displacement vector
Un = full(self.U(loc));
```

- Calcule as tensões de von Mises.

```
self.vonMises(i) = sqrt(DBU'*V*DBU);
```

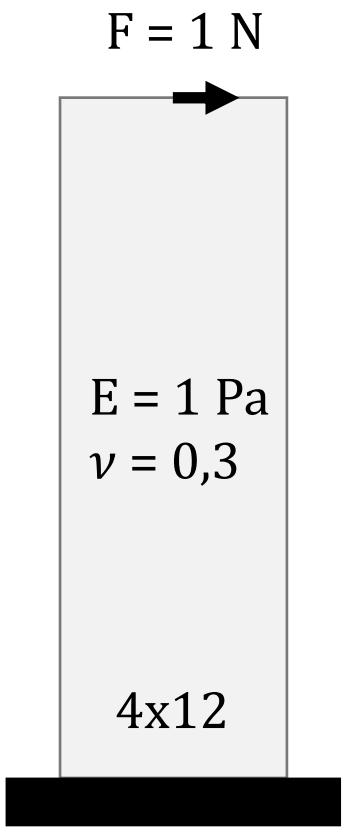
PNV USP

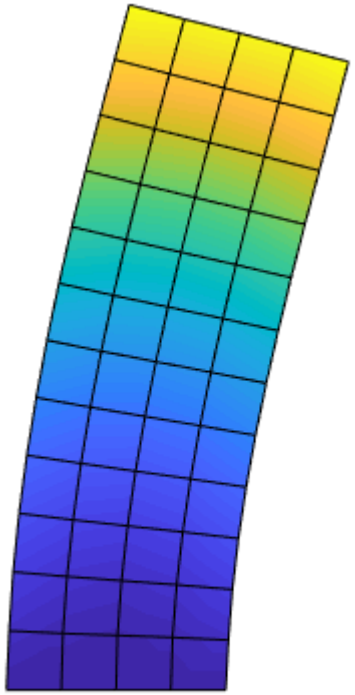
Função PlotScalarPerElement

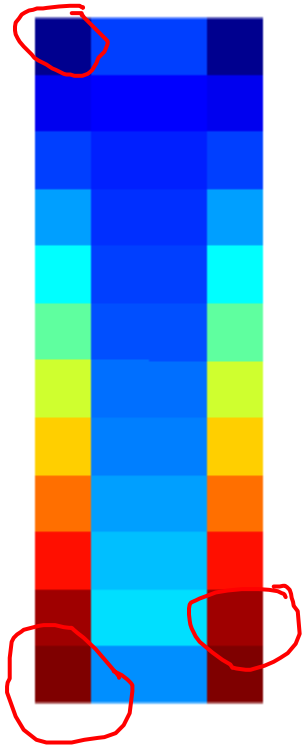


```
function PlotScalarPerElement(self,values,exp)

    % Defining matplot
    matplot = real(values.^(exp));
    matrixplot2 = self.mesh.plot_matrix;
    % Verifying if there are zeros in matrizplot
    if (length(find(matrixplot2 == 0)) >= 1)
        % Defining void for inexistent elements in the plot
        matplot(size(self.mesh.incidence,1)+1) = 0;
        matrixplot2(find(matrixplot2 == 0)) = size(self.mesh.incidence,1)+1;
    end

    % Plotting
    colormap(jet); imagesc(matplot(matrixplot2));
    axis equal; axis off;

end
```

Tensões de von Mises





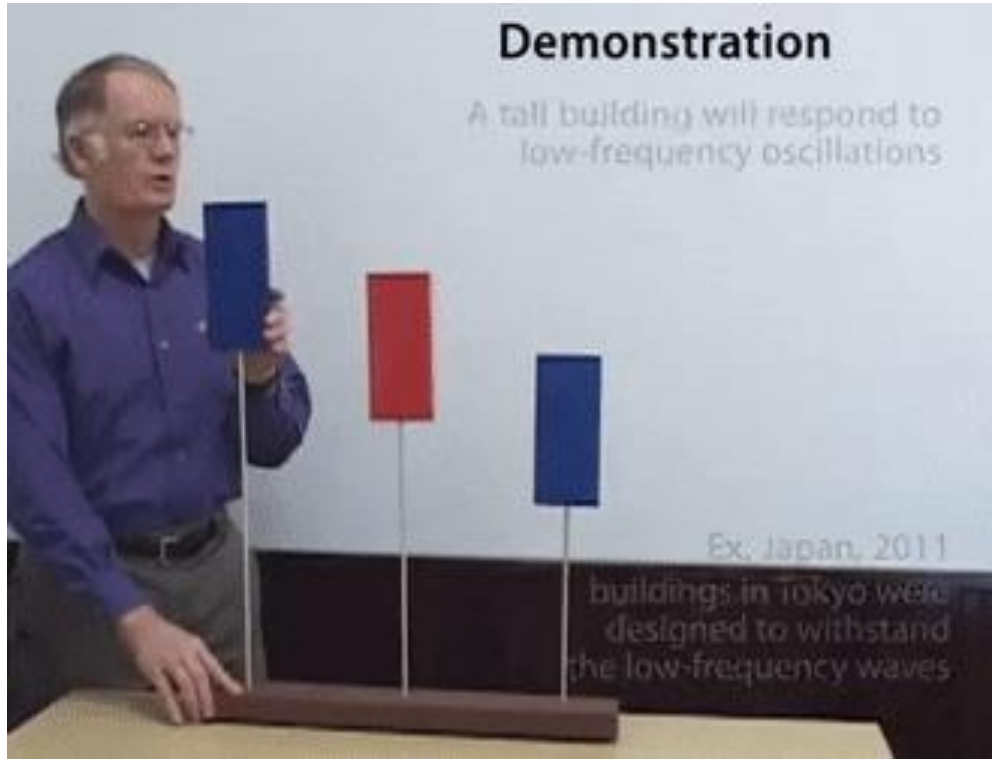


fea.vonMises =

0.0455	0.1479	0.1479	0.0455
0.1052	0.1130	0.1130	0.1052
0.1489	0.1253	0.1253	0.1489
0.1989	0.1342	0.1342	0.1989
0.2518	0.1437	0.1437	0.2518
0.3053	0.1548	0.1548	0.3053
0.3592	0.1672	0.1672	0.3592
0.4132	0.1805	0.1805	0.4132
0.4675	0.1948	0.1948	0.4675
0.5214	0.2120	0.2120	0.5214
0.5769	0.2306	0.2306	0.5769
0.6011	0.1896	0.1896	0.6011

PNV USP

Ressonância

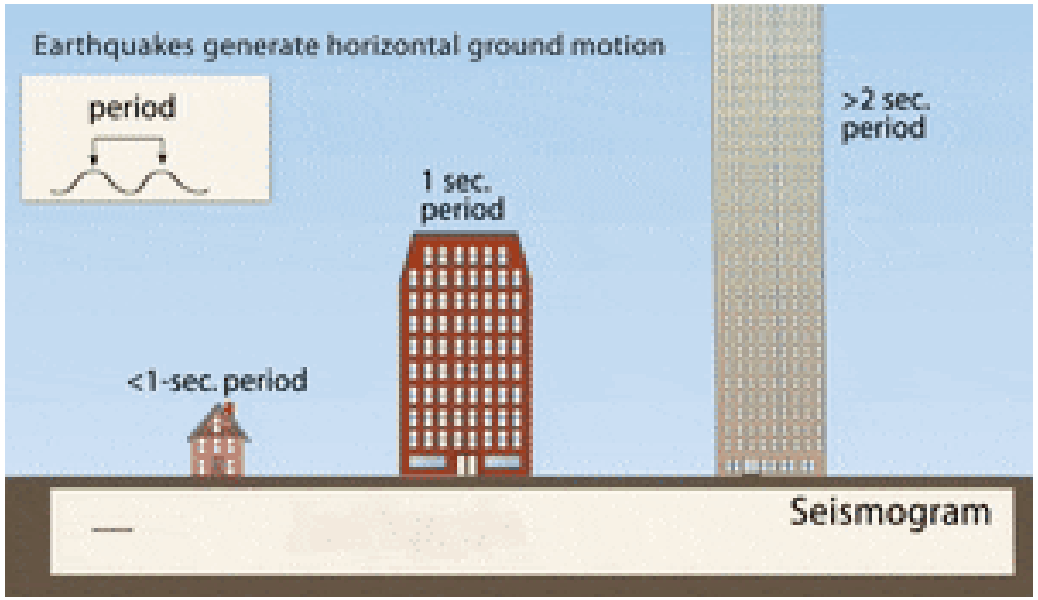


$$\omega_n = \frac{\omega_0}{2\pi} = \frac{1}{2\pi}\sqrt{\frac{k}{m}}$$

O conhecimento das frequências naturais de um sistema é de grande importância em problemas de engenharia!

PNV USP

Ressonância



$$\omega_n = \frac{\omega_0}{2\pi} = \frac{1}{2\pi}\sqrt{\frac{k}{m}}$$

PNV USP

Frequências naturais (vibração livre)

Equação:

$$(\mathbf{K} - \omega^2 \mathbf{M})\mathbf{u} = 0$$

Matriz de massa:

$$\mathbf{M} = \int_{\Omega} \rho \mathbf{H}^T \mathbf{H} d\Omega$$

$$\mathbf{H} = \{N_1, 0, N_2, 0, N_3, 0, N_4, 0; \\ 0, N_1, 0, N_2, 0, N_3, 0, N_4\}$$

```
% Shape functions
N1 = (1/4)*(1-Q)*(1-N);
N2 = (1/4)*(1+Q)*(1-N);
N3 = (1/4)*(1+Q)*(1+N);
N4 = (1/4)*(1-Q)*(1+N);

% Matrix [H] with shape functions
H = [N1 0 N2 0 N3 0 N4 0;
     0 N1 0 N2 0 N3 0 N4];
```

PNV USP

Frequências naturais (vibração livre)

Equação:

$$(\mathbf{K} - \omega^2 \mathbf{M})\mathbf{u} = 0$$

Para se impor condições de contorno (note que há apenas condição de Dirichlet nesse caso), utilize a mesma rotina que encontra os graus de liberdade ativos (`dofa`) na função `SolveStaticStructuralFEA`.

Comando no Matlab:

```
% Free vibration solver (sparse matrix -> eigs)
[AVet,AVal] = eigs(self.K(dofa,dofa),self.M(dofa,dofa),number_of_modes,'SM');
```

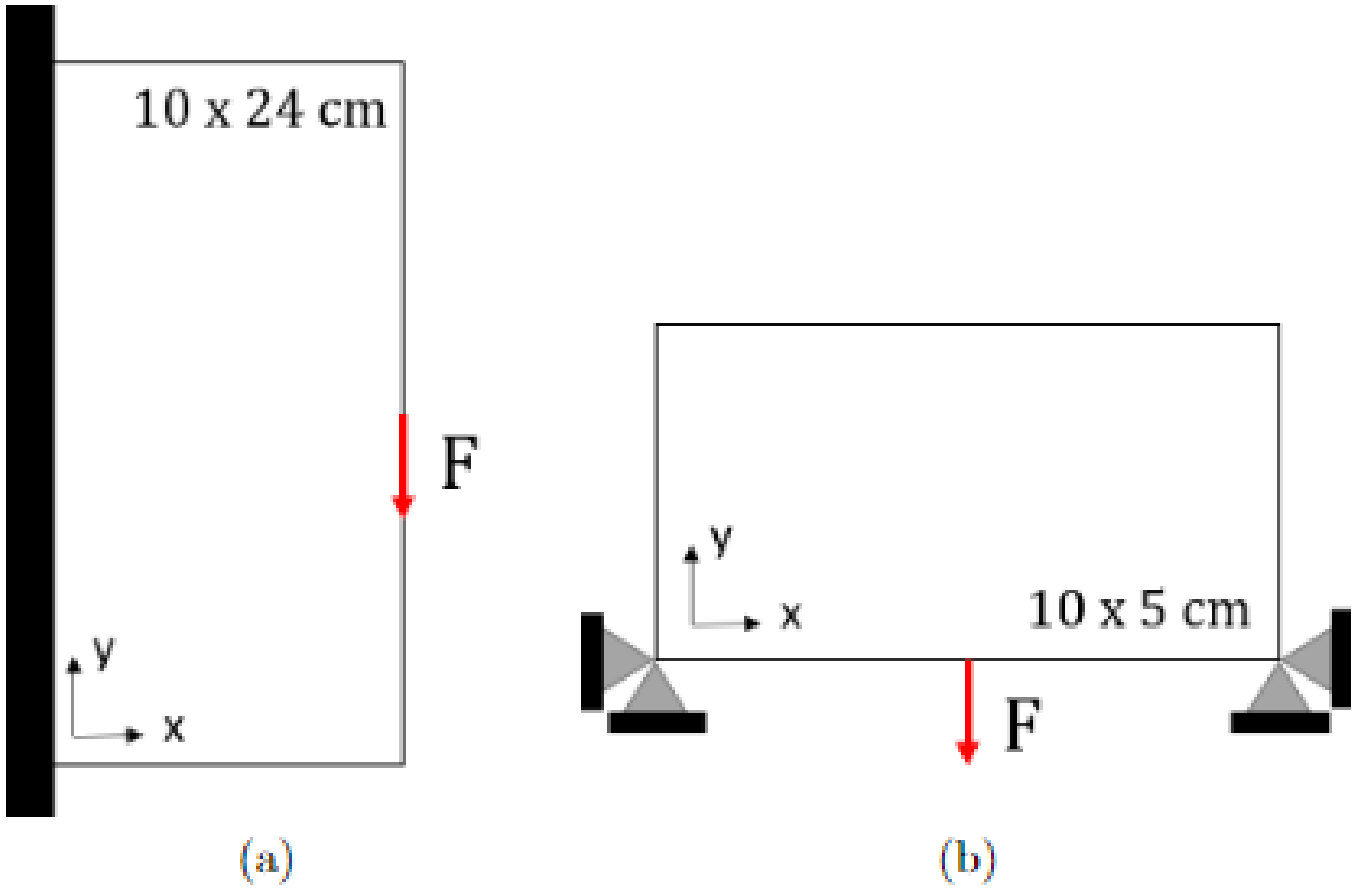
Autovetor (modos de vibrar)

Autovalores ω^2, onde a unidade de ω é em rad/s.
Dividir por 2π para se obter a resposta em Hz.

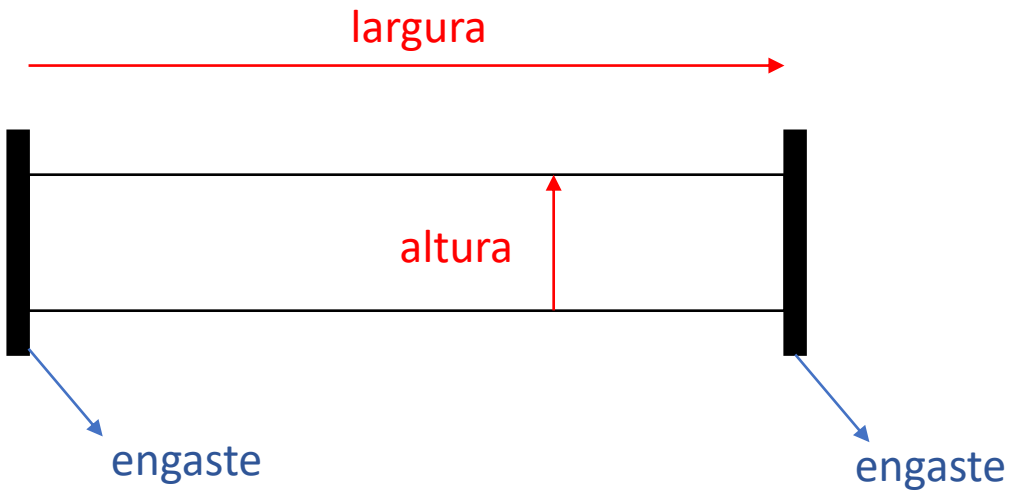
PNV USP

Caso estático

Trabalho 1A. Criar MAIN_01 e MAIN_02.



Frequências naturais (vibração livre)

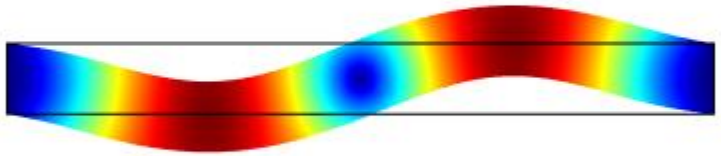


Trabalho 1B. Criar MAIN_03.
Escrever relatório e fornecer o código!

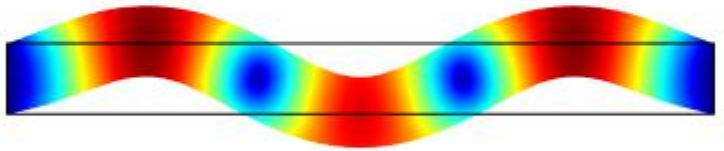
Largura = 10 m
Altura = 1 m
Alumínio ($E = 70 \cdot 10^9$ Pa e $\nu = 0.3$)
Estado plano de tensões
Espessura unitária

```
PlotEigenmodes(self,scale,number_of_modes)
```

ω = 49.3995 Hz



ω = 126.8920 Hz



ω = 229.9360 Hz