IMPLEMENTAÇÃO DE APLICAÇÕES MECÂNICAS NO SOFTWARE "FEM"

- CONDIÇÕES DE CONTORNO
- TENSÃO E DEFORMAÇÃO
- CRITÉRIOS DE FALHA

APLICAÇÃO DE CONDIÇÕES DE CONTORNO USANDO O MÉTODO DE ELEMENTOS FINITOS

MODIFICAÇÃO DA MATRIZ DE RIGIDEZ E DO VETOR DE FORÇA PRA REALIZAR O CÁLCULO DO DESLOCAMENTO

```
function [Knew, Fnew] = KFTransform(obj, K, nodes, Dim)
    Dir = zeros(Dim*size(obj.BCDirichlet, 1), 2);
    Neu = zeros(numel(nodes), 1);


    switch Dim
        case 1
            Dir = obj.BCDirichlet; %x
            Neu = obj.BCNeumann(:, 2); % x

        case 2

            Dir(1:2:end,:) = [(2*obj.BCDirichlet(:, 1) - 1) obj.BCDirichlet(:, 2)]; %x
            Dir(2:2:end, :) = [2*obj.BCDirichlet(:, 1) obj.BCDirichlet(:, 3)]; %y

            Neu((2*obj.BCNeumann(:, 1)-1), :) = obj.BCNeumann(:, 2); % x
            Neu(2*obj.BCNeumann(:, 1), :) = obj.BCNeumann(:, 3); % y


        case 3
            Dir(1:3:end,:) = [(3*obj.BCDirichlet(:, 1) - 2) obj.BCDirichlet(:, 2)]; %x
            Dir(2:3:end, :) = [(3*obj.BCDirichlet(:, 1) - 1) obj.BCDirichlet(:, 3)]; %y
            Dir(3:3:end,:) = [3*obj.BCDirichlet(:, 1) obj.BCDirichlet(:, 4)]; %z

            Neu((3*obj.BCNeumann(:, 1) - 2), :) = obj.BCNeumann(:, 2); % x
            Neu((3*obj.BCNeumann(:, 1) - 1), :) = obj.BCNeumann(:, 3); % y
            Neu(3*obj.BCNeumann(:, 1), :) = obj.BCNeumann(:, 4); % z
    end

    k = zeros(size(Dir, 1), size(K, 2));

    for i = 1:size(Dir, 1)
        if Neu(Dir(i, 1)) ~= 0
            error('Make sure that there are not two types of BC at the same node.')
        end
        k(i, :) = K(Dir(i, 1), :);
        K(Dir(i, 1),:) = 0;
        K(:, Dir(i, 1)) = 0;
        K(Dir(i, 1), Dir(i, 1)) = 1;
    end
    Fnew = (Neu - k'*Dir(:, 2));
    Knew = K;
end
```

CÓDIGO DE TRANSFORMAÇÃO

- RECEBE VETORES DE NEUMANN E DIRICHLET E MODIFICA SEU FORMATO
- A TRANSFORMAÇÃO É DIFERENTE DIFERENTES DIMENSÕES
- REALIZA AS OPERAÇÕES DE MUDANÇA NA MATRIZ DE RIGIDEZ E NO VETOR DE NEUMANN (FORÇA)

CÁLCULO DE TENSÃO E DEFORMAÇÃO

APLICAÇÃO À PARTIR DOS DESLOCAMENTOS NODAIS

```
obj.E = zeros(size(obj.Matrix.Pre.Mesh.Connectivity,1),numel(x),numel(x),3);
obj.S = zeros(size(obj.Matrix.Pre.Mesh.Connectivity,1),numel(x),numel(x),3);

for iel = 1:size(obj.Matrix.Pre.Mesh.Connectivity,1)
    nodes(obj.Matrix.Pre.Mesh.ElOrder) = obj.Matrix.Pre.Mesh.Connectivity(iel,2:end);
    coords = obj.Matrix.Pre.Mesh.Nodes(nodes,:);
    % get the material property
    matC = obj.Matrix.Pre.Material.BuildMat(coords,obj.Matrix.Pre.Material.Hypot);

    n = numel(nodes)*2;
    index(1:2:n) = 2*nodes-1; % x
    index(2:2:n) = 2*nodes; % y

    for iwy = 1 : numel(x)
        for iwx = 1 : numel(x)
            %calc Jacobian
            J = squeeze([dphi(iwx,iwy,:,1) dphi(iwx,iwy,:,2)])' * coords; % aqui calcula-se o J para o ponto iwx,iwy : representa a
            if det(J) <= 0
                error('Negat
            end
            %build N
            aux = (J \ squeeze([dphi(iwx,iwy,:,1) dphi(iwx,iwy,:,2)]))';
            %build B
            B = zeros(3,2*numel(nodes));
            for i = 1 : numel(nodes)
                B(:,(i*2-1):2*i) = [aux(i,1) 0;0 aux(i,2);aux(i,2) aux(i,1)]; % aqui tirar os termos de cizalahamnto. assim a matri
            end

            obj.E(iel,iwy,iwx,:) = B*obj.U(index);%Epsilon
            obj.S(iel,iwy,iwx,:) = matC*squeeze(obj.E(iel,iwy,iwx,:));%Sigma

        end
    end
end
```

```
methods
    %% CONSTRUCTOR
    function obj = MechanicsClass()...
    %% Displacemant
    function CalcDisp(obj)...
    %% Strain and Stress
    function CalcStrainStress(obj)...
    %% Strain and Stress (Mean per element)
    function [epsilon, sigma] = CalcMeanStrainStress(obj)...
    %% Spectral Decomposition (Hydrostatic and Deviator)
    function [Hyd, Dev] = SpectralDecomp(obj, sigma)    ...
    %% Von Mises Criteria
    function [Von] = VonMises(obj, Dev)...
end
```

CRIAÇÃO DA "MECHANICSCLASS"

REÚNE OS MÉTODOS PARA APLICAÇÃO EM PROBLEMAS MECÂNICOS

CÁLCULO DOS DESLOCAMENTOS NODAIS

- Método da modificação de matrizes para aplicação das condições de contorno dentro da classe mecânica
- Vetores de Neumann e Dirichlet definidos da seguinte forma:

[número do nó   valor em X   valor em Y   valor em Z]

- Dependendo da dimensão do problema, os valores nas coordenadas que não fazem parte da dimensão são serão adicionados ao vetor
- O método usado na classe "SolverClass" serve para realizar a operação "F/K" de forma matricial

```
function CalcDisp(obj)
    [K, F] = obj.Matrix.Pre.BC.KFTransform(obj.Matrix.K, obj.Matrix.Pre.Mesh.Nodes, obj.Matrix.Pre.Mesh.Dim);
    obj.U = obj.Solver.SolveDisp(K, F);
end
```

CÁLCULO DAS TENSÕES E DEFORMAÇÕES POR PONTO DE INTEGRAÇÃO

- Método de construção da matriz de operadores "B" semelhante ao adotado para o cálculo da matriz de rigidez
- Valores indexados da seguinte forma:

[Nº do elemento Ponto de int. em X ...em Y ...em Z componente]

- Tendo em vista a aplicação para tensores simétricos, os tensores de deformação e tensão foram reduzidos ao formato de vetor: primeiro vem as componentes da diagonal principal e depois os termos cruzados

```
obj.E(iel,iwx,iwy,iwz, :) = B*obj.U(index);%Epsilon
obj.S(iel,iwx,iwy,iwz, :) = matC*squeeze(obj.E(iel,iwx,iwy,iwz, :));%Sigma
```

CÁLCULO DE TENSÃO E DEFORMAÇÃO MÉDIA PARA CADA ELEMENTO

- Para cada elemento é feita a média aritmética dentre cada componente de todos os seus pontos de integração. Assim os elementos XX por exemplo, são calculados à partir de uma média dos componentes XX de todos os pontos de integração
- As tensões são indexadas em uma matriz, onde o número da linha é o número do elemento, e suas colunas são as componentes do tensor (novamente de forma vetorial)

```
%% Strain and Stress (Mean per element)
function [epsilon, sigma] = CalcMeanStrainStress(obj)
    nel = prod(obj.Matrix.Pre.Mesh.Nel);
    epsilon = zeros(nel ,size(obj.E,(obj.Matrix.Pre.Mesh.Dim + 2)));
    sigma = epsilon;
    for i = 1:nel
        for j = 1:size(obj.E,(obj.Matrix.Pre.Mesh.Dim + 2))
            aux = squeeze(obj.E(i,:,:,j));
            epsilon(i, j) = mean(aux(:));
            aux = squeeze(obj.S(i,:,:,j));
            sigma(i, j) = mean(aux(:));
        end
    end
end
```

DECOMPOSIÇÃO HIDROSTÁTICA

- Separação do estado de tensão de cada elemento em hidrostático e desviador
- Adaptação para o cálculo vetorial, tendo em vista o armazenamento simplificado dos tensores
- Tensores armazenados de forma vetorial

```
%% Spectral Decomposition (Hydrostatic and Deviator)
function [Hyd, Dev] = SpectralDecomp(obj, sigma)
            Hyd = zeros(size(sigma, 1),1);
            Dev = zeros(size(sigma));
            aux = zeros(1, size(sigma, 2));
            aux(1, 1:obj.Matrix.Pre.Mesh.Dim) = 1;
            for i = 1:size(sigma, 1)
                Hyd(i) = (1/3)*(sum(sigma(i,1:obj.Matrix.Pre.Mesh.Dim)));
                Dev(i,:) = sigma(i,:) - Hyd(i)*aux;
            end
end
```

$$\boldsymbol{\sigma}^{dev} = \boldsymbol{\sigma} - \frac{1}{3}(\mathrm{tr}\,\boldsymbol{\sigma})\,\mathbf{I}.$$

```
    function [epsilon, sigma] = CalcMeanStrainStress(obj)
        nel = prod(obj.Matrix.Pre.Mesh.Nel);
        epsilon = zeros(nel ,size(obj.E,(obj.Matrix.Pre.Mesh.Dim + 2)));
        sigma = epsilon;
        for i = 1:nel
            for j = 1:size(obj.E,(obj.Matrix.Pre.Mesh.Dim + 2))
                aux = squeeze(obj.E(i,:,:,j));
                epsilon(i, j) = mean(aux(:));
                aux = squeeze(obj.S(i,:,:,j));
                sigma(i, j) = mean(aux(:));
            end
        end
    end
    %% Spectral Decomposition (Hydrostatic and Deviator)
    function [Hyd, Dev] = SpectralDecomp(obj, sigma)
                Hyd = zeros(size(sigma, 1),1);
                Dev = zeros(size(sigma));
                aux = zeros(1, size(sigma, 2));
                aux(1, 1:obj.Matrix.Pre.Mesh.Dim) = 1;
                for i = 1:size(sigma, 1)
                    Hyd(i) = (1/3)*(sum(sigma(i,1:obj.Matrix.Pre.Mesh.Dim)));
                    Dev(i,:) = sigma(i,:) - Hyd(i)*aux;
                end
    end
    %% Von Mises Criteria
    function [Von] = VonMises(obj, sigma)
        Von = zeros(prod(obj.Matrix.Pre.Mesh.Nel), 1);
        for i = 1:prod(obj.Matrix.Pre.Mesh.Nel)
            switch obj.Matrix.Pre.Mesh.Dim
                case 1
                    Von(i, 1) = sqrt(0.5*((sigma(i, 1))^2+(-sigma(i,1))^2));
                case 2
                    Von(i, 1) = sqrt(0.5*((sigma(i, 1)-sigma(i,2))^2+(sigma(i,2))^2+(-sigma(i,1))^2+6*(sigma(i,3)^2)));
                case 3
                    Von(i, 1) = sqrt(0.5*((sigma(i, 1)-sigma(i,2))^2+(sigma(i,2)-sigma(i,3))^2+(sigma(i,3)-sigma(i,1))^2+6*(sigma(i,4)^2+sigma(i,5)^2+sigma
            end
        end
    end
  end
end
```

CRITÉRIOS DE FALHA

- VON MISES

FORMULAÇÃO À PARTIR DO TENSOR DE TENSÕES

$$\sigma_v = \sqrt{\frac{1}{2}[(\sigma_{11} - \sigma_{22})^2 + (\sigma_{22} - \sigma_{33})^2 + (\sigma_{33} - \sigma_{11})^2 + 6(\sigma_{12}^2 + \sigma_{23}^2 + \sigma_{31}^2)]}$$

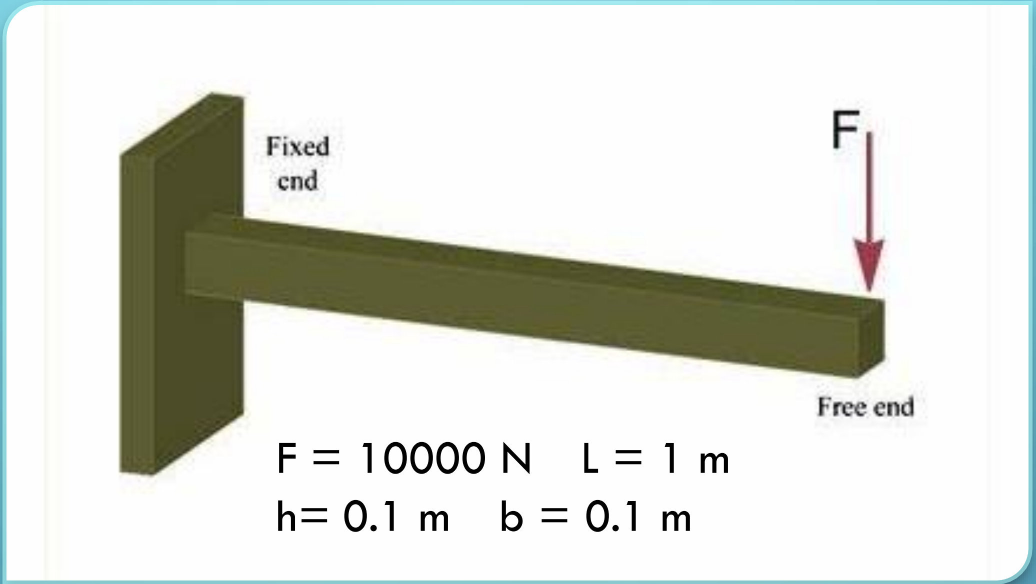
```
%% Von Mises Criteria
function [Von] = VonMises(obj, sigma)
    Von = zeros(prod(obj.Matrix.Pre.Mesh.Nel), 1);
    for i = 1:prod(obj.Matrix.Pre.Mesh.Nel)
        switch obj.Matrix.Pre.Mesh.Dim
            case 1
                Von(i, 1) = sqrt(0.5*((sigma(i, 1))^2+(-sigma(i,1))^2));
            case 2
                Von(i, 1) = sqrt(0.5*((sigma(i, 1)-sigma(i,2))^2+(sigma(i,2))^2+(-sigma(i,1))^2+6*(sigma(i,3)^2)));
            case 3
                Von(i, 1) = sqrt(0.5*((sigma(i, 1)-sigma(i,2))^2+(sigma(i,2)-sigma(i,3))^2+(sigma(i,3)-sigma(i,1))^2+6*(sigma(i,4)^2+sigma(i,5)^2+sigma(i,6)^2)));
        end
    end
end
```

RESULTADOS (ILUSTRATIVOS)

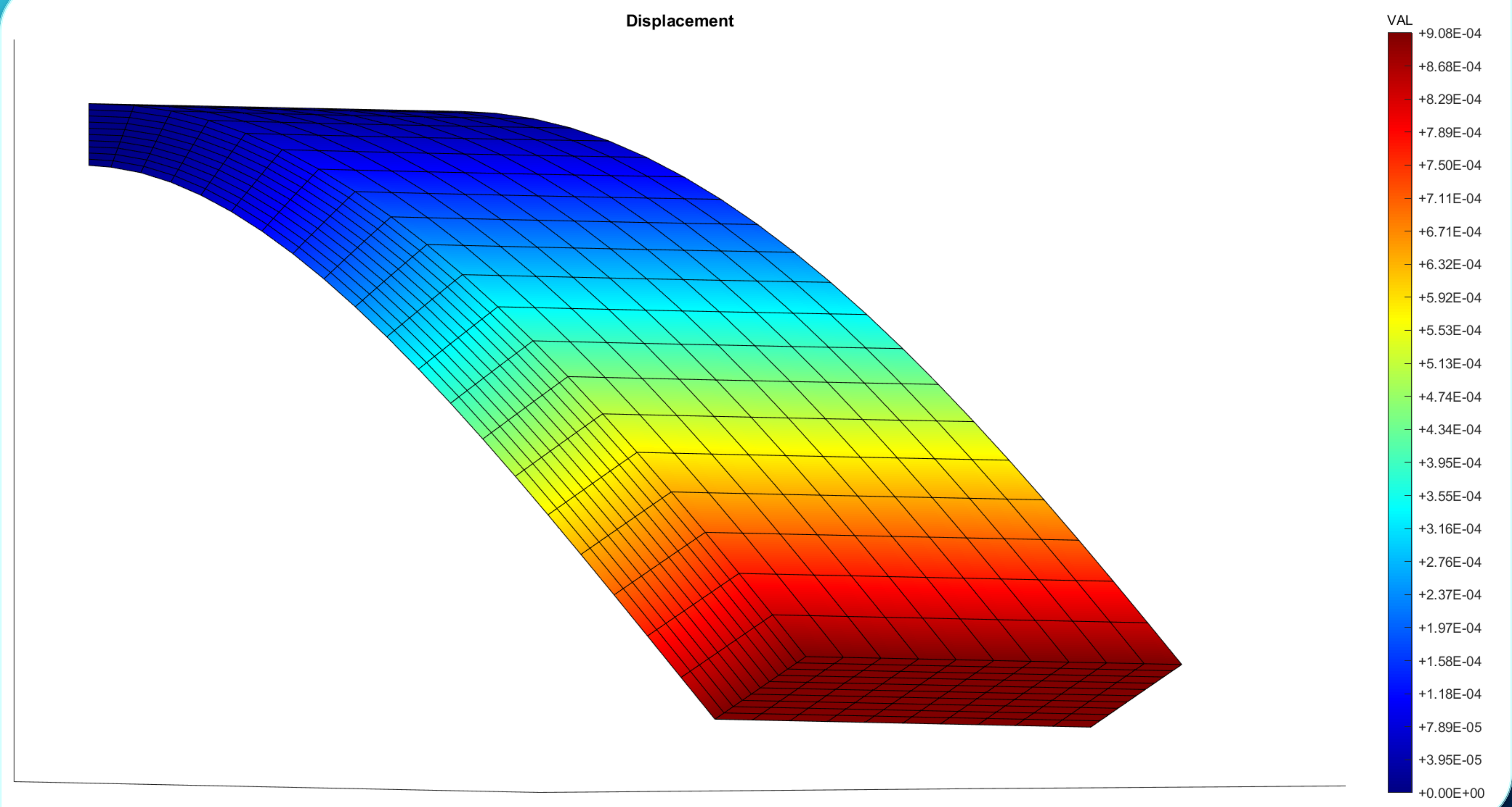
- DESLOCAMENTOS
- VON MISES

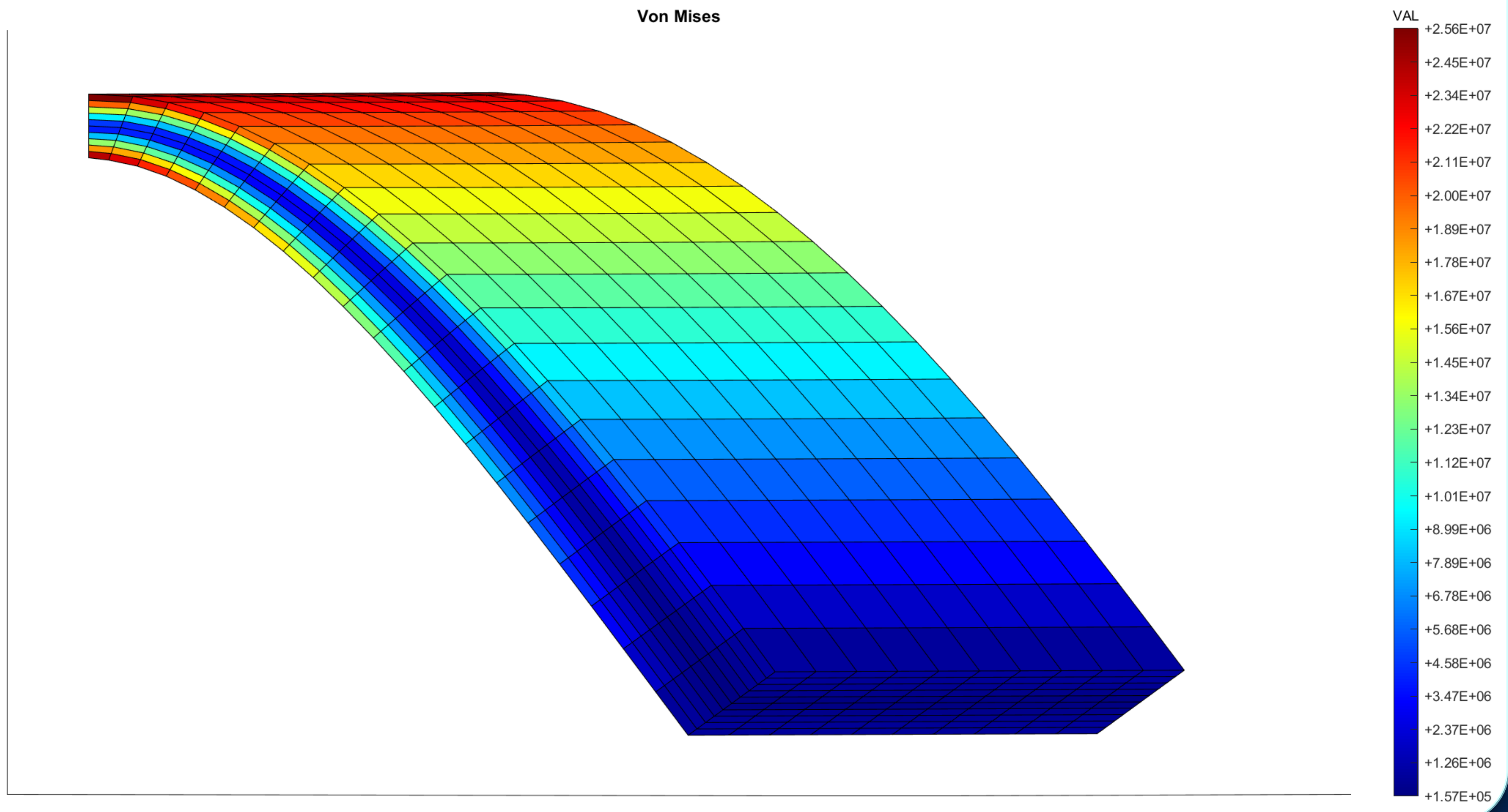
```
function [epsilon, sigma] = CalcMeanStrainStress(obj)
    nel = prod(obj.Matrix.Pre.Mesh.Nel);
    epsilon = zeros(nel ,size(obj.E,(obj.Matrix.Pre.Mesh.Dim + 2)));
    sigma = epsilon;
    for i = 1:nel
        for j = 1:size(obj.E,(obj.Matrix.Pre.Mesh.Dim + 2))
            aux = squeeze(obj.E(i,:,:,j));
            epsilon(i, j) = mean(aux(:));
            aux = squeeze(obj.S(i,:,:,j));
            sigma(i, j) = mean(aux(:));
        end
    end
end
%% Spectral Decomposition (Hydrostatic and Deviator)
function [Hyd, Dev] = SpectralDecomp(obj, sigma)
            Hyd = zeros(size(sigma, 1),1);
            Dev = zeros(size(sigma));
            aux = zeros(1, size(sigma, 2));
            aux(1, 1:obj.Matrix.Pre.Mesh.Dim) = 1;
            for i = 1:size(sigma, 1)
                Hyd(i) = (1/3)*(sum(sigma(i,1:obj.Matrix.Pre.Mesh.Dim)));
                Dev(i,:) = sigma(i,:) - Hyd(i)*aux;
            end
end
%% Von Mises Criteria
function [Von] = VonMises(obj, sigma)
    Von = zeros(prod(obj.Matrix.Pre.Mesh.Nel), 1);
    for i = 1:prod(obj.Matrix.Pre.Mesh.Nel)
        switch obj.Matrix.Pre.Mesh.Dim
            case 1
                Von(i, 1) = sqrt(0.5*((sigma(i, 1))^2+(-sigma(i,1))^2));
            case 2
                Von(i, 1) = sqrt(0.5*((sigma(i, 1)-sigma(i,2))^2+(sigma(i,2))^2+(-sigma(i,1))^2+6*(sigma(i,3)^2)));
            case 3
                Von(i, 1) = sqrt(0.5*((sigma(i, 1)-sigma(i,2))^2+(sigma(i,2)-sigma(i,3))^2+(sigma(i,3)-sigma(i,1))^2+6*(sigma(i,4)^2+sigma(i,5)^2+sigma
        end
    end
end
  end
end
```

PARÂMETROS




```
%% Mesh
or = 1; % polynimial order
dim = 3; % probelm dimention
nel = [20 10 10]; % number of elements [x y z]
domainsize = [0 1; 0 0.1; 0 0.1]; % [x0 x; y0 y; z0 z]
type = 'FEM'; % method type
coordsyst = 'Cartesian'; % coordinate system of the problem
```


Displacement
VAL
+9.08E-04
+8.68E-04
+8.29E-04
+7.89E-04
+7.50E-04
+7.11E-04
+6.71E-04
+6.32E-04
+5.92E-04
+5.53E-04
+5.13E-04
+4.74E-04
+4.34E-04
+3.95E-04
+3.55E-04
+3.16E-04
+2.76E-04
+2.37E-04
+1.97E-04
+1.58E-04
+1.18E-04
+7.89E-05
+3.95E-05
+0.00E+00


Von Mises
VAL
+2.56E+07
+2.45E+07
+2.34E+07
+2.22E+07
+2.11E+07
+2.00E+07
+1.89E+07
+1.78E+07
+1.67E+07
+1.56E+07
+1.45E+07
+1.34E+07
+1.23E+07
+1.12E+07
+1.01E+07
+8.99E+06
+7.89E+06
+6.78E+06
+5.68E+06
+4.58E+06
+3.47E+06
+2.37E+06
+1.26E+06
+1.57E+05