trellis

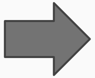
Software livre de dependências para análise de estruturas e treliças

Lendo o arquivo


```python
class FileIn:
    def __init__(self, path):
        self.path = path
        self.number_of_element_groups = None
        self.coordinates = self.getInfo("COORDINATES")
        self.element_groups = self.getElement_groups()
        self.incidences = self.getIncidences()  # todo: better function
        self.materials = self.getInfo("MATERIALS")
        self.geometric_properties = self.getInfo("GEOMETRIC_PROPERTIES")
        self.bc_nodes = self.getInfo("BCNODES")
        self.loads = self.getInfo("LOADS")
```

Preparando para análise

File object



Element object

```python
class Element:
    def __init__(self, element_id, incidence, l, area, c, s, e, restrictions_dof):
        self.element_id = element_id
        self.incidence = incidence
        self.length = l
        self.area = area
        self.cos = c
        self.sin = s
        self.e = e

        self.m = None
        self.rigid = self.computeRigid()
        self.restrictions_dof = restrictions_dof
        self.transformation_matrix = self.computeTransformationMatrix()
```

Funções do Programa

```
def computeGlobalRigid(truss, list_of_elements):
def computeRestrictedDofs(truss):
def computeCleanGlobalRigid(global_rigid_matrix, restricted_dofs):
def computeDisplacementsMatrix(truss, clean_rigid_matrix, restricted_dofs, method,
max_iterations):
def computeReactionForces(m_global, displacement_matrix, loads, number_of_nodes):
def computeStressesStrains(list_of_elements, displacement_matrix):
```

```
python3 main.py -i <arquivo de entrada>.txt -o <arquivo de saida>.txt -m <método de solução> -n <número máximo de iterações>
```

Etapa 1 - Calculando matriz global

```
def computeGlobalRigid(truss, list_of_elements)
```

$$[K_1] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 \end{bmatrix} \times 10^5$$

$$[K_2] = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \times 10^5$$

$$\Rightarrow$$

$$[K_g] = \frac{1}{2} \times 10^5 \begin{bmatrix} 3 & 1 & 0 & 0 & -2 & 0 & -1 & -1 \\ 1 & 3 & 0 & -2 & 0 & 0 & -1 & -1 \\ 0 & 0 & 3 & -1 & -1 & 1 & -2 & 0 \\ 0 & -2 & -1 & 3 & 1 & -1 & 0 & 0 \\ -2 & 0 & -1 & 1 & 3 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 3 & 0 & -2 \\ -1 & -1 & -2 & 0 & 0 & 0 & 3 & 1 \\ -1 & -1 & 0 & 0 & 0 & -2 & 1 & 3 \end{bmatrix}$$

Percorre as matrizes de rigidez somando cada valor em seu devido lugar na global

Etapa 2 - Aplicando condições de contorno

```
def computeRestrictedDofs(truss)
```

```
def computeCleanGlobalRigid(global_rigid_matrix, restricted_dofs)
```

Graus de liberdade restritos

↓

linhas e colunas a serem cortadas da matriz global

percorre a matriz, preenchendo uma nova, sem os elementos das linhas e colunas cortadas

$$\frac{1}{2}\times 10^{5}\begin{bmatrix} 3 & 1 & -2 & 0 & 0 & 0 & -1 & -1 \\ 1 & 3 & 0 & 0 & 0 & -2 & -1 & -1 \\ -2 & 0 & 3 & -1 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 3 & 1 & -1 & 0 & -2 \\ 0 & 0 & -1 & 1 & 3 & -1 & -2 & 0 \\ 0 & -2 & 1 & -1 & -1 & 3 & 0 & 0 \\ -1 & -1 & 0 & 0 & -2 & 0 & 3 & 1 \\ -1 & -1 & 0 & -2 & 0 & 0 & 1 & 3 \end{bmatrix}\begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -1000 \end{Bmatrix}$$

Etapa 3 - Calculando os deslocamentos

```
def computeDisplacementsMatrix(truss, clean_rigid_matrix, restricted_dofs, method, max_iterations):
```

$$\{P_g\} = \{\ 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad -1000\ \}^T$$

$$[K_g] = \frac{1}{2} \times 10^5 \begin{bmatrix} 3 & -1 & 0 & 0 \\ -1 & 3 & 0 & -2 \\ 0 & 0 & 3 & 1 \\ 0 & -2 & 1 & 3 \end{bmatrix}$$

$$\{P_g\} = \{\ 0 \quad 0 \quad 0 \quad -1000\ \}^T$$

$$[K_g]\{U\} = \{P_g\}$$

$$\frac{1}{2} \times 10^5 \begin{bmatrix} 3 & -1 & 0 & 0 \\ -1 & 3 & 0 & -2 \\ 0 & 0 & 3 & 1 \\ 0 & -2 & 1 & 3 \end{bmatrix} \begin{Bmatrix} U_5 \\ U_6 \\ U_7 \\ U_8 \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \\ -1000 \end{Bmatrix}$$

Usa um dos métodos numéricos para calcular o vetor de deslocamentos

Métodos numéricos

Classe Methods.py

Jacobi

```
def Jacobi(it, tolerance, a, f)
```

Gauss-Seidel

```
def Gauss-Seidel(it, tolerance, a, f)
```

Em ambos casos são usados 2 vetores: um da aproximação atual e um da aproximação da iteração anterior

Etapa 4 - Calculando as forças de reação

```python
def computeReactionForces(m_global, displacement_matrix, loads, number_of_nodes):
```

$$\begin{Bmatrix} R_1 \\ R_2 \\ R_3 \\ R_4 \\ 0 \\ 0 \\ 0 \\ -1000 \end{Bmatrix} = \frac{1}{2}\times 10^5 \begin{bmatrix} 3 & 1 & 0 & 0 & -2 & 0 & -1 & -1 \\ 1 & 3 & 0 & -2 & 0 & 0 & -1 & -1 \\ 0 & 0 & 3 & -1 & -1 & 1 & -2 & 0 \\ 0 & -2 & -1 & 3 & 1 & -1 & 0 & 0 \\ -2 & 0 & -1 & 1 & 3 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 3 & 0 & -2 \\ -1 & -1 & -2 & 0 & 0 & 0 & 3 & 1 \\ -1 & -1 & 0 & 0 & 0 & -2 & 1 & 3 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \\ U_5 \\ U_6 \\ U_7 \\ U_8 \end{Bmatrix}$$

$$R_1 = 1000,\ \ R_2 = 571.73,\ \ R_3 = -1000,\ \ R_4 = 428.57$$

Multiplica a matriz global pela de deslocamentos, para produzir um vetor com as forças

Etapa 5 - Calculando a tensão e deformação

```
def computeStressesStrains(list_of_elements, displacement_matrix):
```

Deformação Específica $\epsilon = \frac{1}{l}\begin{bmatrix} -c & -s & c & s \end{bmatrix}\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{Bmatrix}$

Tensão $\sigma = \frac{E}{l}\begin{bmatrix} -c & -s & c & s \end{bmatrix}\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \end{Bmatrix}$

Validação

- Dados da apresentação de slides da aula
- Dados do enunciado do projeto
- Dados do quiz

Demo