**ESTRUCTURES AEROESPACIALS**

# Algorithm for solving structural problems

The scheme in Figure 1 is used for solving structural problems numerically. While the algorithm is the same for all kinds of problems involving the evaluation the deformation and stress state of a (quasi)static structure, the steps in each block may differ depending on the problem's dimensions, type of elements considered, etc.

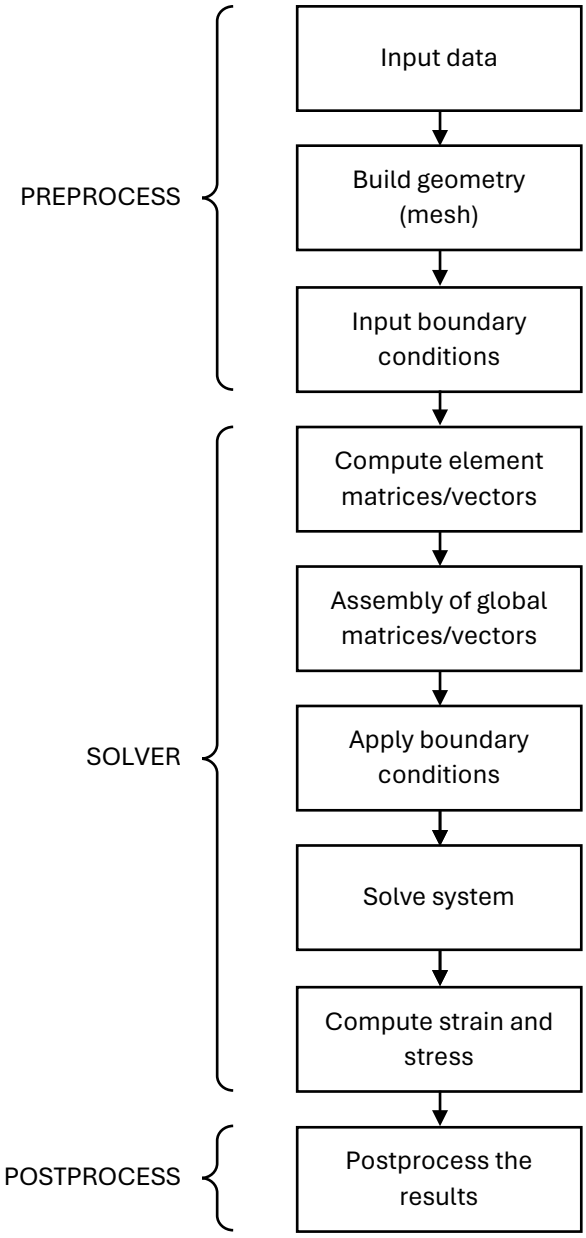


Figure 1. General structural problems resolution algorithm.

# 1 Preprocess

In this section we define the problem and produce the data required by the solver in a comprehensive and compatible format.

## 1.1 Input data

First of all, it is important to identify the key parameters that define the problem. These can be geometrical features, material properties, physical constants, or numerical parameters. Ideally, this should be the only part of the algorithm that one needs to change when executing each run. This guarantees that the rest of the algorithm is general and capable of dealing with similar kinds of problems.

Among the relevant data to define here, one key parameter is the number of degrees of freedom (DOF) associated with each point in our domain, $n_{\mathrm{i}}$. For example, in a 2D bars structure, each node can move horizontally and vertically, so in this case we would define $n_{\mathrm{i}}$ = 2 DOFs for each point.

***MATLAB Tip* #1** *Use of structure arrays for passing input data to functions*

Since it is likely that the problem's data is needed in several parts of the code, to avoid having to pass as input each individual variable it is useful to define a **structure array** and pass it instead. Example:

Instead of defining data variables as

```
a = 1;
b = 1;
c = 1;
some_function(a,b,c);
```

Define them using a `data` structure array

```
data.a = 1;
data.b = 1;
data.c = 0;
some_function(data);
```

## 1.2 Build geometry (mesh)

Another aspect involved in the definition of our problem is the structure's geometry. In order to solve it numerically, it is necessary to split it into a finite number of elements. This is done by first defining a set of *nodes* (points in the spatial domain). Then, each element is defined as a subset of nodes determining its shape. The type/s of elements will depend on the specific problem's dimensions and structural model required in the solver part. The process involved in doing so is called *meshing* and the result is known as the problem's *mesh*. Regardless of how the mesh is obtained—either by hand or as an output of some specific meshing software—the following information is expected:

- Nodal coordinates matrix, $[\mathbf{x}]$. Each coefficient $x_{ij}$ in this matrix represents the coordinate of the $i$-th node in the $j$-th dimension ($x$, $y$ and $z$ components in a Cartesian reference frame).
- Nodal connectivities matrix, $[\mathbf{T}^{\mathrm{n}}]$. Each coefficient $T_{ij}^{\mathrm{n}}$ in this matrix provides the index of the node (i.e., the row in the $[\mathbf{x}]$ matrix) associated with the $j$-th vertex of the $i$-th element.

Other information that is typically part of the matrix is the definition of additional indices that are used to distinguish subsets of elements with distinct features, like, for example, different material properties. For that, the following matrices are also defined:

- Material properties matrix, $[\mathbf{m}]$. Each column in this matrix is associated with a different material property or any other relevant problem parameter that can be linked to the different elements.

Each row corresponds to any possible unique combination of column parameters that exists for the different elements in the problem.

- Material connectivities vector, $\{\mathbf{T}^{\mathrm{m}}\}$. Similar to the $[\mathbf{T}^{\mathrm{n}}]$ matrix, each coefficient $T_i^{\mathrm{m}}$ represents the index of the material (i.e., the row in the $[\mathbf{m}]$ matrix) associated with the $i$-th element.

Notice that the dimensions of these matrices give some relevant data for the problem, namely:

- Problem's dimensions, $n_{\mathrm{d}}$ = number of columns of $[\mathbf{x}]$.
- Total number of nodes, $n_{\mathrm{nod}}$ = number of rows of $[\mathbf{x}]$.
- Total number of elements, $n_{\mathrm{el}}$ = number of rows of either $[\mathbf{T}^{\mathrm{n}}]$ or $\{\mathbf{T}^{\mathrm{m}}\}$.
- Number of vertices of each element, $n_{\mathrm{ne}}$ = number of columns of $[\mathbf{T}^{\mathrm{n}}]$.

Since the number of rows of $[\mathbf{T}^{\mathrm{n}}]$ and $\{\mathbf{T}^{\mathrm{m}}\}$ is the same, sometimes a single connectivities matrix $[\mathbf{T}]$ is defined instead including both the coefficients of $[\mathbf{T}^{\mathrm{n}}]$ and $\{\mathbf{T}^{\mathrm{m}}\}$ (with the latter being given as the either the first or the last column of $[\mathbf{T}]$). Note also that different connectivities matrices can be defined for each different type of element involved in the problem.

***Remark* #1** *Distinction between nodes and degrees of freedom*

Since we will be evaluating the problem's solution at the mesh's nodes and we allow the existence of $n_{\mathrm{i}}$ DOFs at each spatial point, we need an additional indexing system to identify each individual DOF of each node. The straightforward option is to incrementally build this DOF indexing system following the same nodal indexing order. For example, if we have $n_{\mathrm{nod}}$ nodes defined by the indices vector $\{1, 2, \dots, n_{\mathrm{nod}}\}^{\mathrm{n}}$, and our problem has $n_{\mathrm{i}} = 2$, then we can build the global DOFs indices vector as $\{1, 2, 3, 4, \dots, n_{\mathrm{dof}}\}^{\mathrm{d}}$, where $\{1, 2\}^{\mathrm{d}}$ will be the DOFs associated with node $\{1\}^{\mathrm{n}}$, $\{3, 4\}^{\mathrm{d}}$ the DOFs associated with node $\{2\}^{\mathrm{n}}$, etc. Notice that the total number of DOFs of the problem will then be $n_{\mathrm{dof}} = n_{\mathrm{nod}} \times n_{\mathrm{i}}$. This indexing system allows to define a function $f^{\mathrm{n}\to\mathrm{d}}$ that, for a defined $n_{\mathrm{i}}$ value, returns the global index $I$ corresponding to a given nodal index $i$ and a local degree of freedom $j$:

$$I = f^{\mathrm{n}\to\mathrm{d}}(n_{\mathrm{i}}, i, j)$$

Using this function, it is possible to build the corresponding DOFs connectivities matrix $\left[\mathbf{T}^{\mathrm{d}}\right]$ associated with $[\mathbf{T}^{\mathrm{n}}]$. In this case, each column of $\left[\mathbf{T}^{\mathrm{d}}\right]$ provides the DOF indices for the nodal indices of its $[\mathbf{T}^{\mathrm{n}}]$ counterpart. Hence, the number of columns of $\left[\mathbf{T}^{\mathrm{d}}\right]$ is extended to $n_{\mathrm{ne}} \times n_{\mathrm{i}}$.

### 1.3 Input boundary conditions

The last block involved in the problem's definition deals with how to input the information about the boundary conditions. In structural problems these typically are the prescribed DOFs and external forces. To do so, the following matrices are defined:

- Prescribed DOFs matrix, $[\mathbf{p}]$. Each row of $[\mathbf{p}]$ corresponds to a prescribed DOF (not node, i.e., if one node has more than one of its DOFs prescribed, then each will be in its own separate row). The columns will provide information about the nodal index (1st column), the local degree of freedom (2nd column), and the value (3rd column) of the prescribed DOF.

- Point loads matrix, $[\mathbf{F}]$. The structure of this matrix is very similar to that of $[\mathbf{p}]$. Similarly, the 1st and 2nd column give the nodal index and local degree of freedom where each non-zero point load is acting and the value is given in the 3rd column. Again, each separate component of the loads (even if they are acting on the same node) are given in a separate column.

Additional data matrices can be defined here depending on the specific problem setting. For example, if body forces are present, sometimes it might be useful to define a matrix $[\mathbf{b}]$ in which the first row identifies the element indices where they are acting and subsequent columns provide additional data for the element force evaluation.

# 2 Solver

In this section the data produced in the preprocess is used to solve the structural problem, i.e., compute the nodal DOFs, reaction loads at the prescribed nodes and strain and stress of each element.

## 2.1 Compute element matrices/vectors

### 2.1.1 Element stiffness matrices computation

Regardless of the type of element or structural model considered, the following for-loop algorithm is typically used to get the stiffness matrix of each element:

1. Initialize the tri-dimensional matrix structure where the element stiffness matrix of each element will be stored:
   $\left[\mathbf{K}^{\mathrm{el}}\right] = [\mathbf{0}]_{(n_{\mathrm{ne}} \cdot n_{\mathrm{i}}) \times (n_{\mathrm{ne}} \cdot n_{\mathrm{i}}) \times n_{\mathrm{el}}}$
2. In a loop over each element $e$ (from 1 to $n_{\mathrm{el}}$):
   1. Evaluate the element stiffness matrix for the specific structural model considered in terms of the mesh data:
      $[\mathbf{K}^{*}] = \textbf{\textit{stiffness\_function}}([\mathbf{x}], [\mathbf{T}^{\mathrm{n}}], [\mathbf{m}], [\mathbf{T}^{\mathrm{m}}], \dots)$
   2. Assign the element stiffness matrix to $\left[\mathbf{K}^{\mathrm{el}}\right]$:
      $\left[\mathbf{K}^{\mathrm{el}}(:,:,e)\right] = [\mathbf{K}^{*}]$

### 2.1.2 Element force vectors computation

The algorithm to build the element force vectors (if any) is analogous to the stiffness matrix case:

1. Initialize the bi-dimensional matrix structure where the element force vector of each element will be stored in each column:
   $\left[\mathbf{f}^{\mathrm{el}}\right] = [\mathbf{0}]_{(n_{\mathrm{ne}} \cdot n_{\mathrm{i}}) \times n_{\mathrm{el}}}$
2. In a loop over each element $e$ (from 1 to $n_{\mathrm{el}}$):
   1. Evaluate the element force vector for the specific problem in terms of the mesh data and other data parameters:
      $\{\mathbf{f}^{*}\} = \textbf{\textit{force\_function}}([\mathbf{x}], [\mathbf{T}^{\mathrm{n}}], [\mathbf{m}], [\mathbf{T}^{\mathrm{m}}], \dots)$
   2. Assign the element force vector to $\left[\mathbf{f}^{\mathrm{el}}\right]$:
      $\left\{\mathbf{f}^{\mathrm{el}}(:,e)\right\} = \{\mathbf{f}^{*}\}$

## 2.2 Assembly of global matrices/vectors

The assembly process of the stiffness matrix and forces vector consists in placing the computed $\left[\mathbf{K}^{\mathrm{el}}\right]$ and $\left[\mathbf{f}^{\mathrm{el}}\right]$ into the corresponding rows and columns of their global counterparts. The algorithm for this is the same regardless of the structural problem, since it only requires having a nodal-to-DOFs function $f^{\mathrm{n}\to\mathrm{d}}$ defined. In what follows we will assume we previously built the DOFs connectivities matrix $\left[\mathbf{T}^{\mathrm{d}}\right]$.

1. Initialize the global stiffness matrix and force vector with zeros:
   $[\mathbf{K}] = [\mathbf{0}]_{n_{\mathrm{dof}} \times n_{\mathrm{dof}}}$
   $\{\mathbf{f}\} = \{\mathbf{0}\}_{n_{\mathrm{dof}} \times 1}$
2. In a loop over each element $e$ (from 1 to $n_{\mathrm{el}}$):
   1. In a loop over each element DOF $i$ (rows from 1 to $n_{\mathrm{ne}} \times n_{\mathrm{i}}$):
      1. Add the element force vector to the global force vector:
         $\mathbf{f}\left(\mathbf{T}^{\mathrm{d}}(e,i)\right) = \mathbf{f}\left(\mathbf{T}^{\mathrm{d}}(e,i)\right) + \mathbf{f}^{\mathrm{el}}(i,e)$
      2. In a loop over each element DOF $j$ (columns from 1 to $n_{\mathrm{ne}} \times n_{\mathrm{i}}$):
         1. Add the element stiffness matrix to the global stiffness matrix:
            $\mathbf{K}\left(\mathbf{T}^{\mathrm{d}}(e,i), \mathbf{T}^{\mathrm{d}}(e,j)\right) = \mathbf{K}\left(\mathbf{T}^{\mathrm{d}}(e,i), \mathbf{T}^{\mathrm{d}}(e,j)\right) + \mathbf{K}^{\mathrm{el}}(i,j,e)$

## 2.3 Apply boundary conditions

If we have the $[\mathbf{p}]$ and $[\mathbf{F}]$ matrices defined, then the algorithm to apply the boundary conditions is also the same regardless of the problem.

### 2.3.1 Prescribed DOFs

1. Denoting $n_\mathrm{p}$ the number of rows of $[\mathbf{p}]$, initialize the prescribed DOFs vectors:
   $\{\mathbf{v}^\mathrm{p}\} = \{\mathbf{0}\}_{n_\mathrm{p}\times 1}$
   $\{\mathbf{u}^\mathrm{p}\} = \{\mathbf{0}\}_{n_\mathrm{p}\times 1}$
2. In a loop over each row $i$ of $[\mathbf{p}]$ (from 1 to $n_\mathrm{p}$):
   1. Determine the prescribed DOF index:
      $\mathbf{v}^\mathrm{p}(i) = f^{\mathrm{n}\to\mathrm{d}}\big(n_\mathrm{i}, \mathbf{p}(i,1), \mathbf{p}(i,2)\big)$
   2. Assign the prescribed DOF value:
      $\mathbf{u}^\mathrm{p}(i) = \mathbf{p}(i,3)$

### 2.3.2 Point loads

1. In a loop over each row $i$ of $[\mathbf{F}]$:
   1. Determine the associated point load DOF index:
      $I = f^{\mathrm{n}\to\mathrm{d}}\big(n_\mathrm{i}, \mathbf{F}(i,1), \mathbf{F}(i,2)\big)$
   2. Add the point load to the global forces vector:
      $\mathbf{f}(I) = \mathbf{F}(i,3)$

## 2.4 Solve system

Again, the algorithm for solving the system is the same for all problems, as long as we have defined the global stiffness matrix $[\mathbf{K}]$, the global external forces vector $\{\mathbf{f}\}$ and the vectors of prescribed DOFs indices and values, $\{\mathbf{v}^\mathrm{p}\}$ and $\{\mathbf{u}^\mathrm{p}\}$.

1. Determine the free DOFs indices vector:
   $\{\mathbf{v}^\mathrm{f}\} = \{1, \dots, n_\mathrm{dof}\}^\mathrm{T} - \{\mathbf{v}^\mathrm{p}\}$

***MATLAB Tip #2*** *Subtract one set from another*

The subtracting operation of one set of indices from another can be done automatically in MATLAB using the built-in function `setdiff`:

```
vf = setdiff((1:ndof)',vp);
```

The output of this function is a vector containing all the indices in `(1:ndof)'` that are not members of the vp vector.

2. Initialize the global DOFs vector:
   $\{\mathbf{u}\} = \{\mathbf{0}\}_{n_\mathrm{dof}\times 1}$
3. Assign the prescribed DOFs to the global DOFs vector:
   $\{\mathbf{u}(\mathbf{v}^\mathrm{p})\} = \{\mathbf{u}^\mathrm{p}\}$
4. Compute the free DOFs:
   $\{\mathbf{u}(\mathbf{v}^\mathrm{f})\} = \big[\mathbf{K}(\mathbf{v}^\mathrm{f}, \mathbf{v}^\mathrm{f})\big]^{-1}\big(\{\mathbf{f}(\mathbf{v}^\mathrm{f})\} - \big[\mathbf{K}(\mathbf{v}^\mathrm{f}, \mathbf{v}^\mathrm{p})\big]\{\mathbf{u}(\mathbf{v}^\mathrm{p})\}\big)$
5. Compute the reaction loads at the prescribed DOFs:
   $\{\mathbf{r}\} = [\mathbf{K}(\mathbf{v}^\mathrm{p}, :)]\{\mathbf{u}\} - \{\mathbf{f}(\mathbf{v}^\mathrm{p})\}$

***Remark #2*** *Stiffness matrix singularity*

Notice that solving the system requires inverting the stiffness sub-matrix associated to free DOFs only. For this inverse to exist the boundary conditions need to be properly applied, otherwise this sub-matrix will be singular and no solution will be obtained.

### 2.5 Compute strain and stress

Once the nodal DOFs have been obtained, it is now possible to determine the stress state of each element. The general algorithm structure to do so is the same regardless of the problem, but the specific function to evaluate the strains and stresses will depend on the structural model considered.

1. Initialize the stress vector (one stress value for each element is obtained):
   $\{\boldsymbol{\sigma}\} = \{\mathbf{0}\}_{n_{\mathrm{el}}\times 1}$
2. In a loop over each element $e$ (from 1 to $n_{\mathrm{el}}$):
   a. Initialize the element's vertices DOFs vector:
      $\{\mathbf{u}^{\mathrm{el}}\} = \{\mathbf{0}\}_{(n_{\mathrm{ne}}\cdot n_{\mathrm{i}})\times 1}$
   b. In a loop over each element DOF $i$ (from 1 to $n_{\mathrm{ne}} \times n_{\mathrm{i}}$):
      i. Assign the corresponding DOF value:
         $\mathbf{u}^{\mathrm{el}}(i) = \mathbf{u}\left(\mathbf{T}^{\mathrm{d}}(e,i)\right)$
   c. Compute the stress according to the specific structural model considered:
      $\boldsymbol{\sigma}(e) =$ ***stress_function***$([\mathbf{x}], [\mathbf{T}^{\mathrm{n}}], [\mathbf{m}], [\mathbf{T}^{\mathrm{m}}], \{\mathbf{u}^{\mathrm{el}}\}, \dots)$

***Remark* #3** *Stress state generalization*

Here we assumed that the stress state can be determined by a single for each element. In a more general context, we can have different values of the stress for the same element, either because we evaluate it at different points within the element, or because the structural model involves different components of the stress tensor (normal and tangential stresses). In these cases, we simply need to define additional variables, but the global algorithm is the same.

## 3 Postprocess

### 3.1 Postprocess the results

The last step in the resolution algorithm is related to data processing and representation. There are no specific steps involved in this part. As part of the postprocessing, one might have to assess some failure criteria, put the data in a specific format (to process it as part of another problem or to transfer it to another software), or produce plots for visualizing the results.