Lehigh University

Advanced Technology for Large Structural Systems

CEE 466 – Advanced Finite Element Methods

User Manual – Finite Element Analysis Program

By: Soheil Sadeghi

Instructor: Dr. Paolo Bocchini

December 2017

The finite element code is organized as one main file and several subroutines that are called through the main file. The main file is Main_file.m in the package. The operator needs to enter all inputs in different sections, which are introduced in the following part. The manual is designed in a way to scan over all parts of the program using the second verifying example as the illustrative. The problem definition of this example is shown below:

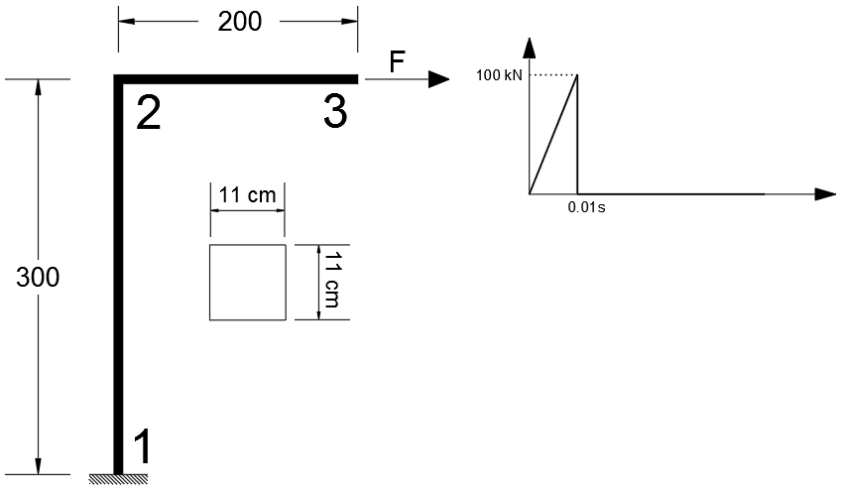

Figure 1. Example two layout

The first INPUT file that has to be completed beforehand is the nodal_coordinate.m file. In this file, the coordinates for all nodes should be entered. In addition, if any nodal loads are applied, these should be introduced as a nodal load matrix p. Each row in p consists three values, indicating nodal forces in x, y and rotational direction. Note that the operator should be consistent in units they use. As a presumed unit set, N and cm are used for all the verifying examples. Please notice that for the time history analysis, all components of the p matrix at each time step will be multiplied by the load factor (which will be explained later).

In addition, the operator needs to enter initial conditions at all DOFs as u0 and u_dot0 variables. The same notation as p matrix is used for these two input matrices. For the illustrative example, the input is as follows:

```
% ------------------ SECOND EXAMPLE -------------------------
ele_num = 5/0.1;                        % calc. the number elements
del = 10;
i = 0:del:300;
y = [i 300*ones(1,20)];
j = 0:del:200;
x = [zeros(1,30) j(1:end)];
p = [zeros(ele_num,3);100e3 0 0];       % Px, Py and moment at node 1,
                                          respectively, in N and cm
u0 = zeros(ele_num+1,3);
u_dot0 = zeros(ele_num+1,3);
```

The next step to input data is to assign elements properties. As described for the first project, the program is set in a way that the operator just needs to call labels associated to the desired material or section and the program automatically call all parameters needed from the designated libraries. The material library is material.m that has all material definitions needed for the verifying examples. For illustration, the material and section assignment input is shown in the box below:

```
% ------------ SECOND Example ---------------
ele_num = 5/0.1;
material_ID = 2*ones(1,ele_num);              % Enter IDs using labels in
'material.m' (material library)
element_type = 3*ones(1,ele_num);               % Enter element types (1:
BAR, 2: TIMOSHENKO, 3: BERNOULLI, 21: FEM-TIMO)
element_prop_ID = 4*ones(1,ele_num);          % Enter prop_ID labels from
sec_prop.m (mechanical props library)
```

In the material.m file, a material with label 2 is defined as follows:

```
        case 2
             MAT(i,1) = 20000000; % Elastic modulus
             MAT(i,2) = 0.29      % Poisson's ratio
             MAT(i,3) = 7.86e-5;  % gram/cm3 divide by 100 to convert to N/cm2
```

In addition, section label 4 which is used in this example is defined in the sec_prop.m library. I_ele, area_ele and areaS_ele are respectively the moment of inertia, gross area and shear area of the section. As the operator proceeds on the Main_file.m script, there is the second INPUT file that has to be filled, which is conn.m. This .m file gets inputs of element connectivity, nodal constraints (supports) and also nodal displacement. For the illustrative example, this file has filled up as follows:

```
% ------------------- SECOND EXAMPLE ----------------------
num_ele = 5/0.1;          % number of elements
j = 1 : num_ele;
conn = [j'   ones(num_ele,6)];
supp = [1   1   1   0;  [j(2:end)' zeros(num_ele-1,3)]; num_ele+1 0 1 0];
nodal_disp = [j' zeros(num_ele,3); num_ele+1 0 0 0];
```

The conn matrix contains the connectivity information for each elements in a row wise manner. supp matrix defines nodal external constraints. For instance, [1 1 1 0] which is the first row, tells that node 1 is free to rotate, but it is constrained to move horizontally and vertically (x and y). The nodal_disp defines nodal applied displacements if there is any. In this example there is no externally imposed displacements, therefore the matrix consists of all zeros.

The last INPUT file that has to be completed by the operator is ele_nodes.m. In this file, the operator defines elements positions and directions, by defining starting and ending nodes for each element. For the example here, the following script works:

```
% ----------------- SECOND Example ------------------------
ele_num = 5/0.1;
tag_nd = 1:length(x);
st_node = [1:ele_num];        % starting nodes for elements
end_node = [2:ele_num + 1];   % ending nodes for elements
DL = zeros(ele_num,3);        % Distributed Load for elements - row per
element (parallel, transverse, moment)
```

The st_node and end_node vectors include all starting points and ending points for elements in the model respectively. For example, the third element is placed between node 3 and 4.

The DL matrix contains all distributed loads over the elements. This matrix has the same number of rows as the elements in a model. The three values at each node indicate distributed axial, transverse and moment loads on each element.

Up to now, we have entered all parameters needed for the static analysis. However, in case of a modal or dynamic analysis, some general and specific parameters should be entered in the main file. These parameters that are parts of the main file, are shown in the box below:

```
% General Analysis Inputs
anlz_typ = 3;                    % type of analysis which is going to be done
(1 for static, 2 for modal, 3 for time history)
n_mode = 4;                      % number of modes needed to be comupted or
considered for modal and time history analysis
diag_mtd = 'cons-DIR';             % mass matrix diagonalization method
('cons-DIR', 'cons-SF', 'HRZ', 'row summing' or 'ad hoc')
dmp_typ = 'Rayleigh';            % damping type ('None', 'Rayleigh' or
'Modal')
```

First of all, as the first input in the main file (line 12), the type of analysis should be given. In the next line, if modal analysis is considered, the number of modes of interest has to be entered. Obviously, other types of analyses are invariant to this parameter.

The next parameter, diag_mtd, asks for the preferred mass matrix diagonalization method. Several options are available for the operator.

- 'cons-DIR' if consistent matrix from direct mass matrix equations is needed.
- 'cons-SF' if consistent matrix from shape function is needed. (works for Bernoulli elements)
- 'HRZ' if HRZ method is selected to convert consistent matrix to a diagonal mass matrix.
- 'row summing' if this technique is chosen for consistent to diagonal mass matrix conversion.
- 'ad hoc' if simple mass lumping is of interest.

The last parameter in this section is dmp_typ, which selects a method to generate the damping matrix of the problem. Three types are available here:

- 'Rayleigh': if Rayleigh damping is entered, the operator is needed to give range of desired frequencies in two and three lines below (f1 and f2). The program automatically find Rayleigh coefficients for the desired range.
- 'Modal': if this type of damping matrix is chosen, the modal damping ratio is also needed to be entered by the operator. This input also can be found a few lines below. As a default, 5% damping is introduced.
- 'None': this option can be selected if an undamped structure is needed to be modeled.

In addition, there is flexibility to enter Newmark method's parameters manually, or select one of the more common sets of coefficient, e.g. constant acceleration or linear acceleration. In line 113 of the main code, there is a parameter called acc_asump. Operator can change this parameter to 1, 2 or 3, which respectively adjust Newmark parameters assuming constant acceleration, linear acceleration or manually given parameters. If 3 is entered, the operator can give their own parameters at lines 126 and 127. The program can automatically check the length of time step with respect to the given parameters and pops up an error if time step is not short enough to solve instability.

As the last input, the load scale factor time history should be introduced to the program. It is necessary for time history analysis, because when solving Newmark equations, system load is calculated as the static loads given in the nodal_coordinate.m file times the scale factor at each time step. To address this need, there is a part in the main file (starts at line 32) that assign different loading time series for each of the validating examples. For instance, for example two, the following code defines anything needed:

```
% Exmaple two - Ramp loading
del_t = 0.0001;          % defines time steps
t = 0:del_t:0.1;         % generate time vector
ld_interval = 0.01;      % impact duration (specified for this problem)
ld_mltp = [0.0:del_t/ld_interval:1 zeros(1,length(t)- ...
length(0.0:del_t/ld_interval:1))];        % Load TH definition
```

Please note that the length of variable ld_mltp should be the same as the time vector. There is a control for this important condition in the checks section of the code.

At this point, the program is completely ready to start. When the operator runs the code, based on the type of analysis, some prompts may come up and ask for parameters. For instance, for modal analysis, a prompt asks the operator to enter a desired mode shape to plot. After entering, it asks for a scale factor to enlarge mode shapes.

In case of time history analysis, program asks for a node id and corresponding DOF (1,2 or 3) to plot response time history of that node in that specific direction.