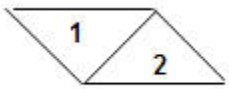
Structures and Handles - Creating a structure for two elements

Author: Rajesh Bhaskaran, Cornell University

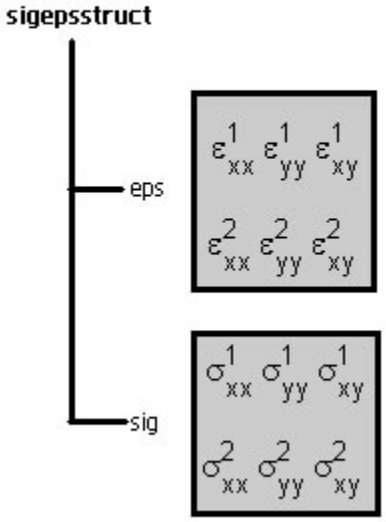
Creating a Structure for Two Elements

Extend to two elements

Let's now consider the case where we would like to create a structure to store the strain and stress values for *two* triangular elements.



The structure from Step 1 can be modified as per below to accommodate the second element.



We'll create the above structure by modifying the code from Step 1. Recall that the statements for creating the structure were

```
eps_val = [2 7 4]; %Nonsense values
R = 2*eye(3); %Dummy R
sig_val = (R*eps_val')'; %Derived nonsense
sigepsstruct = struct('eps', eps_val, ...
'sig', sig_val);
```

We have to create a second row for the *eps* and *sig* fields, corresponding to values for the second element. We can do this by looping over the elements, adding one row at a time.

```
numels = 2;%Total number of elements
for iele = 1:numels

     stuff

{{}}
end
sigepsstruct = struct();
```

We will keep the *R* matrix calculation out of the loop (since it doesn't need to be recalculated each time anew). This leaves us with the `eps_val`, `sig_val` and `sigepsstruct` statements inside the loop. In a real calculation, the strain values for each element , or equivalently, the rows of the *eps* field, are going to be different. To make the rows of the *eps*field different, we'll blithely multiply each row by the loop counter. This is again done to explain programming concepts; it makes no sense from a solid mechanics sense.

```
numels = 2;%Total number of elements
R = 2*eye(3); %Dummy R
for iele = 1:numels
   eps_val = iele*[2 7 4]; %Nonsense values
   sig_val = (R*eps_val')'; %Derived nonsense
end
```

Another thing: `eps_val` and `sig_val` above are row vectors. But the *eps*field is a matrix. We'll create this matrix as follows: after each row of the matrix is calculated, we will "push" it into the appropriate row of the matrix. Let's call this matrix *geps_val*, the *g* prefix standing for global. The following assignment will push the current *eps_val* vector into the second row of the matrix.

```
geps_val(2, : ) = eps_val;
```

Chew over this and make sure you understand it. If need be, look up the help on the ":" operator. The matrix for the *sig* field can be created in a similar fashion. When using the *struct* function to create *sigepsstruct*, we now need to use the *geps_val* and *gsig_val* to assign values to the fields. The following code brings this all together.

```
 1   %Testing the use of structure for two elements
 2 - clear all; %Clear all variables from workspace
 3 - numels = 2;
 4 - R = 2*eye(3); %Dummy R
 5 - for iele = 1:numels
 6 -     eps_val = iele*[2 7 4]; %Nonsense values
 7 -     geps_val(iele,:) = eps_val;
 8               %Creating one row of eps matrix
 9 -     sig_val = (R*eps_val')'; %Derived nonsense
10 -     gsig_val(iele,:) = sig_val;
11               %Creating one row of sig matrix
12 - end
13 - sigepsstruct = struct('eps', geps_val, ...
14       'sig', gsig_val);
```

Run your code and check the values in the Workspace. Once your code is working correctly, test extracting data from the structure.Note that you can look up the names of each field within the structure using the fieldnames() command.

```
%Extract data from structure
myeps = sigepsstruct.eps
sigxy2 = sigepsstruct.sig(2,3)
fieldnames(sigepsstruct)
```

Go to Step 3: Structure of handles

Go to all MATLAB Learning Modules