17.07.2015

# Small oFEM user guide v0.3

This guide is subject to constant change, since the work on oFEM is still ongoing, please consider this when working with it!

- create an oFEM mesh object:
  `mesh=ofem.mesh;`
- The mesh class can now be used to load meshes from `*.inp` files by invoking the method `load_from_inp:`
  `mesh.load_from_inp(inp_file_name);`
- After creation of the mesh object the methods and some basic descriptions of the mesh object can be found by typing `mesh` into the matlab prompt (if your variable is called mesh, as in the example) and then click on the highlighted link. The help browser will open and provide you with some information
- The function space discretization is controlled by the finite element class:
  `elem=ofem.finiteelement.P1;`
- You can also use `finiteelemnt.P2` or several other attributes (only P1 and P2 will work for now)
- An equation object is invoked like this:
  `eq=ofem.laplace(mesh,elem);`
- At the current stage the classes `laplace` and `elastic` exist
- Before you start the assemble process you will have to specify some options like this:
  `opt.S` =1, for assembling the stiffness matrix
  `opt.M` =1, for assembling the mass matrix
  `opt.b` =1, for assembling a right hand side
  `opt.load` = MatlabFunctionHandle (for example @MyRHSFun), the function computes the right hand side for all point and must be specified by the user
  `opt.N` =1, for assembling Neumann boundary conditions
  `opt.neumann` = MatlabFunctionHandle (for example @MyNeumannFun), the function computes the Neumann boundary function for all points of the boundary and must be specified by the user
  `opt.neumannidx` =[IndexVector] (for example [3 4]), the index vector point into the second dimension of mesh.bd and yields to the part of the boundary that is to be equipped with Neumann conditions.
  When no options are specified `opt.S` and `opt.M` are treated as they are set to 1 (i.e, the stiffness and mass matrix will be computed)

- You can now assemble the desired objects by simply typing:
  `[asm,info,aux]=eq.assemble(opt);`
  asm stores the the righthand side (with Neumann boundary conditions included if specified), the stiffness matrices per element part as well as the mass matrices per element
- To add the matrices simply perform something like:
  `S=asm.S{1}+asm.S{2}+asm.S{3};`
- The mesh class comes with a method to get the Dirichlet boundary conditions right, use it like:
  `diriNodes= mesh.dirichlet(Index Vector);`
  here `Index` is a vector pointing into the second dimension of `mesh.bd` yielding that the respective part of the the boundary is equipped with Dirichlet conditions. `diriNodes` is a cell!
- You can plot your solution (rudimental plot, no fancy stuff) by:
  `eq.plot(u)`
- To make it possible to get fancy 3D plots of your solutions we included an export routine to export your data to the UCD format. The data can afterwards be visualized with ParaView®. The export is part of the mesh class. You can invoke it like this:
  `mesh.export_UCD(folderName, fileName, {VarName1, Var1, VarUnit1}, {VarName2, Var2, VarUnit2},…);`
  Here `folderName` and `fileName` are strings. The data cell (yes, it is a cell variable!) is structured as follows `VarName` is a string, `Var` is the actual data (e.g., the solution u or it's gradient components of [ux, uy, uz]) and `VarUnit` is a string again that gives the unit of the output. Don't leave any field empty!
- The class laplace has a method to compute the gradient of the solution in the DOFs. For now this works for 2D and 3D P1 elements. It can be called by:
  `gradu=eq.gradu(u);`
  This function is still under development so please take care when using it!
- We included and updated the elastic class that can solve linear elastic problems. Most things work like with the laplace class, except that you need to be aware of the vectorial nature of your solution (you calculate a deformation field that has dimension of your space). In the following the methods of this class are discussed
- The equation object is invoked by:
  `eq=ofem.elastic(mesh,elem);`
- Material is essential in linear elasticity, therefore material parameters have to be given. We use Lame's constants mu and lambda, depending on you choice in 2D (plane stress or plane strain) you have to compute them before passing the to oFEM. You have to give a vector of either material constants,

one for every element part you defined. Parameters are passed to oFEM by calling:
`eq.setmaterial(lambda, mu);`

- Options and assembling are the same as before
- When considering Dirichlet BCs be aware of the vectorial nature. We order as follows `(u1x; u1y;u1z; u2x; u2y;u2z;…)`. So you have to adjust the node numbers you obtained by `mesh.dirichlet()`.
- After solving you can compute Green's Strain tensor, Couchy's Stress tensor and the deformation gradient tensor as follows:
  `F=eq.defGrad(u);` returns the deformation gradient tensor (F=I+du/dx)
  `[E,e,s] = eq.StrainStress(u);` gives the full strain tensor E (E=0.5*(F^T F -I)), the reduced strain tensor with engineering strain (e=[E_{11} E_{22} E_{33} 2*E_{23} 2*E_{13} 2*E_{12}] in 3D and e=[ E_{11} E_{22} 2*E_{12}] in 2D accordingly) `and the independent components of the stress tensor (s=Ce)`
- To plot the displacement field `u`, stress `s` and strain `E` in 3D type for example:
  `mesh.export_UCD(fullfile(pwd,'solution'), 'deflector', {'U',u, 'm'},{'E', E(:,[1,5,9,6,3,2]), '[]'}, {'S', s{1}', 'GPa'});`