Application of FEM substructuring and superelements technique in stress analysis of thin-walled structures

Barać, Ivan

THE FACULTY OF SCIENCE

FACULTY OF ELECTRICAL ENGINEERING, MECHANICAL ENGINEERING AND NAVAL ARCHITECTURE

MASTER THESIS

APPLICATION OF FEM SUBSTRUCTURING AND SUPERELEMENTS TECHNIQUE IN STRESS ANALYSIS OF THIN – WALLED STRUCTURES

Mentor: Prof. dr.sc. Željan Lozina

Student: Ivan Barać

Split, September 2016

I want to say big Thank you to my mentor, Prof. dr. sc. Željan Lozina, for guiding me through the process of working on this thesis, and to my family and my girlfriend for support.

Ivan Barać

Table of Contents

Abstract

Substructuring and superelements technique is today one of the most important procedures in large scale stress analysis industry design processes. In this thesis the implementation of substructuring and superelements technique in finite element method is presented, with its application in various types of one, two and three – dimensional problems in stress analysis of thin-walled structures. As a ground basis needed to develop substructuring procedure with FEM, first the programming procedures of the classical FEM in Matlab programming language are presented in the Appendix, alongside with the application of classical FEM in two – dimensional plane stress analysis problems. For the plane stress analysis problems solved in Appendix the programming codes were developed using repository of professor dr. sc. Jack Chessa (University of Texas) called FEMLAB [1], as a basis. Analysis performed in Appendix is serving as an essential foreknowledge needed for the substructuring and superelements approach to be implemented and applied.

As for the main part of the thesis, first an overview of FEM implementation in structural analysis is presented. Substructuring and superelements technique is introduced. Physical and mathematical concept of the technique is developed, with the practical aspects emphasized and singled out. Global – local analysis (two levels) is distinguished from multilevel substructuring. The procedure of substructuring is described in detail, with the given data flow diagram in generic multilevel substructuring made of any desired number of levels. Programming concept of the technique is thoroughly described, and the interface between an external mesh generator (Gmsh) and Matlab programming language is developed. The procedure of substructuring was performed with the "top – down" approach, whilst the programming procedure was implemented in both directions, "top – down" and "bottom – up". The substructuring procedure and its programming are then connected together in application to a set of one, two and three – dimensional problems in stress analysis of thin – walled structures. One – dimensional bar element is solved using this technique by hand, to illustrate the need of programming the procedure for more complex problems. The two – dimensional problems were solved using an automated approach programmed in Matlab, and a thin plate with a non-symmetrical hole subjected to an in-plane bending load is analysed by diving it into a total of ten level two

substructures. Programming the technique in 3D is then illustrated through an example of a simple hexahedron case made up of 6 thin plates. Eventually, built on the knowledge collected through working on this thesis, a stress analysis of a 3D ship cover with one bulkhead is performed, according to the drawing made in Catia. Practical aspects of the technique are directly pointed out and emphasized through the analysis of processing and its results, whose visualization, alongside with Matlab, is presented in Paraview Visualization Toolkit. The thesis finishes by describing a concept idea of how to manipulate with large data structures, such as the one obtained and collected through performed analysis of a 3D ship cover.

Introduction

Substructures are sets of elements which have a well defined structural function, most commonly obtained by dividing the complete structure into its functional components. Some of the examples may include: funnel, stern and bow of the ship; fuselage, wings and vertical and horizontal stabiliser of an airplane; deck, cables and towers of a suspension bridge, and on a smaller scale for example - steel plates with bulkheads that make up ship covers on tweendecks of large heavy lift vessels. Substructures are usually defined "top – down" as parts of a complete structure, and they are not to be mixed with the so-called "macroelements", which basically resemble structural components, but are fabricated with simpler elements (called primitive elements or mesh units) in a "bottom – up" direction. The distinction between substructures and macroelements is not clear-cut, so the term *superelement* is often used in a collective sense to embrace element groupings that range from macroelements to substructures. Both macroelements and substructures are treated exactly the same way as regards matrix processing. The basic rule is that associated with the condensation of chosen degrees of freedom, which shall be described in more detail in Chapter 2. The reasons for introducing such technique are several. The disconnection-assembly concept occurs naturally when inspecting many artificial and natural systems. For example, it is easy to visualize a ship, an airplane, a building, an engine, bridge, or even human skeleton by being broken down into their simpler sub-components. Also, for example, multistage rockets naturally decompose into substructures, as depicted in Figure 1.

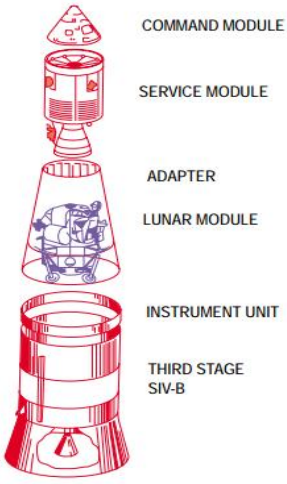

Figure 1: Natural decomposition of multistage Apollo lunar rocket

So, the motivating factors that drive the development of such technique are different and many. Among them, some are to simplify the pre-processing tasks, then to facilitate the division of labour and to take advantage of repetition, since often structures are built of several identical or nearly identical units, so mirroring and symmetry can be involved. Another great reason is to save computational time, since the whole process of condensation significantly reduces the number of degrees of freedom which are entering the system of equations needed to be solved for the global displacements, as shall be more thoroughly explained in the chapters to come.

Today, as the technological development is in a constant rise, a new motivating factor for further development of this technique has arisen and it involves parallel processing. Applied mathematicians working on solution procedures for parallel computation have developed the concept of sub-domains, which is practically an extension and generalization of the substructuring technique entirely motivated by computational considerations, and this is an area with a great potential for development in the future.

Maybe the most significant underlying theme of substructuring and superelements technique can be described as: “divide and conquer”.

1. An overview of FEM implementation in structural analysis

Finite element method has been originally developed on the grounds of engineering methods in 1950s in order to obtain numerical solutions of complex problems in structural mechanics and structural analysis in general. In its simplest form, structural analysis can be described as determination of the effects of loads on physical structures and their components. Structures which are subject to this type of analysis include all that must withstand loads and some type of constraints (supports), such as buildings, bridges, vehicles, machinery, furniture, prostheses, biological tissues, etc. Structural analysis incorporates the fields of applied mechanics, materials science and applied mathematics to compute structure's deformations, internal forces, stresses, support reactions, accelerations and stability. The results of the analysis are used to verify a structure's fitness for use, often saving real physical tests. Structural analysis is thus a key part of the engineering design of structures. There are a dozen of approaches and methods used for modelling problems in structural analysis, such as analytical methods, strength of materials (classical) methods, elasticity methods, and methods using numerical approximation. Each of these methods has their own noteworthy limitations. Strength of materials (or mechanics of materials) method is limited to very simple structural elements under relatively simple loading conditions. Then, the theory of elasticity allows the solution of structural elements of general geometry under general loading conditions in principle, but the analytical solution, however, is limited to relatively simple cases. Elasticity methods are available for an elastic solid of any shape, and individual members such as beams, columns, shafts, plates and shells may be modelled, but the solutions are derived from the equations of linear elasticity and these equations are a system of 15 partial differential equations. So, due to the nature of the mathematics involved which can be very demanding, analytical solutions may only be produced for relatively simple geometries. For complex geometries, a numerical solution method such as the finite element method is a pure necessity.

As already stated, different approaches to FEM emerged over time so various types of finite element method developed, especially in those areas where classical FEM had problems in describing certain phenomena or even defining variational formulations (among many, for

discontinuous functions for example). So, in addition to classical FEM, some of the most represented formulations of finite element method in structural analysis today include Applied element method (AEM), Generalized finite element method (GFEM), Extended finite element method (XFEM), hp-FEM, Mixed finite element method, etc. Some of these methods are based upon the combination of mesh refinements for the purposes of an exponentially fast convergence (hp-FEM), then on a different connection between the elements via nonlinear springs (AEM), furthermore by enriching the solution space with solutions to differential equations with discontinuous functions (XFEM), etc. Many of these formulations of finite element method are today incorporated in commercial finite element software such as Abaqus, Ansys, etc. As for the substructuring and superelements technique, it should not be described as a single type or sub-type of FEM. The whole concept of such technique has its advantage in the fact that for the specific problem given, practically any of these FEM formulations can be incorporated inside substructuring technique. The whole idea of substructuring lies in the disassembly concept for parts of a complete complex structure, so the matter of choice upon the type of FEM procedure that shall be incorporated on the lowest level substructure stays on the analyst. It is important to emphasize here that because of the reasons discussed, substructuring technique is widely used in industry and especially in engineering design of large structures. Now the technique shall be thoroughly described and applied, in chapters to come.

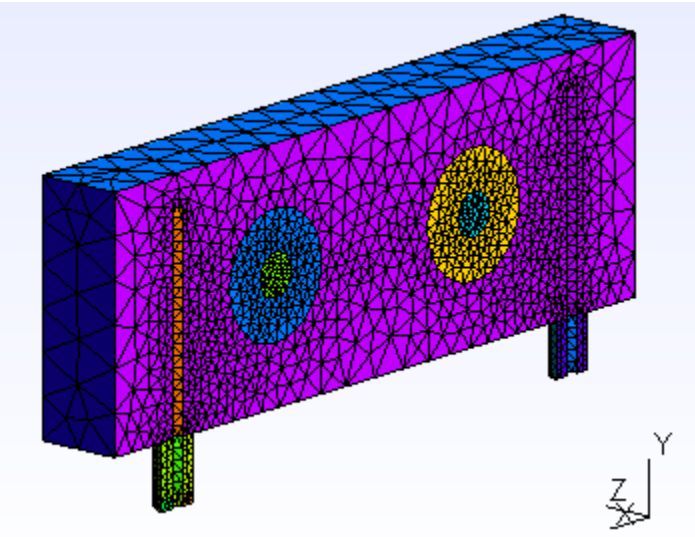

Figure 1.1: Adapted mesh of triangular finite elements created in Gmsh software

2. Substructuring and superelements technique

Superelement is a collective of finite elements which, upon its assembly and mainly for computational purposes, can be considered as an individual element. The motivations for such formulation are most often powered by modelling or processing needs, which shall be discussed in more detail in Chapter 2.1.

It is important to note that any random grouping of elements does not necessarily build up a superelement. To be acknowledged as such, this collection of elements should most often meet certain conditions. Informally we can say that it should, on top levels, form a structural component on its own or at least have some boundary conditions imposed inside, to be able to justify the disassembly in a certain way that surpasses the exact structural components decomposition. Also, in the pre-processing phase geometrical constraints can show up which may not be directly correlated with structural intersections, but for reasons that will be discussed later they can also become a deciding factor in a disassembly type that will not always follow the break up into structural components in an exact way and on all levels of substructuring.

As already stated in introduction, superelements may show up in two overlapping contexts: “top – down” or “bottom – up”. In a top – down context superelements are considered as being large pieces of a complete structure and then they are usually called substructures, whereas the complete structure is called a superstructure. This convention shall also be used in this thesis. In a bottom – up context superelements are built from simpler elements and then they are called macroelements, and the simplest elements from which they are built are called mesh units. As we can see, these contexts often overlap in both cases, but the general idea is that, depending on the direction (assembly or disassembly), macroelements and substructures are distinct features linked through the term superelement.

Regarding matrix processing and mathematical and physical aspects of the technique, the most profound part would be to understand the concept of condensation of chosen (most often internal in practice) degrees of freedom for such collections of elements as described above. Here the emphasis shall be put on superstructures and their appropriate substructures, as we shall largely implement top – down approach to problem solutions dealt with in this thesis.

2.1. Practical aspects of substructuring

Substructuring was invented in the aerospace industry in the early 1960s to perform a first level breakdown of large and complex systems such as a complete airplane, as shown in Figure 2.1. This procedure can continue hierarchically down through further levels, as shown in Figure 2.2. Also, in practically every aspect of engineering design where large and often symmetrical constructions are involved, such as large ships, space vehicles etc., this concept imposes itself almost naturally.

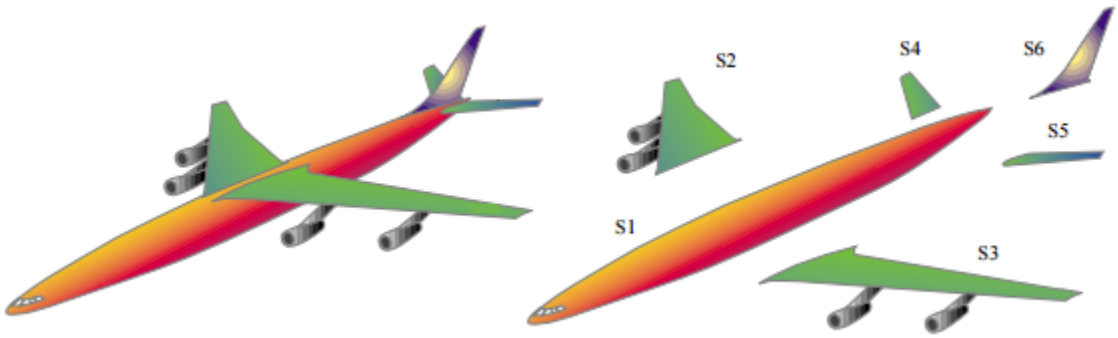

Figure 2.1: Complete airplane (superstructure) broken down into six level two substructures

Following a top – down approach in Figure 2.1, if we consider a complete airplane as a superstructure on level one (if we are dealing with multilevel substructuring there can also be superstructures on lower levels), then the complete first level breakdown would be into substructures S1 to S6 respectively, which are now substructures on level two. The decomposition process can theoretically continue to an individual element level, as depicted in Figure 2.2, where from the first breakdown of a complete airplane we have a total of three levels (even without an individual element acknowledged as a separate level).

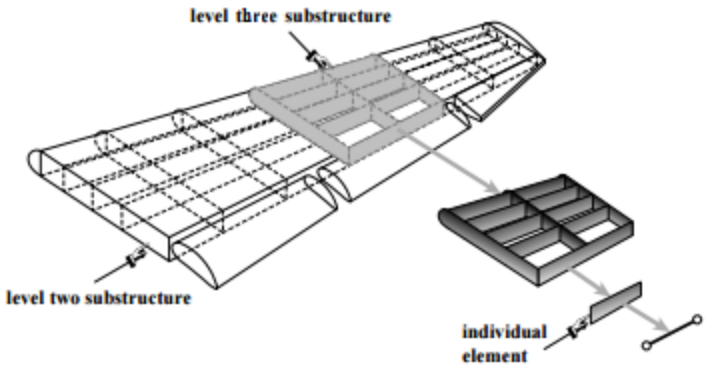

Figure 2.2: Substructuring down to an individual element of wing structure

Among the motivating factors and the practical aspects for the first substructuring technique development, most of them still hold today. The most notable can be singled out as:

1) Simplification of pre-processing tasks – when creating a mesh of the superstructure on the highest level (i.e. the complete structure), this mesh can be rather coarse, compared to the classical FEM approach. The reason is that we only need to retain nodes on the interface of different substructures, and the ones where boundary conditions, fixed displacements or applied load are imposed. As shall be explained in the next chapter, because of the applied process of condensation these nodes on the superstructure (so called master nodes) "preserve" information from all the nodes from lower levels of substructuring, where we actually generate finer meshes. As we shall show later in solved examples, this whole process dramatically reduces the number of nodes and degrees of freedom (dofs) that are entering the system of equations which needs to be solved for the global displacements on the superstructure. From this point onwards, practically whole stress analysis can be performed as in the classical FEM approach, with the addition of appropriate data transfer from one level to another.
2) Facilitation of labour division – separate design groups with specialized knowledge and experience can work on substructures with different functions. Example given, a shipbuilding company can set up a funnel group, propeller and rudder group, tweendeck group, crane group, etc. An aircraft company may set up a wing group, turbine engine group, a fuselage group, horizontal and vertical stabilizer group, cockpit group, etc. So, these groups are protected from "hurry up and wait" constraints, and more important – they can keep on working on refinements, improvements and verification of the

experimental model independent of each other, as long as the interface information (e.g. the wing – fuselage interface in an aircraft example) stays practically unchanged.

3) Taking advantage of repetition – includes involvement of mirroring and symmetry, since airplanes, ships, space vehicles and a large number of other structures are often built of several identical or almost identical components. For example, the wing substructures S2 and S3 from Figure 2.1 are mirror images on reflection around the fuselage mid-plane, and the horizontal stabilizers S4 and S5 are too. So, recognizing patterns and repetitions reduces model preparation time significantly (even if the loading is not symmetric around the stated mid-plane for example).
4) Overcoming computer limitations (past) and reducing computational time (present) – when the substructuring technique was first invented, the computers of the 1960s operated under serious memory limitations. Due to these limitations, it was quite demanding and difficult to fit a complex structure such as an airplane as one entity. So, this technique allowed for the complete analysis to be carried out in stages with the use of tapes and disks as auxiliary storage devices.

   This motivating factor from the past today moved to a different area and it involves a great amount of reduced computational time when using this technique compared to classical FEM, which shall be directly pointed out and verified in the problems solved in chapters to come. Also, with a constant development of this technique to the present day, another motivation rose up through the invention of *subdomains*, which is a concept developed in applied mathematics for parallel processing. Subdomains are sets of finite elements entirely motivated by computational considerations. These subdivisions of the finite element model are done more or less automatically through the program called *domain decomposer*. The concepts of subdomains and substructures overlap in many aspects, but the motivation is different so it's advisable to keep the two separate, although domain decomposers are largely used in dynamic substructuring.

There is another great advantage of using substructuring and superelements technique and it is a direct consequence of its application. It comprises much more efficient visualization in the pre- and post-processing phases, which will be shown in Chapter 4.1.2.

2.2. Physical and mathematical concept of substructuring and superelements technique

Here we shall present the basic physical and mathematical background needed to develop and apply substructuring and superelements technique in real life engineering problems in stress analysis. It is important to note that to properly handle this technique alongside with its modelling and programming aspects, one needs to be familiar with and have a solid background in the theory of classical FEM procedures, its programming and application. For the reasons stated, these procedures are thoroughly explained and presented in the Appendices C and E, so the reader is referred to these sections for more detailed explanations regarding classical FEM.

From the mathematical viewpoint, the superelement is said to be *rank-sufficient* if his only zero – energy modes are rigid body modes. What that basically means is that the superelement does not possess spurious kinematic mechanisms. Verification of the rank-sufficient condition guarantees that the static condensation procedure described below will work properly.

2.2.1. Static condensation

In general, degrees of freedom of a superelement most often can be classified into 2 categories: *internal* and *boundary*. Internal dofs are the ones that are not connected to the dofs of another superelement, and the nodes whose freedoms are internal are called *internal nodes*. Boundary dofs are the ones that are connected to at least one other superelement. They are usually located at *boundary nodes* placed on the periphery of the superelement, as shown in Figure 2.3.

Note that the finite element mesh from Figure 2.3 is shown as two – dimensional for illustrative purposes – for an actual aircraft it will be three – dimensional. Boundary freedoms are those associated to the boundary nodes, labelled *b* (shown in red), and they are connected to the fuselage substructure.

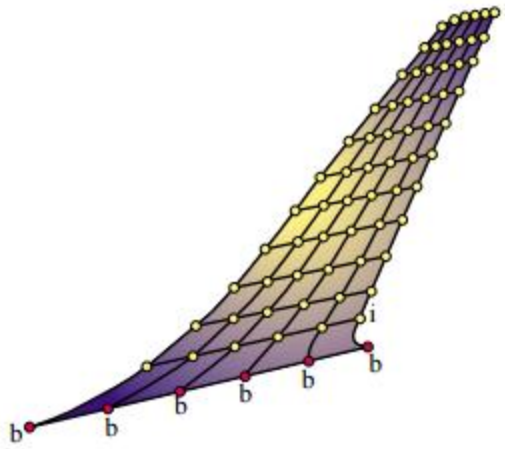

Figure 2.3: Vertical stabilizer substructure S6 from Figure 2.1 – classification of superelement freedoms into boundary and internal

So, on the first level breakdown, or better to say on the substructure's S6 own level (level two), where we generate finer mesh, the objective is to get rid of all displacement dofs associated with *internal freedoms*, i.e. the ones that are not on the interface of another substructure (they do not interact). This elimination process is called *static condensation*, or simply, *condensation*. Of course, if for some reason we want to keep some internal nodes from Figure 2.3 – e.g. we can have some force (natural) boundary conditions imposed on the edge of the stabilizer, this can also be done, the only important thing is to sort all these nodes that we want to keep in the stiffness matrix together. Detailed procedure of how can this be done is explained in Chapters 3 and 4, respectively. Static condensation will now be presented in terms of explicit matrix operations, as shown in the next few paragraphs.

To carry out the condensation process, the general form of the assembled stiffness equation:

$$[K]\cdot[d]=[f], \tag{2.1}$$

needs to be partitioned as follows:

$$\begin{bmatrix} \boldsymbol{K}_{bb} & \boldsymbol{K}_{bi} \\ \boldsymbol{K}_{ib} & \boldsymbol{K}_{ii} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{d}_b \\ \boldsymbol{d}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}_b \\ \boldsymbol{f}_i \end{bmatrix}, \tag{2.2}$$

where sub-vectors $\boldsymbol{d}_b$ and $\boldsymbol{d}_i$ collect boundary and internal degrees of freedom, respectively.

Now, the second matrix equation of the expression (2.2) is:

$$\boldsymbol{K}_{ib} \cdot \boldsymbol{d}_b + \boldsymbol{K}_{ii} \cdot \boldsymbol{d}_i = \boldsymbol{f}_i \tag{2.3}$$

If $\boldsymbol{K}_{ii}$ is non-singular, we can solve this equation for internal dofs:

$$\boldsymbol{d}_i = \boldsymbol{K}_{ii}^{-1} \cdot \left(\boldsymbol{f}_i - \boldsymbol{K}_{ib} \cdot \boldsymbol{d}_b \right) \tag{2.4}$$

Now, replacing $\boldsymbol{d}_i$ from (2.4) into the first matrix equation of (2.2), we obtain the *condensed stiffness equations*:

$$\tilde{\boldsymbol{K}}_{bb} \cdot \boldsymbol{d}_b = \tilde{\boldsymbol{f}}_b \,, \tag{2.5}$$

where

$$\begin{aligned} \tilde{\boldsymbol{K}}_{bb} &= \boldsymbol{K}_{bb} - \boldsymbol{K}_{bi} \cdot \boldsymbol{K}_{ii}^{-1} \cdot \boldsymbol{K}_{ib} \\ \tilde{\boldsymbol{f}}_b &= \boldsymbol{f}_b - \boldsymbol{K}_{bi} \cdot \boldsymbol{K}_{ii}^{-1} \cdot \boldsymbol{f}_i \end{aligned} \,. \tag{2.6}$$

In the equation (2.6), $\tilde{\boldsymbol{K}}_{bb}$ and $\tilde{\boldsymbol{f}}_b$ are called the *condensed* stiffness matrix and the *condensed* force vector, respectively, of the given substructure.

From this point onwards, the condensed superelement may be viewed, from the standpoint of further operations, as an individual element whose element stiffness matrix and nodal force vector are $\tilde{\boldsymbol{K}}_{bb}$ and $\tilde{\boldsymbol{f}}_b$, respectively. Often each superelement has its own "local" coordinate system. If that is the case, a transformation of (2.6) to an overall global coordinate system is necessary upon condensation. Also, in the case of multiple levels, the transformation is done with respect to the next level superelement coordinate system. This coordinate transformation procedure automates the processing of repeated portions.

Note that the feasibility of the condensation process (2.4) - (2.6) rests on the non-singularity of $\boldsymbol{K}_{ii}$. This matrix is non-singular if the superelement is rank-sufficient in the sense stated in Chapter 2.2., and if fixing the boundary freedoms precludes all rigid body motions. If

the former condition is verified but not the latter, the superelement is called *floating*. Processing floating superelements demands usage of much more advanced computational techniques, which exceeds the range of topic covered in this thesis.

After the boundary displacements are obtained by the solution process, the internal displacements can be recovered directly from (2.4). This process may be carried out either directly through a sequence of matrix operations, or equation by equation as a back substitution process.

Direct application of the static condensation procedure described above shall be presented in Chapter 4.1.1., where a one-dimensional bar element will be solved using this technique by hand, to illustrate the need of programming the procedure for more complex problems.

2.3. Global – local analysis and multilevel substructuring

When the procedure of static condensation described in the previous chapter is restricted to two stages (levels) and applied in the context of finite element analysis, it is called global – local analysis in the FEM literature. What that basically means is that we have just two levels in our substructuring procedure. On the first level (superstructure), we recognize the master nodes (these are usually interface nodes and the boundary nodes on separate substructures, but we can also have additional ones for any reason), we choose our substructures, and generate a mesh on the superstructure consisting only of these master nodes. This is a global analysis part. Then, we analyse each of these substructures locally and generate finer meshes on them separately of each other. We perform conventional FEM approach to obtain stiffness matrices of these local substructures, and after the stiffness matrices are obtained we apply the static condensation process and get condensed stiffness matrices of the substructures on the local level. Now we need to return these condensed stiffness matrices to the global model, to be able to build the global stiffness matrix of the superstructure. After the global stiffness matrix is generated, we implement boundary conditions and solve the system $\boldsymbol{K} \cdot \boldsymbol{d} = \boldsymbol{f}$ for global displacements of the master nodes. After the displacements of the master nodes are obtained, we return these displacements to the local analysis to obtain internal displacements from (2.4), (the ones that

were condensed out from the global analysis) and bearing in mind that there are no internal forces that are acting on internal nodes (because of the rank-sufficiency). Eventually, from the obtained total displacements we calculate strains and stresses of the substructures and perform stress analysis and visualize results. The key part lies in the algorithm developed in this thesis, through which variables are transferred from the global to the local analysis, and vice versa. This algorithm can be applied in multilevel substructuring made of any desired number of levels. Detailed procedure of how this is done, with an interface developed between an external mesh generator and Matlab programming language, is explained in Chapter 4.

When analysing complex engineering systems, it is often not sufficient to perform just a global – local analysis. Such systems are today more often modelled in a multilevel fashion following the divide and conquer approach. The multilevel substructuring and superelements technique is a practical realization of that approach.

Multilevel FEM substructuring was invented in the Norwegian offshore industry in the mid/late 1960s, among other things, to take advantage of repetition and to be able to analyse complex structures in more detail. This benefited to the whole industry because of the symmetry and mirroring involvement which resulted in an increased efficiency of industrial design processes. An example of multilevel substructuring is presented in Figure 2.4, where we have 3 levels of substructuring in a ship construction component. Note that the process can also go in the opposite direction, constructing the upper level substructure from the lower one using bottom – up (macroelements) approach. Either way, the emphasis is on the greater number of levels which allows for detailed analysis and efficient usage of symmetry.

Multilevel substructuring differs from the global – local analysis in terms of the intermediate steps existence. Since we have larger number of levels, the transfer of data and variables is done with respect to the next level superelement, which at times can be quite demanding in terms of manipulation with large data structures that build up during the process. Also, we have local superstructures on lower levels, for example after the first level breakdown – level two substructures are also superstructures for their appropriate substructures on level three, and so on. So, this whole process of transferring and transforming (if necessary) variables and data first to the next level local model and down to the lowest level, and then returning this data and new obtained variables back through multiple layers up to the highest level superstructure, collects a

fairly large amount of data on the way which needs to be sorted and manipulated for the post-processing phase.

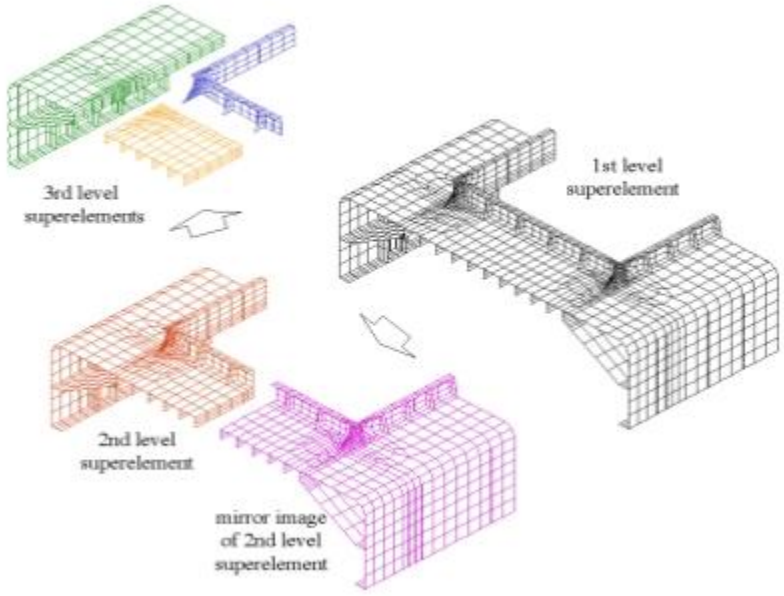

Figure 2.4: Multilevel substructuring of a ship construction component

Concept idea of how to manipulate with large data structures is described in Chapter 6. There we shall present procedures for sorting and managing such large data sets and give an overview in examples solved in this thesis.

3. The procedure of substructuring

This chapter shall give an overview of the procedure of substructuring performed in this thesis which can largely be implemented in a generic way also with the use of other softwares and programming languages. The only difference would then be in the choice of the mesh generator and in the programming syntax through which an interface between the two would be developed.

The procedure of substructuring for global – local analysis type has been described in previous chapter. Here the emphasis shall be put on a more general approach and it shall incorporate multilevel substructuring procedure with the given data flow of a three – level substructuring example. Also, some programming and substructuring design implications of multilevel substructuring cannot be perceived when dealing with global – local analysis only, which shall be discussed here and in the subsequent chapter to come.

Before going thoroughly into the steps of the multilevel procedure, some notes need to be acknowledged. For the lowest level substructures and first step stiffness analysis, an open source finite element package called FEMLAB [1] is used as a basis. Part of this publicly available code, written in Matlab, is adapted for the purposes of analysis performed in this thesis, and a large part of new codes are developed here as a certain extension to the package itself, regarding substructuring technique. So the package serves as a ground basis on which the whole procedure is built upon, and the extension developed is the center course of the thesis main task.

As an external mesh generator Gmsh software [2] is used, which is an automatic three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. Detailed description of Gmsh and its operating possibilities are presented in Appendix B, with the given examples of geometry and meshes for plane stress problems solved in Appendix C.

Now the steps in multilevel substructuring procedure shall be presented. Note that the programming aspects of these procedures shall not be pointed out here, as Chapter 4 deals with the programming concept thoroughly.

3.1. Steps in multilevel substructuring

The basic rules and steps that need to be followed from the very beginning of modelling engineering problems through multilevel substructuring are presented below.

1) **Recognize the level one superstructure, choose the disassembly way (i.e. substructures for the level two), and select the master nodes and mesh density on the first level accordingly.**

   This step is probably the most important one, and the final success of the analysis it withdraws often depends on the experience of the analyst. Chosen master nodes need to have some physical or geometrical significance. Master nodes most often concur with boundary nodes (dofs that are connected to another substructure – see Chapter 2.2.1) and then they are called interface nodes, but we can also have other nodes of our choice as master nodes for whatever reason (we can have some singularities in the construction, boundary conditions outside the interface, some geometrical constraints, holes etc). Holes are good example because most often they are not on the interface with another substructure but master nodes are usually set up on them because of the potential stress concentration which can be important later in the analysis.

2) **Create geometry of the level one superstructure and mesh that geometry following chosen parameters from step one. Read in the nodal coordinate matrix of the master nodes.**

   Mesh of the level one superstructure geometry should consist only of the chosen master nodes from step one. When dealing with thin-walled structures, this initial superstructure mesh is almost always created by generating two-node line elements on the defined geometry, as most often master nodes shall concur with the interface ones.

3) **For each substructure on its appropriate lower level, recognize its appropriate level superstructure and perform steps 1 and 2 for these superstructures. Go sequentially through the next lower level, repeating steps 1 and 2 until you reach the last level.**

This basically means that we are treating the level two superstructure (and every following lower level superstructure) the same as if it is a level one superstructure. We choose the disassembly way if there is any, and perform described steps. It is important to note here that the choice of the mesh density on lower levels is not arbitrary, and it must be the same as in the first level, as that secures that the coordinates of the master nodes from level one match in all levels. If that is not the case, we are dealing with the so called *multifreedom constraints*, which is a technique that is much more complex to deal with, especially regarding its numerical interpretation and coding. It involves the "penalty augmentation" and "Lagrange multiplier adjunction" [3] and demands using much more advanced computational techniques. This is a topic which largely exceeds the range covered in this thesis, so for further information refer to [3].

4) **On the last (lowest) level, create geometry of all the substructures and mesh these geometries using conventional but more advanced two-dimensional finite element types (e.g. CST, LST, quadrilateral finite elements, etc). Sort the dofs on master and internal. Obtain local stiffness matrices of all substructures by adapting and applying functions from the chosen open source finite element package.**

This is where the FEMLAB comes in. The importance of the scattering operator needs to be pointed out. See Appendix A.

5) **Obtain condensed stiffness matrices from Eq. 2.6 for all substructures on the lowest level and transfer master node ids of all the substructures from this last level to the upper level.**

To be able to obtain condensed stiffness matrices in a correct way, the local stiffness matrices need to be arranged in a way that the master node dofs are sorted together, separately from internal ones (the ones we want to condense). This can be done in an elegant way in Gmsh during geometry and mesh generation, and it shall be discussed in Chapter 4 in more detail.

Transferring master node ids to the upper level is done with the help of the Matlab function developed specifically for the purposes of this thesis. The name of the function is *id_local_global_transfer.m*, and it serves for the transformation of the nodal data from the local, lower substructure level to the next upper level, and eventually to the global level (level one). It can, and will, also be applied for the transformation from the upper levels to lower ones, which will be needed for the transfer of the global solution to the lowest local level and the appropriate stress analysis.

6) **From the condensed stiffness matrices of all the substructures on the lowest level build the stiffness matrix of the superstructure on its appropriate upper level. Transfer master node ids of the superstructure from this upper level to the next upper level. Repeat this step for as many superstructures you have on the upper level.**

   In real life engineering problems most often "upper levels" are built of several superstructures, as we shall see in Chapter 5.

7) **From all the stiffness matrices of superstructures on the upper level build the stiffness matrix of the next upper level from the current one. Transfer master node ids to the next upper level accordingly. Repeat this step until you reach the highest level – level one.**

   The last obtained stiffness matrix in this step is the global stiffness matrix of our level one superstructure. It has the size of number of dofs per node times the total number of nodes in the superstructure. It needs to be pointed out that this global stiffness matrix, although not very large compared to conventional FEM approach, has built-in information of all the stiffness of the local substructures on all levels, because of the process of condensation performed. It contains the data of even the internal dofs stiffness of local substructures on the lowest level. Because of the fairly modest number of dofs left, solving the system for the global displacements is now much more efficient.

8) **Implement boundary conditions (essential and natural) to the level one superstructure and solve the system $\boldsymbol{K \cdot d = f}$ to obtain the global displacements of the master nodes.**

Since in this thesis we are dealing with thin-walled structures, during implementation of the boundary conditions we need to take care that if we have master nodes which are not interface ones and are located inside the superstructure, there are no internal forces that are acting upon them so we need to constrain the degrees of freedom perpendicular to the plane in which the given nodes are positioned.

9) **Return global displacements of the master nodes back through levels to the lowest level and transform them to get local master node displacements for each substructure on the lowest level. Use these displacements to get internal displacements from Eq. 2.4.**

Transformation of global displacements of the master nodes down to the lowest level for each substructure is done with the help of previously mentioned function *id_local_global_transfer.m* and with the extensive use of the scattering operator.

10) **From the total obtained displacements on the lowest level (master + internal) calculate the strains and stresses for each substructure, and visualize results.**

This is the final step of the analysis. The analyst is interpreting the results and potentially adapting the boundary conditions if needed. Results are exported to a more advanced visualization toolkit, such as Paraview [4].

3.2. Data flow in multilevel substructuring with FEM

The whole procedure of multilevel substructuring with FEM, together with the described steps in the previous chapter, can be presented in the form of a data flow diagram of such process, presented in Figures 3.1*a*, *b*, *c* and *d*.

Note that the data flow diagram is presented for a generic multilevel substructuring type that can consist of any desired number of levels. Functions from FEMLAB used for this purpose, alongside with new functions developed specifically for this thesis, are given in the Appedix F. The diagram is split into phases for more convenient visualization, but also because of the logical separate stages that occur during the solution process.

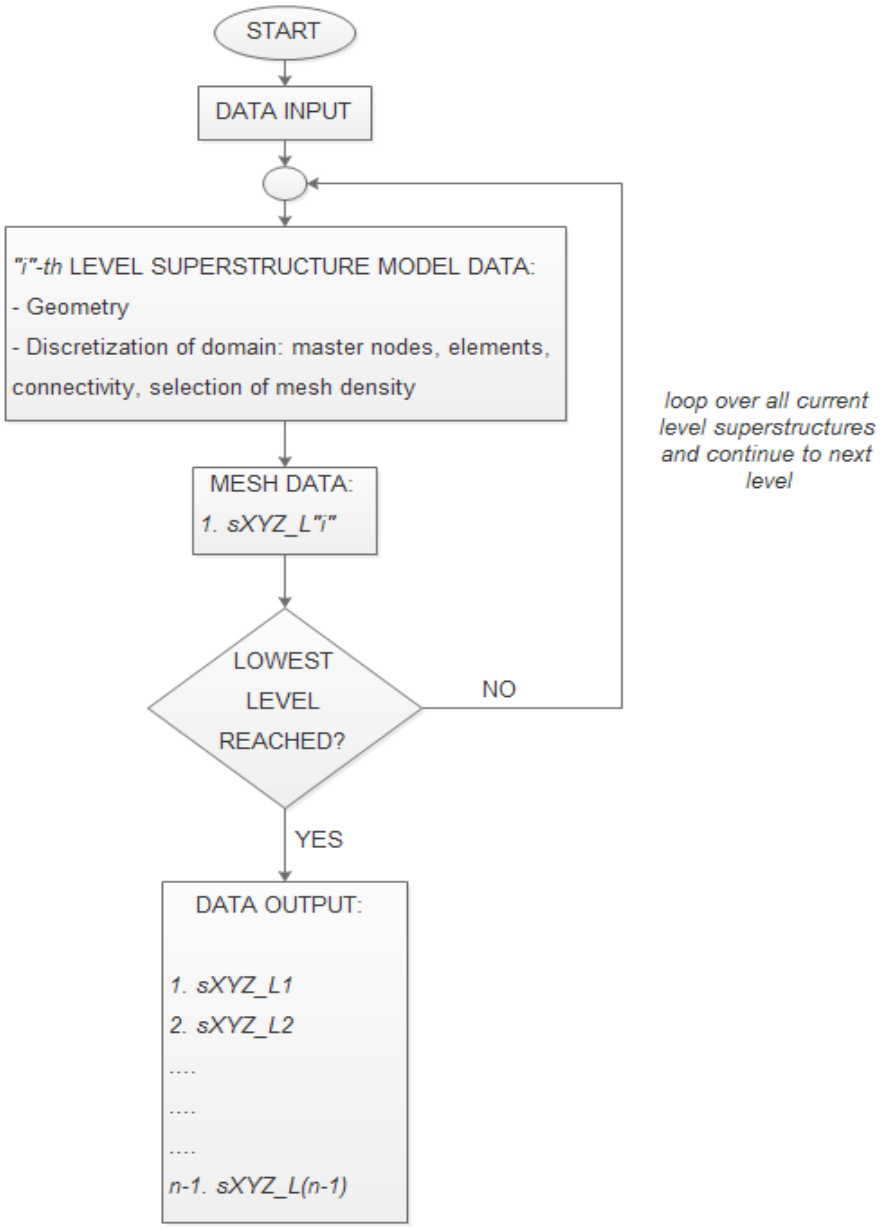

Figure 3.1a: Phase one in multilevel substructuring with FEM

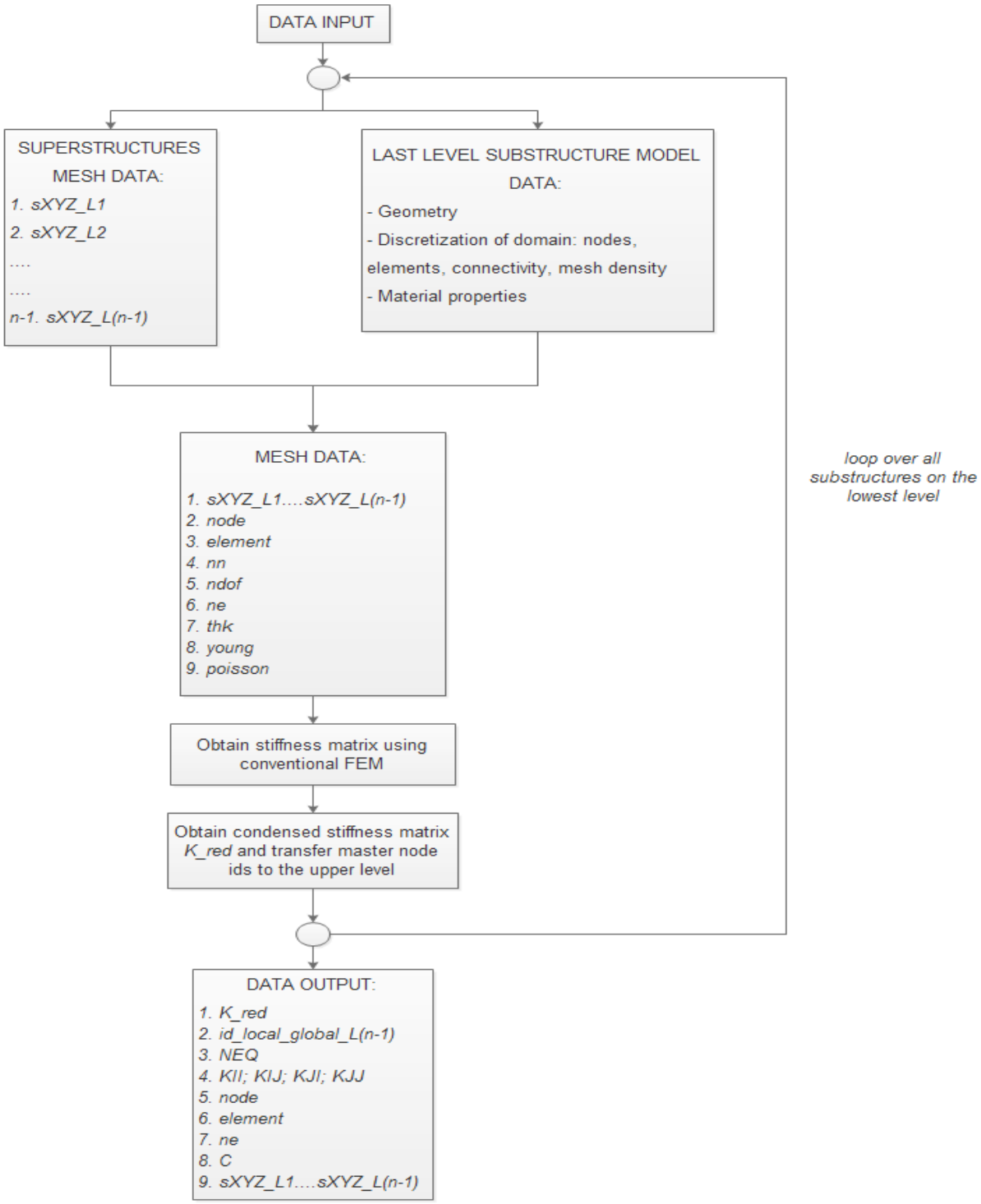

Figure 3.1b: Phase two in multilevel substructuring with FEM

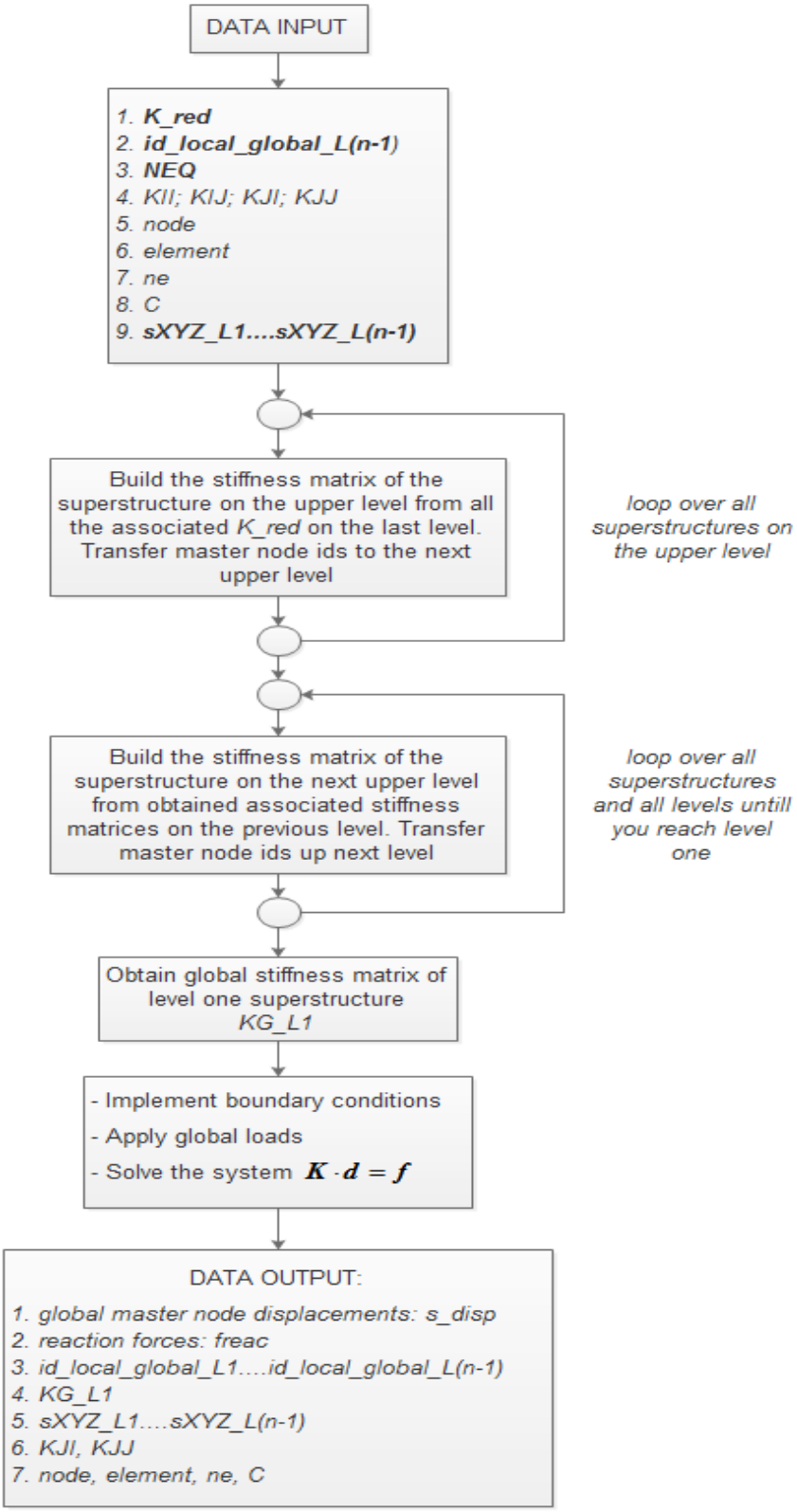

Figure 3.1c: Phase three in multilevel substructuring with FEM

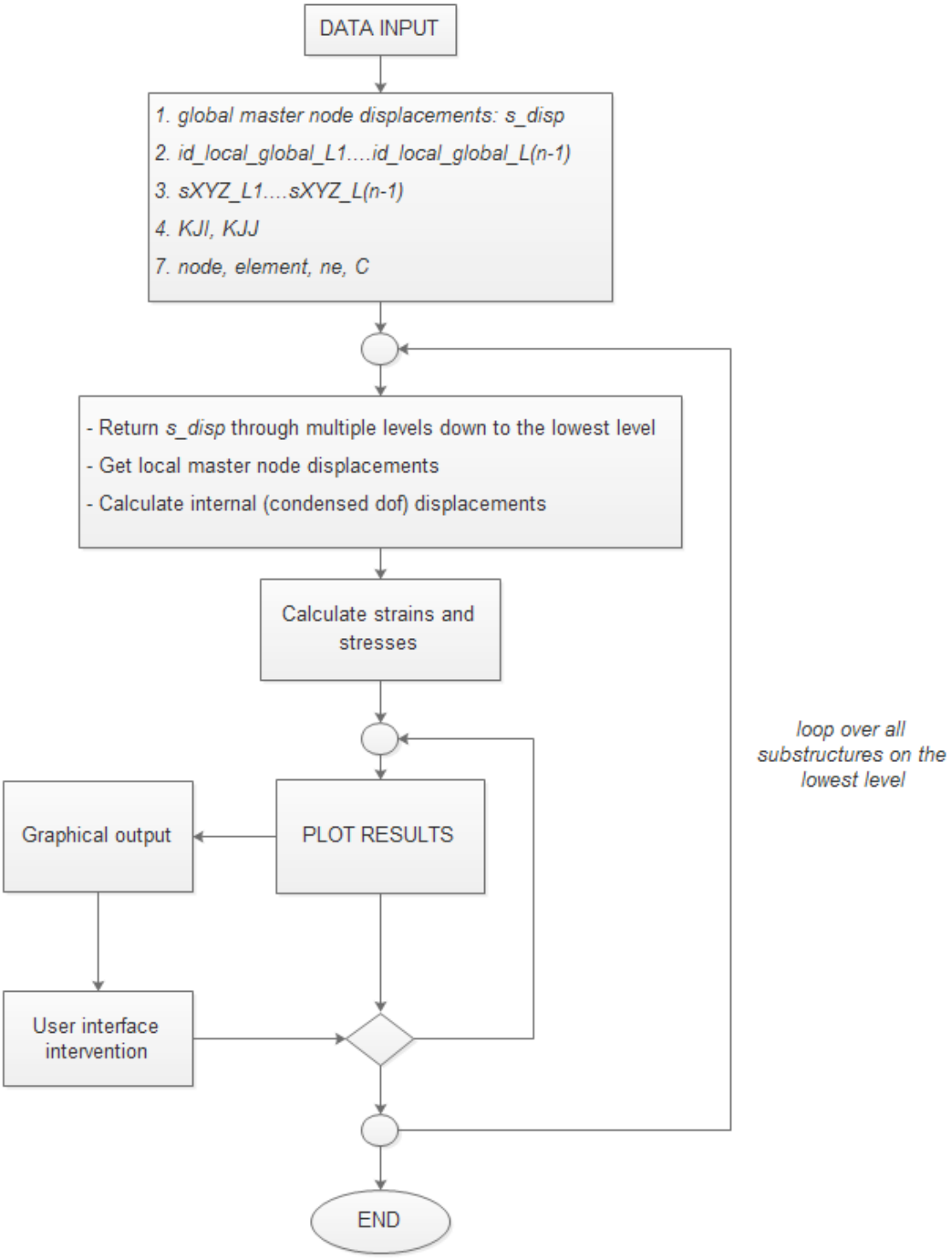

Figure 3.1d: Phase four in multilevel substructuring with FEM

Data flow diagrams presented in Figures 3.1*a* – 3.1*d* sum up all the described steps in multilevel substructuring from previous chapter to four phases.

As we can see, phase one consists of recognizing the first level superstructure and its appropriate master nodes, and performing top – down substructuring to the lowest level by fulfilling this task.

Phase two represents the lowest level substructure processing. On this level we generate finer meshes and basically perform conventional FEM analysis and obtain the classical stiffness matrix of all substructures on this level. The final step of this phase consists of condensing the desired dofs (most often internal) and obtaining condensed stiffness matrix for each substructure.

The third phase consists of processing data from the lowest level local model through multiple layers up to the global level – one model, in a bottom – up direction. We are basically transferring and building stiffness matrices from one level to another, through multiple levels until we reach level one and obtain the global level – one stiffness matrix of our superstructure. Regarding the programming aspects, the last step of obtaining the global stiffness matrix in three – dimensional systems is often not trivial and demands high level programming skills and ground understanding of physics involved, especially in the way of how the dofs are constrained for different types of structures during the analysis. Third phase finishes with implementing the BCs and solving the system of equations to obtain the global displacements of the master nodes.

The final, fourth phase goes in the opposite direction from the previous one as it demands processing new obtained global data from the highest level global model through multiple layers down to the local, lowest level model. It goes in top – down direction, as we need to return the global master node displacements through multiple levels to the lowest level and transform them accordingly to be able to get the previously condensed dof displacements. From these obtained displacements strains and stresses are calculated, and we are now able to continue to post – processing and visualize results of the analysis with the desired user interface intervention until end.

4. Programming concept in substructuring with FEM

The procedure of substructuring presented in the previous chapter introduces the need of developing programming techniques to be able to fully utilise the potential of substructuring concept. Since the whole concept lies on the grounds of various FEM formulations, this programming need imposes itself almost naturally. However, since the technique has some unique characteristics in the concept idea and its formulation which distincts it from practically every standard FEM formulation, for the very same reason the programming aspects are different and need to be examined more closely.

Programming concept can most efficiently be shown through practical implementations. In the next few chapters we shall directly emphasize this approach and apply it to solve various problems in stress analysis of thin-walled structures, ranging from one to three – dimensional and explaining the programming concept on the way. All developed programming codes for the substructuring technique are presented in Appendix E.

4.1. Application of substructuring and superelements technique in 1D and 2D test examples

To illustrate the need of programming the substructuring procedure for real life practical stress analysis problems, first example (one – dimensional bar) will be solved using this technique by hand. This will be the first and basic application of chapters 2 and 3 and will serve as a ground basis to develop programming procedure itself. Together with this first example, two – dimensional thin plates of various complexity shall be analysed and solved, to show how different physical and geometrical constraints can lead to a various pre – processing setup of initial problems. These examples shall incorporate the programming procedure. In all these cases, as later in three – dimensional examples also, pre – processing decision making has very important role and its efficiency often depends on the experience of the analyst.

Regarding the three – dimensional problems in stress analysis of thin – walled structures and its application through this technique, it is better to keep them separate from one and two –

dimensional analysis, since there are some implications regarding the coordinate system transformation and general constraints that make this type of analysis more difficult to handle, especially in multilevel substructuring.

4.1.1. 1D bar element

Consider a bar with a length of *4L* and an axial force *F* on its right edge, with pinned and roller support on both edges, as presented in Figure 4.1. Displacement analysis needs to be performed through an implementation of the substructuring and superelements technique. Displacements and reaction forces need to be determined in chosen nodes, depending on discretization.

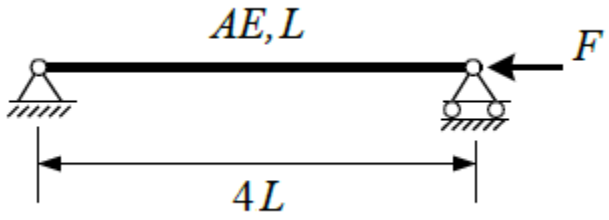

Figure 4.1: One – dimensional bar with an axial force *F* on the right edge

So, the first step is recognizing a level one superstructure, choosing the disassembly way and selecting the global master nodes. The selection of master nodes depends directly on the way how we choose our substructures for the level two, because the interface nodes of different substructures on the level two must be selected as master nodes also. That is not necessary for the other way around, i.e. master nodes are not necessarily the interface ones, but the interface ones (at least on the level two) are master nodes as well. The level one superstructure for the problem from Figure 4.1 is shown in Figure 4.2.

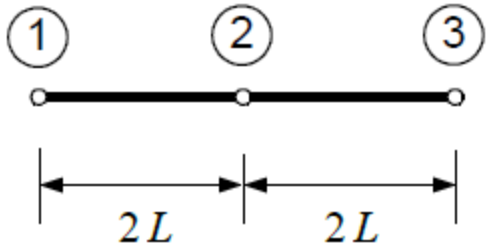

Figure 4.2: Level one superstructure for a one – dimensional bar substructuring analysis

Master nodes for the upper superstructure are chosen in such way that the node number 2 is the interface node for the two substructures on which we decided to divide our structure to.

Now we go down to the level two, where we have two substructures for which we repeat the upper steps. Figure 4.3 shows the discretization on level two.

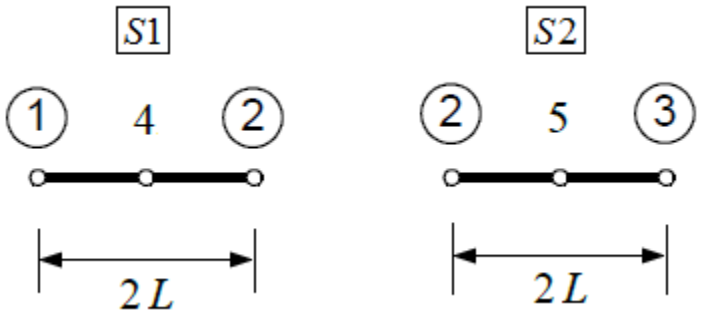

Figure 4.3: Level two substructures and its appropriate global (master) and local nodes

Nodes 4 and 5 are the local nodes for substructures S1 and S2, respectively. It should be noted that if we want to continue with substructuring to another level, substructures S1 and S2 which are substructures of the global superstructure from level one, would then also become superstructures for their appropriate substructures on level three. Now let's take a look on what happens when we continue with substructuring to level three.

On level three, we have four one – dimensional bar elements whose length is *L*, as shown in Figure 4.4.

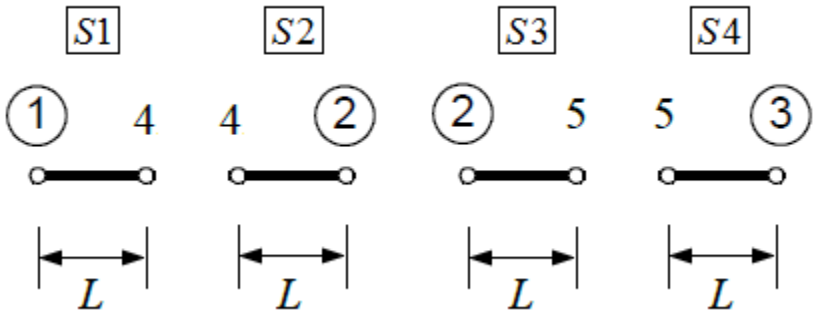

Figure 4.4: Level three bar elements of length *L*

Notice that the substructures on level three which are building the level two are actually simple one – dimensional bar elements whose stiffness matrix is given by:

$$K_{L3\boxed{S1}} = K_{L3\boxed{S2}} = K_{L3\boxed{S3}} = K_{L3\boxed{S4}} = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \tag{4.1}$$

So, these elements play the role that shall later be taken (2D examples) by triangular CST and LST elements, on the lowest level of substructuring where classical FEM approach needs to be implemented. Because of this fact, this example is basically a two – level one and an instance of global – local analysis. In the sense stated in Chapters 2 and 3, the lowest level for this example is actually a level two because further breakdown leads to the basic elements from which the classical FE mesh is constructed. Of course, the stiffness matrices from Eq. 4.1 have different numeration which, following the Figure 4.4 may be expressed as:

$$id_{L3\boxed{S1}} = \begin{Bmatrix} 1 \\ 4 \end{Bmatrix}; \quad id_{L3\boxed{S2}} = \begin{Bmatrix} 4 \\ 2 \end{Bmatrix}; \quad id_{L3\boxed{S3}} = \begin{Bmatrix} 2 \\ 5 \end{Bmatrix}; \quad id_{L3\boxed{S4}} = \begin{Bmatrix} 5 \\ 3 \end{Bmatrix}; \tag{4.2}$$

From these elementary stiffness matrices we are now building the stiffness matrices for the two substructures on level two. We'll substitute $k_1 = \frac{AE}{L}$.

$$K_{L3\boxed{S1}} = k_1 \begin{matrix} \\ 1 \\ -1 \end{matrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{matrix} 1 & 4 \\ 1 \\ 4 \end{matrix} \quad ; \quad K_{L3\boxed{S2}} = k_1 \begin{matrix} \\ 4 \\ 2 \end{matrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{matrix} 4 & 2 \\ 4 \\ 2 \end{matrix} \quad ;$$

$$K_{L3\boxed{S3}} = k_1 \begin{matrix} \\ 2 \\ 5 \end{matrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{matrix} 2 & 5 \\ 2 \\ 5 \end{matrix} \quad ; \quad K_{L3\boxed{S4}} = k_1 \begin{matrix} \\ 5 \\ 3 \end{matrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{matrix} 5 & 3 \\ 5 \\ 3 \end{matrix} \quad .$$

Now the stiffness matrices for the two substructures on level two (Figure 4.3), are:

$$K_{L2\boxed{S1}} = k_1 \begin{matrix} \\ 1 \\ 2 \\ 4 \end{matrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{matrix} 1 & 2 & 4 \\ 1 \\ 2 \\ 4 \end{matrix} \quad ; \quad K_{L2\boxed{S2}} = k_1 \begin{matrix} \\ 2 \\ 3 \\ 5 \end{matrix} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{matrix} 2 & 3 & 5 \\ 2 \\ 3 \\ 5 \end{matrix} \quad ;$$

So, the equations of a level two system for substructures S1 and S2 are:

$$\boxed{S1}: \quad k_1 \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix} \cdot \begin{Bmatrix} d_{1\boxed{S1}} \\ d_{2\boxed{S1}} \\ d_{4\boxed{S1}} \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_4 \end{Bmatrix} \tag{4.3}$$

$$\boxed{S2}: \quad k_1 \begin{bmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ -1 & -1 & 2 \end{bmatrix} \cdot \begin{Bmatrix} d_{2\boxed{S2}} \\ d_{3\boxed{S2}} \\ d_{5\boxed{S2}} \end{Bmatrix} = \begin{Bmatrix} F_2 \\ F_3 \\ F_5 \end{Bmatrix} \tag{4.4}$$

Now we need to condense the chosen degrees of freedom (the ones that are not master dofs). It can clearly be seen that these nodes are node number 4 for substructure S1 and node number 5 for substructure S2 (see Figure 4.3).

The process of static condensation for substructures S1 and S2 is presented below.

SUBSTRUCTURE S1:

$$k_1\left[\begin{array}{cc|c} 1 & 0 & -1 \\ 0 & 1 & -1 \\ \hline -1 & -1 & 2 \end{array}\right]\cdot\left\{\begin{array}{c} d_{1\boxed{S1}} \\ d_{2\boxed{S1}} \\ \hline d_{4\boxed{S1}} \end{array}\right\}=\left\{\begin{array}{c} F_1 \\ F_2 \\ \hline F_4 \end{array}\right\}$$

The above equation can be presented in the form:

$$\left[\begin{array}{c|c} K_{bb} & K_{bi} \\ \hline K_{ib} & K_{ii} \end{array}\right]\cdot\left\{\begin{array}{c} d_b \\ \hline d_i \end{array}\right\}=\left\{\begin{array}{c} F_b \\ \hline F_i \end{array}\right\}, \tag{4.5}$$

which is an equivalent of Eq. 2.2 from Chapter 2. Now following the procedure from stated chapter, we can clearly see that this system of equations can be reduced to the equivalent of Eq. 2.5:

$$\tilde{K}_{bb\boxed{S1}}\cdot d_b=\tilde{F}_b, \tag{4.6}$$

where

$$\begin{aligned}\tilde{K}_{bb\boxed{S1}} &= K_{bb}-K_{bi}\cdot K_{ii}^{-1}\cdot K_{ib} \\ \tilde{F}_b &= F_b-K_{bi}\cdot K_{ii}^{-1}\cdot F_i\end{aligned}.$$

Having in mind that there are no internal forces – the external force does not act on the internal nodes, so $F_i=0$, we have $\tilde{F}_b=F_b$. So the Eq. 4.6 comes down to:

$$\tilde{K}_{bb\boxed{S1}}\cdot d_b=F_b \quad\Rightarrow\quad \left(K_{bb}-K_{bi}K_{ii}^{-1}K_{ib}\right)\cdot d_b=F_b$$

From the above expression, for substructure S1 we now have:

$$\underbrace{k_1\left\{\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}-\begin{bmatrix}-1\\ -1\end{bmatrix}\cdot 2^{(-1)}\cdot\begin{bmatrix}-1 & -1\end{bmatrix}\right\}}_{\tilde{K}_{bb\boxed{S1}}}\cdot\begin{Bmatrix}d_1\\ d_2\end{Bmatrix}=\begin{Bmatrix}F_1\\ F_2\end{Bmatrix}$$

The condensed stiffness matrix is now:

$$\tilde{K}_{bb\boxed{S1}} = k_1 \begin{bmatrix} 1/2 & -1/2 \\ -1/2 & 1/2 \end{bmatrix} = \frac{k_1}{2} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix},$$

so now we have

$$\tilde{K}_{bb\boxed{S1}} = \frac{AE}{2L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}. \tag{4.7}$$

The Eq. 4.7 corresponds to the stiffness equation of one – dimensional bar element whose length is *2L*. From Figure 4.3 we can see that is actually the case with substructure S1. Now the condensed system equation for the substructure S1 on level two is:

$$\frac{AE}{2L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \cdot \begin{Bmatrix} d_1 \\ d_2 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix} \tag{4.8}$$

SUBSTRUCTURE S2:

The same procedure for substructure S2 can be performed, where we need to condense the degree of freedom number 5.

$$k_1 \left[\begin{array}{cc|c} 1 & 0 & -1 \\ 0 & 1 & -1 \\ \hline -1 & -1 & 2 \end{array}\right] \cdot \left\{\begin{array}{c} d_{2\boxed{S2}} \\ d_{3\boxed{S2}} \\ \hline d_{5\boxed{S2}} \end{array}\right\} = \left\{\begin{array}{c} F_2 \\ F_3 \\ \hline F_5 \end{array}\right\}$$

On the same way we get the condensed stiffness matrix:

$$\tilde{K}_{bb\boxed{S2}} = \frac{AE}{2L} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \tag{4.9}$$

and the condensed system equation for substructure S2 on level two:

$$\frac{AE}{2L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\cdot\begin{Bmatrix} d_2 \\ d_3 \end{Bmatrix}=\begin{Bmatrix} F_2 \\ F_3 \end{Bmatrix} \tag{4.10}$$

Here we can see the same analogy.

The condensed system equations 4.8 and 4.10 are describing substructures S1 and S2 after the condensation, where they contain only the global master nodes, as depicted in Figure 4.5.

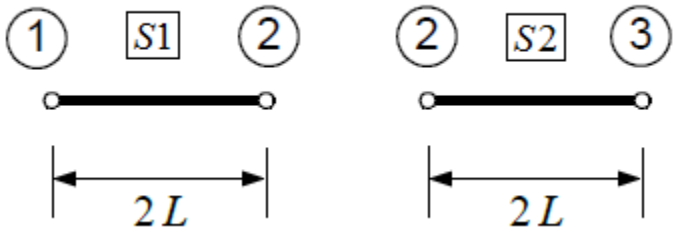

Figure 4.5: Substructures on level two after the condensation

Now from these equations through a conventional approach we are building the stiffness matrix and the matrix equation of the whole system for the superstructure on level one. So, the numeration is as follows:

$$\begin{matrix} & 1 & 2 & \\ \frac{AE}{2L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\cdot\begin{Bmatrix} d_1 \\ d_2 \end{Bmatrix}=\begin{Bmatrix} F_1 \\ F_2 \end{Bmatrix} & \begin{matrix} 1 \\ 2 \end{matrix} \end{matrix}$$

$$\begin{matrix} & 2 & 3 & \\ \frac{AE}{2L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}\cdot\begin{Bmatrix} d_2 \\ d_3 \end{Bmatrix}=\begin{Bmatrix} F_2 \\ F_3 \end{Bmatrix} & \begin{matrix} 2 \\ 3 \end{matrix} \end{matrix}$$

So for the level one superstructure from Figure 4.2 the global matrix equation is now:

$$\begin{matrix} & 1 & 2 & 3 \end{matrix}$$

$$\frac{AE}{2L}\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{Bmatrix} d_1 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ F_2 \\ F_3 \end{Bmatrix} \tag{4.11}$$

In Eq. 4.11 we have:

$$d_1 = 0\,;\, d_2 = ?\,;\, d_3 = ?\,;$$
$$F_1 = ?\,;\, F_2 = 0\,;\, F_3 = F.$$

If we substitute $k_2 = \frac{k_1}{2} = \frac{AE}{2L}$, the system of equations comes down to:

$$k_2\begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{Bmatrix} 0 \\ d_2 \\ d_3 \end{Bmatrix} = \begin{Bmatrix} F_1 \\ 0 \\ F \end{Bmatrix}.$$

Now this system needs to be solved for the unknown global displacements of the superstructure master nodes, d_2 and d_3. Of course, $d_1 = 0$ because of the pinned support at node one. Now we have:

$$(1) \qquad -k_2 d_2 = F_1$$
$$(2) \qquad 2k_2 d_2 - k_2 d_3 = 0$$
$$(3) \qquad -k_2 d_2 + k_2 d_3 = F$$

From these equations we can easily get the needed displacements d_2 and d_3, which are:

$$\boxed{d_2 = \frac{F}{k_2} = \frac{2FL}{AE}}$$

$$\boxed{d_3 = \frac{2F}{k_2} = \frac{4FL}{AE}}$$

Finally, from the first equation we have:

$$\boxed{F_1 = -F}\,.$$

Now all the global displacements and reaction forces for a level one superstructure are obtained.

To get the internal displacements of our local nodes from level two, we need to return the global displacements to that level and use the Eq. 2.4 from Chapter 2.

For the substructure S1 we have:

$$d_i = K_{ii}^{-1} \cdot \left(F_i - K_{ib} \cdot d_b \right)$$

$$F_i = F_4 = 0\,;\, d_i = d_4\ ,$$

that is,

$$d_4 = 2^{(-1)} \cdot \left\{ 0 - \begin{bmatrix} -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ F/k_2 \end{bmatrix} \right\} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ F/k_2 \end{bmatrix} = \frac{1}{2}\frac{F}{k_2}$$

$$\boxed{d_4 = \frac{FL}{AE}}$$

For the substructure S2 we have:

$$d_i = K_{ii}^{-1} \cdot \left(F_i - K_{ib} \cdot d_b \right)$$

$$F_i = F_5 = 0\,;\, d_i = d_5\ ,$$

that is,

$$d_5 = 2^{(-1)} \cdot \left\{ 0 - \begin{bmatrix} -1 & -1 \end{bmatrix} \cdot \begin{bmatrix} F/k_2 \\ 2F/k_2 \end{bmatrix} \right\} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \cdot \begin{bmatrix} F/k_2 \\ 2F/k_2 \end{bmatrix} = \frac{3}{2}\frac{F}{k_2}$$

$$\boxed{d_5 = \frac{3FL}{AE}}\,.$$

Displacements and reaction forces are determined in chosen nodes, so now the whole displacement analysis is performed through an implementation of the substructuring and superelements technique.

When we look at the whole procedure that needed to be carried out for such a simple one – dimensional problem, it is obvious why numerical implementation and programming must be involved, to be able to analyse complex engineering problems in stress analysis in a simpler and more efficient way.

4.1.2. 2D plane stress example

The implementation of substructuring procedure to two – dimensional stress analysis problems shall be performed by incorporating developed algorithms, presented in the Appendix E. Presented below is one of the similar types of problems we dealt with in Appendix C, where we implemented conventional FEM to a range of plane stress analysis examples. Now we shall implement substructuring and superelements technique to this type of problems, perform stress analysis and compare the analysis results with conventional FEM.

Consider a problem presented in Figure 4.6. Thin plate fixed on the left edge is subjected to an in-plane bending load, applied on the right edge.

$L = 5m,\ h = 3m,\ t = 0.01m,\ F = 2kN,\ E = 210GPa,\ \nu = 0.33.$

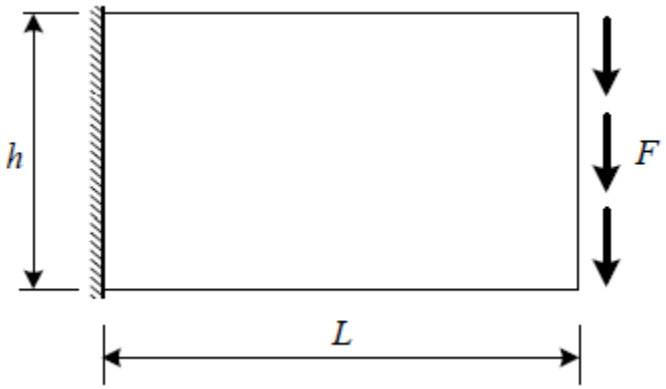

Figure 4.6: Thin plate subjected to an in-plane bending load

To implement substructuring technique to this problem, the first step is to recognize a global level one superstructure. Then we choose in how many substructures we shall divide our global superstructure and where will be the interfaces of all these substructures. We select the global master nodes that we will generate on this structure (using one – dimensional two-node line elements during mesh generation in this case), and choose the mesh density which is also desirable to bring in this phase.

So, for the sake of simplicity, we have chosen our whole structure from Figure 4.6 to be divided into total of 2 substructures which are, as presented in Figure 4.7, marked as S1 and S2, respectively.

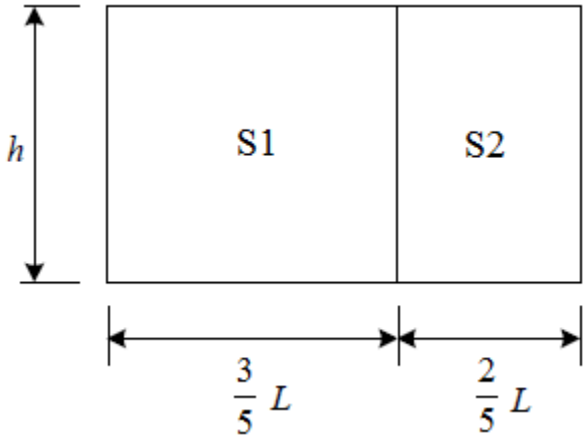

Figure 4.7: Substructuring of the global level one superstructure

So, our superstructure for this problem and its mesh would be as shown in Figure 4.8. As this is our top level for this problem, we call it a level one superstructure. This convention will also be used later in every practical application in this thesis, including a three – dimensional model of a ship cover, consisting of total of seven level two substructures.

What we basically did was defined the geometry from Figure 4.7, and then generated a mesh consisting of one – dimensional two – node line elements to this geometry. This mesh generated nodes along the boundary of our model from Figure 4.6, and nodes on the interface of our two substructures which we have chosen to divide our model to (Figure 4.7). It is very important to define the interface line during geometry definition of the superstructure, since

without it, it would not be possible to have a connectivity of these 2 substructures later in the process of obtaining the stiffness matrix of the global model.

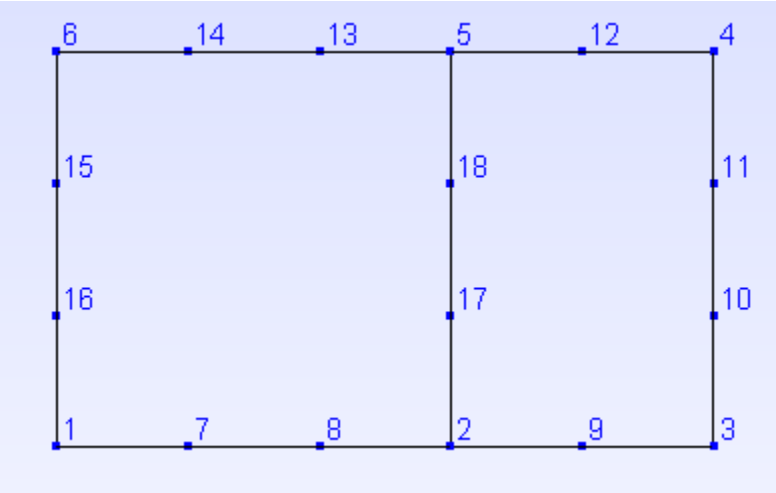

Figure 4.8: Level one superstructure with master nodes and global master node ids shown

Now we read in the nodal coordinate matrix of the global level one master nodes, which is done in the part of the Matlab code shown below. We save this coordinate matrix to disc, as it shall frequently be used later so it is more convenient to load it again instead of re-generating it by calling the function *readnodes.m* which imports it from Gmsh.

```
clc;
clear all;

% Level one superstructure – reading in the global master nodes

[node_L1,nid_L1]=readnodes('test_superstructure_LVL1.msh');
node_L1=node_L1(:,1:2); % just first 2 columns (z-coord =0, 2D example)

% Master („super“) nodes for further levels transfer

sNODE = size(node_L1);
for ii=1:sNODE
   sXYZ(ii,:) = node_L1(nid_L1(ii),:);
end

save('sXYZ');
```

Since the geometry of the problem from Figure 4.6 is fairly simple, there is no need to continue with substructuring to lower levels. So, our lowest substructuring level would be level two, in which we have two substructures presented in Figure 4.9, which we are now dealing with separately. The important thing to note here is that, since we concluded that two levels are enough for this problem, this is now an instance of global – local analysis. The procedure and steps described in Chapter 3 are generic – they refer to the multilevel substructuring analysis, but since the global – local analysis is practically a sub-type of multilevel analysis consisting of just two levels, the procedure presented there and all its steps can also be followed.

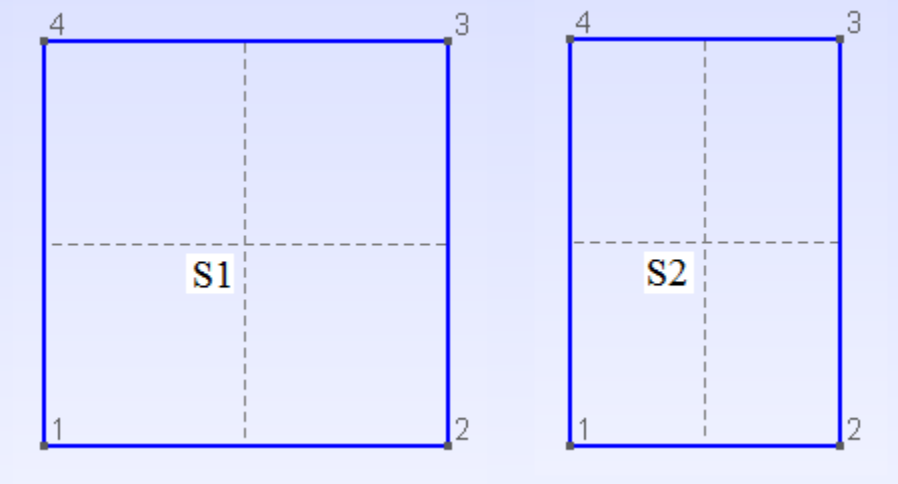

Figure 4.9: Geometry of the substructures S1 and S2 with point labels shown

Since this is our lowest level, the next step is to generate standard FE mesh of the geometry of these substructures, by choosing finite elements of the desired type. It is very important here to notice that the mesh element size which we defined on our superstructure when generating its mesh (i.e. mesh density), should be the same when generating the mesh of the (local) substructures, regardless of the type of finite element we have chosen. That would secure that the nodal coordinates along the edge of the substructures would match those in our (global) superstructure. This fact is very important because then we are able to transfer variables from local to the global analysis, and vice versa, much more easily (even though local and global node ids obviously will not match – we will be discussing later on how to overcome this problem).

Again, for the sake of simplicity and demonstration, we choose CST elements, and now generate a mesh of these triangular elements for the two substructures, presented in Figure 4.10, with local nodes and local node ids shown.

Now, before we continue describing the next step, we shall go back a bit, explaining why we choose our superstructure to be as it is.

Since our problem consisted of boundary conditions defined just on the edges of a plate (the left and the right edge), it was most convenient for us to define the superstructure in exactly that way as in Figure 4.8. What that basically means is that we are now able, through the process of static condensation, to eliminate (condense) all the internal degrees of freedom of both of our substructures, since the whole global model and the boundary of the local model (i.e. the substructures), contain all the nodes to which the boundary conditions are needed to be applied later in the solving process.

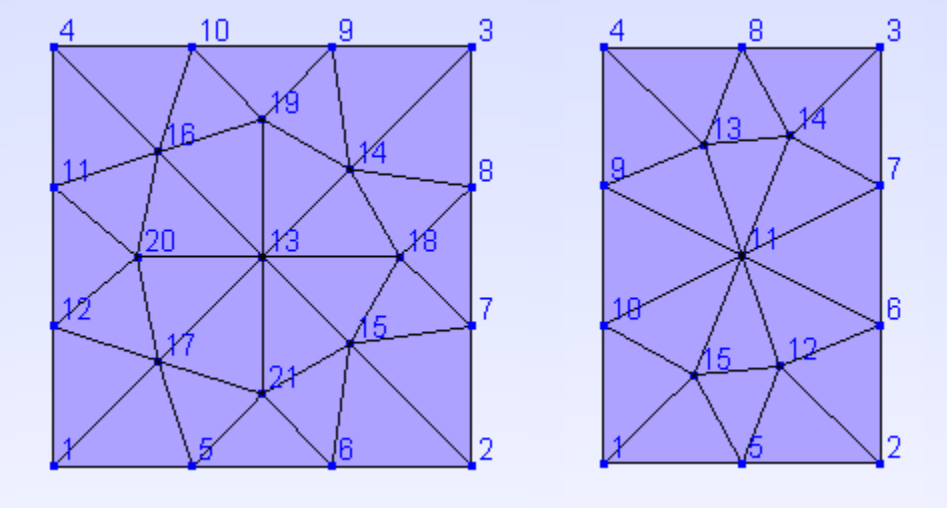

Figure 4.10: Mesh of the substructures S1 and S2 with local node ids shown

So, what we now need to do is to obtain a stiffness matrix of these substructures, which we do using the good old conventional FEM approach, already performed in Appendix C for CST and LST elements. These stiffness matrices would have the size of two times the number of

nodes in the whole substructure (internal and boundary nodes), since this is a two – dimensional problem and we have 2 dofs per each node. The next step is very important. The stiffness matrices of the substructures need to be arranged in such a way that the boundary dofs and the internal dofs are separated, i.e. that it first shows the dofs on the boundary, following the dofs in the interior of the substructure, or vice versa.

The basic idea is that if we want to eliminate (or better to say condense) dofs of our choice, they need to be sorted together in the stiffness matrix – at the beginning or at the end of the matrix.

This can be performed in an elegant way thanks to the way Gmsh is generating the mesh and numbering the nodes. When we generate the geometry in Gmsh and then follow by generating a mesh to that geometry, nodes number always follow the geometry, i.e. nodes at the defined points are numbered first, following by nodes at the defined lines (boundary), and finishing by nodes in the interior of the domain. That secures that the stiffness matrix can always be sorted in such a way that the boundary degrees of freedom are placed first in the stiffness matrix, even when we use second order elements for example. But, we may want to preserve some node in the interior also, where may be a crack or some discontinuation that we would want to analyse later on locally. That can also be done thanks to the way physical ids can be defined in Gmsh. There is a possibility to define a physical point in Gmsh, anywhere in our structure during geometry definition. When we assign a property of a physical point to some coordinate in our geometry that may be in the interior for example, that would secure that Gmsh numbers the corresponding node at that coordinate before any other node in the interior. And consequently, the desired node would also be sorted together with the boundary nodes and not condensed out during the process of reduction.

So, now the process of static condensation can be performed, as already shown in Chapter 2 and applied in Chapter 4.1.1 for the one – dimensional example.

The reduced or condensed stiffness matrix, obtained from Eq. 2.6, describes the substructures S1 and S2 in Figure 4.11, which consist only of the nodes on the boundary. Potentially, as stated, substructures can also contain some nodes in the interior if we chose to

preserve some of them for global analysis, but in this example there was no need to preserve any node in the interior so all internal nodes were condensed out, as shown in Figure 4.11.

It is very important to notice here that because of the form of Eq. 2.6, condensed stiffness matrix has built-in information of the stiffness of the whole substructure, even though its size is far lesser than the size of the stiffness matrix obtained through conventional FEM before condensation.

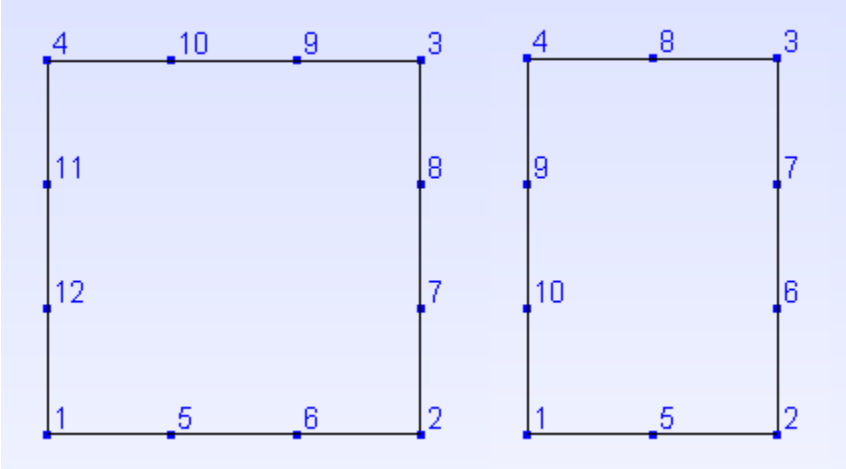

Figure 4.11: Substructures S1 and S2 with condensed internal degrees of freedom

The next step is to connect these condensed stiffness matrices to obtain the global stiffness matrix of the superstructure from Figure 4.8, which can be done by using a scattering operator in a for loop, as presented in the part of the MatLab code shown below.

```
clc;
clear all;

load('sXYZ');

nn_global=size(sXYZ,1); % number of global nodes and dofs
ndof_global=2*nn_global;

KG_L1=zeros(ndof_global,ndof_global); % or through sparse

nSS=2; % number of substructures

for ii=1:nSS
```

```
    substr_ii=sprintf('SUB_L1_S%d',ii);
    load (substr_ii,'K_red','id_local_global','NEQ')

    sctr(1:2:NEQ)=2*id_local_global-1;
    sctr(2:2:NEQ)=2*id_local_global;

    KG_L1(sctr,sctr)=KG_L1(sctr,sctr)+K_red;

    clear sctr
    clear K_red
    clear id_local_global
    clear NEQ
end

save('K_global_SE.mat','KG_L1')
```

Now that we obtained the global stiffness matrix of our level one superstructure, we can apply boundary conditions and solve the system for global displacements of the master nodes. The obtained global stiffness matrix of the superstructure is most convenient to be saved to disc, and then to be loaded again in the solving process.

After the system is solved and displacements of the master nodes obtained, we need to return these global displacements to the local analysis to obtain the internal displacements of the substructures from level two. That can be done easily from Eq. 2.4, but first the global master node displacements need to be transformed and expressed for each substructure on the local level.

For that purposes, one other variable is very useful to be able to transfer data from local to global analysis and vice versa correctly, due to the fact that the global and local node ids generally will not match. That variable is *id_local_global*, and is obtained through a function written in MatLab called *id_local_global_transfer.m*, which is developed for the purposes of this thesis and the implementation of the described technique. The code for the function is presented below.

```
function [id_local_global]=id_local_global_transfer(local_coord,global_coord)

nn_local=size(local_coord,1);
nn_global=size(global_coord,1);

id_local_global=zeros(size(local_coord,1),1);
```

```
for i=1:nn_local

   IND=0;
   for j=1:nn_global

      aa = local_coord(i,:) - global_coord(j,:);
      if ( sqrt(aa*aa') < 1.e-5)

         id_local_global(i,1)=j;
         IND=1;
      end

   end

      if(IND==0)
         disp('STOP in id_local_global_transfer():')
         disp( i )
         pause
      end
end
```

This function basically receives local (i.e. substructure) and global (i.e. superstructure) nodal coordinates, and returns a vector of node ids from the global model with the order of numbering from the local model. The function secures that all the data, displacements of the master nodes etc, is transferred correctly from one model to another through appropriate application of the scattering operator. For the detailed description of scattering operator see Appendix A.

After we obtained the internal displacements of the substructure, and transferred the appropriate master node displacements from the global model to a local one (current substructure), we have the vector of displacements of the whole substructure which we arrange in a way that the displacements on the boundary are always sorted first, following by internal displacements (or to say more generally, we arrange it in a way that the master node displacements are sorted first, following by condensed node displacements). The same procedure is to be performed for as many substructures as we can have in our model – in this example two.

Now that we have all the displacements of substructures on the local level, we can continue through the process of obtaining the stress and strain, which can be done with the re-application of the scattering operator and using functions from FEMLAB.

The plot of von Mises stress, with scaled displacements on the output plot for the two substructures from this example, using CST elements, is presented in Figure 4.12.

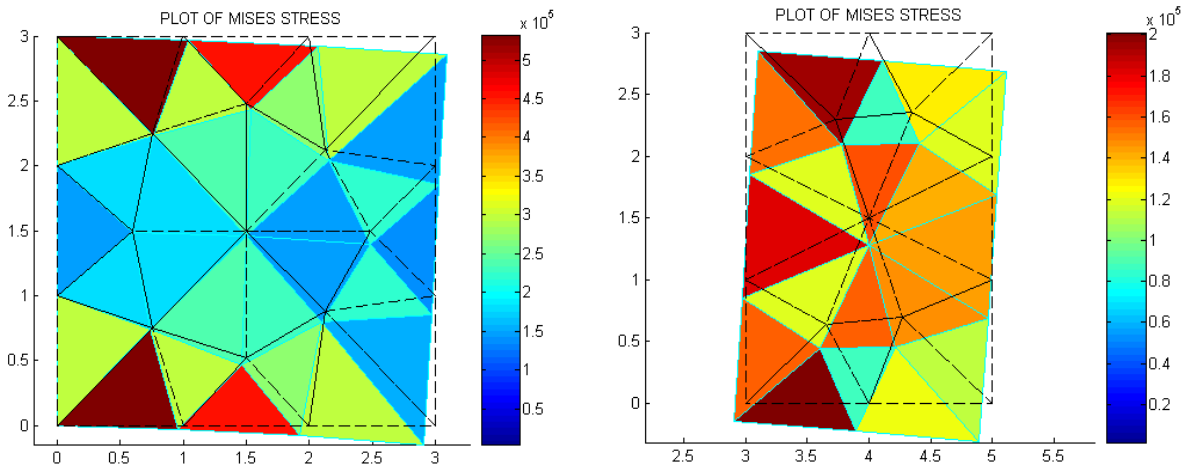

Figure 4.12: Plot of von Mises stress for substructures S1 and S2 using CST elements

Figure 4.13 shows the result of the same procedure, but when using LST elements in the discretization process for substructures on level two. We notice here that the displacements of the master nodes obtained when using LST elements are within the order of magnitude when compared with CST and with a slight difference. We also notice the difference in stress distribution when using LST elements, which is to be expected since the compressive stress which acts at the plate is more significant than the tensile one. This basically points to the fact that the results obtained when using LST elements are more accurate.

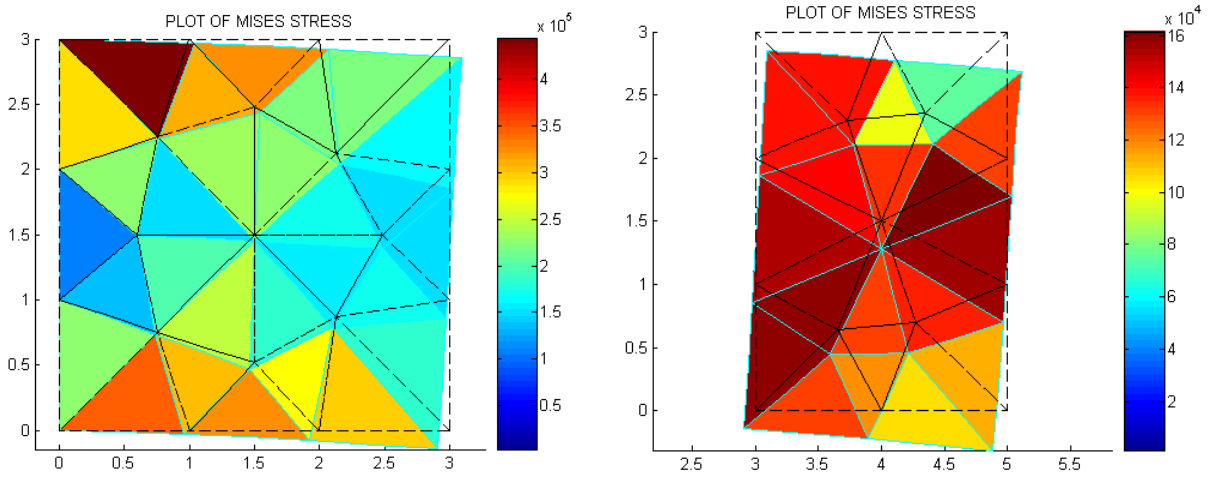

Figure 4.13: Plot of von Mises stress for substructures S1 and S2 using LST elements

Figure 4.14 shows the plot of von Mises stress when the analysis is performed through conventional FEM (CST on the left). First thing to notice here is that since the whole analysis is performed on the global structure, we have just one mesh for each type of finite elements used. The scaling in Figures 4.12 and 4.13 is done separately for each substructure, so the visual form of stress – displacement diagrams is not fully relevant to compare with the conventional FEM plots.

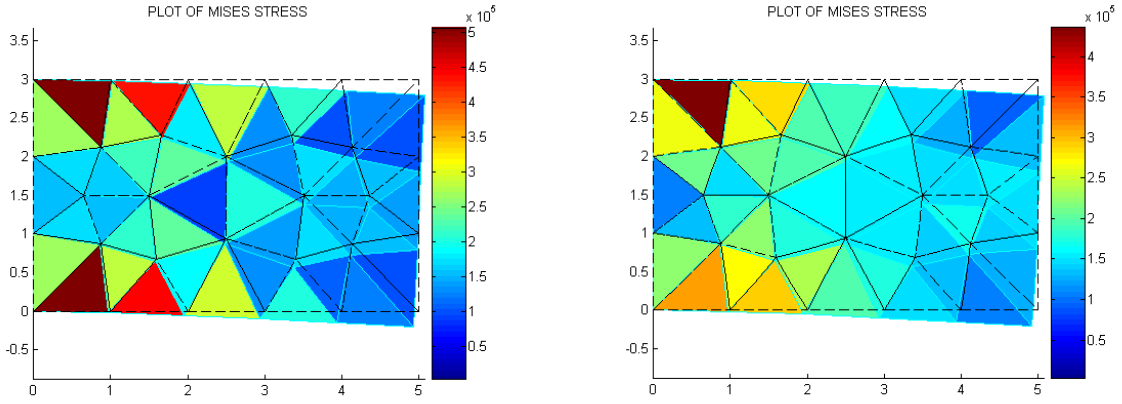

Figure 4.14: Plot of von Mises stress for the problem from Figure 4.6, using conventional FEM (CST on the left)

Table 4.1 shows the comparison of chosen obtained nodal displacements between the substructuring technique and conventional FEM, with two different types of elements used (CST and LST).

Element type	SUBSTRUCTURING TECHNIQUE			CONVENTIONAL FEM		
	node ID	*x – disp [m]*	*y – disp [m]*	*node ID*	*x – disp [m]*	*y – disp [m]*
CST	3	-0.0751	-0.2103	2	-0.0731	-0.2053
	4	0.0750	-0.2103	3	0.0730	-0.2052
	10	-0.0232	-0.2070	9	-0.0228	-0.2028
	11	0.0233	-0.2070	10	0.0229	-0.2027
LST	3	-0.0763	-0.2090	2	-0.0761	-0.2086
	4	0.0763	-0.2091	3	0.0761	-0.2086
	15	-0.0228	-0.2037	14	-0.0227	-0.2037
	16	0.0228	-0.2037	15	0.0227	-0.2037

Table 4.1: Comparison of displacement results between the substructuring technique and conventional 2D FEM approach

We can see that the results for obtained displacements compared to the conventional FEM approach practically match those from the substructuring technique. However, there is a slight difference. To be able to understand why this little difference between two approaches exist, we need to understand first that the substructuring technique is not an approximation technique in the context that it is approximated compared to the conventional FEM. The substructuring technique is only an approximation technique at the order of magnitude at which the standard, conventional FEM is (or any other FEM formulation which is being used for the discretization at the lowest level of substructuring). So, when compared to the standard conventional FEM, the results should not deviate even a little. So, why do they slightly deviate? This is a consequence of discretization. When we are performing substructuring we are separating the whole construction to several smaller parts. Then, for each of these parts (i.e. substructures) we are generating FE mesh on the lowest level. Now, as we are using an automatic external mesh generator, even if we define an equal mesh density on all levels of substructuring, because of the fact that we are generating the mesh e.g. two times (for this example) instead of one (in standard FEM approach), because of that fact some nodes and their coordinates in the interior of the substructures will not fully match those from standard FEM approach mesh. So the boundary nodes will match because the discretization always follows the geometry (on the boundary), but in the interior of our domain the mesh generator creates the elements and nodes following its

own logic (which is a consequence of its integrated algorithm that it uses for the mesh generation). But since for the substructures (which are of smaller dimensions than the whole construction) we are generating the mesh automatically, some of the coordinates inside will just not fully match. That does not mean that the result is not accurate, this small deviation is just a consequence of a slightly different discretization that always exists compared to conventional FEM. When we would be generating more and more dense meshes on substructures and conventional FEM analysed structures, the difference would just decrease more and more. Even now the difference shoes up at third decimal point, so we can safely conclude that the results are correct and the analysis performed through substructuring is accurate.

As a conclusion, we notice here that one of the main advantages of using this approach is the fact that by implying the boundary conditions on the global structure only, we can perform stress analysis on every substructure separately and independent of each other. It is because this technique allows us to transfer global conditions to the local analysis without actually implementing the boundary conditions to the local analysis. Now, why is that good? It is good because we do not need to have a set of boundary conditions for each substructure locally and therefore the whole analysis is much less sensitive to the occurrence of random errors due to potential misinterpretation of BCs, which often happens in practice when dealing with large and complex systems, especially on the interfaces of their sub-models.

Another great advantage is the division of labour, since we can perform analysis of different substructures independently of each other. So, when dealing with complex systems (i.e. ships, aircrafts etc.), separate groups of engineers can work on different sectors of these structures and divide the labour to become much more efficient.

Third and final great advantage is the direct consequence of the first one. When implementing the conventional FEM to complex structures, we can have hundreds of thousands, even millions degrees of freedom in our problem. This technique not just reduces the total number of dofs used for computation (thus reducing computational time), but also facilitates the visualization of different sectors and their analysis. When using conventional FEM, we always have one structure and therefore if we want to look at a specific element in the mesh or a desired node somewhere in the interior, we need to do a lot of graphical manipulations (e.g. splines, cuts and sections) to even be able to visually analyse some local area within this complex structure.

By using superelements approach, all we need to know is to which substructure our desired area belongs. And then we can perform all the analysis locally, by visualizing just that part of our structure.

All MatLab codes for the procedure described in this chapter are given in the Appendix E.

4.1.3. 2D plate with a non symmetrical hole

Consider a problem presented in Figure 4.15. This is the similar problem (slightly different steel alloy) which is solved in Appendix C with the use of classical FEM procedures. Here, for a thin steel plate with a non symmetrical hole subjected to an in-plane bending load, with fixed support on the left edge, stress analysis needs to be performed through the application of substructuring and superelements technique.

Parameters:

$$L = 6m,\ h = 1.1m,\ r = 0.3m,\ t = 0.01m,\ F = 20kN,\ E = 200GPa,\ \nu = 0.3.$$

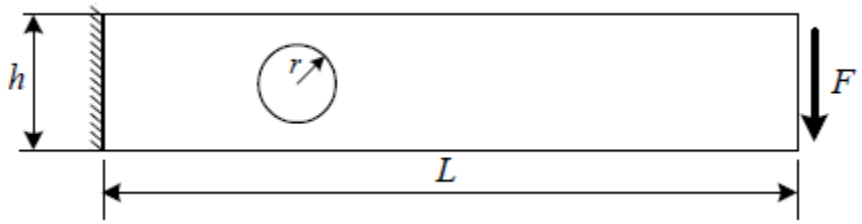

Figure 4.15: Thin plate with a non symmetrical hole

This is an interesting example because of the presence of the hole, not being symmetric in the constructions plane. It is obvious that we would want for our master nodes to include the boundary of the hole, since holes attract stress concentration and this is an important area to analyse later in the post – processing section.

Since the area on the boundary of the hole is described with trigonometric functions, because of its geometry the stress is also expected to change more rapidly. To be able to capture

this behaviour, we need to define our substructuring disassembly way in a clever fashion. We need to find a way to make a connection of the hole and the rest of the boundary, so that we can connect all those stiffness matrices later in the process of obtaining the global level one stiffness matrix. Also, it would be desirable to utilize the potential symmetry and develop it, if possible. So, we generate geometry in Gmsh and divide the whole structure into a total of ten substructures. The superstructure geometry and its mesh are presented in Figures 4.16 and 4.17.

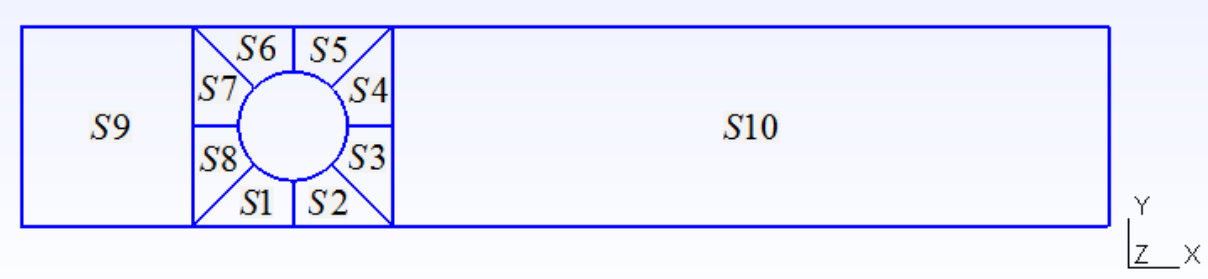

Figure 4.16: Superstructure geometry and its appropriate substructures

Figure 4.17: Superstructure mesh and its appropriate global level one master nodes

Now, we follow the described steps from previous chapters and perform the procedure of substructuring and code it.

This is also an example of global – local analysis, so the procedure is similar as for the example from Chapter 4.1.2. Although, this represents a quite more demanding type of analysis since we have larger number of substructures, more complicated geometry and much larger amount of data. Here for the first time we can open the question of manipulation with large data structures. All this data needs to be sorted and manipulated to be able to efficiently and elegantly shift from between different substructures and their levels.

So, we are transferring the master node ids to the local level, creating geometry and two – dimensional finite element mesh for each substructure, obtaining local stiffness matrices, sorting the dofs and calculating the reduced stiffness matrices.

Finite element meshes generated with triangular elements are presented in Figure 4.18. Again, mesh density needs to follow the one from global analysis, in order for the coordinates of the master nodes in the local model to match those from global one. Figure 4.18 shows the substructures prior to condensation.

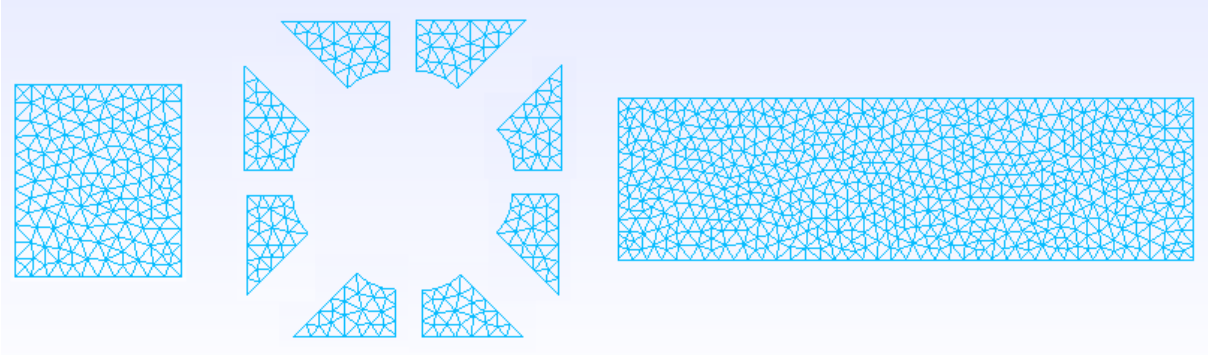

Figure 4.18: Triangular meshes for the substructures on level two before condensation

The rest of the analysis is now the same, following the recipe from Chapter 3 and the programming procedure performed already in Chapter 4.1.2. From the reduced stiffness matrices we are directly building the global stiffness matrix for the superstructure on level one, implement the boundary conditions and solve the system for global displacements. After the global displacements are obtained we transform and return them to the local analysis, calculate the stresses and strains, and visualize results.

Now we will present the stress distribution with scaled displacements on the output plot for the whole structure through its appropriate substructures. This is presented in Figures 4.19 – 4.22.

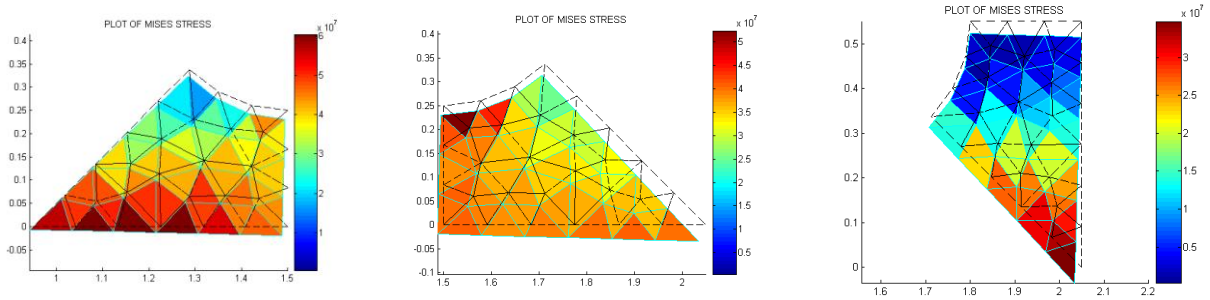

Figure 4.19: Von Mises stress distribution with scaled displacements for substructures S1 – S3

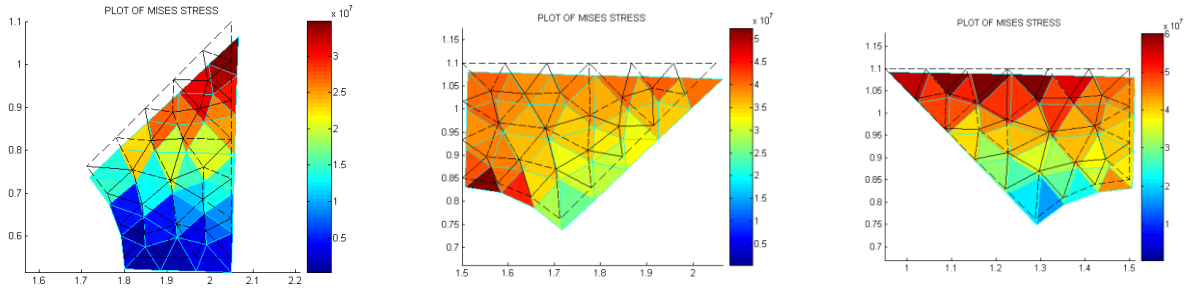

Figure 4.20: Von Mises stress distribution with scaled displacements for substructures S4 – S6

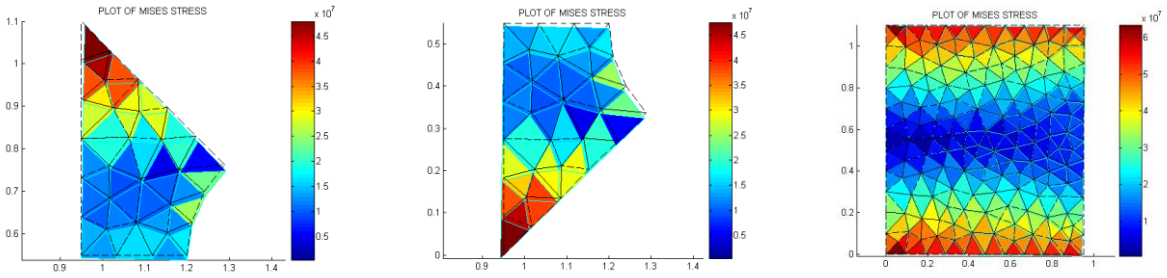

Figure 4.21: Von Mises stress distribution with scaled displacements for substructures S7 – S9

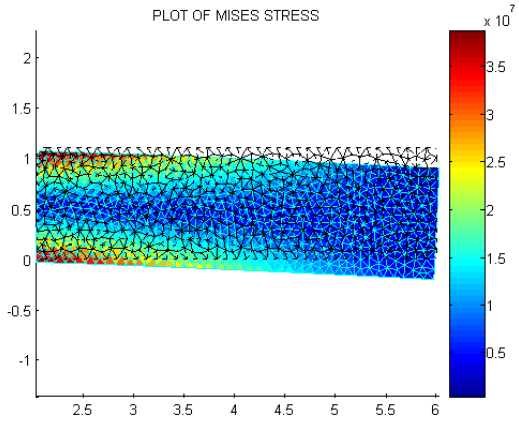

Figure 4.22: Von Mises stress distribution with scaled displacements for substructure S10

Table 4.2 shows the comparison of chosen obtained nodal displacements between the substructuring technique and conventional FEM. The obtained displacement results match the results from classical FEM at several orders of magnitude.

SUBSTRUCTURING TECHNIQUE			CONVENTIONAL FEM		
node ID	*x – disp [m]*	*y – disp [m]*	*node ID*	*x – disp [m]*	*y – disp [m]*
5	-0.0009	-0.0069	2	-0.0009	-0.0068
6	0.0009	-0.0069	3	0.0009	-0.0068
81	-0.0004	-0.0069	68	-0.0004	-0.0068
85	0.0002	-0.0069	72	0.0002	-0.0068
93	0.0009	-0.0060	80	0.0009	-0.0060
74	-0.0009	-0.0060	61	-0.0009	-0.0060

Table 4.2: Comparison of displacement results between the substructuring technique and conventional FEM approach

4.2. Programming substructuring technique in 3D

As already stated at the beginning of Chapter 4, it is better to keep one and two – dimensional stress analysis of thin – walled structures separated from the three – dimensional, especially regarding programming aspects. When performing three – dimensional analysis, very often local substructures on lower levels are disassembled to their appropriate two – dimensional components, and to one extent part of this analysis can be watched through the eyes of a two – dimensional problem. However, when these components and the data they contain need to be transformed, transferred and assembled to the upper spatial level and eventually to the level one superstructure which is also three – dimensional, general constraints in coordinate system transformation appear and potential adaptations to boundary conditions and additional dof constraints need to be made. This process often demands to be performed even on the local substructure level and in the very first steps of analysis, during the local mesh generation and prior to obtaining the stiffness and reduced stiffness matrices of the local level substructures. One of the reasons for this lies in the fact that although on a certain level substructures become two – dimensional, they can still lie in different planes from each other and some steps in the programming procedure just cannot be coded in a simple way for the algorithm to be automated. For the reasons stated, the most demanding part of programming the three – dimensional analysis is the recognition of these patterns and spatial components to which the specific substructure belongs, then in applying the algorithm procedure for the scattering operator, and finally in the local - global transfer (and vice versa) of node ids from one level to another, which may not be a part of the same coordinate system. And when we have large structures and a great number of substructures on different levels (and that is usually the case when working with this technique), and all these levels and substructures even on the same level need not to be in the same coordinate system, one can understand the complexity that this type of analysis brings altogether.

As stated earlier, the programming aspects most often offer best possible insight through practical implementation, and three – dimensional analysis is not an exception.

4.2.1. Simple 3D hexahedron case

Here a simple three – dimensional example shall be presented and analysed through substructuring, to illustrate the differences between the one and two – dimensional analysis performed by now and the spatial stress analysis that most often appears in practice.

Consider a spatial stress analysis problem, presented in Figure 4.23. Simple hexahedron case, six meters in length, 1.1 meter in height and 2.5 meters in width, built of a total of six steel plates of 10mm thickness, is subjected to a bending load of $500kN$ uniformly distributed along its one edge. The opposite side of the case is in fixed support. Young modulus is $200GPa$ and the Poisson's ratio $\nu = 0.3$.

Again, displacement and stress analysis needs to be performed through an application of the substructuring and superelements technique.

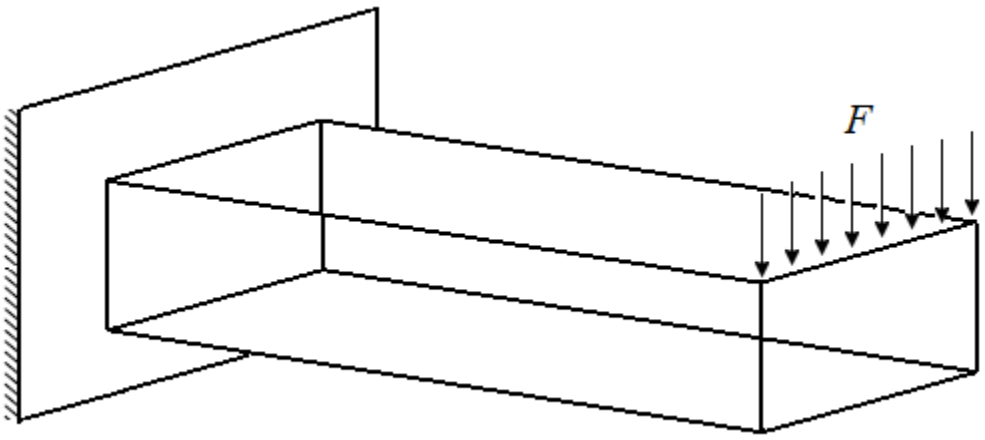

Figure 4.23: Bending of a three – dimensional hexahedron case

Geometry and substructures for this model are presented in Figure 4.24. It is logical do disassemble the superstructure in exactly this way, to the six plates from which it is built.

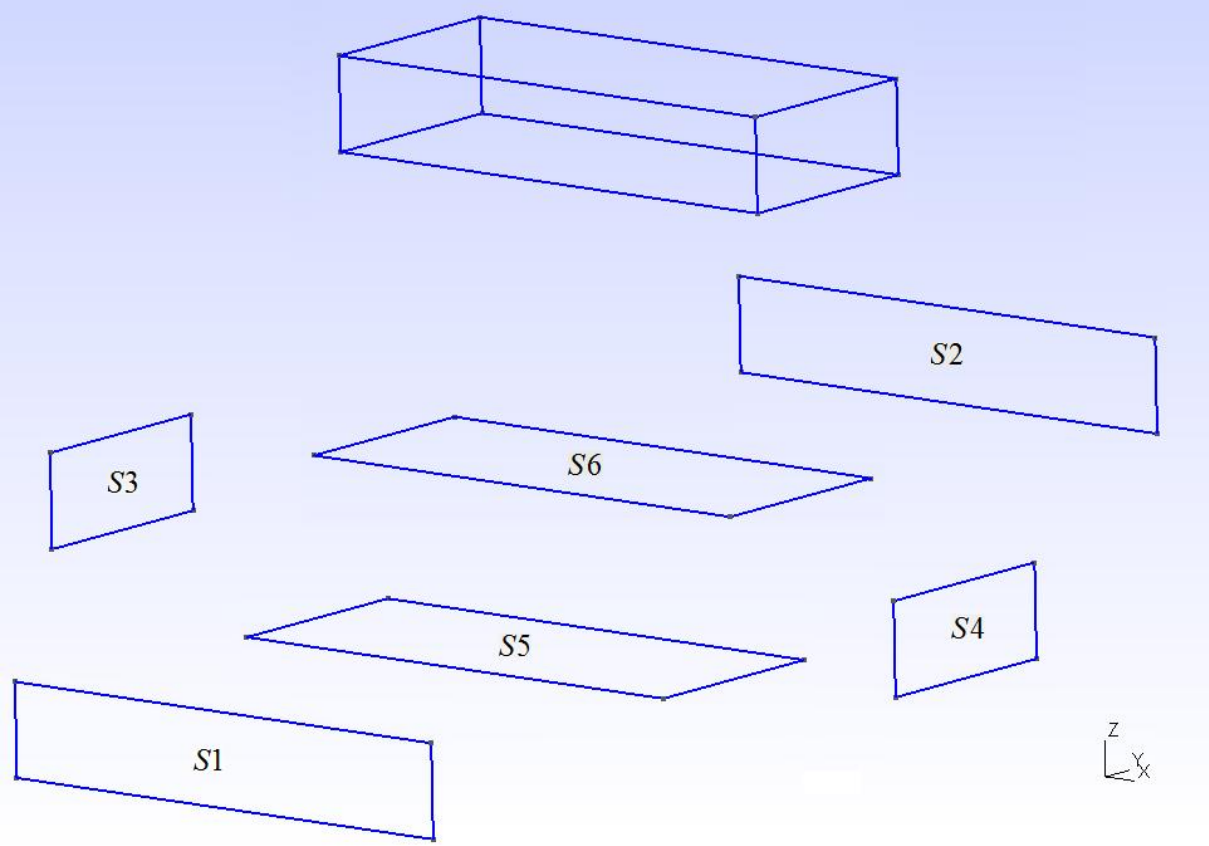

Figure 4.24: Geometry and substructures for a hexahedron case model

Level one superstructure mesh of this model is presented in Figure 4.25. The chosen master nodes are only the interface ones between different plates, for the reasons of simplification and demonstration through this basic spatial example.

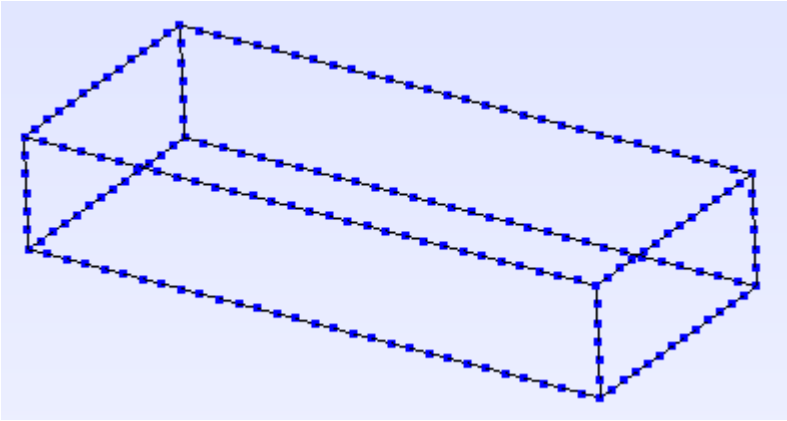

Figure 4.25: Level one superstructure mesh of the hexahedron case model

As before, first we need to read in the nodal coordinate matrix of the global master nodes. But, now we need to retain all three coordinates and import them in this form into Matlab, which is the first difference from 2D analysis.

Since this is also an instance of global – local analysis, now the substructures are analysed locally, and 2D finite element mesh for each substructure is created.

Prior to the process of obtaining the stiffness matrices and the condensed ones, we need to manipulate with the variables that contain local nodal coordinates. Since we have three pairs of substructures that are located in different planes, we can use mirroring and symmetry for them but first we need to route the procedure for each substructure in such way that the program for a 2D analysis which we use for every substructure pair recognises its nodal coordinates as like they are in the *x*-*y* plane, although some of them actually are not. In other words, if we have a plate that lies in the *x*-*z* plane for example, and this plate is translated by some value along the *y* axis, we need to skip the *y* axis and route the nodal coordinate matrix of the substructure in a two – dimensional form containing just *x* and *z* coordinates. The part of the code which utilises this routing is presented below, in the example of the substructure located in the *x*-*z* plane and translated along *y* axis.

```
[node_S2,nid_S2]=readnodes('substructure_S2_LVL2.msh');  % read in mesh from gmsh file
element_S2=readelements('substructure_S2_LVL2.msh',7);

node_S2_2D=node_S2(:,1:2:3); % the plate is in the x-z plane, y=const., route to the x and z coordinates

element_S2=renumber(element_S2,nid_S2);  % renumber the elements and remove duplicate entries

%  edge (master nodes) detection
% read in local master node ids on the substructure
id_edge_snodes=readnodeset('substructure_S2_LVL2.msh',8);

NODE_S = size(id_edge_snodes,1); % number of nodes*number of dof per each node
NEQ = NODE_S * 2;
```

Now we can continue through the process of obtaining the local stiffness matrices and condensed ones afterwards, but having in mind to always send the 2D routed nodal coordinate matrix to all the functions used for the displacement processing phase. After the condensed stiffness matrices are obtained, we can continue the forming of the global stiffness matrix of our level one, three - dimensional superstructure. This process is coded in the algorithm presented

below. We can see that for the pairs of substructures located in the same 2D planes scattering operator loops in the same direction.

```
clc;
clear all;

load('sXYZ');

nn_global=size(sXYZ,1);
ndof_global=3*nn_global;

KG_L1=zeros(ndof_global,ndof_global);

KG_L1_S1=zeros(ndof_global,ndof_global);  % or through sparse
KG_L1_S2=zeros(ndof_global,ndof_global);
KG_L1_S3=zeros(ndof_global,ndof_global);
KG_L1_S4=zeros(ndof_global,ndof_global);
KG_L1_S5=zeros(ndof_global,ndof_global);
KG_L1_S6=zeros(ndof_global,ndof_global);

% ------------------------------------------------------------------------

%  FIRST SUBSTRUCTURE:

   load ('SUB_L2_S1.mat','K_red','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
% sctr( :2:NEQ)=3*id_local_global-1;  % S1 is in the x-z plane, we don't have y freedoms
   sctr(2:2:NEQ)=3*id_local_global;

   KG_L1_S1(sctr,sctr)=KG_L1_S1(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global
   clear NEQ

%  SECOND SUBSTRUCTURE:

   load ('SUB_L2_S2.mat','K_red','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
% sctr( :2:NEQ)=3*id_local_global-1;  % S2 is in the x-z plane
   sctr(2:2:NEQ)=3*id_local_global;

   KG_L1_S2(sctr,sctr)=KG_L1_S2(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global
   clear NEQ
```

```
%  THIRD SUBSTRUCTURE:

   load ('SUB_L2_S3.mat','K_red','id_local_global','NEQ')

% sctr( :2:NEQ)=3*id_local_global-2;    % S3 is in the y-z plane, we don't have x freedoms
   sctr(1:2:NEQ)=3*id_local_global-1;
   sctr(2:2:NEQ)=3*id_local_global;

   KG_L1_S3(sctr,sctr)=KG_L1_S3(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global
   clear NEQ

%  FOURTH SUBSTRUCTURE:

   load ('SUB_L2_S4.mat','K_red','id_local_global','NEQ')

% sctr( :2:NEQ)=3*id_local_global-2;    % S4 is in the y-z plane
   sctr(1:2:NEQ)=3*id_local_global-1;
   sctr(2:2:NEQ)=3*id_local_global;

   KG_L1_S4(sctr,sctr)=KG_L1_S4(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global
   clear NEQ

%  FIFTH SUBSTRUCTURE:

   load ('SUB_L2_S5.mat','K_red','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
   sctr(2:2:NEQ)=3*id_local_global-1;
% sctr( :2:NEQ)=3*id_local_global;  % S5 is in the x-y plane, we don't have z freedoms

   KG_L1_S5(sctr,sctr)=KG_L1_S5(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global
   clear NEQ

%  SIXTH SUBSTRUCTURE:

   load ('SUB_L2_S6.mat','K_red','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
   sctr(2:2:NEQ)=3*id_local_global-1;
```

```
% sctr( :2:NEQ)=3*id_local_global;  % S6 is in the x-y plane

   KG_L1_S6(sctr,sctr)=KG_L1_S6(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global
   clear NEQ

% ------------------------------------------------------------------------

KG_L1=KG_L1_S1+KG_L1_S2+KG_L1_S3+KG_L1_S4+KG_L1_S5+KG_L1_S6;

save('KG_L1')
```

After we obtained the global level one stiffness matrix, the next step is standard, solving the system $K \cdot d = f$. Some implications of the 3D substructuring programming regarding the implementation of specific boundary conditions cannot be discussed here as they demand working on more complex problems to show up. The stated will be discussed in Chapter 5.

Now as the global displacements are obtained we need to return them to the local analysis, but now alongside with the transformation of the master node ids from the global level to a local one, we need to transform the displacements from a three – dimensional system to a two – dimensional one. For that reason, we need to have additional application of the scattering operator prior to the stress calculation, to perform the needed transformation. The part of the code which performs this task and calculates the internal displacements (from the previously condensed dofs) is presented below, for the substructure S1.

```
load('s_disp')
load('SUB_L2_S1','id_local_global','KJI','KJJ','node_S1','element_S1','ne','C')

nn_boundary=size(id_local_global,1);
ndof_boundary=2*nn_boundary;

  sctr(1:2:ndof_boundary)=3*id_local_global-2;
% sctr( :2:ndof_boundary)=3*id_local_global-1;   % x-z plane
  sctr(2:2:ndof_boundary)=3*id_local_global;

id_local_global_dof=sctr';

disp_boundary=s_disp(id_local_global_dof,:);

disp_internal=inv(KJJ)*(-KJI)*disp_boundary;

disp_S1=[disp_boundary; disp_internal];
```

From this obtained data we can now calculate stresses and strains, and visualise and interpret results.

Stress distribution and scaled displacements for the pairs of substructures that are in the same plane, are presented in Figures 4.26 – 4.28.

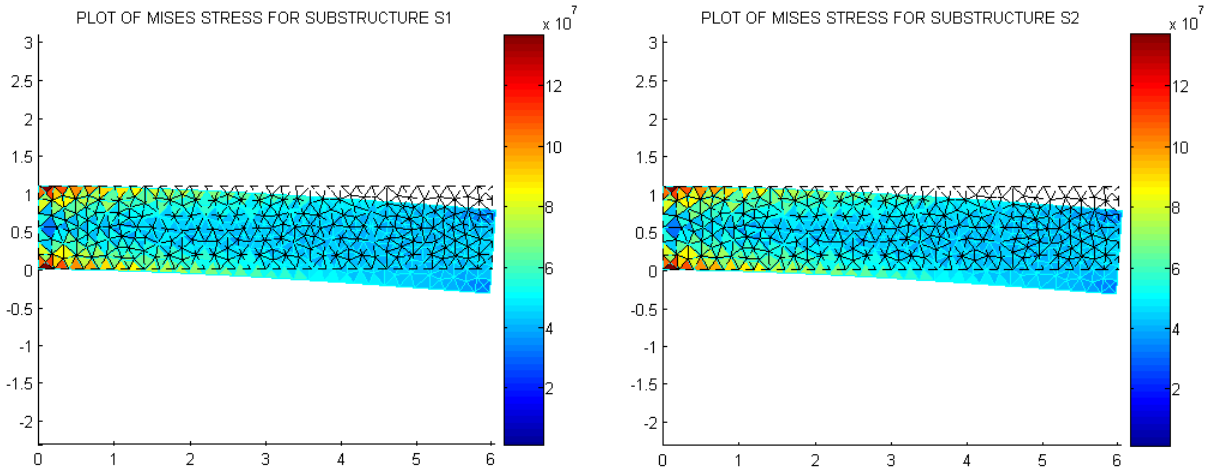

Figure 4.26: Substructures S1 and S2 stress and displacement distribution

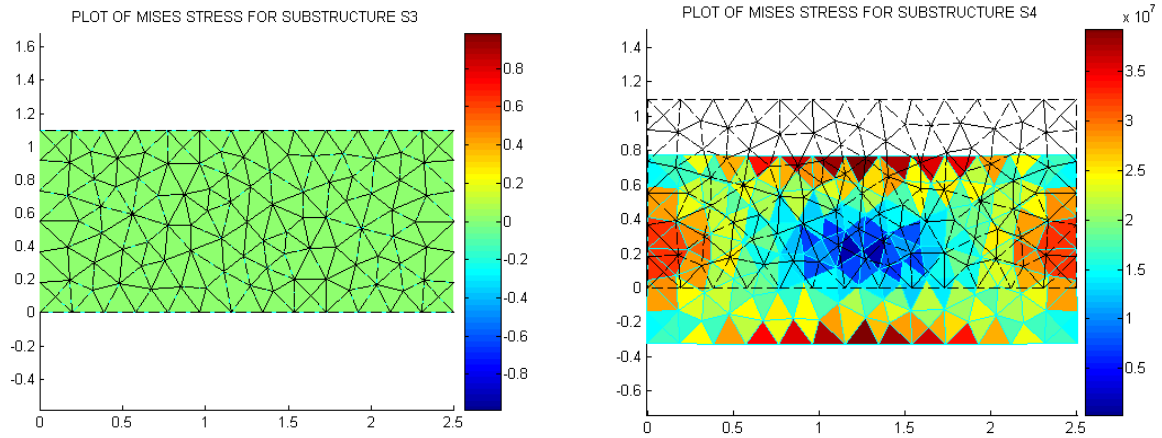

Figure 4.27: Substructures S3 and S4 stress and displacement distribution

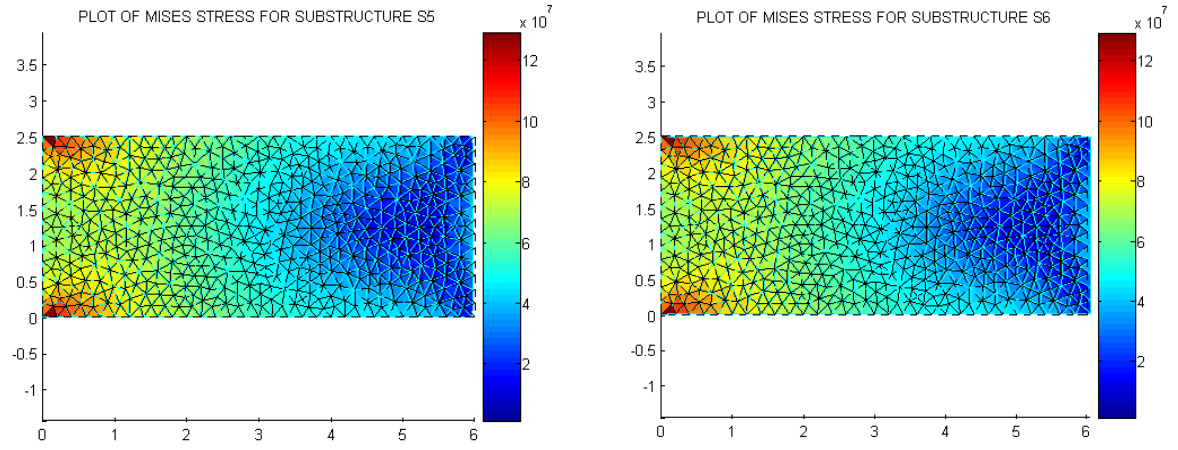

Figure 4.28: Substructures S5 and S6 stress and displacement distribution

There are some interesting conclusions that can be drawn from the performed analysis. Since each stress and displacement diagram is presented in the substructure's own plane, we can see how each plate moves under the given load conditions. Consequently, we see that the substructures S1 and S2 mainly bend (due to compressive and tensile stress) as in the separate 2D analysis performed earlier, with the stress distribution being basically mirrored. Substructures S3 and S4, however, have different behaviour. Substructure S3 is fixed, so the displacements and stresses are zero, while S4 is practically translated in the negative *z* direction (looking from the *x* – perspective) because of the direction of the external load, and with the maximum stress values in the middle of the upper and lower edge, which is to be expected. Because of the highest stress value in the top middle of S4, the displacements in the *z* direction shall also be slightly larger in the middle than in the corners so the translation is not literal. Of course, substructure S4 will also have displacement components in *x* direction, but since we are watching it from the *x* – axis this displacement component cannot be seen here. But, it can be captured by analysing other four substructures, e.g. by looking at the displacement of the right edge of substructures S1 and S2, through visualizing substructures S5 and S6, or analysing the numerical results for the displacements. Substructure S5 has the *x* – displacement component moving in the negative *x* direction (compressive stress), while the S6 in the positive direction (tensile stress), naturally, watching from the *z* perspective. S5 and S6 will also have displacement components in the *z* direction.

To be able to analyse the obtained displacements and compare them to the theoretical results, knowledge from mechanics of materials can be used. The construction from Figure 4.23 with steel plates whose thickness is 10mm can be compared to the cantilever beam with the box rectangle cross section, presented in Figure 4.29.

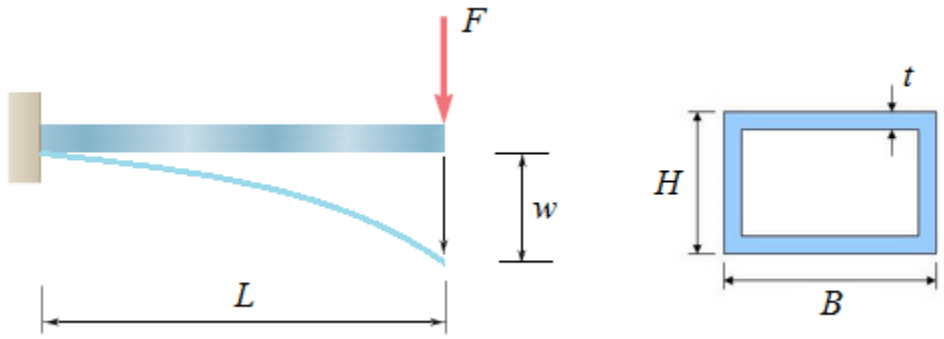

Figure 4.29: Cantilever beam under bending load. Cross section is on the right

So now we need to calculate the deflection w at the right edge of the beam, to get the range and order of magnitude to which the displacements on the edge of the hexahedron case from Figure 4.23 should fall into. So now we have, from the initial parameters for the hexahedron case:

$$L = 6m,\ F = 500kN,\ B = 2.5m,\ H = 1.1m,\ t = 0.01m.$$

From mechanics of materials we know that the deflection at the end of the beam (pure bending) is described with the equation:

$$w = \frac{F L^3}{3 E I_y} \tag{4.12}$$

Area moment of inertia of the given cross section can be calculated by:

$$I_y = I_{y1} - I_{y2} = \frac{B H^3}{12} - \frac{b h^3}{12},$$

where b and h are the internal width and height of the box cross section from Figure 4.29 and they are as follows:

$$b = B - 2t = 2.48m$$
$$h = H - 2t = 1.08m$$

So now the moment of inertia is:

$$I_y = 0.01695\, m^4,$$

and the deflection is:

$$w = \frac{F L^3}{3E I_y} = \frac{500 \cdot 10^3 \cdot 6^3}{3 \cdot 200 \cdot 10^9 \cdot 0.01695} = 0.0106195\, m$$

Table 4.3 shows the comparison of displacements in several nodes along the right edge of a 3D hexahedron case obtained through an application of the substructuring technique, and the calculated theoretical deflection on the cantilever beam using mechanics of materials.

The displacements match at several orders of magnitude with the slight difference appearing just from the fourth decimal point onwards. The difference is reasonable since for the real three - dimensional problem we have the load distributed on the whole plate instead of being concentrated in just one point, so the displacements in different nodes vary around that value.

MECHANICS OF MATERIALS	SUBSTRUCTURING TECHNIQUE	
deflection w [m]	*node ID*	*z – disp [m]*
0.0106195	2	-0.0129027
	6	-0.0129024
	40	-0.0129013
	110	-0.0129012

Table 4.3: Comparison of displacement results between the substructuring technique and theoretical value obtained from mechanics of materials

5. An application of substructuring and superelements technique in stress analysis of a 3D ship cover with one bulkhead

Complex engineering systems often demand interdisciplinary approach for finding efficient solutions and transferring them to the industry design demands. Large ships, especially heavy lift and cargo vessels, contain great number of structuring components, such as several levels of decks and tween decks. Often these structures undergo different types of stress and loads (hydrostatic pressure, wind, thermal and mechanical stresses, etc.), so different types of engineers are often employed to work in different design and analysis groups considering the type of problem being dealt with.

In this chapter a practical three – dimensional application for one ship cover shall be presented, from which the whole tween deck in a vessel, with any given number of bulkheads can be constructed. It will sum up all the gathered knowledge from this thesis and present and apply it in one place through the analysis of such engineering system of higher complexity.

Three – dimensional ship cover with one bulkhead is presented in Figure 5.1, according to the drawing made in Catia. The material being used is A36 structural steel with the Young modulus of elasticity $E = 200GPa$ and Poisson's ratio $\nu = 0.26$. Thickness of the plates is $t = 10mm$.

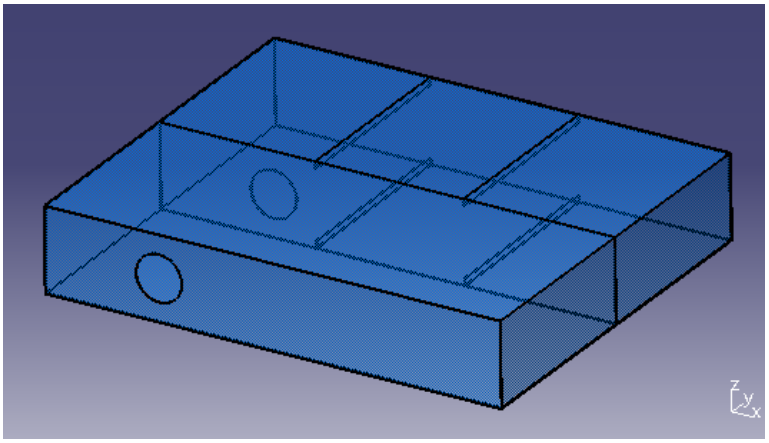

Figure 5.1: Three – dimensional ship cover with one bulkhead model

Later on, during the solution processing of this problem, before entering the system of equations for the global level one displacements of the master nodes, we shall constrain the dofs on the left edge of the whole cover, and apply bending force on the right edge, similar to the problem being analysed in Chapter 4.2.1. We need to note here that, depending on the actual conditions in a vessel, the type of support and loading can be various. These are one of the most often boundary conditions that show up in such systems in practice and that is the reason for such choice upon them.

When beginning the first steps in the substructuring analysis, one needs to be very careful and intuitive. Often the very first step of the analysis is proven to be the most important one, and when cautiously set it can reduce the error analysis and processing time considerably. This is an example of multilevel substructuring and in addition a three – dimensional one, so when building all the levels needed to complete the analysis, mistakes made especially regarding the choice of the master nodes at the beginning can strongly complicate and extend the analysis, and increase the number of iterations through which the satisfactory solution is obtained.

Geometry of the level one superstructure for our model from Figure 5.1 is presented in Figure 5.2.

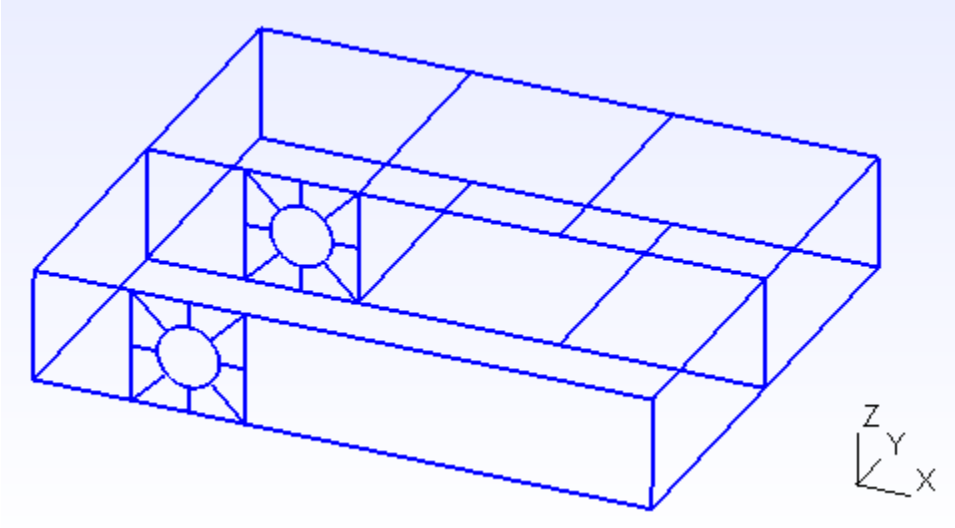

Figure 5.2: Geometry of the level one superstructure for a ship cover model with one bulkhead

Since the cover consists of several plates of various complexity, the form of such geometry is chosen for several reasons. The disassembly part is presented in Figure 5.3. There can be seen that on the first level breakdown, all substructures have physical significance, by representing plates from which the whole cover is built. Furthermore, substructures S6 and S7 have ribs in the construction, so this breakdown is also logical. But, for some substructures, such as S1 and S2, we generated more complex geometry right on the first level, even though some of the lines do not have physical significance at first. Part of the reason for this had already been discussed in Chapter 4.1.3, and considering a more complex problem being present here, one more detail needs to be added.

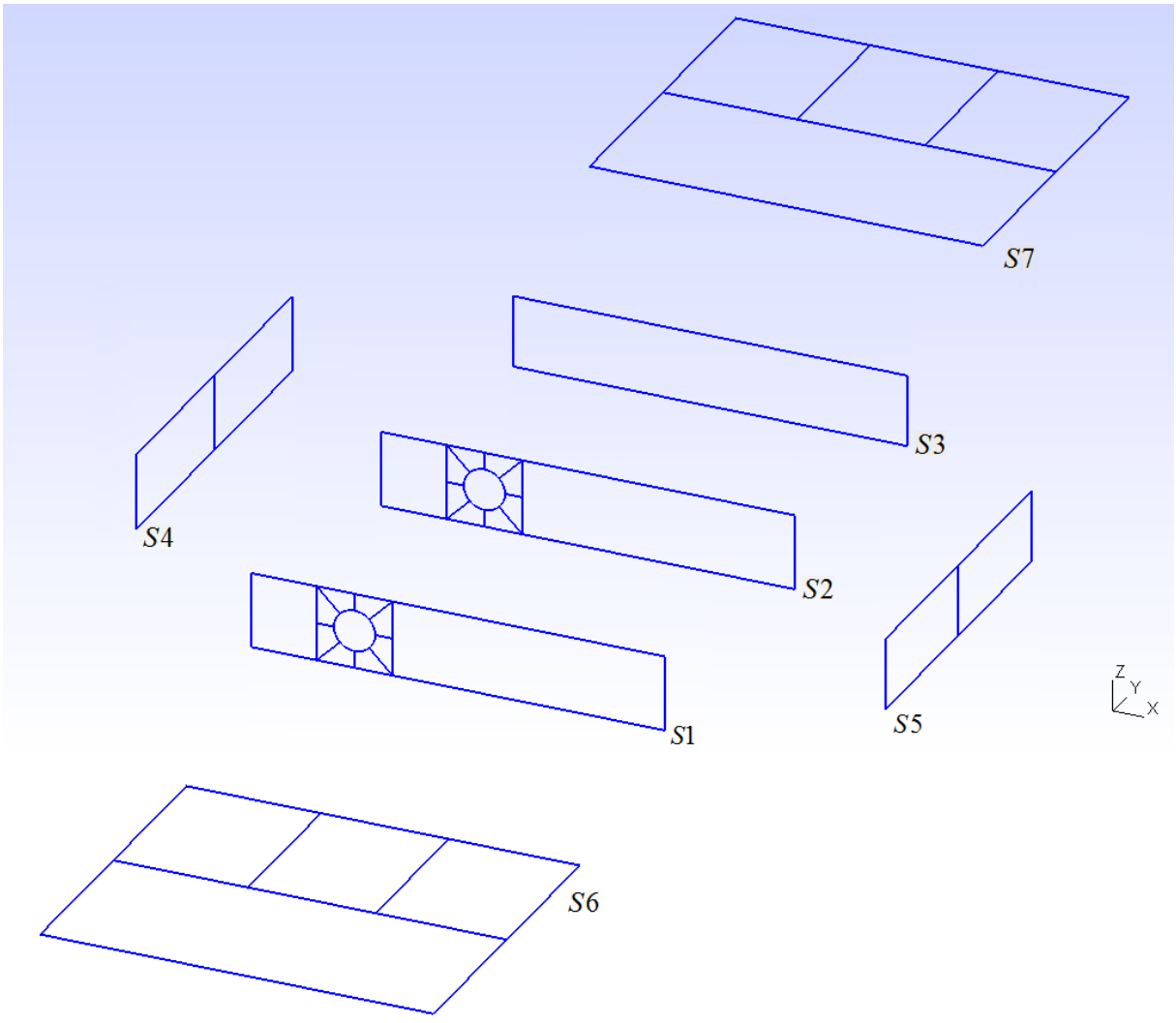

Figure 5.3: First level breakdown in a multilevel substructuring of the ship cover model

The reason for such disassembly lies in the fact that a global solution is possible to obtain in a more elegant and efficient way when using this type of geometry. If we choose our first level master nodes following the geometry in this way and include the nodes in a close proximity to the holes during mesh generation, it is simply more effective to transform global variables and transfer them through levels down to the lowest one and vice versa, since we can connect the levels for these substructures in a direct way during the programming part. This will be elaborated in the paragraphs to come.

Figure 5.4 shows the level one superstructure mesh consisting of a total of 442 global level one master nodes.

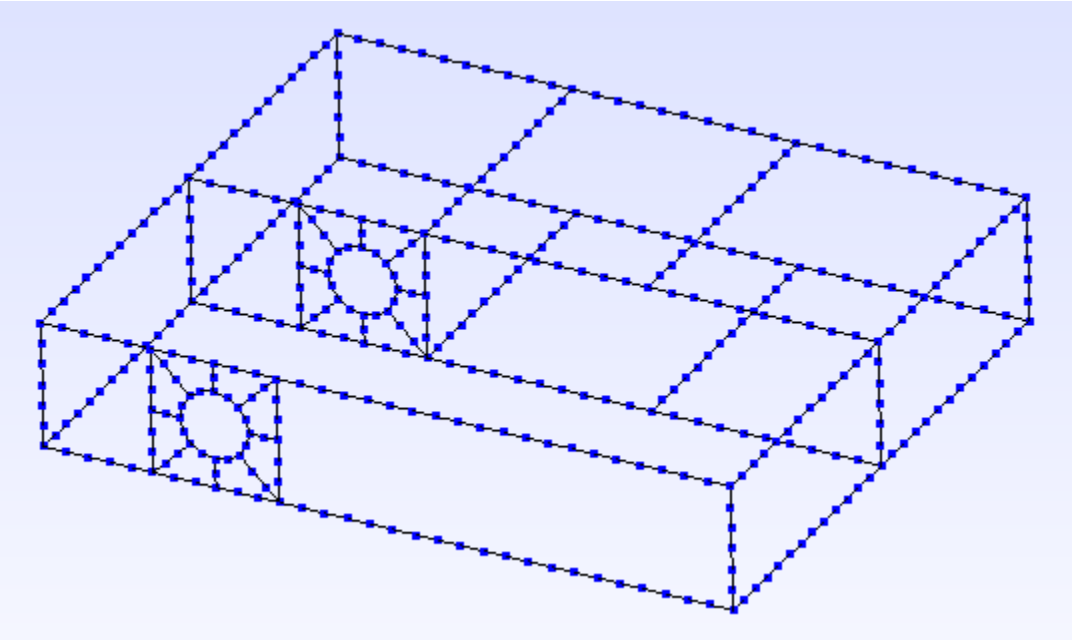

Figure 5.4: Level one superstructure mesh of the ship cover

Since this type of analysis incorporates a three – dimensional problem with multilevel substructuring, we need to note that we have superstructures on lower levels also. From Figure 5.3, every level two substructure (except S3) is also a level two superstructure for the substructures on level three. From the same figure it can be directly seen that on the level three we will have a total of 32 substructures, all of which once we get to this level, need to be processed, stiffness and condensed stiffness matrices obtained and calculated, etc.

So, to complete the first phase we basically need to perform the first three steps in multilevel substructuring described in Chapter 3.1, with one additional iteration performed since we have three levels of substructuring. Here we need to have in mind that from every upper level to a lower one, we need to transfer global master node ids to this next level, since generally node ids from the first level superstructure will not match those from the next level, and so on. So, before going down to the lowest level, we need to create the level two superstructures – their geometries and meshes, with the appropriate level two master nodes, and as in the first step, read in the nodal coordinate matrix of these level two master nodes and transfer and transform their ids to the level three.

After we transferred all the appropriate data down to the lowest level, we now create geometries and generate two – dimensional meshes for each of the 32 disassembled substructures, as presented in Figure 5.5 for the level two substructure, S6. Note how on the figure master nodes are distinguished from the internal ones which will be subjected to condensation. So, on this last level we obtain local stiffness matrices, condensed stiffness matrices, transfer master node ids to the upper level (level two) and finally, from all the condensed stiffness matrices of the appropriate substructures we build the global level two stiffness matrix of the appropriate level two superstructure.

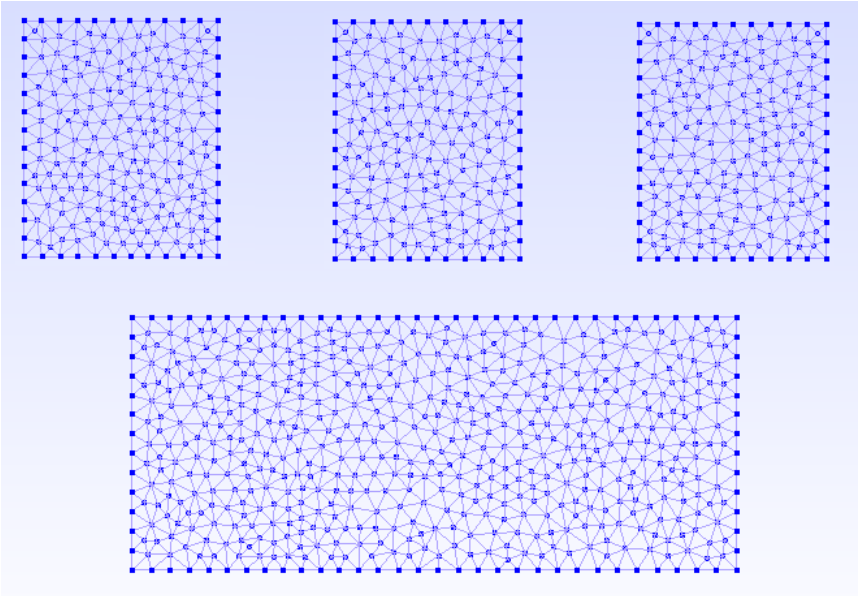

Figure 5.5: Two – dimensional meshes for the level three substructures of the superstructure S6 from level two, prior to condensation

Note that these steps, after we obtain global level two stiffness matrices, if they would finish now, would practically represent a procedure as in the global – local analysis, with the similar code being implemented and differed by the fact that on the level two we have multiple parallel superstructures. But now, instead of solving the system of equations for the level two displacements, from all the global level two stiffness matrices (in this example there are six of them plus the condensed one from the level two substructure S3) we need to build the global stiffness matrix for our level one superstructure.

So this whole procedure (at least when we have just three levels) can be looked at as like we have the two – dimensional global – local analysis as the first phase, and then the three – dimensional global – local analysis as the second phase and these two phases connected together to form a three dimensional multilevel substructuring procedure.

However, when we get to the step of implementing the boundary conditions to our level one superstructure prior to system solving, one additional implication needs to be noted since this was not obvious in the previous 3D solved example through global – local analysis. The difference between the mesh of the hexahedron case from Figure 4.25 and the mesh of the ship cover from Figure 5.4 lies in the fact that in the mesh of the ship cover, due to existing of ribs and the geometrical constraints in level two substructures S1 and S2, we have master nodes which are located at the interior of these plates (these nodes are not the interface ones although they are master). So, as a consequence of the fact that we are working with thin – walled structures and the system is three – dimensional, we assume that the displacements exist only in the substructure's own plane, but not perpendicular to it. This is the restriction that we are dealing with from the beginning, since the whole model which is being used to describe thin – walled structures is a membrane one and there are no internal forces acting on these nodes.

So, what we now need to do before entering the system of equations solver (otherwise we will get singular stiffness matrix during the solving process), is to constrain the displacements of these internal nodes in the direction perpendicular to the given plane. During this procedure we need to watch that the nodes which are interface ones remain intact. This can be done in an elegant way thanks to the possibility of defining physical ids in Gmsh, although some code manipulations need to be done. For additional details about physical ids in Gmsh, refer to Appendix B.

Presented below is the programming procedure for such implementation of boundary conditions and integration of the constraints defined by the used membrane model, into the code. Note that we chose to fix the left edge of the ship cover (*y-z* plane, substructure S4) and to define the bending force of $600kN$ applied and evenly distributed to the right edge of the cover (*y-z* plane, substructure S5). This situation is often the case in practice.

```
clc;
clear all;

% fixed support on the left edge
nfix=readnodeset('superstructure_LVL1.msh',123);

% internal nodes in the x-y plane
nfix_in_xy=readnodeset('superstructure_LVL1.msh',126);
nfix_in_xy=nfix_in_xy(9:size(nfix_in_xy,1));

% internal nodes in the x-z plane

nfix_in_xz=readnodeset('superstructure_LVL1.msh',125);
nfix_in_xz=nfix_in_xz(13:size(nfix_in_xz,1));

% bending load on the right edge
nload=readnodeset('superstructure_LVL1.msh',124);

% constrain all displacements in the fixed support (nfix), internal displacements in
%  z – direction for the x-y plane, and internal displacements in y – direction for the x-z plane

ifix=[ 3*nfix'-2  3*nfix'-1  3*nfix'  3*nfix_in_xy'  3*nfix_in_xz'-1 ];

load('sXYZ');

nn_global=size(sXYZ,1);
ndof_global=3*nn_global;

fext=zeros(ndof_global,1);
fext(3*nload)=-5797.1;

load('KG_L1')

[s_disp,freac]=fesolve(KG_L1,fext,ifix);

save('s_disp')
```

Note that the global displacement nodal vector will have the size of three times the total number of master nodes on level one. Since our level one superstructure mesh consisted of 442 nodes, we now have the displacement field of a total of 1326 displacement components. After

these displacements are obtained, we now need to return them to the local analysis through multiple levels, following the reverse procedure from the beginning. After we obtain internal displacements from the global ones, transformed to the lowest local level, we can calculate stresses and strains and present results.

All stress distribution diagrams shall not be presented here, since the total number of substructures is very large. Instead, we will focus on analysing deviations and maximum stress values at some sections, alongside with the maximum obtained displacements. Results of the stress analysis have been exported to Paraview Visualization Toolkit.

Figure 5.6 shows the largest value of von Mises stress in the whole construction. The stress is captured in two sections near the hole on a level two substructure S2 (bulkhead). These two sections belong to the appropriate level three substructures S2 and S5. This is to be expected since the bulkhead is located at the middle of the construction and bears largest amount of load.

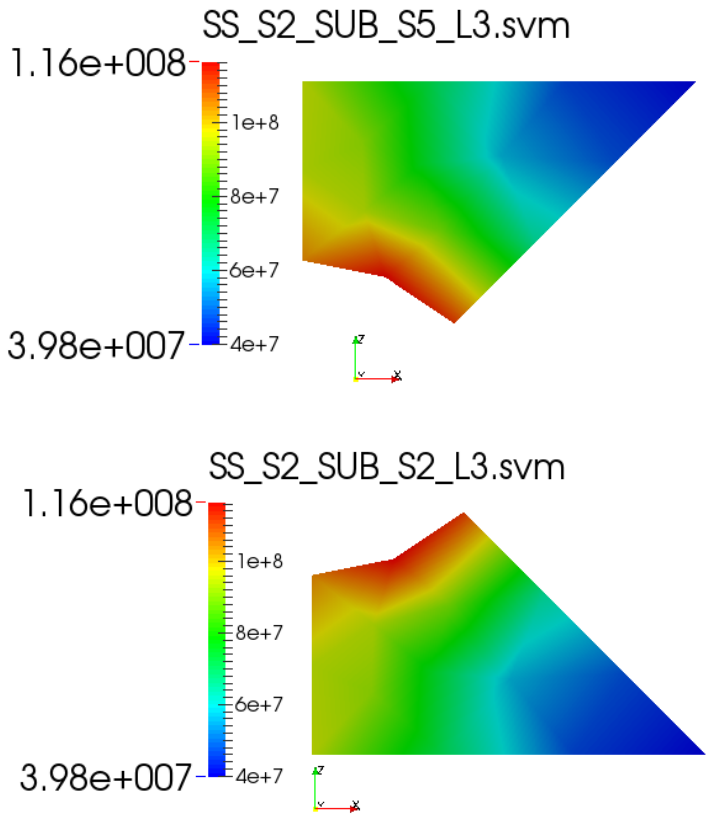

Figure 5.6: Largest value of von Mises stress in the whole construction and its location

Von Mises stress diagram is also presented for few other substructures in the construction. An example is a level two substructure S3, which is the only one that was not subjected to an additional level substructuring. This stress diagram is presented to point at one important conclusion derived from the analysis. Diagram is presented in Figure 5.7.

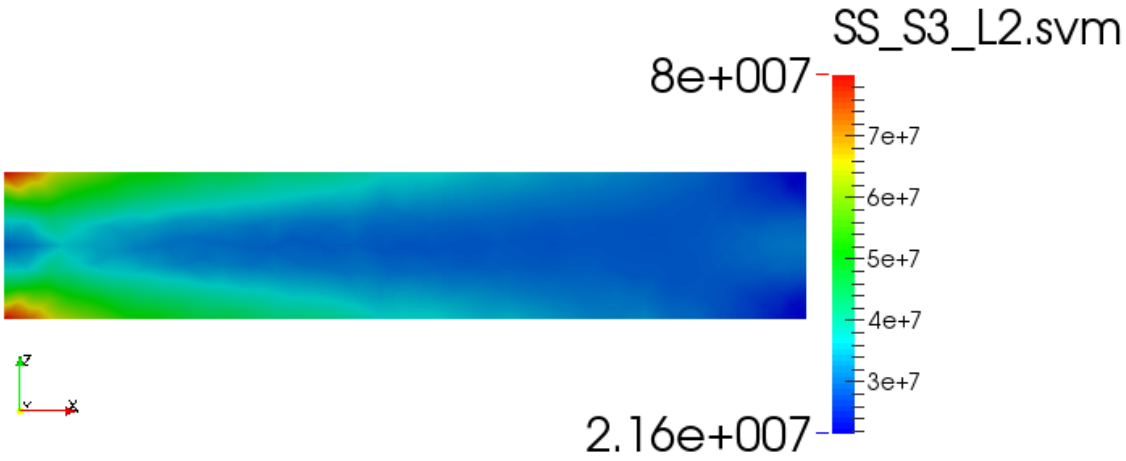

Figure 5.7: Von Mises stress for level two substructure S3

So, we can see from this diagram that the stress concentration shows up at two sections near the fixed support and the left edge. The reason that this is happening is first because we constrained the displacements perpendicular to the plate's plane, and now since stress is acting on the left edge due to the support, the plate tries to resist the bending and would want to contract. But since the bending load acts on the right edge, the plate would want to expand, and it can't because of the fixed support! And that is why the stress concentration shows up. In practice, this situation can be solved by not to constrain all the displacement components, and to allow rotation around the axis perpendicular to the plate's plane.

Because of the fixed support on the left edge there will be no displacements on the substructure S4, nor stresses or strains.

On the opposite side of the construction though, the situation is quite different. It can be compared with the stress diagram for substructure S4 from Chapter 4.2.1.

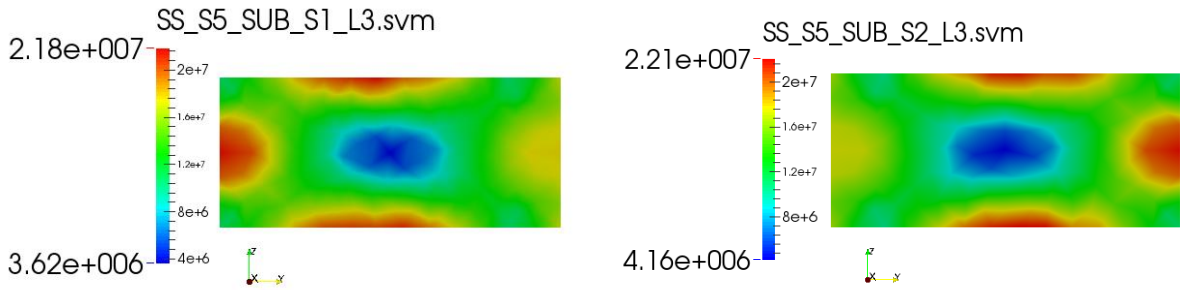

Figure 5.8: Stress distribution for the level two substructure S5 and its appropriate substructures

Figure 5.8 shows the von Mises stress distribution for the level two substructure S5 i.e. its appropriate level three substructures. We can see how the stress on the interface with the bulkhead's plate is considerably lower, again because the bulkhead takes over part of this load on both sides.

It is also important to note here that the largest value of the displacement in the negative *z* direction due to bending, for the whole construction, is in the node number 251 which is located at the bottom edge of the substructure *S1_L3* from Figure 5.8. The value of the displacement is $d = -0.008959\,m$. Along the whole right edge of the cover where the load is applied, displacements in the *z* direction are around 8 *mm*.

Finally, the stress diagram for a level two substructure S6 is presented in Figure 5.9. It is interesting to note that on the edges of substructures near the support where they go outwards, stress concentration also shows up. On the outer line following the ribs (left side of the substructures *S3_L3* and *S4_L3* from the lower figure) there is also an increased amount of stress acting on the ribs.

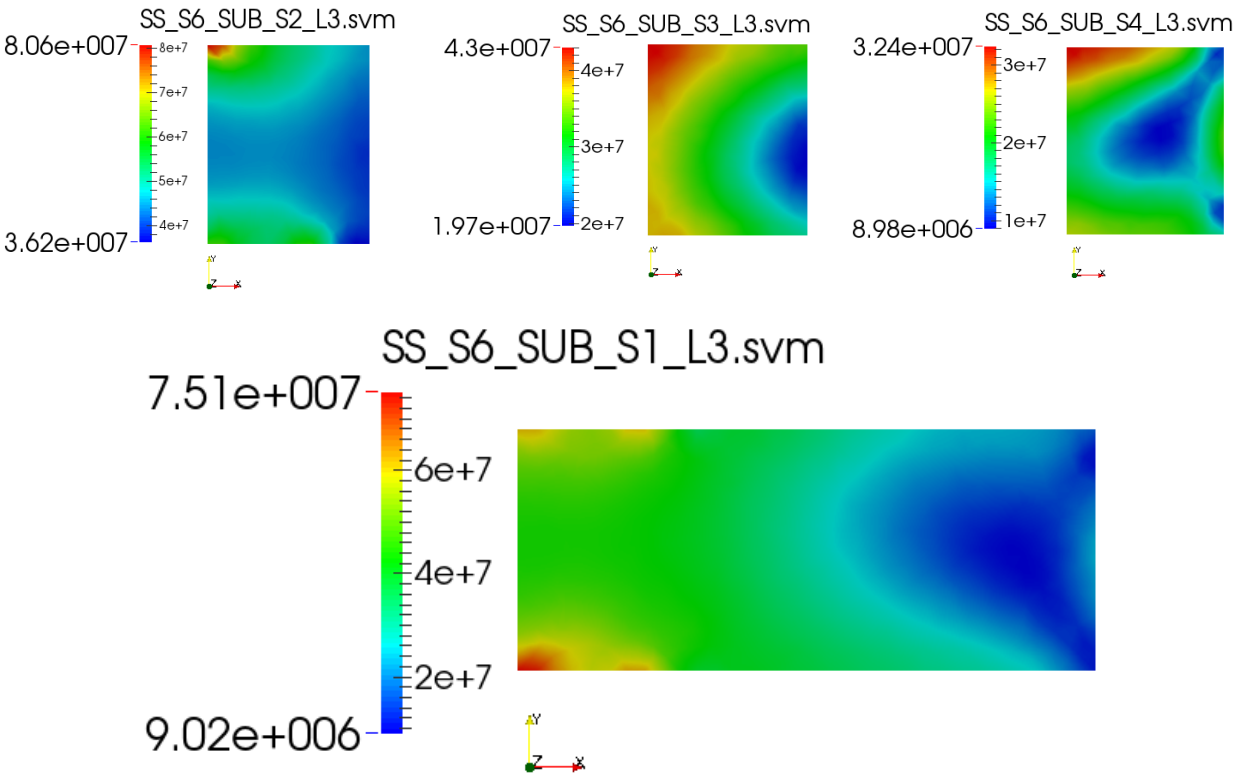

Figure 5.9: Stress distribution for the level two substructure S6 and its appropriate level three substructures

Solutions for the stress concentration that showed up during this analysis are several. For these types of problems, they often include reinforcements in the form of bars which are added to the critical parts of the construction. This type of solution is often applied even in the pre – processing FEM analysis, during mesh generation where simple bar elements are added. They contribute to the increase of stiffness in these sections which in turn decreases the stress concentration.

With this final step, the analysis of a three – dimensional ship cover is performed through an application of the multilevel substructuring and superelements technique.

It should be noted that, even though this problem consisted of three levels, this same procedure can be performed with any desired number of levels. Additionally, for a specific problem being solved, we can expand this analysis by making a final observation that, since the ship cover consisted of a three – dimensional structure whose outer edges were practically six

sides of a hexahedron, this analysis can be expanded by its adaptability to be used to construct and analyse whole tween decks with any number of bulkheads and different geometry that we can have. We can basically construct the outer layer of the cover and then insert as many bulkheads as we want.

All the programming codes for the applied procedure of performing a stress analysis of a three – dimensional ship cover with one bulkhead are given in the Appendix E.

6. Manipulating with large data structures

In the analysis process performed in the previous chapter, we had a total of 32 substructures on the lowest substructuring level, which in addition to the one level two substructure that was not disassembled furthermore, makes the total number of the disassembled components in our system to 33. This means that, following all the procedures and steps described in Chapter 3, while dealing with complex engineering problems like this one and through this technique, data transfer and data collection in such analysis can be very comprehensive.

For the reasons stated, data manipulation must be introduced. Not just that we are obliged to design our data transfer processes as more efficient as we can, we are also typically intertwined with today's more and more urging need to automate our design processes, if possible. To be able to perform that, even at the very beginning of our problem analysis, we need to cleverly design all these data structures and develop procedures for directing and especially sorting them in a way that the efficient processing design through substructuring procedure can be accomplished.

Figure 6.1 collects the lowest level visual data from the analysis performed in the previous chapter at one place.

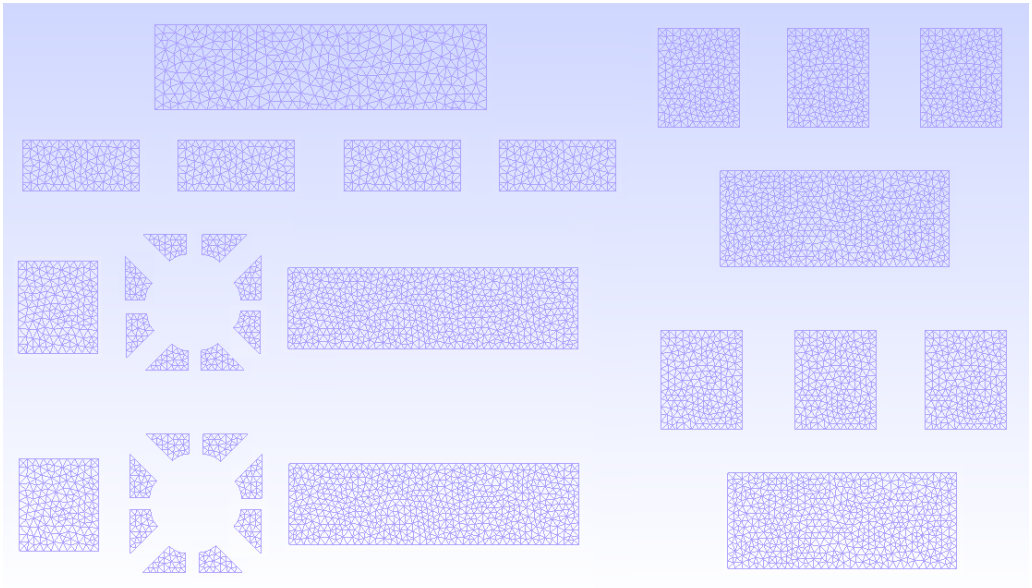

Figure 6.1: Level three visual data for the ship cover analysis

So as we can see, this extremely large number of nodes and elements, very large number of substructures, and in the end, substructures on same levels which are often symmetric and can be mirrored during the processing phase can make the whole managing and sorting of this data quite confusing and demanding.

To be able to efficiently deal with such large amount of data, some simple steps need to be followed to retain consistency and facilitate the analysis to ourselves and the experts we are collaborating with at the end. They can be broken down to two steps, or maybe two intuitive advices to follow:

1) Always sort the data in a repetitive way and utilize symmetry
2) Whenever possible, use these repetitive procedures and previously performed design to automate the solution processing

The example of the application of our first advice is presented in Figure 6.2, where data sorting and manipulation for the practical application from Chapter 5 is reflected in the multilayered hierarchical breakdown of appropriate data structures and data manipulation procedures. Here for the problem analysis from Chapter 5 we can see the whole processing data tree developed, which utilises a practical way to sort the substructures data and to be able to retrieve it without confusion any time later.

Other direct implications of this practical design on data sorting and data manipulation can be viewed from the code for obtaining the global level two stiffness matrices from all the condensed ones, calculated from the generated meshes during the first phase at the lowest substructuring level. The example of this code was already presented in Chapter 4.1.2 in the basic programming example of global – local analysis consisted of only two level two substructures. The code is presented at page ## for the level two substructure of the ship cover analysis from Chapter 5.

This code basically writes the formatted data to a string which we created for the substructures in a similar fashion and sorted it, and then loops over all substructures and scatters the local data from the condensed stiffness matrices to the global (level two in this example) stiffness matrix. If we hadn't designed and sorted our data in a way that makes this possible, we wouldn't be able to utilize this possibility of automating the process for obtaining the global stiffness matrix.

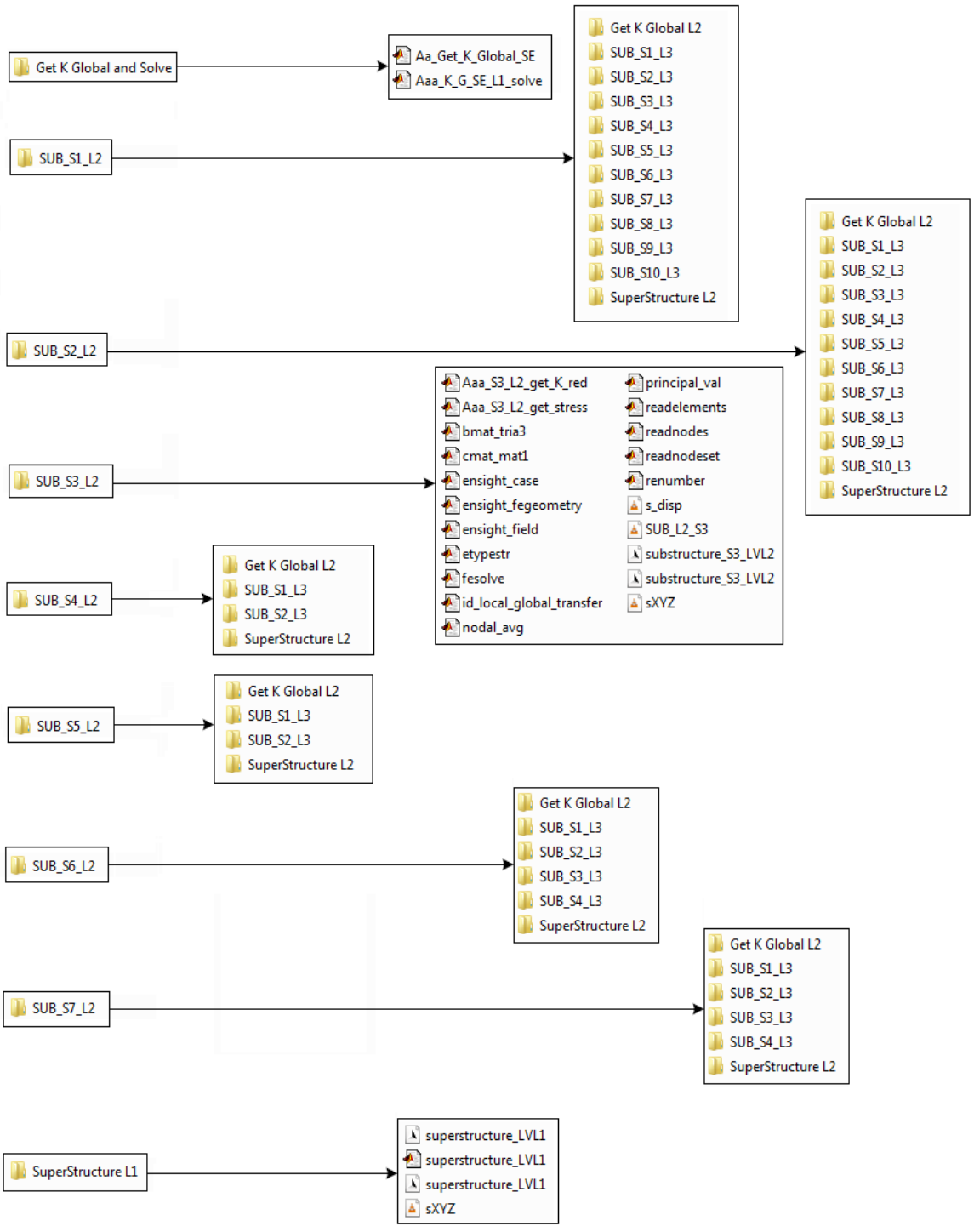

Figure 6.2: Data sorting repetition in a multilayered hierarchical substructuring procedure for the ship cover analysis from Chapter 5

Stated implication of practical design on data sorting and data manipulation is presented in the code below.

```
load('sXYZ_L2');

nn_global=size(sXYZ_L2,1);
ndof_global=2*nn_global;

KG_L2_S1=zeros(ndof_global,ndof_global); % or through sparse
nSS=10; % number of substructures

for ii=1:nSS

   substr_ii=sprintf('SUB_L3_S%d',ii);
   load (substr_ii,'K_red','id_local_global_L2','NEQ')

   sctr(1:2:NEQ)=2*id_local_global_L2-1;
   sctr(2:2:NEQ)=2*id_local_global_L2;

   KG_L2_S1(sctr,sctr)=KG_L2_S1(sctr,sctr)+K_red;

   clear sctr
   clear K_red
   clear id_local_global_L2
   clear NEQ
end

% transfer of data from L2 to L1 (the above procedure is from L3 to L2):
load('sXYZ');
id_local_global=id_local_global_transfer(sXYZ_L2, sXYZ);

NODE_S = size(sXYZ_L2,1); %number of nodes * ndof per node
NEQ = NODE_S * 2;

disp('Nodal vector for substructure S1 (LEVEL 2 ---> 1):')
disp(id_local_global)

save('SUB_L2_S1.mat','KG_L2_S1','id_local_global','NEQ')
```

This chapter covers the basic concept in manipulation with large data structures performed in this thesis. Data manipulation procedures performed here can be extended in several directions regarding further improvement of efficiency in recognizing patterns and routing the data without the need for the one single data component to be copied at several locations inside the data structure. Solutions for such described manipulations demand high level advanced computational techniques such as “data mining” [5], and largely exceed the range of research field covered in this thesis.

7. Conclusion

Projecting and designing complex constructions and developing procedures oriented to multilayered – hierarchical substructuring is a demanding and challenging task for every engineer. Substructuring and superelements technique which has been implemented with classical finite element method to solve various range of problems in stress analysis of thin – walled structures, performed in this thesis, showed that for these types of problems solutions obtained mostly do not differ in accuracy compared to the conventional finite element method and the theoretical results. This is a consequence of the fact that in static analysis the theory used for superelement processing is exact and not based on any assumptions. Reducing the system using the static condensation procedure yields several benefits, as the condensed stiffness matrices often dramatically reduce the total number of dofs which are entering the system of equations needed to be solved for global displacements. The other advantage, clearly shown is the efficient usage of symmetry, which often saves considerable amount of time for preparing the complete model. To follow a technical approach, maybe one of the most important implications and advantages of using this technique is the fact that by implying the boundary conditions to the global model only, we can perform stress analysis of local models independently of each other, without the need of implementing the actual boundary conditions to the local model. This is generally considered a good thing because the whole analysis is that way much less sensitive to random errors which often happen in practice due to potential misinterpretation of sets of boundary conditions that would otherwise needed to be set on every substructure locally.

Some of the results of the analysis performed showed very small deviations from the theoretical ones and the ones obtained through classical FEM, the reason for which can be traced to slight differences in discretization and in the fact that inverting a matrix numerically typically introduces an error, although this error is often hard to get within the order of magnitude of the computer's precision limit.

This procedure demands usage and manipulation with large data structures that build up during the execution of provided steps, and it is often not trivial to find innovative solutions regarding data manipulation for complex problems and multiple levels of substructuring.

Performed stress analysis of a three – dimensional ship cover with one bulkhead showed maximum values of stress along the boundary of the hole in the centrally located bulkhead. Ribs constructed at the top and bottom plate of the cover are also subjected to a slightly increased value of von Mises stress. Solutions for these sections most often include reinforcements in the form of bars which are generally added to such critical parts in the construction. This solution can be applied in the pre – processing phase of the analysis also, as a certain design failure test, during mesh generation where simple bar elements are added to critical sections, as they contribute to the increase of stiffness which in turn lowers the stress concentration.

The procedures developed in this thesis have clear potential for further development and application to even more complex problems and models in stress analysis of not just thin – walled structures, since the code developed is in a non – negligible measure generic, and can be expanded and applied furthermore to a range of problems in structural analysis in general. One of the future researches can aim towards the dynamic substructuring, frontal technique and component mode synthesis, since these methods are widely used in engineering design of complex structures and have space for further development through the utilization of substructuring.

Literature

[1] Finite Element Method Laboratory, open source finite element method code, https://bitbucket.org/jfchessa/femlab

[2] C. Geuzaine and J.-F. Remacle: "Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities", International Journal for Numerical Methods in Engineering 79(11), pp. 1309-1331, 2009.

[3] Carlos A. Felippa: "Introduction to finite element methods", Department of Aerospace Engineering Sciences and Center for Aerospace Structures, University of Colorado at Boulder, Fall 2004.

[4] Paraview Visualization Toolkit, http://www.paraview.org/

[5] Mohammed J. Zaki, Wagner Meira Jr.: "Data Mining and Analysis – Fundamental Concepts and Algorithms", Cambridge University Press, 2014.

[6] Prof. dr. Željan Lozina: "Finite Element Method (lecture notes)", Faculty of Electrical Engineering, Mechanical Engineering and Naval Architecture", Split, 2014.

[7] MSC Nastran Superelement User Guide, MSC. Software Corporation, USA, 2001.

[8] Viktor Larsson: "Analysis of holes and spot weld joints using sub models and superelements" (master thesis), Lulea University of Technology, 2010.

[9] Jack Chessa: "Programming the Finite Element Method with Matlab", Northwestern University, USA, 2002.

[10] Dr. Nazri Kamsah: "Two – Dimensional Problems Using CST Elements", (lecture notes), University of New Hampshire, USA, 2001.

[11] Prof. Gilbert Strang: Computational Science and Engineering I, Massachusetts Institute of Technology: MIT OpenCouseWare, as taught in Fall 2008, http://ocw.mit.edu

Appendix

Appendix A: Programming FEM in Matlab

The goal of this addition to the thesis is to give a brief overview and direction in the writing of finite element code using Matlab, with attention put mostly on the implementation. A test example finite element code for analysing static linear elastic problems written in Matlab is presented in the Appendix C, to illustrate how to program the finite element method, with the given flow diagram, data flow of variables and Matlab script files used. The Matlab script files from test examples and all their supporting files are listed in Appendix D and most of them can be found at Bit Bucket repository of professor dr. sc. Jack Chessa (University of Texas) called FEMLAB [1]. For the purposes of this thesis, the original code is slightly altered and adapted, regarding examples and problems I had to model, and in that personalized form it is given in the Appendix.

Notation

For clarity we adopt the following notation: the bold italics font $\boldsymbol{v}$ denotes a vector quantity of dimension equal to the spatial dimension of the problem *i.e.* the displacement or velocity at a point, the bold non-italicized font $\mathbf{d}$ denotes a vector or matrix which is of dimension of the number of unknowns in the discrete system *i.e.* a system matrix like the stiffness matrix, an uppercase subscript denotes a node number whereas a lowercase subscript in general denotes a vector component along a Cartesian unit vector. So, if $\mathbf{d}$ is the system vector of nodal unknowns, $\boldsymbol{u}_I$ is a displacement vector of node I and u_{Ii} is the component of the displacement at node I in the i direction, or $\boldsymbol{u}_I \cdot \boldsymbol{e}_i$. Often Matlab syntax will be intermixed with mathematical notation which hopefully adds clarity to the explanation. The typewriter font, **font**, is used to indicate that Matlab syntax is being employed.

Writing Matlab Programs

The Matlab programming language is useful in illustrating how to program the finite element method due to the fact it allows one to very quickly code numerical methods and has a vast

predefined mathematical library. This is also due to the fact that matrix (sparse and dense), vector and many linear algebra tools are already defined and the developer can focus entirely on the implementation of the algorithm not defining these data structures. The extensive mathematics and graphics functions further free the developer from the drudgery of developing these functions themselves or finding equivalent pre-existing libraries. A simple two dimensional finite element program in Matlab need only be a few hundred lines of code whereas in Fortran or C++ one might need a few thousand.

Although the Matlab programming language is very complete with respect to its mathematical functions there are a few finite element specific tasks that are helpful to develop as separate functions. These have been programmed and are available at the previously mentioned website.

As usual there is a trade off to this ease of development. Since Matlab is an interpretive language, each line of code is interpreted by the Matlab command line interpreter and executed sequentially at run time. As a consequence, the run times can be much greater than that of compiled programming languages like Fortran or C++. It should be noted that the built-in Matlab functions are already compiled and are extremely efficient and should be used as much as possible. Keeping this slow down due to the interpretive nature of Matlab in mind, one programming construct that should be avoided at all costs is the for loop, especially nested for loops since these can make Matlab programs run time orders of magnitude longer than may be needed. Often for loops can be eliminated using Matlab's vectorized addressing. For example, the following Matlab code which sets the row and column of a matrix **A** to zero and puts one on the diagonal

```
for i=1:size(A,2)
  A(n,i)=0;
end
for i=1:size(A,1)
  A(i,n)=0;
end
A(n,n)=1;
```

should never be used since the following code

```
A(:,n)=0;
A(:,n)=0;
```

A(n,n)=0;

does the same in three interpreted lines as opposed to $nr + nc + 1$ interpreted lines, where **A** is an $nr \times nc$ dimensional matrix. One can easily see that this can quickly add significant overhead when dealing with large systems (as is often the case with finite element codes). Sometimes for loops are unavoidable, but it is surprising how few times this is the case. It is suggested that after developing a Matlab program, one go back and see how/if they can eliminate any of the for loops. With practice this will become second nature.

Sections of a Typical Finite Element Program

A typical finite element program consists of the following sections:

1. Pre-processing section
2. Processing section
3. Post-processing section

In the pre-processing section the data and structures that define the particular problem statement are defined. These include the finite element discretization, material properties, solution parameters *etc*. The processing section is where the finite element objects *i.e.* stiffness matrices, force vectors *etc.* are computed, boundary conditions are enforced and the system is solved. The post-processing section is where the results from the processing section are analyzed. Here stresses may be calculated and data might be visualized. In this chapter we will primarily be concerned with the processing section, since we will be using Matlab mostly for the purposes of solving the problems in stress analysis through finite element code. Many pre and post-processing operations are already programmed in Matlab and are included in the online reference; if interested one can either look directly at the Matlab script files or type **help** *'function name'* at the Matlab command line to get further information on how to use these functions.

Finite Element Data Structures in Matlab

Here we discuss the data structures used in the finite element method and specifically those that are implemented in the test example code. These are somewhat arbitrary in that one can imagine numerous ways to store the data for a finite element program, but we attempt to use structures that are most eligible and conducive to Matlab. The design of these data structures may depend on the programming language used, but usually it is not significantly different from those outlined here.

Nodal Coordinate Matrix

Since we are programming the finite element method it is not unexpected that we need some way of representing the element discretization of the domain. To do so we define a set of nodes and a set of elements that connects these nodes in some way. The node coordinates are stored in the nodal coordinate matrix. This is simply a matrix of the nodal coordinates (*imagine that*). The dimension of this matrix is $nn \times sdim$ where nn is the number of nodes and *sdim* is the number of spatial dimensions of the problem. So, if we consider a nodal coordinate matrix **nodes,** the y-coordinate of the n^{th} node is **nodes(n,2)**. Figure 1 shows a simple finite element discretization. For this simple mesh the nodal coordinate matrix would be as follows

$$\mathbf{nodes} = \begin{bmatrix} 0.0 & 0.0 \\ 2.0 & 0.0 \\ 0.0 & 3.0 \\ 2.0 & 3.0 \\ 0.0 & 6.0 \\ 2.0 & 6.0 \end{bmatrix}. \qquad \text{(A.1)}$$

Element Connectivity Matrix

Element definitions are stored in the element connectivity matrix. This is a matrix of node numbers where each row in the matrix contains the connectivity of an element. So if we consider the connectivity matrix **elements** which describes e.g. a mesh of 4-node quadrilaterals, the 36th element is defined by the connectivity vector **elements(36,:)** which for example may be $\begin{bmatrix}36 & 42 & 13 & 14\end{bmatrix}$ or that the element connects nodes $36 \to 42 \to 13 \to 14$. So for the simple mesh in Figure 1 the element connectivity matrix is

$$\mathbf{elements} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 3 \\ 4 & 5 & 2 \\ 6 & 5 & 4 \end{bmatrix}. \tag{A.2}$$

Note that the element connectivities are all ordered in a counter-clockwise direction; if this is not done so some Jacobians will be negative and thus can cause the stiffness matrices to be singular (and obviously wrong).

Definition of Boundaries

In the finite element method boundary conditions are used to either form force vectors (natural or Neumann boundary conditions) or to specify the value of the unknown field on a boundary (essential or Dirichlet boundary conditions). In either case a definition of the boundary is needed. The most versatile way of accomplishing this is to keep a finite element discretization of the necessary boundaries. The dimension of this mesh will be one order less that the spatial dimension of the problem (*i.e.* a 2D boundary mesh for a 3D problem, 1D boundary mesh for a 2D problem *etc.*). Once again let's consider the simple mesh in Figure 1. Suppose we wish to apply a boundary condition on the right edge of the mesh, then the boundary mesh would be defined by the following element connectivity matrix of 2-node line elements

$$\textbf{rightEdge} = \begin{bmatrix} 2 & 4 \\ 4 & 6 \end{bmatrix}. \tag{A.3}$$

Note that the numbers in the boundary connectivity matrix refer to the same node coordinate matrix as do the numbers in the connectivity matrix of the interior elements. If we wish to apply the essential boundary conditions on this edge we need a list of the node numbers on the edge. This can be easily done in Matlab with the **unique** function.

```
nodesOnBoundary = unique(rightEdge);
```

This will set the vector **nodesOnBoundary** equal to $\begin{bmatrix} 2 & 4 & 6 \end{bmatrix}$. If we wish to form a force vector from a natural boundary condition on this edge we simply loop over the elements and integrate the force on the edge just as we would integrate any finite element operators on the domain interior *i.e.* the stiffness matrix **K**.

Dof Mapping

Ultimately for all finite element programs we solve a linear algebraic system of the form

$$\mathbf{K} \cdot \mathbf{d} = \mathbf{f} \tag{A.4}$$

for the vector **d**. The vector **d** contains the nodal unknowns that define the finite element approximation

$$u^h(x) = \sum_{I=1}^{nn} N_I(x) d_I \tag{A.5}$$

where $N_I(x)$ are the finite element shape functions, d_I are the nodal unknowns for the node I which may be scalar or vector quantities (if $u^h(x)$ is a scalar or vector) and nn is the number of nodes in the discretization. For scalar fields the location of the nodal unknowns in **d** is most obviously as follows

$$d_I = \mathbf{d}(\mathrm{I}), \tag{A.6}$$

but for vector fields the location of the nodal unknown d_{Ii}, where I refers to the node number and i refers to the component of the vector nodal unknown $\boldsymbol{d}_I$, there is some ambiguity. We need

to define a mapping from the node number and vector component to the index of the nodal unknown vector **d**. This mapping can be written as

$$f : \{I, i\} \rightarrow n \tag{A.7}$$

where f is the mapping, I is the node number, i is the component and n is the index in **d**. So the location of unknown u_{Ii} in **d** is as follows

$$u_{Ii} = \mathbf{d}_{f(I,i)}. \tag{A.8}$$

There are two common mappings used. The first is to alternate between each spatial component in the nodal unknown vector **d**. With this arrangement the nodal unknown vector **d** is of the form

$$\mathbf{d} = \begin{bmatrix} u_{1x} \\ u_{1y} \\ \vdots \\ u_{2x} \\ u_{2y} \\ \vdots \\ u_{nnx} \\ u_{nny} \\ \vdots \end{bmatrix} \tag{A.9}$$

where *nn* is again the number of nodes in the discretization. This mapping is

$$n = sdim(I-1) + i. \tag{A.10}$$

With this mapping the i component of the displacement at node I is located as follows in **d**

$$d_{Ii} = \text{d (sdim*(I-1)+i)}. \tag{A.11}$$

The other option is to group all the like components of the nodal unknowns in a contiguous portion of **d** as follows

$$\mathbf{d} = \begin{bmatrix} u_{1x} \\ u_{2x} \\ \vdots \\ u_{nx} \\ u_{1y} \\ u_{2y} \\ \vdots \end{bmatrix} \tag{A.12}$$

The mapping in this case is

$$n = (i-1)nn + I \tag{A.13}$$

So for this structure the *i* component of the displacement at node *I* is located in **d** at

$$d_{Ii} = \text{d ((i-1) *nn + I)}. \tag{A.14}$$

For reasons that will be appreciated when we shall discuss the *scattering* of element operators into system operators, we will adopt the latter dof mapping. It is important to be comfortable with these mappings since this is an operation that is performed regularly in any finite element code. Of course which ever mapping is chosen the stiffness matrix and force vectors should have the same structure.

Computation of Finite Element Operators

At the heart of the finite element program is the computation of finite element operators. For example, in a linear static code they would be the stiffness matrix

$$\mathbf{K} = \int_{\Omega} \mathbf{B}^T \mathbf{C}\, \mathbf{B}\; d\Omega \tag{A.15}$$

and the external force vector

$$\mathbf{f}^{ext} = \int_{\Gamma_t} \mathbf{N}\mathbf{t}\, d\Gamma\,. \tag{A.16}$$

The global operators are evaluated by looping over the elements in the discretization, integrating the operator over the element and then *scattering* the local element operator into the global operator. This procedure is written mathematically with the Assembly operator $\mathbb{A}$:

$$\mathbf{K} = \mathbb{A}_e \int_{\Omega^e} \mathbf{B}^{eT} \mathbf{C}\, \mathbf{B}^e d\Omega \tag{A.17}$$

Quadrature

The integration of an element operator is performed with an appropriate quadrature rule which depends on the element and the function being integrated. In general a quadrature rule is as follows

$$\int_{\xi=-1}^{\xi=1} f(\xi)d\xi = \sum_q f(\xi_q)W_q \tag{A.18}$$

where $f(\xi)$ is the function to be integrated, ξ_q are the quadrature points and W_q the quadrature weights. The function **quadrature** generates a vector of quadrature points and a vector of quadrature weights for a quadrature rule. The syntax of this function is as follows

```
[quadWeights,quadPoints] = quadrature(integrationOrder,
                elementType,dimensionOfQuadrature);
```

so an example quadrature loop to integrate the function $f = x^3$ on a triangular element would be as follows

```
[qPt,qWt]=quadrature(3,'TRIANGULAR',2);
for q=1:length(qWt)
   xi = qPt(q); % quadrature point
   % get the global coordinate x at the quadrature point xi
   % and the Jacobian at the quadrature point, jac
   ...
    f_int = f_int + x^3 * jac*qWt(q);
end
```

Operator "Scattering"

Once the element operator is computed it needs to be scattered into the global operator. An illustration of scattering of an element force vector into a global force vector is shown in Figure 2. The scattering is dependent on the element connectivity and the dof mapping chosen. The following code performs the scatter indicated in Figure A.1:

```
elemConn = element(e,:);                    % element connectivity
enn = length(elemConn);
for I=1:enn;                                % loop over element nodes
   for i=1:2                                % loop over spatial dimensions
      Ii=nn*(i-1)+sctr(I);                  % dof map
      f(Ii) = f(Ii) + f((i-1)*enn+I);
   end
end
```

but uses a nested for loop (bad). This is an even more egregious act considering the fact that it occurs within an element loop so this can really slow down the execution time of the program (by orders of magnitude in many cases). And it gets even worse when scattering a matrix operator (stiffness matrix) since we will have four nested for loops. Fortunately, Matlab allows for an easy solution; the following code performs exactly the same scattering as it is done in the above code but without any for loops, so the execution time is much improved (not to mention that it is much more concise).

```
sctr = element(e,:);                        % element connectivity
sctrVct = [ sctr sctr+nn ];                 % vector scatter
f(sctrVct) = f(sctrVct) + fe;
```

To scatter an element stiffness matrix into a global stiffness matrix the following line does the trick

```
K(sctrVct,sctrVct) = K(sctrVct,sctrVct) + ke;
```

This terse array indexing of Matlab is a bit confusing at first but if one spends a bit of time getting used to it, it will become quite natural and useful.

Enforcement of Essential Boundary Conditions

The final issue before solving the linear algebraic system of finite element equations is the enforcement of essential boundary conditions. Typically this involves modifying the system

$$\mathbf{K} \cdot \mathbf{d} = \mathbf{f} \tag{A.19}$$

so that the essential boundary condition

$$d_n = \bar{d}_n \tag{A.20}$$

is satisfied while retaining the original finite element equations on the unconstrained dofs. In (A.20), the subscript *n* refers to the index of the vector **d**, not to a node number. An easy way to enforce (A.20) would be to modify n^{th} row of the **K** matrix so that

$$K_{nm} = \delta_{nm} \qquad \forall m \in \{1, 2, \ldots, N\} \tag{A.21}$$

where *N* is the dimension of **K,** and to set

$$f_n = \bar{d}_n . \tag{A.22}$$

This reduces the n^{th} equation of (A.19) to (A.20). Unfortunately, this destroys the symmetry of **K** which is a very important property for many efficient linear solvers. By modifying the n^{th} column of **K** as follows

$$K_{m,n} = \delta_{nm} \qquad \forall m \in \{1, 2, \ldots, N\}. \tag{A.23}$$

we can make the system symmetric. Of course this will modify every equation in (A.19) unless we modify the force vector **f**:

$$f_m = K_{mn} \bar{d}_n . \tag{A.24}$$

If we write the modified k^{th} equation in (A.19)

$$K_{k1}d_1 + K_{k2}d_2 + \ldots + K_{k(n-1)}d_{n-1} + K_{k(n+1)}d_{n+1} + \ldots + K_{kN}d_N = f_k - K_{kn}\bar{d}_n \qquad \text{(A.25)}$$

it can be seen that we have the same linear equations as in (A.19), but just with the internal force from the constrained dof. This procedure in Matlab is as follows

```
f = f - K(:,fixedDofs)*fixedDofValues;
K(:,fixedDofs) = 0;
K(fixedDofs,:) = 0;
K(fixedDofs,fixedDofs) = bcwt*speye(length(fixedDofs));
f(fixedDofs) = bcwt*fixedDofValues;
```

where **fixedDofs** is a vector of the indicies in **d** that are fixed, **fixedDofValues** is a vector of the values that **fixedDofs** are assigned to and **bcwt** is a weighing factor to retain the conditioning of the stiffness matrix (typically *bcwt* = *trace*(**K**)/*N*)

Hopefully this brief overview of programming simple finite element procedures with Matlab has helped bridge the gap between reading the theory of finite element method and sitting down and writing one's own finite element codes. Hence, examples in the Appendix can in future be used as a reference to understand the basics of finite element method programming.

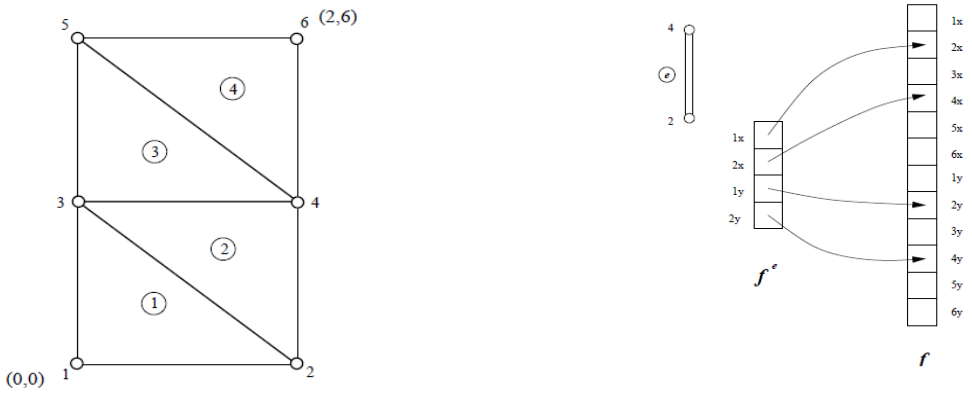

Figure A.1: A simple finite element mesh of triangular elements and an example of an element force vector $\mathbf{f}^e$ *scattered* into a global force vector **f**

Appendix B: Mesh generation – Gmsh

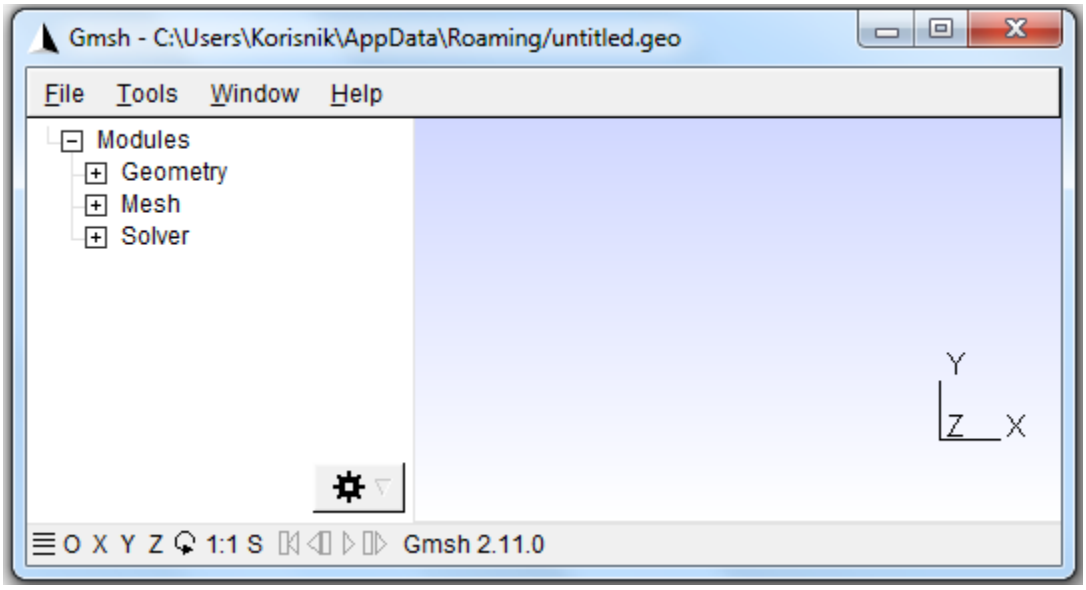

Figure B.1: Gmsh graphical user interface (gui)

Gmsh is an automatic three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. With Gmsh you can create or import 1D, 2D and 3D geometrical models, mesh them, launch external finite element solvers and visualize solutions. Gmsh can be used either as a stand-alone program (graphical or not) or as a C++ library.

For the requirements of this master thesis, Gmsh will serve mainly as a mesh generator used for creating the geometry of the problem and its mesh. As an "external" finite element solver we shall use Matlab programming language, i.e. by importing generated mesh into it and developing an algorithm which shall, with given input parameters and through finite element code, solve the problem and print the numerical results. For the purposes of the graphical visualization, the results will also be exported into Paraview, so the general problem solving approach structure shall be Gmsh + Matlab + Paraview.

Gmsh is built around four modules: geometry, mesh, solver and post-processing. As stated, here the emphasis shall be put on the first two modules: geometry and mesh. The specification of any input to these modules is done either interactively using the graphical user interface or in ASCII text files using Gmsh's own scripting language, or using both of them (gui

and ASCII) simultaneously. The latter can be very convenient because you can directly alter the code generated in ASCII text file, e.g. for the purpose of parametrization, correction of errors made in gui etc., since interactive actions generate language bits in the input files and vice versa. This makes it possible to automate all treatments, using loops, conditionals and external system calls.

The example of such source code which is automatically generated in ASCII text file by defining various geometrical entities in Gmsh's gui (slightly manually altered for the purpose of parametrization) is given in Figure B.2:

```
lc = DefineNumber[ 0.2, Name "Parameters/lc" ];
lc2 = DefineNumber[ 0.1, Name "Parameters/lc2" ];

Point(1) = {0, 0, 0, lc};
Point(2) = {6, 0, 0, lc};
Point(3) = {6, 1.1, 0, lc};
Point(4) = {0, 1.1, 0, lc};
Point(5) = {1.5, 0.55, 0, lc2};
Point(6) = {1.8, 0.55, 0, lc2};
Point(7) = {1.2, 0.55, 0, lc2};
Point(8) = {1.5, 0.85, 0, lc2};
Point(9) = {1.5, 0.25, 0, lc2};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};
Circle(5) = {6, 5, 7};
Circle(6) = {7, 5, 6};

Line Loop(7) = {2, 3, 4, 1};
Line Loop(8) = {5, 6};
Plane Surface(9) = {7, 8};
Physical Surface(10) = {9};
Physical Line(11) = {4};
Physical Line(12) = {2};

```

Figure B.2: Source code in ASCII text file which generates geometry in Gmsh for a simple plate with a hole problem

Here you can see that various geometrical entities are defined, such as points, lines, circle arcs, surfaces etc. Gmsh uses a boundary representation ("BRep") to describe geometries. Models are created in a bottom-up flow by successively defining points, oriented lines (line segments, circles, ellipses, splines, …), oriented surfaces (plane surfaces, ruled surfaces, triangulated surfaces, …) and volumes. Groups of geometrical entities, called "physical groups",

can also be defined based on these elementary geometrical entities, and such are physical surface 10 and physical lines 11 and 12 in the example above, respectively. The usage of physical groups is very important since with the help of these we can transfer the plain geometry to a 2D or 3D object with physical properties, which is essential when working with real-life engineering problems. These can be boundary conditions, fixed edges, applied external load, material properties, etc. Whenever we have some of these defined on a geometrical entity in our problem, it is essential that we assign a property of a physical group to it.

Gmsh's scripting language also allows all geometrical entities to be fully parametrized, which can be done either in gui or writing first two lines in the script in the example above. *lc* is essentially a parameter that represents characteristic length whose value can be assigned to a specific geometrical entity and using it we can achieve different finite element densities in the desired areas when meshing.

The result of such generated source code, seen from Gmsh graphical user interface, is shown in Figure B.3. It represents geometry of a plate with a hole, with point labels shown (.geo file).

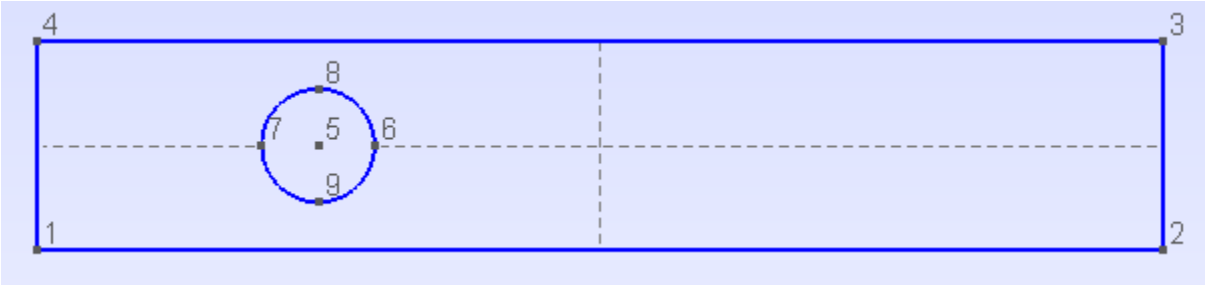

Figure B.3: Geometry of a plate with a hole in Gmsh graphical user interface

Meshing this geometry can be done fairly simple in the Gmsh gui, under the module "Mesh". Mesh module offers many possibilities such as defining order of the finite elements used (when using triangle elements – order 1 for CST, order 2 for LST, etc., refining the mesh, optimizing etc.

The mesh of the geometry from Figure B.3, using 2 characteristic element mesh sizes on different sectors (as from Figure B.2), is shown in Figure B.4:

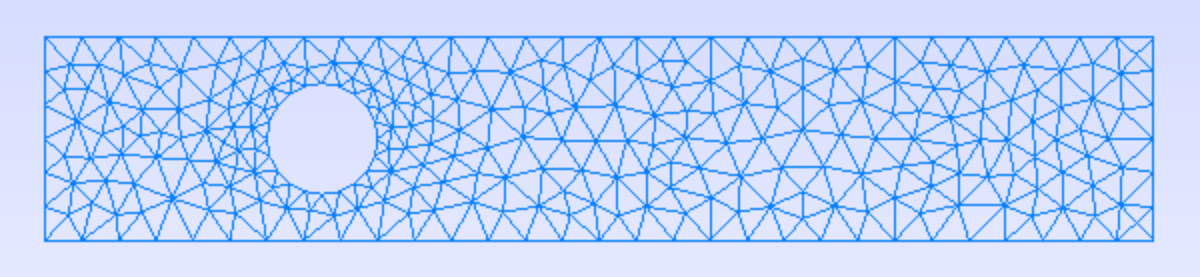

Figure B.4: Mesh of a plate with a hole in Gmsh graphical user interface

The mesh generation is performed in the same bottom-up flow as the geometry creation: lines are discretized first; the mesh of the lines is then used to mesh the surfaces; then the mesh of the surfaces is used to mesh the volumes. In this process, the mesh of an entity is only constrained by the mesh of its boundary.

The .msh file that describes the mesh from Figure B.4, when opened in text editor as ASCII text file, would contain fairly large amount of data, since we have large number of elements and nodes. So, for the purposes of simple explanation how a general .msh file looks like, we shall use much simpler mesh, shown in Figure B.5. The syntax in the .msh file is always the same, regardless of the number of elements and nodes used.

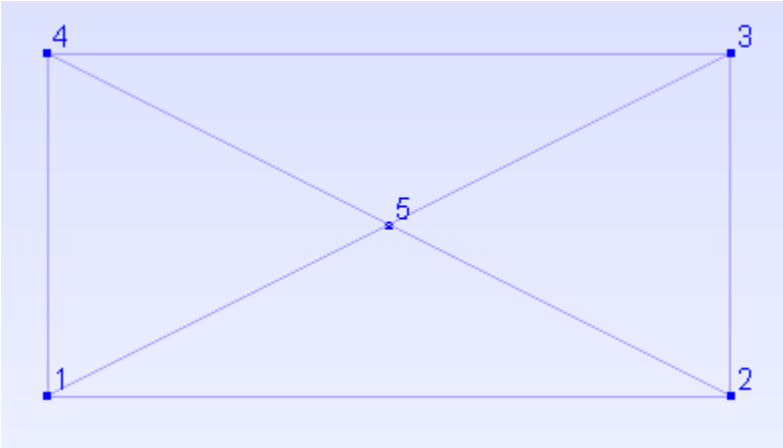

Figure B.5: Simple mesh of a 2D geometry plane stress problem with nodes and node labels shown

The mesh above was used to describe a simple 2D geometry plane stress problem with essential boundary conditions imposed on the left edge (e.g. pinned and roller support), and natural boundary conditions imposed on the right edge (applied load), according to Figure B.6.

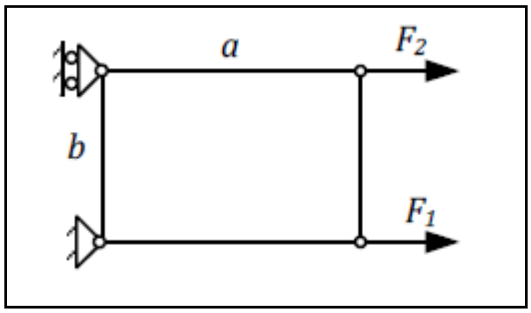

Figure B.6: 2D plane stress problem with essential and natural boundary conditions imposed

Although it may seem from Figure B.5 that we have just four triangular finite elements used in the mesh, when we open the mesh file in a text editor we can see that there are actually six finite elements generated in the mesh altogether, as shown in Figure B.7.

```
1   $MeshFormat
2   2.2 0 8
3   $EndMeshFormat
4
5   $Nodes
6   5
7   1 0 0 0
8   2 2 0 0
9   3 2 1 0
10  4 0 1 0
11  5 1 0.5 0
12  $EndNodes
13
14  $Elements
15  6
16  1 1 2 9 2 2 3
17  2 1 2 8 4 4 1
18  3 2 2 7 6 1 5 4
19  4 2 2 7 6 2 3 5
20  5 2 2 7 6 1 2 5
21  6 2 2 7 6 3 4 5
22  $EndElements
23
```

Figure B.7: Syntax in the .msh file which represents the mesh from Figure 5

Since we had boundary conditions imposed on the left and the right edge of a plate, we had to assign a property of a physical group (“physical line” in this case) to these edges during the geometry definition (making of .geo file). Because of that fact, and since Gmsh is an automatic finite element mesh generator, besides the four triangular elements it also created two

line elements consisting of 2 nodes each, which Gmsh uses to represent a property of a physical group on these edges where the boundary conditions are implied. That can be seen most clearly when looking at the syntax of the .msh file from Figure B.7, and the data it stores.

Every mesh file generated in Gmsh has the same general structure, and is divided in several sections (enclosed in $KEY and $ENDKEY pairs). At the very top we have the $MeshFormat which mostly varies from one software release to another. The next two fields are most important: $Nodes/$EndNodes defines the nodes and $Elements/$EndElements defines the elements.

The syntax is as follows:

$Nodes

number-of-nodes

node-number x-coord y-coord z-coord

...

$EndNodes

$Elements

number-of-elements

elm-number elm-type number-of-tags <tag> ... node-number-list

...

$EndElements

All the syntactic variables stand for integers except x-coord, y-coord and z-coord which stand for floating point values.

The elm-type value defines the geometrical type for the element, as follows:

1: 2-node line
2: 3-node triangle
3: 4-node quadrangle
4: 4-node tetrahedron

5: 8-node hexahedron
6: 6-node prism
7: 5-node pyramid
8: 3-node second order line (2 nodes associated with the vertices and 1 with the edge)
9: 6-node second order triangle (3 nodes associated with the vertices and 3 with the edges)
10: 9-node second order quadrangle (4 nodes associated with the vertices, 4 with the edges and 1 with the face)
11: 10-node second order tetrahedron (4 nodes associated with the vertices and 6 with the edges)
12: 27-node second order hexahedron (8 nodes associated with the vertices, 12 with the edges, 6 with the faces and 1 with the volume)
13: 18-node second order prism (6 nodes associated with the vertices, 9 with the edges and 3 with the quadrangular faces)
14: 14-node second order pyramid (5 nodes associated with the vertices, 8 with the edges and 1 with the quadrangular face)
15: 1-node point
16: 8-node second order quadrangle (4 nodes associated with the vertices and 4 with the edges)
17: 20-node second order hexahedron (8 nodes associated with the vertices and 12 with the edges)
18: 15-node second order prism (6 nodes associated with the vertices and 9 with the edges)
19: 13-node second order pyramid (5 nodes associated with the vertices and 8 with the edges)

The number-of-tags value gives the number of integer tags that follow for the n-th element. By default, the first tag is the number of the physical entity to which the element belongs; the second is the number of the elementary geometrical entity to which the element belongs; the third is the number of a mesh partition to which the element belongs. All tags must be positive integers, or zero. A zero tag is equivalent to no tag.

The node-number-list is essentially the connectivity of all the elements through the connection of their nodes.

To conclude, the nodes section contains data which consists of total number of nodes, their ordinal number, and the nodal coordinates of each node.

The elements section contains data which consists of total number of elements, their ordinal number, element type, total number of tags for the specific element (physical, geometrical and mesh partition tag), tag id (which can be verified and checked in the .geo file) for every tag defined in the mesh, and the connectivity of all the elements through the connection of their nodes. For triangular elements used in our example the order of nodes in Gmsh automatically always goes counterclockwise to ensure that the Jacobian is never less than zero.

So, for our example of a mesh from Figure B.5, we can clearly see from Figure B.7 that our mesh consists of total of five nodes with the given coordinates, and six elements from which the first two are 2-node line elements that have a total of two tags, first tag being the physical id (number of the physical entity to which the elements belongs), and the second one being the geometrical id (number of the elementary geometrical entity to which the elements belongs), and these are physical lines 9 and 8 respectively, and geometrical lines 2 and 4, respectively. The last two columns show the element connectivity, i.e. the first line element connects nodes 2 and 3, the second one connects nodes 4 and 1.

For the remaining four triangular elements the syntax goes the same, with a logical variation in the element connectivity, since for the triangular element connectivity is made from 3 nodes.

Source code of the .geo file which generates the geometry of this example is shown in Figure B.8. From it we can clearly see that the ids of geometrical and physical entities match with those in the .msh file.

```
lc = DefineNumber[ 2, Name "Parameters/lc" ];

Point(1) = {0, 0, 0, lc};
Point(2) = {2, 0, 0, lc};
Point(3) = {2, 1, 0, lc};
Point(4) = {0, 1, 0, lc};
Line(1) = {1, 2};
Line(2) = {2, 3};
Line(3) = {3, 4};
Line(4) = {4, 1};

Line Loop(5) = {1, 2, 3, 4};
Plane Surface(6) = {5};

Physical Surface(7) = {6};
Physical Line(8) = {4};
Physical Line(9) = {2};

```

Figure 8: Source code in .geo file which generates geometry for a 2D plate from Figure 6

Appendix C: Application of classical finite element procedures in 2D plane stress analysis problems

Here in this addition to the thesis we shall present the application of classical finite element procedures in solving 2D plane stress problems. Starting from the first simple test example with just two finite elements, and completing with a more realistic engineering problem of modelling a 2D plate with a non symmetrical hole subjected to an in-plane bending load, we shall incorporate finite element algorithm written in Matlab to these problems and demonstrate how conventional finite element modelling and programming can be used and applied to engineering problems in practice.

Examples will be solved using CST and LST elements separately, and then comparing the results from both. For the illustration and the concept idea, the first test example code will also be given here, together with a flow diagram of a complete modelling process, data flow of variables, and input/output of every function used, listed at the end.

Test example 1

Consider a problem presented at Figure 6.1, where $a = 2m$, $b = 1m$, $t = 0.01m$, $F_1 = 1kN$, $F_2 = 2kN$, $E = 210GPa$, $\nu = 0.33$.

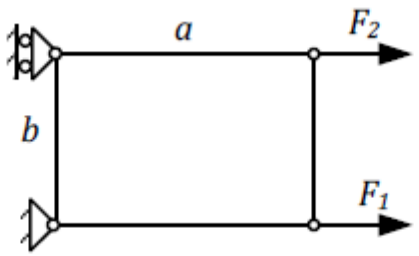

Figure C.1: Simple plane stress problem

This problem will be modelled using just two finite elements, as shown in Figure C.2 (CST example).

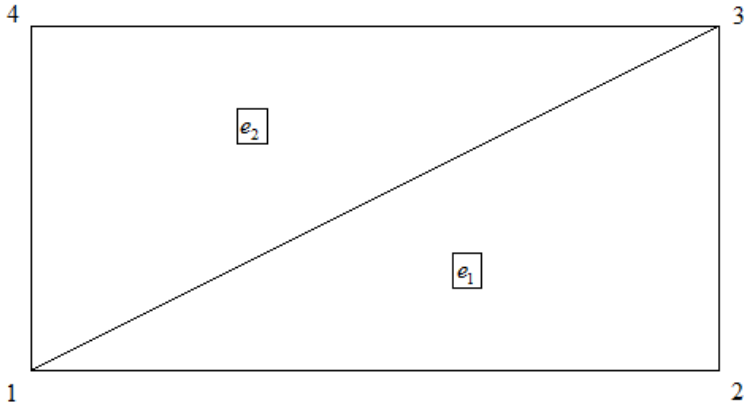

Figure C.2: Mesh of the problem from Figure C.1

a) Constant strain triangle (CST)

Flow diagram of a complete modelling process using CST element is presented in Figure C.3. The process is incorporated in the Matlab script file "*test_example_solve_2FE_CST.m*", given in the Appendix.

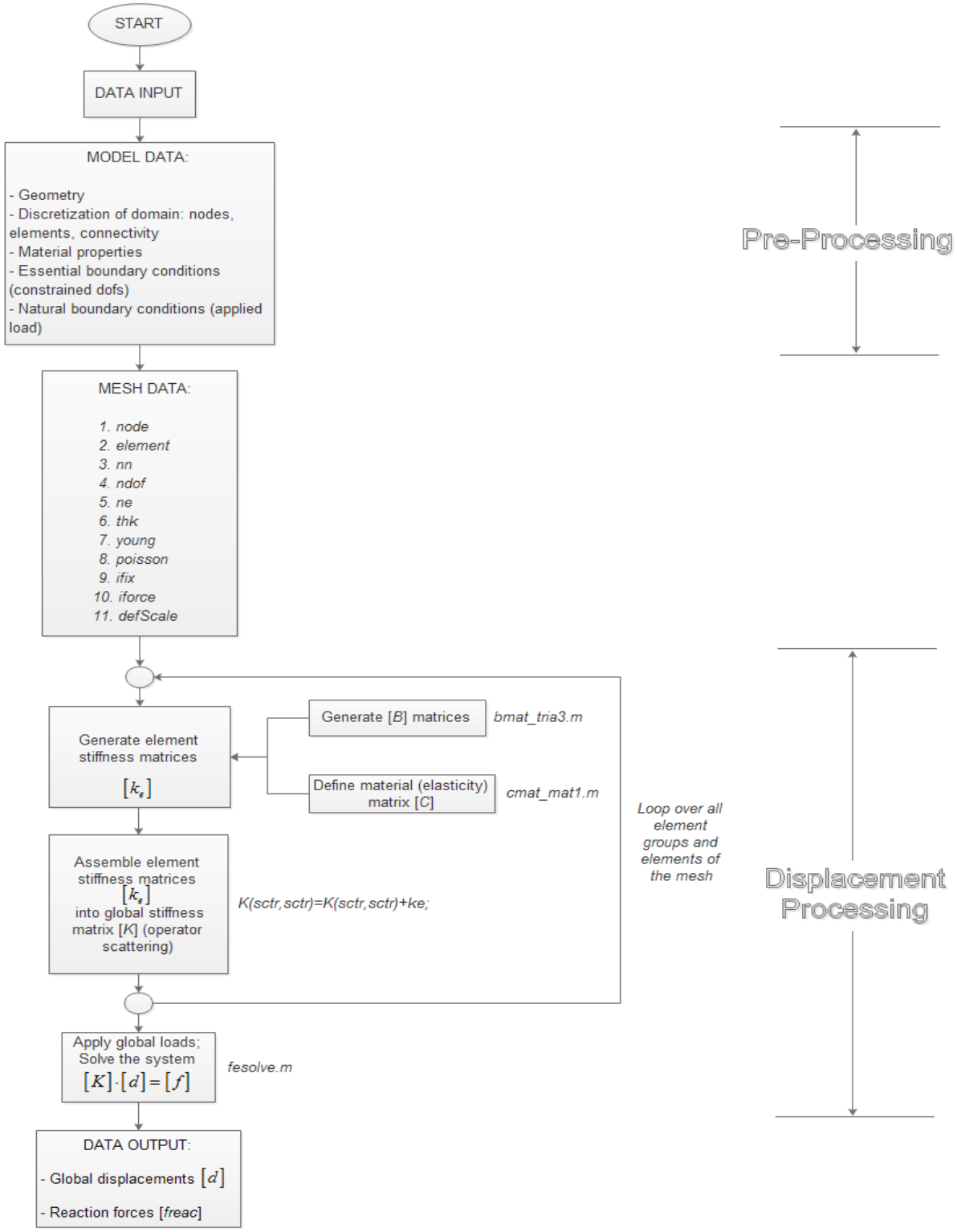

Figure C.3a: Pre-processing and displacement processing phase

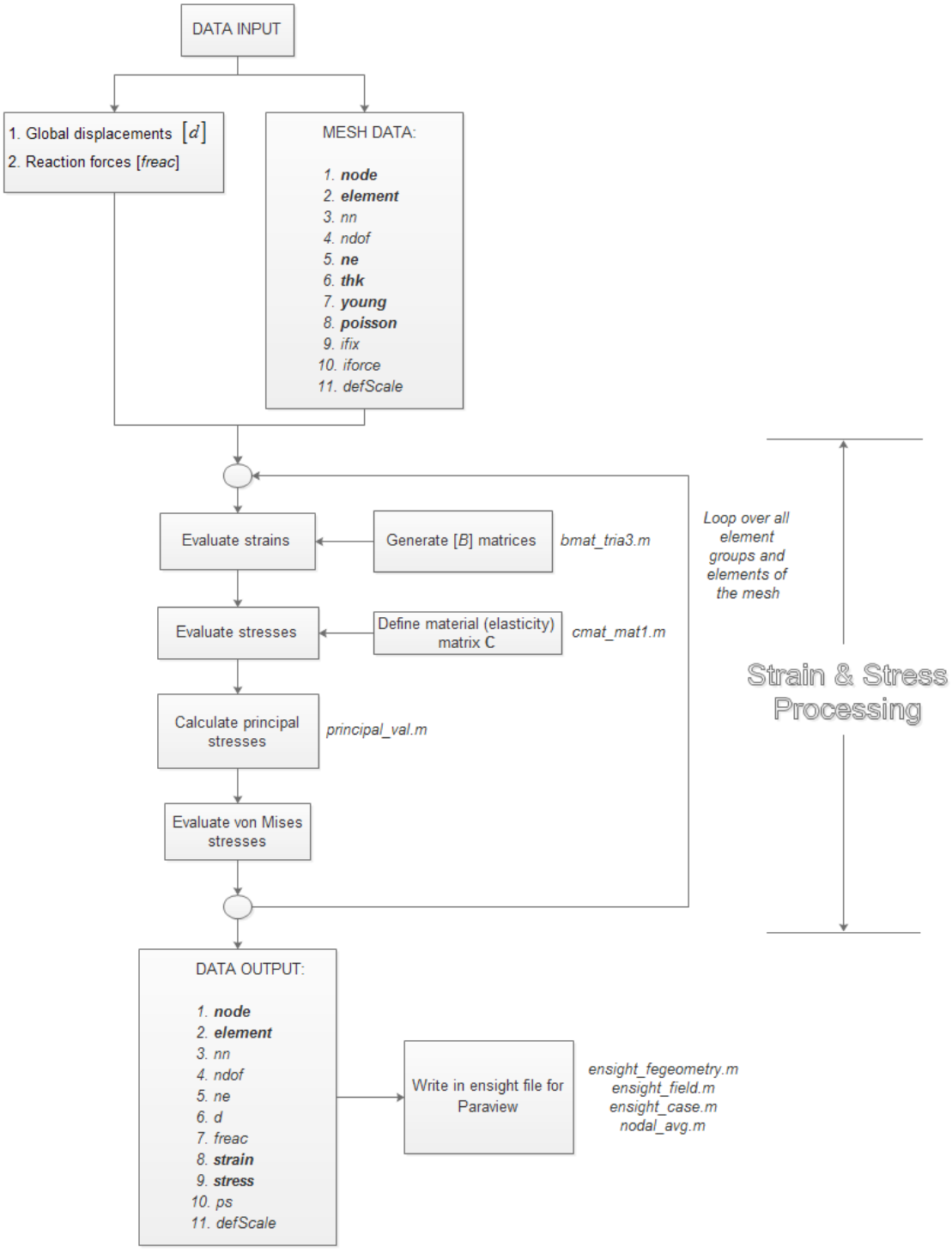

Figure C.3b: Strain & stress processing phase

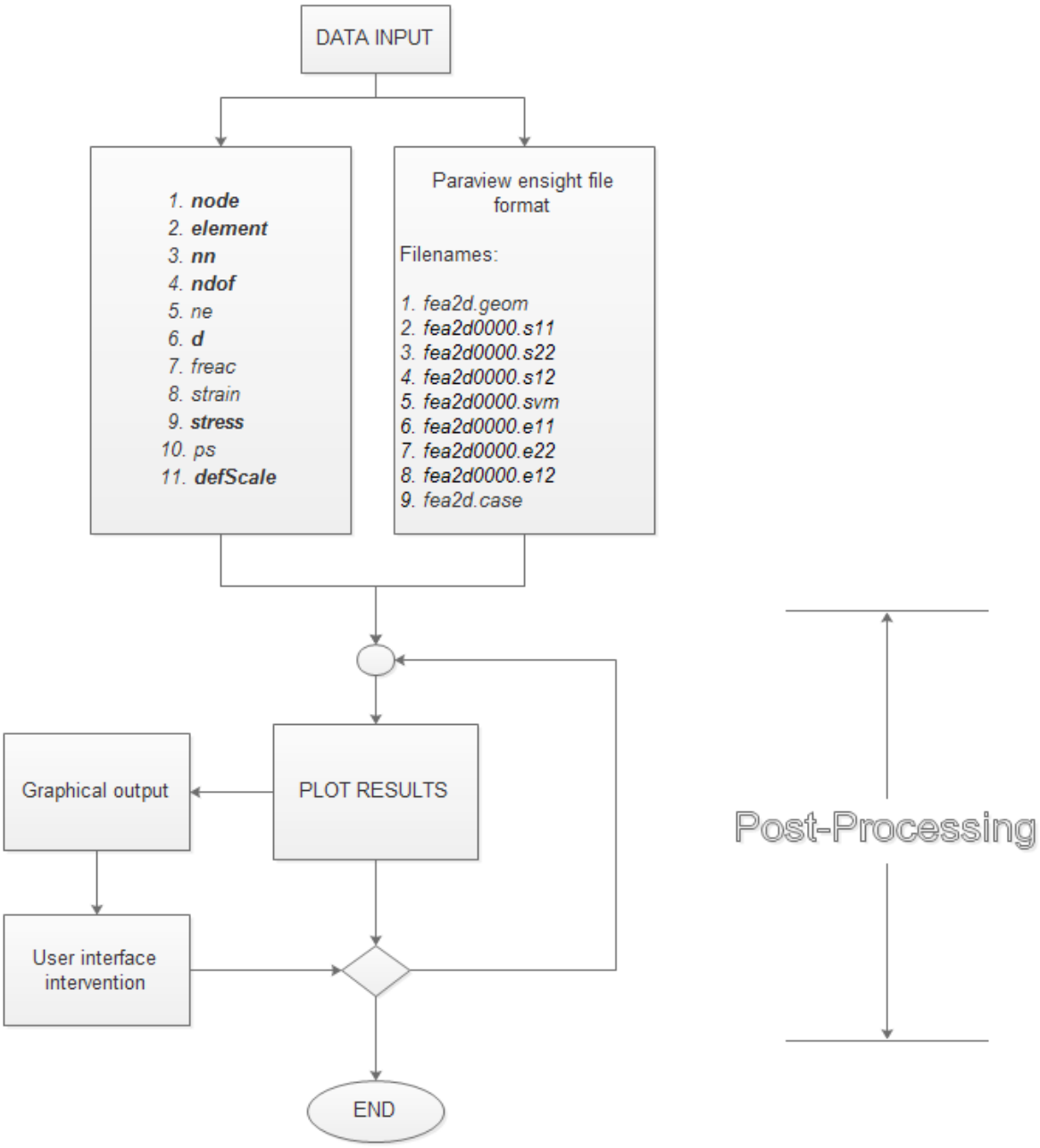

Figure C.3c: Post-processing phase

Data input and data output values for the displacement processing phase are given in Table C.1.

Displacement Processing Data					
Input			Output		
1.	*node*	0 0 2 0 2 1 0 1	1.	*d*	1.0e-005 * 0 0 0.2418 -0.1013 0.3296 -0.1350 0 -0.0606
2.	*element*	1 2 3 1 3 4			
3.	*nn*	4			
4	*ndof*	8			
5.	*ne*	2	2.	*freac*	1.0e+003 * -1.0000 -0.0000 -2.0000
6.	*thk*	0.0100			
7.	*young*	2.1000e+011			
8.	*poisson*	0.3300			
9.	*ifix*	1 2 7			
10.	*iforce*	3 1000 5 2000			
11.	*defScale*	50000			

Table C.1: Displacement processing input & output data

Data input and output values for the strain & stress processing phase are given in Table C.2.

Strain & Stress Processing Data					
Input			Output		
1.	*node*	0 0 2 0 2 1 0 1	1.	*node*	0 0 2 0 2 1 0 1
2.	*element*	1 2 3 1 3 4	2.	*element*	1 2 3 1 3 4
3.	*nn*	4	3.	*nn*	4
4.	*ndof*	8	4.	*ndof*	8
5.	*ne*	2	5.	*ne*	2

6.	*thk*	0.0100	6.	*d*	1.0e-005 * 0 0 0.2418 -0.1013 0.3296 -0.1350 0 -0.0606
7.	*young*	2.1000e+011	7.	*freac*	1.0e+003 * -1.0000 -0.0000 -2.0000
8.	*poisson*	0.3300	8.	*strain*	1.0e-005 * 0.1209 0.1648 -0.0337 -0.0606 0.0372 -0.0372
9.	*ifix*	1 2 7	9.	*stress*	1.0e+005 * 2.5872 3.4128 0.1468 -0.1468 0.2936 -0.2936 2.5679 3.5254
10.	*iforce*	3 1000 5 2000			
11.	*defScale*	50000	10.	*defScale*	50000
12.	*d*	1.0e-005 * 0 0 0.2418 -0.1013 0.3296 -0.1350 0 -0.0606			
13.	*freac*	1.0e+003 * -1.0000 -0.0000 -2.0000			

Table C.2: Strain & stress processing input & output data

Data input values and ensight files for the post-processing phase are given in Table C.3. Notice that there is no output (plotting and visualizing results with optional user interface intervention until end).

Post-processing Input Data				
Variables			Files	
1.	*node*	0 0 2 0 2 1 0 1	1.	*fea2d.geom*
2.	*element*	1 2 3 1 3 4	2.	*fea2d0000.s11*
3.	*nn*	4	3.	*fea2d0000.s22*
4.	*ndof*	8	4.	*fea2d0000.s12*
5.	*ne*	2	5.	*fea2d0000.svm*
6.	*d*	1.0e-005 * 0 0 0.2418 -0.1013 0.3296 -0.1350 0 -0.0606	6.	*fea2d0000.e11*
7.	*freac*	1.0e+003 * -1.0000 -0.0000 -2.0000	7.	*fea2d0000.e22*
8.	*strain*	1.0e-005 * 0.1209 0.1648 -0.0337 -0.0606 0.0372 -0.0372	8.	*fea2d0000.e12*
9.	*stress*	1.0e+005 * 2.5872 3.4128 0.1468 -0.1468 0.2936 -0.2936 2.5679 3.5254	9.	*fea2d.case*
10.	*defScale*	50000		

Table C.3: Post - processing input data

Flow diagram from Figure 6.3 which describes the modelling process in the Matlab script file "*test_example_solve_2FE_CST.m*" has a total of eight functions called. Each of these functions has a specific task, described in the flow diagram. Functions are listed below, together with their input and output.

test_example_solve_2FE_CST.m – functions with I/O		
FUNCTION	INPUT	OUTPUT
cmat_mat1.m	*young;* *poisson;* *formulation ('PSTRESS')*	*material stiffness matrix C for a linear isotropic elastic material*
bmat_tria3.m	*nodal coordinates of the element (not the same as variable "node")*	*strain – displacement matrix B;* *element area A*
fesolve.m	*global stiffness matrix K;* *global load vector fext;* *constrained degrees of freedom ifix*	*global displacements d;* *reaction forces in constrained dofs freac*
principal_val.m	*stress*	*principal stress ps*
ensight_fegeometry.m	*filename;* *node;* *element connectivity matrix;* *element type*	*Ensight Gold format file (.geom) with finite element geometry*
ensight_field.m	*filename;* *data (element or node);* *element type*	*Ensight Gold format grid data file*
ensight_case.m	*jobname;* *geomfile;* *times;* *scalarvar;* *vectorvar;* *tensorvar;* *scalarcell;* *vectorcell;* *tensorcell;*	*Ensight .case file*
nodal_avg.m	*element value;* *element connectivity matrix;* *node*	*nodal average values of element data nval*

Table C.4: Functions used in '*test_example_solve_2FE_CST.m*'

Result of this analysis for stresses, strains, displacements and reaction forces in constrained dofs, as from Table C.2, is given below. Note that the fourth component of stress in the table is the von Mises stress.

Plot of the von Mises stress with scaled displacements on the original mesh is presented in Figure C.4.

STRESS ($\times 10^5$)			STRAIN ($\times 10^{-5}$)		
Stress component	Element 1	Element 2	Strain component	Element1	Element 2
σ_x	2.5872	3.4128	ε_x	0.1209	0.1648
σ_y	0.1468	-0.1468	ε_y	-0.0337	-0.0606
τ_{xy}	0.2936	-0.2936	γ_{xy}	0.0372	-0.0372
σ_v	2.5679	3.5254			

Table C.5: Stress & strain results using CST element

DISPLACEMENT ($\times 10^{-5}$)			REACTION FORCE ($\times 10^3$)		
NODE	*x*	*y*	NODE	*x*	*y*
1.	0	0	1.	-1.0000	0
2.	0.2418	-0.1013	2.	0	0
3.	0.3296	-0.1350	3.	0	0
4.	0	-0.0606	4.	-2.0000	0

Table C.6: Displacement & reaction force results using CST element

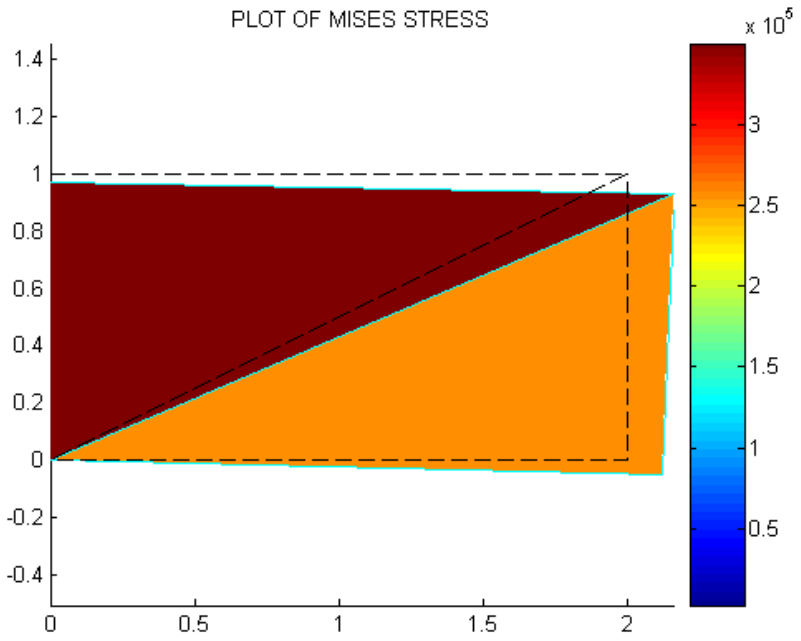

Figure C.4: Von Mises stress with scaled displacements on the original mesh using CST elements

a) Linear strain triangle (LST)

The process is incorporated in the Matlab script file "*test_example_solve_2FE_LST.m*", given in the Appendix.

Since an LST element has six nodes, we will have a total of nine nodes in the mesh model (three nodes on the diagonal intersect).

Flow diagram of a modelling process is basically the same as in Figure C.3, the only differences are:

- The function which generates $[B]$ matrix is now a new one, *bmat_tria6.m*, given in the Appendix
- We had to introduce a new function in post-processing section, *el_renumber.m*, whose code along with its description is given below.

```
function element2 = el_renumber(element)

% function element2 = el_renumber(element)
%
% since for an LST element the nodes are always numbered first at vertices
% and then at the sides, we need to renumber that sequence for the purposes
% of plotting the mesh and the output stress in Matlab correctly.
%
% function el_renumber.m renumbers the element connectivity matrix on an
% LST element so that the neighboring nodes are numbered consecutively from
% 1 to 6, going counterclockwise. It is used just for the purposes of
% plotting the output mesh geometry correctly.

nel=size(element,1); % number of elements

for i=1:nel
  element2(i,1)=element(i,1);
  element2(i,3)=element(i,2);
  element2(i,5)=element(i,3);
  element2(i,2)=element(i,4);
  element2(i,4)=element(i,5);
  element2(i,6)=element(i,6);
end
```

So in the input data in the post-processing section, along with variable *element,* we also have variable *element2* which the function *el_renumber.m* returnes, and which we use to plot the von Mises stress with scaled displacements on the original mesh generated in Matlab.

Results are presented in Table C.7 and Table C.8. The values for stresses and strains, as for displacements and reaction forces are slightly different due to a new middle node on the left and right edge of a plate, to which we applied essential and natural boundary conditions. Consecutively, the y – displacement in the top left corner of the plate is smaller, due to a closer fixed support of the middle node on the left edge.

STRESS ($\times 10^5$)			STRAIN ($\times 10^{-5}$)		
Stress component	Element 1	Element 2	Strain component	Element1	Element 2
σ_x	1.7643	2.5062	ε_x	0.0845	0.1190
σ_y	-0.0301	0.0214	ε_y	-0.0292	-0.0384
τ_{xy}	0.2860	-0.0012	γ_{xy}	0.0362	-0.0001

σ_v	1.8472	2.4955			

Table C.7: Stress & strain results using LST element

DISPLACEMENT ($\times 10^{-5}$)			REACTION FORCE ($\times 10^{3}$)		
NODE	*x*	*y*	NODE	*x*	*y*
1.	0	0	1.	-0.0057	-0.1759
2.	0.0005	-0.0688	2.	0	0
3.	0.0125	-0.2787	3.	0	0
4.	0.1351	-0.2911	4.	0	0
5.	0.3003	-0.3095	5.	0	0
6.	0.1457	-0.0910	6.	0	0
7.	0	-0.0157	7.	-1.0057	0
8.	0	0	8.	-1.9886	0.1759
9.	0.0701	-0.0727	9.	0	0

Table C.8: Displacement & reaction force results using LST element

Plot of the von Mises stress with scaled displacements on the original mesh using LST elements is presented in Figure C.5. Notice how the strain linearly changes within the elements – at middle nodes plot is slightly changing direction.

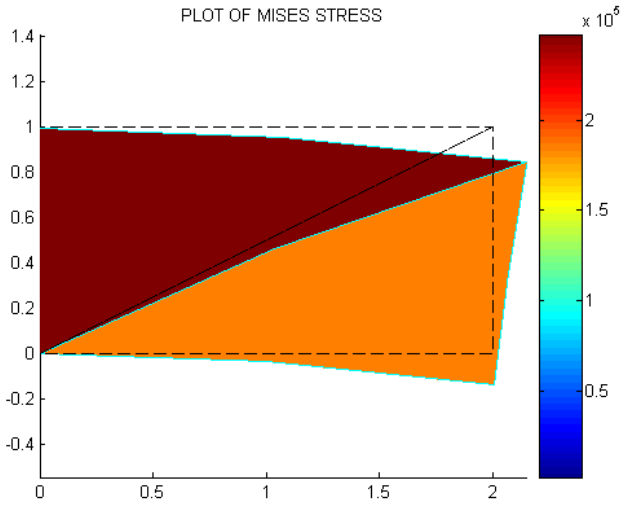

Figure C.5: Von Mises stress with scaled displacements on the original mesh using LST elements

Test example 2

Here we shall present the same problem from Test example 1, but using a different approach in the pre-processing phase. Justification for the stated shall be presented in the following paragraph.

Test example 1 was a basis for understanding the process of modelling a specific engineering problem through finite element code. That is a considerably demanding process, even with just 2 finite elements involved. But luckily, local properties of finite elements can be developed by considering them in isolation, as individual entities. From the standpoint of computer implementation, that means that you can write one subroutine or module that constructs, by suitable parametrization, all elements of one type, instead of writing a new one for each element instance and that is exactly the case with our code. And what that means for our problem is that we can basically apply code developed earlier to a desirable amount of finite elements (i.e. mesh density) which we chose our structure to be discretized with. Although that is a good thing, it is highly unpractical to manually inscribe the input data to Matlab, especially nodal coordinates and element connectivity matrix since for dense meshes this data can be extremely large. In real-life engineering problems we shall always have fine discretizations and dense meshes, even in just some parts of our structure. Therefore, much more effective approach is to use a mesh generator.

For the purposes of this thesis we shall use Gmsh, an automatic three-dimensional finite element mesh generator. For a problem from previous chapter we shall generate a simple mesh made of six finite elements, import the data from this mesh into Matlab, and then follow the same procedure in processing and post-processing phase as before. Finally, in Test example 3 we shall implement this approach to solve a more complex 2D problem which comes closer to problems dealt with in practice.

a) CST element

So, we have the same problem s before, for which the geometry and a simple mesh using CST elements is now made in Gmsh and presented in Figure C.6.

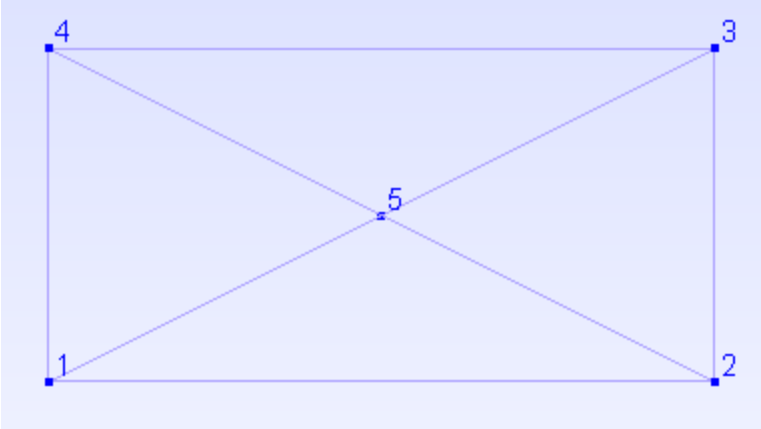

Figure C.6: Simple mesh using CST elements generated in Gmsh

As explained earlier, this simple mesh actually consists of six finite elements, two of which are the 2-node line elements (left and right edge of the plate), and the remaining four triangular CST elements with an intersecting node 5. To the surface of the plane and the left and right edge physical ids were assigned, because of the boundary conditions and the connectivity of elements inside the surface. Boundary conditions are also the reason Gmsh automatically generated those two 2-node line elements at the edges.

Now the only "trick" is in finding a way to import data from this mesh into Matlab, rather than inscribing it manually in Matlab input section. Few Matlab functions that are a part of FEMLAB are doing exactly this. The whole process is incorporated in the Matlab script file "*test_example_solve_6FE_CST.m*", given in the Appendix.

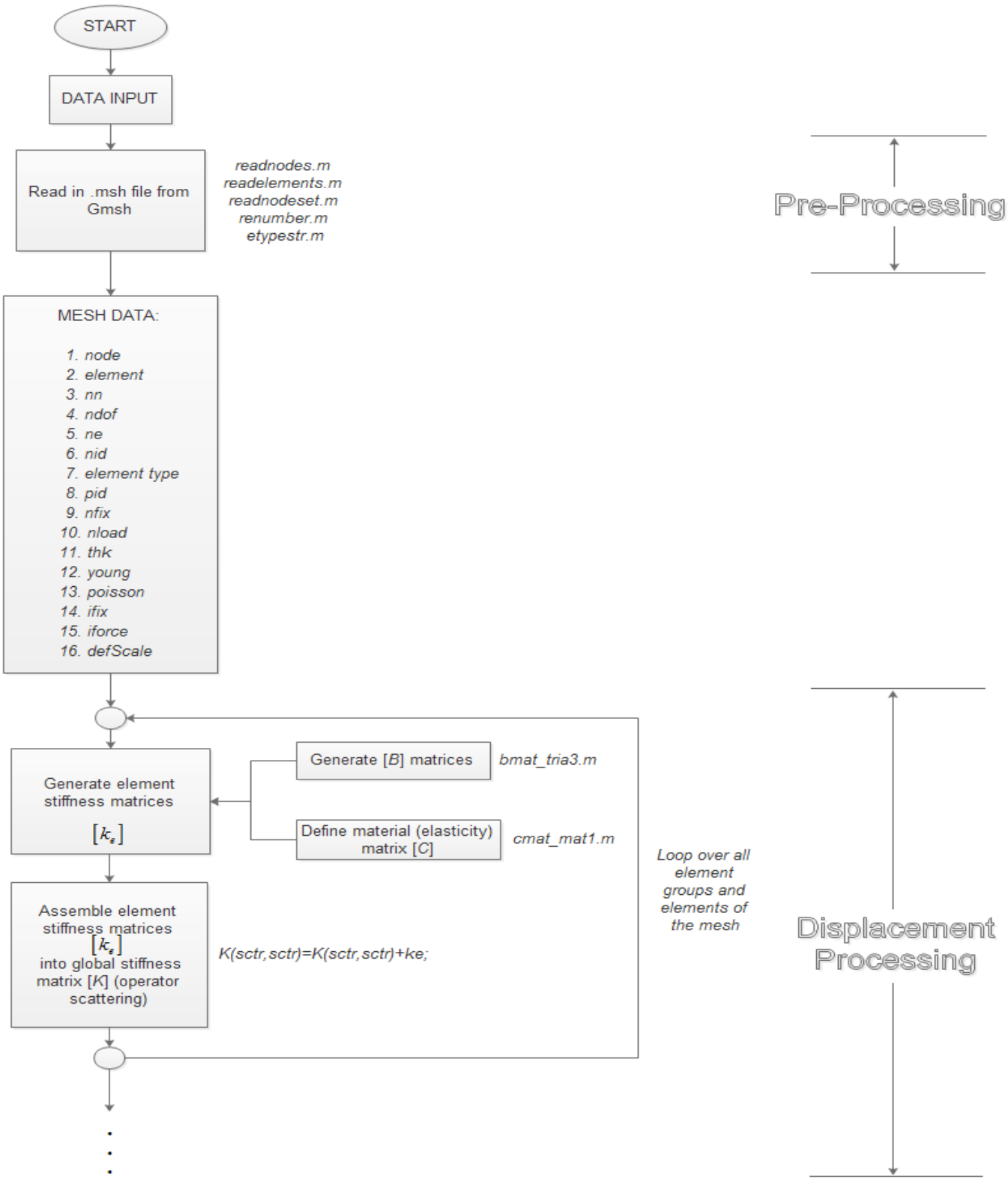

Figure C.7: Pre-processing phase by importing mesh data from Gmsh

Figure C.7 shows the flow diagram which is basically the same as in Figure C.3a, the only difference is that the input data comes from Gmsh, i.e. it is imported from Gmsh into Matlab with the help of functions listed. Code for these functions with complete description is given in the Appendix. In short, *readnodes.m* reads in the node coordinate matrix, *readelements.m* reads in the element connectivity matrix, *readnodeset.m* reads in the physical node ids (the way Gmsh works with boundary conditions), *renumber.m* renumbers the node ids that there are no duplicate entries (sometimes Gmsh messes it up), and *etypestr.m* converts an element type id as from Gmsh to an actual element type string and is used in the function *readelements.m*.

The whole rest of the process is the same as in chapter C.1*a*, i.e. launching a finite element solver in Matlab and presenting, plotting and visualizing results in post-processing.

Plot of the von Mises stress with scaled displacements on the original mesh using CST elements and an imported mesh from Gmsh is presented in Figure C.8.

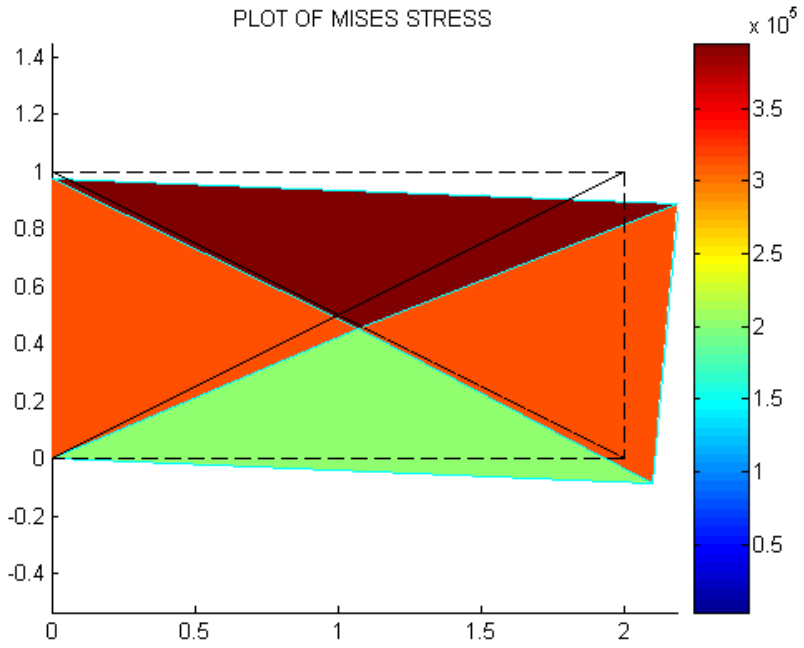

Figure C.8: Von Mises stress with scaled displacements on the original mesh using CST elements and an imported mesh from Gmsh

b) LST element

The procedure for modelling this problem with an LST element in Gmsh is similar, after the command "mesh – 2D" inside Gmsh mesh module, we need to choose the option "Set order 2". Mesh is presented in Figure C.9.

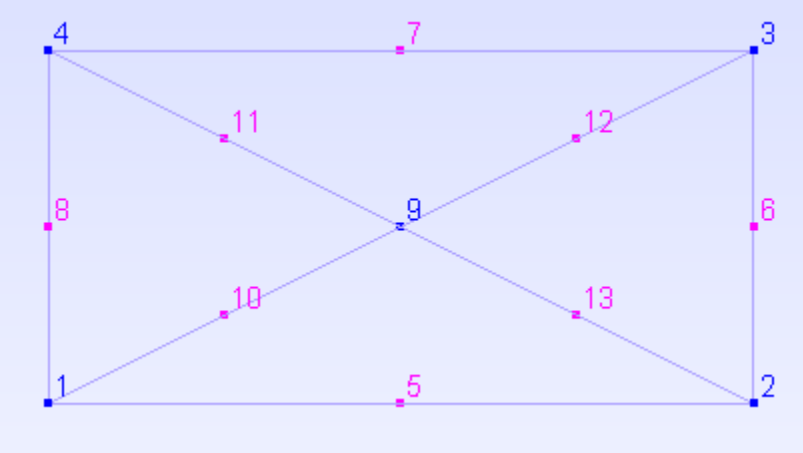

Figure C.9: Mesh using LST elements generated in Gmsh for the problem from chapter 6.1

The rest of the process is basically the same as in chapter C.1*b*, so the function that generates $[B]$ matrix is *bmat_tria6.m* and in the post processing section we have the function *el_renumber.m*, described earlier.

The whole process is incorporated in the Matlab script file "*test_example_solve_6FE_LST.m*", given in the Appendix.

Plot of the von Mises stress with scaled displacements on the original mesh using LST elements and an imported mesh from Gmsh is presented in Figure C.10.

Again we see how the strain linearly changes within the elements, so the more elements we use, the results will be better comparing with CST (for the same number of elements, LST has double number of nodes per each element, so although the mesh has the same density, due to the larger number of nodes the results are considerably better).

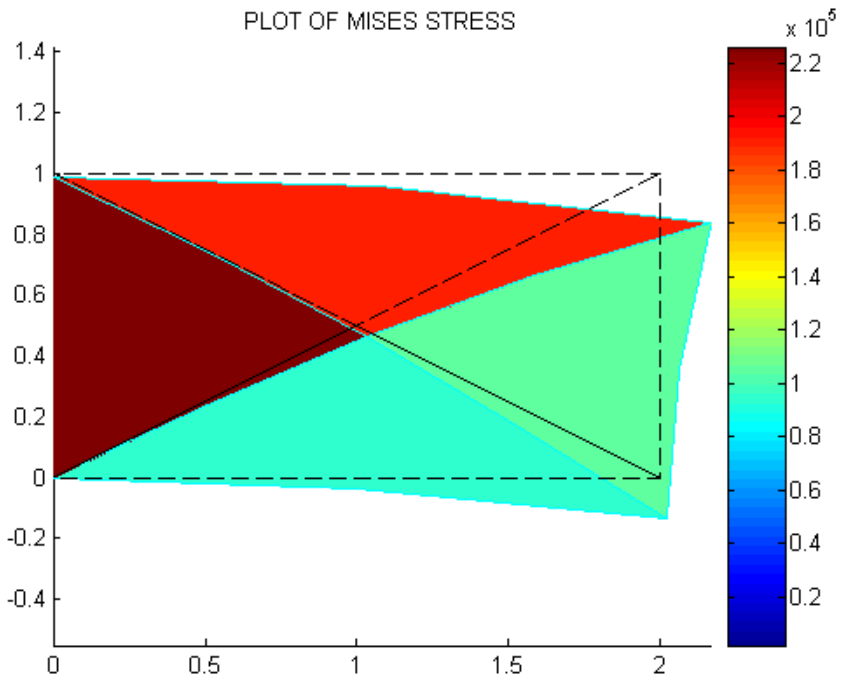

Figure C.10: Von Mises stress with scaled displacements on the original mesh using LST elements and an imported mesh from Gmsh

Test example 3

Consider a problem presented in Figure C.11. For a thin plate with a non symmetrical hole subjected to an in-plane bending load, with fixed support on the left edge, stress analysis needs to be performed.

Parameters:

$$L = 6m,\ h = 1.1m,\ r = 0.3m,\ t = 0.01m,\ F = 20kN,\ E = 210GPa,\ \nu = 0.33.$$

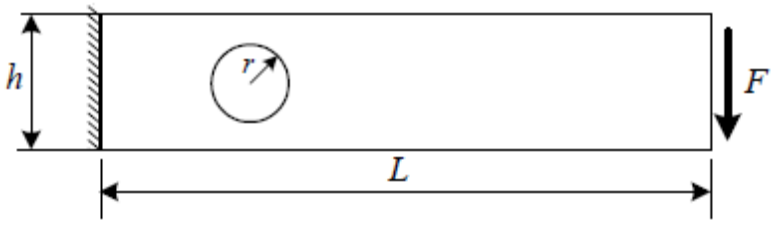

Figure 6.11: Thin plate with a non symmetrical hole

a) CST element

So, we follow the procedure from Test example 2, having in mind that this is a much more realistic problem than those before, so we'll need to have a fine mesh, especially in the area around the hole since the curved geometry of the hole is making it more susceptible to a larger strain gradient.

Mesh for this problem using CST elements and 2 different mesh densities is presented in Figure C.12, with nodes displayed. Notice how the nodes on the left and right edges are in different colours because of the essential and natural boundary conditions imposed through the mechanism of defining physical ids in Gmsh.

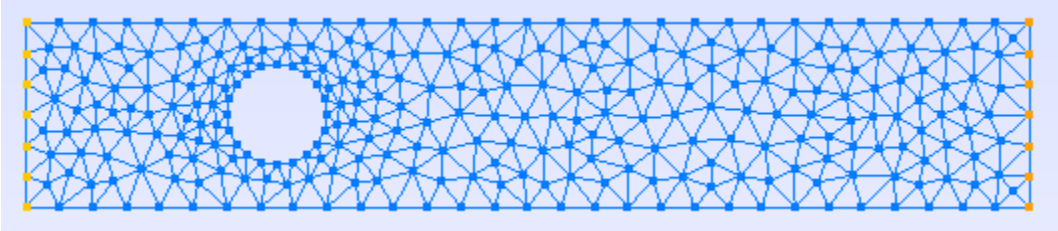

Figure C.12: Mesh using CST elements generated in Gmsh for a thin plate with a hole problem

Now the rest of the procedure is the same as in Test example 2. Mesh data is imported to Matlab, the solving procedure is the same, as is the structure of the post-processing phase.

The whole process is incorporated in the Matlab script file "*plate_w_hole_solve_CST.m*", given in the Appendix.

Plot of the von Mises stress with scaled displacements on the original mesh using CST elements and an imported mesh from Gmsh is presented in Figure C.13.

Notice how the stress in the area around the hole does not have any peaks of higher stress value, which isn't expected even visually when inspecting the results on the plot at first.

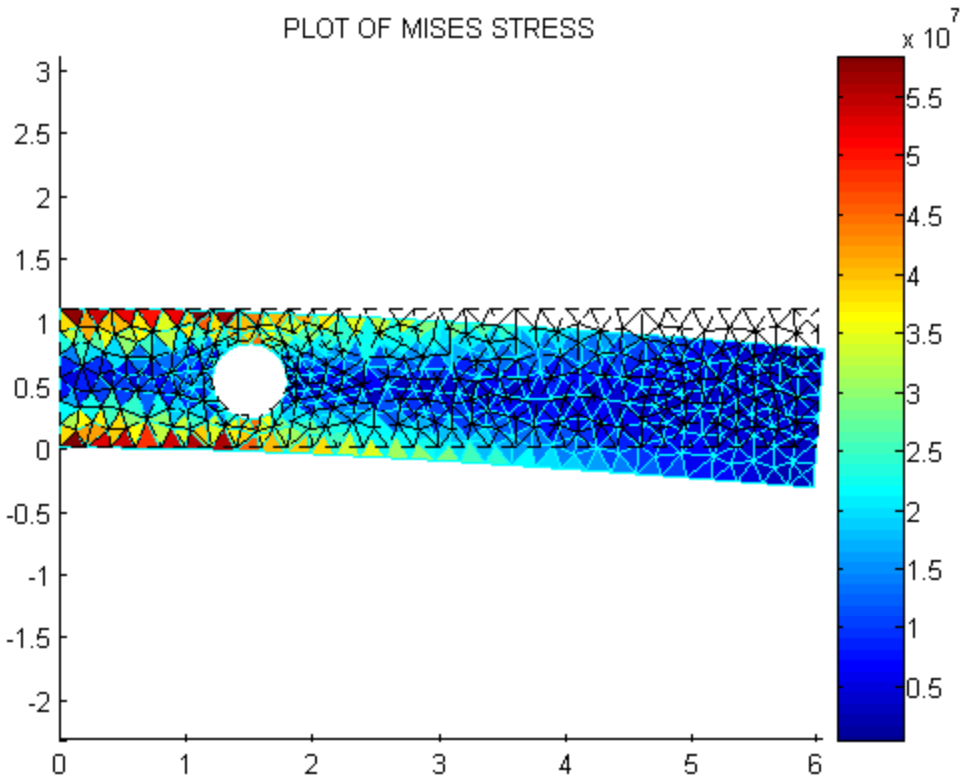

Figure C.13: Von Mises stress with scaled displacements on the original mesh using CST elements and an imported mesh from Gmsh for a thin plate with a hole problem

b) LST element

We follow the procedure from Test example 2*b*.

Mesh for this problem using LST elements and 2 different mesh densities is presented in Figure C.14, with nodes displayed. Notice how the mesh density regarding elements is actually the same as in CST case, but there are far much more nodes.

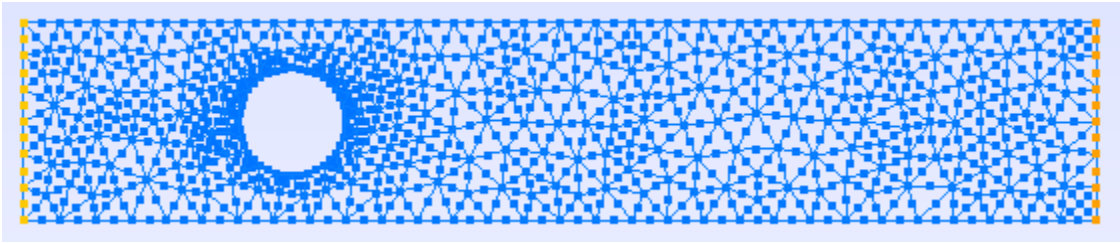

Figure C.14: Mesh using LST elements generated in Gmsh for a thin plate with a hole problem

The rest of the procedure is the same as in Test example 2*b*. Mesh data is imported to Matlab, the solving procedure is the same, as is the structure of the post-processing phase.

The whole process is incorporated in the Matlab script file "*plate_w_hole_solve_LST.m*", given in the Appendix.

Plot of the von Mises stress with scaled displacements on the original mesh using LST elements and an imported mesh from Gmsh is presented in Figure C.15.

Notice how the stress in the area around the hole does have peaks of higher stress value, especially in the top and bottom corner of the hole, which is expected when visually inspecting the results on the plot at first. To conclude, LST element has much greater accuracy, especially around curved edges such as the hole in this example.

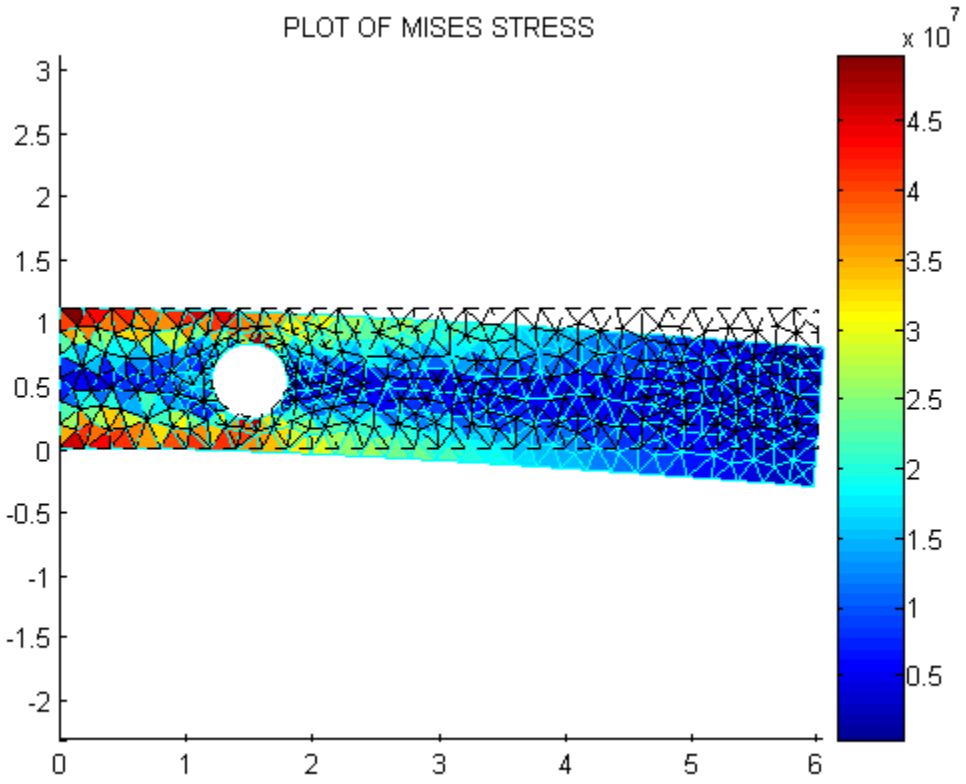

Figure C.15: Von Mises stress with scaled displacements on the original mesh using LST elements and an imported mesh from Gmsh for a thin plate with a hole problem

Comparison of the area around a hole is presented in Figures C.16 and C.17.

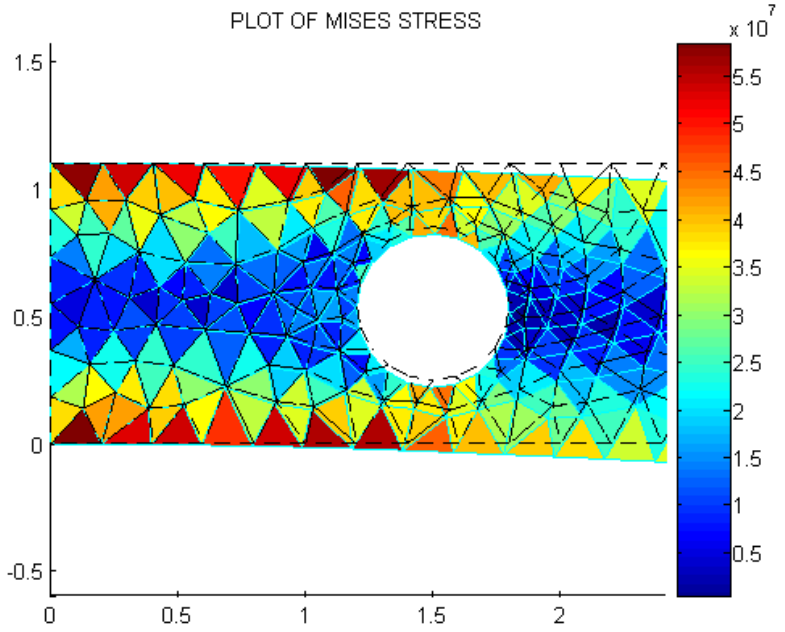

Figure C.16: Area around the hole – CST element

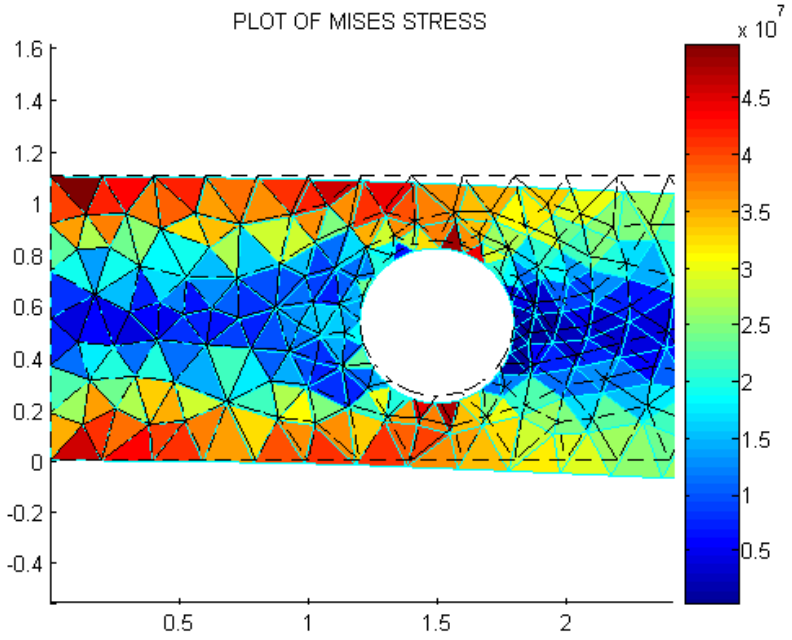

Figure 6.17: Area around the hole – LST element

Appendix D: Matlab codes for classical finite element procedures

```
% fea2d.m
%
% A two dimensional finite element code
%
%--------------------------------------------------------------------
%
%  Input Variables (must be defined)
%
%       node - nodal coordinate matrix (2xnn matrix)
%       conn - element connectivity matrix (2xnumelem matrix)
%       area - element cross sectional area (column vector)
%       young - element Young's modulus (column vector)
%
%       ifix - fixed global dofs
%       iforce - global dofs where point forces are applied (1st col)
%                and value of nodal forces (2nd col)
%
%       defScale - optional parameters that scales the displacement on
%                  output plot
%
%--------------------------------------------------------------------

clear all;
clc;

% ---------------------------------------------------------------------
%         S T A R T   O F  I N P U T   S E C T I O N
% ---------------------------------------------------------------------
node=[0.0 0.0;
      2.0 0.0;
      2.0 1.0;
      0.0 1.0 ];

element=[1 2 3;
         1 3 4];

thk=0.01;

young=210e9;
poisson=0.33;

ifix=[ 2*1-1 2*1 2*4-1 ];

iforce = [ 2*2-1 1000;
           2*3-1 2000];

defScale = 50000;
```

```
%-------------------------------------------------------------------------
%          E N D    O F  I N P U T    S E C T I O N
%-------------------------------------------------------------------------

%
% You should not need to touch anything below
%
fprintf('\n\n-------------------------------------------------\n');
fprintf('|    2D PLANE STRESS FINITE ELEMENT CODE      |\n');
fprintf('-------------------------------------------------\n');

nn=size(node,1);  % number of nodes
ndof=2*nn;        % number of dofs
ne=size(element,1);  % number of elements

% ----------------------- COMPUTATION SECTION -------------------------

% assemble K
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');

c1=young/(1-poisson^2);  % plane stress material stiffness matrix
c2=poisson*c1;
c3=0.5*(1-poisson)*c1;
C=cmat_mat1(young,poisson,'PSTRESS');

K=sparse(ndof,ndof);
for e=1:ne

  conne=element(e,:);

  sctr(1:2:6)=2*conne-1;
  sctr(2:2:6)=2*conne;

  [B,A]=bmat_tria3( node(conne,:) );
  ke=B'*C*B*A*thk;

  K(sctr,sctr) = K(sctr,sctr) + ke;
end

% compute the external force
fext=zeros(ndof,1);
fext(iforce(:,1))=iforce(:,2);

% solve the system
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');
[d,freac]=fesolve(K,fext,ifix);

% compute the strains and stresses and plot results
fprintf(' CALCULATE THE STRESSES\n');
stress=zeros(4,ne);  % matrix of element stresses and strains
strain=zeros(3,ne);  %
```

```
for e=1:ne

  conne=element(e,:);
  B=bmat_tria3( node(conne,:) );
  sctr(1:2:6)=2*conne-1;
  sctr(2:2:6)=2*conne;
  strain(:,e)=B*d(sctr);
  stress(1:3,e)=C*strain(:,e);

  ps=principal_val(stress(1:3,e));  % principal stresses
  stress(4,e)=sqrt( 0.5*( (ps(2)-ps(1))^2 + (ps(3)-ps(2))^2 + ...
    (ps(1)-ps(3))^2 ) );   % the von Mises stresses - equivalent shear
                    % stresses according to von Mises

end

% ---------------------- POST PROCESSING SECTION -------------------------
% plot the stresses
if ( ~exist('defScale') )
   defScale = 10;
end
x = node + defScale*[d(1:2:ndof) d(2:2:ndof)];
fprintf(' PLOTTING RESULTS\n');
clf
hold on
trisurf(element,x(:,1),x(:,2),zeros(nn,1),stress(4,:),'EdgeColor','cyan')
trimesh(element,node(:,1),node(:,2),zeros(nn,1),'FaceColor','none','EdgeColor','black','LineStyle','--')
title('PLOT OF MISES STRESS');
colorbar
view(2)
axis equal

% write the results to an ensight file (you can  read this with
% paraview,  www.paraview.org)
fprintf(' WRITING RESULTS\n');
ensight_fegeometry('fea2d.geom',node,element,'Tria3');
ensight_field('fea2d0000.s11',nodal_avg(stress(1,:),element,node));
ensight_field('fea2d0000.s22',nodal_avg(stress(2,:),element,node));
ensight_field('fea2d0000.s12',nodal_avg(stress(3,:),element,node));
ensight_field('fea2d0000.svm',nodal_avg(stress(4,:),element,node));
ensight_field('fea2d0000.e11',nodal_avg(strain(1,:),element,node));
ensight_field('fea2d0000.e22',nodal_avg(strain(2,:),element,node));
ensight_field('fea2d0000.e12',nodal_avg(strain(3,:),element,node));
ensight_case('fea2d','fea2d.geom',0,...
  {'s11','s22','s12','svm','e11','e22','e12'});

fprintf('\n-------------------------------------------------\n');
fprintf(  '|              END OF PROGRAM              |');
fprintf('\n-------------------------------------------------\n');
```

```
%--------------------------------------------------------------------
%
% fea2d.m
%
% A two dimensional finite element code
%
%--------------------------------------------------------------------
%
%  Input Variables (must be defined)
%
%       node - nodal coordinate matrix (2xnn matrix)
%       conn - element connectivity matrix (2xnumelem matrix)
%       area - element cross sectional area (column vector)
%       young - element Young's modulus (column vector)
%
%       ifix - fixed global dofs
%       iforce - global dofs where point forces are applied (1st col)
%                and value of nodal forces (2nd col)
%
%       defScale - optional parameters that scales the displacement on
%                  output plot
%
%--------------------------------------------------------------------

clear all;
clc;

% --------------------------------------------------------------------
%         S T A R T   O F  I N P U T   S E C T I O N
% --------------------------------------------------------------------
node=[0.0 0.0;
     1.0 0.0;
     2.0 0.0;
     2.0 0.5;
     2.0 1.0;
     1.0 1.0;
     0.0 1.0;
     0.0 0.5;
     1.0 0.5 ];

element=[1 3 5 2 4 9;
        1 5 7 9 6 8];

young=210e9;
poisson=0.33;

thk=0.01;


iforce = [ 2*3-1 500;
          2*4-1 1000;
          2*5-1 1500];
```

```
ifix=[ 2*1-1 2*1 2*7-1 2*8-1 2*8 ];


defScale = 50000;

%-------------------------------------------------------------------------
%          E N D    O F  I N P U T    S E C T I O N
%-------------------------------------------------------------------------

%
% You should not need to touch anything below
%
fprintf('\n\n-------------------------------------------------\n');
fprintf('|    2D PLANE STRESS FINITE ELEMENT CODE      |\n');
fprintf('-------------------------------------------------\n');

nn=size(node,1);  % number of nodes
ndof=2*nn;        % number of dofs
ne=size(element,1);  % number of elements

% ----------------------- COMPUTATION SECTION -------------------------

% assemble K
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');

c1=young/(1-poisson^2);  % plane stress material stiffness matrix
c2=poisson*c1;
c3=0.5*(1-poisson)*c1;
C=cmat_mat1(young,poisson,'PSTRESS');

K=sparse(ndof,ndof);
for e=1:ne

  conne=element(e,:);

  sctr(1:2:12)=2*conne-1;
  sctr(2:2:12)=2*conne;

qpt = [ 0.1666666666667, 0.1666666666667 ;
    0.6666666666667, 0.1666666666667 ;
    0.1666666666667, 0.6666666666667 ];

qwt = [0.3333333333333, 0.3333333333333, 0.3333333333333];

ke=zeros(12,12);
for q=1:3
  xi=qpt(q,:);
  [B,jac]=bmat_tria6( node(conne,:),xi );
  ke = ke + B'*C*B*jac*thk*qwt(q);
end
  K(sctr,sctr) = K(sctr,sctr) + ke;
end
```

```
% compute the external force
fext=zeros(ndof,1);
fext(iforce(:,1))=iforce(:,2);

% solve the system
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');
[d,freac]=fesolve(K,fext,ifix);

% compute the strains and stresses and plot results
fprintf(' CALCULATE THE STRESSES\n');
stress=zeros(4,ne);  % matrix of element stresses and strains
strain=zeros(3,ne);  %

for e=1:ne

  conne=element(e,:);

  sctr(1:2:12)=2*conne-1;
  sctr(2:2:12)=2*conne;

  for q=1:3
    xi=qpt(q,:);
    B=bmat_tria6( node(conne,:),xi );
    strain(:,e)=B*d(sctr);
  end

  stress(1:3,e)=C*strain(:,e);

  ps=principal_val(stress(1:3,e));  % principal streses
  stress(4,e)=sqrt( 0.5*( (ps(2)-ps(1))^2 + (ps(3)-ps(2))^2 + ...
    (ps(1)-ps(3))^2 ) );   % the von Mises stresses - equivalent shear
                        % stresses according to von Mises

end

% ---------------------- POST PROCESSING SECTION -------------------------
% plot the stresses
if ( ~exist('defScale') )
   defScale = 10;
end
x = node + defScale*[d(1:2:ndof) d(2:2:ndof)];
fprintf(' PLOTTING RESULTS\n');

% el_plot=[1 2 3 4 5 9;        % renumbering the sequence of nodes on LST
%          1 9 5 6 7 8];       % element by hand

element2 = el_renumber(element); % renumbering function for the LST element
                            % renumbers the sequence of nodes on the
                            % LST element (for the neccesity of
                            % plotting the stresses and displacements
                            % on the output plot)
```

```
clf
hold on
trisurf(element2,x(:,1),x(:,2),zeros(nn,1),stress(4,:),'EdgeColor','cyan')
trimesh(element2, node(:,1),node(:,2),zeros(nn,1),'FaceColor','none','EdgeColor','black','LineStyle','--')
title('PLOT OF MISES STRESS');
colorbar
view(2)
axis equal

% write the results to an ensight file (you can  read this with
% paraview,  www.paraview.org)
fprintf(' WRITING RESULTS\n');
ensight_fegeometry('fea2d.geom',node,element,'Tria6');
ensight_field('fea2d0000.s11',nodal_avg(stress(1,:),element,node));
ensight_field('fea2d0000.s22',nodal_avg(stress(2,:),element,node));
ensight_field('fea2d0000.s12',nodal_avg(stress(3,:),element,node));
ensight_field('fea2d0000.svm',nodal_avg(stress(4,:),element,node));
ensight_field('fea2d0000.e11',nodal_avg(strain(1,:),element,node));
ensight_field('fea2d0000.e22',nodal_avg(strain(2,:),element,node));
ensight_field('fea2d0000.e12',nodal_avg(strain(3,:),element,node));
ensight_case('fea2d','fea2d.geom',0,...
  {'s11','s22','s12','svm','e11','e22','e12'});

fprintf('\n-------------------------------------------------\n');
fprintf(  '|              END OF PROGRAM              |');
fprintf('\n-------------------------------------------------\n');




% ------------------------------------------------------------------------
%         S T A R T   O F   I N P U T   S E C T I O N
% ------------------------------------------------------------------------

clear all;
clc;
fprintf('\n READING INPUT\n');

% ------------------------- DATA INPUT SECTION -------------------------
[node,nid]=readnodes('plate_w_hole_CST.msh');       % read in mesh from
element=readelements('plate_w_hole_CST.msh',10);    % gmsh file
nfix=readnodeset('plate_w_hole_CST.msh',11);
nload=readnodeset('plate_w_hole_CST.msh',12);

node=node(:,1:2);               % renumber node ids in element, nfix and
element=renumber(element,nid);  % nload since gmsh does not always number
nfix=renumber(nfix,nid);        % them  consecutively
nload=renumber(nload,nid);



young = 210e9;
poisson = 0.33;
thk=0.01;
```

```
ifix=[ 2*nfix'-1 2*nfix' ];     % fix nfix nodes in x and y direction,
                        % constraining dofs on the left edge

% ifix=[ 2*1-1 2*1 2*4-1 2*4 2*31-1 2*31 2*32-1 2*32 ]; % fixing individual
                                    % nodes by hand


fext(2*nload)=-2857.14;   % line load at nload in y-direction - load on the
                    % whole right edge


% iforce = [ 2*2 -20000;     % optional, applied load in individual nodes
%           3*2 -20000;     % and directions
%           18*2 -20000;
%           19*2 -20000];

defScale = 50;


%------------------------------------------------------------------------
%           E N D    O F  I N P U T    S E C T I O N
%------------------------------------------------------------------------
%
% You should not need to touch anything below
%
fprintf('\n\n-------------------------------------------------\n');
fprintf('|    2D PLANE STRESS FINITE ELEMENT CODE      |\n');
fprintf('-------------------------------------------------\n');

nn=size(node,1);  % number of nodes
ndof=2*nn;        % number of dofs
ne=size(element,1);  % number of elements

% ----------------------- COMPUTATION SECTION -------------------------

% assemble K
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');

c1=young/(1-poisson^2);  % plane stress material stiffness matrix
c2=poisson*c1;
c3=0.5*(1-poisson)*c1;
C=cmat_mat1(young,poisson,'PSTRESS');

K=sparse(ndof,ndof);
for e=1:ne

  conne=element(e,:);

  sctr(1:2:6)=2*conne-1;
  sctr(2:2:6)=2*conne;
```

```
  [B,A]=bmat_tria3( node(conne,:) );
  ke=B'*C*B*A*thk;

  K(sctr,sctr) = K(sctr,sctr) + ke;

end

% compute the external force
fext=zeros(ndof,1);
fext(2*nload)=-2857.14;
% fext(iforce(:,1))=iforce(:,2);  % when defining iforce in input

% solve the system
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');
[d,freac]=fesolve(K,fext,ifix);

% compute the strains and stresses and plot results
fprintf(' CALCULATE THE STRESSES\n');
stress=zeros(4,ne);  % matrix of element stresses and strains
strain=zeros(3,ne);  %

for e=1:ne

  conne=element(e,:);
  B=bmat_tria3( node(conne,:) );
  sctr(1:2:6)=2*conne-1;
  sctr(2:2:6)=2*conne;
  strain(:,e)=B*d(sctr);
  stress(1:3,e)=C*strain(:,e);

  ps=principal_val(stress(1:3,e));  % principal streses
  stress(4,e)=sqrt( 0.5*( (ps(2)-ps(1))^2 + (ps(3)-ps(2))^2 + ...
    (ps(1)-ps(3))^2 ) );    % the von Mises stresses - equivalent shear
                          % stresses according to von Mises



end

% ---------------------- POST PROCESSING SECTION -------------------------
% plot the stresses
if ( ~exist('defScale') )
   defScale = 0.1;
end
x = node + defScale*[d(1:2:ndof) d(2:2:ndof)];
fprintf(' PLOTTING RESULTS\n');
clf
hold on
trisurf(element,x(:,1),x(:,2),zeros(nn,1),stress(4,:),'EdgeColor','cyan')
trimesh(element,node(:,1),node(:,2),zeros(nn,1),'FaceColor','none','EdgeColor','black','LineStyle','--')
title('PLOT OF MISES STRESS');
colorbar
view(2)
axis equal
```

```
% write the results to an ensight file (you can  read this with
% paraview,  www.paraview.org)
fprintf(' WRITING RESULTS\n');
ensight_fegeometry('fea2d.geom',node,element,'Tria3');
ensight_field('fea2d0000.s11',nodal_avg(stress(1,:),element,node));
ensight_field('fea2d0000.s22',nodal_avg(stress(2,:),element,node));
ensight_field('fea2d0000.s12',nodal_avg(stress(3,:),element,node));
ensight_field('fea2d0000.svm',nodal_avg(stress(4,:),element,node));
ensight_field('fea2d0000.e11',nodal_avg(strain(1,:),element,node));
ensight_field('fea2d0000.e22',nodal_avg(strain(2,:),element,node));
ensight_field('fea2d0000.e12',nodal_avg(strain(3,:),element,node));
ensight_case('fea2d','fea2d.geom',0,...
  {'s11','s22','s12','svm','e11','e22','e12'});

fprintf('\n-----------------------------------------------\n');
fprintf(  '|              END OF PROGRAM              |');
fprintf('\n-----------------------------------------------\n');
```

Appendix E: Matlab codes for substructuring and superelements technique

```
%Superstructure – read in the master nodes
[node_L1,nid_L1]=readnodes('superstructure_LVL1.msh');

% node_L1=node_L1(:,1:2); % just first 2 columns (z-coord =0, 2D example)

%master nodes to work with:
sNODE = size(node_L1);
for ii=1:sNODE
   sXYZ(ii,:) = node_L1(nid_L1(ii),:);
end

save('sXYZ');
 --- END-----



%Level two superstructure – read in nodes
[node_L2,nid_L2]=readnodes('superstructure_LVL2.msh');  %

% node_L1=node_L1(:,1:2); % just first 2 columns (z-coord =0, 2D example)

sNODE = size(node_L2);
for ii=1:sNODE
   sXYZ_L2(ii,:) = node_L2(nid_L2(ii),:);
end

save('sXYZ_L2');

----END-----



% ------------------------------------------------------------------------
%        S T A R T   O F   I N P U T   S E C T I O N
% ------------------------------------------------------------------------

clear all;
clc;
fprintf('\n READING INPUT\n');

% ------------------------ DATA INPUT SECTION -------------------------
[node_S1,nid_S1]=readnodes('substructure_S1_LVL3.msh');   % read in mesh from
element_S1=readelements('substructure_S1_LVL3.msh',7);    % gmsh file

node_S1_2D=node_S1(:,1:2:3); element_S1=renumber(element_S1,nid_S1);  %
```

```
% edge detection

%read in master node ids od the substructure

id_edge_snodes=readnodeset('substructure_S1_LVL3.msh',8);

NODE_S = size(id_edge_snodes,1);
NEQ = NODE_S * 2;

young = 200e9;
poisson = 0.3;
thk=0.01;

%--------------------------------------------------------------------------
%           E N D    O F  I N P U T    S E C T I O N
%--------------------------------------------------------------------------
%
% You should not need to touch anything below
%
fprintf('\n\n-------------------------------------------------\n');
fprintf('|    2D PLANE STRESS FINITE ELEMENT CODE      |\n');
fprintf('-------------------------------------------------\n');

nn=size(node_S1,1);  % number of nodes
ndof=2*nn;        % number of dofs
ne=size(element_S1,1);  % number of elements

% ----------------------- COMPUTATION SECTION -------------------------

% assemble K
fprintf(' ASSEMBLING STIFFNESS MATRIX\n');

c1=young/(1-poisson^2);  % plane stress material stiffness matrix
c2=poisson*c1;
c3=0.5*(1-poisson)*c1;
C=cmat_mat1(young,poisson,'PSTRESS');

K=sparse(ndof,ndof);
for e=1:ne

  conne=element_S1(e,:);

  sctr(1:2:6)=2*conne-1;
  sctr(2:2:6)=2*conne;

  [B,A]=bmat_tria3( node_S1_2D(conne,:) );
  ke=B'*C*B*A*thk;

  K(sctr,sctr) = K(sctr,sctr) + ke;

end

KII = K(1:NEQ,1:NEQ); KIJ=K(1:NEQ, NEQ+1:ndof);
```

```
KJI = K(NEQ+1:ndof,1:NEQ);  KJJ =K(NEQ+1:ndof, NEQ+1:ndof);

K_red=KII-KIJ*inv(KJJ)*KJI;

load('sXYZ_L2');
node_S1_boundary=node_S1(1:NODE_S,:);

id_local_global_L2=id_local_global_transfer(node_S1_boundary, sXYZ_L2);

disp('Nodal vector for substructure S1 (LEVEL 3 ---> 2):')
disp(id_local_global_L2)

save('SUB_L3_S1.mat','K_red','id_local_global_L2','NEQ','KII','KIJ','KJI','KJJ','node_S1','element_S1','ne','C')

fprintf('\n-------------------------------------------------\n');
fprintf(  '|              END OF PROGRAM              |');
fprintf('\n-------------------------------------------------\n');




clear all;
clc;

load('s_disp')
load('SUB_L2_S1.mat','id_local_global') %

nn_L2=size(id_local_global,1);
ndof_L2=2*nn_L2;

  sctr(1:2:ndof_L2)=3*id_local_global-2;
% sctr( :2:ndof_L2)=3*id_local_global-1;
  sctr(2:2:ndof_L2)=3*id_local_global;

id_local_global_dof_L2=sctr'; %

s_disp_L2=s_disp(id_local_global_dof_L2,:); % L2

load('SUB_L3_S1','id_local_global_L2','KJI','KJJ','node_S1','element_S1','ne','C')

nn_boundary=size(id_local_global_L2,1);
ndof_boundary=2*nn_boundary;

  sctr2(1:2:ndof_boundary)=2*id_local_global_L2-1;
  sctr2(2:2:ndof_boundary)=2*id_local_global_L2;

id_local_global_dof=sctr2';

disp_boundary=s_disp_L2(id_local_global_dof,:);
```

```
disp_internal=inv(KJJ)*(-KJI)*disp_boundary;


disp_S1=[disp_boundary; disp_internal];

fprintf(' CALCULATE THE STRESSES\n');

stress=zeros(4,ne);  % matrix of element stresses and strains
strain=zeros(3,ne);  %

node_S1_2D=node_S1(:,1:2:3); %

% obtaining stress
for e=1:ne

  conne=element_S1(e,:);
  B=bmat_tria3( node_S1_2D(conne,:) );
  sctr3(1:2:6)=2*conne-1;
  sctr3(2:2:6)=2*conne;
  strain(:,e)=B*disp_S1(sctr3);
  stress(1:3,e)=C*strain(:,e);

  ps=principal_val(stress(1:3,e));  % principal streses
  stress(4,e)=sqrt( 0.5*( (ps(2)-ps(1))^2 + (ps(3)-ps(2))^2 + ...
    (ps(1)-ps(3))^2 ) );    % the von Mises stresses – equivalent tensile
                        % stresses according to von Mises

end


defScale = 20;

nn=size(node_S1_2D,1);              % number of nodes of substructure S1
ndof=2*nn;                          % number of dofs of S1


if ( ~exist('defScale') )
   defScale = 0.1;
end
x = node_S1_2D + defScale*[disp_S1(1:2:ndof) disp_S1(2:2:ndof)];
fprintf(' PLOTTING RESULTS\n');
clf
hold on
trisurf(element_S1,x(:,1),x(:,2),zeros(nn,1),stress(4,:),'EdgeColor','cyan')
trimesh(element_S1,node_S1_2D(:,1),node_S1_2D(:,2),zeros(nn,1),'FaceColor','none','EdgeColor','black','LineStyle'
,'--')
title('PLOT OF MISES STRESS FOR SUBSTRUCTURE S1 (L3)');
colorbar
view(2)
axis equal


% write the results to Paraview (ensight file format):
```

```
fprintf(' WRITING RESULTS\n');
ensight_fegeometry('fea2d.geom',node_S1,element_S1,'Tria3');
ensight_field('fea2d0000.s11',nodal_avg(stress(1,:),element_S1,node_S1));
ensight_field('fea2d0000.s22',nodal_avg(stress(2,:),element_S1,node_S1));
ensight_field('fea2d0000.s12',nodal_avg(stress(3,:),element_S1,node_S1));
ensight_field('fea2d0000.svm',nodal_avg(stress(4,:),element_S1,node_S1));
ensight_field('fea2d0000.e11',nodal_avg(strain(1,:),element_S1,node_S1));
ensight_field('fea2d0000.e22',nodal_avg(strain(2,:),element_S1,node_S1));
ensight_field('fea2d0000.e12',nodal_avg(strain(3,:),element_S1,node_S1));
ensight_case('fea2d','fea2d.geom',0,...
  {'s11','s22','s12','svm','e11','e22','e12'});

fprintf('\n-------------------------------------------------\n');
fprintf(  '|              END OF PROGRAM              |');
fprintf('\n-------------------------------------------------\n');



load('sXYZ_L2');

nn_global=size(sXYZ_L2,1); %
ndof_global=2*nn_global;

KG_L2_S1=zeros(ndof_global,ndof_global); %

nSS=10; % number of substructures

for ii=1:nSS

    substr_ii=sprintf('SUB_L3_S%d',ii);
    load (substr_ii,'K_red','id_local_global_L2','NEQ')

    sctr(1:2:NEQ)=2*id_local_global_L2-1;
    sctr(2:2:NEQ)=2*id_local_global_L2;

    KG_L2_S1(sctr,sctr)=KG_L2_S1(sctr,sctr)+K_red;

    clear sctr
    clear K_red
    clear id_local_global_L2
    clear NEQ
end

% transfer of dana from L2 to L1

load('sXYZ');
id_local_global=id_local_global_transfer(sXYZ_L2, sXYZ);

NODE_S = size(sXYZ_L2,1);
NEQ = NODE_S * 2;
disp('Nodal vector for substructure S1 (LEVEL 2 ---> 1):')
disp(id_local_global)
save('SUB_L2_S1.mat','KG_L2_S1','id_local_global','NEQ'
```

```
clc;
clear all;

load('sXYZ');

nn_global=size(sXYZ,1);
ndof_global=3*nn_global;

KG_L1=zeros(ndof_global,ndof_global);

KG_L1_S1=zeros(ndof_global,ndof_global);
KG_L1_S2=zeros(ndof_global,ndof_global);
KG_L1_S3=zeros(ndof_global,ndof_global);
KG_L1_S4=zeros(ndof_global,ndof_global);
KG_L1_S5=zeros(ndof_global,ndof_global);
KG_L1_S6=zeros(ndof_global,ndof_global);
KG_L1_S7=zeros(ndof_global,ndof_global);

% ------------------------------------------------------------------------

% S1:

   load ('SUB_L2_S1.mat','KG_L2_S1','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
%   sctr( :2:NEQ)=3*id_local_global-1;  %
   sctr(2:2:NEQ)=3*id_local_global;

   KG_L1_S1(sctr,sctr)=KG_L1_S1(sctr,sctr)+KG_L2_S1;

   clear sctr

   clear id_local_global
   clear NEQ


% S2:

   load ('SUB_L2_S2.mat','KG_L2_S2','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
%   sctr( :2:NEQ)=3*id_local_global-1;  %
   sctr(2:2:NEQ)=3*id_local_global;

   KG_L1_S2(sctr,sctr)=KG_L1_S2(sctr,sctr)+KG_L2_S2;

   clear sctr

   clear id_local_global
```

```
    clear NEQ


%  S3:

    load ('SUB_L2_S3.mat','K_red','id_local_global','NEQ')

    sctr(1:2:NEQ)=3*id_local_global-2;
%   sctr( :2:NEQ)=3*id_local_global-1;  %
    sctr(2:2:NEQ)=3*id_local_global;

    KG_L1_S3(sctr,sctr)=KG_L1_S3(sctr,sctr)+K_red;

    clear sctr
    clear K_red
    clear id_local_global
    clear NEQ


%  S4:

    load ('SUB_L2_S4.mat','KG_L2_S4','id_local_global','NEQ')

%   sctr( :2:NEQ)=3*id_local_global-2;    %
    sctr(1:2:NEQ)=3*id_local_global-1;
    sctr(2:2:NEQ)=3*id_local_global;

    KG_L1_S4(sctr,sctr)=KG_L1_S4(sctr,sctr)+KG_L2_S4;

    clear sctr

    clear id_local_global
    clear NEQ


%  S5:

    load ('SUB_L2_S5.mat','KG_L2_S5','id_local_global','NEQ')

%   sctr( :2:NEQ)=3*id_local_global-2;    %
    sctr(1:2:NEQ)=3*id_local_global-1;
    sctr(2:2:NEQ)=3*id_local_global;

    KG_L1_S5(sctr,sctr)=KG_L1_S5(sctr,sctr)+KG_L2_S5;

    clear sctr

    clear id_local_global
    clear NEQ
```

```
%  S6:

   load ('SUB_L2_S6.mat','KG_L2_S6','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
   sctr(2:2:NEQ)=3*id_local_global-1;
%   sctr( :2:NEQ)=3*id_local_global;  %

   KG_L1_S6(sctr,sctr)=KG_L1_S6(sctr,sctr)+KG_L2_S6;

   clear sctr

   clear id_local_global
   clear NEQ


%  S7:

   load ('SUB_L2_S7.mat','KG_L2_S7','id_local_global','NEQ')

   sctr(1:2:NEQ)=3*id_local_global-2;
   sctr(2:2:NEQ)=3*id_local_global-1;
%   sctr( :2:NEQ)=3*id_local_global;  %

   KG_L1_S7(sctr,sctr)=KG_L1_S7(sctr,sctr)+KG_L2_S7;

   clear sctr

   clear id_local_global
   clear NEQ

% ------------------------------------------------------------------------

KG_L1=KG_L1_S1+KG_L1_S2+KG_L1_S3+KG_L1_S4+KG_L1_S5+KG_L1_S6+KG_L1_S7;

save('KG_L1')
```

-----END-----

```
nfix=readnodeset('superstructure_LVL1.msh',123);


nfix_in_xy=readnodeset('superstructure_LVL1.msh',126);
nfix_in_xy=nfix_in_xy(9:size(nfix_in_xy,1));

nfix_in_xz=readnodeset('superstructure_LVL1.msh',125);
nfix_in_xz=nfix_in_xz(13:size(nfix_in_xz,1));

nload=readnodeset('superstructure_LVL1.msh',124);
```

```
ifix=[ 3*nfix'-2 3*nfix'-1 3*nfix' 3*nfix_in_xy' 3*nfix_in_xz'-1 ];

load('sXYZ');

nn_global=size(sXYZ,1);
ndof_global=3*nn_global;

fext=zeros(ndof_global,1);
fext(3*nload)=-8695.65;

load('KG_L1')

[s_disp,freac]=fesolve(KG_L1,fext,ifix);

save('s_disp')
```

---END---