POLITECNICO DI TORINO

MASTER's Degree in **MECHATRONIC ENGINEERING**

MASTER's Degree Thesis

CONCEPT MODELING OF AN AUTOMOTIVE BODY IN WHITE FOR EARLY-STAGE DESIGN EXPLORATION AND OPTIMIZATION

Supervisors:

Prof. Malan Stefano Alberto

Eng. Orlando Stefano

Eng. Brughmans Marc

Candidate:

Piero Brigida

APRIL 2021

Summary

Introduction

The development and application of the vehicle advanced *Computer Aided Engineering* (*CAE*) tools in the automotive sector is a certified approach to predict the various functional performance attributes (crashworthiness, ride and handling, noise, vibration & harshness, etc.) and adapt the design based on the outcomes of virtual simulations and their relation to targets and requirements. Alternatively, *Body in White* (*BIW*) *CAE* models have reached a high level of detail and complexity, resulting in heavy simulation models and long computation times.

As a result, the design of an automotive car body is a lengthy and iterative process where the vehicle performances for the different attributes can be evaluated and balanced only at an advanced design stage, when the detailed geometry and simulation model are available.

The reduction of the time to market has been one of the main thrusts in the automotive industry in the last years, pushing researchers to find more efficient methods to solve design problems.

In order to frontload potential design issues in an early design stage, concept modelling methods are often employed by *Original Equipment Manufacturer (OEMs).*

In the conceptual stages of design, engineers need simplified models to extrapolate the structural and functional characteristics and apply custom modifications for achieving the best vehicle design in order to balance the different attributes. Unlike the detailed *Finite Element* (*FE*) model, the concept CAE model can be created without any need of the detailed *computer-aided design* (*CAD*) data.

The usage of detailed finite-element models of the vehicle at early design stage is too time-consuming. Therefore, engineers have the need of utilizing trade-off simplified models of Body-in-White (*BIW*), composed of only the basic structural elements (beams, joints, panels) while at the same time being able to capture the static and dynamic behaviour of the vehicle.

The work described in this thesis consists of creating a *BIW* concept model that offers the advantages of geometric parameterization of the car body main structural members while retaining a reliable stiffness

representation of them based on beam and joint libraries from existing car bodies. The created model consists of a network of 1-D stiffness elements (beams and joints) which can be assembled in a very short time, allowing for fast design space explorations and optimization.

I. Review of main principles of Finite Element Analysis (FEA) or Finite Element Method (FEM) theory

Introduction

In this opening chapter, the basic theory of finite element analysis (FEA) is indicated by providing examples on different techniques of meshing and structural static analysis. This technique is well implemented inside the main concept of the thesis. It provides an empirical method for improving and forecasting product performance and reliability by performing different types of simulations (as shown in Figure I-1). Flexibility is the greatest advantage as it offers a way to evaluate several types of designs and materials. It is a key element in the optimization process which reduces the physical prototyping and testing.

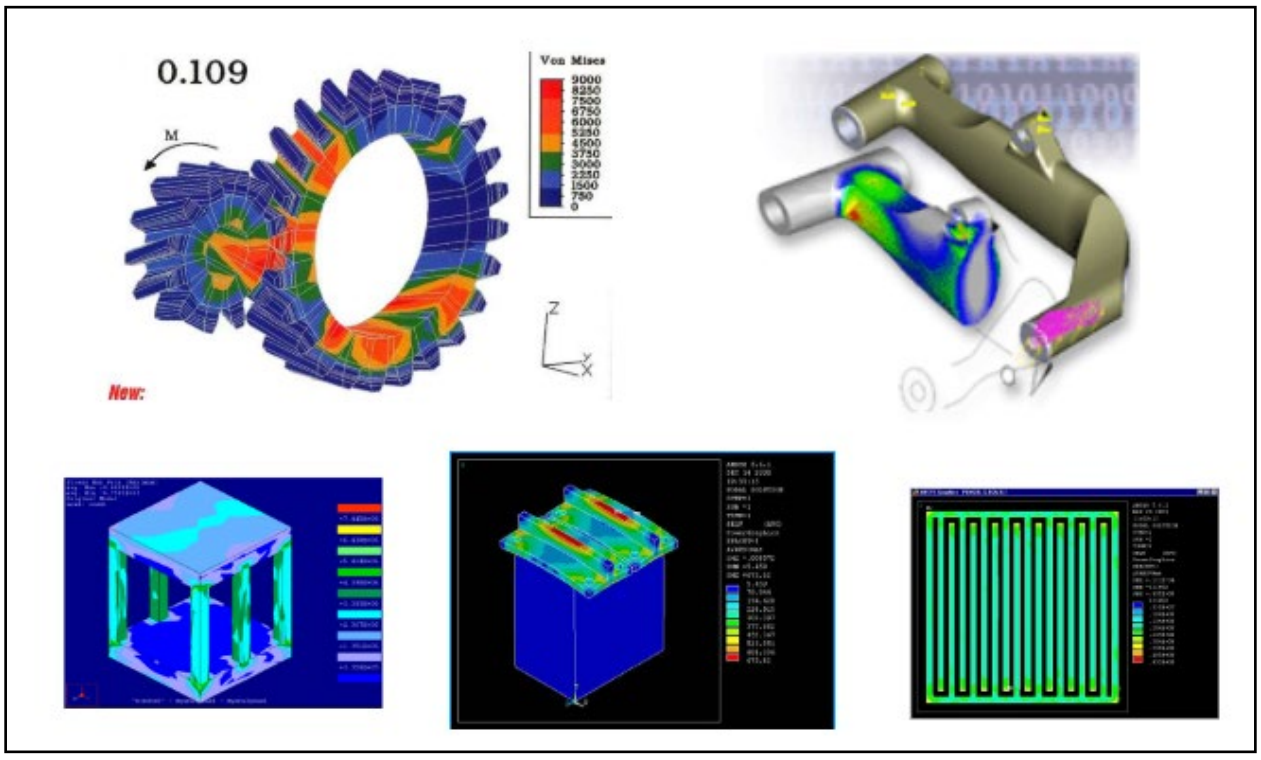


Figure I-1 Examples of different FEA applications (Introduction to Finite Element Analysis)

Definitions and common FEA Applications

Finite element analysis (or FEA) is a computer-based procedure adopted for solving physical phenomena by applying a mathematic technique also known as *Finite Element Method (or FEM).* For years, engineers used it to reduce the number of experiments for optimizing products in their design phase and to develop better components, faster. In substance, it is defined as a virtual simulation that allows precise and efficient modelling and solving of mechanical, thermal or other complex systems and improving performances in manufacturing processes. Indeed, it is primarily applied in automotive, aerospace, civil and biomechanical industries. The different types of applications are:

- *Structural/Stress Analysis*
- *Static/Dynamic*
- *Linear/Non linear*
- *Fluid flow*
- *Heat transfer*
- *Electromagnetic fields*
- *Soil mechanics*
- *Acoustics*
- *Biomechanics*

Concepts and workflow

The FEA represents the bridge that interconnects the world of complex objects, which is characterized by complex/arbitrary geometry or material discontinuity and simple analysis by means of *discretization*.

Moreover, discretization consists in the generation of a simplified and idealized physical model that tries to reproduce (as faithfully as possible) the real world. Then, this model is transformed into a mathematical one and, in the end, is discretized into a mesh model.

Finite element definition and classification

The key element during this process is the *finite element (FE).* It is described through a set of points called ''*nodes*'' which are interconnected with each other by segments that form the mesh. All the calculations are made at a limited number of points (*nodes*). Analytically, a node corresponds to a point in which a displacement is localized, due to an applied stress/strain.

An element is the entity joining nodes and forming a specific shape such as a triangle or quadrilateral. Each *FE* can have a different thickness or density and can be described in terms of section inertia or material properties. The *FE* tends to reproduce the real geometry and structural properties of an object, focusing on the possible reaction under certain conditions of stress. Certain regions could present a different mesh density than others depending on the stress level of the different area. *Figure I-2* shows how the different areas of a component can present different levels of stress, depending on the location and colours.

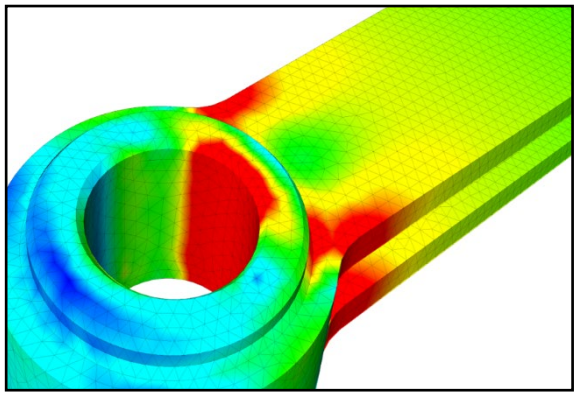


Figure I-2 Stress distribution on a mechanical component (SIMSCALE)

Geometry and properties modelling

Meshing is defined as:

> *"the practice of creating a mesh, a subdivision of a continuous geometric space into discrete geometric and topological cells."* (Wikipedia, s.d.)

Depending on the complexity of the domain and the type of mesh desired (triangular/quadrilateral), the different meshes are created through computer algorithms. *Figure I-3* shows a discretization of a 3-D object by means of quadratic tetrahedral elements:

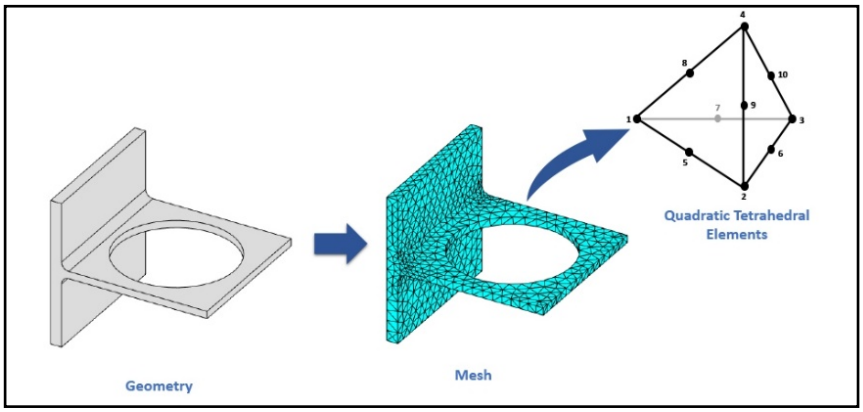


Figure I-3 Example of meshing of a 3-D object (Matlab)

In order to predict real-world behaviour with high precision, a computer-aided design (CAD) model is used for representing the concrete parts being reproduced as well as the material properties and the applied forces and constraints. As shown in *Figure I-4,* the whole structure of a Body-In-White (BIW) is simplified and modelled mathematically by choosing appropriate finite elements.

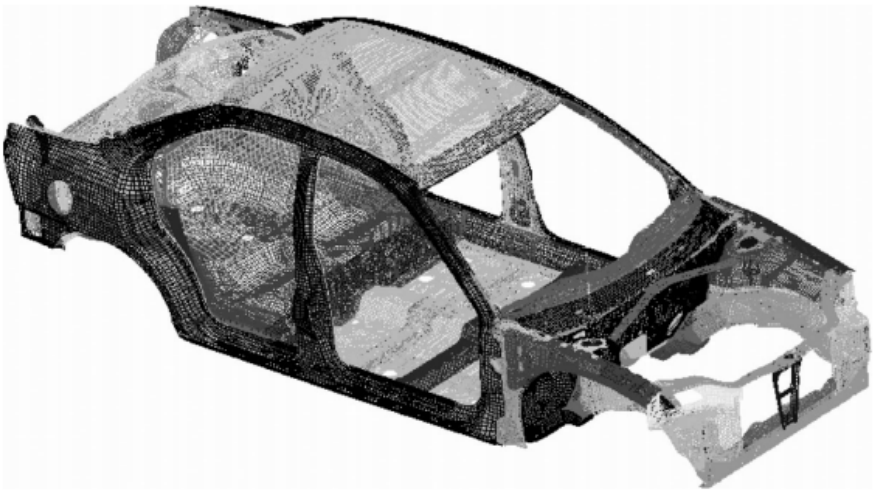


Figure I-4 Body-In-White in FEM Environment

The mesh of the finite element affects the accuracy that can be achieved from a Finite Element model. Indeed, the finite element mesh plays an important role in FEA analysis. Through partitioning the initial CAD model into smaller domains (*elements*), it is possible to apply a set of polynomial functions that then will be solved. The computational effort of the calculator is proportional to the number of governing equations that are defined over each element. As these elements are made smaller and smaller through a process called *mesh refinement (Figure I-5)*, the computed solution will approach the expected result.

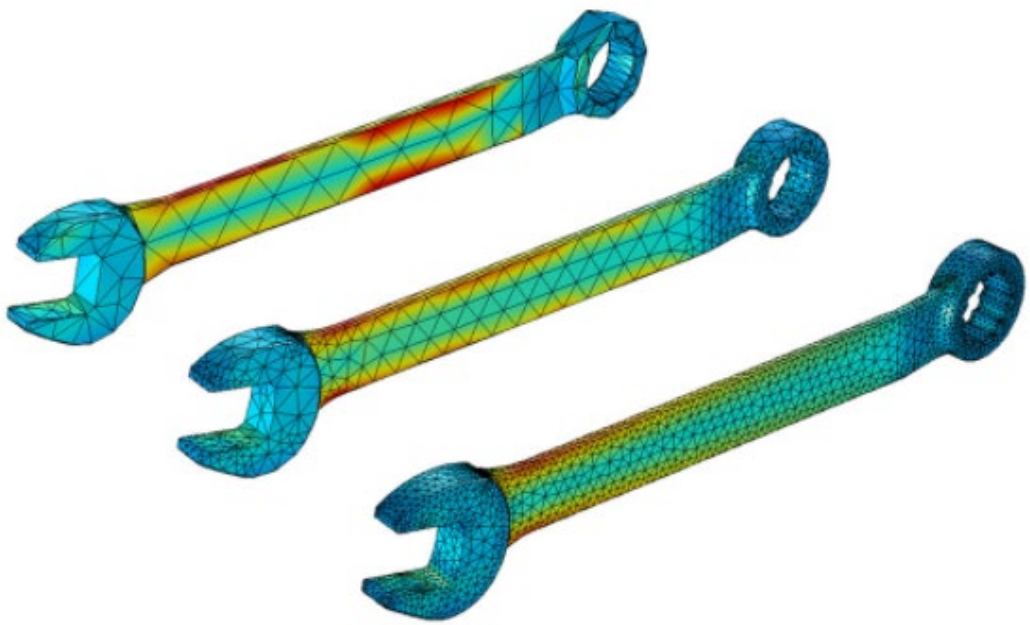


Figure I-5 Three iteration of a mesh refinement study of a wrench (COMSOL)

Different types of techniques for meshing have been developed by engineers. The two most famous are the *h-refinement* and *p-refinement*. The first one is the easiest to actuate. It consists in increasing the number of elements used to model a given domain. In this way, the element size is drastically reduced. The second one, instead, exploits the order of interpolation functions, leaving intact the element size. Increasing the element order can be advantageous for complex 3-D geometries but the computational requirements grow faster than the other mesh refinement technique. In *Figure I-6* and *Figure I-7,* a graphical comparison between the two techniques applied at the same plate CAD model is proposed.

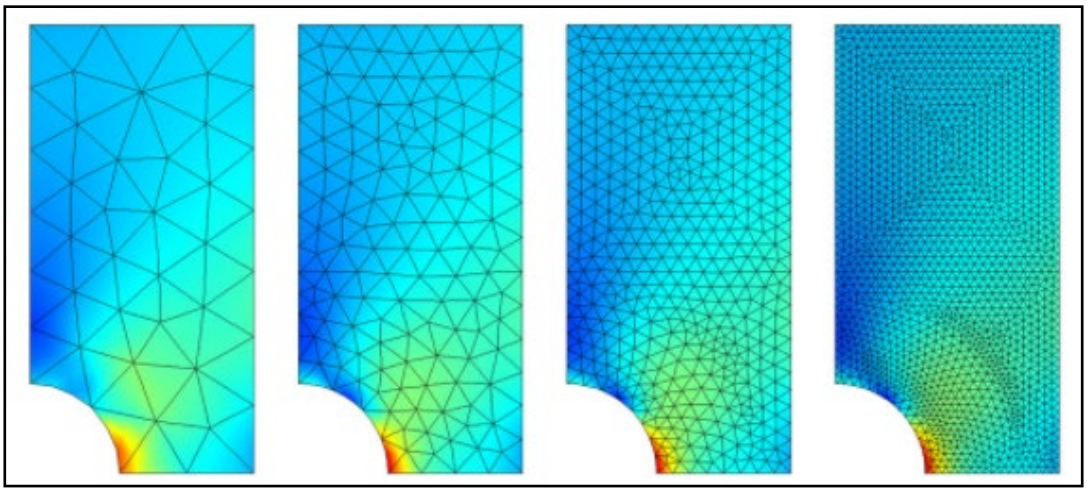


Figure I-6 h-refinement of a plate

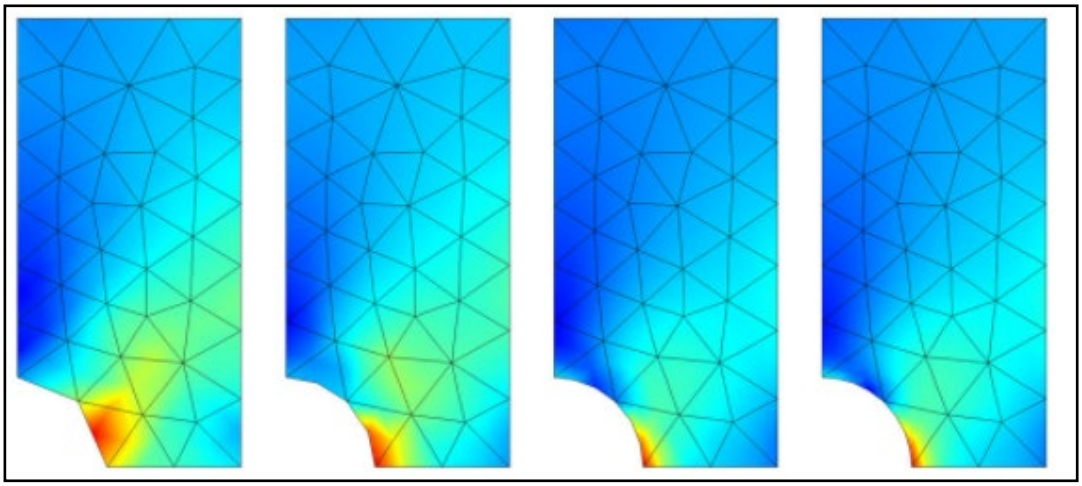


Figure I-7 p-refinement of a plate (COMSOL)

II. Software adopted: Simcenter 3-D and Simcenter Nastran

Introduction

The previously described FEA theory is well implemented inside the finite element environment called Simcenter. In Figure II-1, a common application is shown. This software is provided from Siemens PLM Software and helps in facilitating the development of high-performance and optimized designs. In this section, an overview on the static finite element structural analysis using Simcenter Nastran is given. Starting from the description of the main structures adopted, a typical example of input ASCII text file is commented in all the different sections, according to the Siemens Quick Reference Guide of Simcenter Nastran. This chapter will introduce the reader for a better understanding of the software and data management in the script development.

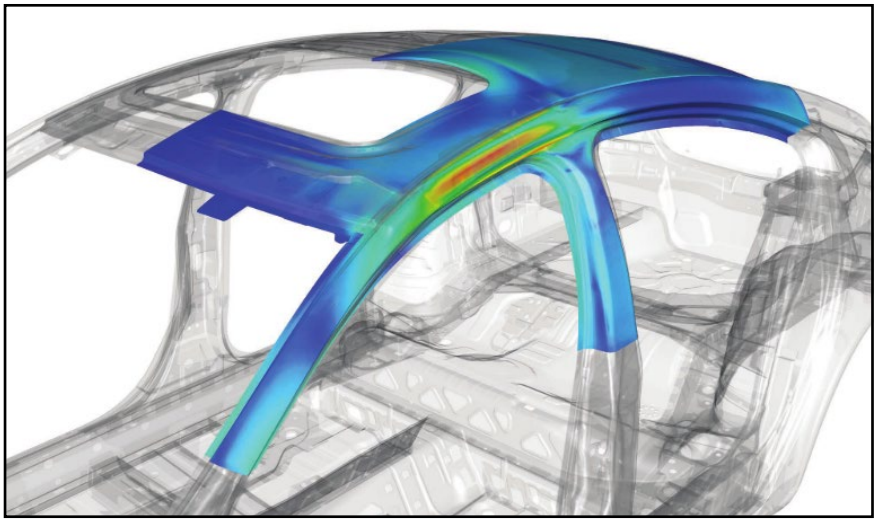


Figure II-1 Structural simulation of a car body on Simcenter Nastran.

Software adopted

According to the Siemens website, *Simcenter 3-D* is defined as

> *"a unified, scalable, open and extensible environment for 3-D CAE with connections to design, 1-D simulation, test and data management."*

Simcenter delivers engineering optimization techniques for achieving the best design criteria. For example, it reduces component weight or finds the combination of parameters to improve product performance through comprehensive topology, geometry, and parameter optimization capabilities.

The software provides a *Graphic User Interface* (*GUI*) as shown in the example below (*Figure II-2*).

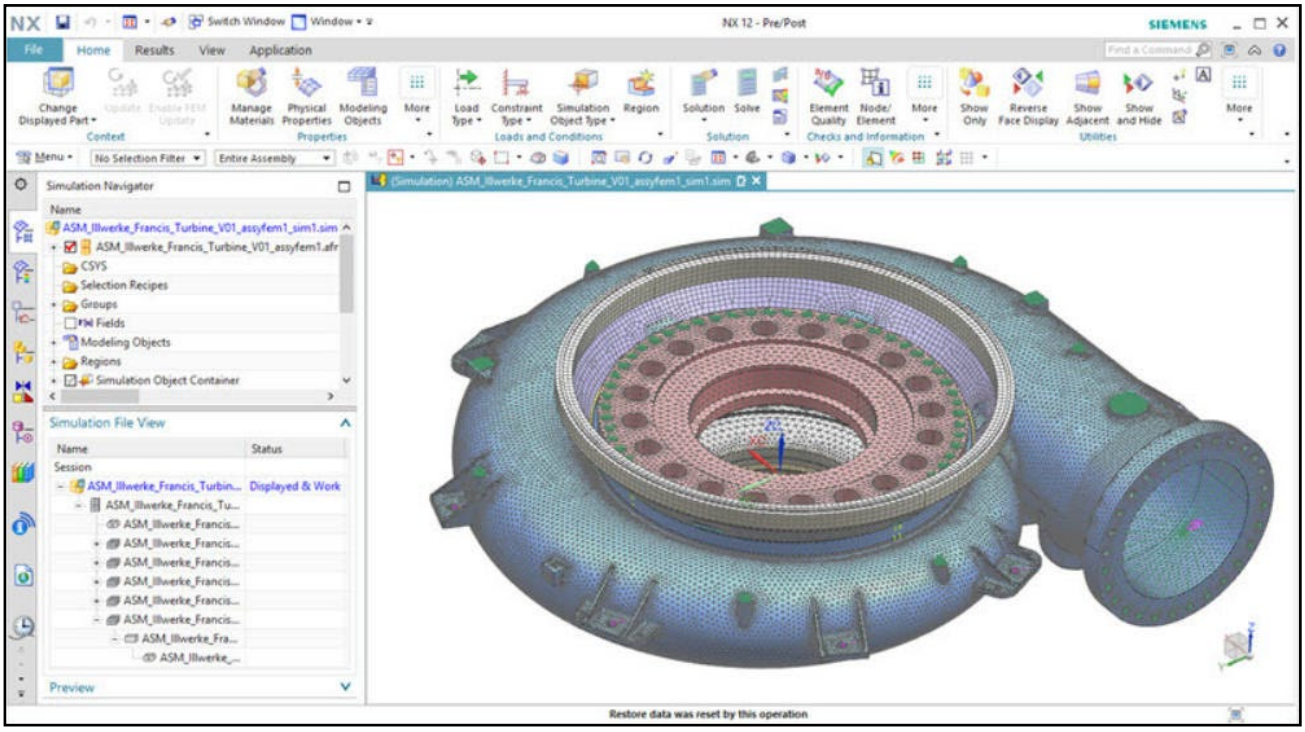


Figure II-2 Simcenter 3-D Graphic User Interface (GUI)

One of the main features is the structural simulation. In this way, it is possible to understand how a generic component reacts under the application of loads, stresses or vibrations. It is a powerful tool in industry for tackling the continuous increase of the material and product complexity. Through the user interface, it is possible to conduct different types of analysis. Linear analysis is used for solving static problems, where deformations are small and linear material assumptions are valid. It can be used for determining if a structure will fail under a prescribed load and can be used to solve transient problems where loads change over time.

Simcenter 3-D is available as a standalone simulation environment. It is also available completely integrated with Nastran delivering a seamless CAD/CAE experience.

Simcenter Nastran is a finite element (FE) solver that can be used to solve most structural analysis problems: linear and nonlinear analysis, dynamic response, rotor dynamics, vibro-acoustics, aero elasticity and optimization. The advantage to having all of these solutions available in a single solver is that input/output file formats are the same for all solution types, greatly simplifying modelling processes.

Nastran file description

Before launching any kind of analysis with Simcenter Nastran, the generation of an input ASCII text file is needed. It includes details on the finite element model regarding geometry, types of elements, materials, loads, boundary conditions and coordinate systems. Additionally, the analysis type needed to perform is specified as well as the type of data output.

A generic input NX Nastran file is organized in five separate sections, as show in *Figure II-3*.

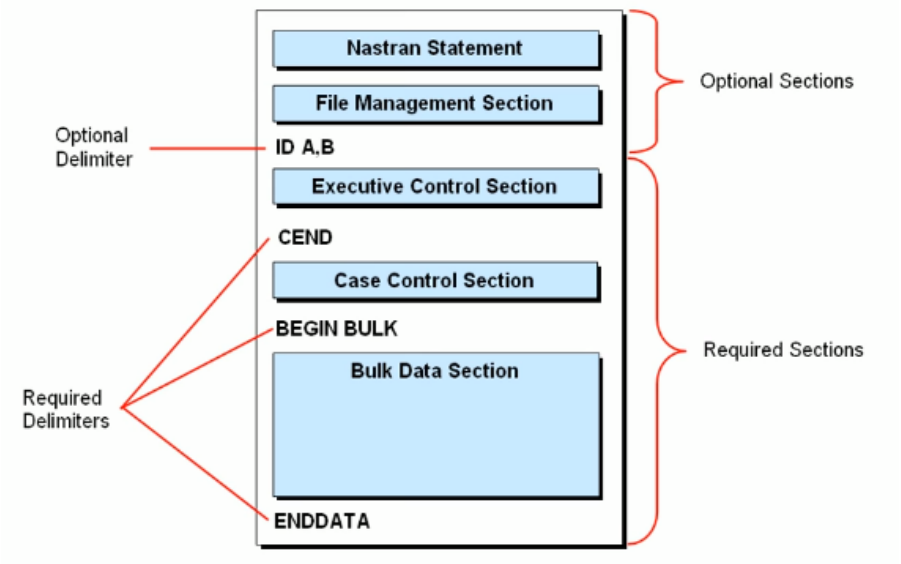


Figure II-3 NX Nastran Input File Structure

Nastran Statement

The Nastran Statement includes operational parameters named system cells used to control internal solution processing or provide specific diagnostic. This section is mostly used for controlling the available memory, data block size and data block parameters.

File Management Section

The File Management Section (FMS) is an optional section of the input Nastran file. It provides the possibility of assigning files, attaching or initializing NX Nastran database sets and FORTRAN files or performing restart analyses.

Executive Control Section

The Executive Control Section is the first required section in the NX Nastran input files. It includes the control of the required solution. Through this section, it is possible to set the specific ID of the job, to specify the type of analysis to be computed and other options (TIME limits, required diagnostics, etc.).

The type of analysis to be performed is specified using the SOL (SOLution) statement. For our type of example, the *Solution 101* is chosen, which is the linear static analysis solution sequence.

Case Control Section

The Case Control Section contains statements for:

- Defining analysis subcases (multiple loadings in a single job execution, *Figure II-4*)

```
CEND
TITLE=Equipment Rack Analysis
ECHO=BOTH
SPC=20
SET 1=1 THRU 50
DISP=1
SUBCASE 1
      SUBTITLE=Dead Load
      LOAD=10
      DISP=All
SUBCASE 2
      SUBTITLE=NW Wind Load
      LOAD=20
      SET 10=2,4,6
      DISP=10
SUBCASE 3
      SUBTITLE=SW Wind Load
      LOAD=30
```

1	Case Control requests above the first subcase level apply to all subcases unless overridden by requests within a subcase.
2	Case Control requests defined within a subcase are limited to that subcase only.

Figure II-4 NX Nastran input files with different subcases

- Setting type of analysis output produced in terms of forces, stresses and displacements
- Selecting output requests
- Definition of titles, subtitles and labels for documenting the produced output

Bulk Data Section

The principal part of the content of the input file is integrated in the Bulk Data Section. After the BEGIN BULK statement, the finite element model in this section is defined in terms of geometry, coordinate systems, finite elements, element properties, loads, boundary conditions and material properties.

Format Bulk Data

The Bulk Data section is composed of entries, of which each entry contains data distributed in fields. Each line contains ten fields. Each field in a particular entry has a pre-defined purpose and data format. Data

can only be integer, real or string. A string has at most eight characters. Reals can be written in several ways; for example, $7.0, .7E1, 0.7+1, .70+1, 7.E+0\ y\ 70.-1$ are the same number. The entry name is given in the first field of the first line. In the example below, a GRID entry example is shown *(Figure II-5)*.

GRID

Grid Point

Defines the location of a geometric grid point, the directions of its displacement, and its permanent single-point constraints.

FORMAT:

1	2	3	4	5	6	7	8	9	10
GRID	ID	CP	X1	X2	X3	CD	PS	SEID	

EXAMPLE:

GRID	2	3	1.0	-2.0	3.0		316		

FIELDS:

Field	Contents
ID	Grid point identification number. (0 < Integer)
CP	Identification number of coordinate system in which the location of the grid point is defined. (Integer ≥ 0 or blank*)
X1, X2, X3	Location of the grid point in coordinate system CP. (Real; Default = 0.0)
CD	Identification number of coordinate system in which the displacements, degrees-of-freedom, constraints, and solution vectors are defined at the grid point. (Integer ≥ -1 or blank *)
PS	Permanent single-point constraints associated with the grid point. (Any of the Integers 1 through 6 with no embedded blanks, or blank*.)
SEID	Superelement identification number. (Integer ≥ 0; Default = 0)

Figure II-5 GRID structure description

Each Nastran input file line contains eighty columns. There are three field formats for entering data in these eighty columns:

- Small Field Format: Each line is divided into ten fields, each field is eight characters long
- Large Field Format: used when the small field format does not provide enough significant digit. This format is used when some applications require a high degree of accuracy. There are ten contiguous fields, each of them is sixteen characters long.
- Free Field Format: fields are separated by commas or blanks.

The Finite Element types are explained inside the Quick Reference Guide of NX Nastran. There are different types of entries supported: *CTRIA3, CTRIA6, CQUAD4, CQUAD8, CTETRA, CPENTA* and *CHEXA* as examples. This kind of element constitutes the mesh of the Body in White, consequently exported from Simcenter 3-D and then acquired into the script.

Usually the input file is indicated through a "*.dat*", "*.bdf*" or "*.blk*" extension. Submitting these types of files to NX Nastran for execution, the software will process the files by launching a batch process and generating, in the end, different output files.

Most of the time, the interested output file is the "*.f06*" file. This type of the file contains the results of the analysis in terms of displacements, stresses, element forces or any diagnostic messages.

III. Static linear analysis and normal modes analysis

Introduction

The NX Nastran solver provides powerful analysis capabilities for linear statics and normal modes analysis. In this section, these two different types of solutions are analyzed in a theoretical way. The first one, so called Static Analysis, is based on a linear relation between forces and displacements. Principally, this type of approach is applicable when the stresses (as shown in Figure III-1) remain in the linear elastic range of the chosen material. In this case, the stiffness matrix is constant and therefore the computation is shorter compared to nonlinear analysis of the same model. For this reason, it is the most adopted solution.

On the other hand, Normal Modes Analysis provides a method for studying the dynamic properties of systems in the frequency domain. Through the evaluation of the normal modes, it is possible to have a reasonably valid expectation in terms of frequency boundaries of our final equivalent car body model, mitigating resonance phenomena.

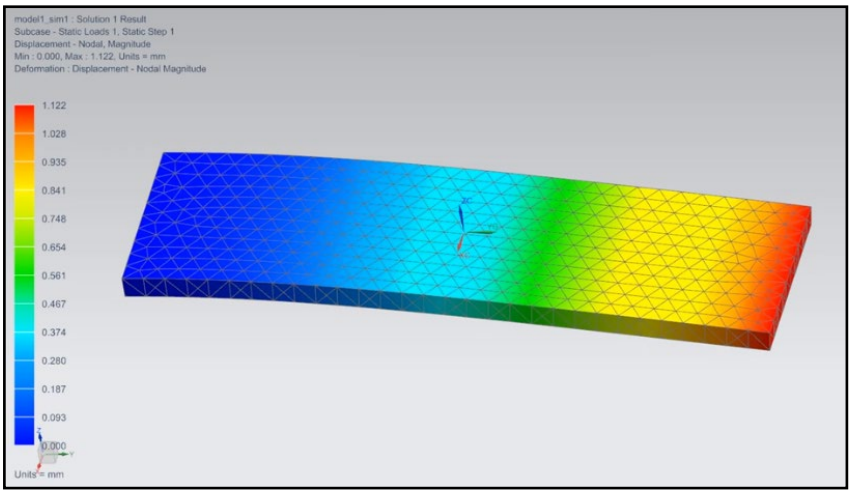


Figure III-1 Linear static analysis of a simple beam in Simcenter 3-D using NX Nastran solver.

Linear static analysis

The application of loads to a generic object determines the deformation of the body. The action of the external forces is transferred all over the body, inducing internal forces inside the body. A new state of equilibrium is therefore reached.

This case represents a typical situation in which Linear Static method can be employed for the calculation of displacements, strains, stresses and reaction forces.

This type of analysis required the following hypotheses:

- ***Static Hypothesis****:* the forces are applied *slowly and continuously* until they reach their complete magnitudes. When this condition is reached, loads must remain constant with respect of time (time-invariant). Thanks to this assumption, it is possible to neglect the effect of inertia and damping forces due to negligible variations in velocity and acceleration inside the body. For example, a body spinning with constant angular velocity or moving with constant acceleration does not generate time variant loads. Therefore, static analysis can be launched also on this kind of problem.

- ***Linearity Hypothesis:*** in *Figure III-2* a stress-strain for ductile materials is reported.

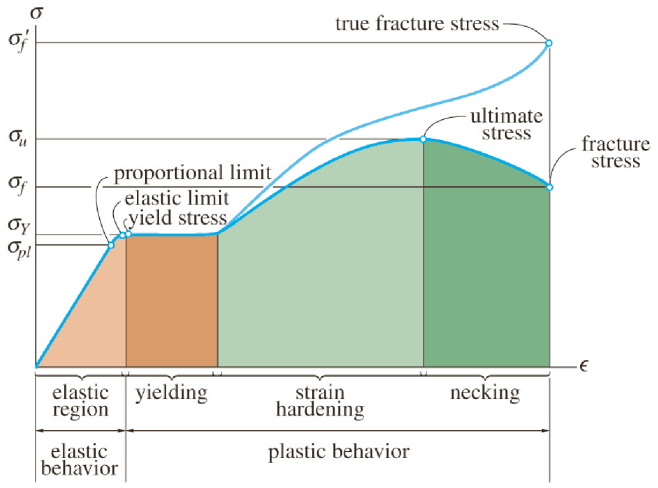


Figure III-2 Conventional and true stress-strain diagrams for ductile material (steel)

The linearity hypothesis is valid in the elastic region where the relationship between loads and induced responses is linear. In this region, the small induced displacement makes negligible the variation in stiffness procured by loadings. In general, when a material complies with Hooke's law *[Equation III-01],* the tensile stress σ $[MPa]$ is directly proportional to axial strain ϵ and the body behaves as a spring subjected to a load (*Figure III-3*):

$$\sigma = E * \epsilon \qquad [III\text{-}01]$$

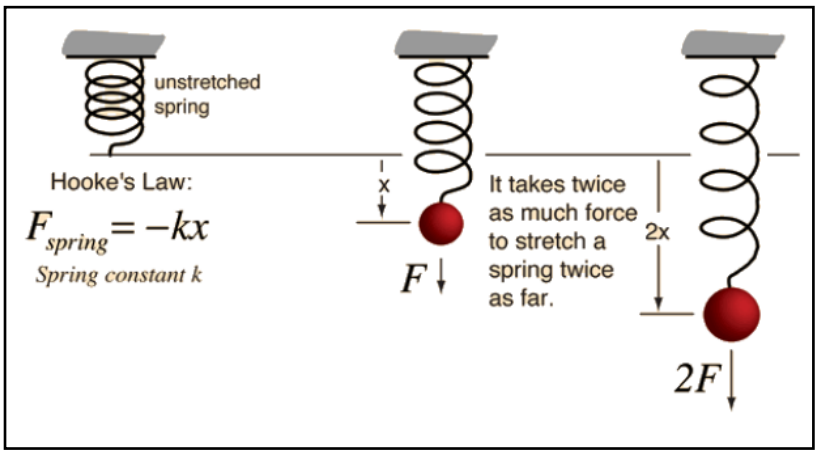


Figure III-3 Deformation of a spring under the application of different loads.

The Young module E $[GPa]$ describes the slope of the curve, as shown in *Figure III-4*. This value describes the mechanical property in terms of stiffness of the solid material treated. Moreover, during the model deformation, the loads must remain constant in magnitude, direction and distribution for avoiding nonlinear effects.

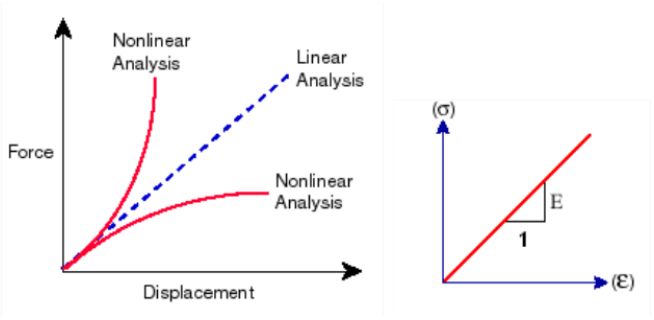


Figure III-4 Displacement - Force relation on different types of analysis.

In a 3-D case, the stiffness of the body is no longer represented by the Young module but with the stiffness matrix *[Equation III-2]*. Therefore, if this hypothesis is valid, the matrix must remain constant.

$$[K]\{u\} = \{f\} \qquad \textit{[III-2]}$$

Where:

[K]: stiffness matrix

{u}: displacement vector

{f}: load vector.

The majority of the analysis performed are linear static, thanks to its simplicity and its validity most of the times. The computational time is relatively small compared to the time required from a non-linear analysis on the same model.

Modal Analysis

Every mechanical system presents the tendency to vibrate at specific frequencies, called natural frequency. As a consequence, the structure tends to assume the same shape associated with the natural frequency. This shape is called mode shape. In the *Figure* below (*Figure III-5*) are shown seven different mode shapes of a cantilever.

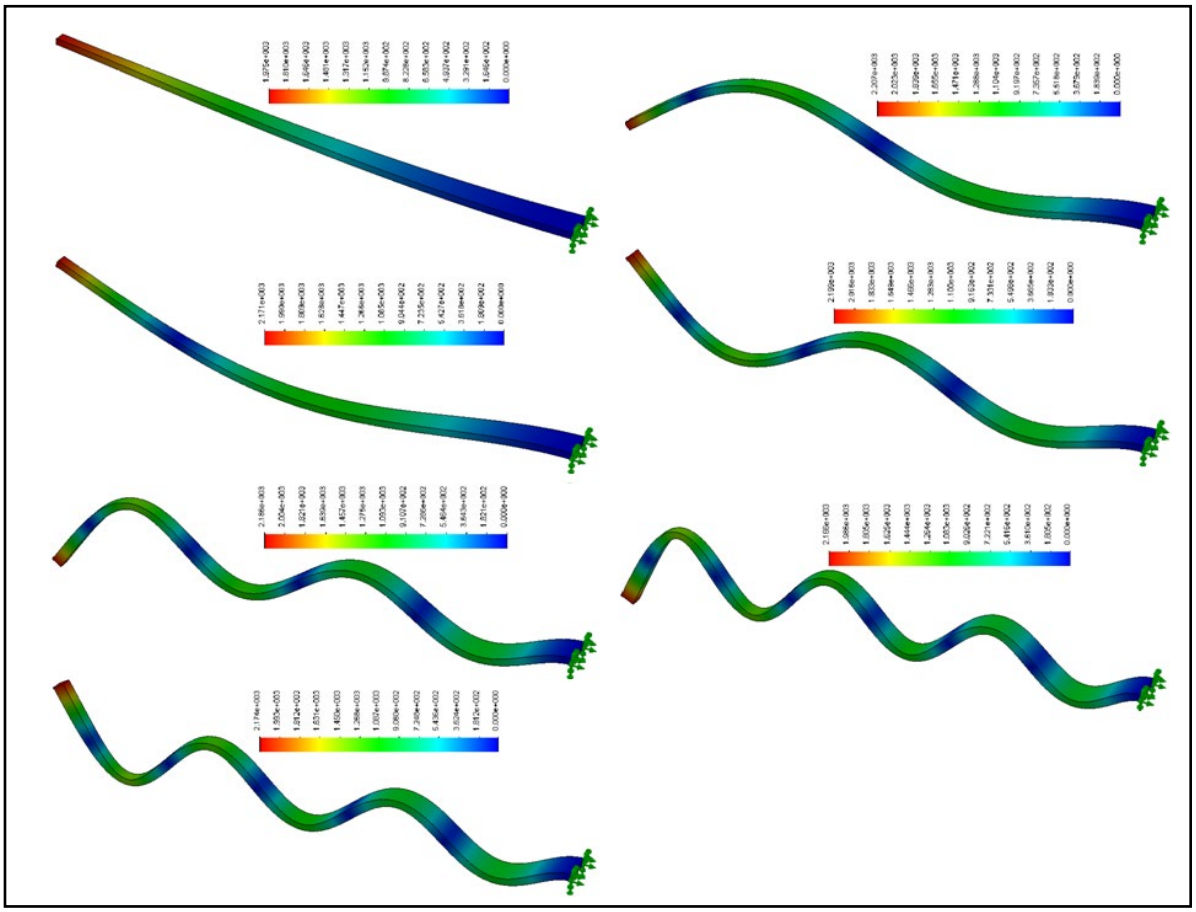


Figure III-5 Modes shape of a cantilever

If the frequency of the applied loads coincides with the natural frequency of the structure, the mechanical system responds with a greater amplitude that causes oscillations. This phenomenon is called resonance. Under these solicitations caused by dynamic load, the structure undergoes large displacements and stresses. The dynamic response of the system is given from the summation of all contributions of the modes considered in the study.

Static analysis cannot predict this type of response. For this reason, it is important to study the dynamic environments in which the design is subjected.

The way in which a structure vibrates depends from the geometry, material properties and boundary conditions. All of these properties are taken into account during the computation of natural frequencies and mode shapes. This analysis is called *normal mode analysis*.

Generally, a real model presents an infinite number of mode shapes. However, a finite element model has a limited number of resonance frequencies equal to the degrees of freedom of the considered model. Most of the times, only the first few modes are considered.

One method for validating a finite element model is by looking at its mode shapes. As a matter of fact, *normal mode analysis* provides a powerful method for evaluating the acceptability of a prototype.

Modal Assurance Criterion (MAC)

This procedure is followed in the conclusive step of the thesis by NX Nastran. It consists in comparing the mode shapes of a real model of a Body-in-White with the mode shapes of the equivalent one composed by joints and beams connected to each other.

According to the Siemens website, the MAC value is defined as:

> *"the normalized dot product of the complex modal vector at each common node (i.e. point)"*

Mathematically this relation is expressed as follows *[Equation III-3]:*

$$MAC(\{\psi_r\},\{\psi_s\}) = \frac{(|\{\psi_r\}^t\{\psi_s\}|^2}{(\{\psi_r\}^t\{\psi_r\})(\{\psi_s\}^t\{\psi_s\})} \qquad [III\text{-}3]$$

Where ψ_s and ψ_r are the complex modal vectors. When the vectors move the same way, a linear relationship exists and the MAC value is near to one. If they are linearly independent, the MAC value will be lower.

In particular, the *Modal Assurance Criterion (or MAC value)* states the similarity of two mode shapes. In case two mode shapes are identical, the MAC will assume the value of 100%. If two mode shapes are different, the MAC will be close to zero. A MAC matrix can express the relation between two mode shape sets (*Figure III-6*).

Auto Modal Assurance Criterion (%) — List Options | Table / Geometry

	Mode No.	Frequency	Mode 1 135.001	Mode 2 304.692	Mode 3 385.467	Mode 4 474.231	Mode 5 554.199	Mode 6 675.410
1	Mode 1	135.001	100.000	3.347	1.157	6.185	64.614	27.940
2	Mode 2	304.692	3.347	100.000	2.926	1.381	2.683	0.468
3	Mode 3	385.467	1.157	2.926	100.000	2.194	1.567	0.761
4	Mode 4	474.231	6.185	1.381	2.194	100.000	1.346	2.889
5	Mode 5	554.199	64.614	2.683	1.567	1.346	100.000	0.130
6	Mode 6	675.410	27.940	0.468	0.761	2.889	0.130	100.000
7	Mode 7	764.601	1.287	1.669	96.018	4.403	3.622	1.751

Figure III-6 This 7x7 matrix contains at each cell the MAC value.

Mode shapes compared to themselves are the diagonal terms and present the MAC value equal to 100%. The non-diagonal terms present a lower MAC value.

Alternately, it is possible to have a graphical representation by means of a histogram as shown in *Figure III-7*. The height of each parallelepiped represents the MAC value.

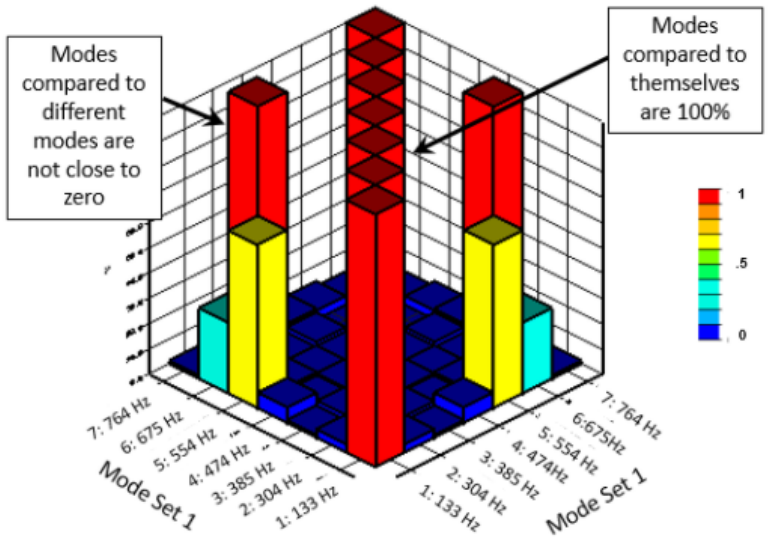


Figure III-7 Experimental modal analysis of a plate

IV. Development and discussion of the “Cutting Tool”

Introduction

The beginning of the process involves the creation of an equivalent car model starting from the full CAD description of the structure of the Body-In-White (Figure IV-1) of a generic vehicle. In this section, the workflow for beams and joints generation is reported as well as all the key points in the script development of a cutting tool on Matlab. Through this program, each beam-type element of the original FEM model is approximated into a new beam (PBEAM or PBEAML), compatible in terms of geometric properties in each specified section. Concept models of beams and joints are created, respectively, by means of a dimensional analysis on area, inertia, centroid and shear-center of beam-member cross-sections.

At the end of the study, a body-parts library is created. It is also defined as a “reference library” for the next applications described in the next chapters.

Figure IV-1 Body Structure

Body-In-White definition

The *Body-in-White (BIW*) is defined as:

> *"the stage in automotive design in which sheet metal components are welded or assembled together in the form of a basic skeleton of the vehicle."*

This stage is performed before painting or before assembling moving parts (doors, hoods, deck lids as well as fenders), engine, chassis sub-assemblies or trim in the frame structure of the vehicle.

Commonly, the term *BIW* refers to the body shell design of an automotive product such as a car and the name derives from the metallic appearance of the model, as shown in *Figure IV-2.*

Figure IV-2 Picture of an automotive Body in White (BIW)

The body frame can be seen as a simplified network of panels, joints and beams connected together. This consideration is crucial for the generation of concept design of a beam-joint like structure in a vehicle FE model.

Concept design of beams and joints

The suggested methodology for car body concept design is based on the consecutive selection which targets regions of vehicle beams or joint layout is identified. The original FE mesh is replaced by concept models that constitute the reduced model of the vehicle.

Through a sufficient number of cross-sections, the detailed mesh is replaced by equivalent beam members. During this operation the original shape and curvature is preserved. Each section is defined on *Simcenter 3-D* by means of local axes system, chosen transversal with respect to the beam direction. By studying the shell elements of each cross-section, the different geometric properties are evaluated through the Cutting Tool script in the *Matlab* environment. The concept beam members are defined as one-dimensional longitudinal elements, accurately described by a *CBEAM* entry in which the connection between different GRID points is described with a specific attitude. These *GRID* points coincide with the shear-center of each section. The mechanical properties, such as inertia and area of the original section, are evaluated and then embedded inside the property beams card (*PBEAML* or *PBEAM)*.

In this way, the new concept beams (*Figure IV-3*) guarantee the same mechanical response of each original component of the BIW.

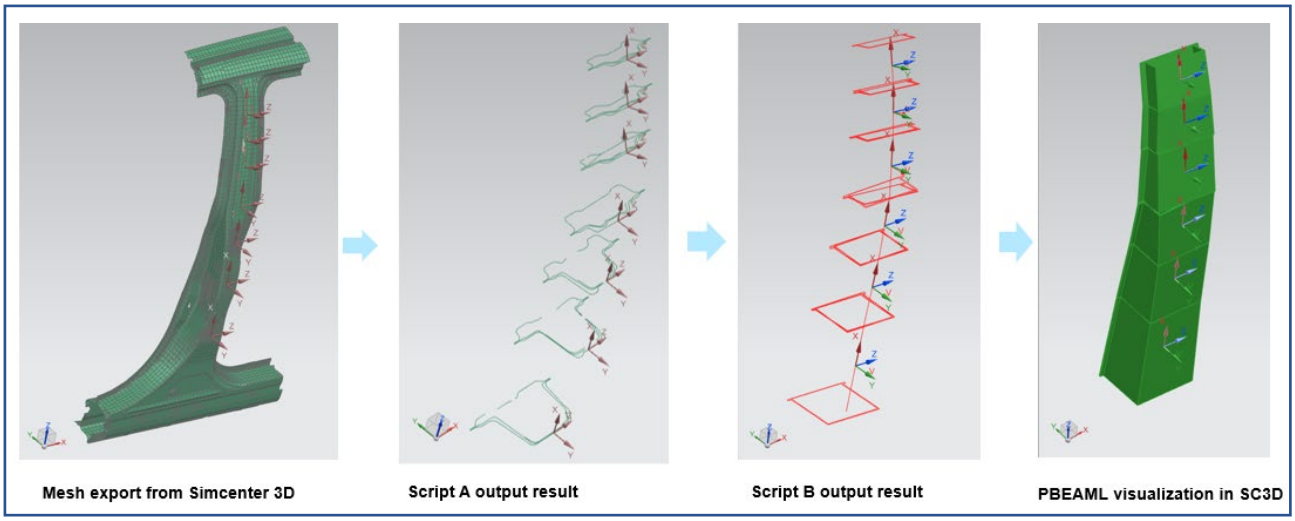


Figure IV-3 Brief graphical overview on concept beam-member creation.

This operation is iterated during the vehicle concept design. At the end of the process, the fully detailed mesh is replaced by a series of equivalent concept beams. The structural continuity is guaranteed thanks to concept joint models that interconnect converging beams at the respective terminal center nodes located at the shear-centers of the previously identified cross sections. A graphical example is shown in *Figure IV-4.*

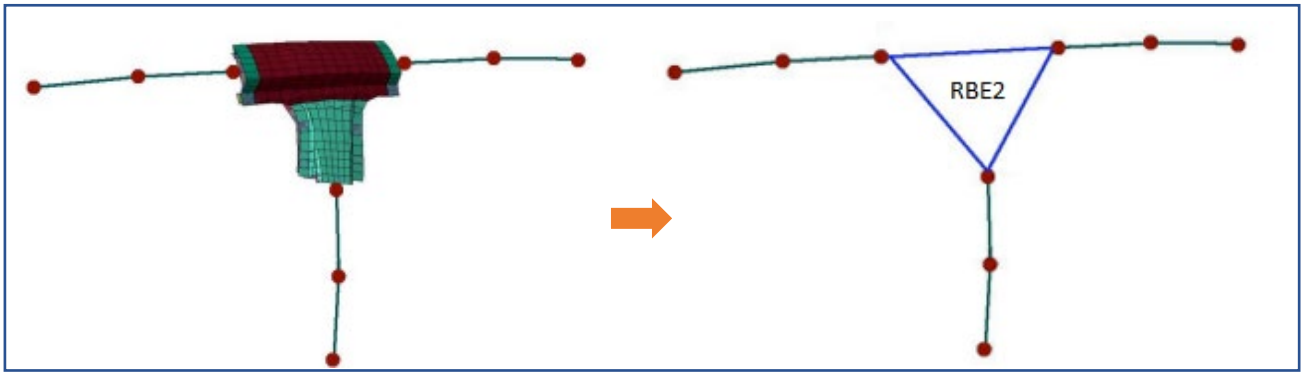


Figure IV-4 Concept joint described by a RBE2 element

In conclusion, an equivalent vehicle body and a reference library are created, as shown in *Figure IV-5.* For verification purposes, the body-parts library is imported on *Simcenter 3-D*.

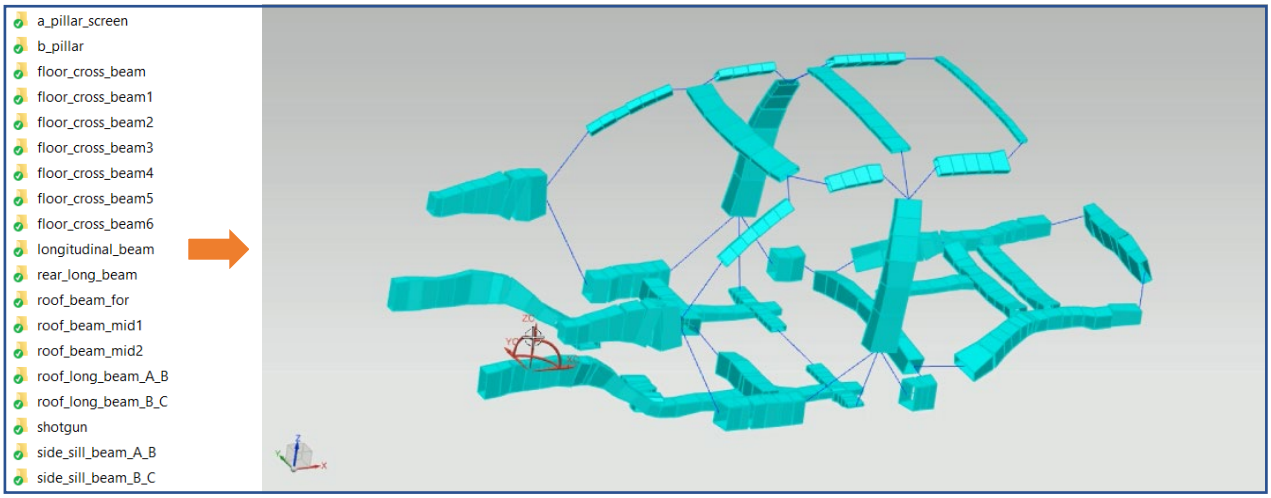


Figure IV-5 Body-part library imported on Simcenter 3-D. Each folder contains PBEAM and PBEAML Nx Nastran cards.

Workflow for beam library creation

In the *Figure* below (*Figure IV-6*), the plan followed is illustrated by means of a concept scheme. The task of *Simcenter 3-D* includes the mesh beams export and the final visualization and check of the results produced. Instead, in the *Matlab* Environment, all the intermediate steps are implemented that lead to the creation of equivalent beams. In order to interconnect the whole structure, joints are manually created by looking at the end nodes of each conceptual beam.

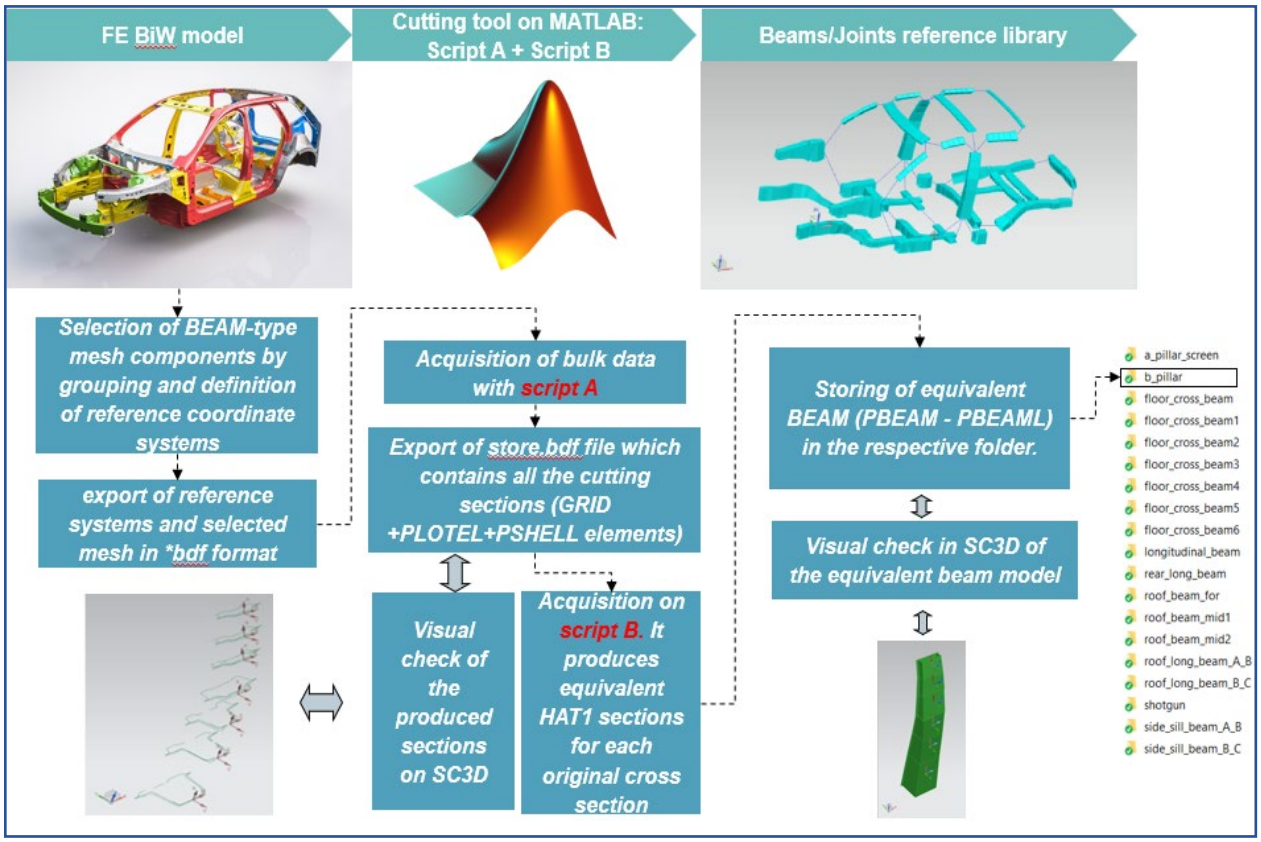


Figure IV-6 General workflow for concept beam generation

The software is divided in two main programs: *Script A* and *Script B*. The first one is responsible for the acquisition of the beam mesh exported from *Simcenter 3-D* comprising the cutting planes and for the creation of the cross section of the selected beam member. The second script instead, works on the previously exported sections for generating equivalent *HAT1* sections, in terms of inertia, area, centroid or

shear-center. The division of the program in two scripts helps in the validation procedure and makes the software stronger and more resistant against possible misbehaviours. Through intermediate checks of the cut section, it is possible to have a graphical idea of the geometric shape of each one. Both scripts follow a step-by-step approach.

Export of a generic beam element

The first phase is the selection by grouping of the beam members from the complete CAD description of the fully detailed BIW on Simcenter 3-D. For example, *Figure IV-7* describes the conceptual selection of the left B-Pillar beam of a vehicle body.

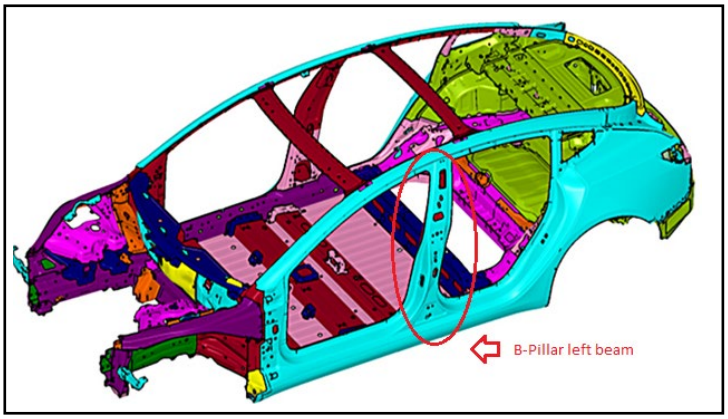


Figure IV-7 Selection by grouping of a B-Pillar beam on a BIW.

The exported mesh of the beam is saved as a "*beam_name.bdf*" file. It is generally composed of:

- *GRID CARDS*: contain all the GRID entries that define all the nodes of the FE model
- *CQUAD4/CTRIA3* elements: all the mesh elements that define the FE model. *CTRIA3* is defined as a triangular plate element connection, *CQUAD4* as a quadrilateral plate element connection.
- *PSHELL* elements: contain all the information regarding the shell element properties

- *MATERIAL CARDS*: defines a material properties class, in terms of Young's modulus, Shear modulus, Poisson's ratio, Mass density, etc. Each group is identified with the respective material identification number (MID).

The type of data format is fundamental for the acquisition and management of data in *Matlab*. For more details about each type of entry, the *Quick Reference Guide of NX Nastran* contains all the entry descriptions treated.

Setup of coordinate reference systems

Each cross-section is defined on Simcenter 3-D with a cutting plane. The cutting plane is defined with a rectangular coordinate system. In particular:

- The YZ plane of the coordinate system is the datum plane. It is inclined perpendicularly with respect to the longitudinal direction of the beam.
- The X axis is normal to the cutting plane and it points to the next cross-section.
- All the cutting planes are approximately equidistant and chosen where there is a change in the profile or in the curvature of the beam.
- A reasonable number of cutting sections leads to an acceptable approximation of the original beam. In the *Figure IV-8*, an example of a cylinder with two cross sections is depicted.

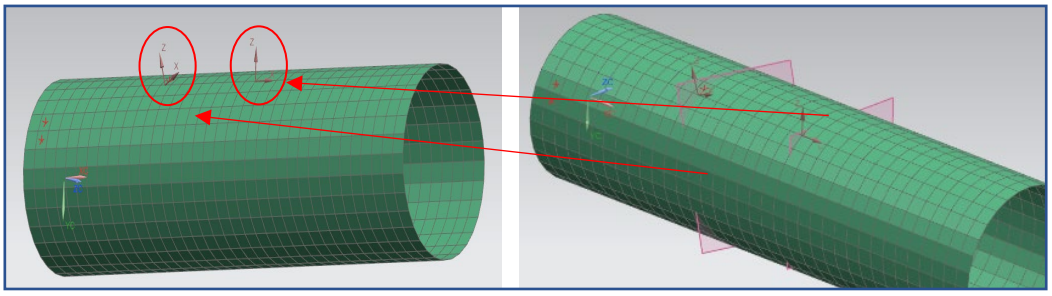
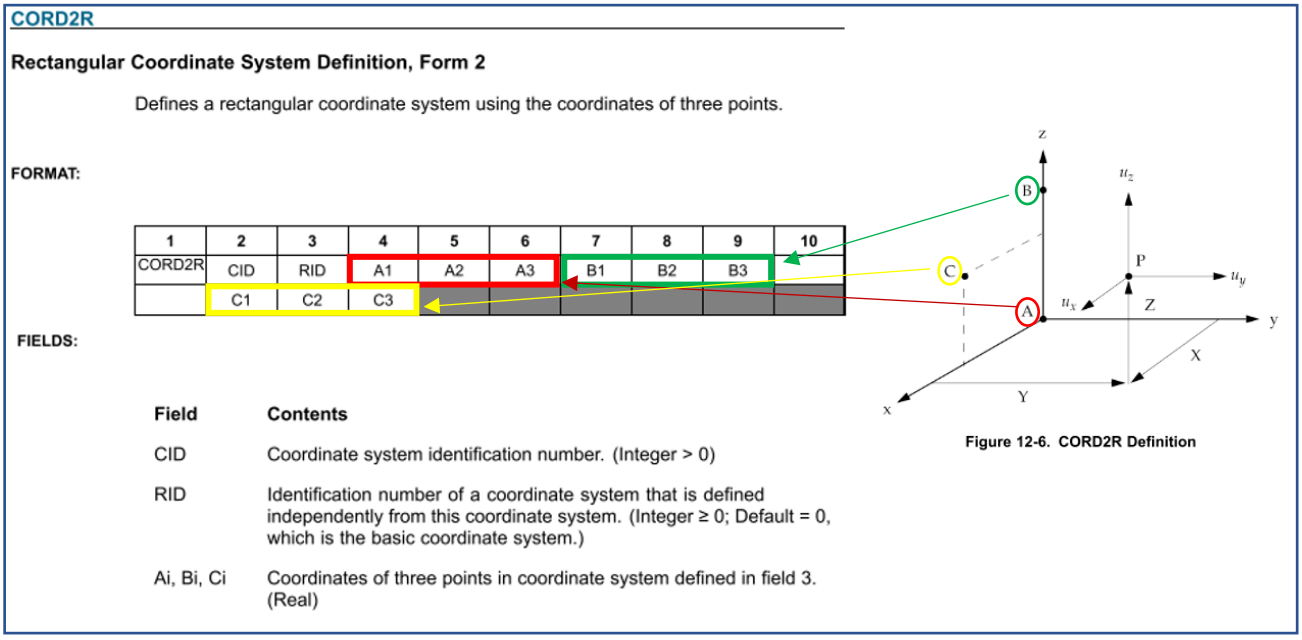


Figure IV-8 YZ Datum planes on Simcenter 3-D

According to the NX Nastran Quick Reference Guide, the spatial location of a plane in the space is determined by three points:

- Point A: coincides with the origin of the reference system
- Point B: is placed on the Z-axis
- Point C: belongs to the X-Z plane

All of these data are embedded inside the respective CORD2R entry, as depicted in *Figure IV-9*.



CORD2R

Rectangular Coordinate System Definition, Form 2

Defines a rectangular coordinate system using the coordinates of three points.

FORMAT:

1	2	3	4	5	6	7	8	9	10
CORD2R	CID	RID	A1	A2	A3	B1	B2	B3	
	C1	C2	C3						

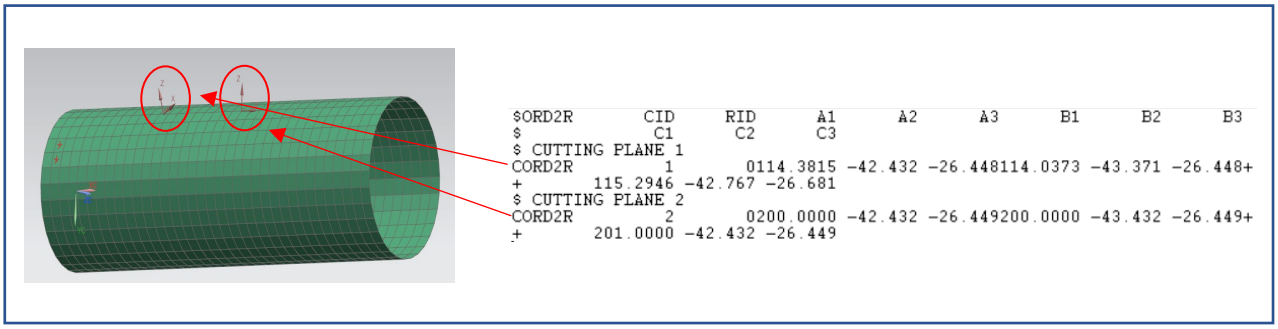
FIELDS:

Field	Contents
CID	Coordinate system identification number. (Integer > 0)
RID	Identification number of a coordinate system that is defined independently from this coordinate system. (Integer ≥ 0; Default = 0, which is the basic coordinate system.)
Ai, Bi, Ci	Coordinates of three points in coordinate system defined in field 3. (Real)

Figure 12-6. CORD2R Definition

Figure IV-9 CORD2R format entry according to the Quick Reference Guide of NX Nastran.

The cutting planes are exported inside an output "*beam_name_CORD2R.bdf*" file (*Figure IV-10*).



```
$ORD2R        CID      RID       A1       A2       A3       B1       B2       B3
$               C1       C2       C3
$ CUTTING PLANE 1
CORD2R          1          0114.3815 -42.432 -26.448114.0373 -43.371 -26.448+
+       115.2946 -42.767 -26.681
$ CUTTING PLANE 2
CORD2R          2          0200.0000 -42.432 -26.449200.0000 -43.432 -26.449+
+       201.0000 -42.432 -26.449
```

Figure IV-10 Cutting planes NX Nastran definition on a cylinder FE examples and output ".bdf" file.

Expected results from Script A

The FE model is made up of interconnections of *CQUAD4* or *CTRIA3* elements. The first task of the script concerns the evaluation of each cross-section of the original cut beam. In *Figure IV-11* the cross-section number one of a cylinder *FE* model is displayed. It is made of a series of section lines interconnected at the intersection points. Each section line is described through the intersection of the cutting plane with one quadrilateral (*QUAD)* or triangular (*TRIA)* element.

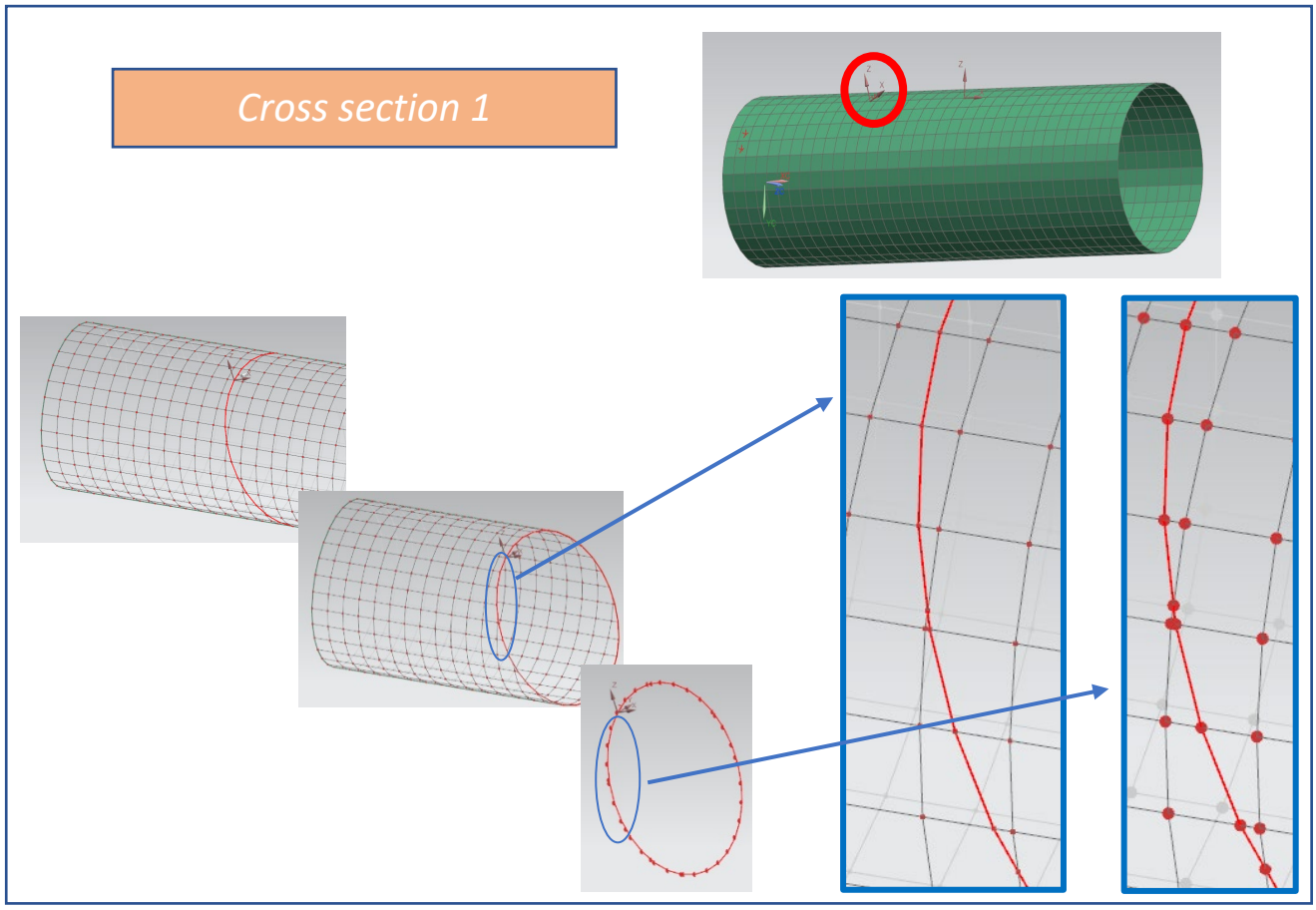


Figure IV-11 Cross-section result

The cutting section comprises:

- Series of Section Points → New GRIDs to be created

- Series of Section Lines → Each Section Line is written as *PLOTEL* (plot element) that connects two *GRIDs* nodes
- Thickness information → Each Section Line has an associated thickness. This value is the thickness of the *QUAD*/*TRIA* element previously cut by the cutting plane. The thickness of the intersected FE element can be found on the *PSHELL* card, as shown in *Figure IV-12.*

1	2	3	4	5	6	7	8	9	10
CQUAD4	EID	PID	G1	G2	G3	G4	THETA or MCID	ZOFFS	

1	2	3	4	5	6	7	8	9	10
PSHELL	PID	MID1	T	MID2	12I/T**3	MID3	TS/T	NSM	

Figure IV-12 Thickness information of a generic CQUAD4 element.

Matlab Script A

The *Matlab Script A* is responsible for the generation of the cross-sections of the FE model exported from *Simcenter 3-D*. As shown in *Figure IV-13*, the code receives two ".bdf" input files: the beam mesh export file and the coordinate reference system. After an estimated time which is based on the complexity of the model (generally no more than ~40 seconds), it generates a "*store.bdf*" output file which contains the cross sections data, described through *PLOTEL* and *GRID* elements.

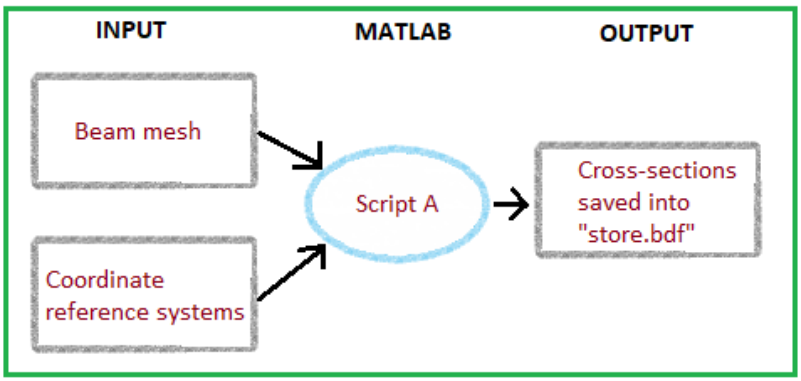


Figure IV-13 Conceptual scheme of Script A

In the following picture (*Figure IV-14*), the *Matlab* code of *Script A* is displayed. It is organized in five steps, well discussed in the next paragraphs. Each of them includes one or more call-back functions, appropriately called by the system during the execution.

STEP1 File opening and model coordinates/structures acquisition + plotting

```
beam_name=input('Please, insert the beam name:\n','s');
filename=strcat('./DIR/',beam_name,'.bdf');
[GRID,CQUAD4,CTRIA3,PSHELL]=Coordinates(filename);
```

STEP2 Local reference system acquisition and calculation

```
filename=strcat('./DIR/',beam_name,'_COORD2R.bdf');
[CORD2R,text]=ReferenceSystemsCoordinates(filename);
[AXIS,InterpolatePlane]=axes_plane(CORD2R);
```

STEP3 Neighborhood elements calculation

```
threshold=10;   % Set the minimum distance btw cutting planes and CQUAD4/CTRIA3 elements.
[CQUAD4_sel,CTRIA3_sel]=CentrumCalculation(CQUAD4,CTRIA3,CORD2R,GRID,AXIS,threshold);
```

STEP 4 Intersection plane-CQUAD4-CTRIA3 elements

```
[PQ,SAVE_EID]=IntersectionFunction(CORD2R,CQUAD4_sel,CTRIA3_sel,CQUAD4,CTRIA3,AXIS,GRID);
```

STEP 5 Store results

```
[OK]=StoringResults(PQ,SAVE_EID,CQUAD4,CTRIA3,PSHELL,GRID,CORD2R,text);
```

Figure IV-14 Script A STEP subdivision.

By means of the command window, it is possible to interact with the script through the selection of the desired beam. After typing the beam name, the program automatically recognizes the *.bdf* files to import (mesh of the beam + relative local reference systems). The script starts to work and updates the user of the current state of execution, as displayed in *Figure IV-15*.

Command Window

```
Please, insert the beam name:
b_pillar_left
STEP1: Starting coordinates and data structure acquistion
GRID CQUAD4 CTRIA3 PSHELL data structure acquired
STEP2: Starting reference system coordinates acquisition
Coordinates acquired
Acquisition of local axis
Local axis acquired
STEP3: Identification of closest CQUAD4/CTRIA3 elements
CQUAD4/CTRIA3 elements acquired
STEP4: Finding intersection plane-CQUAD4-CTRIA3 elements
Intersection points and relative eid acquired
STEP5: Starting storing results
Results stored in store.bdf
```

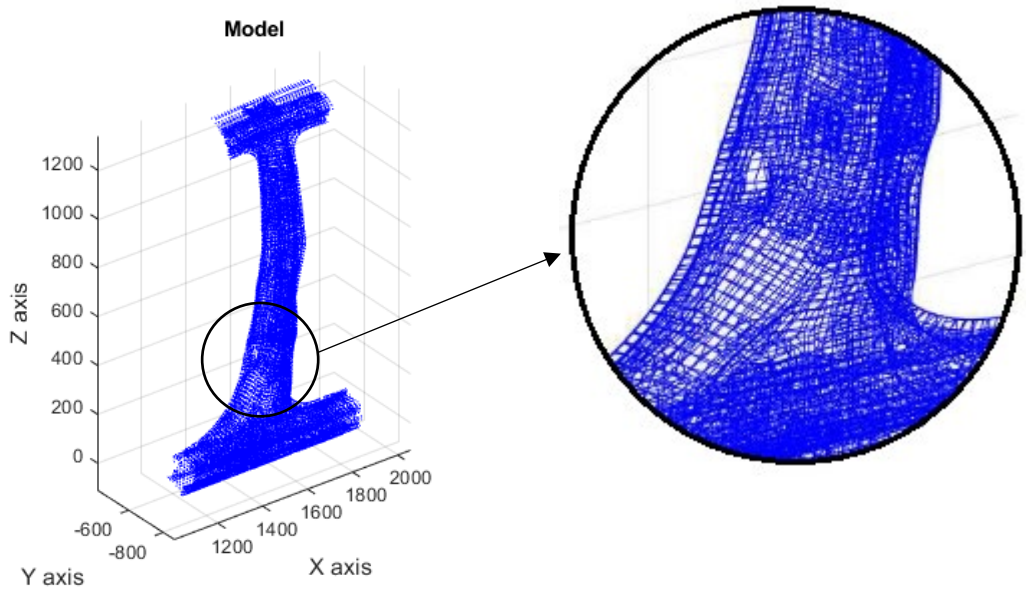
Figure IV-15 Command Window relative to Script A execution

Matlab Script A – STEP 1: File opening and model coordinates acquisition

STEP1 File opening and model coordinates/structures acquisition + plotting

```
beam_name=input('Please, insert the beam name:\n','s');
filename=strcat('./DIR/',beam_name,'.bdf');
[GRID,CQUAD4,CTRIA3,PSHELL]=Coordinates(filename);
```

```
STEP1: Starting coordinates and data structure acquistion
```




```
GRID CQUAD4 CTRIA3 PSHELL data structure acquired
```

Figure IV-16 STEP1 section execution of the Script A and details on mesh.

For demonstration purposes, the B-pillar left is chosen by typing on the command window "*b_pillar_left*". As displayed in *Figure IV-16*, the program retrieves the "*b_pillar_left.bdf*" file, previously stored inside "*DIR*" folder and places it into the main directory. After the beam selection, the system starts to acquire the "*b_pillar_left.bdf*" file.

The input ASCII file is seen as a collection of strings. According to the specific Nastran field format, the "*Coordinates.m*" function recognizes each format field entry. The file is read string by string by means of the function "*fgets*" implemented inside a while loop. At each iteration, a new statement is acquired and temporarily portioned and converted in support variables (strings or vectors). At the end of the loop, the ASCII file is completely read and all the data are stored inside the relative structure array. Hence, the workspace is populated of *CQUAD4, CTRIA4, GRID* and *PSHELL* data structures (*Figure IV-17*). The beam model is plotted by using the function "*plot3*" which interconnects graphically all the *QUAD*/*TRIA* elements.

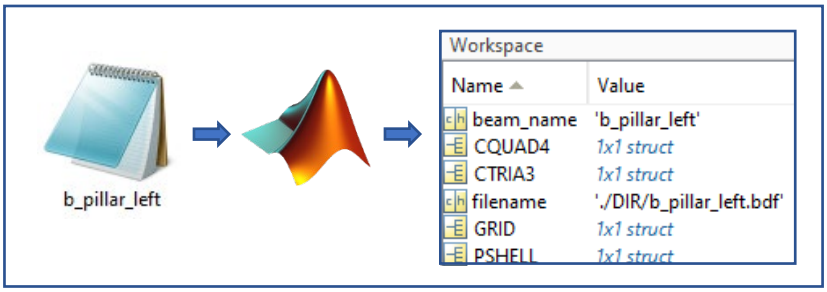


Figure IV-17 Import of b_pillar_left.bdf file in Matlab Workspace after STEP1 execution.

Matlab Script A – STEP 2: Local reference systems acquisition of the cutting planes

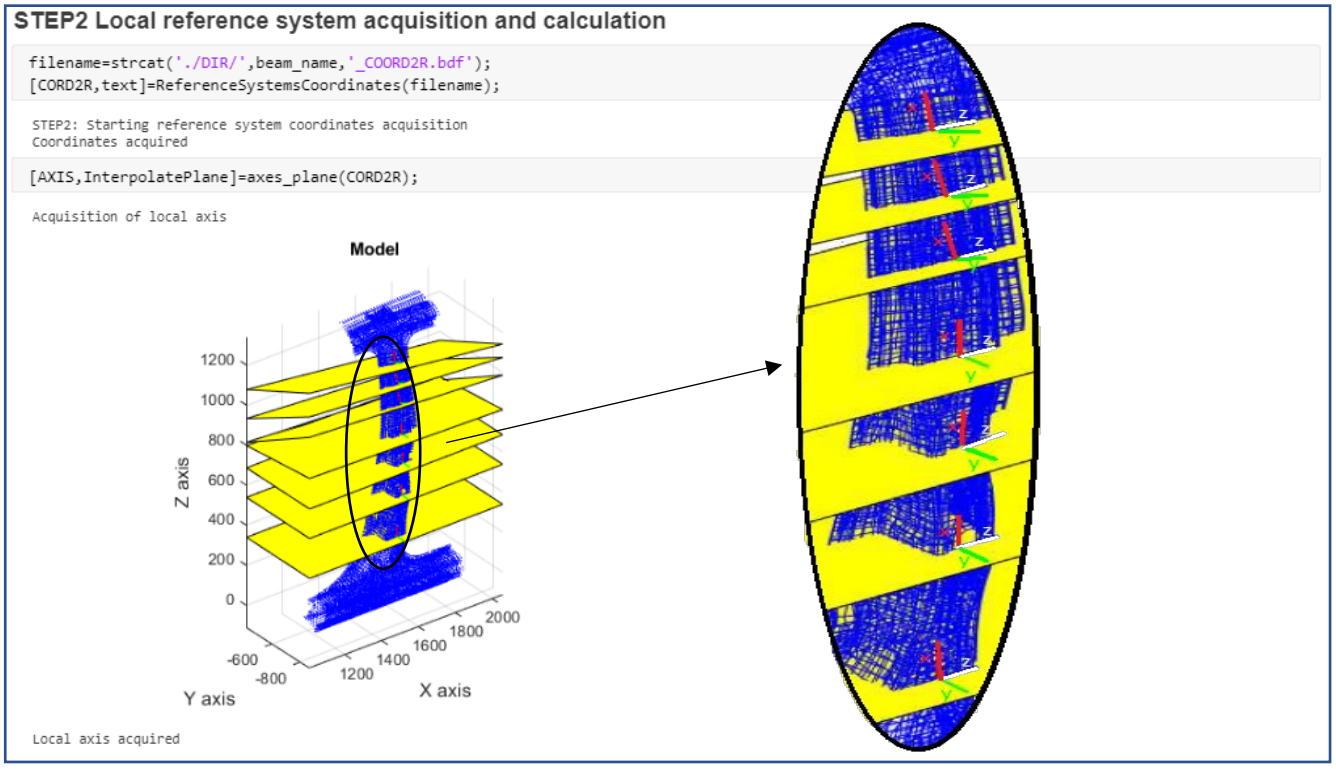


Figure IV-18 STEP2 section execution of the Script A

In this stage, the different local reference frames stored inside "*b_pillar_left_CORD2R.bdf*" are acquired from *Matlab* (*Figure IV-18*). After this operation, the *Matlab* workspace (*Figure IV-19*) is populated of AXIS

and CORD2R structures. These data types contain all the needed information for defining the orientation of each plane in the 3-D space.

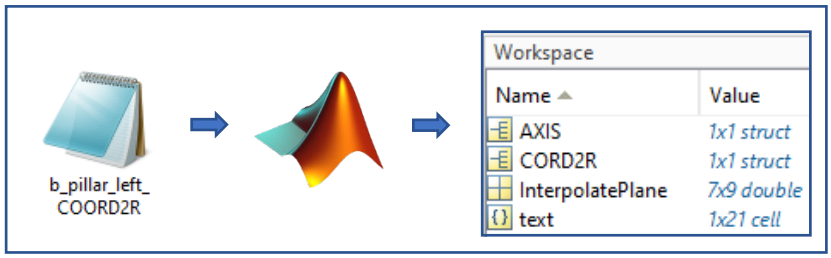


Figure IV-19 Import of b_pillar_left_CORD2R.bdf file in Matlab Workspace.

The function "*ReferenceSystemsCoordinates.m*" is in charge of the reference systems file opening and saving of the relative data structure *CORD2R*. As done in *STEP1*, the function reads string by string and saves all the data inside the output structure. In the second step, the function "*axes_plane.m*" receives input of *CORD2R* and produces the data structure *AXIS*, which contains the 3-D coordinates of the relative axes *x,y,z* of each cutting plane. This operation is done by exploiting cross product operations among the three vectors described by the three points *A, B* and *C* (see the *CORD2R* entry) with respect to the origin *O*. It is also responsible for plotting the cutting planes and all the local axes. As depicted in *Figure IV-20,* all the reference frames are correctly imported from Simcenter 3-D to *Matlab*.

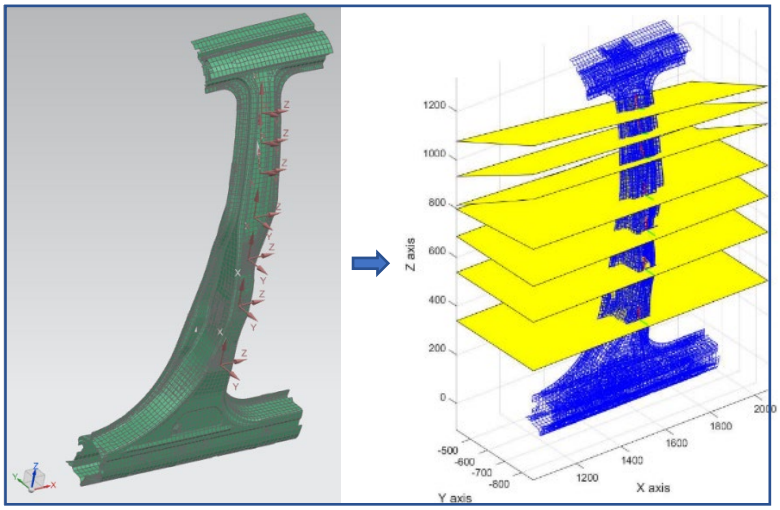


Figure IV-20 B-pillar and local reference system on Simcenter 3-D and Matlab.

Matlab Script A – STEP 3: Local reference systems acquisition of the cutting planes

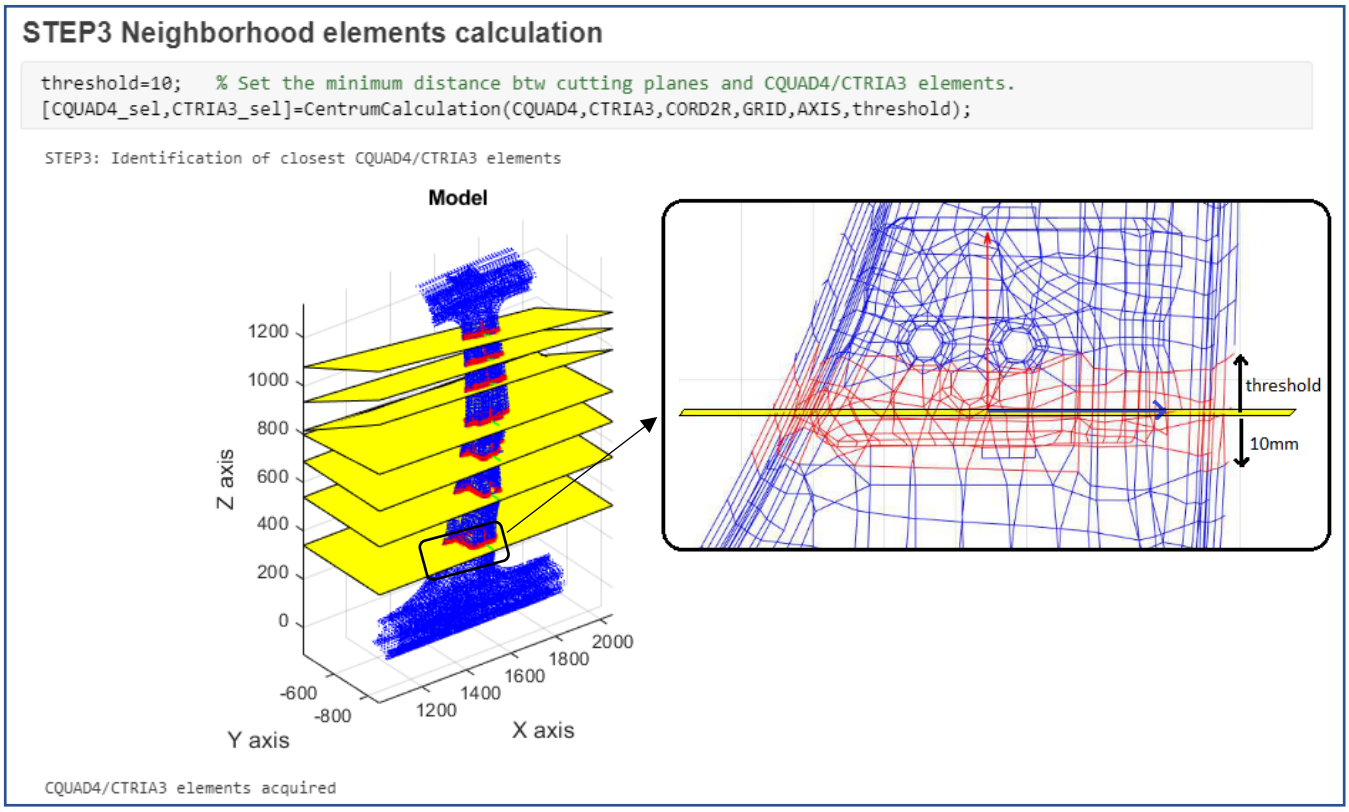


Figure IV-21 STEP3 section execution of the Script A

The aim of the *Script* is the successive creation of cross-sections, given from the intersection of the *QUAD*/*TRIA* elements and each cutting plane. The process is sped up by considering all the "neighbourhood" mesh elements to each cutting plane. In this manner, during the identification of each point intersection, the time consumption is reduced and the general performance of the code results improved. This is done in *STEP3* (*Figure IV-21*) by means of the consecutive assessment of mesh regions for each cutting plane through the setting of a threshold value, expressed in millimeters. This number represents the minimum distance to consider between cutting plane and mesh elements centers, in both positive and negative directions of the normal of the plane (*x-axis*). To avoid possible misbehaviours, the threshold value is set at least double the mesh size.

Input values for the function "*CentrumCalculation.m*" are the *threshold* values and *CQUAD4, CTRIA3, CORD2R, GRID, AXIS* data structure. At each cycle, the external for-loop cycle sets a different plane. The inner for-loop cycle, instead, selects a different mesh element (quad/tria) at each iteration. For each type of element (quad/tria), the distance *OP* between the correspondent geometric center (*P*) and the origin of the cutting plane (*O*) is considered, as shown in *Figure IV-22*.

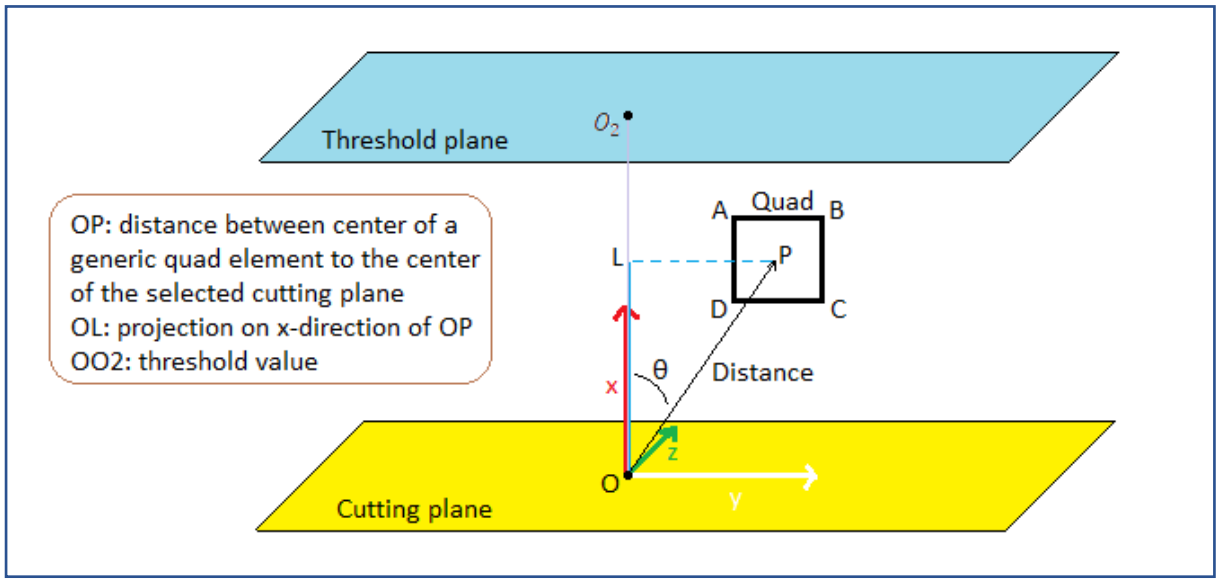


Figure IV-22 Geometric description of dot-product projection of the generic center distance of a quad element.

By exploiting the dot product relation, the angle θ between the vector ***OP*** and the unitary **x** vector of the selected plane is evaluated, as shown in *Equation IV-1* and *Equation IV-2*.

$$\boldsymbol{x} * \boldsymbol{OP} = |\boldsymbol{x}||\boldsymbol{OP}|\cos(\theta) \qquad [IV\text{-}1] \qquad \rightarrow \qquad \theta = arcos(\frac{\boldsymbol{x}*\boldsymbol{OP}}{|\boldsymbol{x}||\boldsymbol{OP}|}) \qquad [IV\text{-}2]$$

At this point, the ***OP*** vector is projected on the x-direction by multiplying for the cosine of θ into a new vector called ***OL*** (*Equation IV-3*). Exploiting the *Matlab* function "*norm*", the magnitude of the ***OL*** vector is finally evaluated (*Equation IV-4*).

$$\boldsymbol{OL} = \boldsymbol{OP} * \cos(\theta) \qquad [IV\text{-}3] \qquad \rightarrow \qquad On\ Matlab:\ \ OL = norm(\boldsymbol{OL}) \qquad [IV\text{-}4]$$

In those cases, in which the norm of the projection (*OL*) is greater than the selected threshold (OO_2), the quad/tria element is considered "far" from the plane and hence it is not considered. In the opposite case,

the mesh element is accommodated on dedicated support vectors. At the end of the for-loop cycle, these support arrays are stored inside *CQUAD4_sel* or *CTRIA3_sel* data structure. These output structures (displayed in *Figure IV-23*) represent two subsets of the original *CQUAD4* and *CTRIA3*.

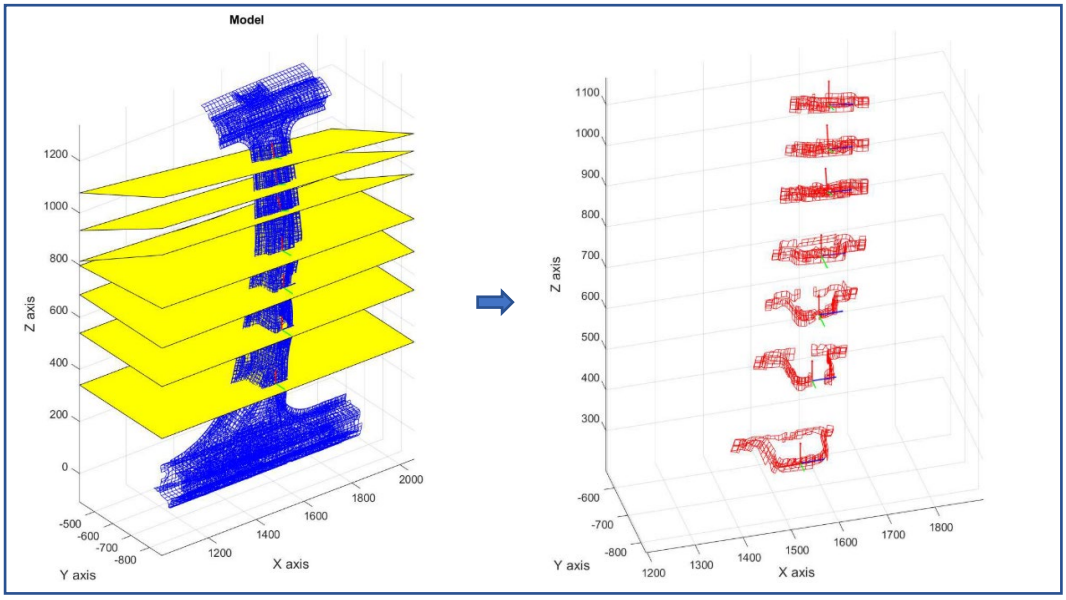


Figure IV-23 Inputs and outputs of STEP3 section code.

Matlab Script A – STEP 4: Cross-section points evaluation

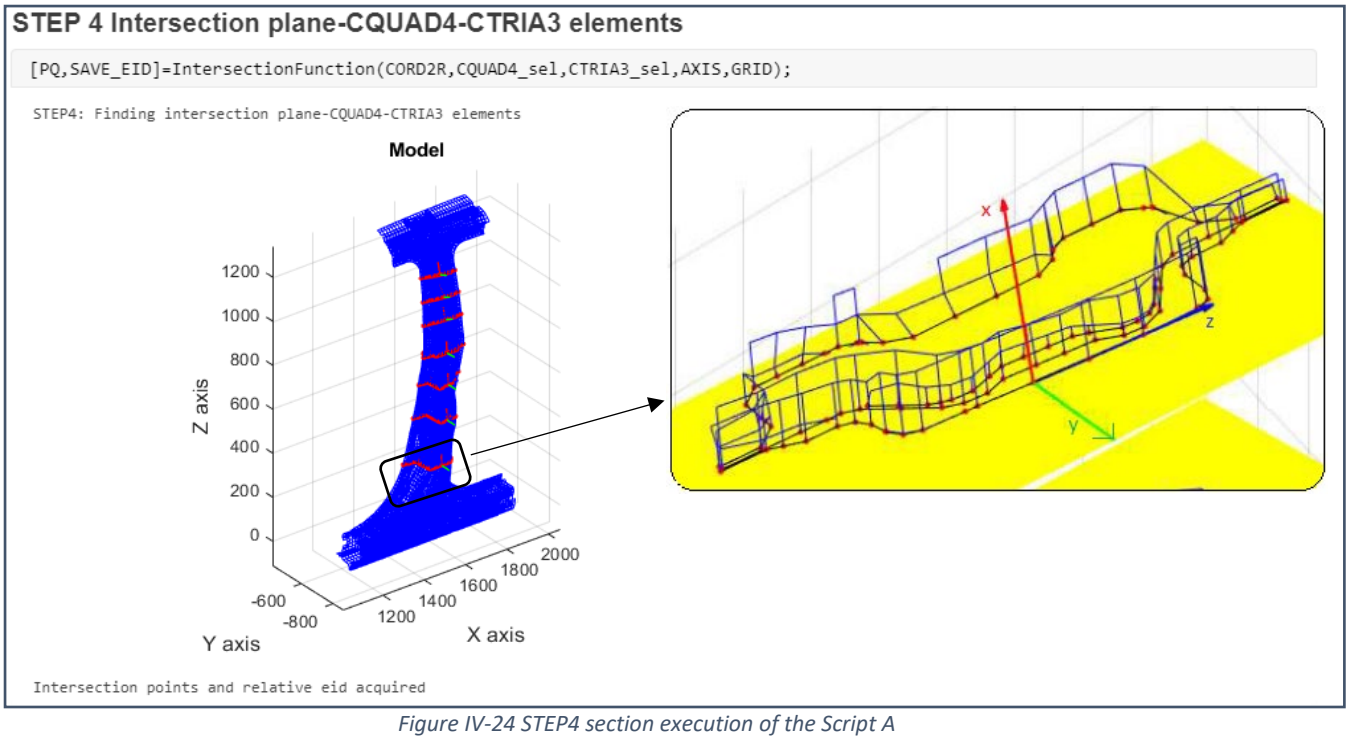


Figure IV-24 STEP4 section execution of the Script A

As shown in *Figure IV-24*, the main algorithm for finding the cross-sections is implemented. Each *QUAD/TRIA* element can be described as the interconnection of lines. The function called "*IntersectionFunction.m*" exploits this consideration for finding the intersection points between each cutting plane and the mesh of the original model. An external for-loop cycle evolves through all the planes with an index "*k*". At each iteration, a *k-th* plane is created by the function "*createPlane.m*", which acquires the points *A, B* and *C* belonging to each plane. The inner loop is responsible for the generation of the intersection point. At each instance of this cycle, a new mesh (*QUAD/TRIA*) is selected from the *CQUAD4_sel* or *CTRIA3_sel* data structure. The extremes geometric vertices are connected through line elements which are produced by means of the function "*createLine3D.m*". Therefore, the intersection between the planes and each line is retrieved through "*intersectLinePlane.m*" which finds the intersection point between a 3-D line and the *k-th* plane. All of those intersections which do not belong to the plane are not saved inside the support matrix *P* which accommodates all the remaining cutting points.

At the end of the loops, two structures are created. The structure "*PQ*" contains the points which describe each cross-section (*Figure IV-25*). Each point is expressed through 3-D coordinates. The structure "SAVE_EID" contains the *element identification number* (*EID*) relative to the quad/tria element intersected. This information is necessary for retrieving the respective thickness of each section segment.

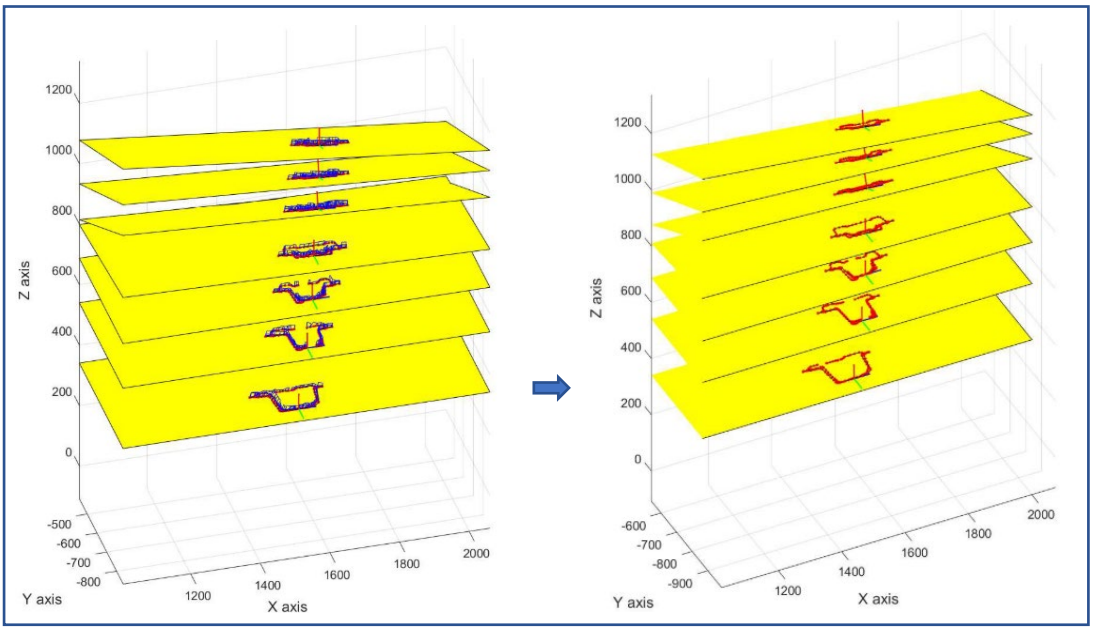


Figure IV-25 Inputs and outputs of STEP4 section code.

Matlab Script A – STEP 5: Storing the results

STEP 5 Storing results

```
[OK]=StoringResults(PQ,SAVE_EID,CQUAD4,CTRIA3,PSHELL,GRID,CORD2R,text);

STEP5: Starting storing results
Results stored in store.bdf
```

Figure IV-26 STEP 5 section execution of the Script A

At this phase, the *Script A* translates all the relevant data on an output "*store.bdf*" file, as shown in *Figure IV-26*. The function "*StoringResults.m"* is responsible for the creation and writing of the ASCII file. The *Matlab* function "*fopen('store.bdf','w')"* is in charge of generating the file whereas the function-call "*fprintf*" writes all the lines row-by-row. At the end of this step, this *Nastran* file will describe all the cutting planes and cross-sections in terms of *GRID*, *PLOTEL*, *PSHELL* and *CORD2R* elements, as depicted in *Figure IV-27.* Every *PLOTEL* element delineates the connection between two new *GRID* points. Each result is directly associated with the respective *PSHELL* element, which describes the correspondent thickness.

```
BEGIN BULK
GRID      626258       0 1607.05 -629.28 1093.56       0
GRID      626259       0 1606.73 -628.90 1093.45       0
.
.
GRID      627838       0 1495.35 -808.20  339.94       0
GRID      627839       0 1498.86 -808.20  339.94       0
PLOTEL         1  626258  626259
.
.
PLOTEL       791  627838  627839
PSHELL         1     100    1.20
.
.
PSHELL       791     100    0.60
$ CUTTING PLANE 1
CORD2R         1       0 1624.33-673.206 1109.82 1625.32-673.294 1109.74+
+        1624.44-672.823 1110.74
$ CUTTING PLANE 2
CORD2R         2       0 1613.30-708.870 1028.01 1614.29-708.930 1027.97+
+        1613.36-708.442 1028.91
$ CUTTING PLANE 3
CORD2R         3       0 1602.48-739.712944.3015 1603.48-739.760944.2693+
+        1602.53-739.256945.1901
$ CUTTING PLANE 4
CORD2R         4       0 1580.27-775.249816.2630 1581.27-775.281816.2409+
+        1580.29-775.158817.2585
$ CUTTING PLANE 5
CORD2R         5       0 1569.79-795.030686.7480 1570.79-795.029686.7480+
+        1569.79-795.030687.7480
$ CUTTING PLANE 6
CORD2R         6       0 1551.22-811.194539.4340 1552.22-811.193539.4340+
+        1551.22-811.194540.4340
$ CUTTING PLANE 7
CORD2R         7       0 1527.44-808.171339.9430 1528.44-808.170339.9430+
+        1527.44-808.171340.9430
ENDDATA
```

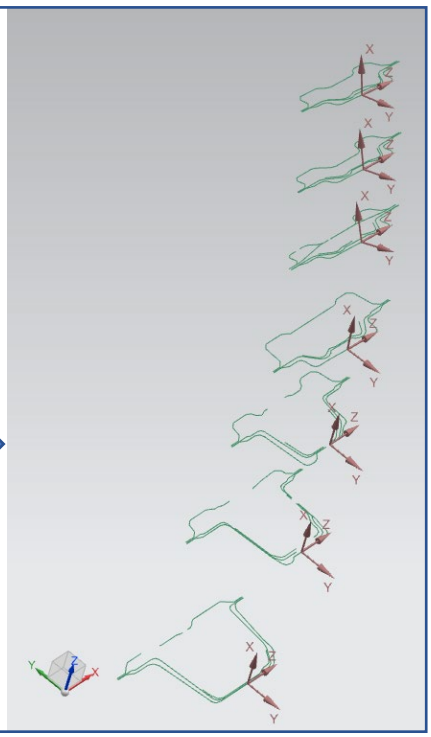


Figure IV-27 Import of the output NASTRAN "store.bdf" on Simcenter of a B-pillar beam.

Matlab Script B

The *Matlab Script B* is responsible for the creation of an equivalent beam, beginning from the geometric knowledge of the cross-sections in the space of the original model.

In the following illustration (*Figure IV-28*), the *Matlab* code of Script B is indicated. It is structured in nine steps and well discussed in the next paragraphs.

STEP1 File opening and model coordinates/structures acquisition + plotting

```
start=str2num(input('Please, insert the beam GRID ID:\n','s'));
[GRID,PLOTEL,PSHELL,CORD2R]=Coordinates_bdf('store.bdf');
```

STEP2 Axes calculation

```
disp("STEP2: Local axes calculation")
[AXIS,InterpolatePlane,PLANE]=axes_plane(CORD2R);
```

STEP 3 Connecting GRID POINTS and finding the associated thickness at each PLOTEL element

```
[PQ,T]=SortingPLOTEL(GRID,PLOTEL,CORD2R,PSHELL,InterpolatePlane);
```

STEP 4 Evaluation of centroid and area

```
[CENTROID,AREA] = Centroid_Area(CORD2R,PQ,T);
```

STEP 5 Expressing points wrt cutting planes RFs

```
[P] = Coordinates_Transformation(CORD2R,PQ,AXIS,CENTROID);
```

STEP 6 Properties Calculation

```
[I] = GeomProperties(CORD2R,P,T,CENTROID);
```

STEP 7 Equivalent section creation

```
[DIM_HAT1] = DIM_SECTIONS(P,I,AREA,CORD2R);  %% Starting from inertia --> find dimension box - hat1
```

STEP 8 Shear center calculation

```
[SCENTER,DELTA] = ShearCenter_(DIM_HAT1,CENTROID,PLANE,CORD2R);
```

STEP 9 Storing

```
[OK]=StoringResultsPBEAM_hat1_adapted(I,CENTROID,AREA,AXIS,DIM_HAT1,SCENTER,DELTA,start); %% saves the cbeam hat1
[OK]=StoringResultsPBEAML_hat1_adapted(I,CENTROID,AREA,AXIS,DIM_HAT1,SCENTER,DELTA,start); %% saves the cbeam hat1
```

Figure IV-28 Script B STEP subdivision

As shown in *Figure IV-29,* the code receives as input the "*store.bdf*" relative to the selected beam. After the main geometric properties (area, inertia, centroid, shear-center) are estimated, two types of output files are produced: "*beam_name_PBEAM.bdf*" and "*beam_name_PBEAML.bdf*". Each of them contains a different 3-D description of the concept beams.

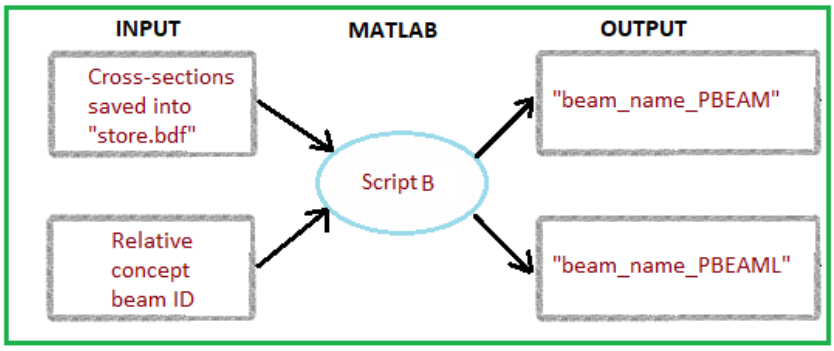


Figure IV-29 Conceptual scheme of Script B

The script starts to work and to update the user of the current state of execution, as displayed in *Figure IV-30*.

```
Command Window
  Please, insert the beam GRID ID:
  710001
  STEP1: Starting coordinates and data structure acquistion
  GRID PLOTEL CORD2R PSHELL data structure acquired
  STEP2: Local axis calculation
  Acquisition of local axis
  Local axis acquired
  STEP3: Connecting GRID POINTS and finding the associated thickness at each plotel
  Points and thichkness data structure saved
  STEP4: Evaluation of centroid coordinates and area of each cutting section
  Centroid coordinates and area acquired
  STEP5: Expressing all the section points wrt centroid of each relative cutting section
  Coordinates transformed.
  STEP6: Evaluation of inertia wrt local y and z axes relative at each centroid of each cutting plane
  Inertia acquired
  STEP7: Creating the equivalent HAT1 sections
  Equivalent HAT1 structure created
  STEP8: Calculating the shear center position wrt the absolute reference frame and the relative
  displacement DELTA wrt the centroid
  Shear center position and Delta acquired
  STEP9: Starting storing results
  Results stored in cbeams_hat1_PBEAM.bdf
  STEP9: Starting storing results
fx Results stored in cbeams_hat1_PBEAML.bdf
```

Figure IV-30 Command Window relative to Script B execution

Matlab Script B – STEP 1: Setting of beam GRID ID and acquisition of "store.bdf"

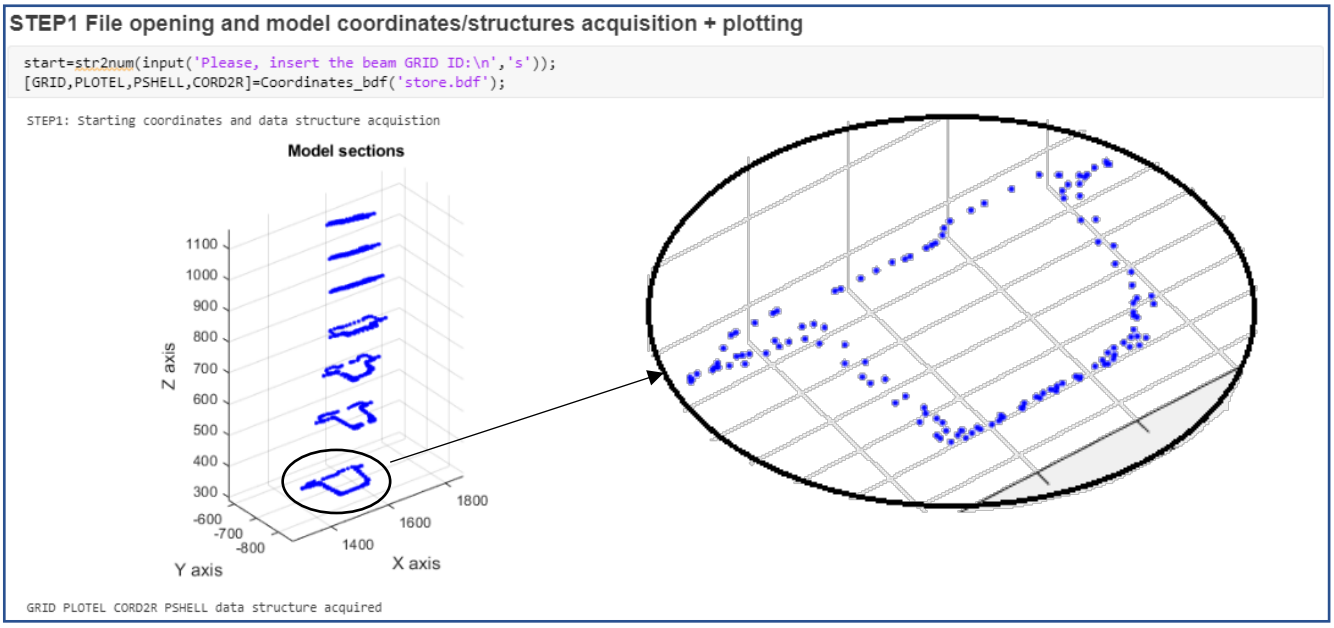


Figure IV-31 STEP 1 section execution of the Script B.

After the "run" of the simulation in *Matlab* (*Figure IV-31*), the script interacts with the user via the command window. As displayed in *Figure IV-32,* the script requests the specific starting identification number (ID) to assign to the concept beam. This type of organization avoids superimposing identification number definitions in the whole concept car model. Each number is chosen a-priori and it is unique with respect of each single beam. This ID can be retrieved by looking at the table present on the "*Organization_concept_beams.xls*" Excel file (*Figure IV-33*), where all the concept beam IDs are organized according to the respective numeric range.

Figure IV-32 Interaction with the command window after the execution of Script B. The ID of the "b_pillar_left" is selected.

LEFT SIDE	Starting ID	CENTRAL SIDE	Starting ID	RIGHT SIDE	Starting ID
a_pillar_screen_left	710011			a_pillar_screen_right	720011
b_pillar_left	710001			b_pillar_right	720001
		floor_cross_beam	706011		
floor_cross_beam1_left	714001			floor_cross_beam1_right	724001
floor_cross_beam2_left	715001			floor_cross_beam2_right	725001
floor_cross_beamx_left	714301			floor_cross_beamx_right	724301
		floor_cross_beam3	706001		
		floor_cross_beam4	707001		
		floor_cross_beam5	708001		
		floor_cross_beam6	709001		
longitudinal_beam_left	713001			longitudinal_beam_right	723001
rear_longitudinal_beam_left	718001			rear_longitudinal_beam_right	728001
		roof_beam_mid1	705201		
		roof_beam_mid2	706201		
		roof_beam_for	704201		
roof_long_beam_A_B_left	710021			roof_long_beam_A_B_right	720021
roof_long_beam_B_C_left	710031			roof_long_beam_B_C_right	720031
shotgun_left	713201			shotgun_right	723201
side_sill_beam_A_B_left	710041			side_sill_beam_A_B_right	720041
side_sill_beam_B_C_left	710051			side_sill_beam_B_C_right	720051

Figure IV-33 Selection of b_pillar_left identification number (ID) from the table present on "Organization_Concept_Beams.xls" Excel file.

At this point the "*store.bdf*" Nastran file relative to the beam selected, is acquired in the *Matlab* Environment. The "*Coordinates_bdf.m*" function opens and reads from the script directory the Nastran file as a set of strings. According to the first word of each statement (for ex.: *GRID*, *PLOTEL*…), the function stores opportunely the data in structures recognizing the type of entry. At the end of this phase, all the entries of the input file are converted into data-structures that populates the workspace (*Figure IV-34*).

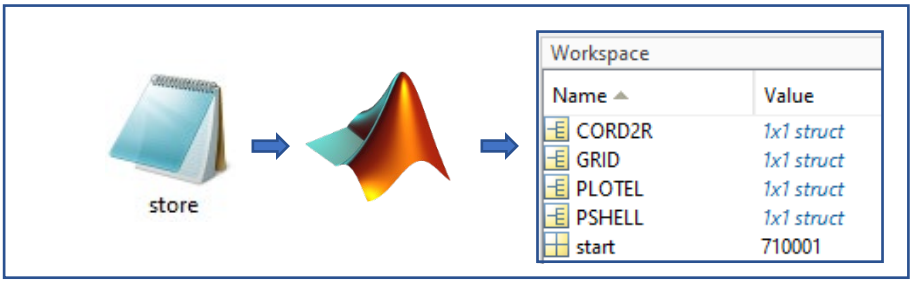


Figure IV-34 Acquisition of the "store.bdf" file into Matlab workspace

Matlab Script B – STEP 2: Acquisition of local axes and cutting planes

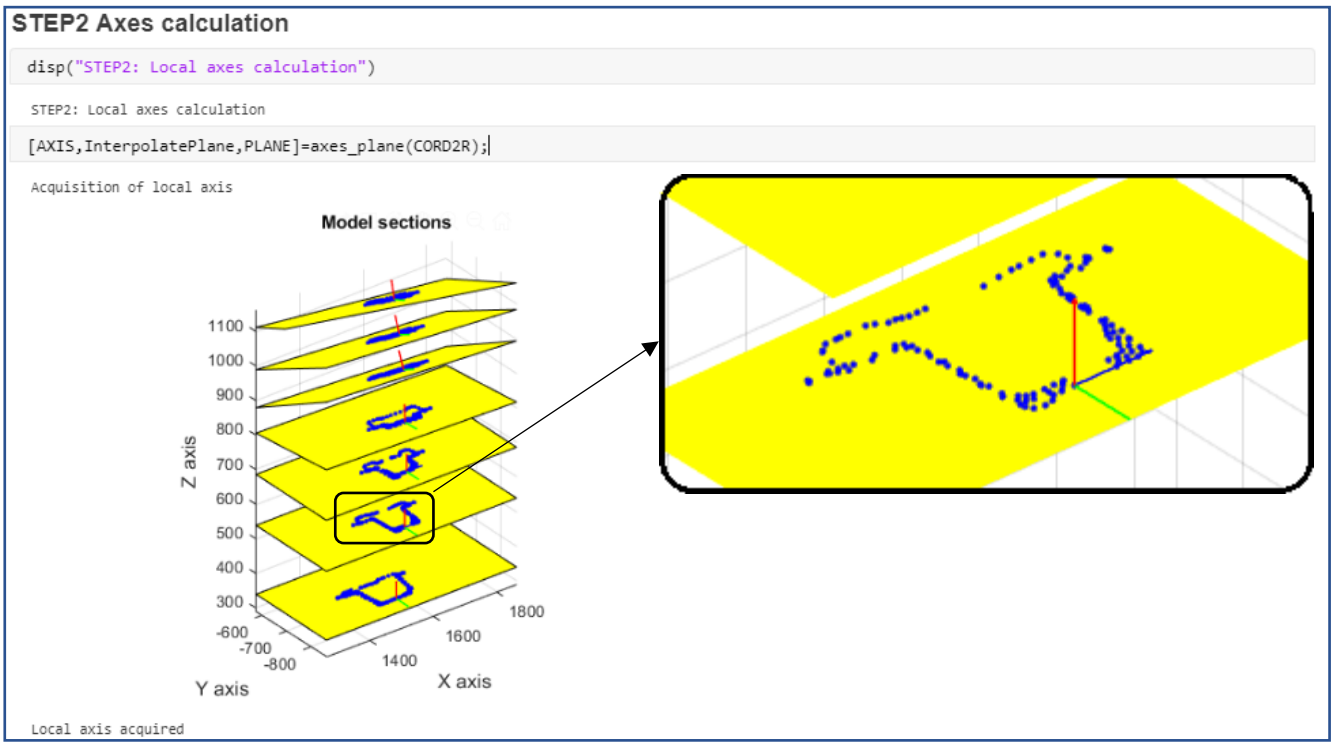


Figure IV-35 STEP 2 section execution of the Script B. This step exploits the previously descripted function "axes_plane.m".

The local reference systems are stated in the form of *CORD2R* structure. This type of structure expresses the reference frame by defining in the space the three points *A, B* and *C*, as shown previously in the *CORD2R* entry from the *Nastran Quick Reference Guide*. For a better management of the local axes and planes, the same procedure explained in *STEP 2* of the *Script A* is followed (*Figure IV-35*). The "*axes_plane.m*" function deals with the elaboration of the *CORD2R* input structure. As before, by exploiting cross product operations between vectors *OA*, *OB* and *OC,* the three local axes are found and expressed with respect to the Cartesian coordinates. Furthermore, by means of the function "*fitPlane.m*", it is possible to fit a 3-D y-z plane to a set of points for each cutting section.

Matlab Script B – STEP 3: GRID points joining and assessment of section thickness.

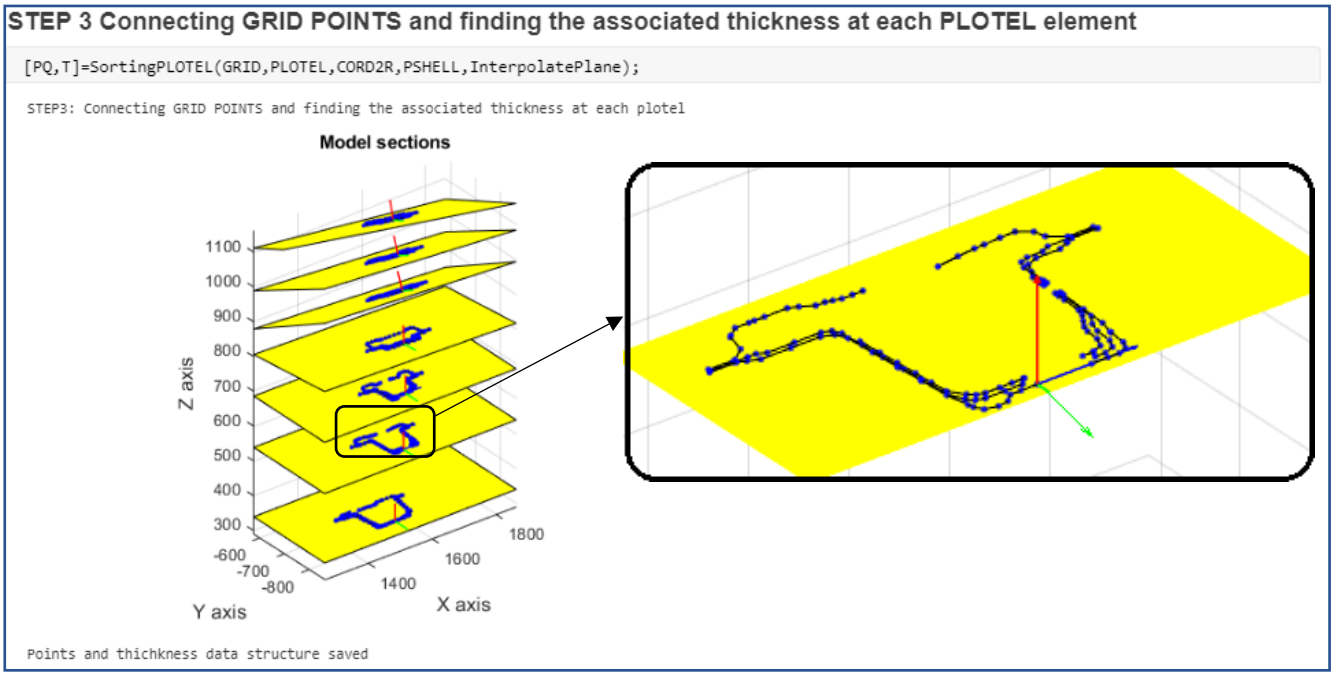


Figure IV-36 STEP 3 section execution of the Script B.

Each section is dimensionally described by segments with different widths. As a consequence, it is crucial to estimate the thickness of the cross-section for deriving the corresponding geometrical properties.

In the *STEP 3 (Figure IV-36)*, the "*SortingPLOTEL.m*" function is responsible for finding the association of the width of all the connected elements. By means of a for-loop cycle, the *GRID* points are sorted in the same manner in which they are linked in the *PLOTEL* statements previously imported. In this way, the thickness stored inside the *PSHELL* element is exported inside a *T* data structure.

At the end of this process, the new set of 3-D points *PQ* results to be directly associated with the respective *PLOTEL* thickness *T.* As a result, all the cross-sections, are completely defined.

Matlab Script B – STEP 4: Evaluation of centroid and area.

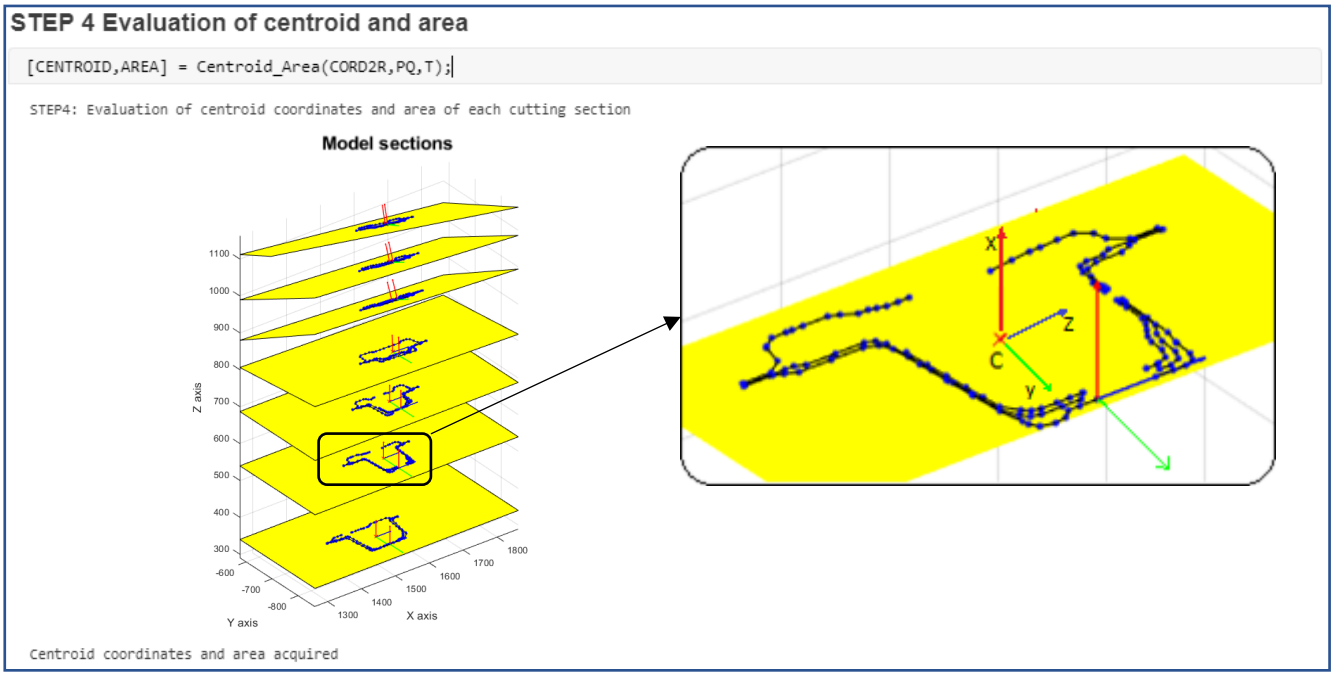


Figure IV-27 STEP 4 section execution of the Script B.

In this step, the section area and the centroid are calculated as shown in *Figure IV-37*. Each segment element is described by an area A_i of thickness T_i and a length L equal to the norm of the difference between vectors $\boldsymbol{OP_2}$ and $\boldsymbol{OP_1}$, as represented in *Figure IV-38*. The centroid coincides with the geometric center of the rectangular element.

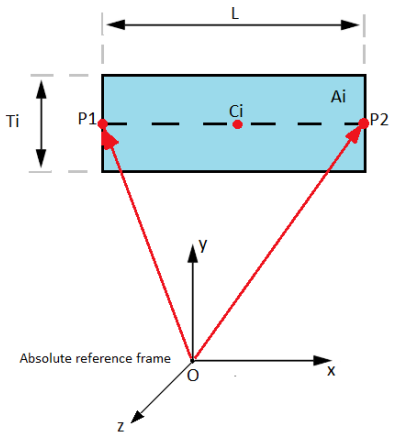


Figure IV-38 Geometric description of a PLOTEL element. P1 and P2 are two GRID points of a generic section.

The total area is given by the summation of all the areas *Ai [Equation IV-5]*:

$$A = \sum Ai = \sum Ti * L = \sum Ti * norm(P2 - P1) \qquad [IV\text{-}5]$$

By exploiting the vector properties, the centroid of each *i-th PLOTEL* element is calculated as follows in equation *[Equation IV-6]*:

$$\boldsymbol{Ci} = \frac{(\boldsymbol{OP_1}+\boldsymbol{OP_2})}{2} \qquad [IV\text{-}6]$$

The centroid coordinates components of the entire section with respect to the absolute reference frame are therefore computed *[Equations IV-7, IV-8, IV-9]*:

$$C_{ix} = \frac{\sum C_{ix}*A_i}{A} \qquad C_{iy} = \frac{\sum C_{iy}*A_i}{A} \qquad C_{iz} = \frac{\sum C_{iz}*A_i}{A} \qquad [IV\text{-}7, IV\text{-}8, IV\text{-}9]$$

The entire procedure is implemented inside the function "*Centroid_Area.m*" inside a for-loop cycle that works up on a different cross-section at each cycle. The output of this step are the *CENTROID* and *AREA* data structures.

Matlab Script B – STEP 5: Expressing points with respect to local RFs

STEP 5 Expressing points wrt cutting planes RFs

```
[P] = Coordinates_Transformation(CORD2R,PQ,AXIS,CENTROID);
```

```
STEP5: Expressing all the section points wrt centroid of each relative cutting section
Coordinates transformed.
```

Figure IV-39 STEP 5 section execution of the Script B.

Dealing with 2-D section points is essential for retrieving inertia properties of the section. For this reason, the *STEP 5* section (*Figure IV-39*) is in charge of expressing the points of each cross section with respect to the local frames located at the centroid location (assumed to be the origin). By means of the function "*Coordinates_Transformation*" at each repetition of a for-loop cycle, the sub-function "*planeposition.m*" computes the position of each point present on the cutting plane of interest. In conclusion, each point previously described by means of 3-D coordinates of the absolute reference frame (X-Y-Z) is expressed in the y-z local plane of each cutting plane and plotted (*Figure IV-40*).

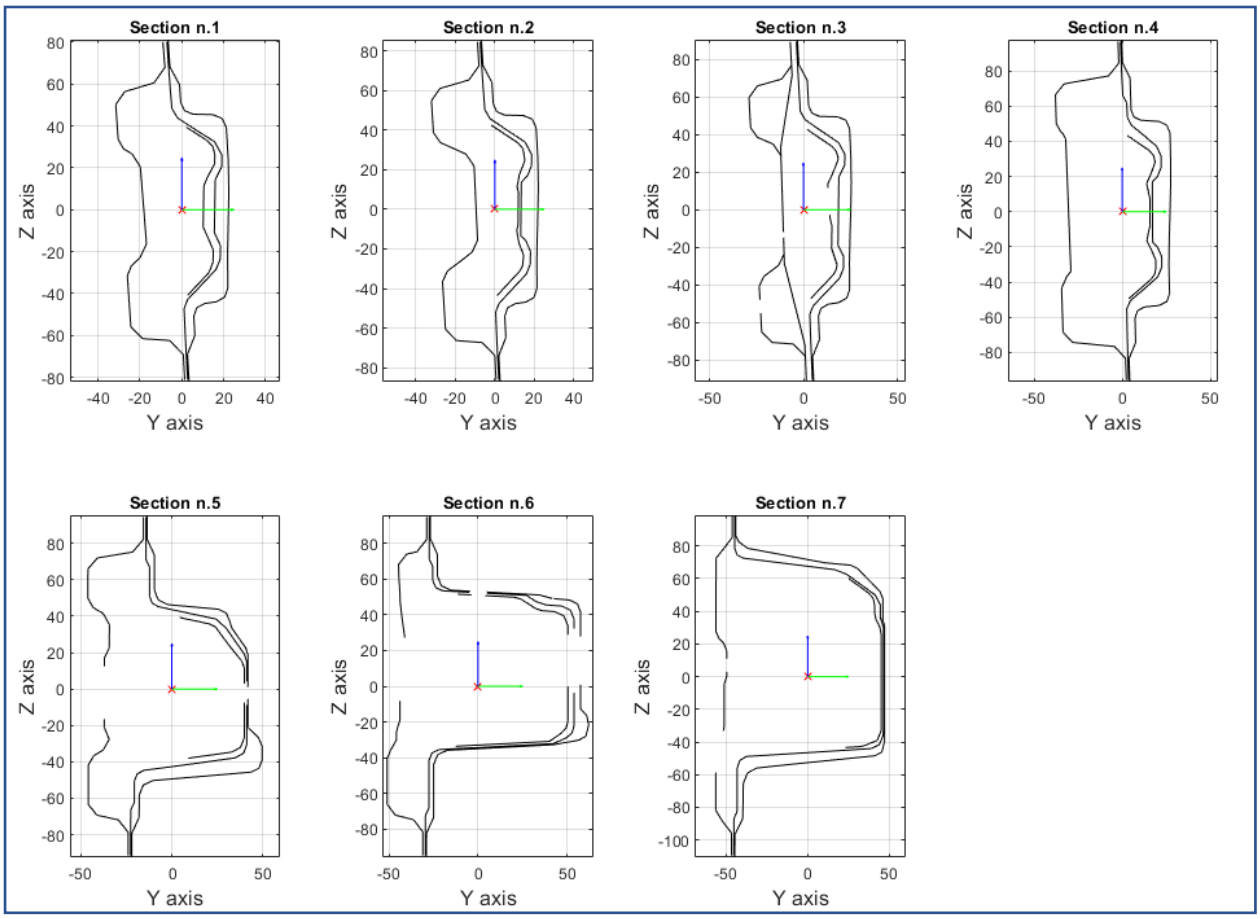


Figure IV-40 Cross sections expressed with respect the centroids of each cutting section on the y-z plane.

Matlab Script B – STEP 6: Inertia Calculation

STEP 6 Inertia Calculation

```
[I] = GeomProperties(CORD2R,P,T,CENTROID);
```

```
STEP6: Evaluation of inertia wrt local y and z axes relative at each centroid of each cutting plane
Inertia acquired
```

Figure IV-41 STEP 6 section execution of the Script B.

The mechanical behavior of a general section is well described by knowing its moment of inertia. In this part of the code (*Figure IV-41*), each cross-section is treated as a summation of its *PLOTEL* elements. Going

into more detail through the "*GeomProperties.m*" function, it is possible to evaluate inertia properties expressed according to the local reference frame axes of each cross section.

In the algorithm, each *PLOTEL* element is selected one at a time and then discretized in a sufficient number *N* of rectangles of thickness *T* and width *a*. This dimension *a* coincides with the algebraic mean of the difference between the vectors OP_1 and OP_2 on *N* elements as shown in *Figure IV-42*.

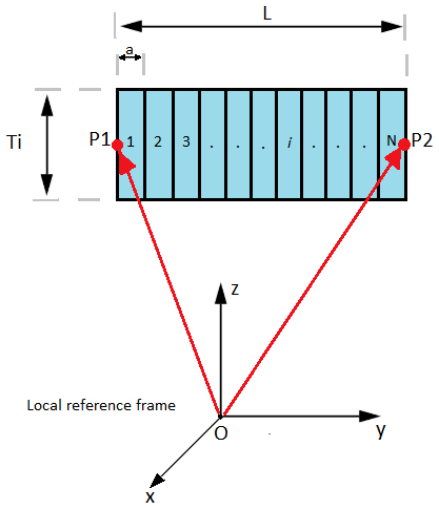


Figure IV-42 Discretization on N partitions of a generic PLOTEL element.

Then, the coordinate y_i , z_i of the centre of each *i-th* discretized element are calculated and used into the following equations (*Equations IV-10, IV-11, IV-12, IV-13*):

$$Iyy = \sum Ai * yi^2 \qquad [IV\text{-}10]$$

$$Izz = \sum Ai * zi^2 \qquad [IV\text{-}11]$$

$$Iyz = \sum Ai * yi * zi \qquad [IV\text{-}12]$$

$$J = Ixx = Iyy + Izz \qquad [IV\text{-}13]$$

The amount of resistance of a generic object to any bend-deflect or twist is known as moment of inertia. According to the local reference frames of the treated beams, the inertia Iyy and Izz should be more in order to resist the bending. At the same time, a high amount of opposition against torsion is guaranteed by high value of the polar moment of inertia J.

Matlab Script B – STEP 7: Equivalent cross-section creation

```
STEP 7 Equivalent cross-section creation
[DIM_HAT1] = DIM_SECTIONS(P,I,AREA,CORD2R);

STEP7: Creating the equivalent HAT1 sections
Equivalent HAT1 structure created
```

Figure IV-43 STEP 7 section execution of the Script B.

At this point, each cross-section is converted into an equivalent *HAT1* section (*Figure IV-43*), properly described in the *Beam Cross-Section Property* section of the *Quick Reference Guide* of *NX Nastran* (*Figure IV-44*).

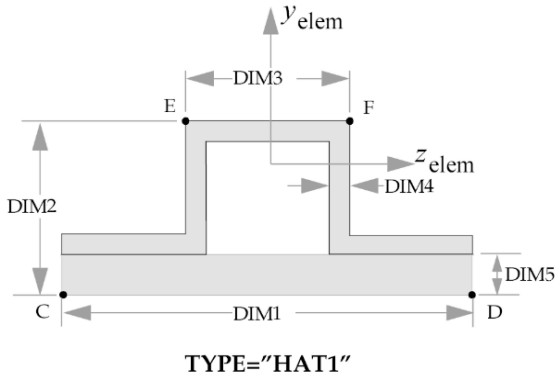


Figure IV-44 HAT1 section, from Quick Reference Guide (Siemens)

The area and inertia equations of the section are expressed with respect to the five dimensions: *DIM1*, *DIM2*, *DIM3*, *DIM4* and *DIM5*. Since the system contains five unknowns in three equations (*Iyy*, *Izz* and *Area*), some assumptions on dimensions are made. The section is re-designed as described in *Figure IV-45:*

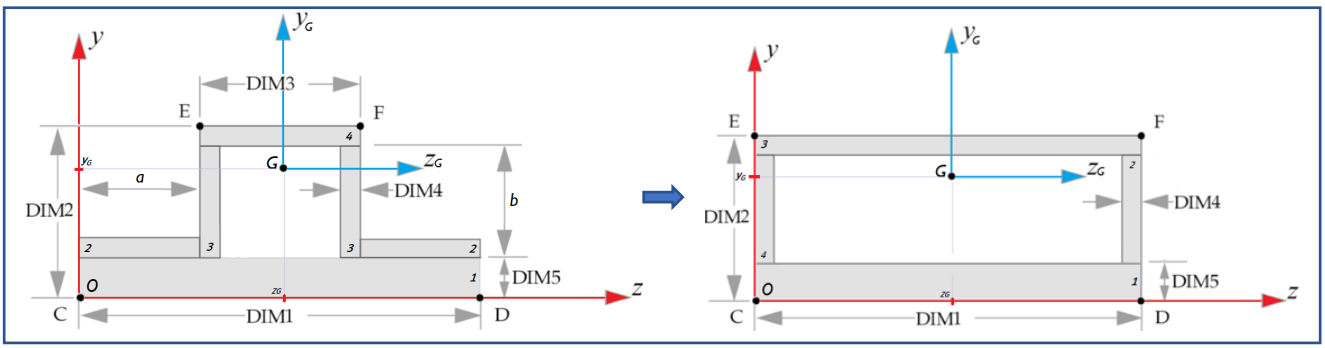


Figure IV-45 New shape of the HAT1 section under the assumption DIM3=DIM1 and DIM4=2mm.

The function "*DIM_SECTIONS.m*" is responsible for defining the properties of the optimization problem. Each dimension is treated as a parameter to be optimized by the *Matlab* non-linear least square solver called "*lsqnonlin*". In this manner, the optimization problem results defined and bounded into a set of lower and upper bounds of the design variables (*DIM1*, *DIM2* and *DIM3*). After the launch, the solver minimizes the difference between the actual data available (*Area* and *Inertia*) and the expressed equations. In conclusion, the different dimensions are found and the equivalent sections are plotted, as displayed in *Figure IV-46.*

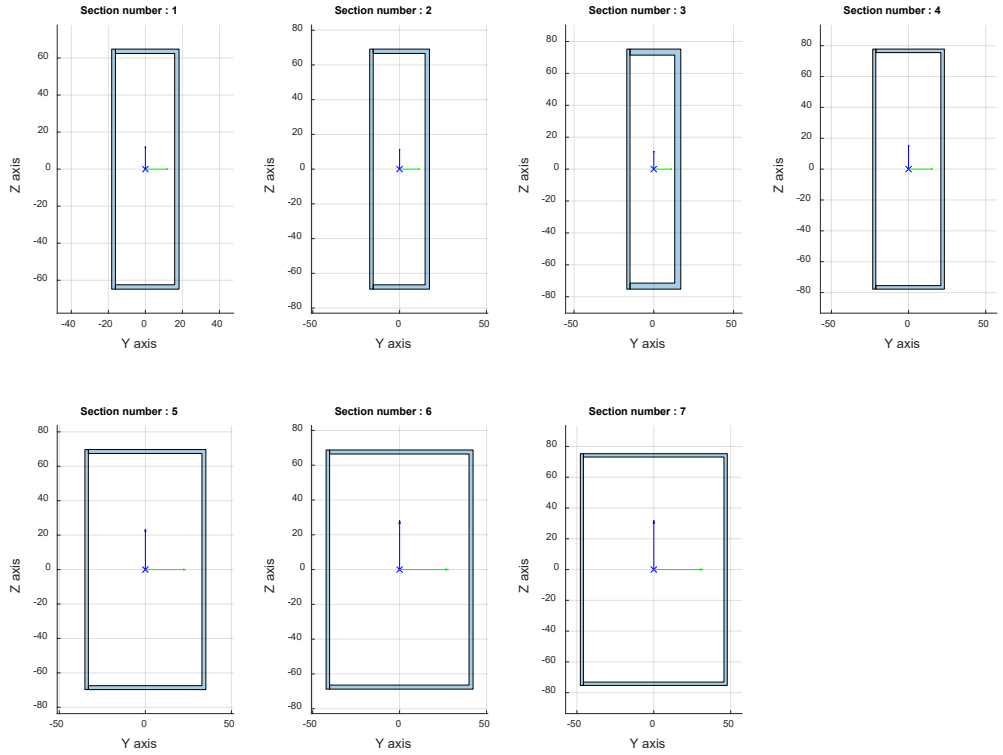


Figure IV-46 Matlab plot of equivalent HAT1 sections.

Matlab Script B – STEP 8: Shear-center calculation

STEP 8 Shear center calculation

```
[SCENTER,DELTA] = ShearCenter_(DIM_HAT1,CENTROID,PLANE,CORD2R);

STEP8: Calculating the shear center position wrt the absolute reference frame and the relative
displacement DELTA wrt the centroid
Shear center position and Delta acquired
```

Figure IV-47 STEP 8 section execution of the Script B.

According to the *PBEAM* description of a generic beam, the local reference system *y-z* is centred in the shear-center of each cutting section (as done in STEP 8, *Figure IV-47*). The centroid is located at a certain distance and interconnects the neutral axis between successive cross-sections, *Figure IV-48*.

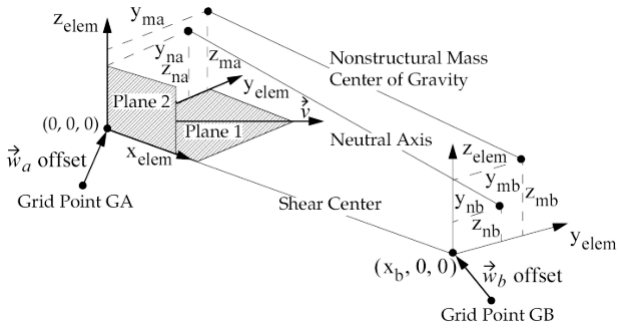


Figure IV-48 PBEAM Element Coordinate System from Quick Reference Guide (Siemens).

For symmetric sections, the shear-center coincides with the centroid. The *HAT1* section results do not respect symmetry in z- direction. For this reason, a further study on shear-center properties is performed.

The position of the shear-center (*Figure IV-49*) is calculated in such a way that an applied shear force can act on the cross-section without producing any twist in the section. The outputs of "*ShearCenter.m*" function are the coordinates of the shear-center with respect to the absolute reference frame and the relative displacement named as "*DELTA*" of each centroid.

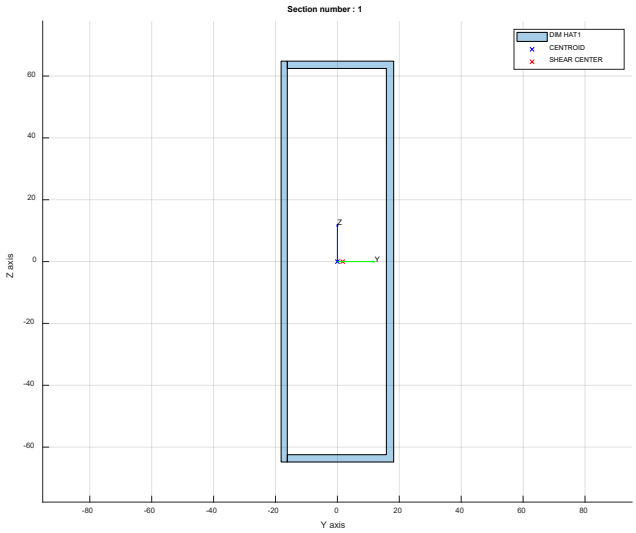


Figure IV-49 HAT 1 equivalent section with shear-center point.

Matlab Script B – STEP 9: Storing results

```
STEP 9 Storing results
[OK]=StoringResultsPBEAM_hat1_adapted(I,CENTROID,AREA,AXIS,DIM_HAT1,SCENTER,DELTA,start);

STEP9: Starting storing results
Results stored in b_pillar_left_PBEAM.bdf
[OK]=StoringResultsPBEAML_hat1_adapted(I,CENTROID,AREA,AXIS,DIM_HAT1,SCENTER,DELTA,start);

STEP9: Starting storing results
Results stored in b_pillar_left_PBEAML.bdf
```

Figure IV-50 STEP 9 section execution of the Script B.

In this last section (*Figure IV-50*), all the beams are defined by *GRID* points interconnected by *CBEAM* elements. Each *CBEAM* entry defines a beam element connection by means of the grid point identification numbers of connection points.

The *GRID* points are located at the shear-centers. In particular the *PBEAM* entry characterizes the properties of a beam element (*CBEAM* entry) in terms of area, moment of inertia and torsional stiffness of each beam cross section. The *PBEAML* entry defines the properties of a beam element by cross-sectional dimensions. The cross-section shape *HAT1* is selected and all the cross-sections dimensions are inserted.

Three concept beam examples are shown (*Figure IV-51, IV-52, IV-53*) in which it is possible to appreciate the quality of the approximation.

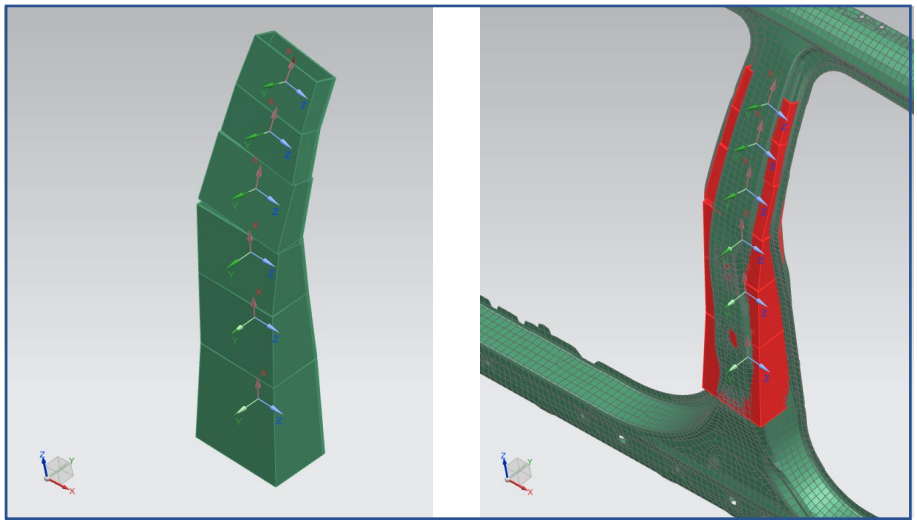


Figure IV-51 Left b-pillar concept beam

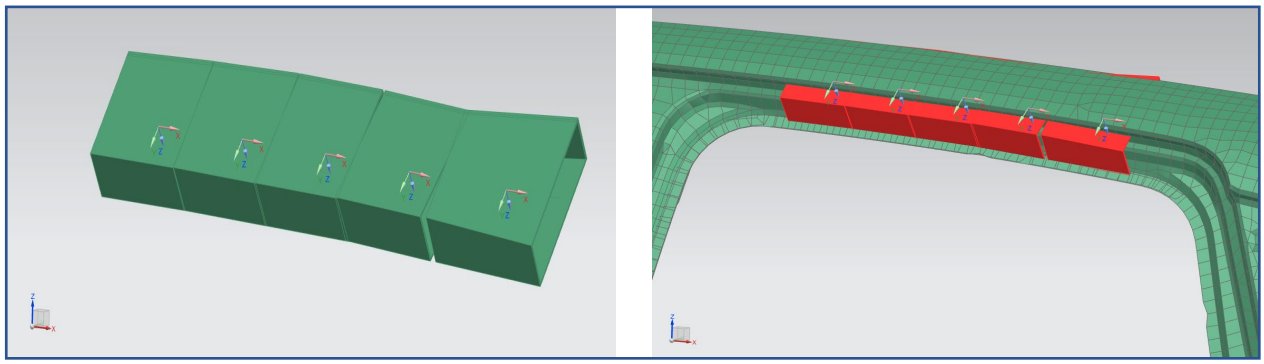


Figure IV-52 Roof long beam B-C (left side)

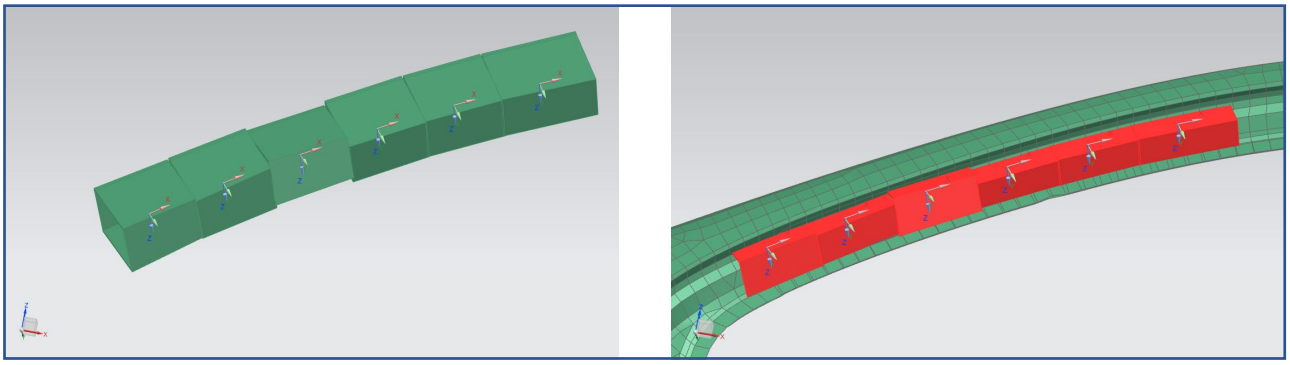


Figure IV-33 A pillar (left side)

V. Mapping procedure

Introduction

The wireframe represents a geometric simplified representation of a generic car vehicle structure. Each segment represents a beam while each node represents either a joint or an interface between beams. The associativity between beams/joints and parametric wireframe is realized by means of using "selection recipes" in Simcenter 3-D. This is basically a user-defined naming convention given to each node of the CAD wireframe. When changing height, length, breadth or some characteristic angle of the wireframe, the Matlab Mapping script projects the original beams and joints of the "baseline" library onto the modified wireframe and will create a new library, which will contain the new beams connected through the joints.

The Matlab script uses as inputs the body-part library compressed into a ".mat" file and the ".bdf" wireframe (Figure V-1) previously exported from Simcenter 3-D and creating an equivalent car body model library. In this chapter, the main steps for the creation of the script are treated, along with all the technical details.

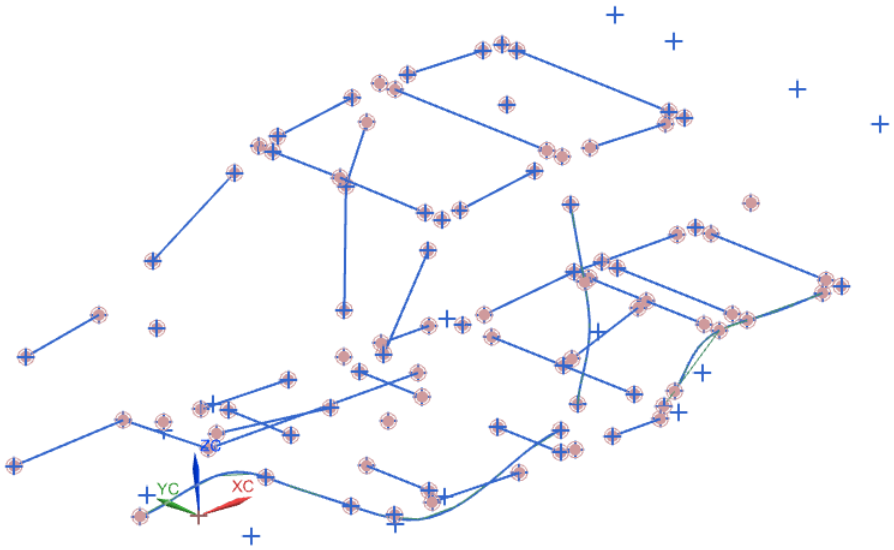


Figure V-1 Wireframe of a generic BIW in Simcenter 3-D

Parametric body CAD

In the early (concept) design phase, it is fundamental to have a generic idea of the main proportions of the vehicles in terms of beams dimensions and respective angular positions.

The wireframe can be helpful for this purpose because it represents a simplified and parametric model of a *BIW*. Some of the main advantages and considerations of the parametric body CAD creation and development are:

- The concept CAE model can be created without any need for the detailed CAD data. It can be either an output of the **styling and packaging process** or it can be a generated **from scratch**;
- As shown in *Figure V-2*, it is possible to set in the *NX Nastran Environment* all the vehicle global dimensions (angles, vehicle height, etc.) by means of an interface composed by several sliders. According to the needs, different kinds of configurations can be exported.

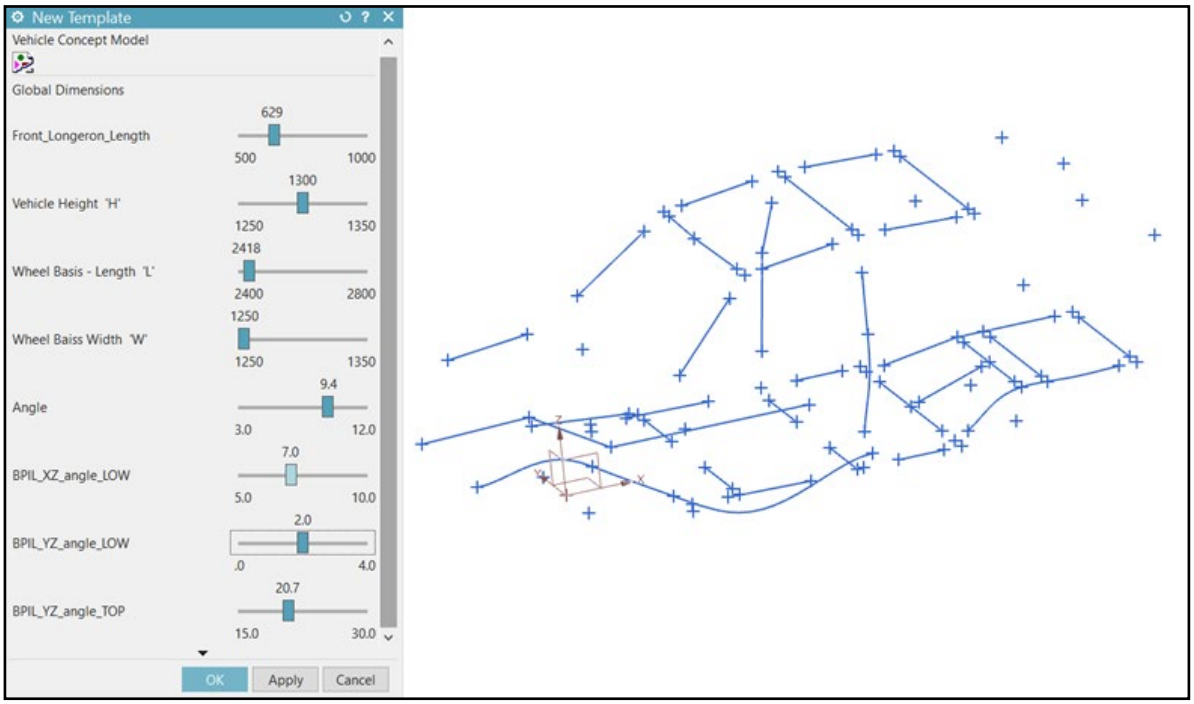


Figure V-2 Setting of different wireframe parameters.U

- In order to prevent possible damage of the vehicle due to torsional and inertial effects, it is possible to perform 1-D torsional analysis by using the *FEA Nastran solver*.

- The only requirement of the parametric model is the use of a naming convention for the association of the corresponding beams and joints for existing libraries. This is done by defining the "**selection recipes**" on *Simcenter 3-D,* as shown in *Figure V-3*. This information is contained in the wireframe export file and reference library *Matlab* file. It is managed from the mapping tool for the generation of the equivalent BIW directory.

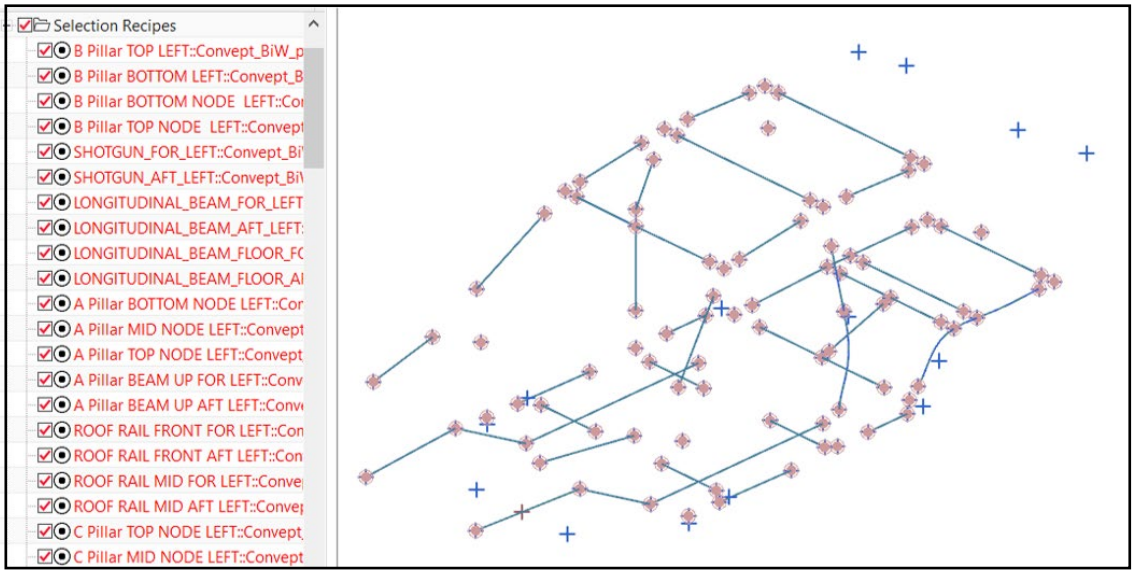


Figure V-3 Insertion of selection recipes at each node of the BIW wireframe.

- It is possible to extend the parametric model with new beams and to generate a more sophisticated 1-D model. Each segment can be manually inserted or automatically generated by using the spline command in NX Nastran.

- A next step in the development of this method would consist in the automatic wireframe model creation from a template Excel file where the user selects the topology (sedan, hatchback, SUV,..) and enters the main geometric dimensions (e.g. wheel base distance, vehicle height,..)

Workflow for mapping the beams onto the parametric CAD wireframe

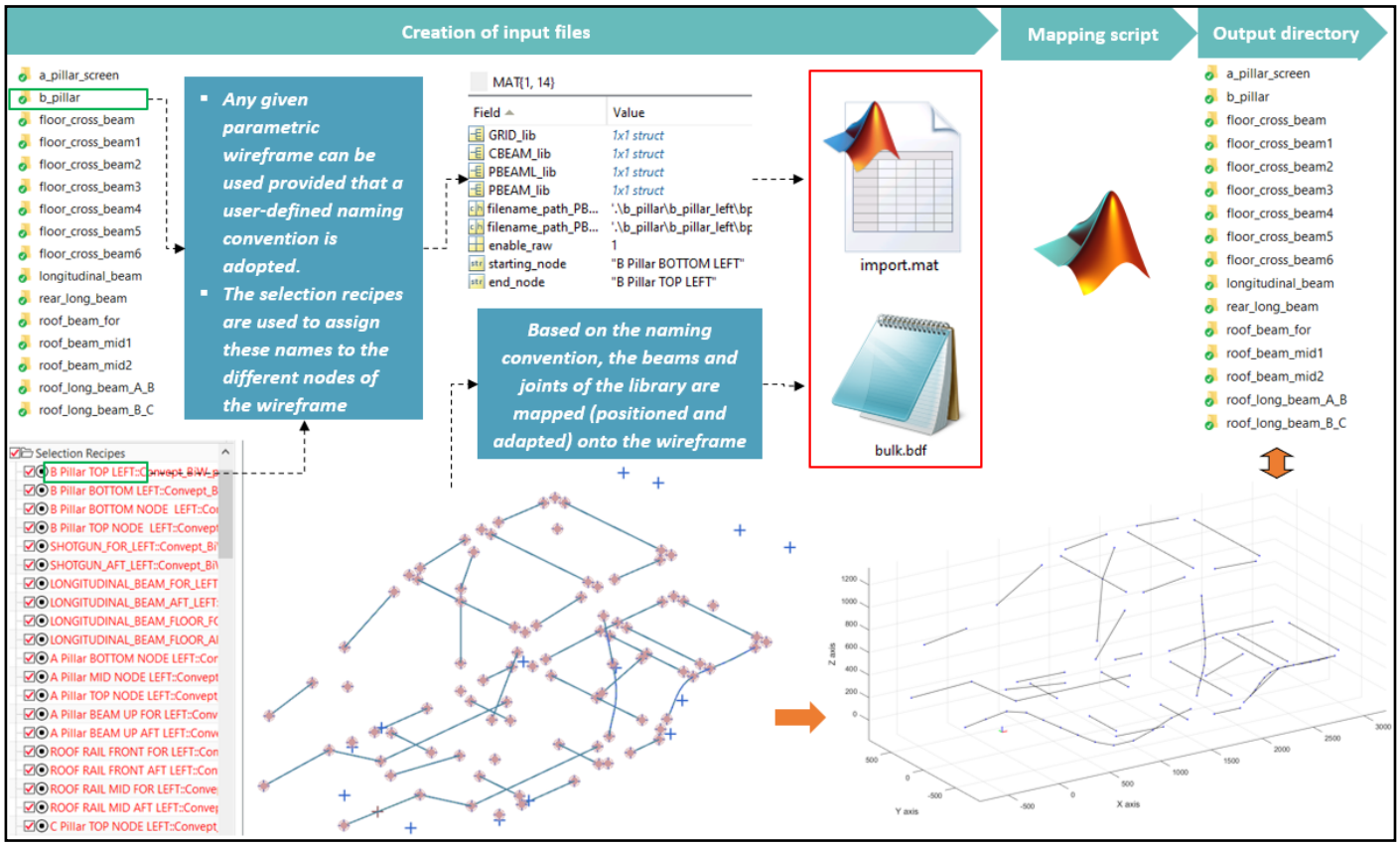


Figure V-4 Summary scheme on mapping procedure of existent beams onto BIW wireframe configuration.

The illustrated workflow (*Figure V-4*) shows the main steps followed during the mapping operation. The reference library, constituted from concept elements of a generic BIW, is compressed inside a *Matlab* data structure and then exported inside a file named "*import.mat*".

At the same time, the wireframe nodes are assigned by means of "*selection recipes*". This is a user-based name convention that allows all the wireframe nodes to be directly associated with the concept beams used as reference. At the end of this operation, the wireframe configuration is exported from NX Nastran in the form of a *".bdf"* file named as "*bulk.bdf*".

Both files "*import.mat*" and "*bulk.bdf*" are the inputs files of the ***Matlab Mapping Script.*** This tool is in charge of re-scaling all the new concept beams while maintaining unchanged the mechanical behaviours

of the generated BIW. It inputs the reference library *Matlab* file and the wireframe configuration for generating the output directory.

Creation of the "import.mat" file

By means of an ad-hoc *Matlab* script, the reference library is compressed inside a data structure called "*MAT*", opportunely described in *Figure* V-5. This operation makes the next mapping procedure more manageable and it is done for the future inclusion of the reference library inside the optimization process onto *Heeds* software (*Chapter 6*).

In the first operation, the library is acquired in the *Matlab* workspace. All the concept beams included in the wireframe are converted in cells of the *MAT* structure. Each cell contains:

- "***GRID_lib***": *GRID* points of each library concept beam (shear-centers);
- "***CBEAM_lib***": *CBEAM* entry of each library concept beam.
- "***PBEAML_lib***" / "***PBEAM_lib***": *PBEAML* and *PBEAM entries.*
- "***Filename_path_PBEAM***" / "***Filename_path_PBEAML***": destination output path;
- "***Starting_node***" / "***End_node***": node names. These tags coincide with the selection recipes of the wireframe;
- "***Enable_row***" flag. This value assumes two logical value. "*1*" if the wireframe beam is present inside the reference library, "*0*" in the opposite case.

Field	Value
GRID_lib	*1x1 struct*
CBEAM_lib	*1x1 struct*
PBEAML_lib	*1x1 struct*
PBEAM_lib	*1x1 struct*
filename_path_PB...	'.\roof_beam_mid1\cbeams_hat1_PBEAML.bdf'
filename_path_PB...	'.\roof_beam_mid1\cbeams_hat1_PBEAM.bdf'
enable_raw	1
starting_node	"ROOF BEAM MID1 LEFT"
end_node	"ROOF BEAM MID1 RIGHT"

Figure V-5 Cell of the MAT structure, relative to the roof beam mid1.

Mapping script

Data acquisition wire-frame and reference library

```
filename_wfm='bulk.dat';   % Wireframe export file
[GRID_wfm,CBEAM_wfm,RECIPE_wfm]=Data_acquisition_wfm_large(filename_wfm);
load('import.mat');  % load the reference beam library saved inside "import.mat"
```

Wire-frame beam selection: each beam present in the wireframe is associated with the beam of the library

```
for i=1:length(MAT)
    if MAT{i}.enable_raw==1   %selects only the lib. beams associated with the wireframe
        [GRID_BEAM_wfm]=Beam_selection_wfm(CBEAM_wfm,GRID_wfm,RECIPE_wfm,MAT{i}.starting_node,MAT{i}.end_node,MAT{i}.GRID_lib,MAT{i}.CBEAM_lib);
```

Option selection for the considered beam: basing on the number of intermediate points in the wireframe

```
        n=length(GRID_BEAM_wfm.ID)-2;  %number of intermediate points
        if n>0 & n<3
            option='option2';
        end
        if n==0
            option='option3';
        end
        if n>3
            option='option1';
        end
```

Algorithm

STEP1: Interpolate beam in library with a spline called spline_lib

```
        [spline_lib,t2,L2,d,dd]=Step1(MAT{i}.GRID_lib);
```

STEP2: Calculate x_L2=t2(i)/L1 where t is a parameter 0<=t2<=L2 + tangent in spline_lib

```
        [x_L2,tangent_lib]=Step2(spline_lib,t2,L2,d,dd);
```

STEP3: Create a spline (spline_wfm) passing through all wireframe points

```
        if option=='option1'
            [points_section,L_fin] = Step3_opt1(tangent_lib,GRID_BEAM_wfm,x_L2,dd);
        end
        if option=='option2'
            [points_section,L_fin] = Step3_opt2(spline_lib,tangent_lib,L2,GRID_BEAM_wfm,x_L2,dd,d,n);
        end
        if option=='option3'
            [points_section,L_fin] = Step3_opt3(GRID_BEAM_wfm,x_L2);
        end
```

STEP 4 Retrieve cross sections properties

```
        [GRID,CBEAM,PBEAML,PBEAM]=Step4(points_section,L_fin,L2,MAT{i}.GRID_lib,MAT{i}.PBEAML_lib,MAT{i}.CBEAM_lib,MAT{i}.PBEAM_lib);
```

Storing

```
        [path_PBEAM,path_PBEAML]=Converting_paths(MAT)
        [OK]=Storing_PBEAML(GRID,CBEAM,PBEAML,path_PBEAML,directory);  %% storing pbeaml files inside N directory
        [OK]=Storing_PBEAM(GRID,CBEAM,PBEAM,path_PBEAM);    %% storing pbeam file inside N directory
    end
end
disp("Results stored")
```

Figure V-6 Matlab mapping script section subdivision.

In *Figure V-6,* the principal steps of the *Matlab Mapping Script* are described. After the acquisition of the body parts library and the wireframe configuration, at each iteration a different wireframe beam is selected and processed. At each cycle, the algorithm evaluates the most suitable solution by selecting the more opportune type of option, based on the number of intermediate nodes of the selected wireframe beam. In order to guarantee a reliable procedure, the number of intermediate points is increased and less geometric constraints are required in terms of tangent conditions.

The algorithm is articulated in four steps. Then, the beam is saved in the respective output directory and the loop proceeds until the end. The command windows will show the conclusion of the loop and will inform the user of the finish of the procedure.

Data acquisition (wireframe configuration and reference body-parts library)

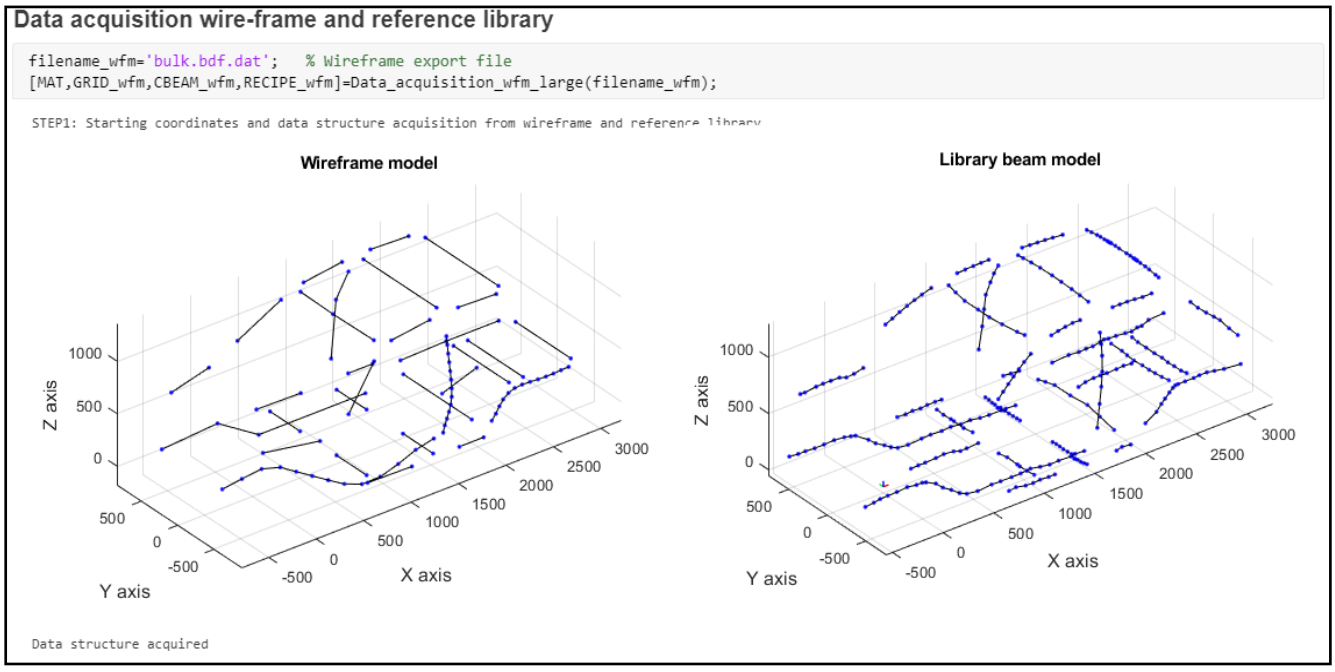


Figure V-7 Input data acquisition, Mapping script

The data acquisition (*Figure V-7*) is managed by means of a proper data-acquisition function. In this code section, the wireframe "*.bdf*" file and the reference library (*import.mat* file) contained in the script

directory are imported in the *Matlab* Workspace, as shown in *Figure V-8.* All the *GRID*, *CBEAM* entries of the wireframe are acquired, as well as the recipe node names.

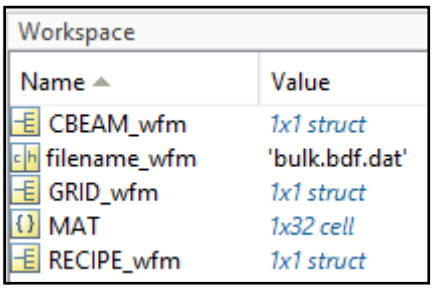


Figure V-8 Matlab workspace after the data acquisition of wireframe and reference library

Wireframe beam selection and matching

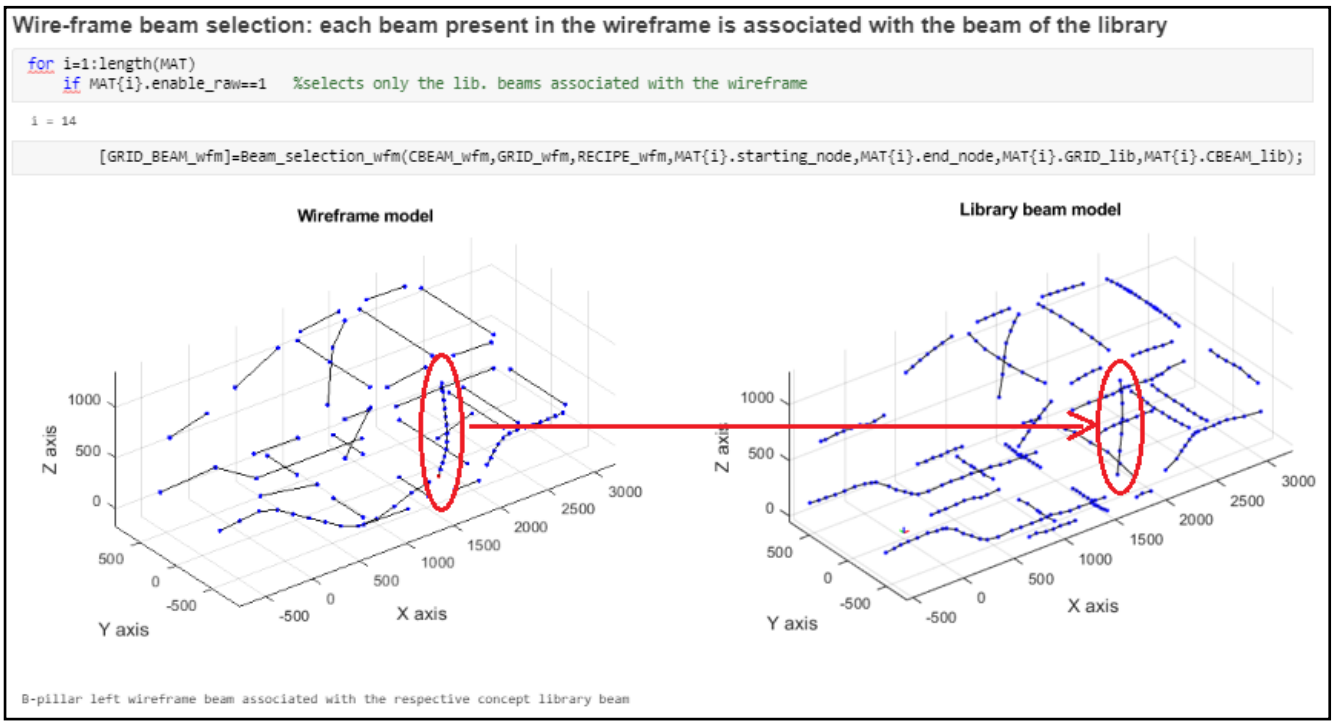


Figure V-9 Code section responsible for matching the wireframe beam with the concept beam of the reference library.

At this stage, the for-loop cycle evolves for each concept beam of the reference library (*Figure V-9*). At each cycle, the matching process starts unless the "*enabled_row*" variable is equal to one. This means that

the corresponding wireframe beam exists inside the reference beam collection. In fact, the wireframe can be described with fewer beams than the ones available in the reference directory. In that case, the "*if*" statement is false ("*enabled_row"* equal to zero), the for-loop cycle ends the iteration and the restarts with a new one.

The "*beam_selection_wfm.m*" is accountable for picking out the single wireframe beam. This is performed by looking at the grid identification number of the starting and ending nodes (selection recipes names). Then the *CBEAM* gives the succession of grid interconnections. The output represents a subset of the original *GRID* data structure, called *GRID_BEAM_wfm*. It contains all the wireframe points of the selected wireframe beam.

Option selection

Option selection for the considered beam: basing on the number of intermediate points in the wireframe

```
n=length(GRID_BEAM_wfm.ID)-2;  %number of intermediate points
if n>0 & n<3
    option='option2';
end
if n==0
    option='option3';
end
if n>3
    option='option1';
end
```

Figure V-10 Code section responsible for choosing the type of option.

This fragment of code (*Figure V-10*) corresponds to a preliminary phase for *STEP 3* of the algorithm. According to the number of intermediate points of the wireframe beam, a different option is selected:

- **Option 1:** the wireframe beam is described with more than three intermediate points;
- **Option 2**: the wireframe beam presents one or two intermediate points;
- **Option 3**: the wireframe beam corresponds to a segment (no middle points).

During the wireframe design phase, it is suggested to use as many intermediate points as possible in order to have a better 3-D representation. In *STEP 3,* more details on the types of options are available.

Step 1: Library beam interpolation

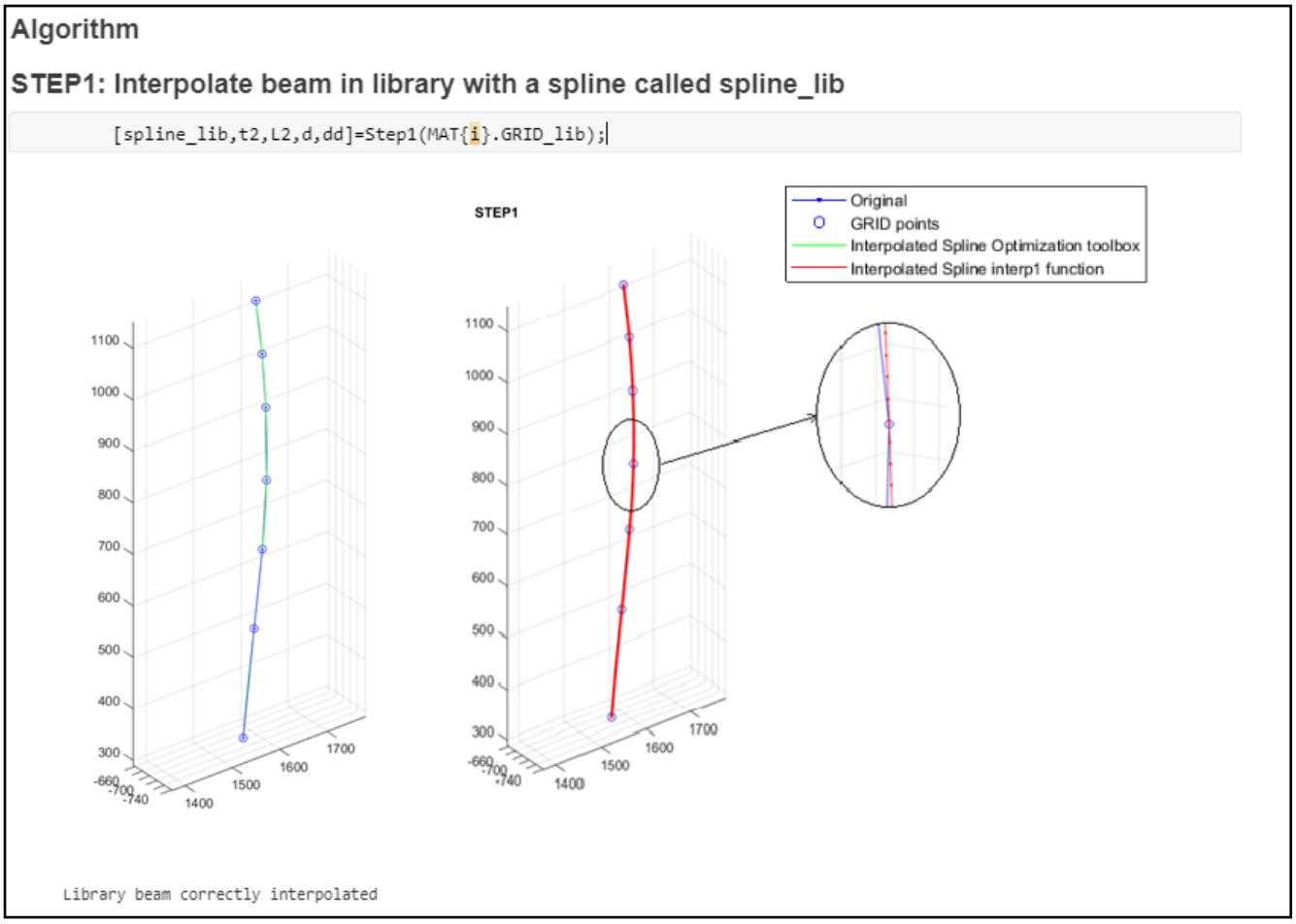


Figure V-11 STEP 1 of the mapping algorithm.

The first objective is to determine for each library beam an interpolated function which passes through each consecutive *GRID* point.

At this point, the process of each concept beam starts. As a result, two different geometric descriptions of the selected interpolated curve (*Figure V-11*) are provided. The interpolation method chosen is the "*cubic spline*".

The green spline represents a parametric description of the curve described by a *univariate piecewise polynomial* (command "*cscvn*" from *Optimization Matlab toolbox*). This function returns a parametric variational, or *natural,* cubic spline curve passing through the given sequence of points. The spline is

described by a continuous curve ("*spline_lib*") in which t_2 is the parameter value that describes the length of each curved part and L_2 is the total length of the library spline.

The second type of spline (the red curve) is constructed by using the native *Matlab* function called "*interp1*". It generates a piecewise cubic spline interpolation described by 3-D points (*3 x n* matrix called "*dd*", where n is the precision of the curve). By tuning the accuracy, it is possible to change the number of segments which describes the curve. This second type of spline representation helps during the operation of adding constraints to the final spline by finding tangent relations. The tangent points are managed by adopting the definition of incremental ratio of a 3-D curve. In particular, the algorithm is based on fitting the same neighborhood behavior of the spline of library onto the final interpolated curve.

Step 2: Evaluation of proportionality and tangent behaviour of the library spline

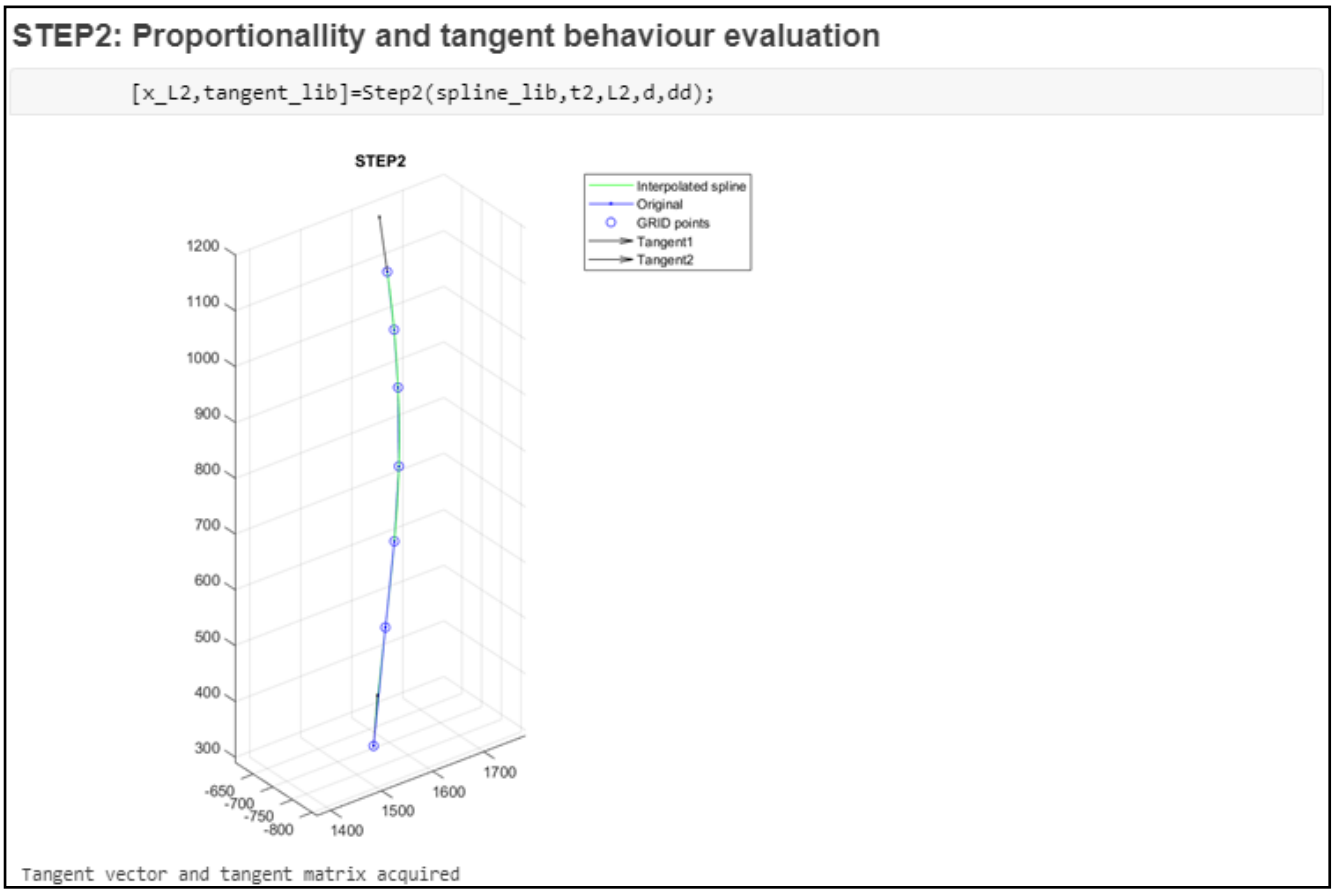


Figure V-12 STEP 2 of the mapping algorithm.

In order to have a reliable reproduction of the curvature of each referenced concept beam, it is appropriate to find the tangent behaviour of the library spline. In this step (*Figure V-12*) $x_{L2} = \frac{t_2(i)}{L_1}$ is calculated where t_2 is a parameter ($0 <= t_2 <= L_2$) of the green spline. The vector x_{L2} represents the vector of element of proportionality of each *GRID* point of the spline library with respect the total length L_2 of the curve.

As a further result, the tangent vectors at the start and at the end nodes are calculated and displayed.

Step 3: Creation of the final spline

STEP3: Creation of the final spline

```
if option=='option1'
    [points_section] = Step3_opt1(tangent_lib,GRID_BEAM_wfm,x_L2,dd);
end
if option=='option2'
    [points_section] = Step3_opt2(spline_lib,tangent_lib,L2,GRID_BEAM_wfm,x_L2,dd,d,n);
end
if option=='option3'
    [points_section] = Step3_opt3(GRID_BEAM_wfm,x_L2);
end
```

Figure V-13 STEP 3 of the mapping algorithm.

As discussed previously, three different options are available. In STEP 3 (*Figure V-13*), each option is automatically selected according to the geometric configuration of the wireframe beam associated.

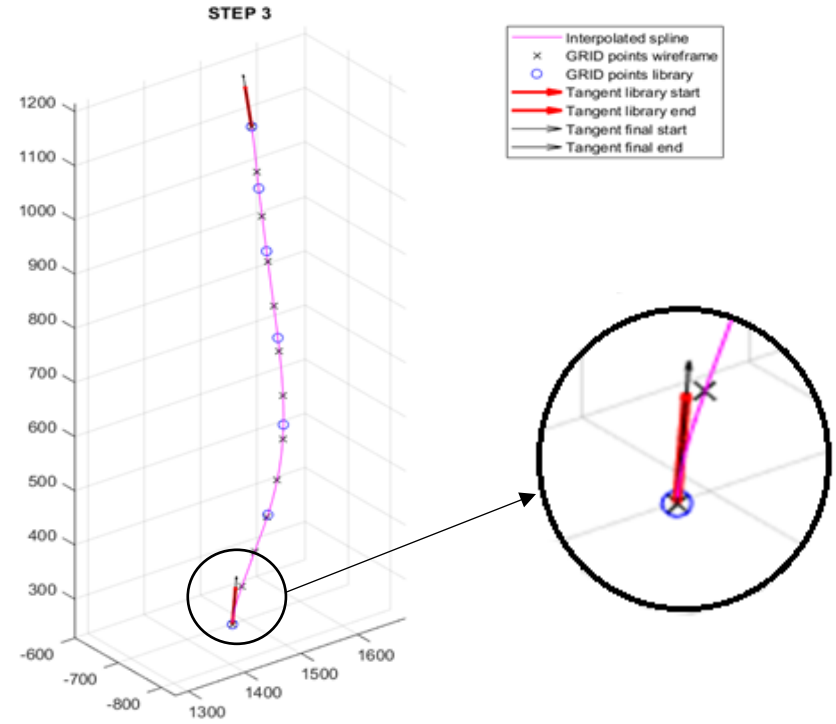
Therefore, the final spline is built by respecting the boundary conditions. In particular, the spline will pass through each wireframe point. According to the option selected, the spline satisfies fewer or more constraints (in terms of tangent conditions). By adopting a "*case by case*" approach, the interpolation problem results differently constrained.

The output of this step is the "*points_section*" matrix. It is a *3 x n* matrix, where *n* is the number of cutting sections of the selected library beam. Each column contains the spatial coordinates of the projected library *GRID* point in the space. Each of these points are found by means of the *Matlab* function "*fnval*". Given the final spline and the entire break sequence ($t_{eq} = x_{L2} * L_{fin}$), it provides the values at each breakpoint.

Step 3 – Option 1: the wireframe beam is described with more than three intermediate points

STEP3: Creation of the final spline

```
if option=='option1'
    [points_section] = Step3_opt1(tangent_lib,GRID_BEAM_wfm,x_L2,dd);
end
```



Tangent condition applied at start and end node. Final spline created and corresponding section points computed.

Figure V-14 STEP 3 - Option 1

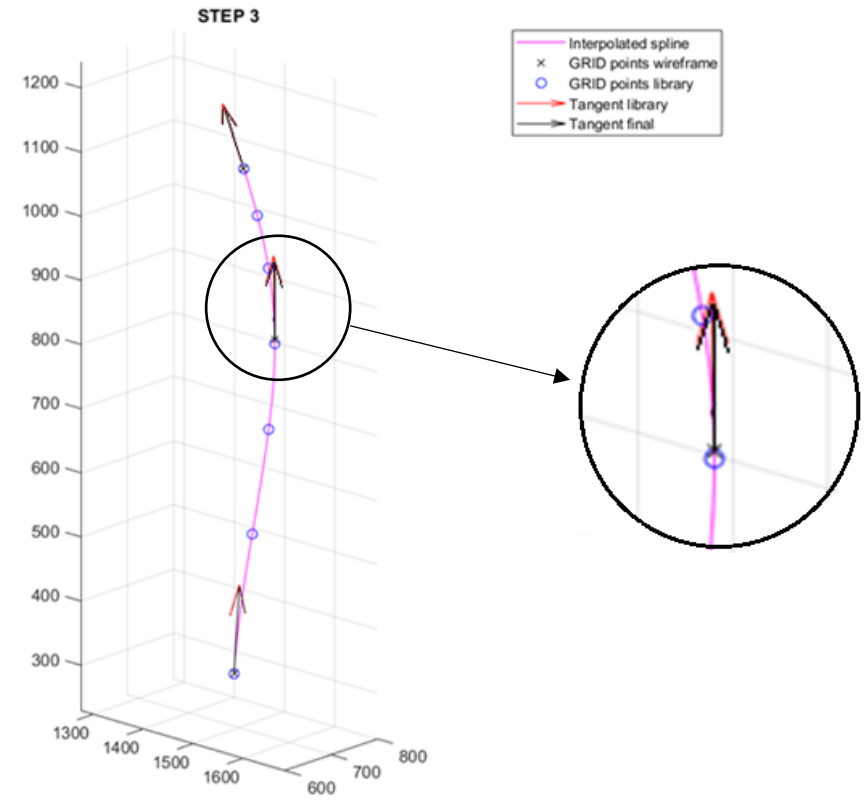
In *option 1* (*Figure V-14*) the beam of the wireframe is described by many intermediate points ($n > 3$). In this case, the wireframe beam has a fine interpolation. Therefore, the final spline goes through the wireframe points and it has the same tangent at the end and start nodes of the library spline, previously exported in *STEP 2*.

The example reported is the left *B-pilar beam*. As a test of the respected conditions, the tangent arrows at the start and end nodes of the final spline are the same as the corresponding beam of the library.

Step 3 – Option 2: the wireframe beam presents one or two intermediate points

STEP3: Creation of the final spline

```
if option=='option2'
    [points_section] = Step3_opt2(spline_lib,tangent_lib,L2,GRID_BEAM_wfm,x_L2,dd,d,n);
end
```




```
Tangent condition applied at start, middle and end nodes. Final spline created and corresponding section points computed.
```

Figure V-15 STEP 3 - Option 2

In *option 2 (Figure V-15)* the beam of the wireframe is described by few intermediate points ($0 < n < 3$). In this situation, the wireframe beam has a course interpolation. Thus, the final spline goes through the wireframe points and it has the same tangent at the end and start nodes of the library spline, previously exported in *STEP 2*. As a further condition, the same tangent in the equivalent point of the wireframe reported on the spline of the library (intermediate points) is imposed.

Step 3 – Option 3: the wireframe beam presents no intermediate points

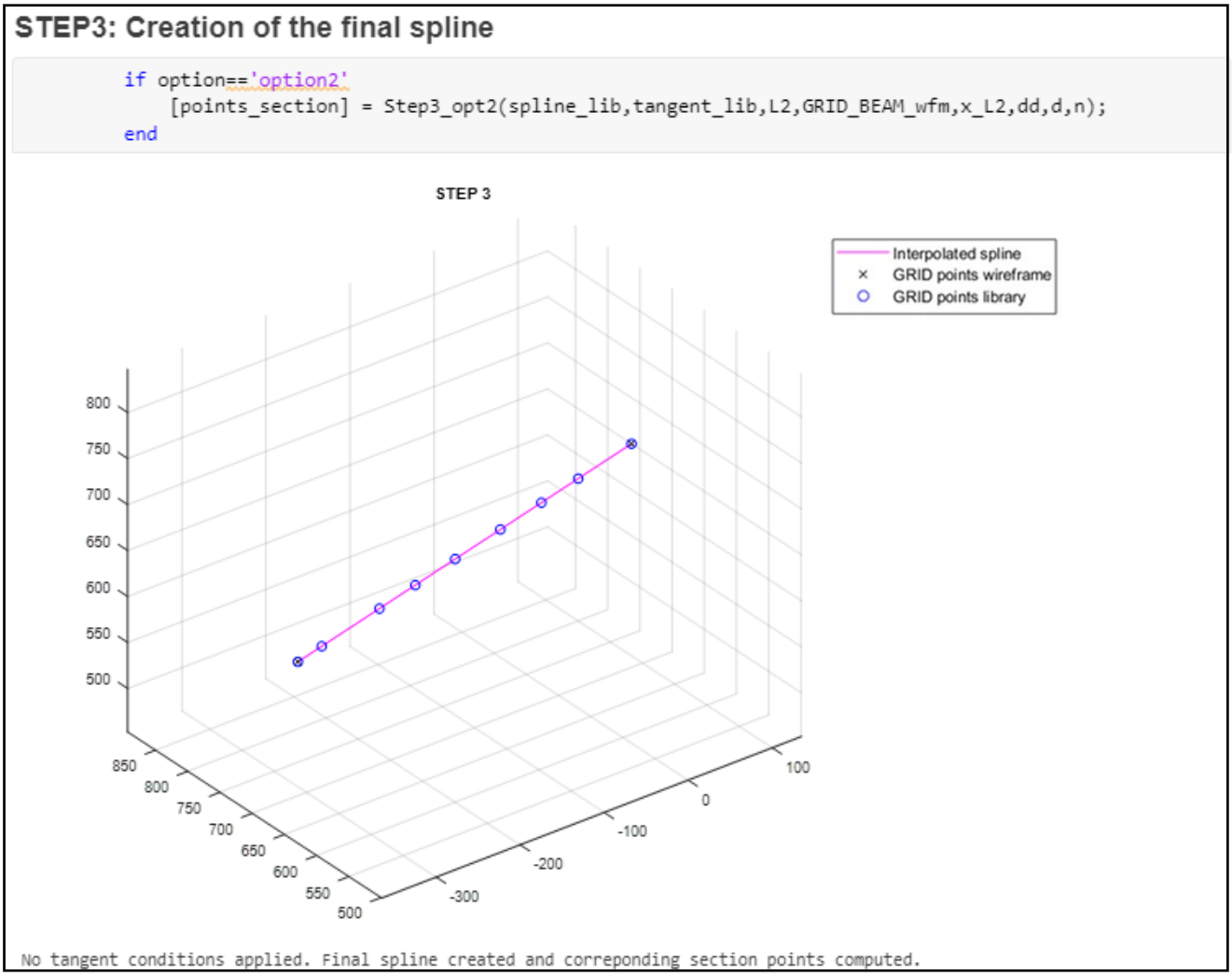


Figure V-16 STEP 3 - Option 3

In *option 3 (Figure V-16)* the beam of the wireframe is described by no intermediate points ($n = 0$). This means that the wireframe beam has a course interpolation.

For this type of case, the final spline goes only through the two extreme wireframe points. No further constraints are imposed on tangent points.

Creation of the output data structures

Recovery of the output beam properties

```
[GRID,CBEAM,PBEAML,PBEAM]=Step4(points_section,MAT);
```

```
GRID, CBEAM, PBEAML, PBEAM structure created
```

Figure V-17 Recovery of the output beam properties (mapping script).

At this step (*Figure V-17*), the previously found 3-D points are saved into a *GRID* data structure. At the same time, all the information stored inside the *MAT* data structure are translated into *CBEAM*, *PBEAM* and *PBEAML*. In this way, the output beam preserves the same mechanical properties of the original library beam with the same number of equivalent sections. Each cutting section is located at a new specific *GRID* point.

Storing the output beam

Storing

```
        [path_PBEAM,path_PBEAML]=Converting_paths(MAT)
        [OK]=Storing_PBEAML(GRID,CBEAM,PBEAML,path_PBEAML,directory);
        [OK]=Storing_PBEAM(GRID,CBEAM,PBEAM,path_PBEAM);
        end
end
disp("Results stored")
```

```
Results stored
```

Figure V-18 Storing output beams (mapping script)

In the final stage (*Figure V-18*), each new *PBEAM and PBEAML ".bdf"* files are stored in the respective directory path for each beam. At the end of the external for-loop cycle, a new directory is automatically created inside the script folder. It reflects the same organization of the reference library, with the updated beams.

Graphical results check on Simcenter 3-D

Graphical examples are provided of the new beams (*Figure V-19* and *Figure V-20*), imported on Simcenter 3-D.

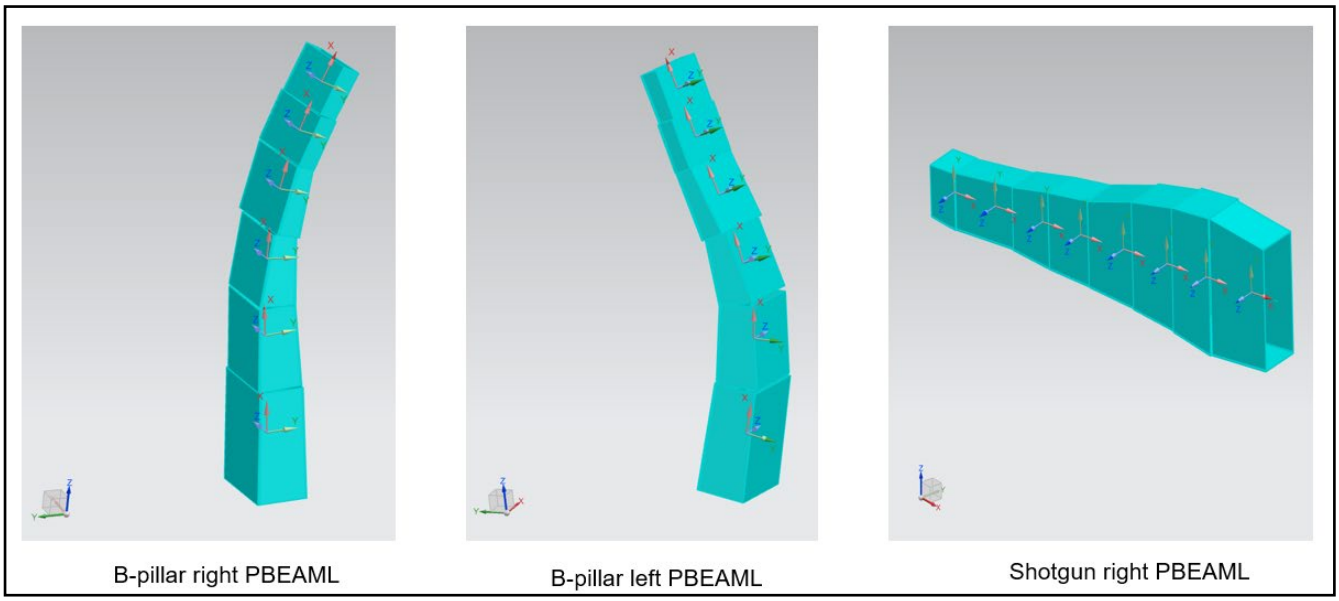


Figure V-19 From left to right: B-pillar right beam (Option 1), B-pillar left beam (Option 2), shotgun right beam (Option 3).

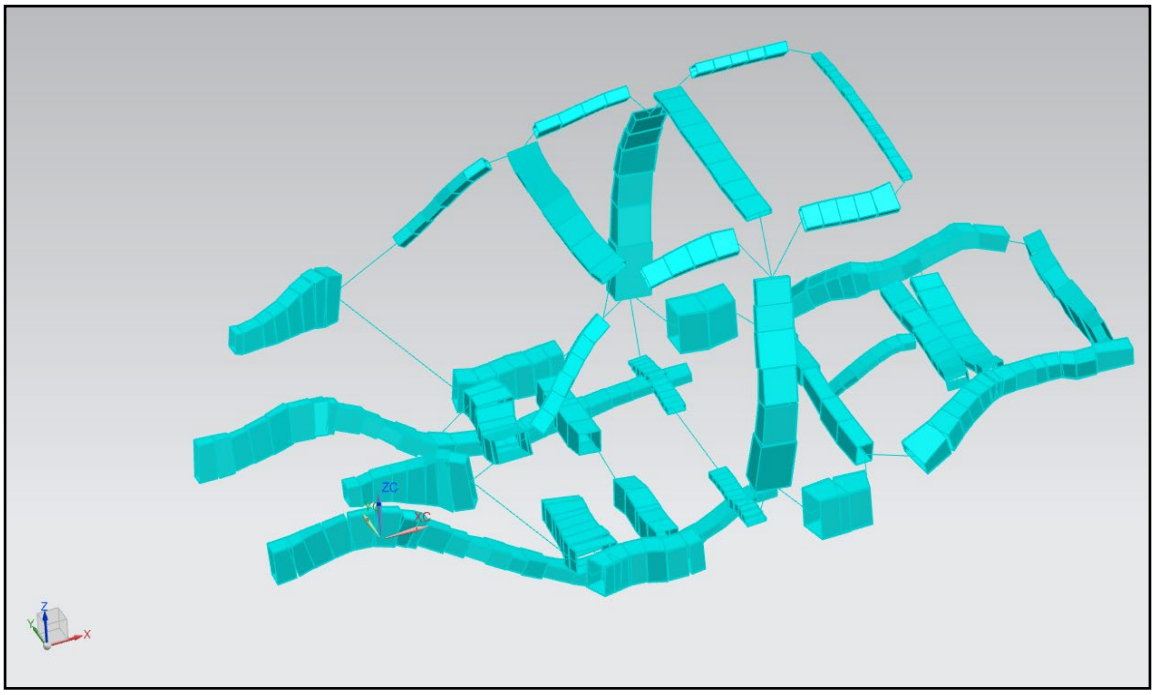


Figure V-20 Output concept BIW displayed on Simcenter 3-D.

VI. Setup of the optimization process

Introduction

In this stage, the process is automated through the use of the Siemens HEEDS toolbox. HEEDS is a powerful software package that automates the design space exploration process. Through the comparison over a wide range of designs that exhibit desirable characteristics, it is possible to conduct analyses and experiments. The data is automatically shared between different modelling and simulation products to assess design performance trade-offs and design robustness, competing objectives and constraints.

An overview of the automation loop implementation (Figure VI-1) is provided describing each sub-sequential block: Simcenter 3-D, Matlab, Nastran SOL103 for normal modes calculation, MAC and modes tracking. Two optimization problems are performed and analysed in terms of objectives: frequency and mass reduction.

The process is validated by means of different wireframes with a peculiar geometry. The same optimization is carried out and similar results are obtained.

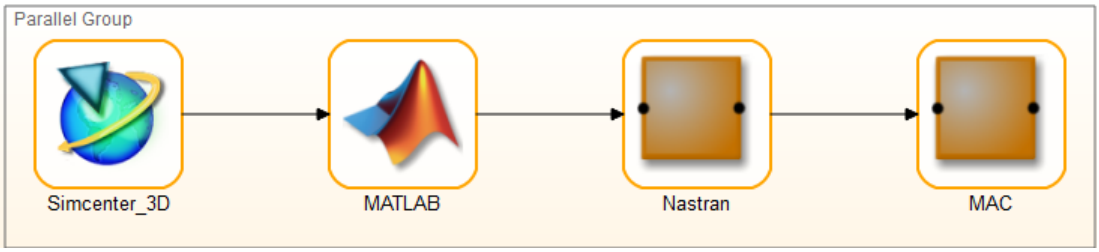


Figure VI-1 Implementation of loop on HEEDS.

Overview on HEEDS and optimization problem

HEEDS stands for "*Hierarchical Evolutionary Engineering Design System*". This software implements robust design exploration and optimization techniques able to look for recommended robust solutions within a given design space, in a reasonable computational time.

The idea is based on the production of multiple possible designs, while concurrently satisfying various criteria and using a wide number of evaluation variables. In a single run, *HEEDS* can drastically reduce manual efforts by producing different types of solutions. This operation may improve one or more aspects of the design, by optimizing the entire project under exam.

The optimization problem involves the selection of the *objective* or goal (desired aspect to be maximized or minimized) and the setting of the limitations needs to be applied (*constraints*). A feasible design is the one that most successfully meets the objective while staying within the set of constrains. Before launching the optimization problem, a *baseline* design is used as the frame of reference.

In the examined problem, the selected objective can be either the mass reduction or the frequency decrease. The constraints are the parameter range within which the wireframe can change its main dimensions. The baseline is a selected wireframe configuration associated with the reference beam library. The process has been validated by means of using two different wireframes with a different geometry, as shown in *Figure VI-2*.

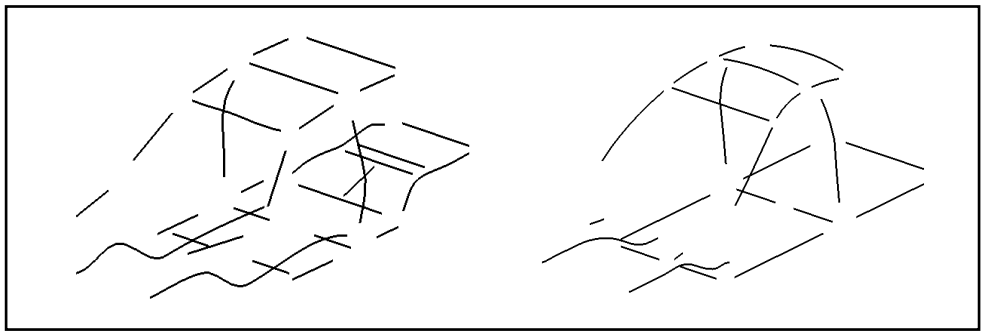


Figure VI-2 Wireframe configurations adopted in the optimization problem

At this point, *HEEDS* starts executing and intelligently searching in the design space the optimal design by picking values for the project variables. This operation can vary based on the specified number of times that it is executed and/or the number of variables involved. In no more than an hour, more than one hundred experiments are run with success.

Every design results directly associated with the respective performance rating. This index tells the degree of satisfaction of the specific design for the objective and constraints considered. The preferrable experiments are the ones with a high-performance rating. This strategy is adopted for making performance trade-offs and comparing design robustness.

By looking at the design of experiments (*DOE*) study, a better interpretation of the physics is provided by determining how variables influence the performance of the final design.

Setting up the project

Before launching *HEEDS*, the project is set by defining:

- ***Input and output files***. The input files correspond to the *baseline* design. It is a template that contains the variables used in the evaluation process. The template is represented from the wireframe geometry used as a reference. The output files contain the *responses* from a successful analysis run. The reference beam library is the expected output from a generic *HEEDS* run.

- ***Different commands.*** They execute the different tools used for the analysis. They are needed during the analysis for launching the different programs implemented on *HEEDS* in the proper way.

- ***Project variables and responses***. Project variables are the quantities modified at each run of *HEEDS* according to the set specifications. They are represented from the wireframe parameters of the main dimensions. Responses are the designated values in the output files.

- ***Tag of the values and files.*** The variables and responses located in the input and output files are selected, by choosing the desired I/O files directory location. Thanks to these tags, the variables of the optimization problem are linked to *HEEDS*.

- ***Loop blocks and assembly of the project***. The project is structured as a succession of different blocks linked together. At each block, a program is associated (*Matlab* for example). The blocks interact amongst each other by managing and exchanging the relative input and output files.

Workflow in HEEDS

Each experiment is the result of the launch of a simulation. In *Figure VI-3* an overview of the automation loop in *HEEDS* is shown.

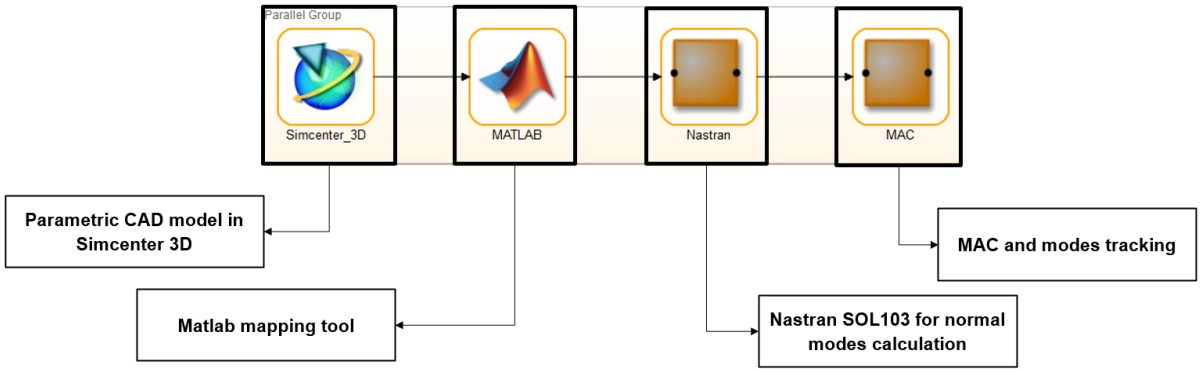


Figure VI-3 Description of subsequential blocks of the HEEDS loop.

The loop is composed by four sub-sequential blocks:

- ***Simcenter 3-D***: it is responsible for acquiring the parameter configuration selected from *HEEDS*. It generates the parametric *CAD* model for the wireframe configuration. This automatic operation is implemented by means of a basic *Python* code.

- ***Matlab***: creates the new beam library. As discussed in the previous chapter, the *Matlab* mapping tool uses the wireframe configuration and the reference library and automatically maps the new concept beams.

- ***Nastran solver:*** performs the selected types of analysis on the new concept library. The analysis selected is the normal modes calculation (*SOL103*). A static load case (*SOL101*) can be considered as well. Target mode is chosen as the global torsion mode.

- ***MAC and modes tracking***: the final check of the frequency response of the final model is shown in *Figure VI-4*. The target mode(s) can be tracked during the optimization/DOE process based on the MAC with the baseline target mode. The corresponding frequency is used in the definition of the optimization target function along with the total mass of the model.

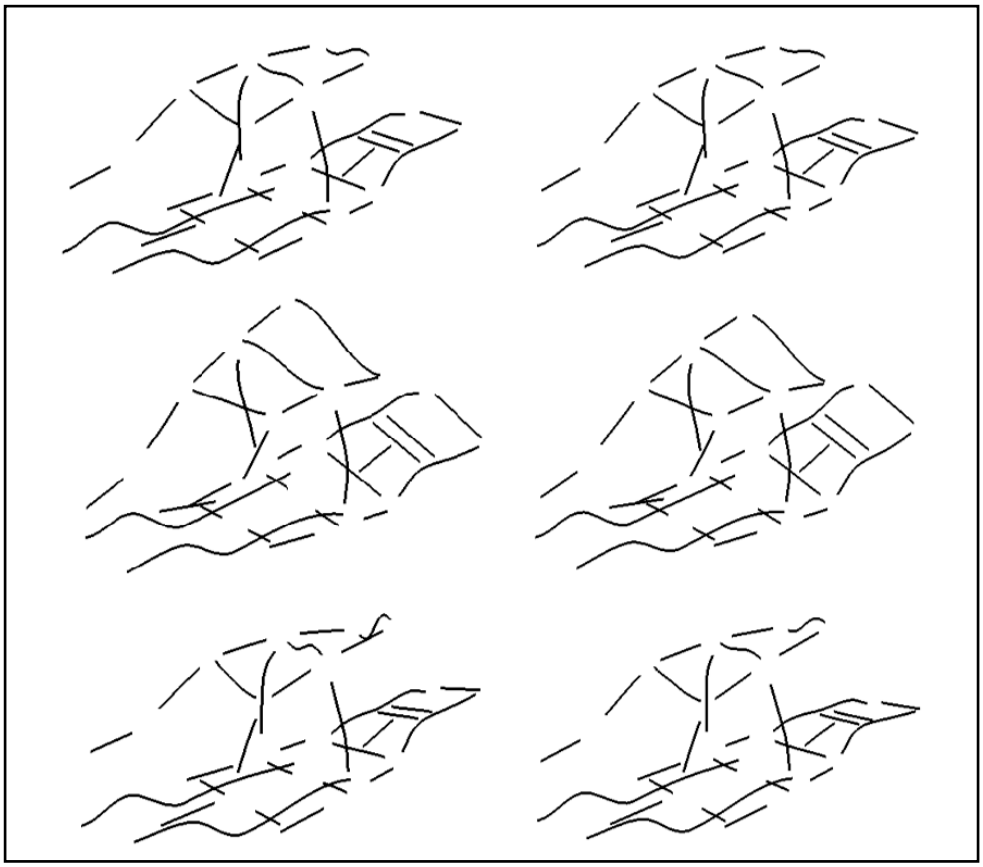


Figure VI-4 Wireframe responses to global torsion. On the left the new generated wireframe, on the right the baseline model used as reference during the modes tracking.

Output results

Figure VI-5 shows two optimization runs which have been performed. Target function for both runs is *torsion frequency/mass.* A total of one hundred experiments have been carried out per run with an elapsed time of half an hour. The created concept model can be run in about one second in Nastran.

In the first optimization, all geometric parameters of the wireframe were allowed to change within 20% w.r.t. baseline values. In the second optimization, the overall vehicle dimensions were "locked": vehicle length (*x*), breadth (*y*) and height (*z*); In both cases, with a very limited number of experiments the algorithm managed to increase the torsion stiffness (frequency) while reducing the total vehicle mass.

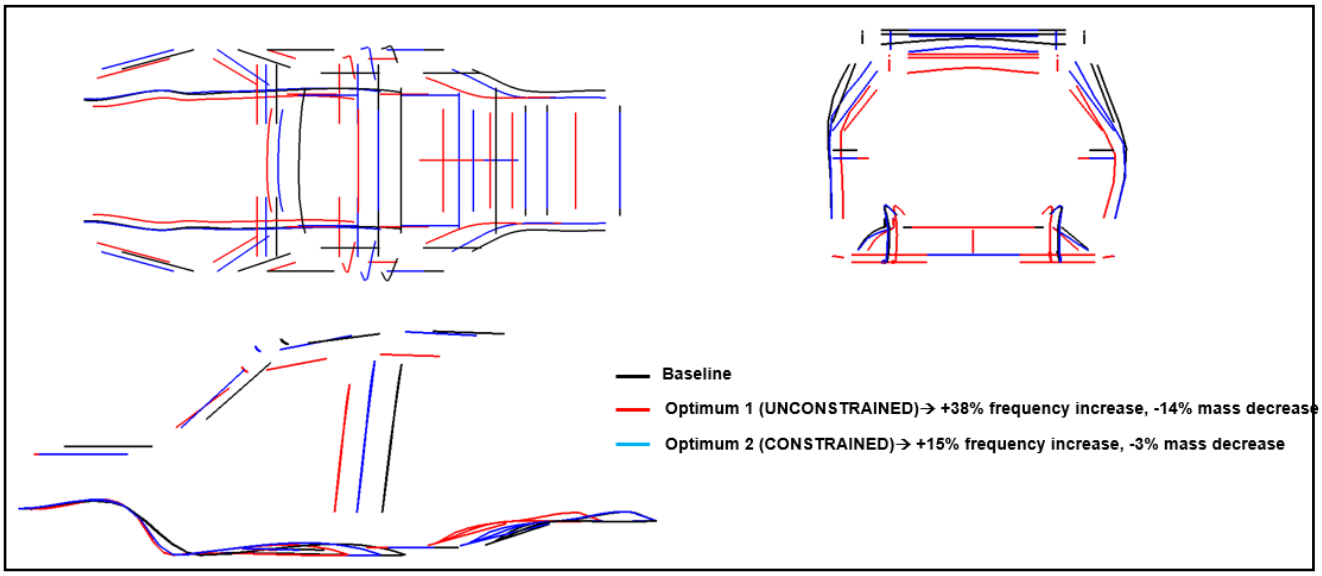


Figure VI-5 Optimization results.

Conclusions

This thesis work has been developed in collaboration with *Siemens Digital Industries Software* in order to develop a methodology for overcoming the current limitation on *CAE* models, in the early-stage design of a car.

By means of successive oriented cutting sections, evaluating the local properties and generating *HAT1* equivalent sections, each structural element is predicted. The geometric compatibility is preserved by keeping intact the mechanical properties. At the end of the procedure, equivalent beams are created.

Each beam is composed by *HAT1* sections linked together. The equivalent BIW results connected by different joints and *RBE2* elements. The whole structure belongs to a reference library used in the later stages.

As a second step, a simplified model called wireframe is generated. It represents a simplified version of the vehicle, in which all the beams and joints are predicted by means of straight lines, splines and nodes. It gives the possibility of tuning the different parameter dimensions that describe the geometric attitude of the vehicle. In this way, according to the customer choices, it is feasible to produce a wireframe that respects the main input specifications. The possibility of linking the physical design and target requirements, i.e., design parameters, represents one of the main prospectives of this method. It also allows to check the physics behind the model (torsion-bending stiffness behaviour).

The subsequent action consisted in the creation of a mapping script able to associate the defined wireframe with the existent reference library. Each beam is automatically rescaled, maintaining coherence with the respective geometric properties as inertia, section dimensions and orientation in the space.

The creation of a reference library allows the successive validation of the method, introducing the process into a basic *Heeds* loop. In this way, it is possible to automatically generate hundreds of equivalent simplified BIW models. The beam creation tool and the mapping script are optimized by checking the

correctness of the results generated resorting to Simcenter, Nastran, i.e., *SOL101 Static Analysis* and *SOL103 Normal Modes Analysis*.

The current limitations are related to a possible extension of the reference library, by including roof panels, other front beams and a better detailed representation of the remaining joints.

As a further improvement, the whole procedure could be applied for different types of vehicle models or for exploiting replacement of a single component by generating a hybrid car body. This is intended for improved design exploration at early-stage design.

Differently from the monocoque structure (where all body members are carrying load with chassis in-built with BIW and are integrated with each other) the capability of interchange structural elements makes the process flexible and adaptable to additional modifications and comparisons. It constitutes one of the main potentialities of this technique.

Bibliography

COMSOL. (n.d.). *multiphysics CYCLOPEDIA.* Retrieved from https://www.comsol.it/multiphysics/mesh-refinement.

Dean, J. D. (n.d.). *University of Cambridge Department of Materials Science & Metallurgy.* Retrieved from Introduction to the Finite Element Method (FEM).

Introduction to Finite Element Analysis. (n.d.). Retrieved from https://www.engr.uvic.ca/~mech410/lectures/FEA_Theory.pdf.

Matlab. (n.d.). *Mathworks.* Retrieved from www.mathworks.com.

Siemens. (n.d.). *NX Nastran 10, Quick Reference Guide.* Retrieved from https://docs.plm.automation.siemens.com/data_services/resources/nxnastran/10/help/en_US/tdocExt/pdf/QRG.pdf.

SIMSCALE. (n.d.). Retrieved from https://www.simscale.com/blog/2019/05/fea-for-beginners/#:~:text=FEA%20is%20the%20acronym%20for,heat%20conduction%2C%20and%20fluid%20flow.

Wikipedia. (n.d.). Retrieved from en.wikipedia.com.