Development of Mechanical/Structural Engineering tools for Analysis and Post Processing

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF TECHNOLOGY
IN
MECHANICAL ENGINEERING

Submitted by:
Dhruvin Jasoliya (14BME026)

MECHANICAL ENGINEERING DEPARTMENT
INSTITUTE OF TECHNOLOGY
NIRMA UNIVERSITY

Declaration

This is certify to that

- The thesis comprises my original work towards the degree of Bachelor of Technology in Mechanical Engineering at Nirma University and has not been submitted elsewhere for degree.
- Due acknowledgement has been made in the text to all other material used

Sign:

Name:

Roll No:

Undertaking for Originality of the Work

I, **Dhruvin Jasoliya (14bme026)** give undertaking that the Major Project entitled "**Development of mechanical/structural engineering tools for analysis and post processing**" submitted by me, towards the partial fulfillment of the requirements for the degree of Bachelor of Technology in Mechanical Engineering of Nirma University, Ahmedabad, is the original work carried out by me. I give assurance that no attempt of plagiarism has been made. I understand that in the event of any similarity found subsequently with any published work or any dissertation work elsewhere; it will result in severe disciplinary action.

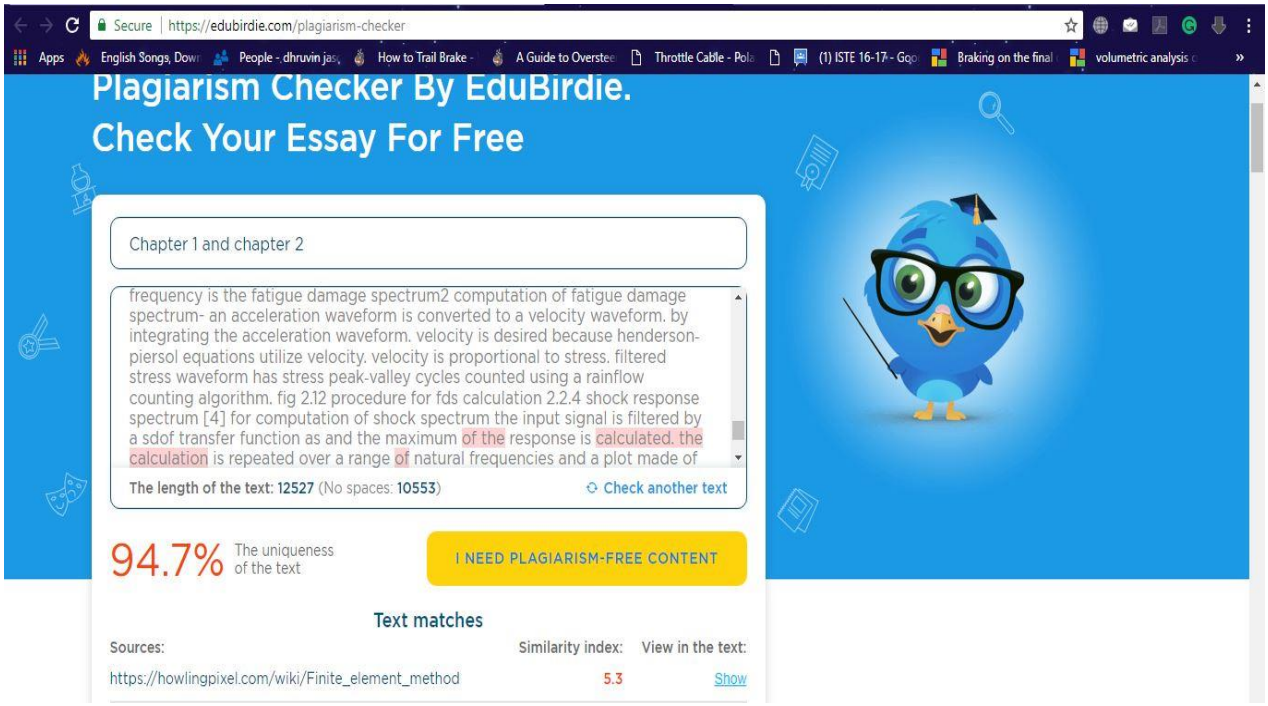
Signature of Student

Date:
Place: Ahmedabad

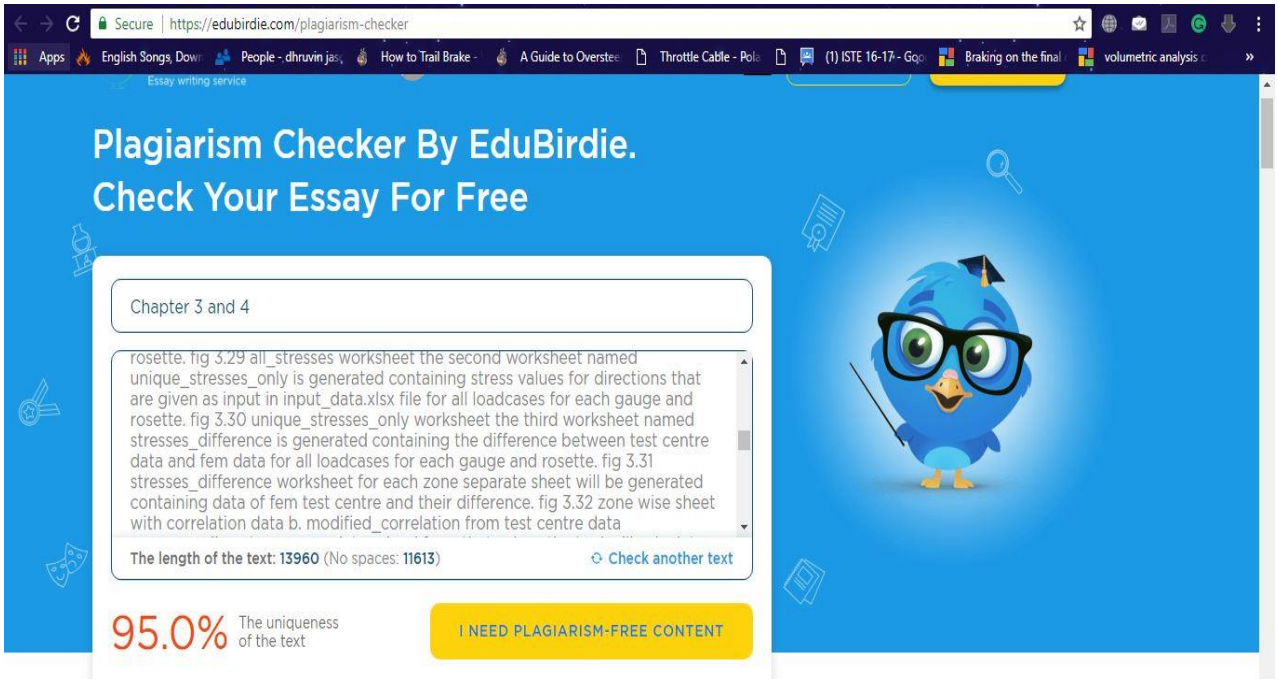
Endorsed by
(Signature of Guide)

Plagiarism Report

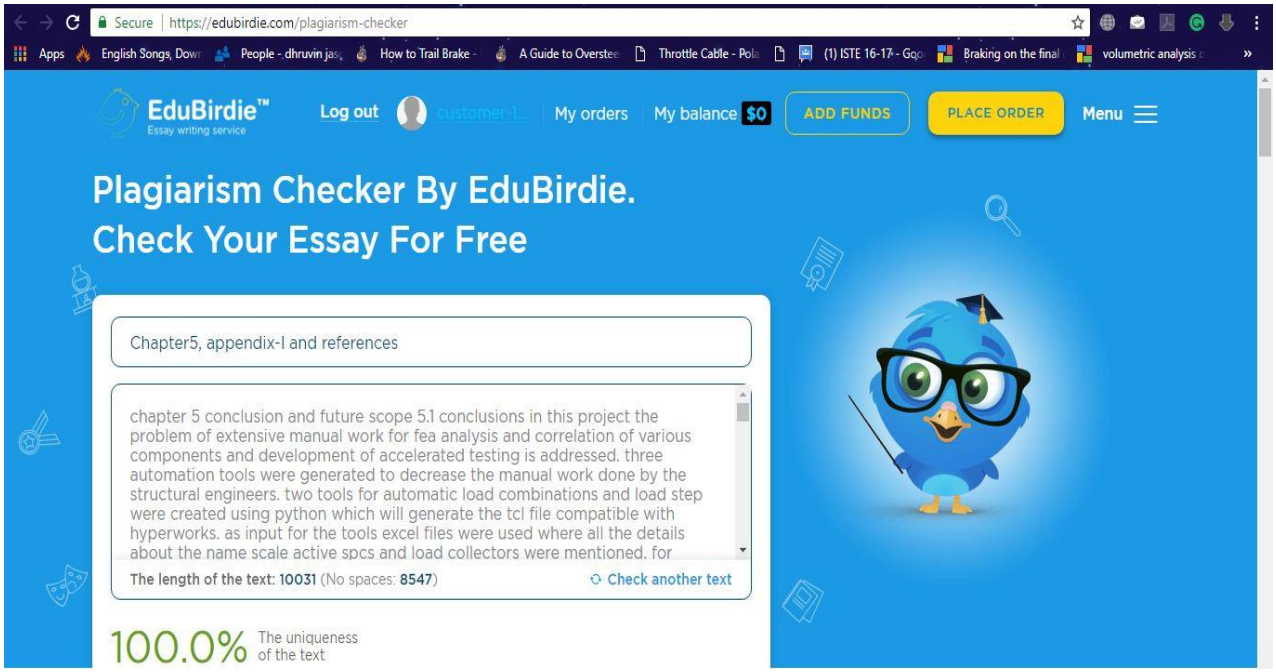
Chapter 1 and Chapter 2



Chapter 3 and Chapter 4



Chapter 5, Appendix-I and References



Certificate

TO WHOM IT MAY CONCERN

This is to certify that, Mr. Dhruvin Jasoliya, student of Mechanical engineering, 8th Semester of, Institute of Technology, Nirma University has satisfactorily completed the project report titled Development of mechanical/structural engineering tools for analysis and post processing.

Date:

Prof. Dhaval V. Patel
Guide, Assistant Professor,
Department of Mechanical Engineering,
Institute of Technology
Nirma University, Ahmedabad

Industry Guides:
Dr. Giovanni Rungen,
Chief Site engineer,
Rolling Stock Equipment,
Engineering centre, Bombardier Transportation, Hyderabad

Mr. Saurov Dey,
Lead structural mechanics Engineer,
Rolling Stock Equipment,
Engineering centre, Bombardier Transportation, Hyderabad

Dr. V.J. Lakhera
Head and Professor,
Department of Mechanical Engineering,
Institute of Technology
Nirma University, Ahmedabad

BOMBARDIER

TRANSPORTATION
Bombardier Transportation India Private Limited
(erstwhile Bombardier Transportation India Limited)

CIN: U74899DL1995PTC074938

c/o Cyient Ltd.
Tower B, A-Block Plot No. 2, IT Park,
Nanakramguda, Manikonda,
Hyderabad – 500 032, India

Tel: +91 40 23139096
Fax: +91 40 2311 5659

Registered Office: Asset No. 2, 3rd Floor, Novotel-Pullman Commercial Tower, Aero City, Delhi International Airport, New Delhi – 110 037

Hyderabad, 27th April 2018

CERTIFICATE OF TRAINING

Name	Mr. Dhruvin Jasoliya
Course	B. Tech (Mechanical Engineering)
Institute	Nirma University Institute Of Technology
Training Duration	2nd February 2018 to 27th April 2018
Department	Bogie
Projects Undertaken	Development of Mechanical /Structural Engineering Tools for Analysis and Post- Processing
Remarks	Hardworking and keen learner

We wish him all the very best for his future endeavour.

For and on behalf of
Bombardier Transportation India Private Limited

Sankalp Shobhit
HRBP - Hyderabad

Approval Sheet

The Project entitled **Development of mechanical/structural engineering tools for analysis and post processing** by **Dhruvin Jasoliya** is approved for the degree of Bachelor of Technology in Mechanical Engineering

Examiners

_____

_____

_____

Date: _____

Place: _____

ACKNOWLEDGMENT

First of all, I would like to thank, Engineering Centre, Bombardier Transportation India Private Limited, Hyderabad for giving me a great opportunity to carry out my final semester project.

I am very to Dr. Giovanni Rungen, for guiding me and providing an opportunity to work on this project. Also, special thanks to Mr. Saurov Dey, for continuous encouragement and support.

I would like to thank Prof. Dhaval V. Patel for their constant support and guidance throughout the project.

We would also like to thank to Mechanical Engineering Department of Institute of Technology, Nirma University for including industrial project program in the course curriculum to enhance the practical and industrial approach towards various concepts.

We are also grateful to one and all, which have helped me with their efforts in this venture.

ABSTRACT

For finite element analysis of bogie and carbody frame, large number of load-combinations and load-steps are defined in the Hypermesh. This process consumes extensive amount of time for an analyst. Also, for correlation of test centre results and FEM results, engineers were spending considerable amount of time in data organization and validation. Solution to these problems was to develop a python code which can automate the process of generation of load-combinations, load-steps and correlations files, taking input from the excel files which doesn't take much time in preparation. The same has been realized in the present report.

For vibration testing of the prototype before starting production, it must be tested in the same environment in which it is going to operate. Vibration response spectrum has to be obtained and is to be given to the shaker table for prototype testing. Earlier this process was outsourced to another company based in France. This reports describes the methodology for development of accelerated testing spectrums. The spectrum generation layout has been prepared in N-code. The spectrums generated are tested with the earlier available results to verify the reliable operation of the methodology.

Key words: Load-combinations, Load-steps, correlation, Vibration response spectrum, accelerated testing.

Table of Contents

List of Figures

Page no.

CHAPTER 1

Introduction

1.1 FEA procedure followed in industries

Finite element analysis is a numerical method for solving engineering problems. The finite element method formulates the problems in form of matrix and algebraic equations to solve it. Today, all the industries relies on FEA results to validate their product structural integrity. Based on the FEA results, the product design is finalized and drawings are prepared for production.

But the problems with FEA is the time spent behind pre-processing of model. Pre-processing includes definition of boundary conditions and different load cases. Also, correlation of FE results and Test centre data is very complex process and also the vibration testing of the component which takes lot of time during product introduction cycle. Today, with increasing competition in the market, each and every company has to address these problems and decrease time for product introduction to survive the stiff competition in market.

1.2 Company introduction

Bombardier Transportation is a worldwide pioneer in the rail business. The organization cover the full range of rail solutions, ranging from finish trains to sub-frameworks, support administrations, framework combination and flagging. The introduced base of rolling stock surpasses 100,000 rail products and trains around the world. Organization delivers ingenious rail transportation arrangements, including:

Rail vehicles - automated movers, monorails, light rail vehicles, rapid transit, metros, commuter trains, intercity/high-speed trains and locomotives

Propulsion and controls – Various traction motors and control systems

Bogies – Various range of bogies according to customer requirement

Services - maintenance, operations, vehicle modernization, and material management

Transportation systems - customized transportation system solutions

Rail control solutions - advanced signalling solutions for various systems

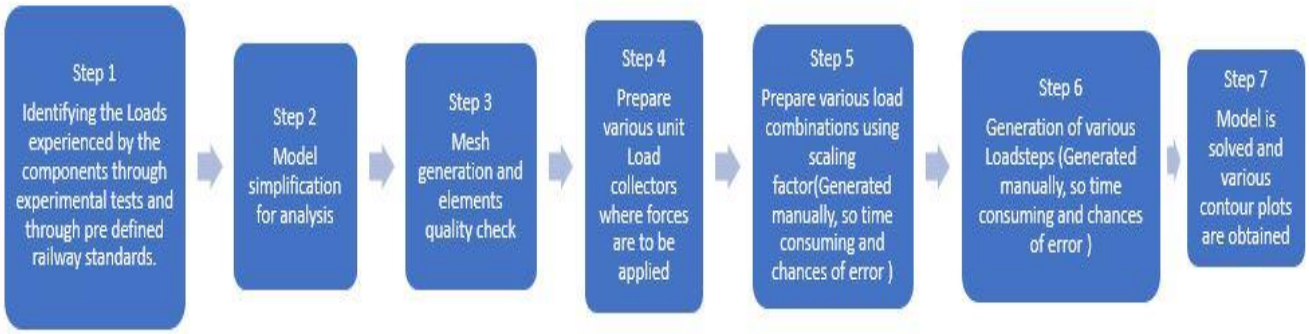
Fig 1.1 Queensland locomotive

Fig 1.2 Traxx locomotive

1.3 Objective identification

Structural mechanics team carries out static, fatigue, nonlinear, modal and thermal analysis of various bogie and carbody frames to validate the design of the various components as per the requirement of the customer.

General procedure followed for FE Analysis:



Problem in above procedure:

Load-combination and Load-step generation manually caused many errors and it was also very time consuming as the number of combinations and cases are very large. So, the objective was to automate this process.

Problem in correlation of test centre data and FE data:

- For the validation of the results obtained through FEM, a prototype is manufactured and then it is sent to test center. At test center, strain gauges and rosettes are fixed at the predefined nodes which are under high stress as per the simulations results and the readings are taken at these locations under different load cases and then correlation is done between test center data and data obtained from FE analysis.
- For, correlation of test centre data, engineer has to manually correlate for each gauge and it was time consuming process and further many other calculations also had to be carried put for rosettes. So, the objective was to automate this process.

Problem in accelerated testing:

Bombardier Transportation doesn't had any tool for generation of vibration response spectrum for prototype shaker testing, the objective is to develop the vibration response spectrum generation layout in the N-code such that vibration are similar to operating environment and also the amount of life time fatigue damage is also same. Parallel to this, also identify the errors in frequency spectrum generation and to rectify it.

1.4 Scope of work

- For automatic generation of Load combination and Load step, a python code has to be prepared which will generate a TCL file(compatible with Hypermesh), based on the input data given in the form of excel.
- For correlation process, a python code has to be prepared which will identify all the unique stress direction for all gauges and rosettes and correlate it. Also, prepare zone wise separate sheets for easy interpretations of result.
- A standard layout has to prepared in N-code where the time series signal obtaned from accelerometer is processed and using that signal an accelerated testing spectrum is prepared which will decrease the testing time of the component and generate realistc vibration results.

CHAPTER 2

Literature Review

2.1 Python-overview [1]

Python is most popular language for mechanical engineering programming problems and especially for automation. Some of the features which makes python more user-friendly compared to other programming languages are-

1. Python is Interpreted
   This language is not compiled as C or C++ but rather it is interpreted line by line, therefore it is easy to debug the code for any error.
2. Python is interactive
3. Python is object oriented
4. Python is beginner's language

2.1.1 Decision Making statements

Decision making is checking the value of the variables present in the program while execution and taking the actions accordingly. Decision structures evaluate various expressions and produce TRUE or FALSE condition, users instruct further actions for each outcome.

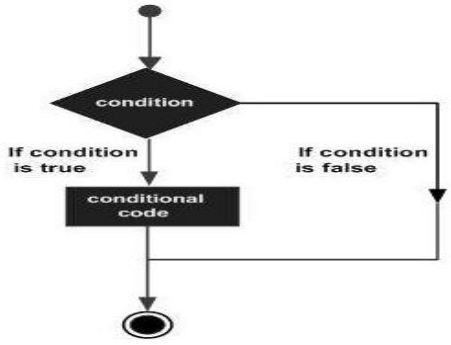


Fig 2.1 General decision-making structure

General decision-making conditions used in python-

1. If statements
2. If…. else statements
3. While statements
4. For statements

2.1.2 Functions

In python, there are some in-built functions and we can also define our own function and call it whenever it is needed. Recalling function every time make the code compact and avoids the duplication in the code. The example of defining the function and recalling it is given in the figure 2.2

```
#!/usr/bin/python

# Function definition is here
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist.append([1,2,3,4]);
   print "Values inside the function: ", mylist
   return

# Now you can call changeme function
mylist = [10,20,30];
changeme( mylist );
print "Values outside the function: ", mylist
```

Fig 2.2 function example

2.1.3 Modules

Modules in python, allows user to work more logically and easily. Without modules, code becomes lengthy and complex as each module has some predefined functions which makes code simple. Modules are used in python using *Import…* statement. Some of the modules used in code of automations for this project are as below-

1. Openpyxl module

   Openpyxl is the library module to create and manipulate data in excel files. Some of the examples of data handling using openpyxl is in the figures below.

write2cell.py

```
#!/usr/bin/python3

from openpyxl import Workbook

book = Workbook()
sheet = book.active

sheet['A1'] = 1
sheet.cell(row=2, column=2).value = 2

book.save('write2cell.xlsx')
```

Fig 2.3 writing in cell

appending_values.py

```
#!/usr/bin/python3

from openpyxl import Workbook

book = Workbook()
sheet = book.active

rows = (
    (88, 46, 57),
    (89, 38, 12),
    (23, 59, 78),
    (56, 21, 98),
    (24, 18, 43),
    (34, 15, 67)
)

for row in rows:
    sheet.append(row)

book.save('appending.xlsx')
```

Fig 2.4 appending values

read_cells2.py

```
#!/usr/bin/python3

import openpyxl

book = openpyxl.load_workbook('items.xlsx')

sheet = book.active

cells = sheet['A1': 'B6']

for c1, c2 in cells:
    print("{0:8} {1:8}".format(c1.value, c2.value))
```

Fig 2.5 reading cells

2. WX module

   WX module is used for preparing GUI (graphical user interface) for the code. Using this module user can give create prompt for the code execution as per the requirement. In this project, it is used for taking the location of the input file, location and name of the output file, user defined inputs and code execution commands.

3. Pyinstaller module

   It is the module that combined the python code and convert it into an executable file on windows, Linux and IOS. The main advantage of using Pyinstaller is that the executable files can be shared easily for large scale use and the other users apart from the programmer doesn't have to install python and any other extensions for the use. All three python codes created in the project are converted into executable file using Pyinstaller.

2.2 Accelerated Testing

2.2.1 Vibrations in Bogie Frames [2]

In train, during its operation vibrations will be there in longitudinal, lateral and transverse directions due to various factors.

1. The train will vibrate in longitudinal directions due to the tractive and braking forces that are changing continuously.
2. The vibrations in transverse directions are due to the differential action of the axle which is helping the train to take turns without slipping and also due the changing of tracks while in operation.

Fig 2.6 Axle on straight track

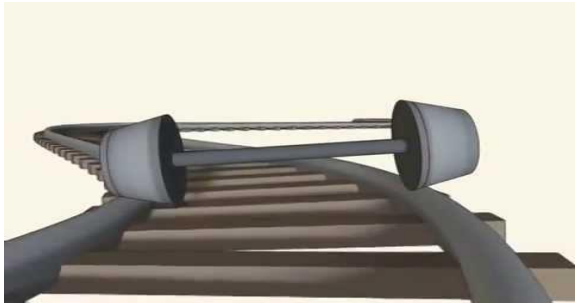


Fig 2.7 Axle while turning or changing track

3. The vibrations in lateral direction will be due to the changing stiffness of the track, as track support would be rigid when the wheel is on the wooden bile but there will be bending when wheel is not supported by wooden bile.

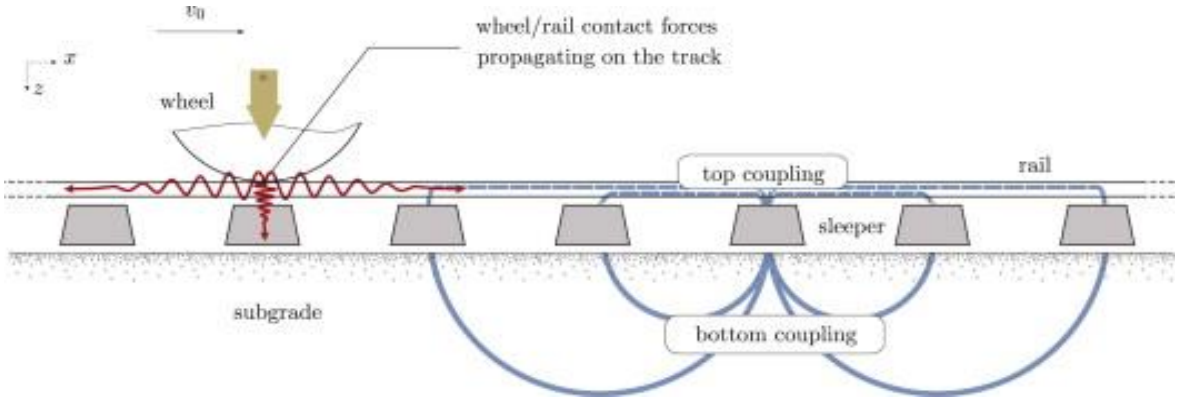


Fig 2.8 Lateral vibration Explanation

Why vibrations are important to monitor?

Vibrations are important to monitor for mainly two reasons:

1. The natural frequency of the component should not match with the frequency of vibrations as due to the resonance the component might fail.
2. Constant vibrations are kind of fatigue load only, if not kept under limit it might develop crack in the component and fail afterwards.

Hence, in design phase only the check on natural frequency of the component is kept through FE analysis and as Bombardier follows European standards, the natural frequency of all the components is generally kept above 100 Hz. But to validate the FEA results, before starting the production on the components, a prototype is made and is tested on the vibration table. If the components pass the vibration test then the production is started, else changes in the design of the component is done and again it is sent for testing.

For testing on vibration table, the excitation data is obtained through accelerometer attached to the train on the track, this time series data has to be repeated for no of cycles the train is designed to be sustained. But generally, if we test vibrations for whole life it would take more than 1 lakh hours, so the test time decreased by increasing the magnitude of the excitation such that the damage caused is same, but it should be within the limit such that component should not fail, this whole process is known as ACCELERATED LIFE TESTING.

2.2.2 Accelerated Life testing

Acceleration life testing is used to reduce the prototype testing time by increasing the amplitude or frequency of the vibration that the component experience in relative to the vibration level faced during normal service. This procedure is popular today because it makes the design validation accurate and make the new product development time less.

Vibration-based accelerated testing is done in four steps:

1. Environments assessments
2. Failure Mechanisms Identification

3. Damage Accumulation Rule identification and Specification of an Accelerated Test Sequence
4. Verification and monitoring

CLASSIFICATION OF ACCELERATED VIBRATION TESTING:

1. Increase vibration input amplitudes above service level;
2. Edit time series field data and use most severe observed amplitudes;
3. Find the operating condition that led to rapid damage growth and operate continuously at that level;
4. Increase the frequency level of the applied forcing function

2.2.3 Fatigue Damage spectrum [3]

For calculating the accelerated time for testing the fatigue damage spectrum is used.

CALCULATION OF FATIGUE DAMAGE SPECTRUM

Calculating Fatigue: S-N Curve, Miner's Rule

"The most commonly used method for calculating a reduction in test duration is the Miner-Palmgren hypothesis that uses a fatigue-based ***power law relationship*** to relate exposure time and amplitude."

A major assumption that usually accompanies the Palmgren-Miner linear damage hypothesis is that the ***slope of the S-N curve is approximately linear on a log-log plot.*** In the Henderson-Piersol1 approximations that undergird much of this algorithm, this assumption is stated as:

$$N = c \times S^{-b}$$

1. C is experimentally found between 0.7 and 2.2.
2. C is assumed to be 1.
3. -b is related to the slope of the S-N curve

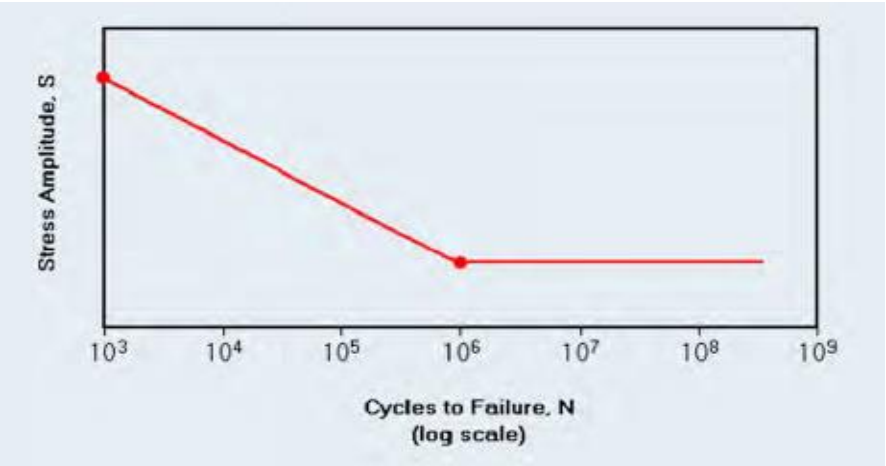


Fig 2.9 S-N curve

- Cycles are accumulated to get accumulated fatigue at that frequency; according to Miner's Rule

$$D = \sum_i \frac{n_i}{N_i} \qquad D \propto \sum_i n_i S_i^b$$

n*i*= number of cycles applied with peak stress, Si

N*i*= number of cycles with peak stress, Si, needed to cause failure

D= total damage

- Process is repeated at a spectrum of frequencies, with one fatigue number at each frequency.

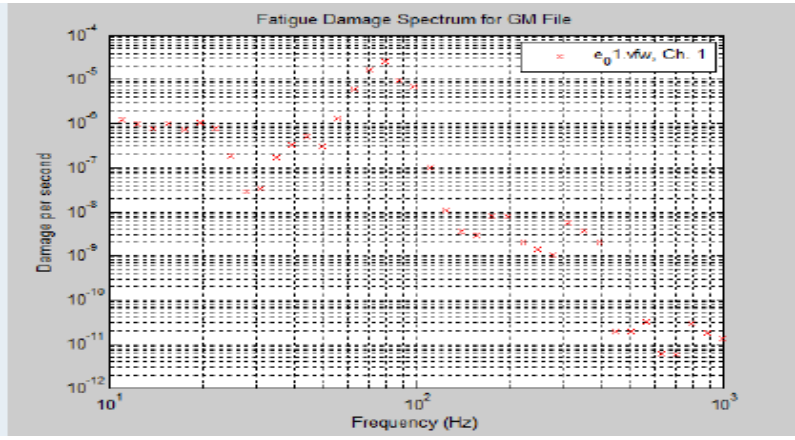


Fig 2.10 Damage calculated for each Frequency band

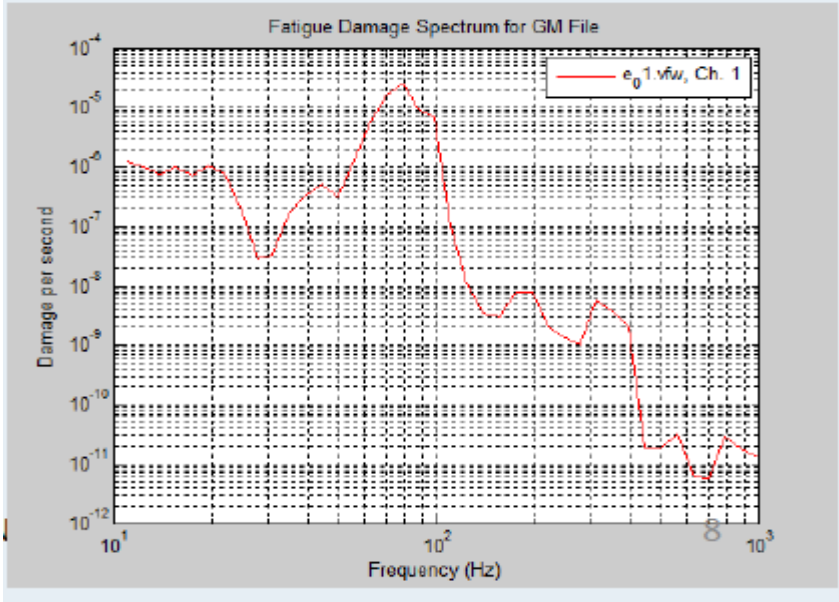


Fig 2.11 Plot of Fatigue vs. Frequency is the fatigue damage spectrum2

Computation of Fatigue damage spectrum-

An acceleration waveform is converted to a velocity waveform. By integrating the acceleration waveform. Velocity is desired because Henderson-Piersol equations utilize velocity. Velocity is proportional to stress. Filtered stress waveform has stress peak-valley cycles counted using a Rainflow counting algorithm.

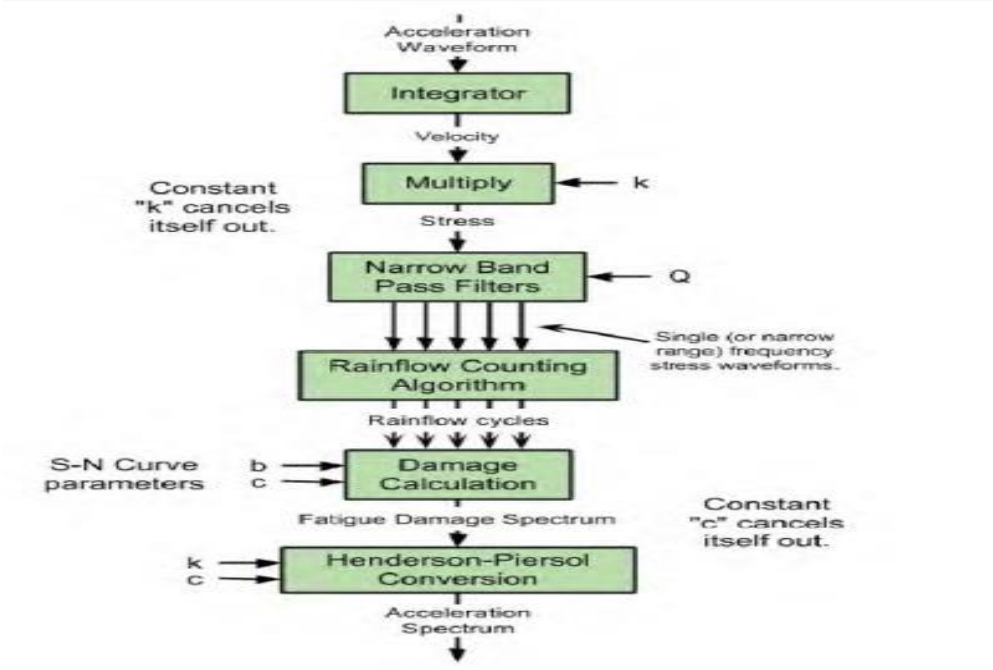


Fig 2.12 Procedure for FDS calculation

2.2.4 Shock response spectrum [4]

For computation of Shock Spectrum, the input signal is filtered by a SDOF transfer function as and the maximum of the response is calculated. The calculation is repeated over a range of natural frequencies and a plot made of the maximum response, the natural frequency. The Shock Response Spectrum (SRS) can be expressed in terms of acceleration or displacement response depending on the frequency response function used. For fatigue purposes, most important is displacement response. Fatigue cracks initiate and grow through the cyclic release of strain energy and therefore the displacement response provides a proportional relationship with the energy driving fatigue failure. Acceleration is the origin of the load but it's the resulting strain (displacement) that drives the structural failure. The SRS of displacement can therefore be used to quantify the damaging effect of the input a for any SDOF system over a range of natural frequencies.

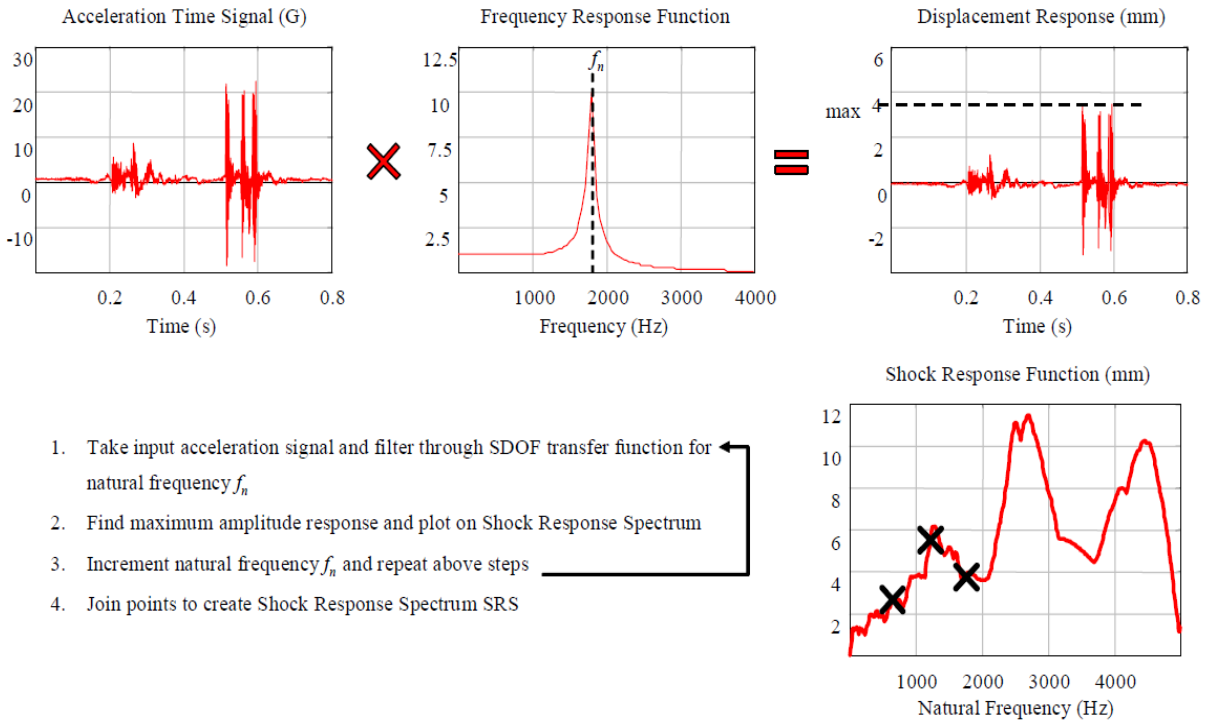


Fig 2.13 SRS calculation Procedure

CHAPTER 3

Automation tools for post processing and correlation

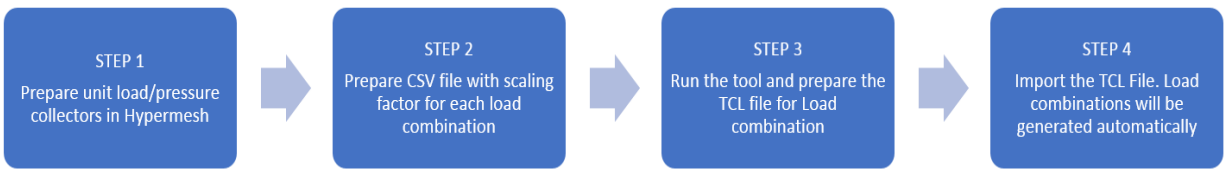
3.1 Load case combination

For a given set of unit loads, each load case should be entered manually in Altair Hyperworks software which is a tedious job. The task was to automate the input process of different load case combinations for a pre-set unit load cases. This task can be automated by using TCL (Tool Control Language) script, which is the only script that can be recognized by the Altair Hyperworks. To create the TCL script, Python programming language is used. The python code takes the inputs from a csv file containing the load case combinations in a pre-set format and outputs a TCL file which can be recognized by Altair Hyperworks. The TCL code when executed in Hyperworks automatically generates the required load cases. GUI is also made for the same in which we need to enter the required inputs to create the desirable output. (The GUI is in an executable file, i.e., it doesn't require the python software to be pre-installed to run the software in windows)

3.1.1 Python code logic for load case combination generation

- Reading data from the CSV file prepared by the engineer with IDs of all unit load collectors, name and corresponding scale for each load combination.
- After, reading the data from the csv file, the code will generate a TCL for each Load combination with IDs and scale for each unit load collector.
- The part of the code is written in TCL language as Hypermesh only accepts automation files in this language.
- At last, GUI layout and dimensions are prepared for easy use of the tool without any complications and errors.

3.1.2 Overview for load combination generation



- Previously in Hypermesh file itself, load combination was done by combining the unit loads/pressures collectors with a scaling factor.
- With load combination tool, any number of combinations can be done.

3.1.3 Test run for load combination generation

Working folder of macro tool (For load combination generation):

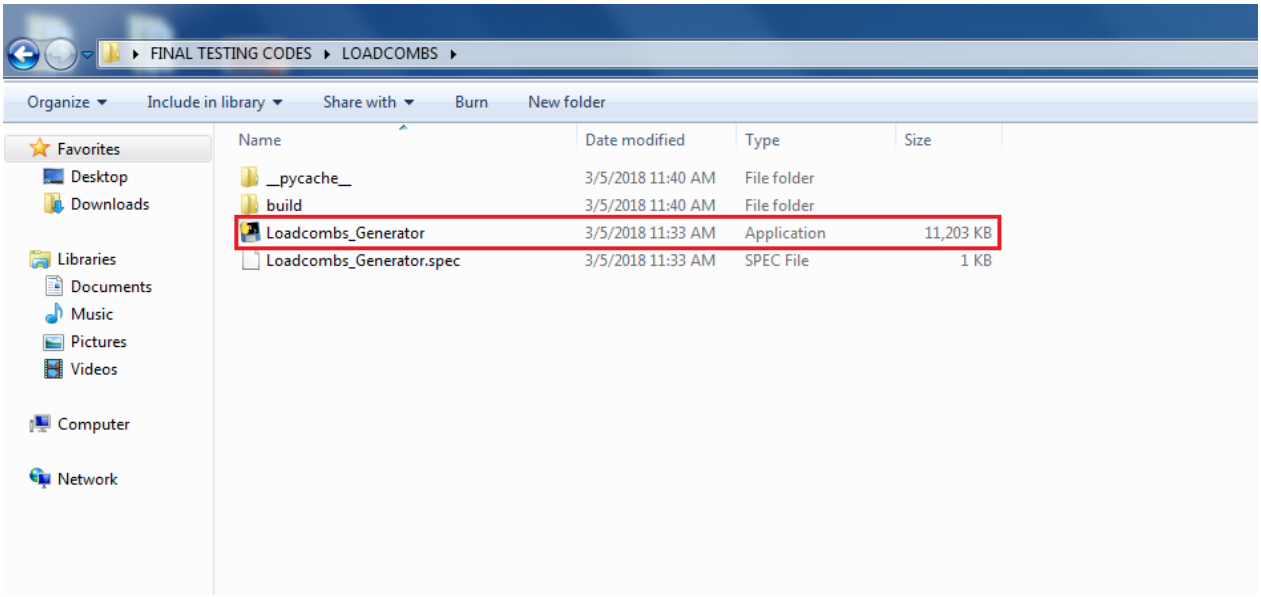


Fig 3.1 working folder

- Step-1

  Prepare the unit load collectors in the Hypermesh.

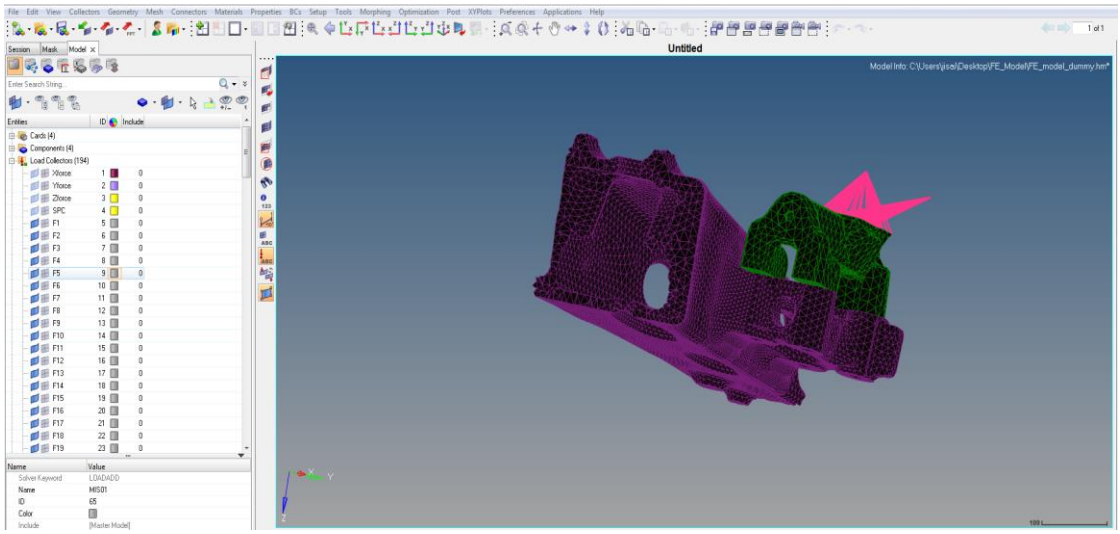


Fig 3.2 Unit load collectors in Hypermesh

- Step 2

  Prepare the input csv file containing the load cases for a set of unit loads in the format as shown in the fig 3.3. Read the instructions carefully. (Note that the unit load cases IDs in CSV file should be same as the HM file)

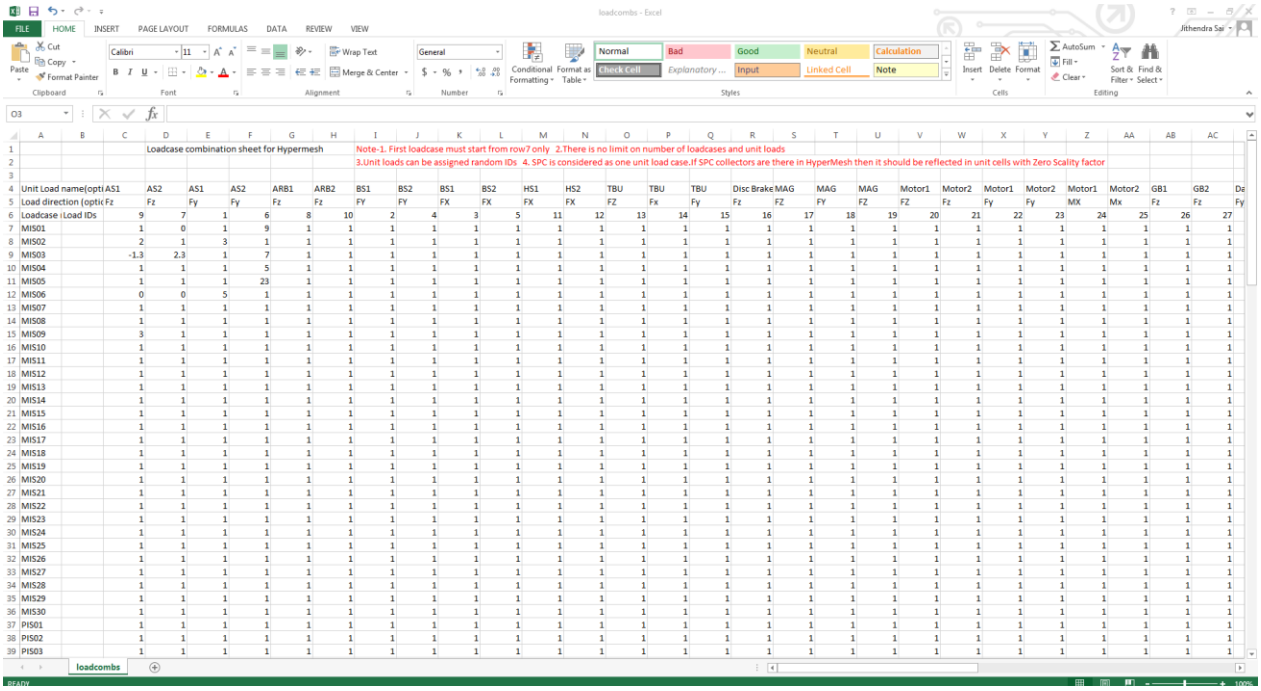


Fig 3.3 Sample excel file for load combination generation

- STEP 3

  Open the software and enter the following details as shown in fig 3.4.

- STEP 4

  Click generate to create the TCL file.

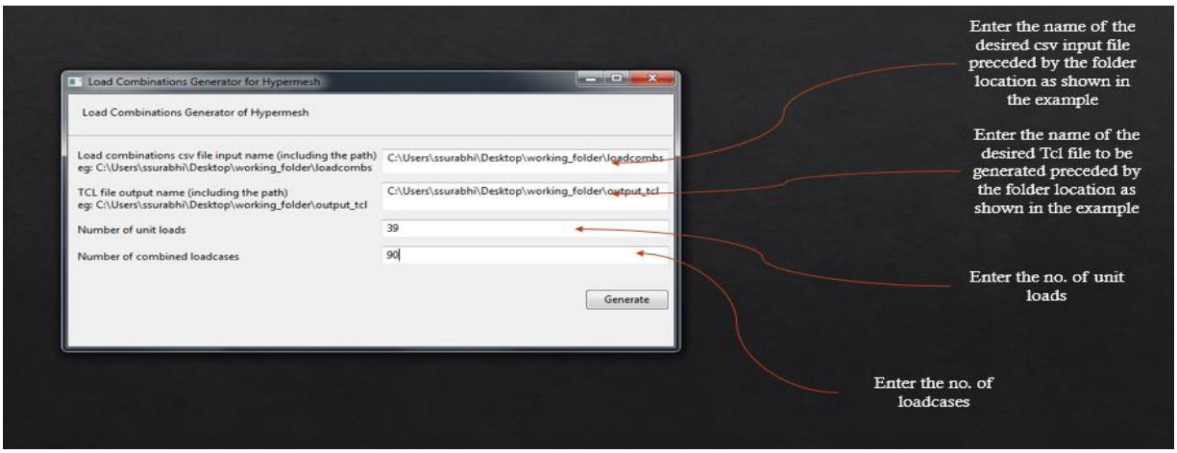


Fig 3.4 GUI for load combination generation

- STEP 5

  Run the TCL file in Hyperworks. (File>Run>TCL script)

- STEP 6

  All the required load cases are generated in Hypermesh as shown in the fig 3.5.

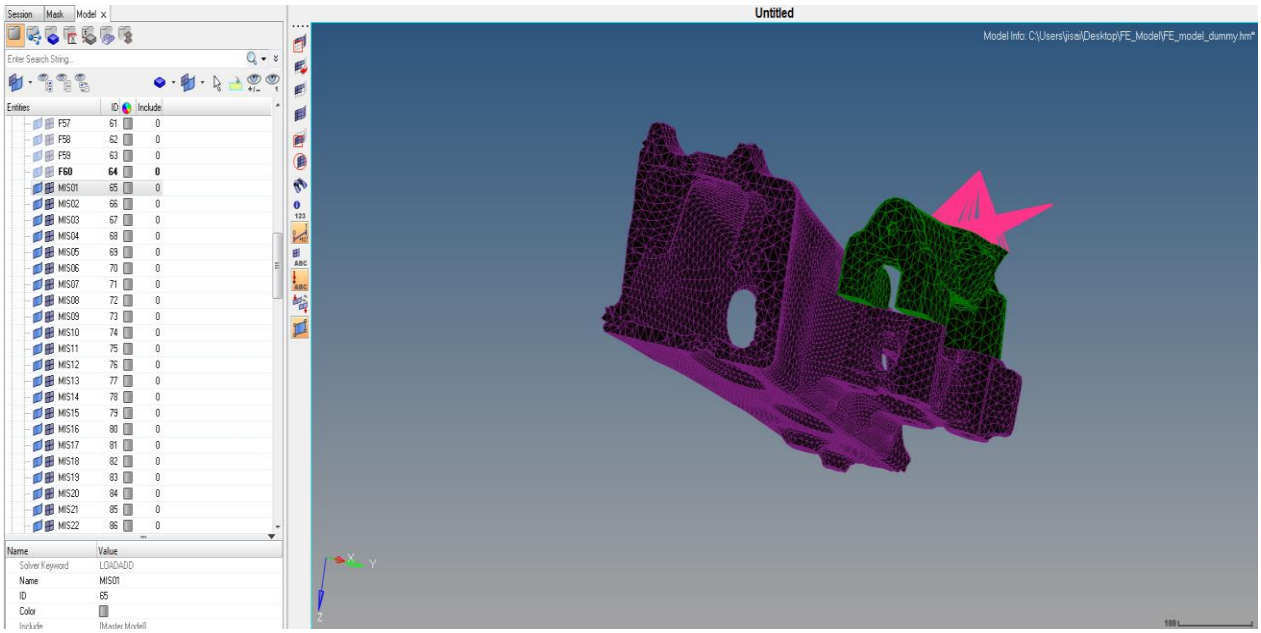


Fig 3.5 Generated Load combinations in Hypermesh

Error notification

- After, clicking on generate button, “LOADCOMBS TCL FILE IS GENERATED” in the .exe window is program is successfully completed. If there is some error in input file, it will display ‘LOADCOMBS TCL FILE IS NOT GENERATED’. Both cases are displayed in figures below.

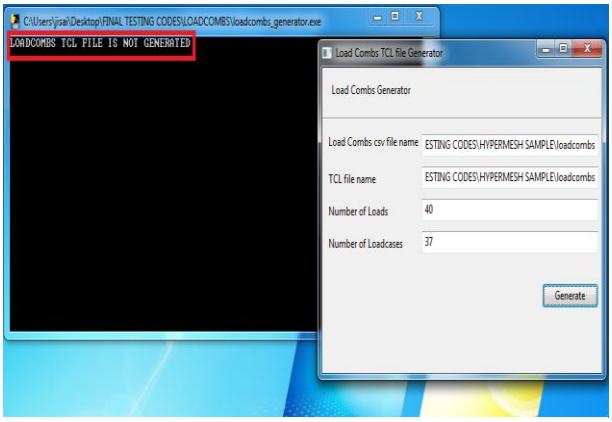


Fig 3.6 Case of error

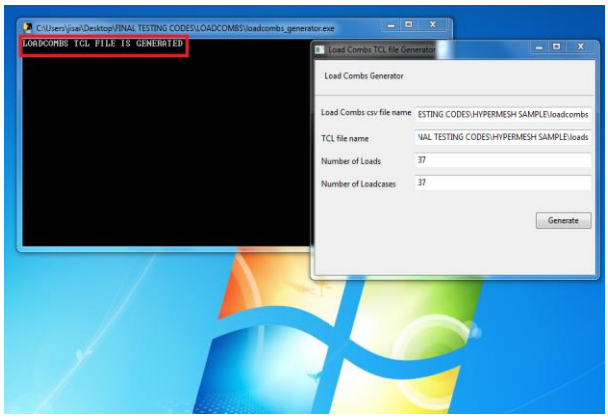


Fig 3.7 Successful execution

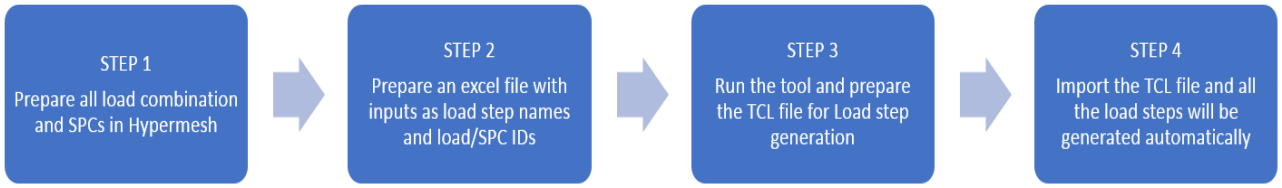
3.2 Load-step generation

- Similar to Load combination Generator, large number of load steps are to be generated manually. For the automation of it, python code is generated which will take input from the .xlsx file and prepare a TCL file containing the name of the Loadstep and its corresponding LOAD and SPC IDs and names. Hence, by the use of this tool pre processing time is reduced by great extent.

3.2.1 Python code logic for Load Step generation

- Reading data from the excel file prepared by the engineer with names and IDs of all Load collectors and SPC along with the names of each Load step.
- After, reading the data from the excel file, the code will generate a TCL for each Load step name and its corresponding Load collector and SPC IDs and name.
- The part of the code indicated in green is written in TCL language as Hypermesh only accepts automation files in this language.

3.2.2 Overview of Load step generation



- Previously in Hypermesh file itself, for Load steps LOAD and SPCADD (for constrain) must be selected manually.
- With load step tool, any number of combinations can be done.

3.2.3 Test run for load step generation

Working folder of macro tool (For load step generation)

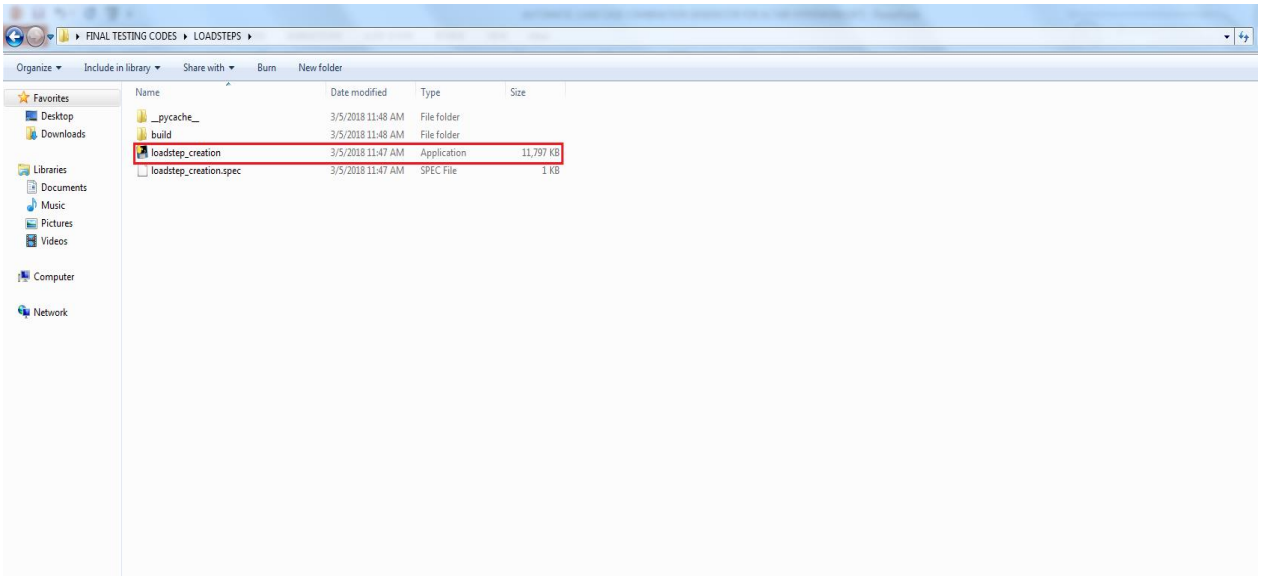


Fig 3.8 Tool working folder

- Step 1

  Create all the Load-combinations and SPC's which are required for generating Load steps in Hypermesh.

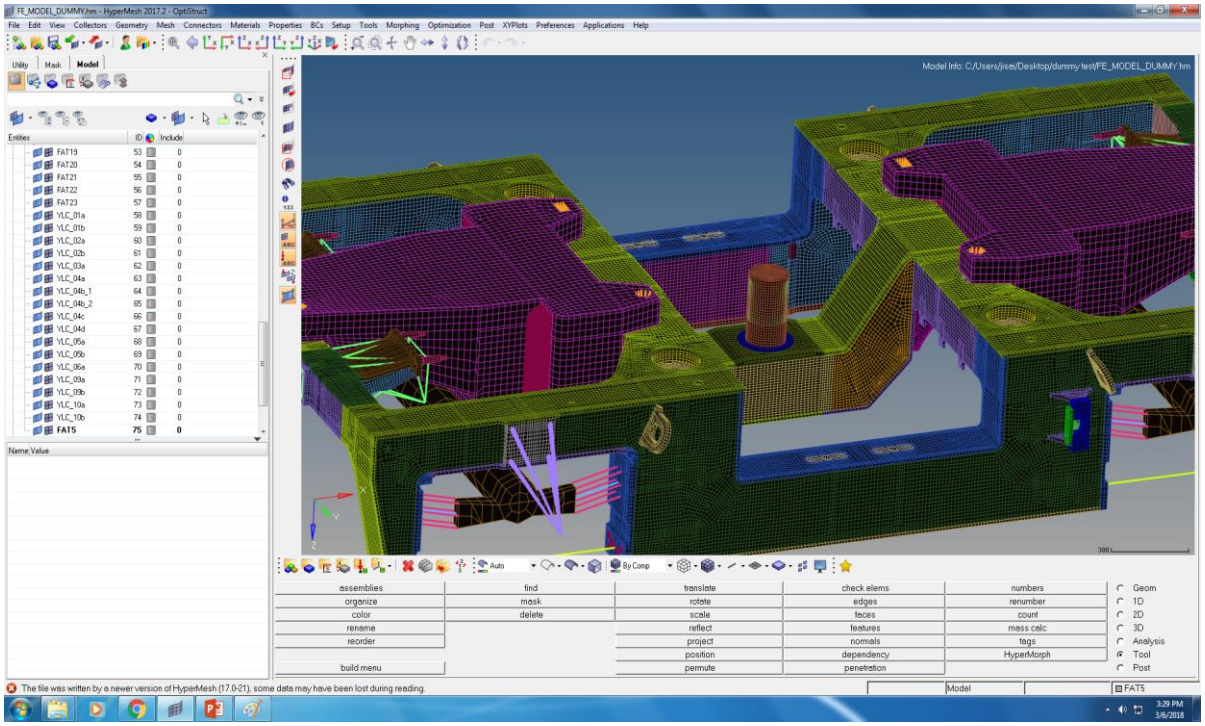
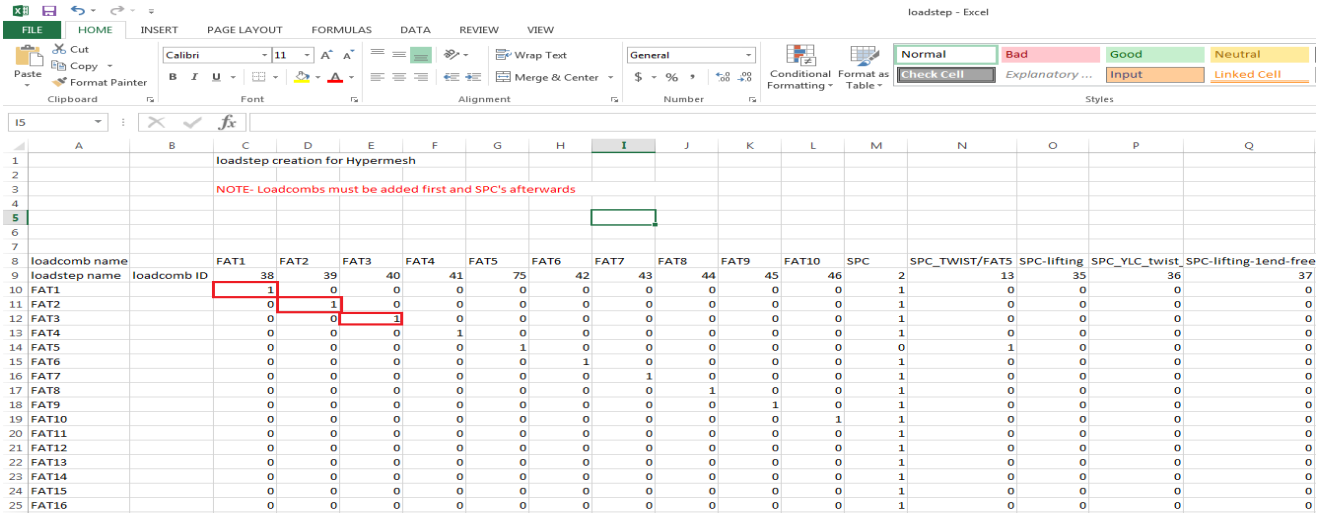


Fig 3.9 Creating Load combination and SPC in Hypermesh

- Step 2

  An excel file has to be generated as shown in the fig. Each Loadstep is defined using two ‘1’, for load and SPC. Rest all cells in that column must be zero.



loadstep - Excel

loadstep creation for Hypermesh

NOTE- Loadcombs must be added first and SPC's afterwards

loadcomb name		FAT1	FAT2	FAT3	FAT4	FAT5	FAT6	FAT7	FAT8	FAT9	FAT10	SPC	SPC_TWIST/FAT5	SPC-lifting	SPC_YLC_twist_	SPC-lifting-1end-free
loadstep name	loadcomb ID	38	39	40	41	75	42	43	44	45	46	2	13	35	36	37
FAT1		1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT2		0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT3		0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
FAT4		0	0	0	1	0	0	0	0	0	0	1	0	0	0	0
FAT5		0	0	0	0	1	0	0	0	0	0	0	1	0	0	0
FAT6		0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
FAT7		0	0	0	0	0	0	1	0	0	0	1	0	0	0	0
FAT8		0	0	0	0	0	0	0	1	0	0	1	0	0	0	0
FAT9		0	0	0	0	0	0	0	0	1	0	1	0	0	0	0
FAT10		0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
FAT11		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT12		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT13		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT14		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT15		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
FAT16		0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Fig 3.10 Excel preparation for Loadstep

- Step 3

  Open the software and enter the following details as shown in fig 3.11.

- Step 4

  Click generate to create the TCL file.

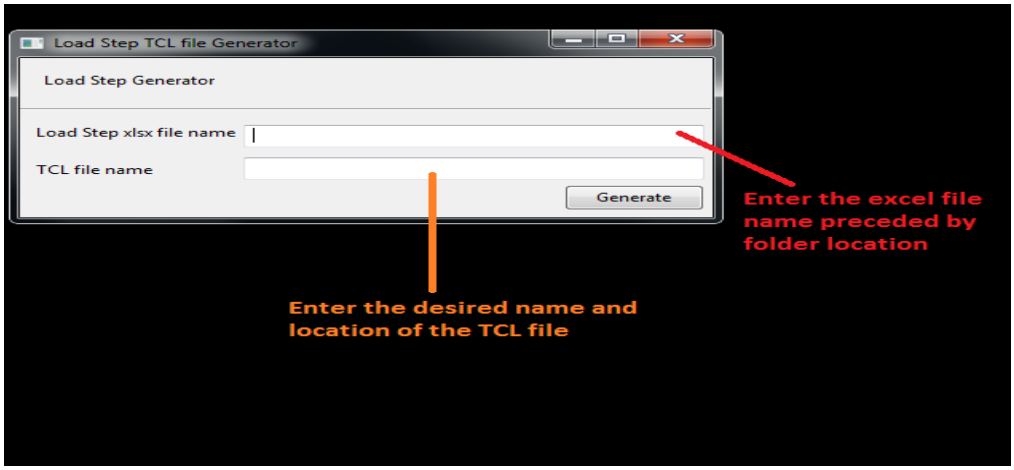


Fig 3.11 GUI for Load step generation

3.3 Correlation tool

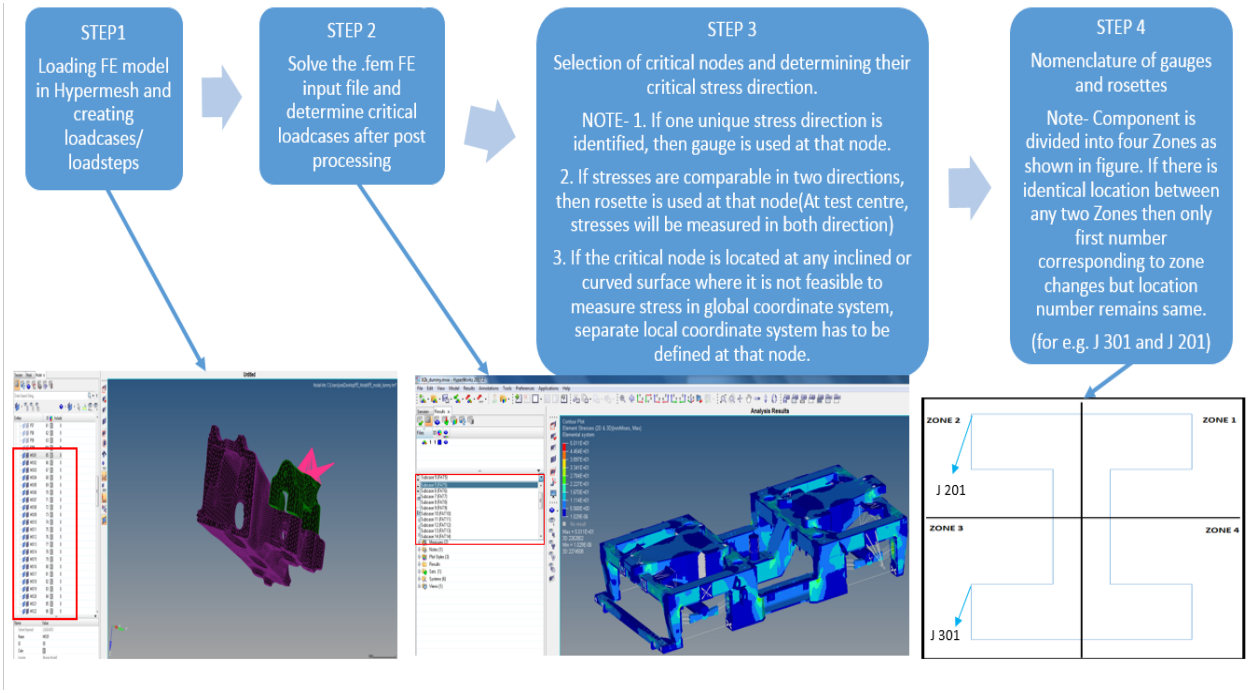
- For the validation of the results obtained through FEM, a prototype is manufactured and then it is sent to test center. At test center, strain gauges and rosettes are fixed at the predefined nodes which are under high stress as per the simulations results and the readings are taken at these locations under different load cases and then correlation is done between test center data and data obtained from FE analysis.
- It is also a time-consuming process to correlate results at each node, so for this purpose also python programme is created to automize the process of correlation and validation and if there is any problem, changes are done accordingly.

3.3.1 Python code logic for Stress Sheets Generator for Correlation

- In python, openpyxl module is used, which used for reading, creating and writing in excel files.
- In this part of program, input_data excel is read by the program and there are two worksheets jauges and rosettes. So, values from each cell of both worksheets is read and recorded by the code.
- Different variables are there for storing the data read from the excel file such as gauge_nos, node_ids, Jauge_stress_direction, Coord_sys, etc.
- . append function is used to add all the data into the assigned variables.
- Code for GUI (Graphical User Interface) using WX module available in python.
- Size of the GUI tab, position of the radio buttons and their names are defined in the code.
- From in input_data excel file, all nodes are stored along with their coordinate system.
- A variable named systems is defined, if the coordinate system is already present in the code, the node will be added in that set else a set containing new coordinate system is corresponding to the node will be created.
- After that, the text files for each coordinate system is made containing all the corresponding nodes along with the syntax required to be imported in Hyperworks as node set text files.

- The text files generated will be used later for generating stress values at corresponding nodes in their pre-defined coordinate system which will be used as input for second part of code.
- For reading the data from the nodal stress reports and storing it into a variable matrix.
- These matrix values are then displayed in the new worksheet named ‘all_stresses’, where for each gauge and rosette the value of XX, YY, ZZ, P(min), P(mid) and P(max) are there for each loadcase.
- Sorting is done for unique stress direction stresses for each gauge and rosette, and it is stored in the variable matrix.
- The variable matrix containing unique stress values are printed in the new excel worksheet named ‘unique_stresses_only’.
- Data from correlation file obtained from Test centre is stored into a variable matrix.
- A new matrix is defined containing the difference of FE data and test centre data and is displayed in new worksheet named ‘stresses_difference’.
- The values of sigma 1, sigma 2 and phi has to be calculated from the test centre data as it is not readily available. So separate worksheet named ‘Update PI PII’ is created, where calculated values of above stresses is displayed
- It is also stored in separate variable as this matrix will be used while creating zone sheets for rosettes.
- For zone wise sheets generation, sorting of each gauge and rosette is done to their corresponding zone.
- The FEM data is first displayed for each loadcase and after that test centre data is displayed.
- Now, the difference of test centre data and FEM data is displayed and if the difference is significant then appropriate steps should be taken.
- Atlast, the excel sheet is saved under name of ‘stress_sheets_final’.

3.3.2 Overview of correlation process



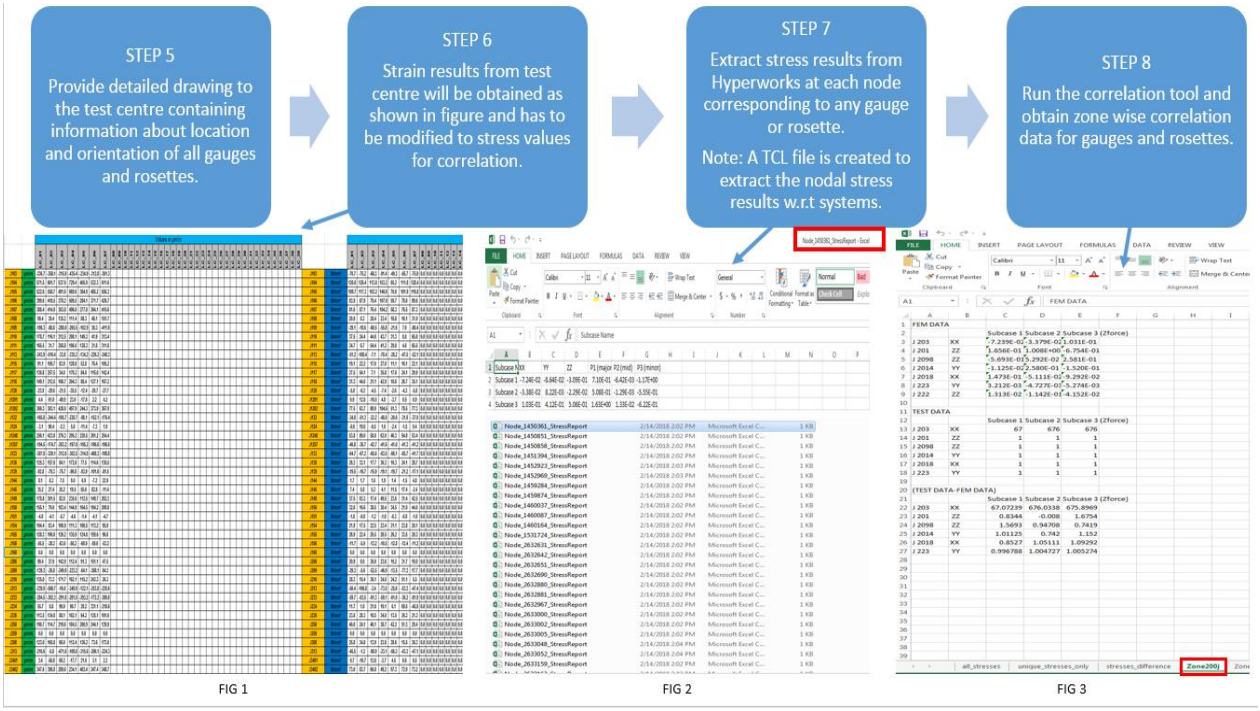


FIG 1 FIG 2 FIG 3

3.3.3 Test run for correlation tool

Working folder of tool

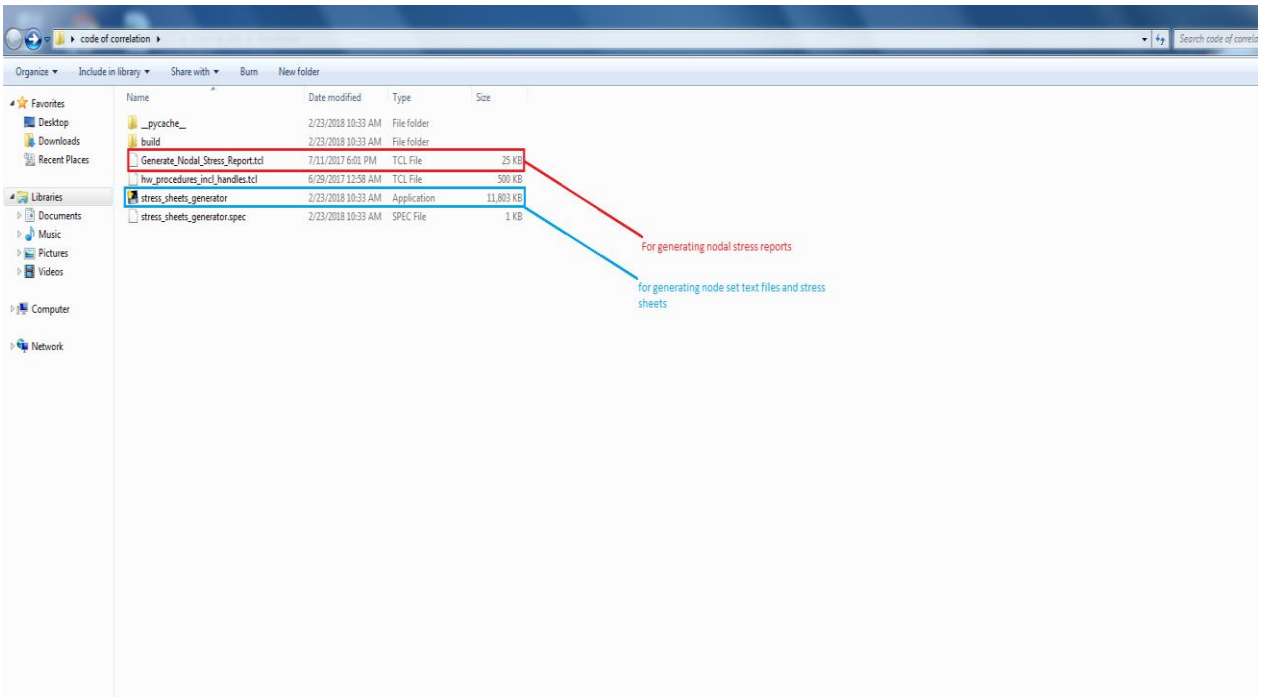


Fig 3.12 Working folder for correlation tool

Step 1 – Open Hyperworks, analyses results for all loadcases, determine nodes for gauges and rosettes along with their node number, stress direction and coordinate system (i.e. local or global)

a. For creating a new local coordinate system, create>Systems (shown in the fig 3.13)

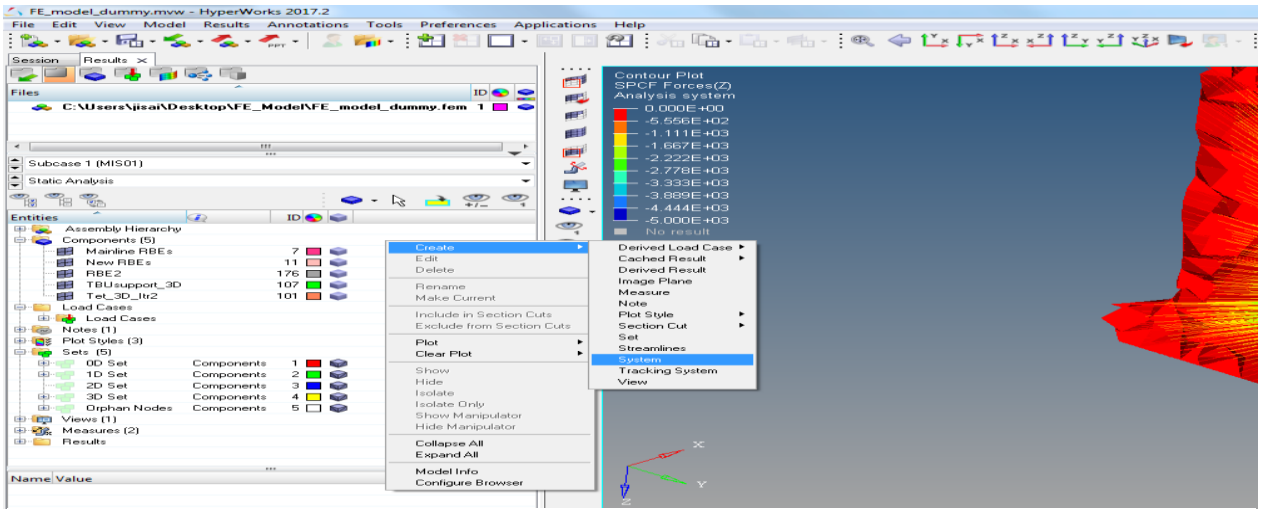
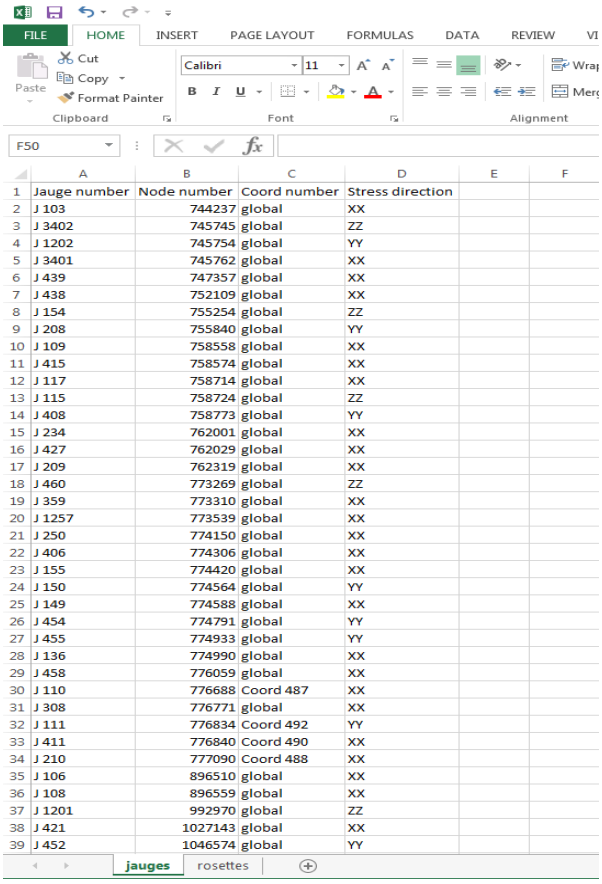


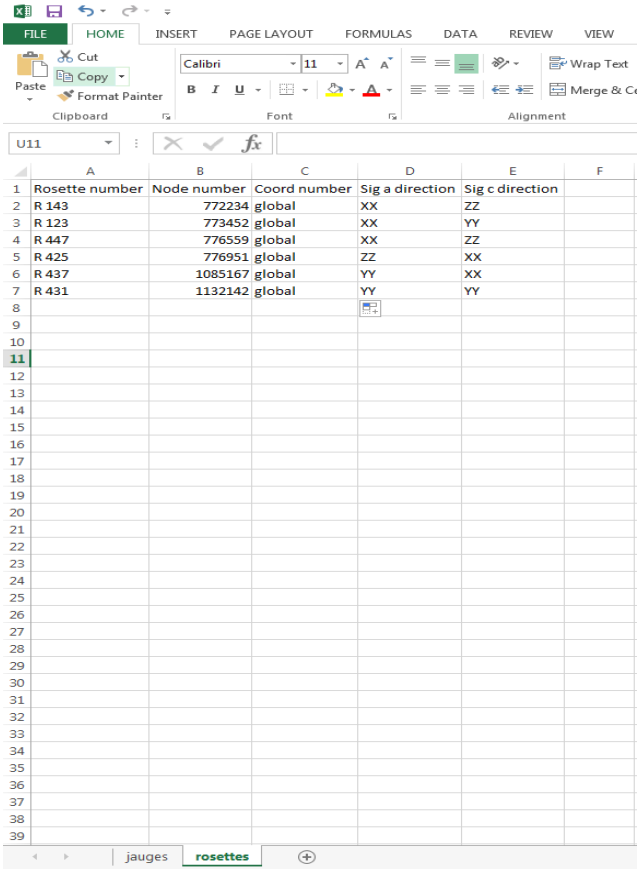
Fig 3.13 Creating Local system

Step 2-Create an input excel (.xlsx) file named input_data with the required nodes and their details as shown in fig. (Note that two separate sheets should be created for gauges and rosettes named jauges and rosettes respectively as shown in fig 3.14 and fig 3.15)



	A	B	C	D
1	Jauge number	Node number	Coord number	Stress direction
2	J 103	744237	global	XX
3	J 3402	745745	global	ZZ
4	J 1202	745754	global	YY
5	J 3401	745762	global	XX
6	J 439	747357	global	XX
7	J 438	752109	global	XX
8	J 154	755254	global	ZZ
9	J 208	755840	global	YY
10	J 109	758558	global	XX
11	J 415	758574	global	XX
12	J 117	758714	global	XX
13	J 115	758724	global	ZZ
14	J 408	758773	global	YY
15	J 234	762001	global	XX
16	J 427	762029	global	XX
17	J 209	762319	global	XX
18	J 460	773269	global	ZZ
19	J 359	773310	global	XX
20	J 1257	773539	global	XX
21	J 250	774150	global	XX
22	J 406	774306	global	XX
23	J 155	774420	global	XX
24	J 150	774564	global	YY
25	J 149	774588	global	XX
26	J 454	774791	global	YY
27	J 455	774933	global	YY
28	J 136	774990	global	XX
29	J 458	776059	global	XX
30	J 110	776688	Coord 487	XX
31	J 308	776771	global	XX
32	J 111	776834	Coord 492	YY
33	J 411	776840	Coord 490	XX
34	J 210	777090	Coord 488	XX
35	J 106	896510	global	XX
36	J 108	896559	global	XX
37	J 1201	992970	global	ZZ
38	J 421	1027143	global	XX
39	J 452	1046574	global	YY

Fig 3.14 input sheet for gauges



	A	B	C	D	E
1	Rosette number	Node number	Coord number	Sig a direction	Sig c direction
2	R 143	772234	global	XX	ZZ
3	R 123	773452	global	XX	YY
4	R 447	776559	global	XX	ZZ
5	R 425	776951	global	ZZ	XX
6	R 437	1085167	global	YY	XX
7	R 431	1132142	global	YY	YY

Fig 3.15 input sheet for rosettes

Step 3 – Run the .exe file named stress_sheets_generator and as the dialog box appears, click on ‘Generate node set text files’ (fig 3.16). All the node set text files will be generated in same folder as per the assigned coordinate system.

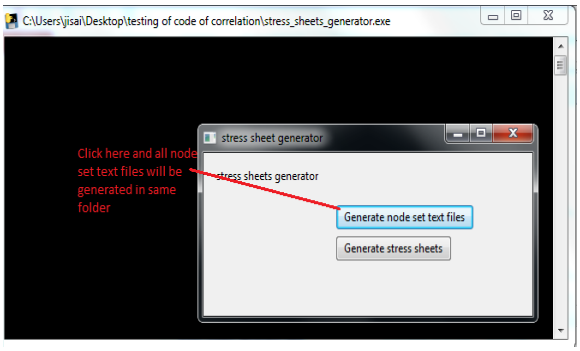


Fig 3.16 GUI

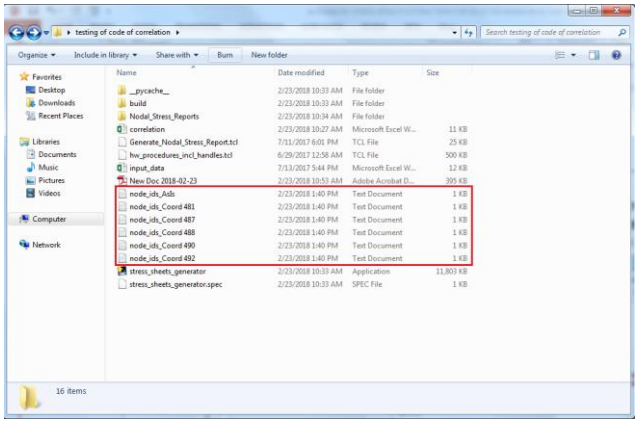


Fig 3.17 Generated Text files

Step 4 - Import all the node sets in the text files (Tools>Import>Sets) as shown in fig. one after the other.

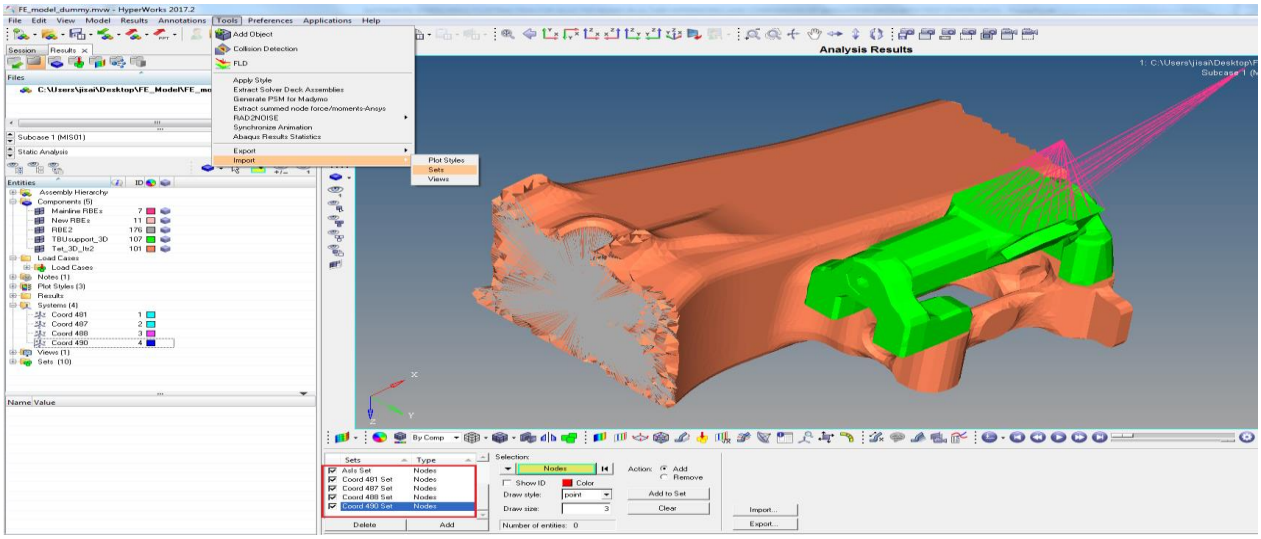


Fig 3.18 Importing node set text files

Step 5 -Run the TCL script named Generate_Nodal_Stress_Report (File>Run>TCL Script) as shown in fig 3.19.

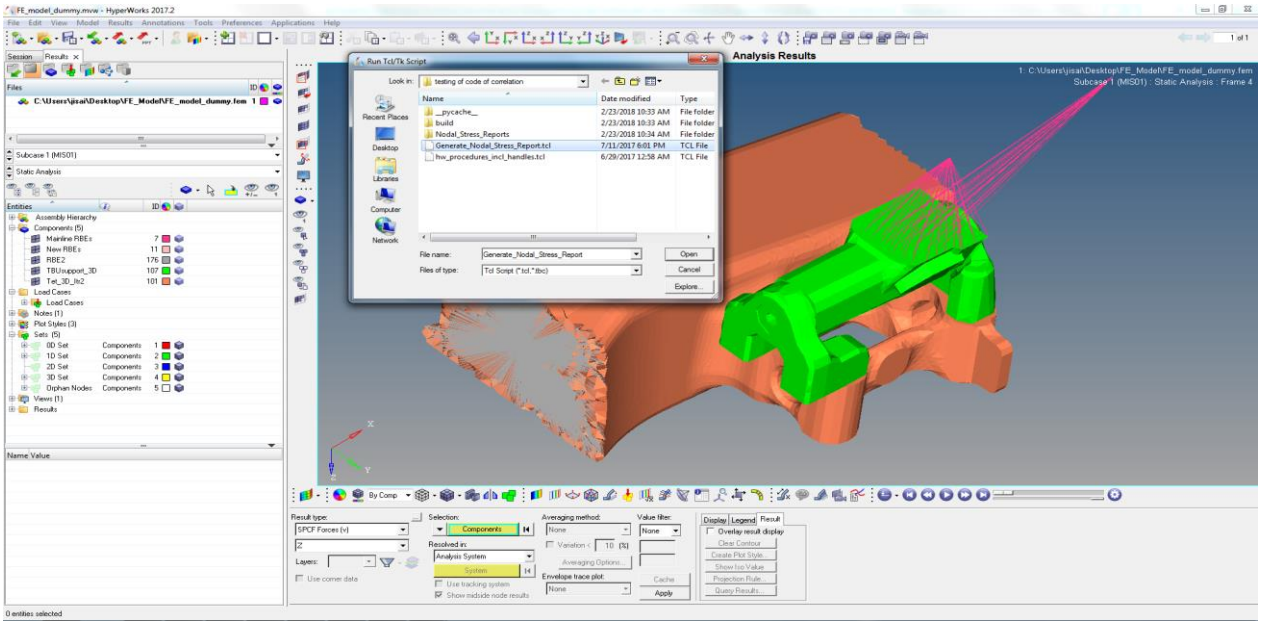


Fig 3.19 Running nodal stress generator tool

Step 6 – As shown in the figure, when tcl file is selected, the dialog box will appear, select the corresponding coordinate system, in which the results are to be exported. Then click on Generate stress report.

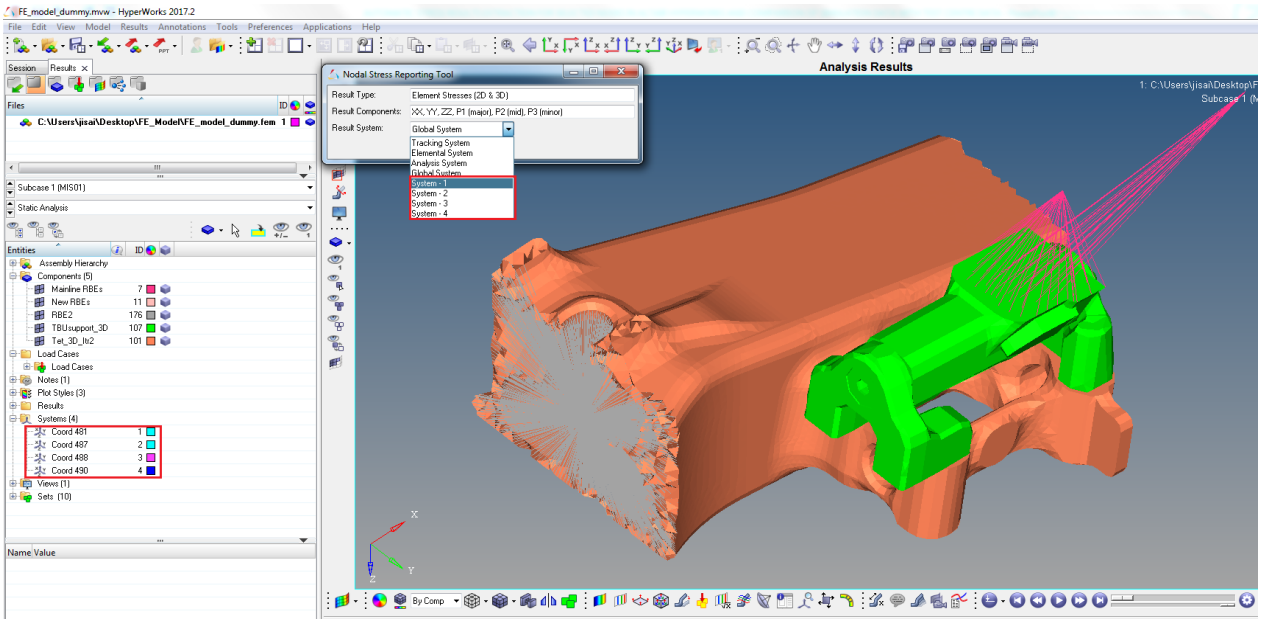


Fig 3.20 Selection of coordinate system

Step 7 - Click on “Nodes” button, then click on “By Set” button. (as shown in fig 3.21)

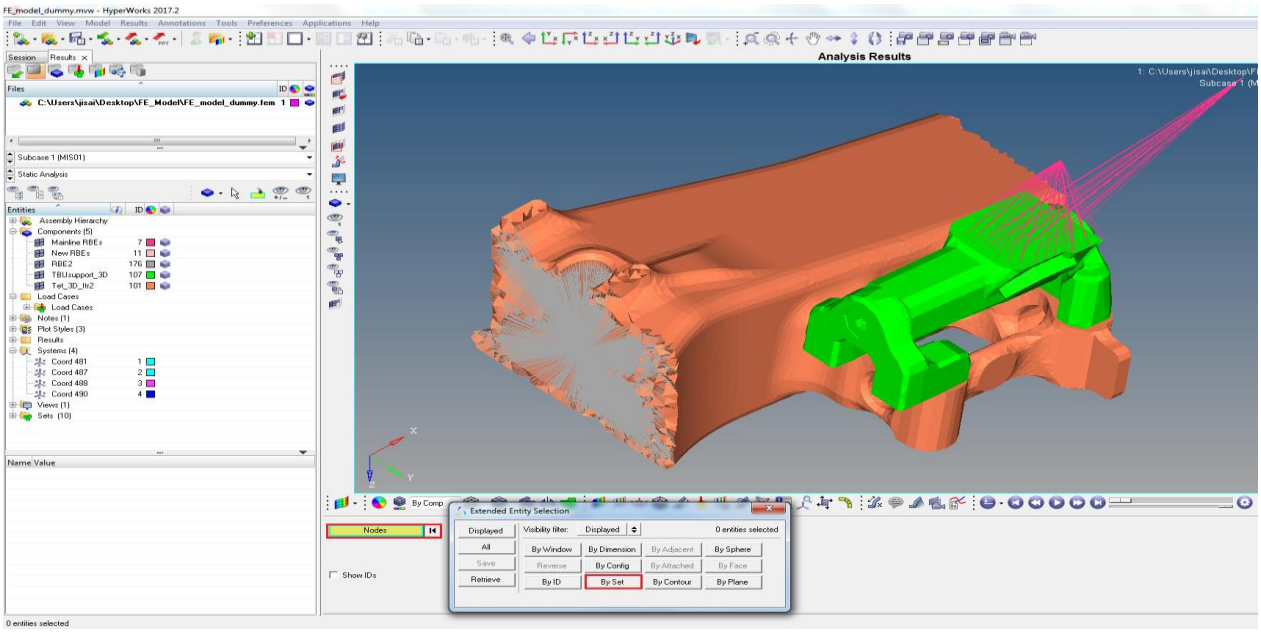


Fig 3.21 Node group selection

Step 8 - Select the node set (global or previously defined), click “Add”, click “Return” and click “proceed” as shown in fig 3.22.

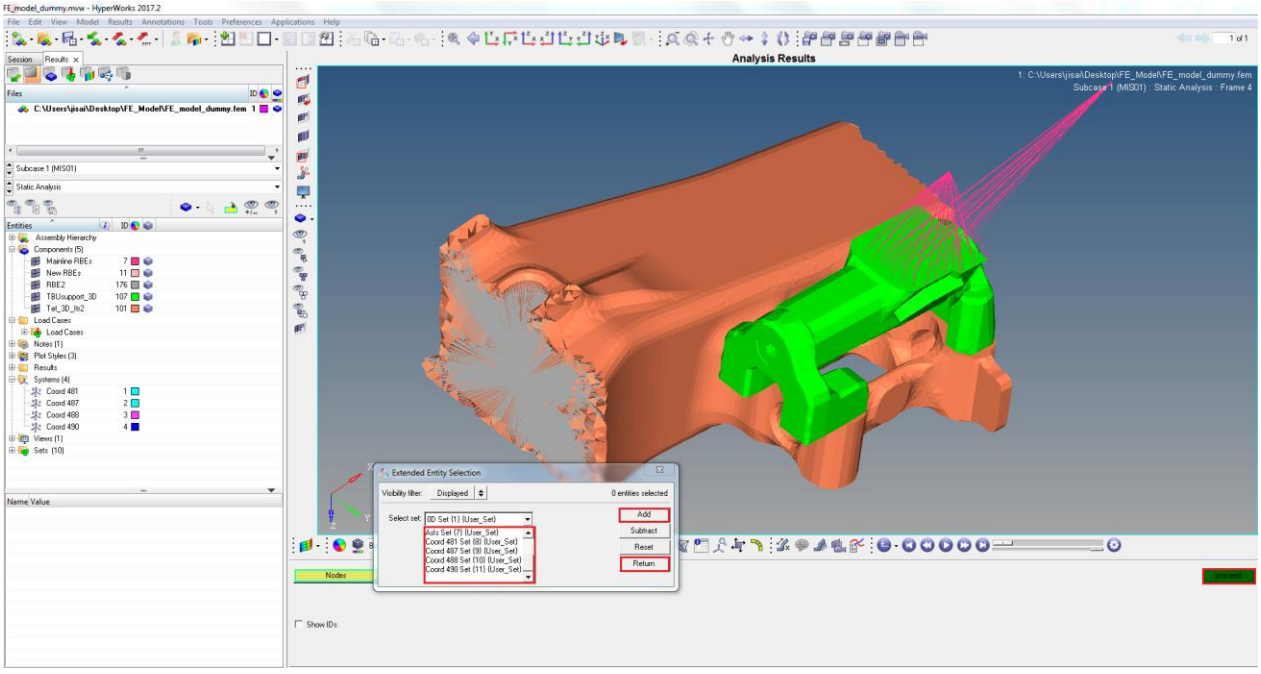


Fig 3.22 Selection of required node group

Step 9 – Create folder named ‘Nodal_Stress_Reports’ and copy all the generated .csv files to that folder before extracting the results for another coordinate system.

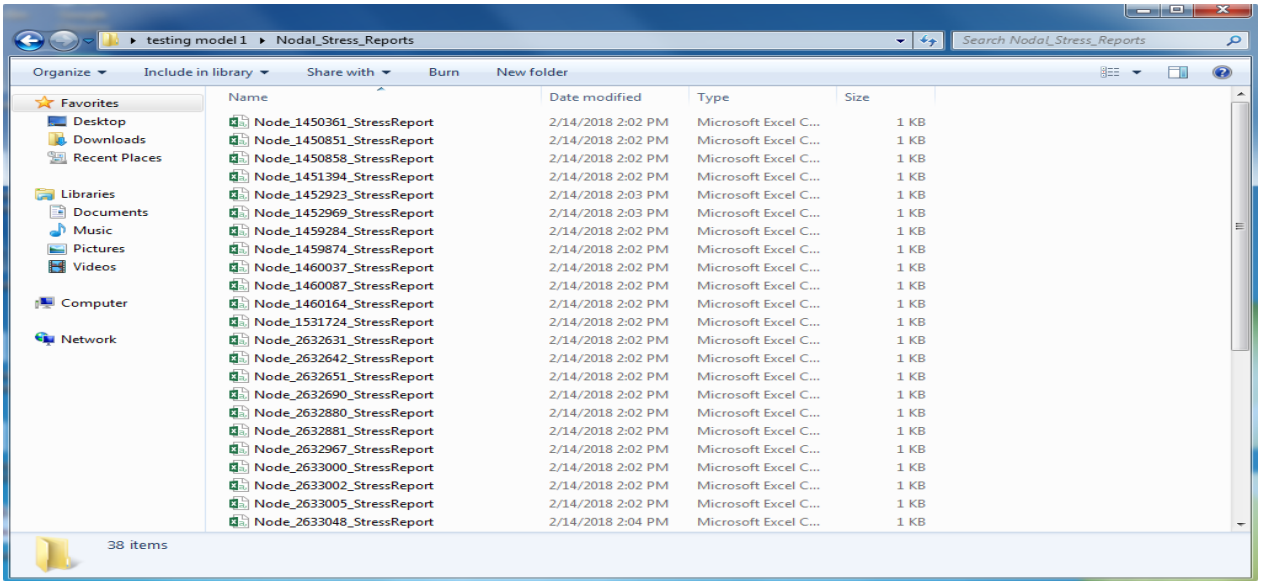


Fig 3.23 Generated nodal stress reports

Step 10- Arrange the data from the test centre in excel (.xlsx) file named correlation in the following format as shown in the fig 3.24.

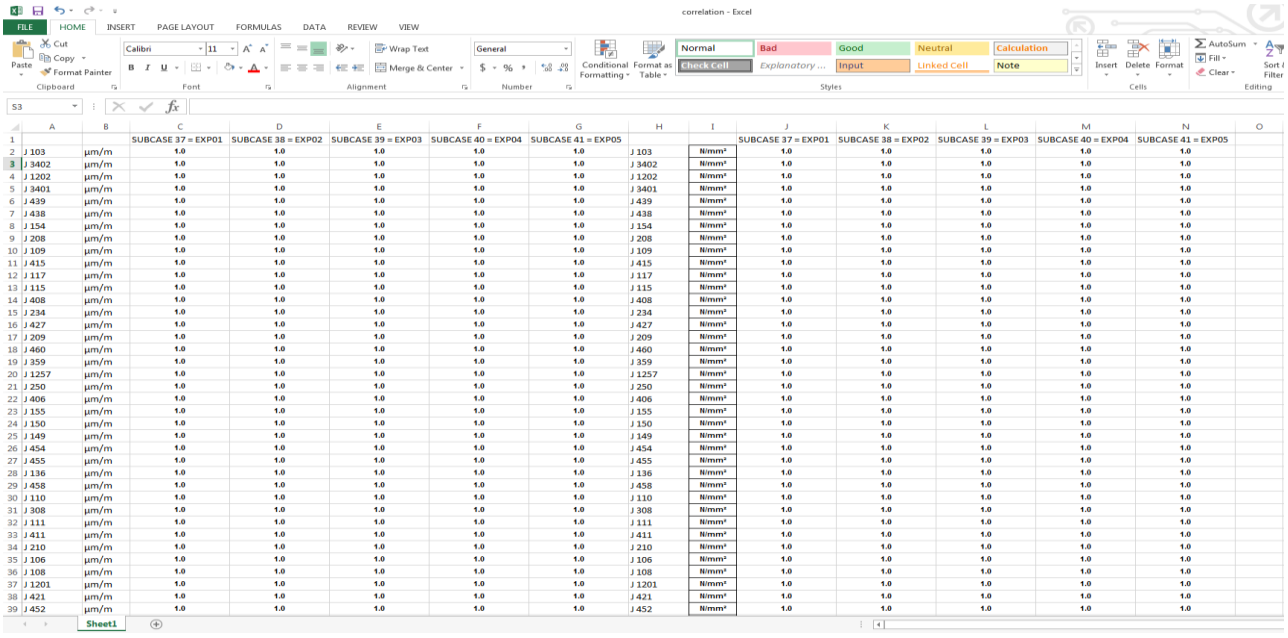


Fig 3.24 Preparation of correlation excel sheet

Step 11- Run the Stress_sheets_generator.exe file again. Now, click on ‘Generate Stress sheets’.

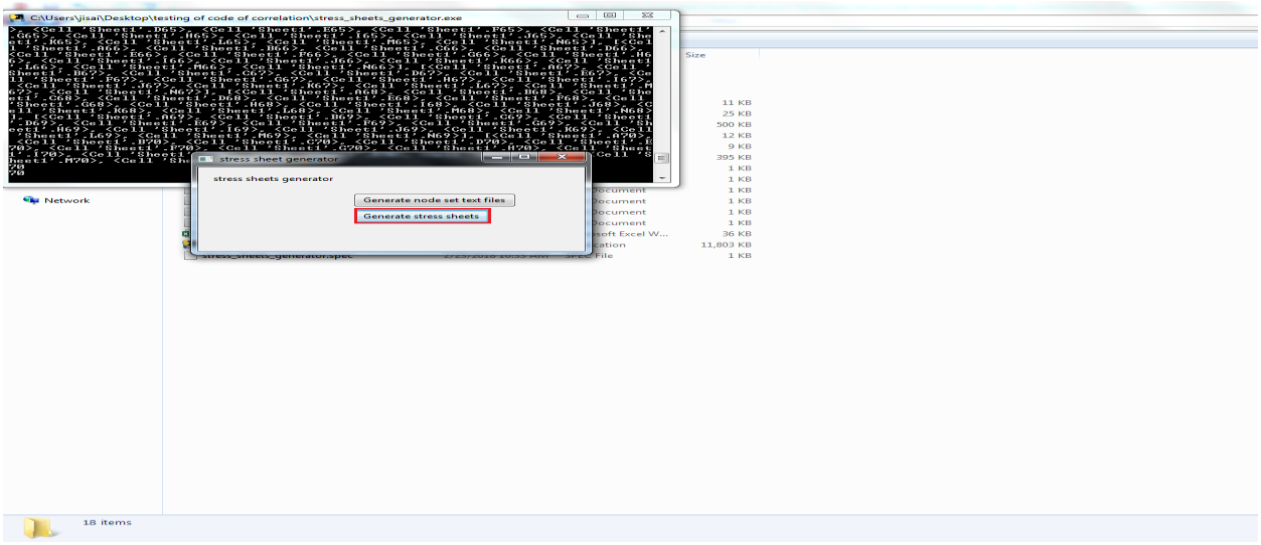


Fig 3.25 Execution of code

Error verification

- After, clicking on generate stress sheets button, “STRESS SHEETS ARE GENERATED” is displayed in the .exe window is program is successfully completed. If there is some error in input file, it will display ‘STRESS SHEETS ARE NOT GENERATED’. Both cases are displayed in figures below.

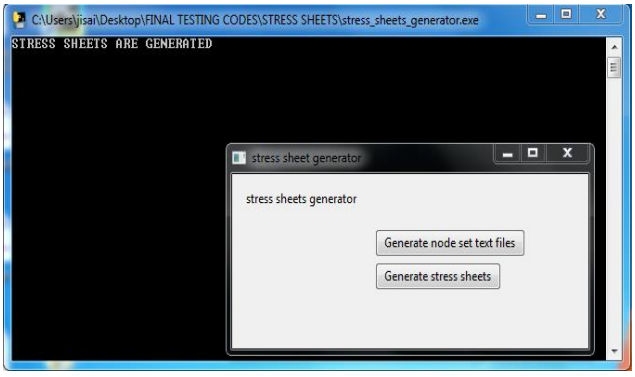


Fig 3.26 Case of Error

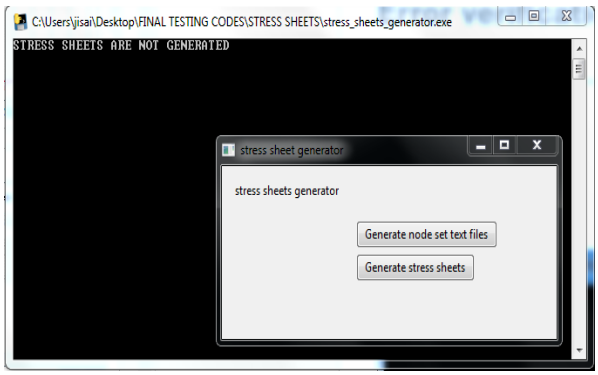


Fig 3.27 Successful execution

- Two excel files named stress_sheets_final and Modified correlation are obtained as the output files which contains all the required results and comparisons with the correlation data and calculated values of sigma stresses.

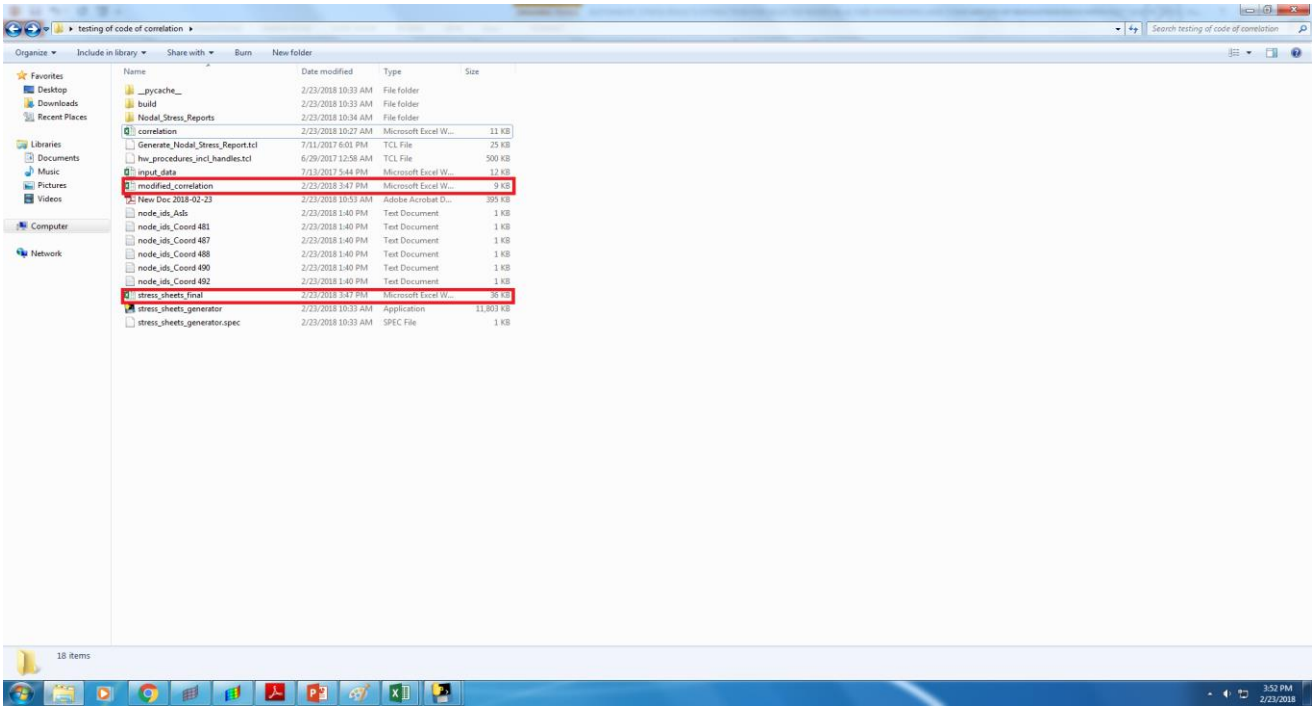


Fig 3.28 Generated and modified excel files

3.3.4 Description of Output files

a. Stress_sheets_final

- The first worksheet named ‘all_stresses’ is generated containing stress values for all directions for all loadcases for each gauge and rosette.

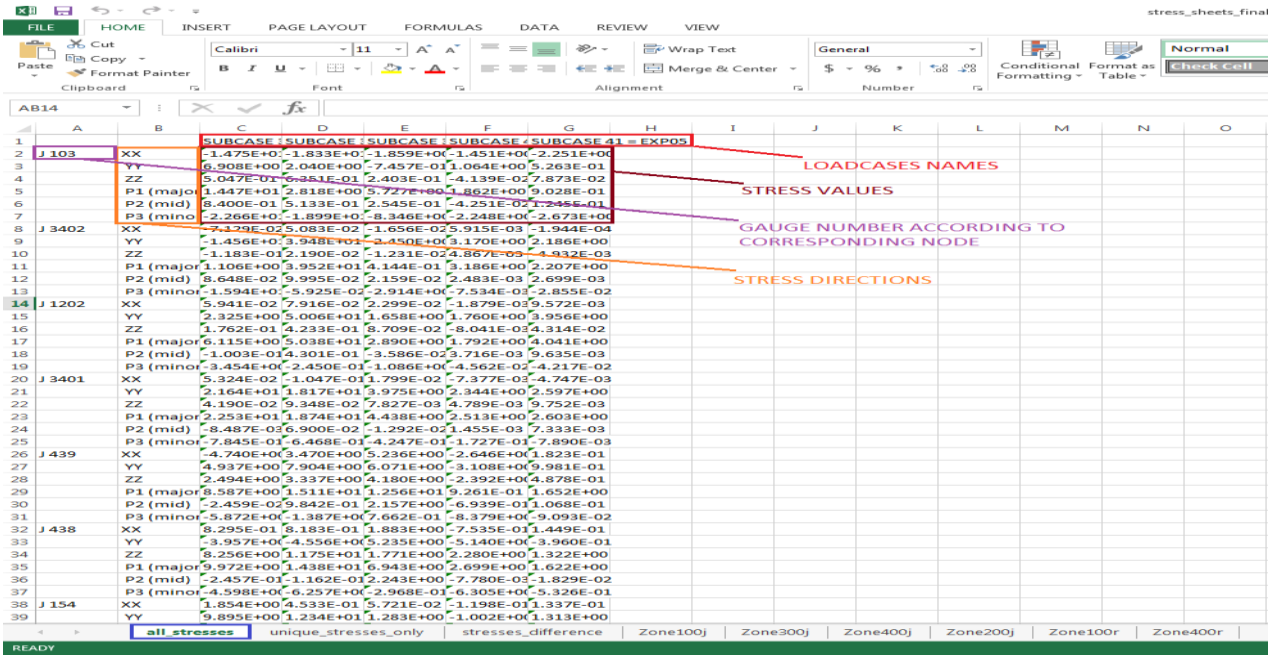


Fig 3.29 'all_stresses' worksheet

- The second worksheet named 'unique_stresses_only' is generated containing stress values for directions that are given as input in input_data.xlsx file for all loadcases for each gauge and rosette.

+

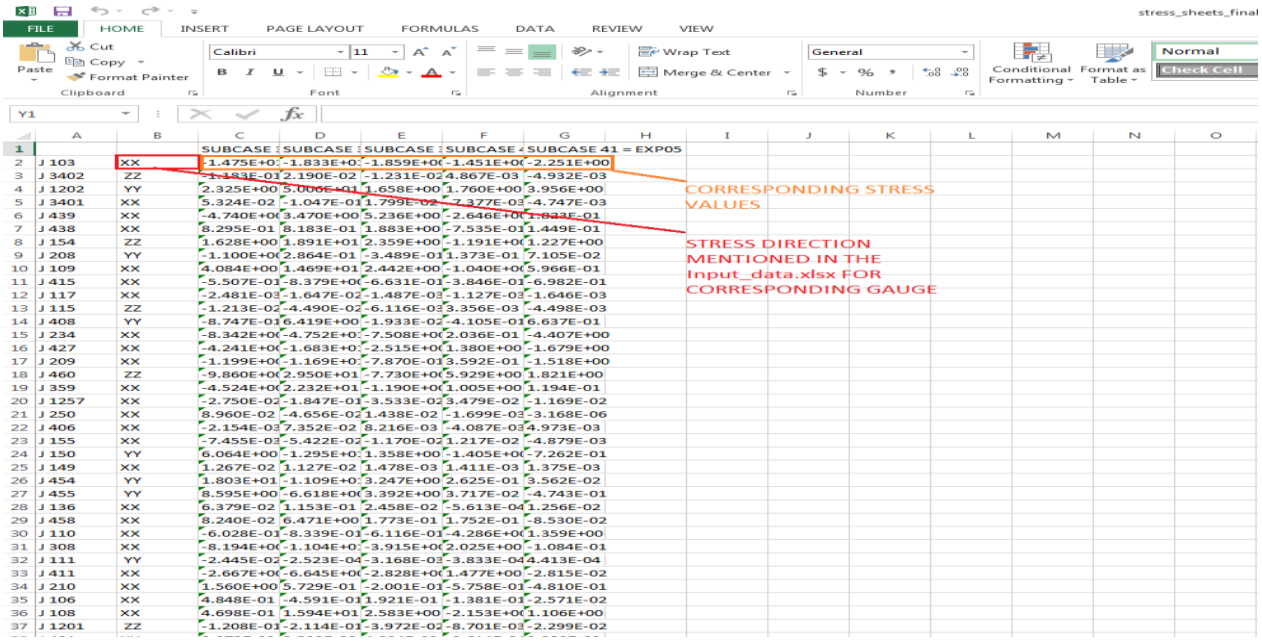


Fig 3.30 'unique_stresses_only' worksheet

- The third worksheet named ‘stresses_difference’ is generated containing the difference between test centre data and Fem data for all loadcases for each gauge and rosette.

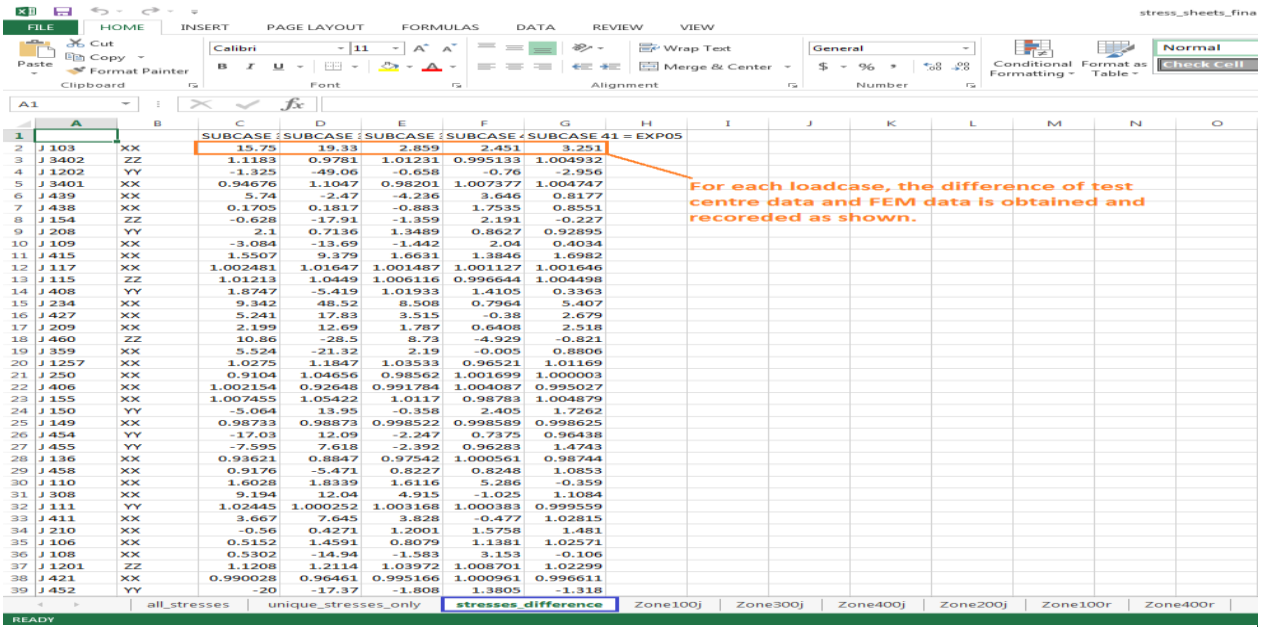
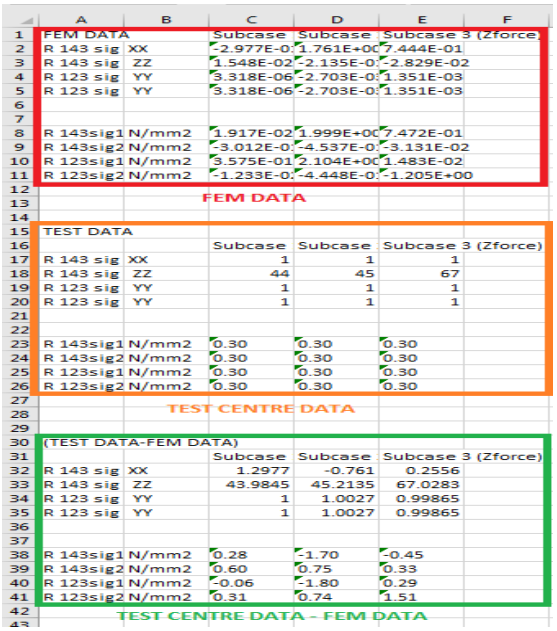


Fig 3.31 ‘stresses_difference’ worksheet

- For each zone separate sheet will be generated, containing data of FEM, TEST CENTRE and their DIFFERENCE.



	A	B	C	D	E	F
1	FEM DATA		Subcase	Subcase	Subcase 3 (Zforce)	
2	R 143 sig	XX	-2.977E-01	1.761E+00	7.444E-01	
3	R 143 sig	ZZ	1.548E-02	-2.135E-01	-2.829E-02	
4	R 123 sig	YY	3.318E-06	-2.703E-01	1.351E-03	
5	R 123 sig	YY	3.318E-06	-2.703E-01	1.351E-03	
6						
7						
8	R 143sig1	N/mm2	1.917E-02	1.999E+00	7.472E-01	
9	R 143sig2	N/mm2	-3.012E-01	-4.537E-01	-3.131E-02	
10	R 123sig1	N/mm2	3.575E-01	2.104E+00	1.483E-02	
11	R 123sig2	N/mm2	-1.233E-01	-4.448E-01	-1.205E+00	
12						
13			FEM DATA			
14						
15	TEST DATA					
16			Subcase	Subcase	Subcase 3 (Zforce)	
17	R 143 sig	XX	1	1	1	
18	R 143 sig	ZZ	44	45	67	
19	R 123 sig	YY	1	1	1	
20	R 123 sig	YY	1	1	1	
21						
22						
23	R 143sig1	N/mm2	0.30	0.30	0.30	
24	R 143sig2	N/mm2	0.30	0.30	0.30	
25	R 123sig1	N/mm2	0.30	0.30	0.30	
26	R 123sig2	N/mm2	0.30	0.30	0.30	
27						
28			TEST CENTRE DATA			
29						
30	(TEST DATA-FEM DATA)					
31			Subcase	Subcase	Subcase 3 (Zforce)	
32	R 143 sig	XX	1.2977	-0.761	0.2556	
33	R 143 sig	ZZ	43.9845	45.2135	67.0283	
34	R 123 sig	YY	1	1.0027	0.99865	
35	R 123 sig	YY	1	1.0027	0.99865	
36						
37						
38	R 143sig1	N/mm2	0.28	-1.70	-0.45	
39	R 143sig2	N/mm2	0.60	0.75	0.33	
40	R 123sig1	N/mm2	-0.06	-1.80	0.29	
41	R 123sig2	N/mm2	0.31	0.74	1.51	
42						
43			TEST CENTRE DATA - FEM DATA			

Fig 3.32 Zone wise sheet with correlation data

b. Modified_correlation

- From test centre data, corresponding stresses are determined, from that values the tool will calculate the value of sig1, sig2 and grad values for each rosette.

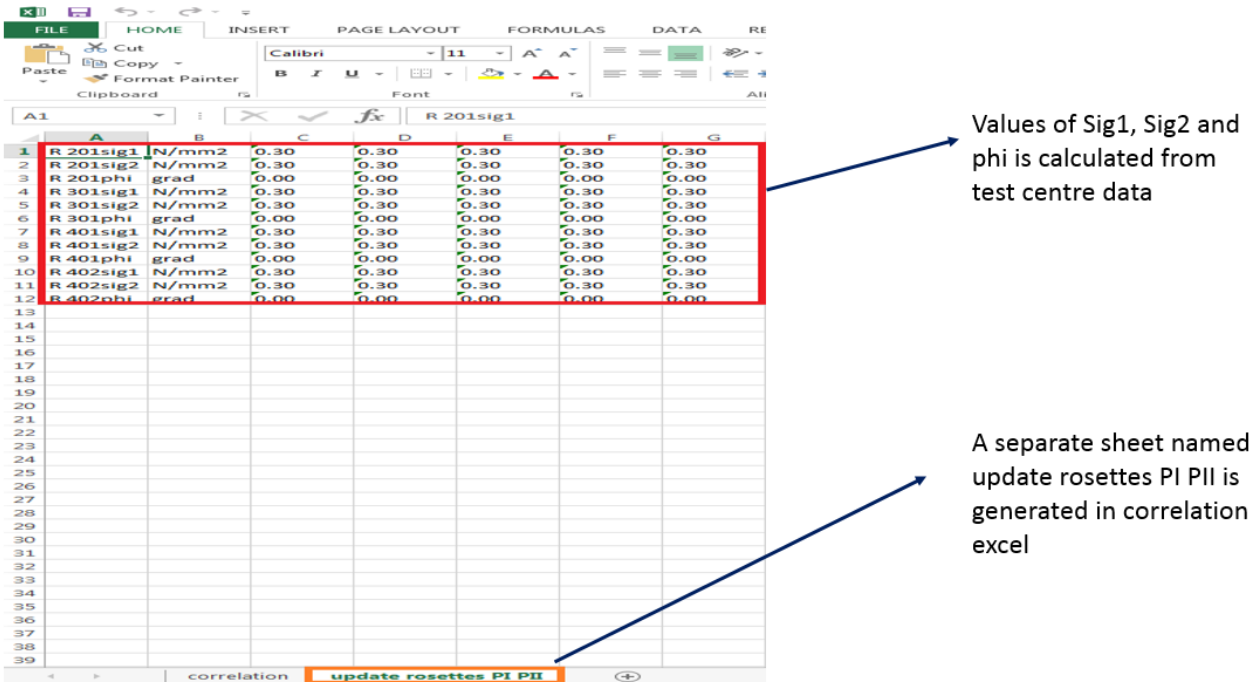


Fig 3.33 modified correlation excel sheet

CHAPTER 4

Accelerated Testing

4.1 Time-series treatment

Time-series treatment has to be done for two reasons, to remove the running mean and spike removal. The running mean is generally introduced in the time series signal due to alleviated environmental conditions and improper installation. Spikes are introduced due to sensor or data acquisition errors.

Process flow for signal treatment:

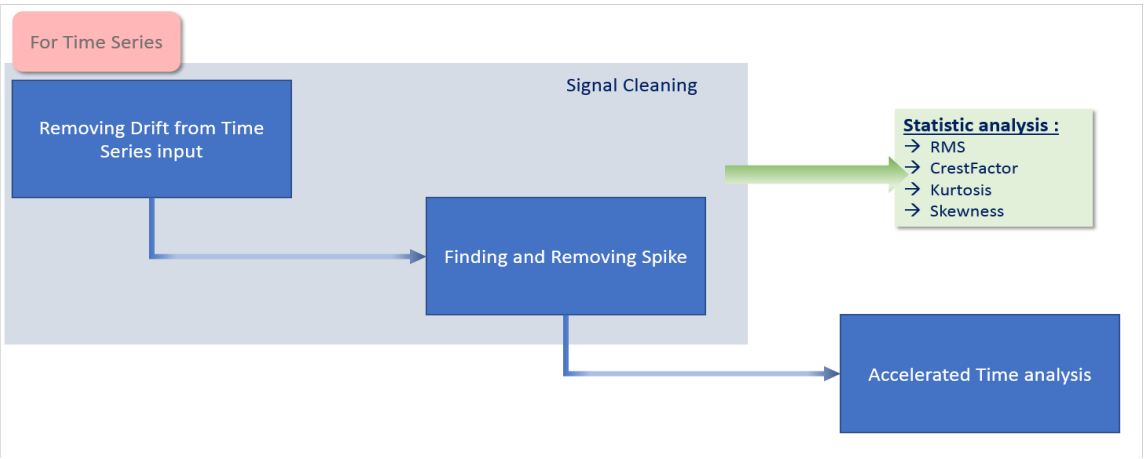


Fig 4.1 signal treatment flow

Layout in N-code:

In N-code, time series input is given. Using Butterworth filter, running mean in the data is removed and then it is fed into spike detection. In the spike detection if the amplitude is more than 50 percent within 2Hz frequency band, then it will consider it as spike and will remove it. After spike detection, data is fed into statistics glyph and to SRS glyph. Statistics glyph will give information about skewness, kurtosis, RMS, etc. SRS glyph will give two outputs that are Fatigue damage spectrum for the given time series data and shock response spectrum which gives information about the maximum acceleration response for given time series data.

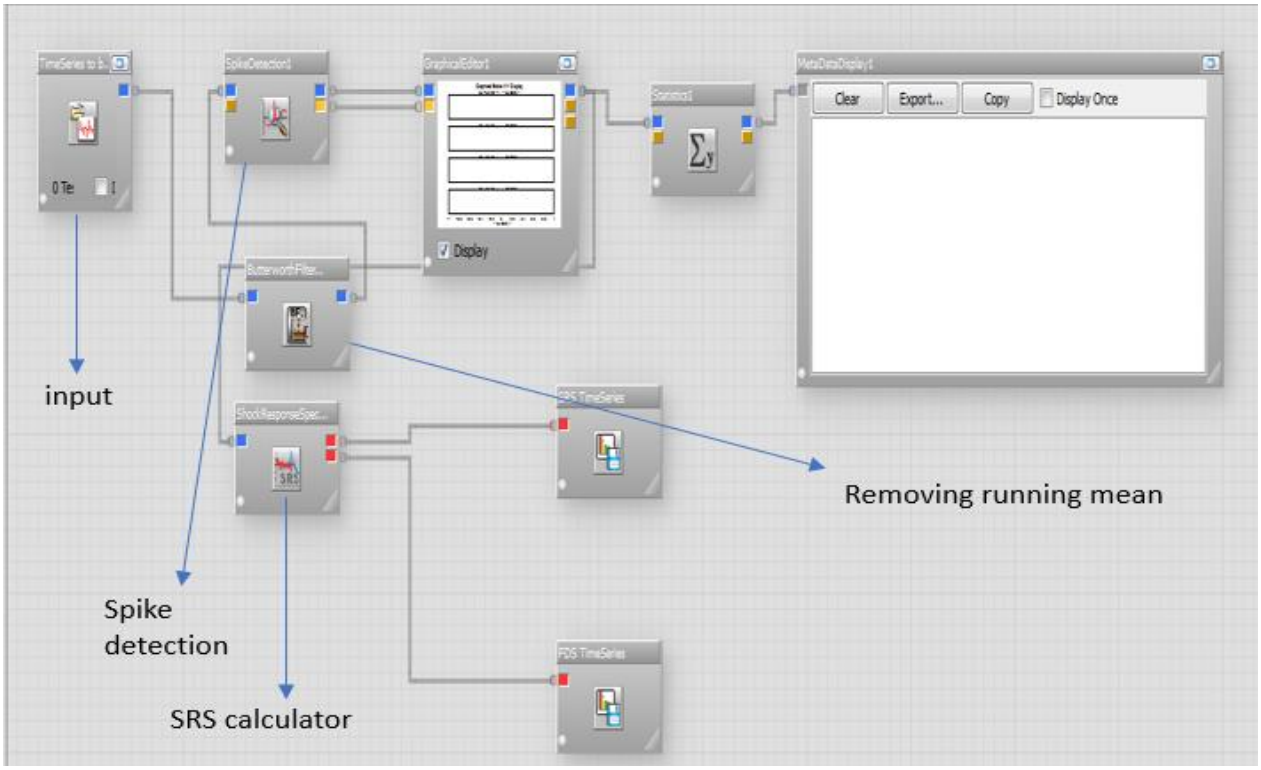


Fig 4.2 N-code layout for signal treatment

4.1.1 Response spectrum generations

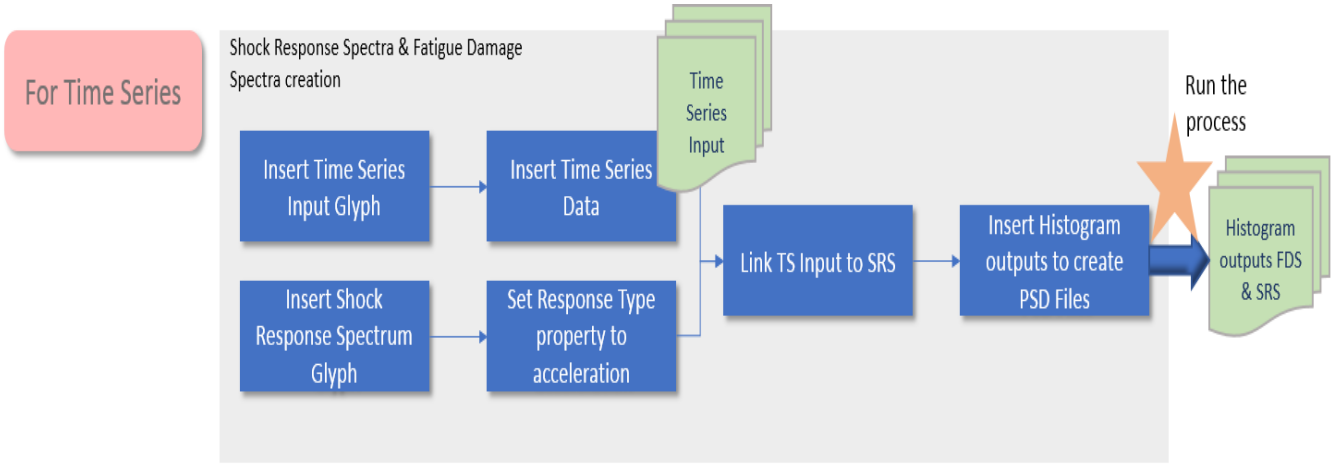


Fig 4.3 Generating response spectrums using Time series input

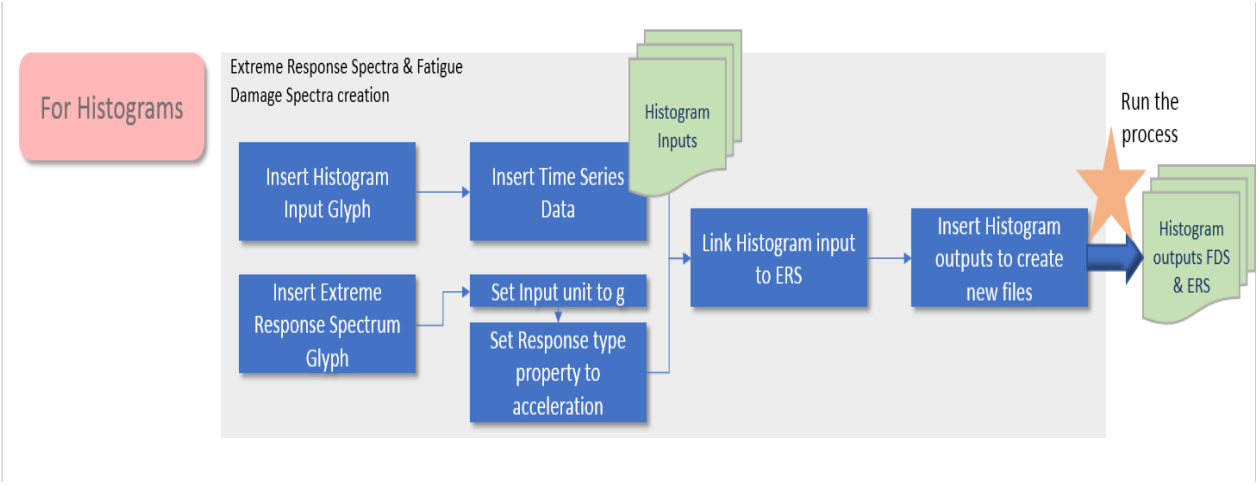


Fig 4.4 Generating response spectrums using Histograms input

4.2 Developing mission profile

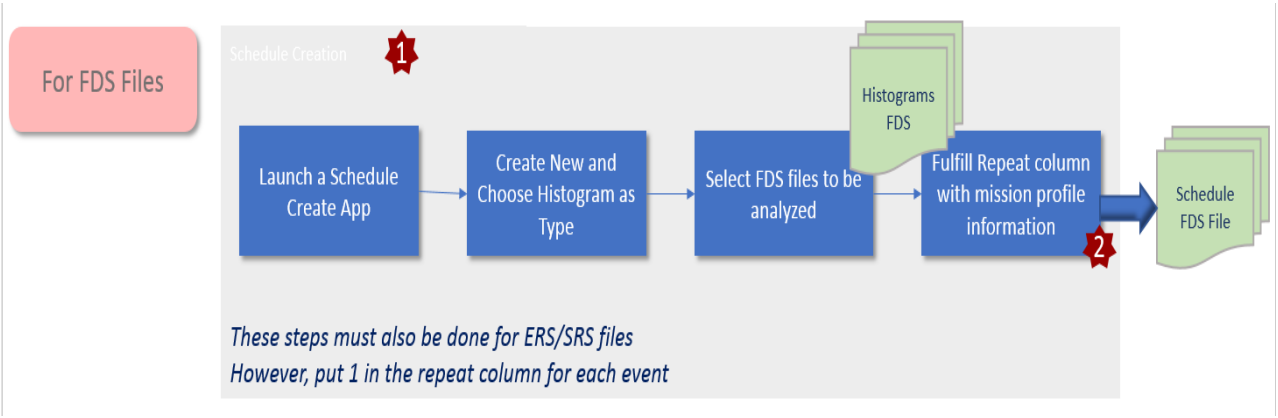


Fig 4.5 Creating schedule

The fatigue damage spectrum, that is obtained earlier is for the recorded time series data only, to estimate the total fatigue damage for whole life, the FDS must be repeated to the number of cycles and then the final response spectrum is obtained which is applied at the vibration table. For estimating the life time fatigue damage and deriving response spectrum from it, a schedule has to be created in N-code and number of repeats has to mentioned in it. Afterwards, that schedule file is used is estimate the response spectrum for whole life and is also used for deriving final accelerated test spectrum.

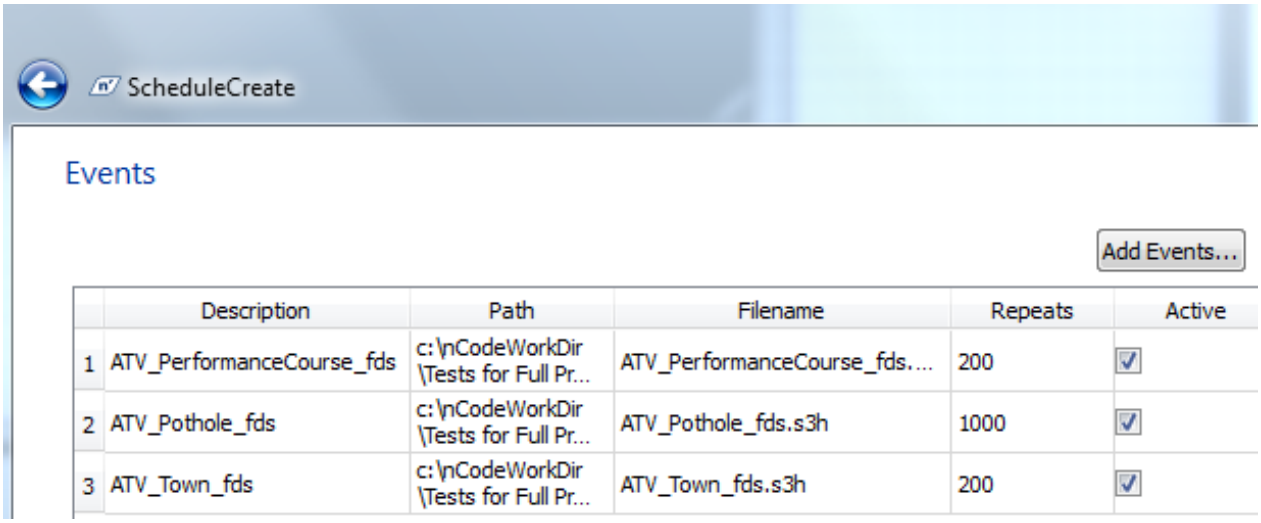


Fig 4.6 Example of mission profile definition

- It is mandatory to create a Schedule. A result can't be used if it has only been ran once. This step simulates the use of a component a certain amount of times (1000 repeats for example)
- The data in this column will be use for the following steps in the Duration Hours parameter

4.3 Accelerated testing layout in N-code

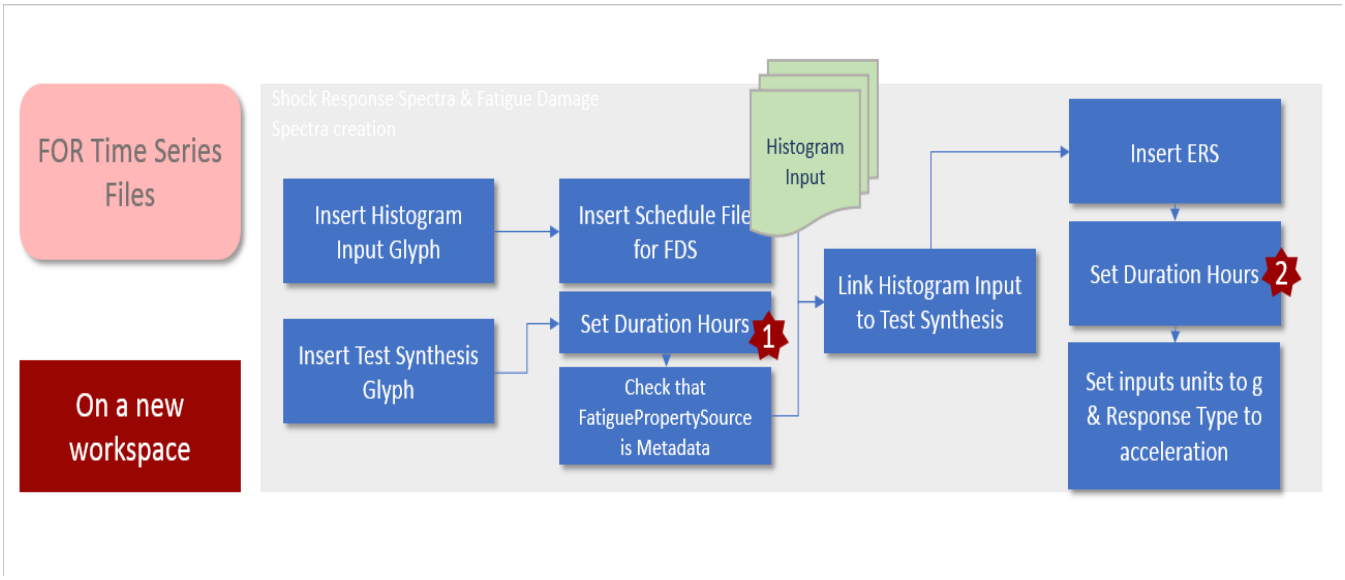


Fig 4.7 process flow for accelerated testing

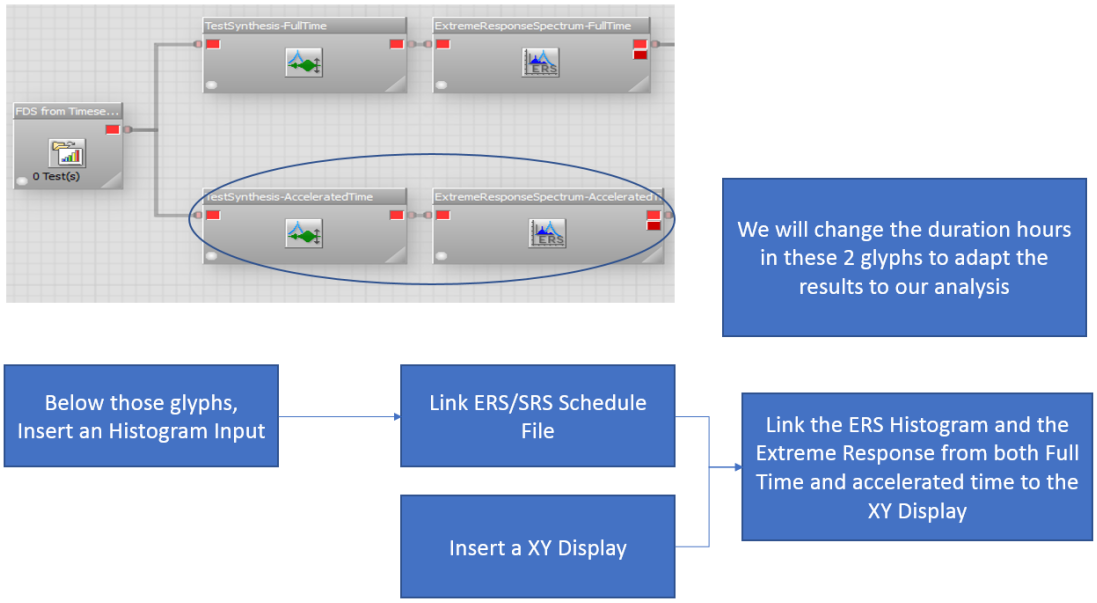


Fig 4.8 process flow for display

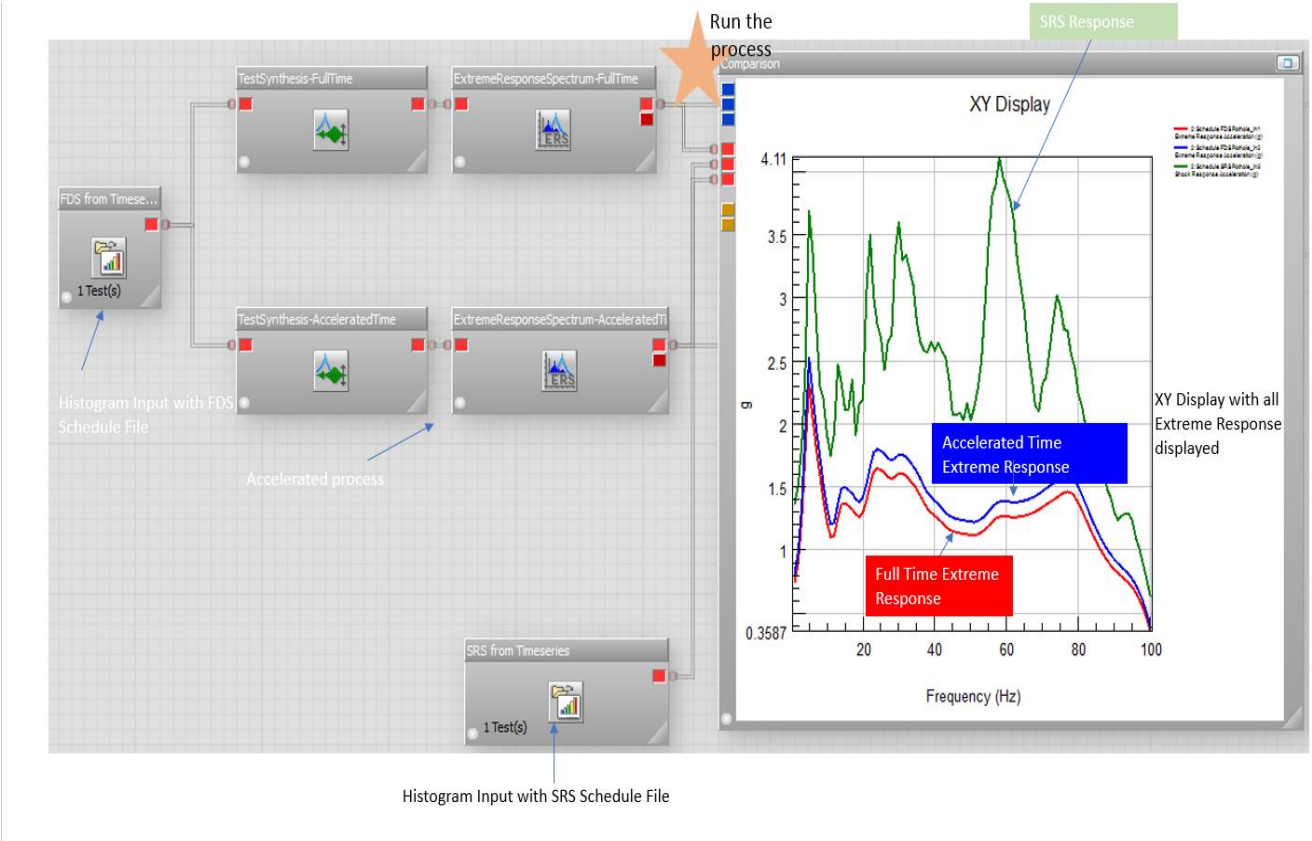


Fig 4.9 Final layout for accelerated testing in N-code

- For generation of extreme response spectrum for accelerated testing, first of all, histogram input is taken from the scheduled FDS which was generated earlier.
- For, full life time extreme response spectrum, test synthesis glyph is used for generating power spectral density, which will give same amount of damage as in FDS for the testing time duration of full life (time for which product has to be operated).
- After that, for that PSD, using ERS glyph, response spectrum is obtained for full life time.
- For accelerated testing, ERS also same procedure is followed but the testing time is decreased, so PSD and ERS magnitude increases accordingly, this is compared with SRS that was generated earlier using time series data, which gives us information about maximum acceleration level that component has sustained while collecting data and give us reference for our accelerated time duration.
- Each tome comparison is done between the accelerated ERS and SRS and when optimized ERS is obtained which has decreased the testing time significantly and also is not above the maximum levels which can be observed from SRS, the spectrum is given as input to the vibration tester and the component is tested with reduced testing time.

4.4 Test Runs for accelerated testing

4.4.1 Accelerated test for ATV Brake Mounting

All-terrain vehicle experiences large amount of random loads due to the uncertainty of terrain. For collection of terrain data, three-axis accelerometers are used. The accelerometer is mounted as near as possible to brake mounting bracket, to get the proper estimate of vibrations. Data is collected for over some pre-defined terrains for limited time only.

Steps for generating accelerated testing are as follows-

- First of all, FDS and SRS are obtained from the time series data. The data is collected for three directions as shown in the fig 4.10. The generated FDS and SRS histograms as stored in the current directory for further process.

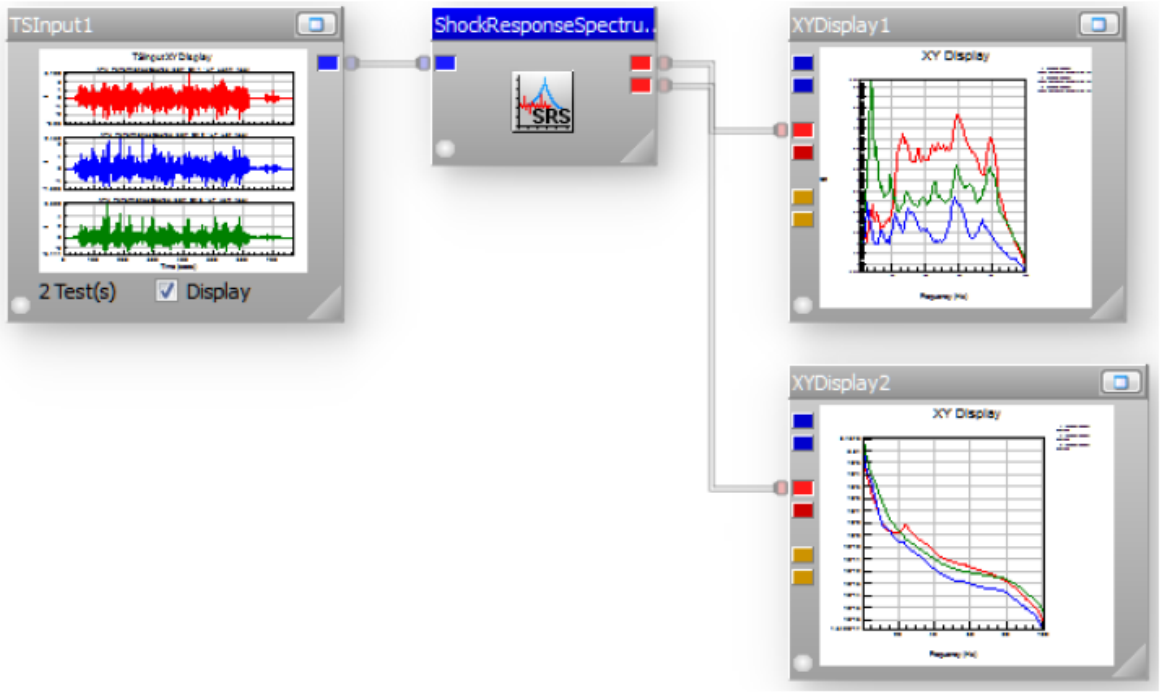


Fig 4.10 Generating FDS and SRS

- Next step, is to define mission profile and creating the schedule according to it. So, in schedule FDS of all three-time series data is added along with their number of repeats which will define the total damage sustained by the component during its define full time life.

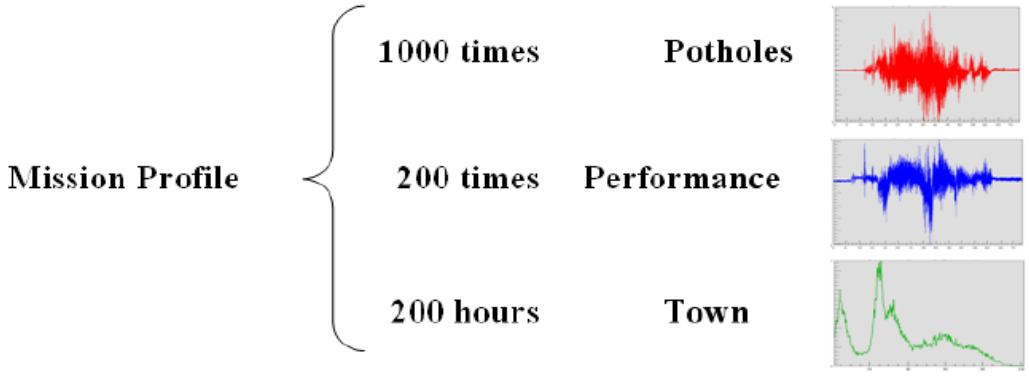


Fig 4.11 Mission profile for brake bracket

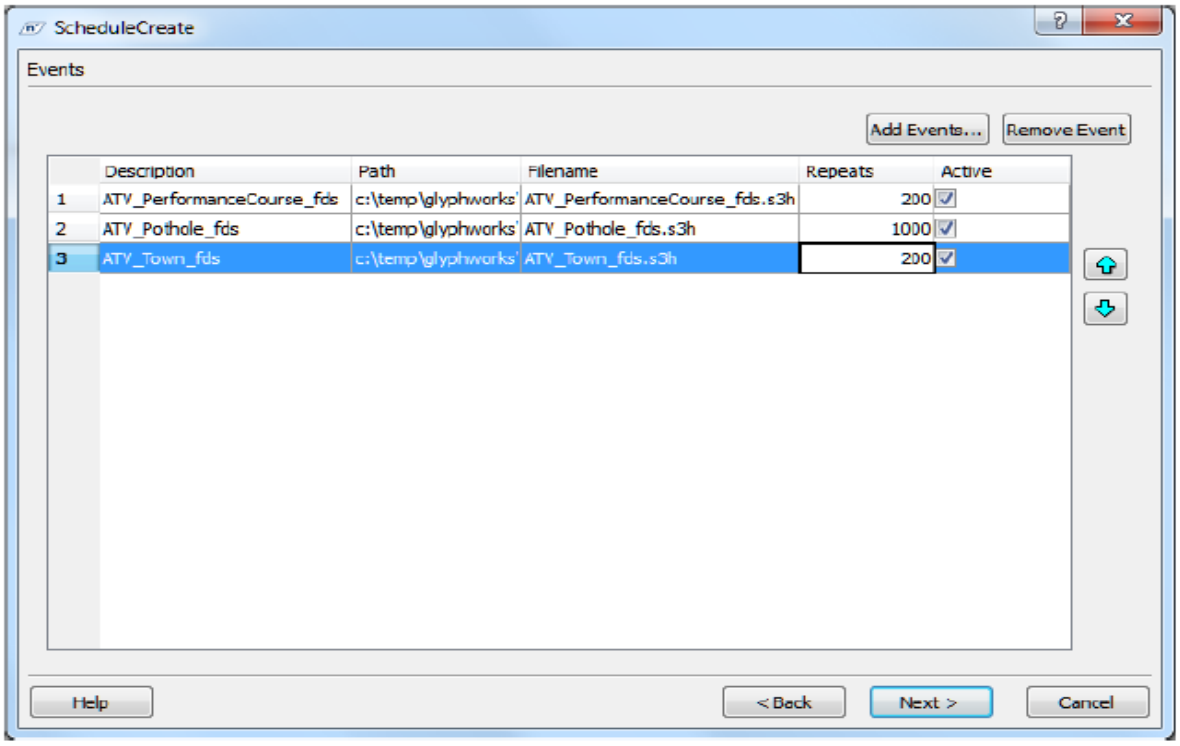


Fig 4.12 Schedule generation according to mission profile

- Now, with the use of accelerated test layout the full life time of 225 hrs was reduced to 150 hrs and comparison was made with SRS and as shown in fig 4.13, there are still chances of accelerating test as still it is quite below the SRS level.

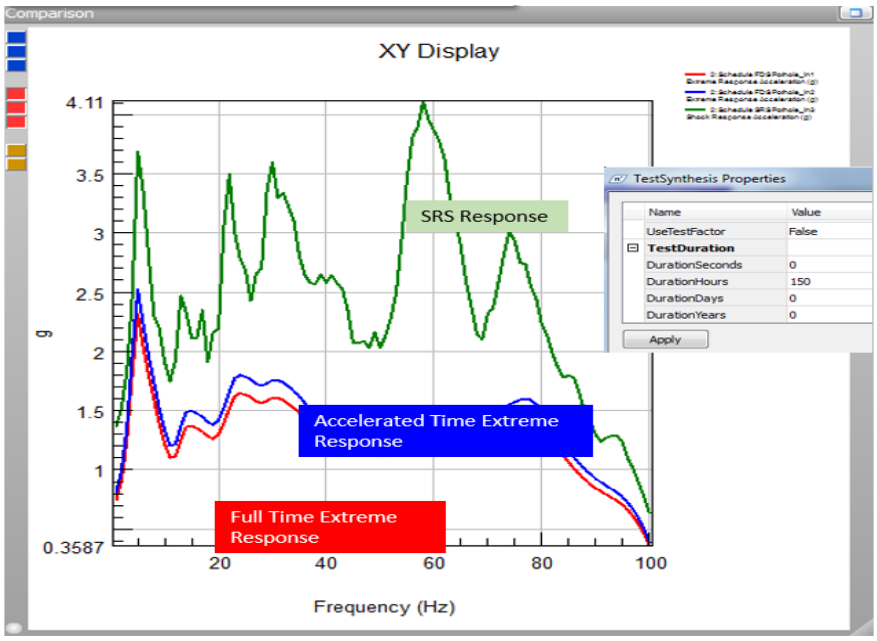


Fig 4.13 Reducing to 150 hrs

- The testing time for acceleration is decreased to 1 hr and from the fig 4.14, we can observe that accelerated testing ERS is above SRS. Hence, there are chances for physical failure of the component, so we have to increase testing time.

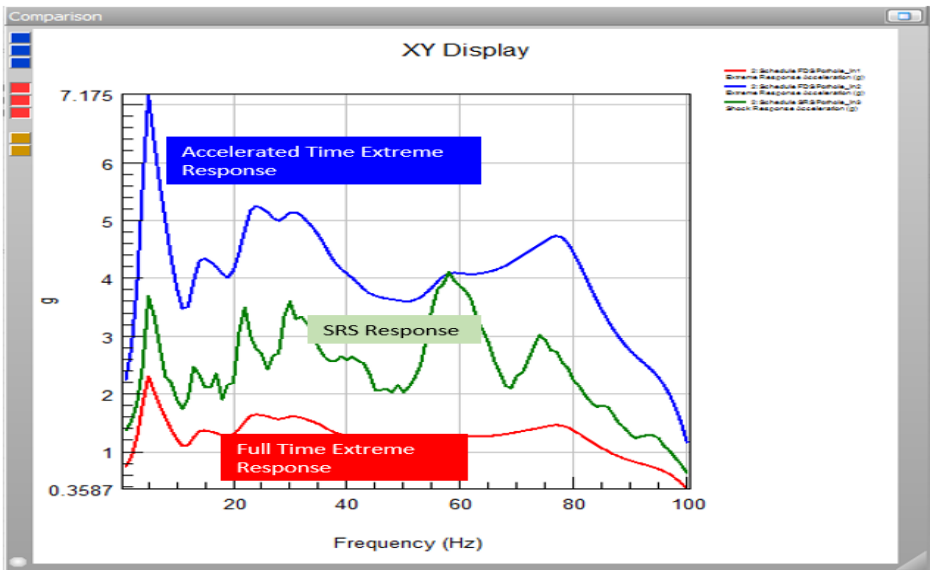


Fig 4.14 Reducing to 1 hr

- The testing time is now set to 20 hrs, from fig 4.15, it can be observed that the accelerated testing ERS and SRS are almost comparable. So, it is the optimized time for testing, less than 20 hrs will generate higher level vibration that actual, while more than 20 hrs will increase testing time and cost.

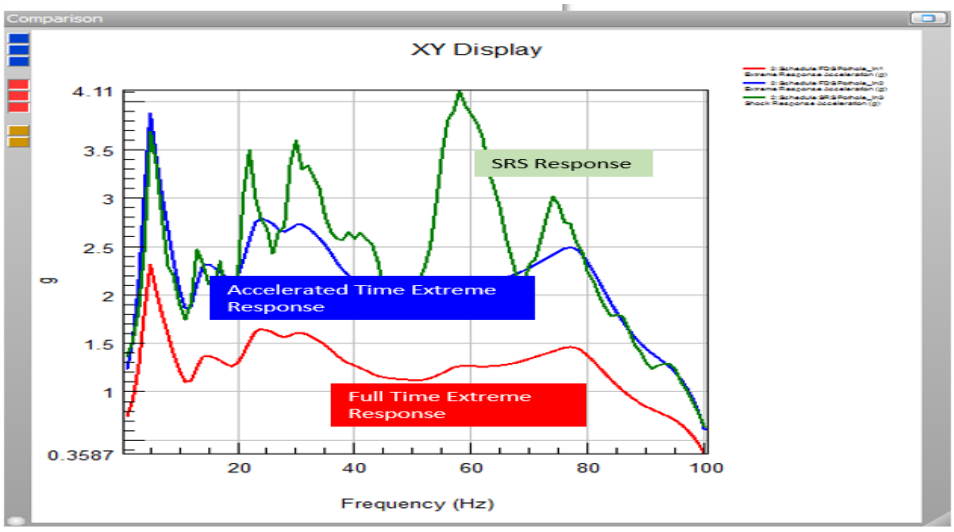


Fig 4.15 Reducing to 20 hrs

4.4.2 Accelerated test for bogie brake mounting

The data is collected between two stations which are 36 km apart, the mean speed of the train was around 50 km/hr and the time duration was 2560 seconds. The design life is 40 years with 1,60,000 km per year.

Acquired time signal

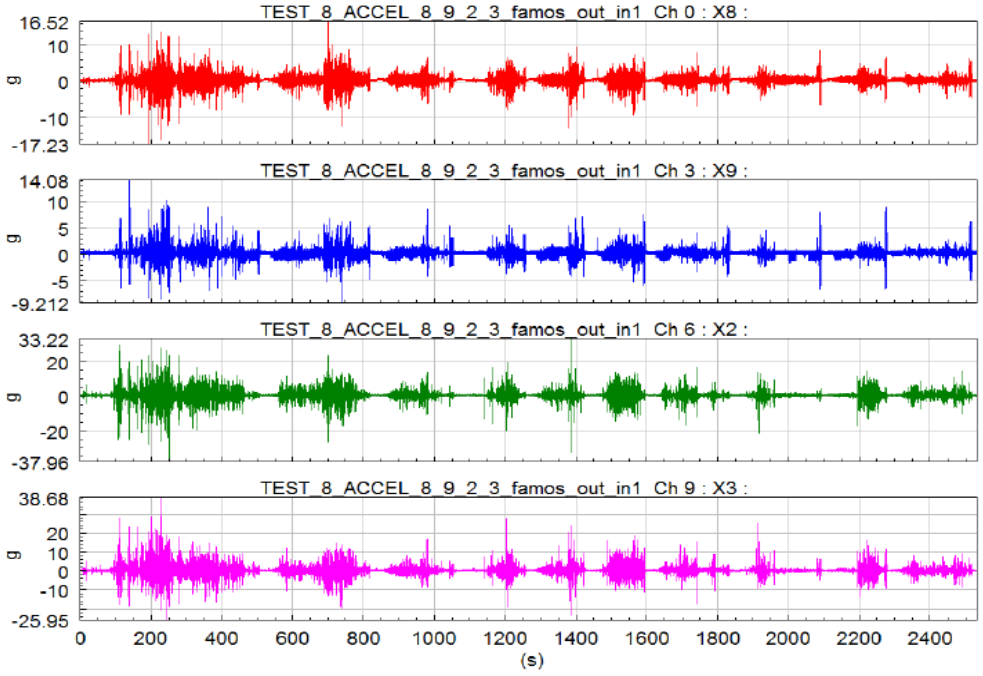


Fig 4.16 Time series data acquired

- Using time series data, FDS and SRS is calculated for the given time signal and it is saved for further processing
- As, the data is for 2560 seconds, for the defined life it has to be repeated for 173000 times, this is used to create the schedule file for total fatigue damage.
- The same layout for accelerated is used that was used for earlier example.
- The full life time is 173000 hours and for first iteration the accelerated test time is set to 100 hours and comparison was made between ERS and SRS. As ERS level was quite large than SRS, the test time has to be decreased.

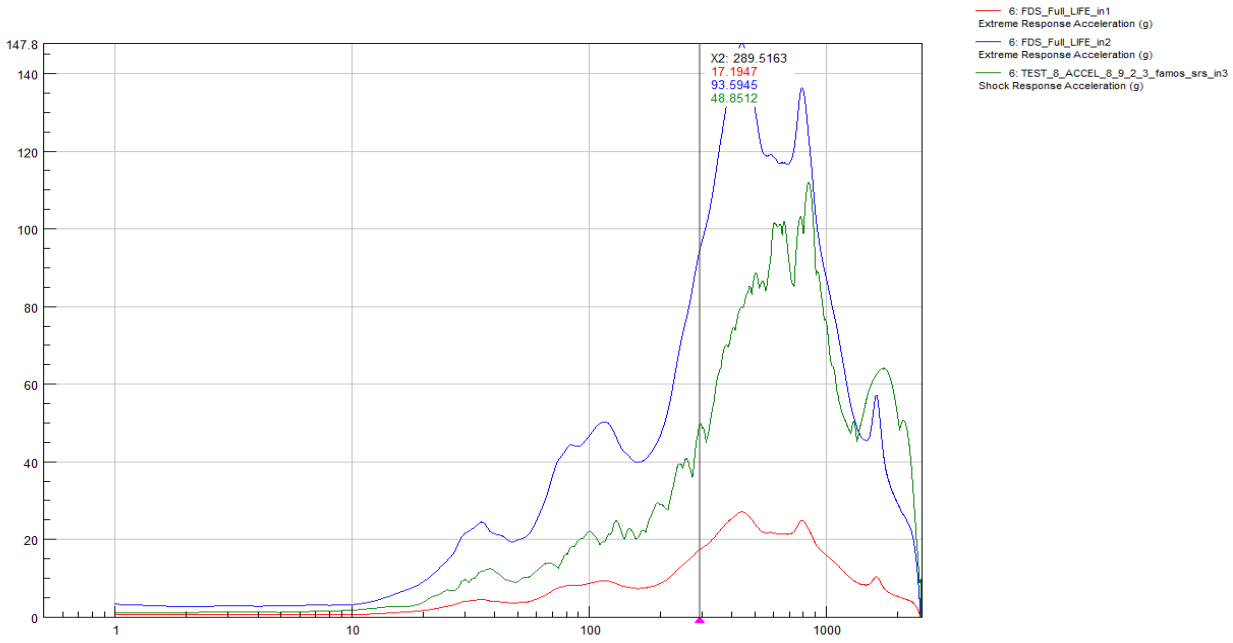


Fig 4.17 Accelerated time reduced to 100 hrs

- In second iteration the time was increased to 300 hrs and from fig 4.18, it can be observed that ERS and SRS are comparable and within range, this signifies that the testing time has decreased to 300 hours from 1,73,000 hours with realistic vibration levels.

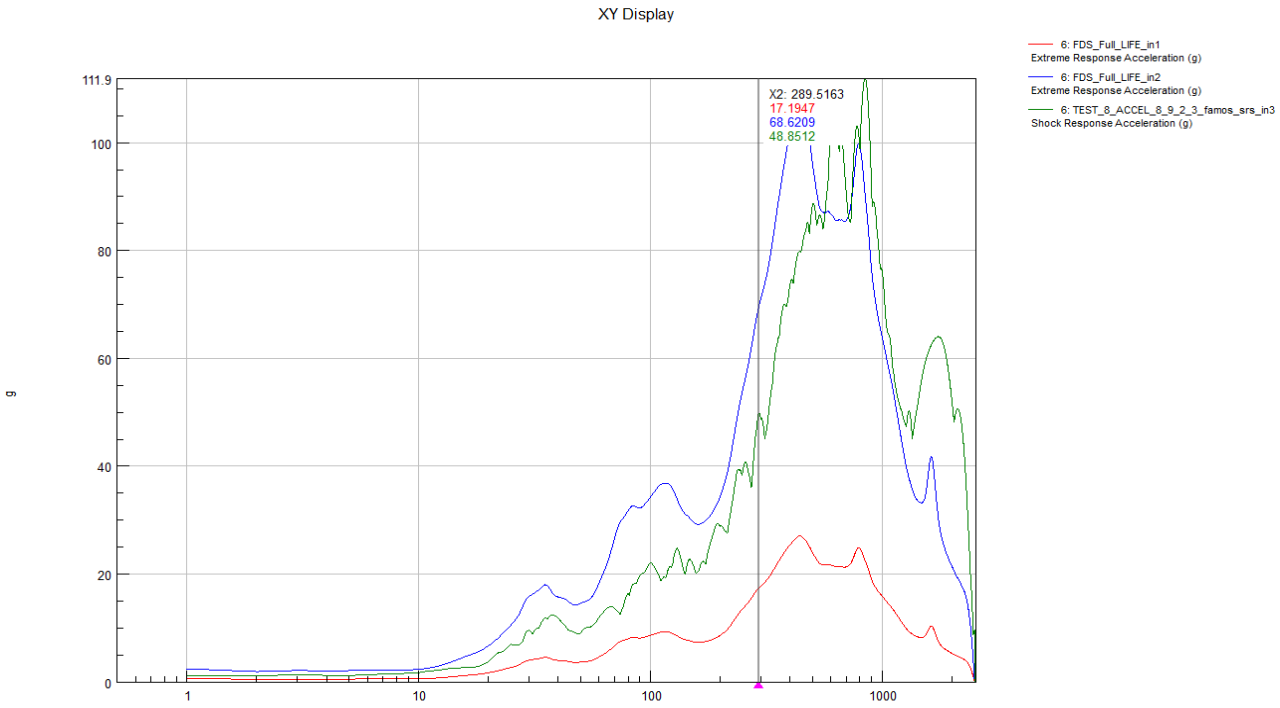


Fig 4.18 Accelerated time increased to 300 hrs

CHAPTER 5

Conclusion and Future Scope

5.1 Conclusions

In this project, the problem of extensive manual work for FEA analysis and correlation of various components and development of accelerated testing is addressed. Three automation tools were generated to decrease the manual work done by the structural engineers. Two tools for automatic Load combinations and Load step were created using python which will generate the TCL file compatible with Hyperworks. As input for the tools, Excel files were used where all the details about the name, scale, active SPCs and Load collectors were mentioned. For correlation tool, input was taken in form of excel and node set text files were generated from it. Using node set text files, Nodal stress results were exported in the excel. Test centre data was obtained at the same nodes using strain gauges and rosettes. The tool will divide the FE data and Test data into various zones and do the correlation between them for validation.

For accelerated testing, literature review was carried out regarding time series data acquisition, Power spectral density, Fatigue damage spectrum, shock response spectrum, Extreme response spectrum, rain flow counting and basic concept behind accelerated testing. For developing spectrums for accelerated testing and processing of time series data, N-code 17 is used. The time series data for accelerated testing is measured in real environment, then it is processed in N-code for spikes and running mean. After that, fatigue damage spectrum and shock response spectrum are derived using SRS functional glyph. The FDS is then repeated over for full life time using schedule for total damage in life time. Based on that the test is accelerated by decreasing the testing time and comparing it with the SRS to check that vibrations levels are realistic only and significant physical damage is not done to the testing prototype.

5.2 Future scope

With the use of the developed tools, the pre-processing time for FE analysis of components was decreased significantly as earlier they were manually giving input in the Hypermesh for around 100-120 Load-combinations and 80-90 Loadsteps. The tools are tested and now are in regular use

for all the structural teams in Bombardier transportation. Similarly, Correlation tool has decreased the post-processing time of FE results and its validation. Earlier, all the zone wise sheets were prepared manually by the test and validation engineer, which is now done by the tool increasing the productivity and removing the chances for manual error. These tools can be developed further also, as Load-combination tool is for Load collectors only but has to be developed for SPCs, Load-step tool is valid for static linear analysis only, not for thermal and non-linear analyses, so development can be done in there.

By the use of the spectrums generated through accelerated testing, the testing time of the product has decreased significantly, and the vibration levels generated by the spectrum are also realistic ones. As, Bombardier Transportation had no earlier base for accelerated testing and was outsourcing the work, but now, they can generate spectrums in-house only, cutting out sourcing cost also. The accelerated test generated in this project is based on load amplitude modification but there are also other methods which can be used, and generalized process can be developed for them also.

Appendix-I

Python code for Load-combination Generation:

```
def inputs(self, e):
    self.load_combs_file = self.tc1.GetValue()
    self.tcl_file = self.tc2.GetValue()
    self.number_of_loadcases = self.tc3.GetValue()
    self.number_of_loads = self.tc4.GetValue()

def generate_file(self, e):
    try:
        load_combs_file_name = self.load_combs_file + '.csv'
        f = open(load_combs_file_name, 'r')
        file_table = []
        for I in f:
            file_table.append(I.split())
        f.close()

        N = int(self.number_of_loads)
        O = int(self.number_of_loadcases)
        # S = 2 #first load ID number

        all_combs = []
        for A in range(5, N + 6):
            all_combs.append(file_table[A][0].split(","))
        for B in range(1, N + 1):
            del all_combs[B][1]

        load_IDs = file_table[5][2].split(',')
        load_IDs.remove('IDs')

        loadcombs = []
        for X in range(1, O + 1):
            lcombs = all_combs[1][X]
            for J in range(2, N + 1):
                lcombs += "," + all_combs[J][X]
            loadcombs.append(lcombs)

        tcl_file_name = self.tcl_file + '.tcl'
        h = open(tcl_file_name, 'w')
        for j in range(O):
            line = 'set lst' + str(j + 1) + ' [split "' + loadcombs[j] + '" ","]\n'
            h.write(line)

        names = []
        for n in range(N + 1):
            names.append(all_combs[n][0])

        for i in range(1, N + 1):
            load_places = []
            for lp in range(1, O + 1):
                if all_combs[i][lp] != '0':
                    load_places.append(lp)
            M = len(load_places)
            line = 'set name ' + names[i] + '\n'
            h.write(line)
            line = 'set j [expr ' + str(i) + '+' + str(O) + ']\n'
            h.write(line)
            line = 'set k ' + str(M) + '\n'
```

```
            h.write(line)
            h.write('*startnotehistorystate {Created loadcollector $name}\n')
            h.write('*collectorcreate loadcols $name "" 11\n')
            h.write('*createmark loadcols 2 $name\n')
            h.write(
                '*dictionaryload loadcols 2 "C:/Program
Files/Altair/2017/templates/feoutput/optistruct/optistruct" "LOADADD"\n')
            h.write('*startnotehistorystate {Attached attributes to loadcol $name}\n')
            h.write('*attributeupdateint loadcols $j 3240 1 2 0 1\n')
            h.write('*attributeupdatedouble loadcols $j 379 1 2 0 1\n')
            h.write('*attributeupdateint loadcols $j 3236 1 0 0 1\n')
            h.write('*createdoublearray 1 0\n')
            h.write('*attributeupdatedoublearray loadcols $j 380 1 2 0 1 1\n')
            h.write('*createarray 1 0\n')
            h.write('*attributeupdateentityidarray loadcols $j 383 1 2 0 loadcols 1
1\n')
            h.write('*endnotehistorystate {Attached attributes to loadcol $name}\n')
            h.write('*startnotehistorystate {Attached attributes to loadcol $name}\n')
            h.write('*attributeupdateint loadcols $j 3236 1 0 0 $k\n')
            line = 'set load1 [lindex $lst' + str(load_places[0]) + ' [expr ' + str(i)
+ '-1]]\n'
            h.write(line)
            loads = "$load1"
            for K in range(2, M + 1):
                line = 'set load' + str(K) + ' [lindex $lst' + str(load_places[K - 1])
+ ' [expr ' + str(
                    i) + '-1]]\n'
                h.write(line)
                loads += ' $load' + str(K)
            line = '*createdoublearray $k ' + loads + '\n'
            h.write(line)
            h.write('*attributeupdatedoublearray loadcols $j 380 1 2 0 1 $k\n')
            load_p = str(load_IDs[load_places[0] - 1])
            for P in range(1, M):
                load_p += ' ' + str(load_IDs[load_places[P] - 1])
            line = '*createarray $k ' + load_p + '\n'
            h.write(line)
            h.write('*attributeupdateentityidarray loadcols $j 383 1 2 0 loadcols 1
$k\n')
            h.write('*endnotehistorystate {Attached attributes to loadcol $name}\n')
            h.write('*endnotehistorystate {Created loadcollector $name}\n')
            line = '#iteration ' + str(i) + ' ends\n'
            h.write(line)
        h.close()
        print('LOADCOMBS TCL FILE IS GENERATED')
    except:
        print("LOADCOMBS TCL FILE IS NOT GENERATED")
```

Python code for Load-step Generation:

```
def inputs(self, e):
        self.load_step_file = self.tc1.GetValue()
        self.tcl_file = self.tc2.GetValue()

    def generate_file(self, e):
        try:
            import openpyxl

            excel_input = openpyxl.load_workbook(self.load_step_file + '.xlsx')
            loadstep_sheet = excel_input.active
```

```
            loadstep_matrix = [[str(i.value) for i in j] for j in loadstep_sheet]
            no_of_loadsteps = len(loadstep_matrix) - 4
            no_of_loadcombs = len(loadstep_matrix[4]) - 2
            loadsteps_IDs = []

            for j1 in range(9, len(loadstep_matrix)):
                for j2 in range(2, len(loadstep_matrix[4])):
                    if loadstep_matrix[j1][j2] == "1":
                        loadsteps_IDs.append(loadstep_matrix[7][j2])
                        loadsteps_IDs.append(loadstep_matrix[8][j2])

            no = []
            g = 0
            for l in range(1, no_of_loadsteps + 1):
                g += 1
                no.append(g)

            #print(loadsteps_IDs)

            k = 0
            u = 0
            tcl_file_name = self.tcl_file + '.tcl'
            h = open(tcl_file_name, 'w')
            for i in range(9, len(loadstep_matrix)):
                h.write('*startnotehistorystate {LoadSteps Creation}\n')
                line = '*createmark loadcols 1"' + loadsteps_IDs[k] + '" "' +
loadsteps_IDs[k + 2] + '"' + '\n'
                h.write(line)
                h.write('*createmark outputblocks 1\n')
                h.write('*createmark groups 1\n')
                line = '*loadstepscreate "' + loadstep_matrix[i][0] + '" 1' + '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 4143 1 1 0 1'
+ '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 4709 1 1 0 1'
+ '\n'
                h.write(line)
                line = '*attributeupdateentity loadsteps ' + str(no[u]) + ' 4145 1 1 0
loadcols ' + loadsteps_IDs[
                    k + 3] + '\n'
                h.write(line)
                line = '*attributeupdateentity loadsteps ' + str(no[u]) + ' 4147 1 1 0
loadcols ' + loadsteps_IDs[
                    k + 1] + '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 3800 1 1 0 1'
+ '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 707 1 1 0 1' +
'\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 2396 1 1 0 1'
+ '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 8134 1 1 0 1'
+ '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 2160 1 1 0 1'
+ '\n'
                h.write(line)
                line = '*attributeupdateint loadsteps ' + str(no[u]) + ' 10212 1 1 0 1'
+ '\n'
```

```
                h.write(line)
                h.write('*endnotehistorystate {LoadSteps Creation}\n')
                k += 4
                u += 1
            h.close()
            print('LOADSTEP TCL FILE IS GENERATED')

        except:
            print('LOADSTEP TCL FILE IS NOT GENERATED')


if __name__ == '__main__':
    app = wx.App()
    Example(None, title="Load Step TCL file Generator")
    app.MainLoop()
```

Python code for stress sheet generator:

```
import openpyxl

def node_zone_seperator(nodes):
    zones = {}
    for node in nodes:
        if node[:3] not in zones:
                zones[node[:3]] = [node]
        else:
            zones[node[:3]].append(node)
           # print(zones)
    return zones
excel_input = openpyxl.load_workbook('input_data.xlsx')
jauges_sheet = excel_input['jauges']
rosettes_sheet = excel_input['rosettes']
jauges_matrix = [[str(i.value) for i in j] for j in jauges_sheet]
rosettes_matrix = [[str(i.value) for i in j] for j in rosettes_sheet]
#print(rosettes_matrix)
gauge_nos = []
node_ids = ""
jauge_stress_direction = []
coord_sys = []
for j2 in range(1,len(jauges_matrix)):
    node_ids += jauges_matrix[j2][1] + ' '
    gauge_nos.append(jauges_matrix[j2][0])
    jauge_stress_direction.append(jauges_matrix[j2][3])
    coord_sys.append(jauges_matrix[j2][2])
rosette_a_direction = []
rosette_c_direction = []
for j2 in range(1,len(rosettes_matrix)):
    node_ids += rosettes_matrix[j2][1] + ' '
    gauge_nos.append(rosettes_matrix[j2][0])
    rosette_a_direction.append(rosettes_matrix[j2][3])
    rosette_c_direction.append(rosettes_matrix[j2][4])
    coord_sys.append(rosettes_matrix[j2][2])
nodes_list = node_ids.split(' ')[:-1]
#print(nodes_list)

node_gauge = {}
for j9 in range(len(nodes_list)):
    node_gauge[nodes_list[j9]] = gauge_nos[j9]

    import wx
```

```
class Example(wx.Frame):

    def __init__(self, parent, title):
        super(Example, self).__init__(parent, title=title,
                                      size=(400, 200))

        self.InitUI()
        self.Centre()
        self.Show()

    def InitUI(self):
        panel = wx.Panel(self)

        sizer = wx.GridBagSizer(5, 5)

        text1 = wx.StaticText(panel, label="stress sheets generator")
        sizer.Add(text1, pos=(0, 0), flag=wx.TOP | wx.LEFT | wx.BOTTOM,
                  border=15)
        button1 = wx.Button(panel, label="Generate node set text files")
        sizer.Add(button1, pos=(1,2))
        button1.Bind(wx.EVT_BUTTON, self.generatetextsfiles)

        button2 = wx.Button(panel, label="Generate stress sheets")
        sizer.Add(button2, pos=(2, 2))
        button2.Bind(wx.EVT_BUTTON, self.generatestresssheets)



        sizer.AddGrowableCol(2)

        panel.SetSizer(sizer)

    def generatetextsfiles(self,e):
        try:
            node_cord = {}
            for j20 in range(len(nodes_list)):
                node_cord[nodes_list[j20]] = coord_sys[j20]

            systems = {}
            for j21 in node_cord:
                if node_cord[j21] in systems:
                    systems[node_cord[j21]].append(j21)
                else:
                    systems[node_cord[j21]] = [j21]

            for j22 in systems:
                txt_file_name = 'node_ids_' + j22 + '.txt'
                G = open(txt_file_name, 'w')
                line1 = '*Group "' + j22 + ' Set" "node" "255   0   0"\n'
                G.write(line1)
                G.write('*Attributes point 3\n')
                G.write('*Pool "Node"\n')
                node_pool = ''
                #print(node_pool)
                #print(systems[j22])
                for j23 in systems[j22]:
                    node_pool += j23 + ' '
                G.write(node_pool)
                G.close()
            print('NODE SET TEXT FILES ARE GENERATED')
        except:
```

```
                print('NODE SET TEXT FILES ARE NOT GENERATED')


        def generatestresssheets(self,e):

                all_values = []
                for node in nodes_list:
                    filename = "Nodal_Stress_Reports\\Node_" + node +
'_StressReport.csv'

                    temp_mat = []
                    G = open(filename, 'r')
                    for j3 in G:
                        temp_mat.append(j3[:-1].split(','))
                    G.close()
                    all_values.append(temp_mat)
                all_values_t = [
                    [[all_values[k][j][i] for j in range(len(all_values[0]))] for i in
range(len(all_values[0][0]))] for
                    k
                    in range(len(all_values))]

                excel_main = openpyxl.Workbook()
                sheet_main = excel_main.active
                sheet_main.title = 'all_stresses'

                rng = 1

                cng = 2
                for j4 in range(len(all_values_t)):
                    for j5 in range(1, len(all_values_t[0])):
                        for j6 in range(len(all_values_t[0][0])):
                            rn = rng + j5
                            cn = cng + j6
                            sheet_main.cell(row=rn, column=cn).value =
all_values_t[j4][j5][j6]
                    rng += j5
                # print(rng)

                for j7 in range(3, len(all_values_t[0][0]) + 2):
                    sheet_main.cell(row=1, column=j7).value = all_values_t[0][0][j7 -
2]

                nn = -1
                for j8 in range(2, len(all_values_t) * 6 + 2):
                    if ((j8 - 2) % 6 == 0):
                        nn += 1
                        sheet_main.cell(row=j8, column=1).value =
node_gauge[str(nodes_list[nn])]
                        # print(nn)

                all_rows_gen_cs = sheet_main.rows
                all_columns_gen_cs = sheet_main.columns
                all_rows_cs = [[i for i in j] for j in all_rows_gen_cs]
                # print(all_rows_cs)
                all_columns_cs = [[i for i in j] for j in all_columns_gen_cs]
                R_cs = len(all_rows_cs)
                C_cs = len(all_columns_cs)
                nodes = [str(all_columns_cs[0][n].value) for n in range(1, R_cs, 6)]
                # print(C_cs)

                J_nodes = [node for node in nodes if node[0] == 'J']
                R_nodes = [node for node in nodes if node[0] == 'R']
```

```
                J_nodes_zones = node_zone_seperator(J_nodes)
                R_nodes_zones = node_zone_seperator(R_nodes)

                stresses_compr = [[all_columns_cs[0][j].value] for j in
                                  range(1, len(all_columns_cs[0]) - len(R_nodes) * 6)
                                  if all_columns_cs[0][j].value != None]
                zrsiga = [[all_columns_cs[0][j].value] for j in range(1,
len(all_columns_cs[0])) if
                          all_columns_cs[0][j].value != None]
                zrsigc = [[all_columns_cs[0][j].value] for j in range(1,
len(all_columns_cs[0])) if
                          all_columns_cs[0][j].value != None]
                z = (len(zrsiga))
                for j in range(1, z, 1):
                    if (str(zrsiga[j][0])[:1] == 'R'):
                        rosette = zrsiga[j][0]
                        zrsiga[j][0] = [rosette + ' sig a']
                zrsiga.insert(0, [i.value for i in all_rows_cs[0]][1:])

                for j in range(1, z, 1):
                    if (str(zrsigc[j][0])[:1] == 'R'):
                        rosette = zrsigc[j][0]
                        zrsigc[j][0] = [rosette + ' sig c']
                zrsigc.insert(0, [i.value for i in all_rows_cs[0]][1:])

                for i in range(len(stresses_compr) + 1, len(stresses_compr) +
len(R_nodes) + 1):
                    stresses_compr.append(zrsiga[i][0])
                    stresses_compr.append(zrsigc[i][0])

                stresses_compr.insert(0, [i.value for i in all_rows_cs[0]][1:])
                node_i = 0
                rosette_directions = []
                for i in range(len(rosette_a_direction)):
                    rosette_directions.append(rosette_a_direction[i])
                    rosette_directions.append(rosette_c_direction[i])
                gauges_rosettes_a_c_indices = [i for i in range(1, R_cs - len(R_nodes)
* 6)]
                # for rac in range(R_cs - len(R_nodes)*6,R_cs):
                #    gauges_rosettes_a_c_indices.append(rac)
                #    gauges_rosettes_a_c_indices.append(rac)
                for j3 in gauges_rosettes_a_c_indices:
                    if ((j3 - 1) % 6 == 0):
                        node_i += 1
                    stress_type = [jauge_stress_direction +
rosette_directions][0][node_i - 1]
                    for j4 in range(1, C_cs):
                        if (str(all_rows_cs[j3][1].value).strip() ==
stress_type.strip()):
                            stresses_compr[node_i].append(all_rows_cs[j3][j4].value)

                node_i = len(J_nodes) - 1
                gauges_rosettes_a_c_indices = [i for i in range(R_cs - len(R_nodes) *
6, R_cs)]
                for j3 in gauges_rosettes_a_c_indices:
                    if ((j3 - 1) % 6 == 0):
                        node_i += 2
                    stress_type = [jauge_stress_direction +
rosette_directions][0][node_i - 1]
                    for j4 in range(1, C_cs):
                        if (str(all_rows_cs[j3][1].value).strip() ==
stress_type.strip()):
                            stresses_compr[node_i].append(all_rows_cs[j3][j4].value)
```

```
                node_i = len(J_nodes)
                gauges_rosettes_a_c_indices = [i for i in range(R_cs - len(R_nodes) *
6, R_cs)]
                for j3 in gauges_rosettes_a_c_indices:
                    if ((j3 - 1) % 6 == 0):
                        node_i += 2
                    stress_type = [jauge_stress_direction +
rosette_directions][0][node_i - 1]
                    for j4 in range(1, C_cs):
                        if (str(all_rows_cs[j3][1].value).strip() ==
stress_type.strip()):
                            stresses_compr[node_i].append(all_rows_cs[j3][j4].value)

                sheet_cs_u = excel_main.create_sheet('unique_stresses_only')

                for j20 in range(len(stresses_compr[0])):
                    sheet_cs_u.cell(row=1, column=j20 + 2).value =
stresses_compr[0][j20]

                for j5 in range(2, len(stresses_compr) + 1):
                    for j6 in range(1, len(stresses_compr[j5 - 1]) + 1):
                        sheet_cs_u.cell(row=j5, column=j6).value = stresses_compr[j5 -
1][j6 - 1]

                excel_co = openpyxl.load_workbook('correlation.xlsx')
                sheet_co = excel_co['correlation']
                all_rows_gen_co = sheet_co.rows
                all_rows_ca = [[i for i in j] for j in all_rows_gen_co]
                # print(all_rows_ca)

                for j25 in range(0, len(J_nodes) + (len(R_nodes) * 3 + 1)):
                    for j26 in range(0, C_cs):
                        del all_rows_ca[j25][0]

                for j27 in range(len(J_nodes) + 2, len(J_nodes) + (len(R_nodes) * 3),
3):
                    for j28 in range(0, C_cs):
                        del all_rows_ca[j27][0]

                all_rows_co = []

                for j29 in range(0, len(J_nodes) + (len(R_nodes) * 3 + 1)):
                    if all_rows_ca[j29] != []:
                        all_rows_co.append(all_rows_ca[j29])

                diff_error = [[all_rows_co[j][0].value] for j in range(2,
len(all_rows_co))]
                diff_error.insert(0, all_rows_cs[0])
                print(stresses_compr)
                print(all_rows_co)

                diff_mat = [[all_rows_co[i][j].value - float(stresses_compr[i][j]) for
j in range(2, len(stresses_compr[i]))]
                    for i in range(1, (len(stresses_compr)))]

                sheet_cs_u_diff = excel_main.copy_worksheet(sheet_cs_u)
                sheet_cs_u_diff.title = 'stresses_difference'
                for j7 in range(2, len(diff_mat) + 2):
                    for j8 in range(2, len(diff_mat[j7 - 2]) + 2):
                        sheet_cs_u_diff.cell(row=j7, column=j8 + 1).value = diff_mat[j7
- 2][j8 - 2]
```

```
                import math

                excel_input1 = openpyxl.load_workbook('correlation.xlsx')
                correlation_sheet = excel_input1['correlation']
                correlation_matrix = [[str(i.value) for i in j] for j in
correlation_sheet]

                all_rows_gen_cs1 = correlation_sheet.rows
                all_columns_gen_cs1 = correlation_sheet.columns
                all_rows_cs1 = [[i for i in j] for j in all_rows_gen_cs1]
                all_columns_cs1 = [[i for i in j] for j in all_columns_gen_cs1]
                R_cs1 = len(all_rows_cs1)
                C_cs1 = len(all_columns_cs1)
                loadcase = int(C_cs1 / 2)

                j = 0
                for j1 in range(0, R_cs1):
                    if correlation_matrix[j1][0][:1] == "J":
                        j += 1
                j += 1

                sheet_correlate_u = excel_input1.create_sheet('update rosettes PI PII')

                sigmastress = [correlation_matrix[i] for i in range(j, R_cs1)]

                rsigma1 = []
                rsigma2 = []
                rphi = []

                for j2 in range(0, len(sigmastress), 3):
                    for j3 in range(2, loadcase):
                        r = float(sigmastress[j2][j3])
                        c = float(sigmastress[j2 + 1][j3])
                        d = float(sigmastress[j2 + 2][j3])
                        k = 0.15 * (r + d) + (0.11422 * math.sqrt((r - c) * (r - c) +
(d - c) * (d - c)))
                        e = 0.15 * (r + d) - (0.11422 * math.sqrt((r - c) * (r - c) +
(d - c) * (d - c)))
                        p = (math.atan2((2 * c - r - d), (r - d)))
                        p = p * 28.64
                        rsigma1.append(k)
                        rsigma2.append(e)
                        rphi.append(p)
                        sigma1 = ['%.2f' % elem for elem in rsigma1]
                        sigma2 = ['%.2f' % elem for elem in rsigma2]
                        phi = ['%.2f' % elem for elem in rphi]

                k = 0
                for j5 in range(1, R_cs1 + 1 - j, 3):
                    for j6 in range(3, 1 + loadcase):
                        sheet_correlate_u.cell(row=j5, column=j6).value = sigma1[k]
                        k += 1
                w = 0
                for j7 in range(2, R_cs1 + 1 - j, 3):
                    for j8 in range(3, 1 + loadcase):
                        sheet_correlate_u.cell(row=j7, column=j8).value = sigma2[w]
                        w += 1
                i = 0
                for j11 in range(3, R_cs1 + 1 - j, 3):
                    for j12 in range(3, 1 + loadcase):
                        sheet_correlate_u.cell(row=j11, column=j12).value = phi[i]
                        i += 1
```

```
                name = []
                for j9 in range(0, len(sigmastress), 3):
                    name.append(sigmastress[j9][0][:5])

                y = 0
                for j10 in range(1, R_cs1 - j, 3):
                    sheet_correlate_u.cell(row=j10, column=1).value = name[y] + "sig1"
                    sheet_correlate_u.cell(row=j10, column=2).value = "N/mm2"
                    sheet_correlate_u.cell(row=j10 + 1, column=1).value = name[y] +
"sig2"
                    sheet_correlate_u.cell(row=j10 + 1, column=2).value = "N/mm2"
                    sheet_correlate_u.cell(row=j10 + 2, column=1).value = name[y] +
"phi"
                    sheet_correlate_u.cell(row=j10 + 2, column=2).value = "grad"
                    y += 1

                    excel_input1.save("correlation.xlsx")

                for zone in J_nodes_zones:
                    sheet_zone_name = 'Zone' + zone[-1] + '00' + zone[0].lower()
                    sheet_zone = excel_main.create_sheet(sheet_zone_name)
                    for j9 in range(len(stresses_compr[0])):
                        sheet_zone.cell(row=2, column=j9 + 2).value =
stresses_compr[0][j9]

                    sheet_zone.cell(row=1, column=1).value = 'FEM DATA'
                    row_n = 2
                    for j10 in range(1, len(stresses_compr)):
                        if str(stresses_compr[j10][0][:3]) == zone and
str(stresses_compr[j10][0]) in J_nodes_zones[zone]:
                            row_n += 1
                            for j11 in range(len(stresses_compr[j10])):
                                sheet_zone.cell(row=row_n, column=j11 + 1).value =
stresses_compr[j10][j11]
                                # print(stresses_compr)

                    row_n += 2
                    sheet_zone.cell(row=row_n, column=1).value = "TEST DATA"
                    row_n += 1

                    r = row_n-1
                    for j9 in range(len(all_rows_co[0])):
                        sheet_zone.cell(row=row_n, column=j9 + 1).value =
all_rows_co[0][j9].value
                    for j10 in range(1, len(all_rows_co)):
                        if str(all_rows_co[j10][0].value[:3]) == zone and
str(all_rows_co[j10][0].value) in \
                                J_nodes_zones[zone]:
                            row_n += 1
                            for j11 in range(len(all_rows_co[j10])):
                                sheet_zone.cell(row=row_n, column=j11 + 1).value =
all_rows_co[j10][j11].value
                    s = row_n + 1
                    k = 1
                    for z in range(r, s):
                        sheet_zone.cell(row=z, column=2).value = sheet_zone.cell(row=k,
column=2).value
                        k += 1

                    row_n += 2
                    sheet_zone.cell(row=row_n, column=1).value = "(TEST DATA-FEM DATA)"
                    row_n += 1
```

```
                    d = row_n-1
                    for j9 in range(len(all_rows_co[0])):
                        sheet_zone.cell(row=row_n, column=j9 + 1).value =
all_rows_co[0][j9].value
                    for j10 in range(1, len(diff_mat) + 1):
                        if str(all_rows_co[j10][0].value[:3]) == zone and
str(all_rows_co[j10][0].value) in \
                                J_nodes_zones[zone]:

                            row_n += 1
                            sheet_zone.cell(row=row_n, column=1).value =
all_rows_co[j10][0].value
                            for j11 in range(1, len(diff_mat[j10 - 1]) + 1):
                                sheet_zone.cell(row=row_n, column=j11 + 2).value =
diff_mat[j10 - 1][j11 - 1]
                    e = row_n + 1
                    q = 1
                    for z in range(d, e):
                        sheet_zone.cell(row=z, column=2).value = sheet_zone.cell(row=q,
column=2).value
                        q += 1

                excel_sigma = openpyxl.load_workbook('correlation.xlsx')
                sheet_sigma = excel_sigma['update rosettes PI PII']
                sigma_zone = [[str(i.value) for i in j] for j in sheet_sigma]



                for zone in R_nodes_zones:
                    sheet_zone_name = 'Zone' + zone[-1] + '00' + zone[0].lower()
                    sheet_zone = excel_main.create_sheet(sheet_zone_name)
                    for j9 in range(len(stresses_compr[0])):
                        sheet_zone.cell(row=1, column=j9 + 2).value =
stresses_compr[0][j9]
                    sheet_zone.cell(row=1, column=1).value = 'FEM DATA'

                    row_n = 1
                    for j10 in range(1, len(stresses_compr)):
                        if str(stresses_compr[j10][0][:3]) == zone and
str(stresses_compr[j10][0][:5]) in R_nodes_zones[zone]:

                            row_n += 1
                            for j11 in range(len(stresses_compr[j10])):
                                sheet_zone.cell(row=row_n, column=j11 + 1).value =
stresses_compr[j10][j11]
                    number_case = row_n - 1

                    row_n += 2

                    for j90 in range(1, len(all_rows_cs), 6):
                        if str(all_rows_cs[j90][0].value[:3]) == zone and
str(all_rows_cs[j90][0].value) in \
                                R_nodes_zones[zone]:
                            row_n += 1
                            for j51 in range(1, len(all_rows_cs[j90]) - 1):
                                sheet_zone.cell(row=row_n, column=j51 + 2).value =
all_rows_cs[j90 + 3][j51 + 1].value
                                sheet_zone.cell(row=row_n, column=2).value = 'N/mm2'

                            row_n += 1
                            for j51 in range(1, len(all_rows_cs[j90]) - 1):
                                sheet_zone.cell(row=row_n, column=j51 + 2).value =
```

```
all_rows_cs[j90 + 5][j51 + 1].value
                                sheet_zone.cell(row=row_n, column=2).value = 'N/mm2'

                    fem = [[sheet_zone.cell(row=z, column=n).value for n in range(3,
len(all_rows_cs[0]) + 1)] for z in
                           range(number_case + 4, (2 * number_case + 4))]
                    # print(fem)
                    row_n += 4
                    sheet_zone.cell(row=row_n, column=1).value = 'TEST DATA'
                    row_n += 1
                    l = row_n

                    for j9 in range(len(all_rows_co[0])):
                        sheet_zone.cell(row=row_n, column=j9 + 1).value =
all_rows_co[0][j9].value
                    for j10 in range(1, len(all_rows_co)):
                        if str(all_rows_co[j10][0].value[:3]) == zone and
str(all_rows_co[j10][0].value[:5]) in \
                                R_nodes_zones[zone]:

                            row_n += 1
                            for j11 in range(len(all_rows_co[j10])):
                                sheet_zone.cell(row=row_n, column=j11 + 1).value =
all_rows_co[j10][j11].value
                    w = row_n + 1
                    u = 1
                    for z in range(l, w):
                        sheet_zone.cell(row=z, column=2).value = sheet_zone.cell(row=u,
column=2).value
                        u += 1

                    row_n += 2

                    for j10 in range(len(sigma_zone)):
                        if str(sigma_zone[j10][0][:3]) == zone and
str(sigma_zone[j10][0][:5]) in R_nodes_zones[zone]:
                            if str(sigma_zone[j10][0][5:]) != 'phi':
                                row_n += 1
                                for j11 in range(len(sigma_zone[j10])):
                                    sheet_zone.cell(row=row_n, column=j11 + 1).value =
sigma_zone[j10][j11]
                    test = [[sheet_zone.cell(row=z, column=n).value for n in range(3,
len(all_rows_cs[0]) + 1)] for z in
                            range(((3 * number_case) + 11), ((4 * number_case) + 11))]
                    # print(test)
                    correlate = [[float(test[i][j]) - float(fem[i][j]) for j in
range(len(fem[0]))] for i in
                                 range(len(fem))]
                    correlate_print = []
                    for j101 in range(len(correlate)):
                        correlate1 = ['%.2f' % elem for elem in correlate[j101]]
                        correlate_print.append(correlate1)
                    # print(correlate_print)
                    row_n += 4
                    sheet_zone.cell(row=row_n, column=1).value = '(TEST DATA-FEM DATA)'
                    row_n += 1

                    f = row_n
                    for j9 in range(len(all_rows_co[0])):
                        sheet_zone.cell(row=row_n, column=j9 + 1).value =
all_rows_co[0][j9].value
                    for j10 in range(1, len(diff_mat) + 1):
                        if str(all_rows_co[j10][0].value[:3]) == zone and
```

```
str(all_rows_co[j10][0].value[:5]) in \
                                R_nodes_zones[zone]:
                            row_n += 1
                            sheet_zone.cell(row=row_n, column=1).value =
all_rows_co[j10][0].value
                            for j11 in range(1, len(diff_mat[j10 - 1]) + 1):
                                sheet_zone.cell(row=row_n, column=j11 + 2).value =
diff_mat[j10 - 1][j11 - 1]
                    c = row_n + 1
                    b = 1
                    for z in range(f, c):
                        sheet_zone.cell(row=z, column=2).value = sheet_zone.cell(row=b,
column=2).value
                        b += 1

                    row_n += 3
                    for j102 in range(len(correlate_print)):
                        for j103 in range(len(correlate_print[j102])):
                            sheet_zone.cell(row=row_n, column=j103 + 3).value =
correlate_print[j102][j103]
                            sheet_zone.cell(row=row_n, column=2).value = 'N/mm2'
                        row_n += 1

                    re = (3 * number_case) + 11
                    for z in range(number_case + 4, 2 * number_case + 4):
                        sheet_zone.cell(row=z, column=1).value =
sheet_zone.cell(row=re, column=1).value
                        re += 1

                    ra = (3 * number_case) + 11
                    for z in range(5 * number_case + 18, 6 * number_case + 18):
                        sheet_zone.cell(row=z, column=1).value =
sheet_zone.cell(row=ra, column=1).value
                        ra += 1

                    excel_main.save("stress_sheets_final.xlsx")
                print('STRESS SHEETS ARE GENERATED')




if __name__ == '__main__':
    app = wx.App()
    Example(None, title="stress sheet generator")
    app.MainLoop()
```

REFERENCES

[1] Ascher, D. and Lutz, M., 1999. *Learning Python*. O'Reilly.

[2] McNeill, S.I., 2008. Implementing the fatigue damage spectrum and fatigue damage equivalent vibration testing. In *Presented at the 79th Shock and Vibration Symposium: October* (Vol. 26, p. 30).

[3] Irvine, T., 2002. An introduction to the shock response spectrum. *Rev P, Vibrationdata*.

[4] Van Baren, J., Van Baren, P. and Jenison, M.I., 2012. The fatigue damage spectrum and kurtosis control. *Sound and Vibration*, *46*(10), p.10.

[5] Farrar, C.R., Duffey, T.A., Cornwell, P.J. and Bement, M.T., 1999, February. A review of methods for developing accelerated testing criteria. In *Proc. of the 17th International Modal Analysis Conference, Kissimmee, FL* (pp. 608-614).

[6] Feinberg, A., 2015, January. Thermodynamic damage measurements of an operating system. In *Reliability and Maintainability Symposium (RAMS), 2015 Annual* (pp. 1-6). IEEE.

[7] ZAhARIA, S.M., Martinescu, I. and Morariu, C.O., 2012. LIFE TIME PREDICTION USING ACCELERATED TEST DATA OF THE SPECIMENS FROM MECHANICAL ELEMENT PROGNOZOWANIE CZASU PRACY ELEMENTU MECHANICZNEGO Z WYKORZYSTANIEM DANYCH Z BADAŃ PRZYSPIESZONYCH. *EKSPLOATACJA I NIEZAWODNOSC*, *14*(2).

[8] Lantieri, P. and Bignonnet, A., 2011. Optimization of an Accelerated Step-Stress Fatigue Test Plan. *Quality Technology & Quantitative Management*, *8*(4), pp.429-438.

[9] Using Fatigue Damage Spectrum for Accelerated Testing with Correlation to End-Use Environment, Tom Achatz, PE and John VanBaren, PE

[10] Halfpenny, A., 2006, July. Methods for accelerating dynamic durability tests. In *Proceedings of the 9th International Conference on Recent Advances in Structural Dynamics, Southampton, UK* (pp. 17-19).