Indian Institute of Technology, Roorkee

DYNAMIC ANALYSIS OF MEMBRANE REFLECTOR FOR SPACE APPLICATION

by

AYUSH TIWARI (12117017)

HARSHIT CHOPRA (12117027)

A thesis submitted in partial fulfillment for the
degree of Doctor of Philosophy

in the

Faculty Name
Department of Mechanical and Industrial Engineering

May 2016

INDIAN INSTITUTE OF TECHNOLOGY

Candidate's Declaration

I, AUTHOR NAME, declare that this thesis titled, 'THESIS TITLE' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

In space agencies nowadays, light weight structures are popular because of its cost for launching from ground station to space. Inflated structures means literally "can be inflated by internal gas pressure".

The inflatable space structure consists of assembly of torus, strut and reflector. These are the main inflatable parts of antenna. Strut and torus are the supporting part of reflector.

In space, atmosphere and challenges for surviving is different than earth. Inflatable antenna faces thermal expansion, bombarding particles stone randomly and vibration challenges while launching by launch vehicle's high frequency. One possible way to minimize the vibration of gossamer structure is use of smart materials. Smart material posses the intelligent properties e.g. PVDF piezo patches, shape memory alloy etc.

In this B.Tech Report, experimental work on Kapton membrane is carried out for the measurements of out of plane displacement of kapton membrane with the help of actuation by PZT and PVDF. Dynamic analysis has been done for temperature variation ranging from -200 to 200 degree Celsius. The optimal magnitude of the forces applied on the boundaries by the torus are calculated using Genetic Algorithm. Mode shapes and points where wrinkles are formed on a membrane are calculated.

Acknowledgements

We wish to express our deep sense of gratitude and sincere thanks to our guide **Dr. S. H. Upadhyay**, Associate Professor, Department of Mechanical and Industrial Engineering, IIT Roorkee, for being helpful and a great source of inspiration. We would like to thank him for providing us with an opportunity to work on this excellent and innovative field of research. His keen interest and constant encouragement gave us the confidence to complete our work. We wish to thank him for his constant guidance and suggestions without which we could not have successfully completed this project.

We are thankful to **Mr Anil C. Mathur** Group Director and **Mr. Kripa Shankar Singh**, Scientist/Engineer SD, Antenna System, Mechanical Group (ASMG), Mechanical Engineering System Area(MESA), SAC, ISRO, Ahmedabad for their valuable and significant suggestions which substansially improved our research.

We would also like to thank **Prof. Dinesh Kumar**, Head of Department, Department of Mechanical and Industrial Engineering, IIT Roorkee for his constant support during our study. Also we would like to thank all the teaching and non-teaching staff members of the department who have contributed directly or indirectly in successful completion of our project work.

We are very thankful to our parents & all of our friends for their never ending encouragement in bringing out this project report.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Gossamer Structure

Gossamer is fine spider web that floats on the wind. It gives its name to a special class of structures known as gossamer structures. These structures resemble a spider web with their high flexibility and ultra-low-mass. Small stowed volume and ultra-low-mass are highly desirable characteristics for space applications. Space launch vehicles (space rockets) have restricted dimensions in the payload bays and can carry limited payload mass to space. Gossamer structures technology allows engineers to devise large space structures that would be unfeasible with more traditional structures technology. Gossamer space structures are a recent development in structures technology.

1.1.1 Characteristics of gossamer structures

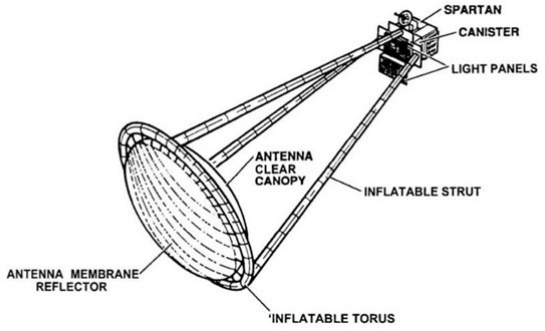
A key characteristic of gossamer structures is the areal density, area divided by mass, and is usually expressed in grams per square metre (gm/m2). Areal densities for membrane and gossamer structures in space tend to be very low, less than 10 gm/m2 for some structures. Compare this figure to standard photocopy paper, which has a density of 80 gm/m2 . Flexibility is another characteristic of gossamer structures. Gossamer structures typically use membranes as the main component. The structural definition of a membrane is a sheet of material that has negligible bending stiffness, hence the high flexibility of gossamer structures. The lack of significant bending stiffness means that the membrane cannot support load via bending, nor can it support load via compression as it buckles and wrinkles at negligible load. Tension is the only load path that is available in a membrane.

Working with structures made from flexible membranes requires a change in mindset for engineers, who tend to be familiar with more conventional structures where bending and compression can be used to carry load. Developing intuition for the behaviour of membrane materials and gossamer structures is essential for successful design, construction and operation.

1.2 Antenna Configuration

An antenna (or aerial) is an electrical device which converts electric power into radio waves, and vice versa. It is usually used with a radio transmitter or radio receiver. In transmission, a radio transmitter supplies an electric current oscillating at radio frequency (i.e. a high frequency alternating current (AC)) to the antenna's terminals, and the antenna radiates the energy from the current as electromagnetic waves (radio waves). In reception, an antenna intercepts some of the power of an electromagnetic wave in order to produce a tiny voltage at its terminals that is applied to a receiver to be amplified.

1.2.1 Inflatable Antenna



1.3 Membrane

A membrane is a thin, synthetic or natural, pliable material that constitutes the lightest material means for spatial organization and environmental modulation. Membrane systems are well suited for a wide range of uses in relation to the built environment, both as selfcontained architectures or as supplementary intermediate spatial interventions. Such supplementary architectures provide potential for a different approach to environmentally sustainable design.

1.3.1 Membrane Sctructure

Membrane structures consist of thin membrane or fabric as a major structural element. A membrane has no compression or bending stiffness, therefore it has to be prestressed to act as a structural element. The type of structures that is of interest in the present study is high-precision deployables for spacecraft, where is a growing requirement for furlable reflecting surfaces for antennae, reflectors and solar arrays. The configurations that are being considered include flat prestressed membrane panels and paraboloidal prestressed membranes formed by contiguous cylindrical pieces. The performance efficiency of these reflective surfaces depends not only on the geometric accuracy of the surface but also on its vibration characteristics.

Earlier, the traditional structures which were made of Shape memory alloys ,Aluminium, having some intrinsic disadvantages, such as high weight, high cost and high deployable shock effect, which will take up much room in spacecraft and decrease the effectiveness of aerospace missions. The large advantages of some new kinds of materials and rigidizable space structures in increased bandwidth capacity, stowage volume, less weight will result in significant savings in the cost of space science and communications missions. Hence an inflated structure can be used in space antenna structure, synthetic aperture radar etc.

Kapton, Kevlar, Mylar are the most used membrane materials by spatial organization. All three have good mechanical, chemical and electrical properties that required for space applications. The properties of the membranes are mentioned in table 1. as below :

Properties of Membrane					
Materials	Modulus of elasticity (GPa)	Tensile Strength (MPa)	Density (g/cm3)	Elongation at break	Thermal Conductivity (w/m.K)
Kapton	2-3	139-231	1.43	65-75	0.46
Kevlar	70-130	3300-3600	1.4	2.5-3.6	0.04
Mylar	5-5.1	170-230	1.39	75-125	0.14
SMA	20-80	800-1900	6.45	20-25	18
Aluminium	70-130	70-700	2.7	10-25	205

TABLE 1.1: Different membrane materials and their properties

1.4 Wrinkling

Wrinkles continue to bother younger and older researchers alike in the field of membrane mirrors and deployable satellites. Membrane materials inherently cannot sustain a compressive stress field, and react accordingly by creating out-of-plane displacements wrinkles. Thin membranes wrinkle whenever they are subjected to compressive load. This is due to the inherently small bending resistance of membrane structures which buckle out-of-plane under the action of even small in-plane compressive stress. This phenomenon is common and it can be observed in many structures in daily use, ranging from umbrellas and temporary tents to large fabric roofs for airports and stadiums. Wrinkling is not a concern for designing small structures because it has no structural consequences. In large scale structures, such as membrane roofs, wrinkling can be counteracted by applying a biaxial stress state by means of cables or compressive rings, as shown in Figure 1.1. More often a compensation scheme is used, in which the fabric is cut to a smaller size than that required to generate purely the geometric shapes that are required according to the formfinding process. Additional stretching of the fabric is thus required in order to eliminate potential wrinkles.

1.4.1 Types of Wrinkling

Wrinkles due to loading or boundary conditions are completely reversible if the membrane does not yield. These wrinkles are termed structural/elastic wrinkles; their magnitude varies with the loading and boundary conditions. Wrinkles where the material yields are permanent and irreversible, and are termed material/plastic wrinkles. This type of wrinkles can be random and can be thought of as an initial set of very extreme imperfections (Wong and Pellegrino, 2006). Only structural wrinkles are considered in the present study.

1.4.2 Wrinkling Criteria

The wrinkling criteria described above are summarized below, where σ_1, σ_2 and ϵ_1 , ϵ_2 are the major and minor principal stresses and strains respectively (hence $\sigma_1 \geq \sigma_2$ and $\epsilon_1 \geq \epsilon_2$).

1. Principal stress criterion
Taut: $\sigma_2 > 0$

Wrinkled: $\sigma_2 \leq 0, \sigma_1 > 0$

Slack: $\sigma_2 \leq 0, \sigma_1 \leq 0$

2. Principal strain criterion

Taut: $\epsilon_1 > 0, \epsilon_2 > \upsilon\epsilon_1$

Wrinkled: $\epsilon_1 > 0, \epsilon_1 \leq \upsilon\epsilon_1$

Slack: $\epsilon_1 \leq 0, \epsilon_1 \leq 0$

3. Combined criterion

Taut: $\sigma_2 > 0$

Wrinkled: $\sigma_2 \leq 0, \epsilon_1 > 0$

Slack: $\sigma_2 \leq 0, \epsilon_1 \leq 0$

The first two criteria, based on principal stresses or strains only, are fairly easy to apply. However, the slack regions defined by the principal stress criterion may in fact experience positive strain. Whereas, the principal strain criterion may underestimate the taut regions in the membrane since negative strain may exist together with a positive minor principal stress due to Poissons effects. The combined criterion overcomes the potential shortcomings mentioned above and provides the most accurate description of a wrinkled membrane.

1.5 Genetic Algorithm

1.5.1 Control of a Membrane Structure

The nonlinear behaviour of the membrane antenna with several shape memory alloy actuators is very difficult to characterize, and finding the tension combination that yields a maximally flat membrane surface can be formulated as an optimization problem. Most conventional optimization methods are based upon calculus and require a good initial guess of the parameter as well as derivatives of the objective function. Conventional methods can fall into local minima, rather than the the global minimum, as they are point-to-point search techniques. Genetic algorithms on the other hand are stochastic algorithms capable of searching the entire solution space with more likelihood of finding the global optimum when the system is either hard to characterize, nonlinear, or little is known about it ahead of time. Genetic algorithms are based on Darwin's 'survival of the fittest' principle of evolution and consist of a population of individuals that each represent a possible solution to the optimization problem. This population of possible solutions is then used to create a new population by evolutionary means such as selection, crossover, and mutation.

By searching in parallel, the algorithm is much less likely to fall into local minima, and more likely to find the global minima. However, GAs are typically much more computationally intensive to run than conventional optimization methods, and thus take longer to produce a solution.

Each candidate function is coded as a chromosome, and the population is initialized randomly. The fitness (or objective) function that is evaluated is

$$F(ci) = drms$$

where drms is the root-mean-square deviation of the membrane surface above a plane of best fit as measured by the photogrammetry system. After each generation, the relative fitness of each individual is evaluated. If the population size is N and the frequency of crossover is given by pc, then the number of individuals, nc, selected for crossover is

$$nc = N \times pc/2$$

The offspring can replace their parents in subsequent generations provided that their fitness values are superior. To avoid the possibility of zero-probability for a given solution to propagate, we allow for mutation at a rate given by pm. Finally, the maximum number of generations, Gmax, limits the number of times that crossover and mutation can occur while searching for optimal solutions. Once the maximum number of generations is reached, the process ends.

1.6 Fast Fourier Transformation

A fast Fourier transform (FFT) algorithm computes the discrete Fourier transform (DFT) of a sequence, or its inverse. Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors. As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to **O(n logn)**, where n is the data size.
Source : Wikipedia

Chapter 2

Literature Review

Deformation in inflatable membrane has been analyzed and experimented at different stages and there have been many approaches tried over the years. Work on both thermal as as well as force load had been done and focusing on minimizing the out of plane displacement. There are number of researchers who have worked or working on different aspects of inflatable membrane.

2.1 Wrinkling

[1] : Salama M and Jenkins C H investigate a technique for shaping the membranes which uses curved boundary elements to achieve coarse shaping, and a pair of precision rails shaped by moments and forces at the ends, and lightly pushed into the surface, to provide fine shape control.An experimental test bed has been constructed to demonstrate the ideas using a 70cm by 80cm membrane reflector.

[2] : S.C. Gajbhiye et. al. shown about Internal pressure becomes the major source of strength and rigidity, essentially stiffen the structure. The toroidal shaped structure helps to support the reflector in space application. This paper discusses the finite-element analysis of an inflated torus. The eigen-frequencies are obtained via three-dimensional small-strain elasticity theory, based on extremum energy principle.

[3] : S. C. Gajbhiye et. al. discussed about an inflated toroidal structure (torus) is often used to provide structural support to many inflatable-structure systems. he results obtained by Jordan eliminated the incompatibility of the displacements. Presented an extensive study of the free vibration analysis of a toroidal membrane

subjected to an internal pressure using the finite difference method. He demonstrated that only lower frequencies are affected due to internal pressure. cases have been generated using a commercial finite-element package.

[4] : H Fang et al developed and analyzed Analytical model (integrated reflector, actuator, and controller) Fabrication process for Electroactive Polymer (EAP) actuators has been developed by him with fabrication of EAP actuators.

[5] : Allik H and Hughes TJR this paper presents the finite-element investigation of a rectangular, flat thin membrane using polyvinylidene fluoride (PVDF) piezo-actuated material as an actuator/sensor. Investigation shows that rather than using the various numbers of patches to the practical system for controlling their vibration behaviour, the single patch with the appropriate thickness can easily control the desired vibration behaviour.

[6] : They present a general analytical model for determining the location and pattern of wrinkles in thin membranes and for making preliminary estimates of their wavelength and amplitude. A rectangular membrane under simple shear and a square membrane subject to corner loads are analysed. In the first problem, our model predicts the wavelength and the wrinkle amplitude to be respectively inversely proportional and directly proportional to the fourth root of the shear angle.

[7] : High-fidelity, geometrically nonlinear finite element models of membrane structures, based on thin-shell elements, are used to simulate the onset and growth of wrinkles. The simulations are carried out with the ABAQUS finite element package.

[8] : A novel shell-membrane concept is presented in this paper to define the wrinkling analytical object. A stress field model is established and applied to determine the different regions (taut, wrinkled and slack regions) of the shell-membrane. An analytical model based on the bifurcation theory of thin-plate is introduced to predict the wrinkling wavelength and amplitude.

[9] : An active flatness control system is developed to control and maintain the required surface flatness of the membrane antenna. Taking a mechatronics approach, twenty shape memory alloy (SMA) actuators are installed around the membrane boundary to apply tension forces to the membrane. Wrinkling-related surface deviations have a highly nonlinear relationship with applied tension forces, which can have different forms when the membrane is subject to different thermal disturbances.

[10] : Umesh A. Korde explained about Membrane mirrors may be attractive in some space applications where light weight, deploy ability, and conformability are

desired. This article investigates large displacement closed-loop control of electrostatically driven mirrors being used in focusing and steering of laser beams. A dynamic observer based on quad-cell beam-position detector measurements and a fully nonlinear plant model is also studied. The article discusses numerical simulation results, which show that the controllers under study are fairly successful at providing closed-loop deflections approaching the full gap size at the required bandwidth.

[11] : This PHD thesis by Jeffrey Ray Hill proposes to achieve high levels of surface control of lightweight reflectors by using PVDF actuators

There are also others such as [12] who worked on Corner wrinkling of a square membrane, [13], which did experimental measurement of wrinkling in membranes undergoing planar deformation, [14] , who worked on nonlinear dynamic model and free vibration analysis of deployable mesh reflectors. These also helped us in our research process.

2.2 Thermal variation

[15] : Thermal loads can cause significant surface errors of deployable mesh reflectors. In this work, a nonlinear dynamic model describing elastic-thermal dynamic properties of this type of space structures is developed.

[16] : Bladino et al worked on effect of asymmetric mechanical and thermal loading on membrane wrinkling developing methods to measure wrinkling. There are other works such as [17], [18], [19], [20] by Peng et al which had been on space reflectors actuation.

2.3 Force Analysis

[21] : Mounting of piezos changes the natural frequencies of existing structure. Multi-objective genetic algorithm is shown to be useful for optimization of placement and sizing of piezoactuators for such problems

[22] : This paper presents experimental studies on active flatness control of a membrane structure using genetic algorithm. A re-weighting algorithm is used which optimises both the flatness as well as the tension forces that are applied. The experiment also studies the effect of changing the mutation rate versus keeping it constant.

Chapter 3

Methodology

3.1 Force Analysis

The membranes that will be utilized for space application will be kept taut by the application of corner forces applied through a torres. The magnitude of each of these forces will depend on the shape of the membrane as well as other environment conditions faced in outer space like temperature variations, stress effects and other forces. All of these can lead to out of plane displacements of the membrane elements that can greatly hamper the efficiency of the membrane. A genetic algorithm based model is used to find the optimal value of these forces that would help to minimize these displacements. The root-mean-square value of the out of plane displacements of all the elements is taken as the objective function to be minimized.

$$d_{rms} = \sqrt{\frac{\sum_{i=1}^{N} d_i^2}{N}}$$

These d_{rms} values obtained from ABAQUS are the basis for calculating the fitness of an individual. The higher fitness scores(obtained by taking the reciprocal of d_{rms}) are then used to generate the next population.

3.1.1 Square Membrane

A square membrane is taken with in-plane forces applied at 16 different places. Of these, all the corner forces are applied across 21 points (the membrane is cut slightly from the corners and doesnot have sharp edges). Rest of the 12 forces are applied

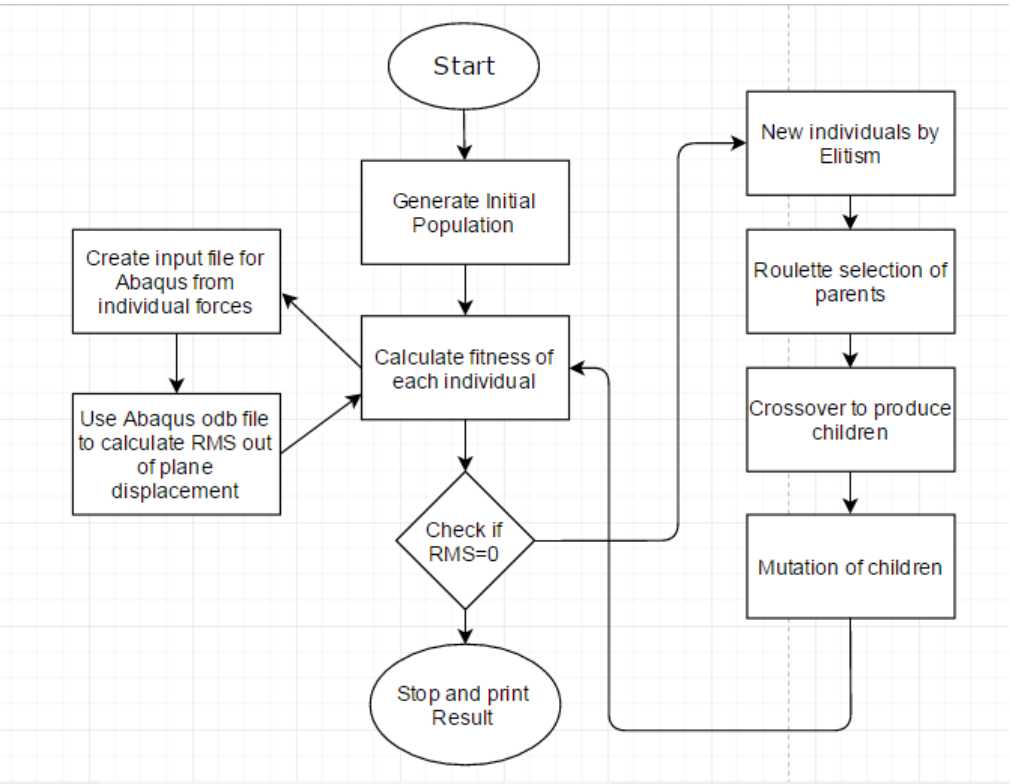


FIGURE 3.1: Flow-chart for minimising out-of-plane displacement of Membrane

on the boundary on a single point. These forces are both in the x and y axis and the maximum value of the resultant force is given in the table.

Genetic Algorithm logic

A binary Genetic algorithm based model is used where a particular force is either activated or deactivated. Activated forces are applied with full value. Thus a force is either applied or not depending on whether the respective value in the chromosome is 0 or 1.

The input file where force values are updated -

```
** LOADS
**
** Name: Load-1   Type: Concentrated force
*Cload
H1, 1, -0.01
** Name: Load-2   Type: Concentrated force
*Cload
H2, 2, -0.01
** Name: Load-3   Type: Concentrated force
*Cload
H3, 1, 0.01
** Name: Load-4   Type: Concentrated force
*Cload
H4, 2, 0.01
```

Maximum force values	
Load Numbers	Applied Force
H1 - H4	0.1 × 21
H5 - H8	1.0
H9 - H16	0.6

TABLE 3.1: Force values for square membrane

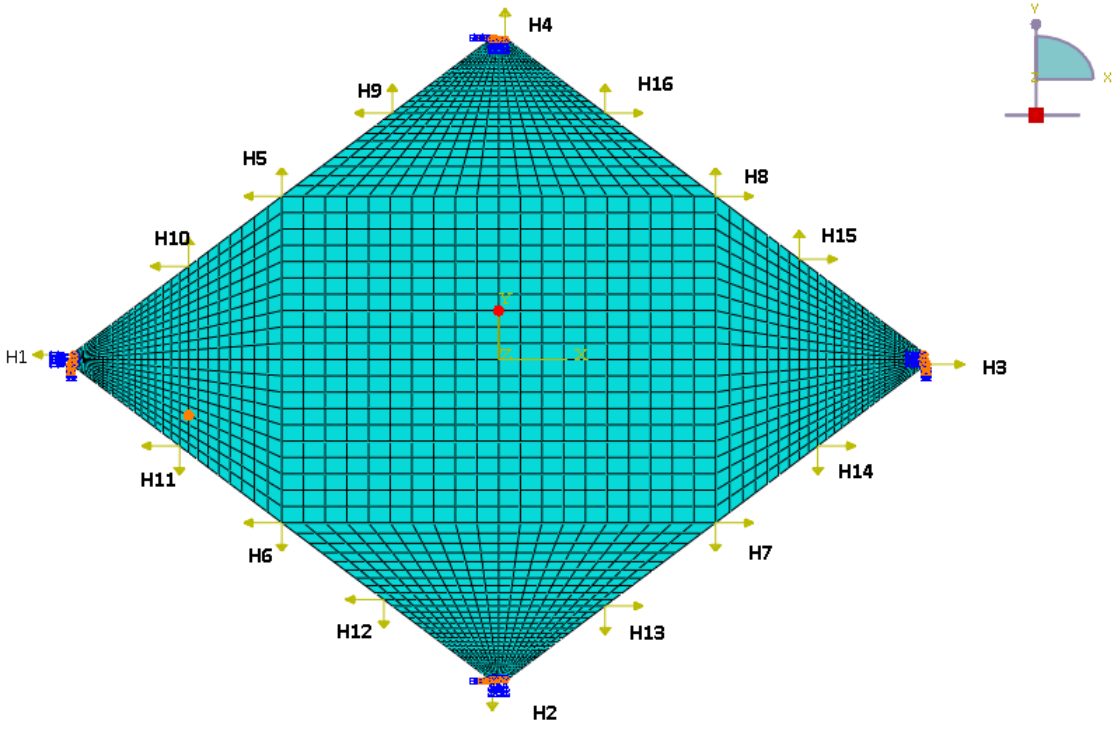


FIGURE 3.2: Membrane structure with location of 16 boundary forces

3.1.2 Structured Membrane

The following membrane more accurately depicts the membranes that would be sent in space. 36 point forces are applied. These forces are grouped in sets, where the force values across a given set is kept the same. This is done to keep symmetry and maintain the state of rest of the membrane.

Genetic Algorithm logic

Each chromosome consists of 5 elements depicting the values of the 5 sets. The values vary between 0 and 1 depending on how the population is generated. These values are then multiplied to the force on the loads to get the final values. This process is repeated till the maximum number of iterations are reached to get the minimum out-of-plane displacement.

The following code changes the input file to update the forces as per the individual generated. This input file is then fed to ABAQUS for calculating the displacement values.

```
load_set = {
    1:0, 2:1, 3:2, 4:3, 5:4, 6:3, 7:2, 8:1, 9:0,
    10:0, 11:1, 12:2, 13:3, 14:4, 15:3, 16:2, 17:1, 18:0,
    19:0, 20:1, 21:2, 22:3, 23:4, 24:3, 25:2, 26:1, 27:0,
    28:0, 29:1, 30:2, 31:3, 32:4, 33:3, 34:2, 35:1, 36:0,
}

for line in f:
                try:
                    if previous_line.startswith('*Cload'):
                        checker = True

                    if line.startswith('** Name') and checker == True:
                        checker = False
                        count+=1

                    if checker:
                        n_lin = line.split(',')
                        set_num = load_set[count]
                        po = n_lin[2].split('\r')
                        aber = float(chromo[set_num])*float(po[0])
                        line_to_add = n_lin[0]+','+n_lin[1]+', '+str(aber)+'\r\n'
                        new_input_file.write(line_to_add)
                    else:
                        new_input_file.write(line)
                    previous_line = line
```

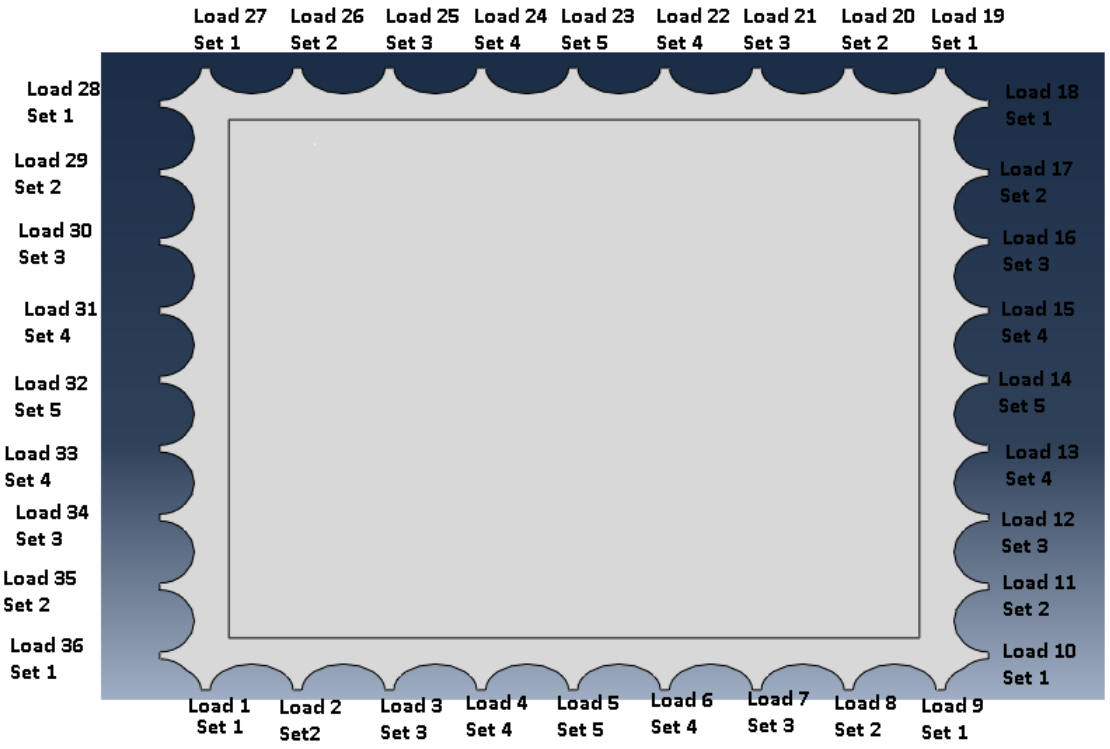


FIGURE 3.3: Membrane structure with 36 force values

Force Sets		
Load Numbers	Set Number	Maximum Force
L1,l9,L10,L18,L19,L27,L28,L36	1	25
L2,L8,L11,L17,L20,L26,L29,L35	2	25
L3,L7,L12,L16,L21,L25,L30,L34	3	25
L4,l6,L13,L15,L22,L24,L31,L33	4	25
L5,l4,L23,L32	5	25

TABLE 3.2: Load Sets for 36 force membrane

3.1.3 Temperature Variation

The thermal analysis done above gives the out-of-plane displacements for a range of temperatures. One such case where the rms value exceeds the permissible value is selected. Optmial forces are calculated to reduce this displacement on this model.

This process can be repeated for all the cases when displacement values cross a specific limit. This can give force distributions for various temperature conditions which can be fed in the system. This would enable the system to change the forces depending on the temperature distribution of the membrane and achieve minimum displacements for all the possible situations.

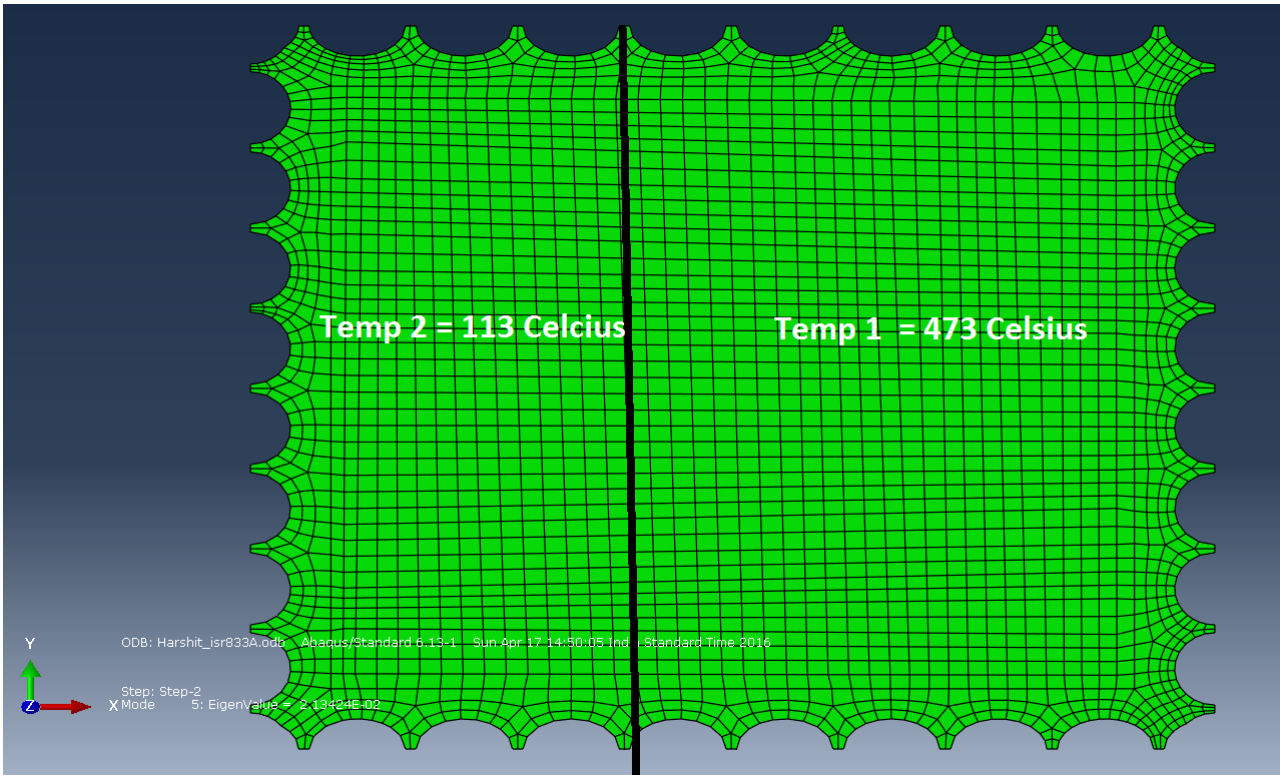


FIGURE 3.4: Membrane with different temperatures at different locations

For the given membrane, one section has been heated to 473 degrees Celsius whereas the other section is at 113 degrees. (A very common situation in space depending on which side is facing the Sun and which side is shielded from it)

3.2 Abaqus Model

For most of the case, a square membrane is taken for the ABAQUS model with same properties and different Boundary condition. To check for analysis, these boundary conditions were varied accordingly by the help of **Python Post-processing**. There are mainly two types of PostProcessing being done here:

1. Temperature Variation for thermal loading.
2. Load variation

Let us look in details at how this is done :

- First, a general model through ABAQUS is being created. Now, if we need to vary boundary condition manually, we'll end up wasting a lot of human time in waiting for each simulation to run and then change the conditions and run again. This problem can be solved by automating the process.
- The model created previously will generate a **input file**. This file contains all the information regarding the model from material properties to the meshing to loading. For example the part of input filee with temperature loading for specific model. Let's say this file be isro_1.inp

```
** PREDEFINED FIELDS
**
** Name: Predefined Field-2   Type: Temperature
*Temperature
Set-7, 400.
** Name: Predefined Field-3   Type: Temperature
*Temperature
Set-8, 100.
```

- Now if somehow, we change this file and run it again via `abaqus j=isro_1.inp int cpus=4` will run an ABAQUS process and create an ODB(output database) file.
- The generated odb file has all the data for new model. We can now extract the displacement values from the odb file which has been done in these snippets.

```
def return_rms(self, def_path):
    """
    Same as created last year with monor modifications to handle errors.
    Used to generate rms value from the data and show them in a text
    file.
    """
    rms_value = 0
    odb = visualization.openOdb(path=odbPath)
    try:
        lastFrame = odb.steps['Step-2'].frames[1]
        displacement = lastFrame.fieldOutputs['U']

        #load = odb.rootAssembly.nodeSets['LOAD']

        iteration_count=1
        #rms_values=[]

        endDisplacement = displacement.getSubset()
        workbook = xlsxwriter.Workbook(def_path+'.xlsx')
        worksheet = workbook.add_worksheet()

        row=0
        rms=0
        for v in endDisplacement.values:
            disp=v.data[2]
            worksheet.write(row,0,float(v.nodeLabel))
            worksheet.write(row,1,float(disp))
            rms=rms+pow(disp,2)
            row=row+1

        rms_value=math.sqrt(rms/row)
```

- The above takes data from odb and saves the rms in an excel file.
- We can also create field reports from the odb file which can be used to analyse data. There is already an inbuilt function in ABAQUS API for this.

```
def writefieldreport(self, def_path):
    """
    Function to write field reports for specific obd files in a text file.
    """

    # Take the file name of input file as we expect odb to be made from the same file
    odbPath = def_path+".odb"
    o1 = session.openOdb(name=odbPath)
    session.viewports['Viewport: 1'].setValues(displayedObject=o1)
    odb = session.odbs[odbPath]

    # Now write filed report in a spearate text file
    report_file_name = '%s_report.txt'%def_path
    try:
        session.writeFieldReport(fileName=report_file_name, append=ON,
                                 sortItem='Node Label', odb=odb, step=1,
                                 frame=1, outputPosition=NODAL,
                                 variable=(('U', NODAL), ('UR', NODAL), ('E',
                                 INTEGRATION_POINT), ('S',
                                 INTEGRATION_POINT), ('SENER',
                                 INTEGRATION_POINT), ))
    except Exception:
        pass
```

- Now, in Analysis and chapter result, we'll see how we use these functions to get varied data.

3.3 Matlab Model

3.3.1 Out of plane displacement

A square membrane is taken and meshed using square elements. The position of each node and member nodes of each element are calculated.

```
for i=1:NXE
    for j=1:NYE

        a_x=(i-1)*deltax; a_y=(j-1)*deltay;
        b_x=a_x; b_y=a_y+deltay;
        c_x=a_x+deltax; c_y=a_y+deltay;
        d_x=a_x+deltax; d_y=a_y;

        num_a=(i-1)*(NYE+1)+j;
        num_b=num_a+1;
        num_d=i*(NYE+1)+j;
        num_c=num_d+1;

        points(num_a,:)=[a_x a_y];
        points(num_b,:)=[b_x b_y];
        points(num_c,:)=[c_x c_y];
        points(num_d,:)=[d_x d_y];

        edges(counter,:)=[num_a num_b num_c num_d];
        counter = counter+1;
    end
end
```

The global force matrix is created with forces applied along the 4 corner points.

```
load=zeros(total_DOF,1);

load(1) = -1*F_x;
load(2) = -1*F_y;
load(NXE*DOF_node+1) = F_x;
load(NXE*DOF_node+2) = -1*F_y;
load((total_nodes-NYE-1)*DOF_node+1) = -1*F_x;
load((total_nodes-NYE-1)*DOF_node+2) = -1*F_y;
load((total_nodes-1)*DOF_node+1) = F_x;
load((total_nodes-1)*DOF_node+2) = F_y;
```

For a particular element, the shape function and its derivatives are calculated which are then used to calculate the Jacobian matrix.

$$[.\psi] = \begin{bmatrix} \psi_1 & 0 & \psi_2 & 0 & \psi_3 & 0 & \psi_4 & 0 \\ 0 & \psi_1 & 0 & \psi_2 & 0 & \psi_3 & 0 & \psi_4 \end{bmatrix}.$$

The local stiffness matrix is calculated which is then used to populate the Global stiffness matrix. Once the global stiffness matrix is completed, displacements are calculated -

$$disp = GS \setminus load$$

Once the displacements of all the nodes are known, different approaches are applied depending on whether we wish to find the mode shapes or wrinkles in the membrane.

3.3.2 Vibration Analysis

The first three mode shapes are calculated for the pre-tensioned membrane. The eigen values are calculated which are then used to find the natural frequencies by using the following equation -

$$det(MM^{-1}GS - \lambda I) = 0$$

3.3.3 Wrinkle Formation

The strains are calculated using the formula given below

$$\left\{ \begin{array}{c} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{array} \right\} = \left\{ \begin{array}{r} du \setminus dx \\ dv \setminus dy \\ du \setminus dy + dv \setminus dx\emptyset \end{array} \right\}.$$

The stress values are calculated from the strain values by multiplying with the **D matrix** where D is given by -

$$D = \frac{E}{1-\mu^2} \begin{bmatrix} 1 & \mu & 0 \\ \mu & 1 & 0 \\ 0 & 0 & \frac{1-\mu}{2} \end{bmatrix}.$$

Once the stress and strain value are known, the location of points where wrinkles will be formed are found by using the Wrinkle Formation Criteria as discussed in 1.4.2. The graphs for the stress-strain distributions as well as wrinkle formation can be found in section 4.3.

Chapter 4

Experiment

An experimental investigation of the vibration behaviour of a corner-loaded square wrinkled membrane in air is described in this section. Detailed specification of devices used in the experimentation is given. This section also includes the procedure and method used in experimentation.

4.1 Objective

1. Monitor real time vibration of square flat membrane.
2. Validate out-of-plane displacement value for PZT and PVDF actuated membrane

4.2 Equipments

Following equipments are used in the experimental work :

1. Piezoelectric sensors and piezoelectric actuators (PZT patches)
2. Square Flat Kapton Membrane
3. Vibrometer
4. Connector cables
5. Power Supply 240V, 50Hz

Specifications of equipments	
Membrane material	Kapton
Area	200mm*200mm
Thickness	0.25mm
Voltage Applied	10V(sine wave)
Weights on end	200gm
Diameter of PZT	20mm

TABLE 4.1: Different membrane materials and their properties

4.3 Experimental Setup

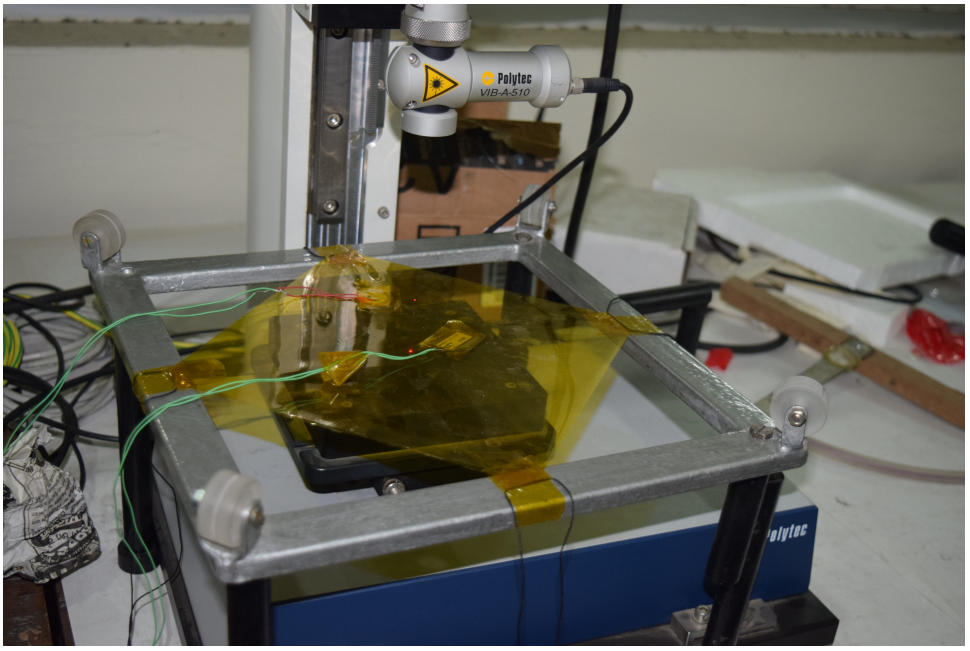


FIGURE 4.1: Experiment setup with PZT and PVDF attached

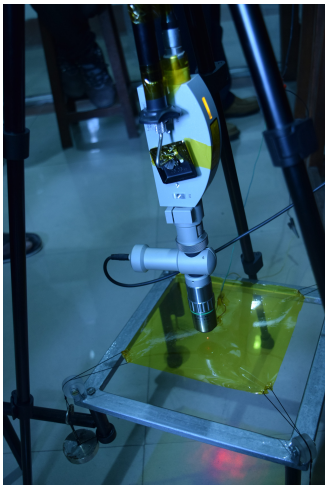


FIGURE 4.2: Experimental setup without actuators

4.3.1 PiezoElectric Material

Smart material used in the experiment is PZT patch made by the Sparkler Ceramics Pvt Ltd. A PZT patch is used to provide excitation to the membrane. The PZT is mounted on the center of Membrane. Before PZT patch is bonded to the membrane, terminals are soldered. Patch is bonded to the structure using good epoxy. When the membrane is excited with signal generator, the vibration of membrane starts and the sensor when deformed mechanically develops the voltage across it terminals.

4.3.2 PVDF Material

FIGURE 4.3: SunRobotics PVDF used for experiments

The PVDF used for the experiment is manufactured by SunRobotics Technologies and can be used for flex, touch, vibration and shock measurements. A small AC and large voltage (up to +/-90V) can be used to move the film back and forth. One of the major benefits of this sensor is that larger input voltages can be given as compared to the PZT used.

4.3.3 Vibrometer

Vibrometer is a scientific instrument used to make non contact measurements of a surface. The laser beam from the vibrometer is directed at the surface of interest, and the vibration amplitude and frequency are extracted from the Doppler shift of the reflected laser beam frequency due to the motion of the surface.

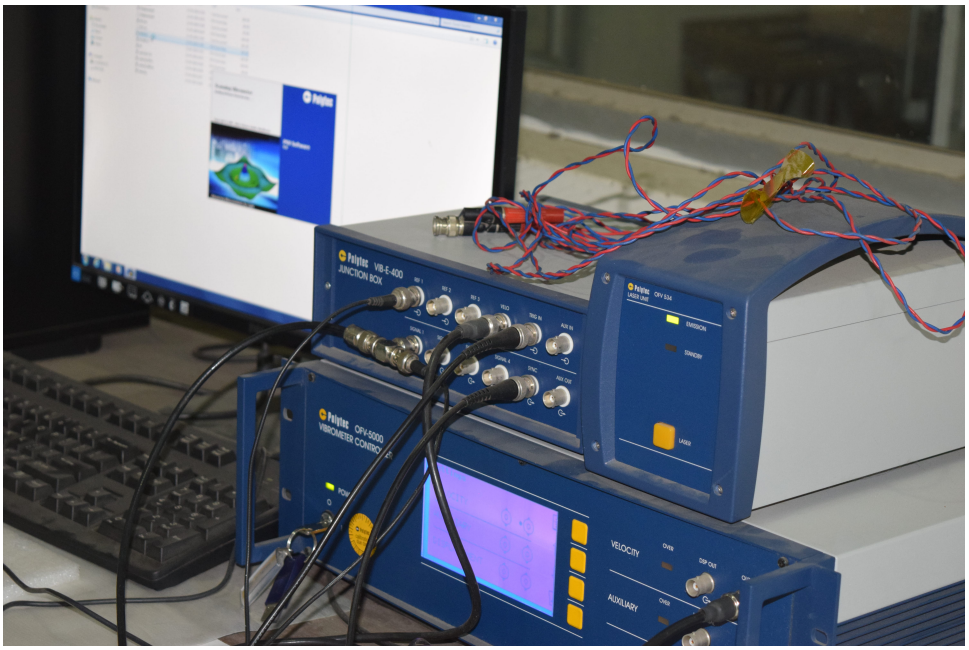


FIGURE 4.4: Vibrometer measuring out of plane displacement

4.4 Experimental Values

The displacement and velocity were recorded at 50, 500 and 5000 Hz. An FFT was performed on these values to obtain the peaks at various frequencies. These were then validated using the ABAQUS model for the same.

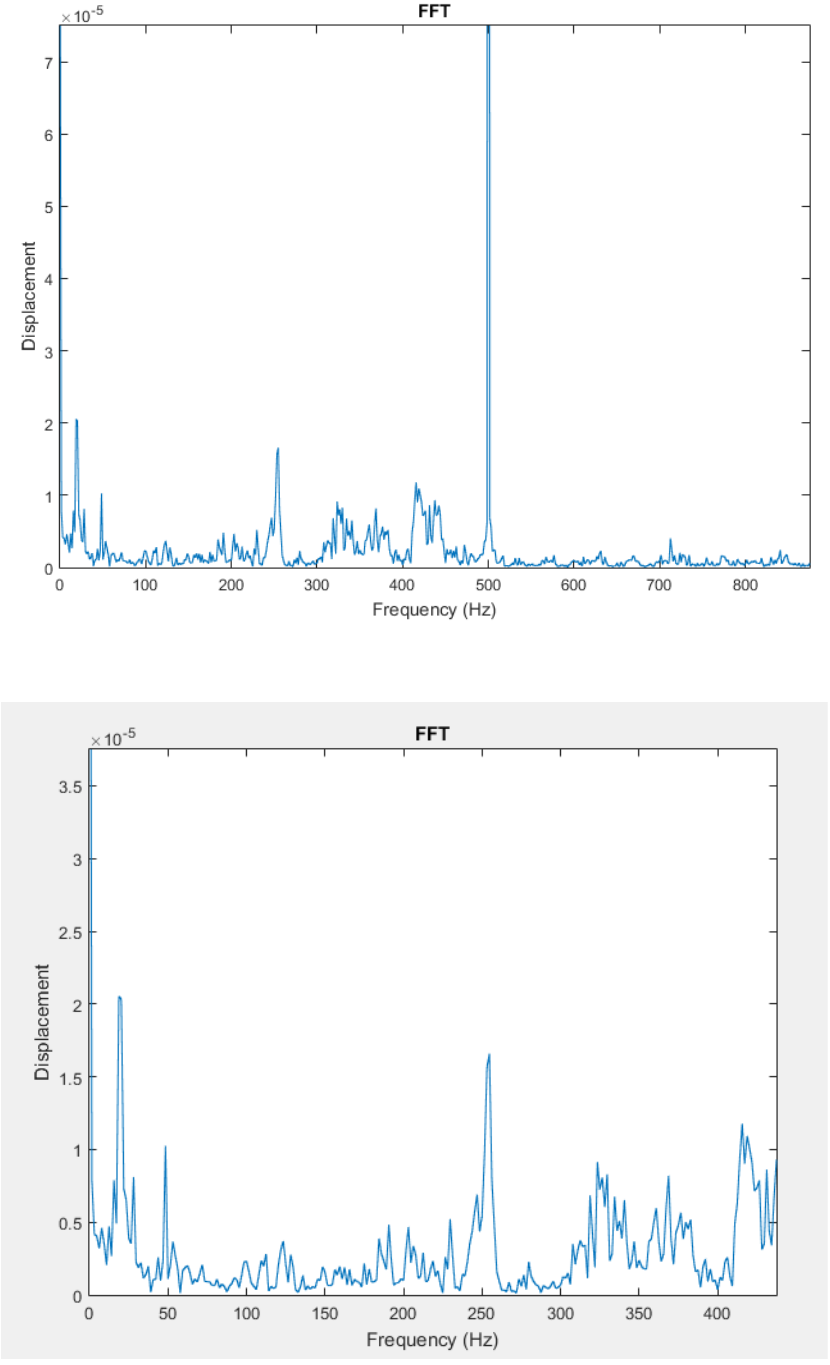


FIGURE 4.5: FFT graphs pf displacement at 500Hz excitation

Frequency Peaks : 22, 48, 256

4.5 Validation

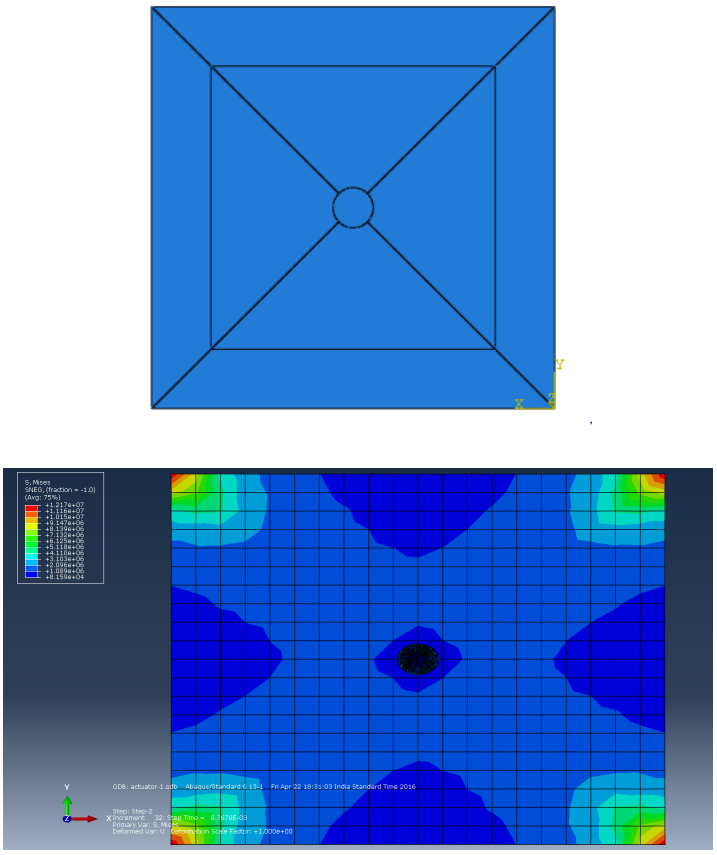


FIGURE 4.6: ABAQUS model with PZT attached at center

Frequency Peaks : 24.7, 54.3

The natural frequencies as received from the ABAQUS model are close to the ones as obtained from experiment. Hence the results are validated for an 500Hz sinusoidal excitation.

Chapter 5

Analysis and Results

5.1 ABAQUS Post-Processing Analysis

5.1.1 Temperature Variation on whole surface

We created an Abaqus model and vary temperature across all its surface. Temperature range is from -200 to 200 degree celcius.

Input file snippet where we'll vary temperature

```
** Name: Predefined Field-2 Type: Temperature
*Temperature
Set-7, 400.
```

Python API snippets to create multiple input file

```
# Open the original input file and start readling lines from it
    with open(input_file_to_change, 'r') as f:
        previous_line = ""
        for line in f:
        if previous_line=="*Temperature\n" and line.startswith('Set-7'):
            line = 'Set-7, %s.\n'%new_temperature
        # Now write the new lines to the new input file
        new_input_file.write(line)
        previous_line = line
```

```
temp_range = range(-200, 210, 10)
 no_of_files = len(temp_range)
input_file_to_change = new_input_file_name
```

Result for temperature variation

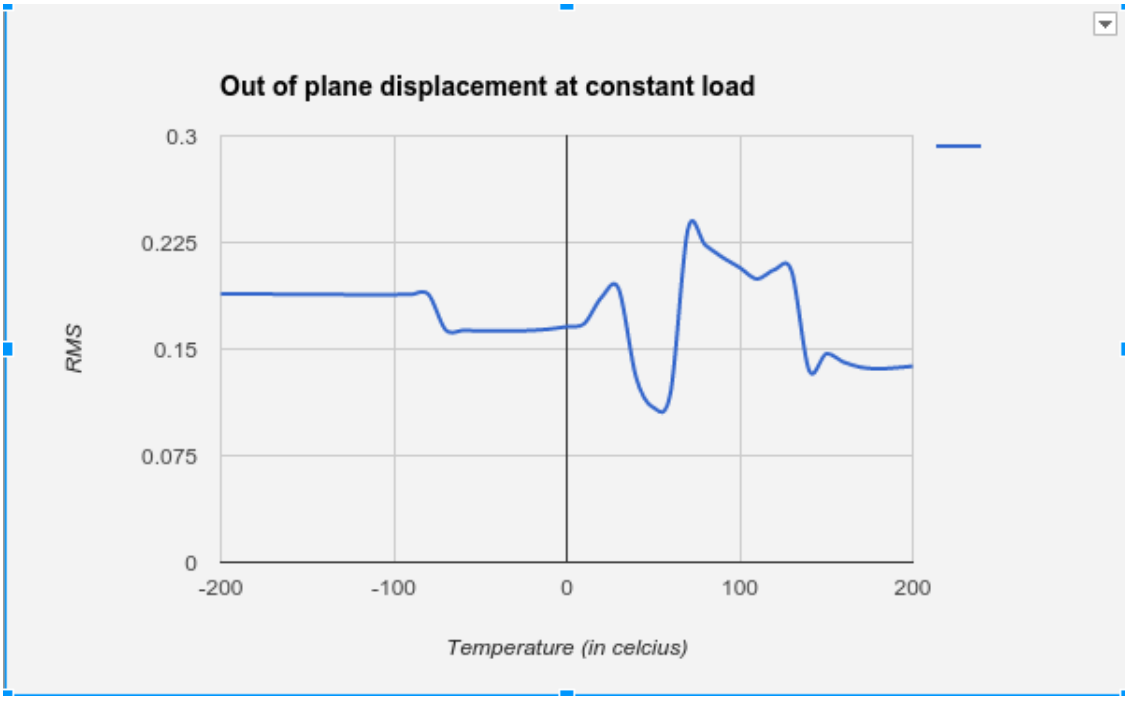


FIGURE 5.1: Out of plane displacement for temp variation across surface

5.1.2 Temperature and Load variation together

We created an ABAQUS model and give boundary condition for both temperature as well as load variation. So, now we get the result for temperature at different loads.

Input File snippet where we vary temp and load

```
** Name: BC-4 Type: Displacement/Rotation
*Boundary, op=NEW, load case=1
Set-4, 1, 1
Set-4, 2, 2, 2.
Set-4, 3, 3
Set-4, 4, 4
Set-4, 5, 5
Set-4, 6, 6
*Boundary, op=NEW, load case=2
Set-4, 1, 1
Set-4, 2, 2, 2.
Set-4, 3, 3
Set-4, 4, 4
Set-4, 5, 5
Set-4, 6, 6
**
** PREDEFINED FIELDS
**
** Name: Predefined Field-2   Type: Temperature
*Temperature
Set-7, 400.
```

Python API snippets to create multiple input file

```
def create_input_file_load (self, load_range, no_of_files =1):
    """
    Create input files for load variation for different loading
    """

    self.temp_file_number = 0
    file_number = 0

    for new_load in load_range:

        # Create new input file name and append it to list
        self.load_file_number = load_range.index(new_load)+1
        new_input_file_name = 'input_'+'load_'+str(self.load_file_number)+'_temper_'+str(self.temp_file_number)+'.inp'

        # Create and open new input file for the new load
        new_input_file = open(new_input_file_name, 'w')

        # Open original input file and start reading lines from it
        with open(self.input_file, 'r') as f:
            lines_to_change_with_positive_load = [
                        'Set-3, 1, 1, 2',
                        'Set-4, 2, 2, 2'
            ]

            lines_to_change_with_negative_load = [
                        'Set-1, 1, 1, -2',
                        'Set-2, 2, 2, -2'
            ]

            for line in f:
                if line.startswith(tuple(lines_to_change_with_positive_load)):
                    line = line[:12]+'  '+str(new_load)+'.\n'
                if line.startswith(tuple(lines_to_change_with_negative_load)):
                    line = line[:12]+'  '+str(new_load*(-1))+'.\n'
                new_input_file.write(line)

        new_input_file.close()
        temp_range = range(-200, 210, 10)
```

```
no_of_files  = len(temp_range)
input_file_to_change  = new_input_file_name
self . create_input_file_temp (temp_range, self . load_file_number ,  input_file_to_change ,  no_of_files )
```

Result for load as well as temperature variation

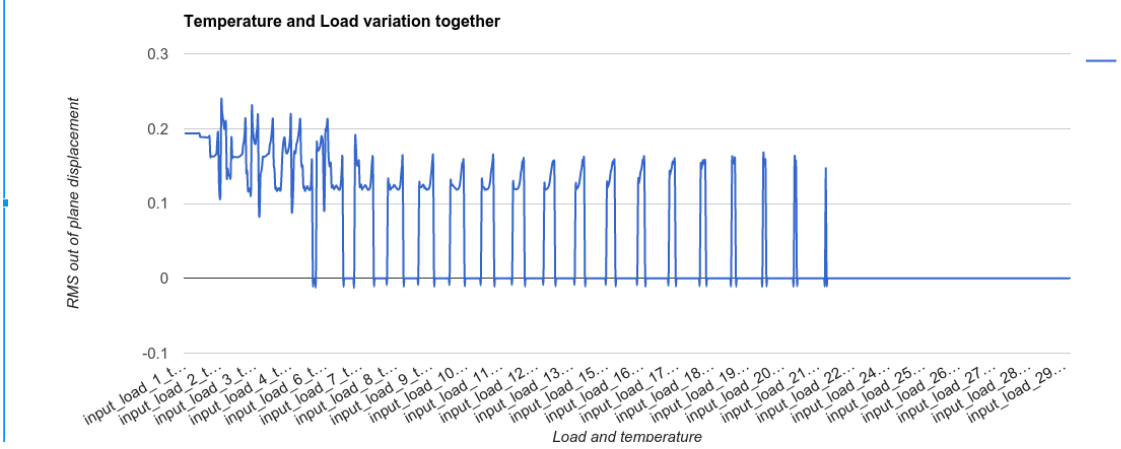


FIGURE 5.2: Out of plane displacement for temp and load variation across surface

5.1.3 Temperature variation in parts

Now, we divide membrane surface in 2 parts and then vary the temperature of both parts independent of each other.

Input file where we vary temperature for both parts

```
** PREDEFINED FIELDS
**
** Name: Predefined Field-1    Type: Temperature
*Temperature
temp_1, 200.
** Name: Predefined Field-2    Type: Temperature
*Temperature
temp_2, -200.
```

Python API snippets to create multiple input file

```
with open(self. input_file , 'r') as f:
    previous_line = ""
      for line in f:
        if previous_line=="*Temperature\n" and line.startswith('temp_1'):
          line = 'temp_1, %s.\n'%temp_1
        if previous_line=="*Temperature\n" and line.startswith('temp_2'):
          line = 'temp_2, %s.\n'%temp_2
        # Now write the new lines to the new input file
        new_input_file.write(line)
        previous_line = line
```

Result for temperature variation across both parts

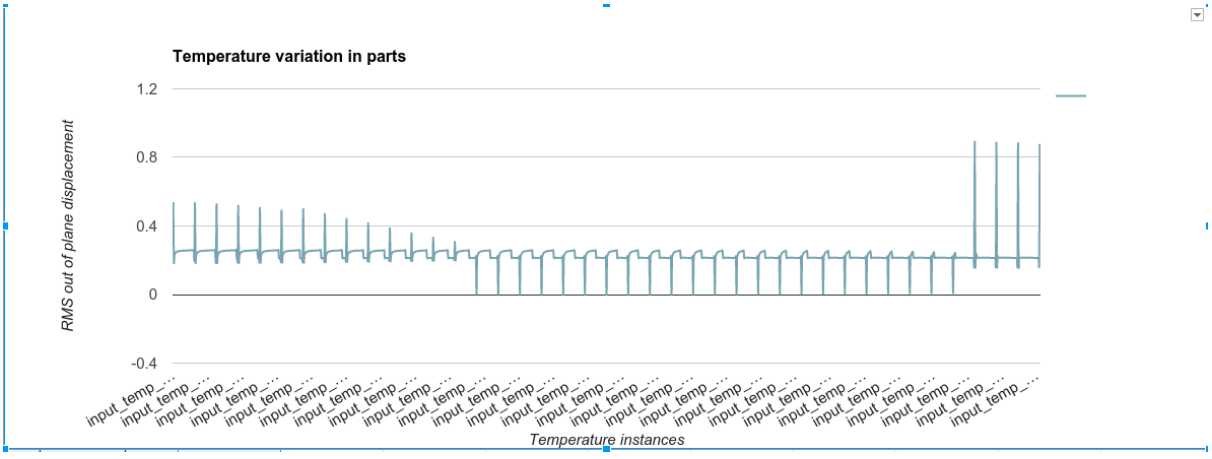


FIGURE 5.3: Out of plane displacement for temperature variation across 2 parts on a surface

5.1.4 Temperature variation on surface divided in parts

Now we divide the surface in 5 parts, cause in space we know that it might be possible that only 20% of surface faces sun and other opposite side. So the temperature would be different on both parts. The divided parts would be :

- 20 - 80
- 40 - 60
- 60 - 40
- 80 - 20

Input file where we vary temperature for both parts Same as the previous one

Python API snippets to create multiple input file

```
def create_input_file (self, no_of_files =1):
    """
    Number of input files created will depend upon the variation of
    different loading as well as change in input conditions and separate
    file will be made for each ones which would be later on run will
    different inputs.
    """
    # List to save the names of new input files created
    self.new_input_file_list = []

    temp_division_list = ['temp_1', 'temp_2', 'temp_3', 'temp_4',
    'temp_5']

    temp_list = [ (temp_division_list [:x],
    temp_division_list [x:len(temp_division_list )]) for x in range(1,
    len(temp_division_list )) ]

    temp_range = range(73, 483, 10)
    no_of_files = len(temp_range)

    # Loop across same range to get two temperature variations
    for temp_1 in temp_range:
        for temp_2 in temp_range:

            temp_dict = {}
            for l in temp_list:
                for q in l[0]:
                    temp_dict[q] = temp_1
                for r in l[1]:
                    temp_dict[r] = temp_2

                # Create name according to temp_1 with temp_2 fixed at
                initial value
                new_input_file_name = 'input_'+str(temp_dict['temp_1'])+'_'+str(temp_dict['te
                mp_2'])+'_'+str(temp_dict['temp_3'])+'_'+str(temp_dict[
                'temp_4'])+'_'+str(temp_dict['temp_5'])+'.inp'

                # Create new input file for this particular temperature
                new_input_file = open(new_input_file_name,'w')

                # Open the original input file and start readling lines from it
                with open(self.input_file , 'r') as f:
```

```
        previous_line = ""
        for line in f:
            if previous_line=="*Temperature\n" and line.startswith('temp_1'):
                line = 'temp_1, %s.\n'%temp_dict['temp_1']
            if previous_line=="*Temperature\n" and line.startswith('temp_2'):
                line = 'temp_2, %s.\n'%temp_dict['temp_2']
            if previous_line=="*Temperature\n" and line.startswith('temp_3'):
                line = 'temp_3, %s.\n'%temp_dict['temp_3']
            if previous_line=="*Temperature\n" and line.startswith('temp_4'):
                line = 'temp_4, %s.\n'%temp_dict['temp_4']
            if previous_line=="*Temperature\n" and line.startswith('temp_5'):
                line = 'temp_5, %s.\n'%temp_dict['temp_5']
            # Now write the new lines to the new input file
            new_input_file.write(line)
            previous_line = line

    # Finally close the new input file created after updating temperature
    new_input_file.close()


    self. new_input_file_list .append(new_input_file_name)
```

Results for different temperature range for Part V

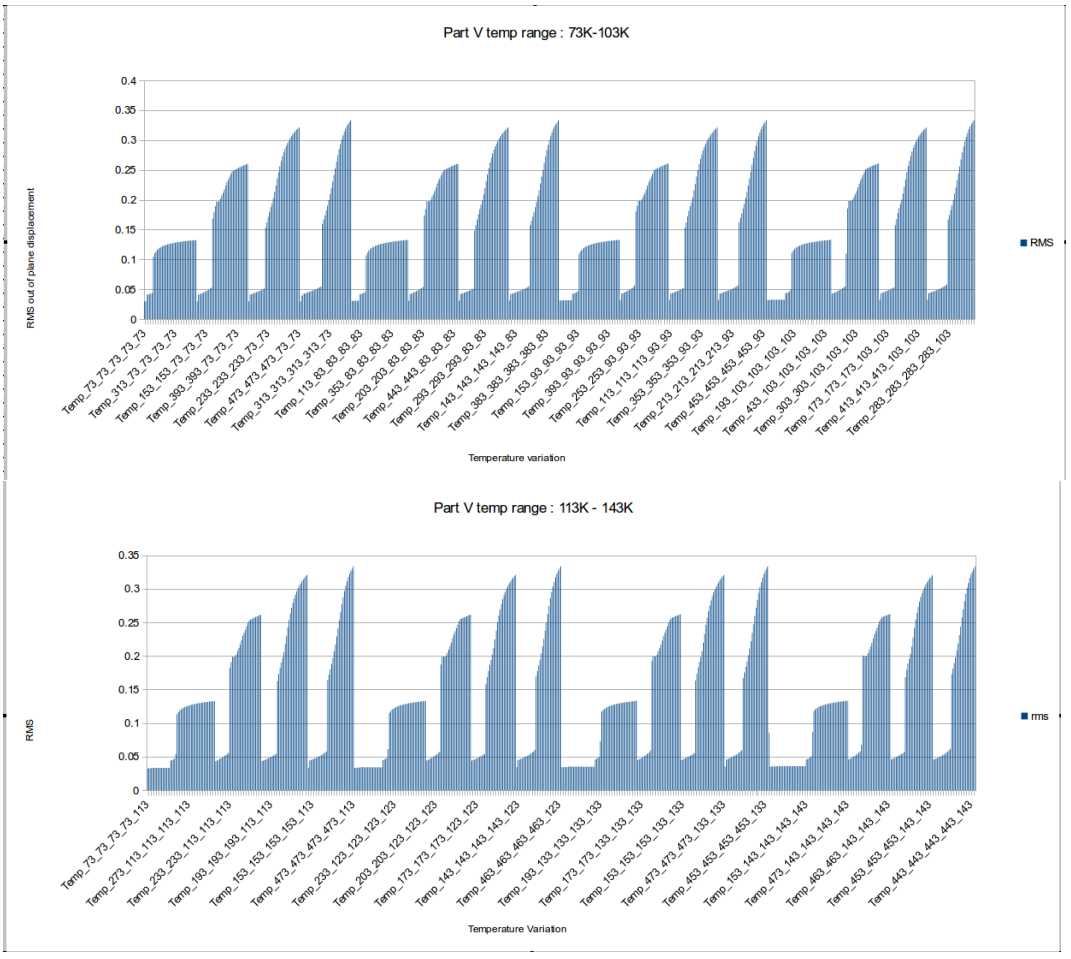


FIGURE 5.4: Temp range from 73K to 153K

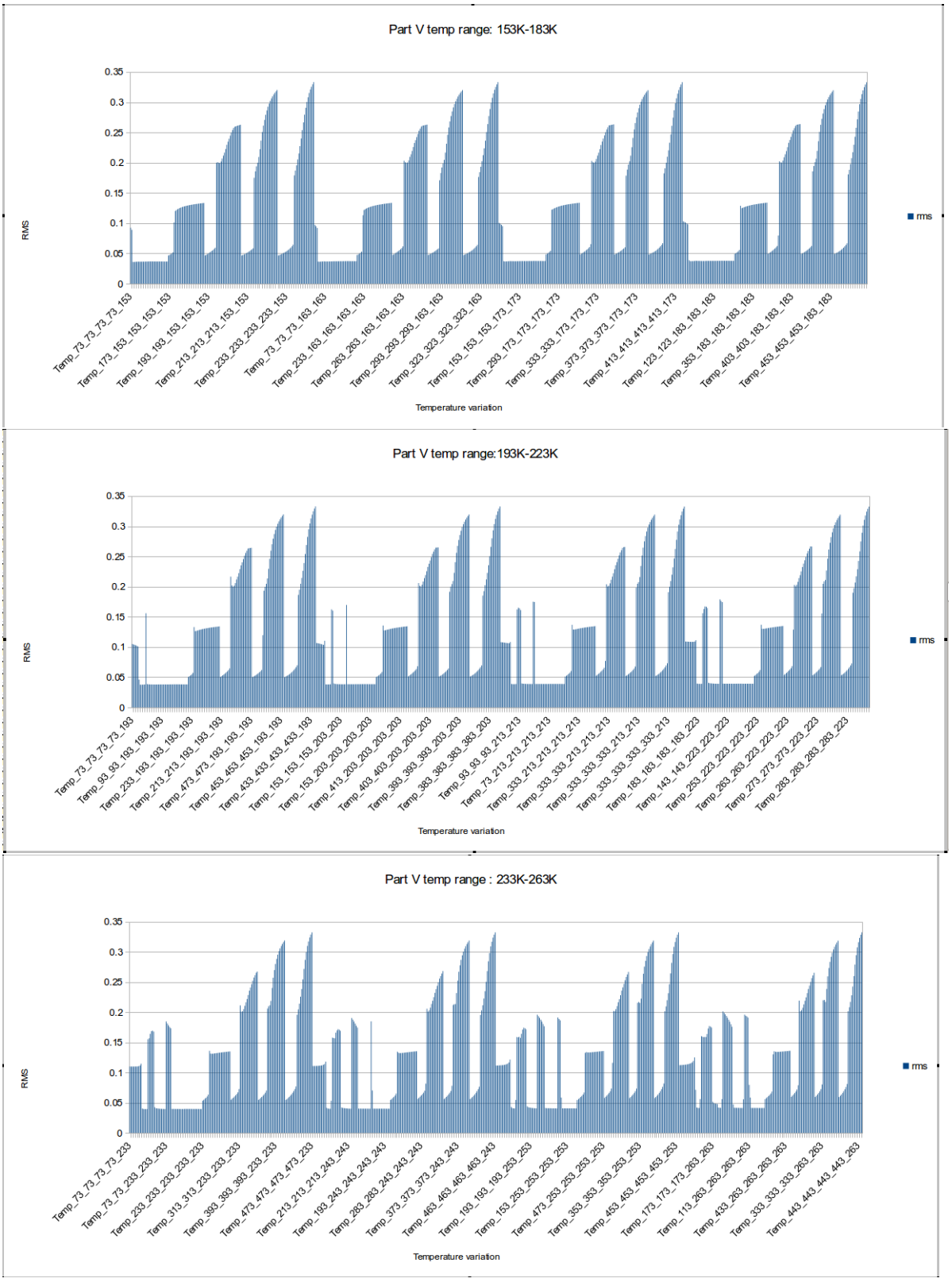


FIGURE 5.5: Temp range from 163K to 273K

FIGURE 5.6: Temp range from 283K to 393K

FIGURE 5.7: Temp range from 393K to 473K

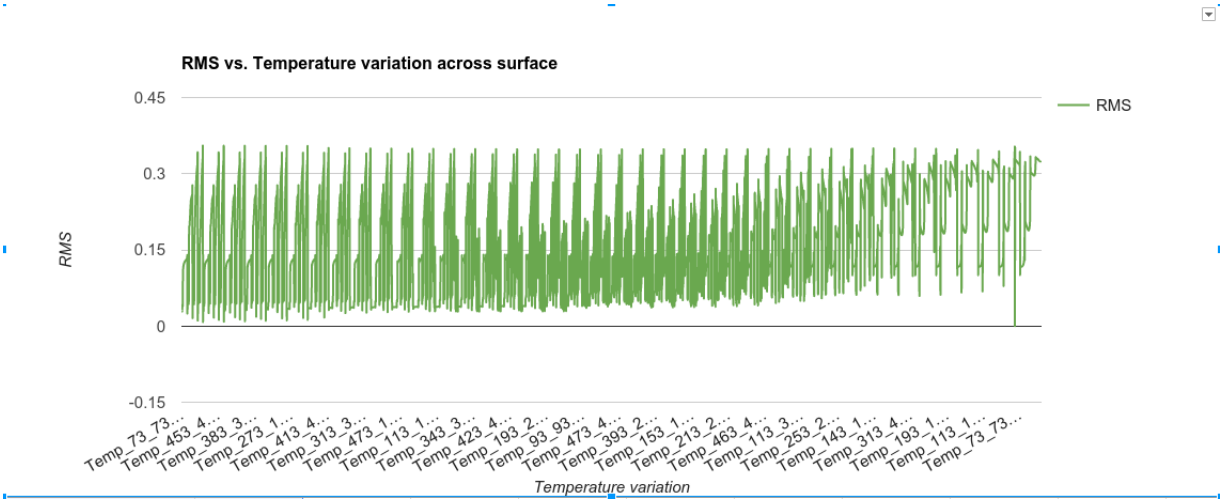


FIGURE 5.8: Temp range acorss 73K - 473K

5.2 Force Analysis

5.2.1 Square Membrane

The minimum value of out-of-plane displacement that is obtained is - **0.0263504173836**
The chrome(activation scheme of the forces) corresponding to this value - **[1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0]**

Observation - The minimum value is not obtained when all the forces are activated. The out-of-plane displacement in that case is actually almost 10% higher than the optimal value.

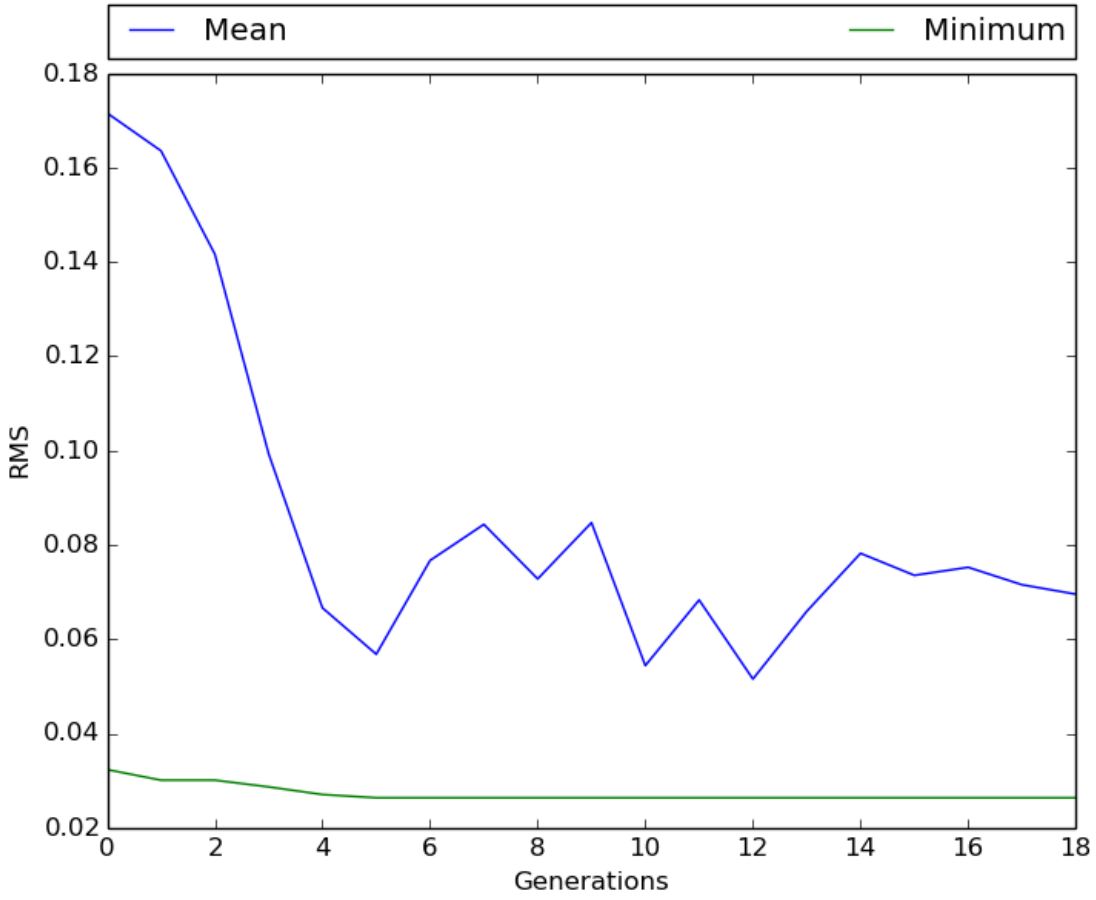


FIGURE 5.9: Average rms values of each generation of a square membrane under 16 loads

5.2.2 Structured Membrane

The minimum value of out-of-plane displacement that is obtained is - **0.000117675922656**
The chrome(activation scheme of the forces) corresponding to this value - **[0.6416015, 0.4052734, 0.880859, 0.986328, 0.75292968]**

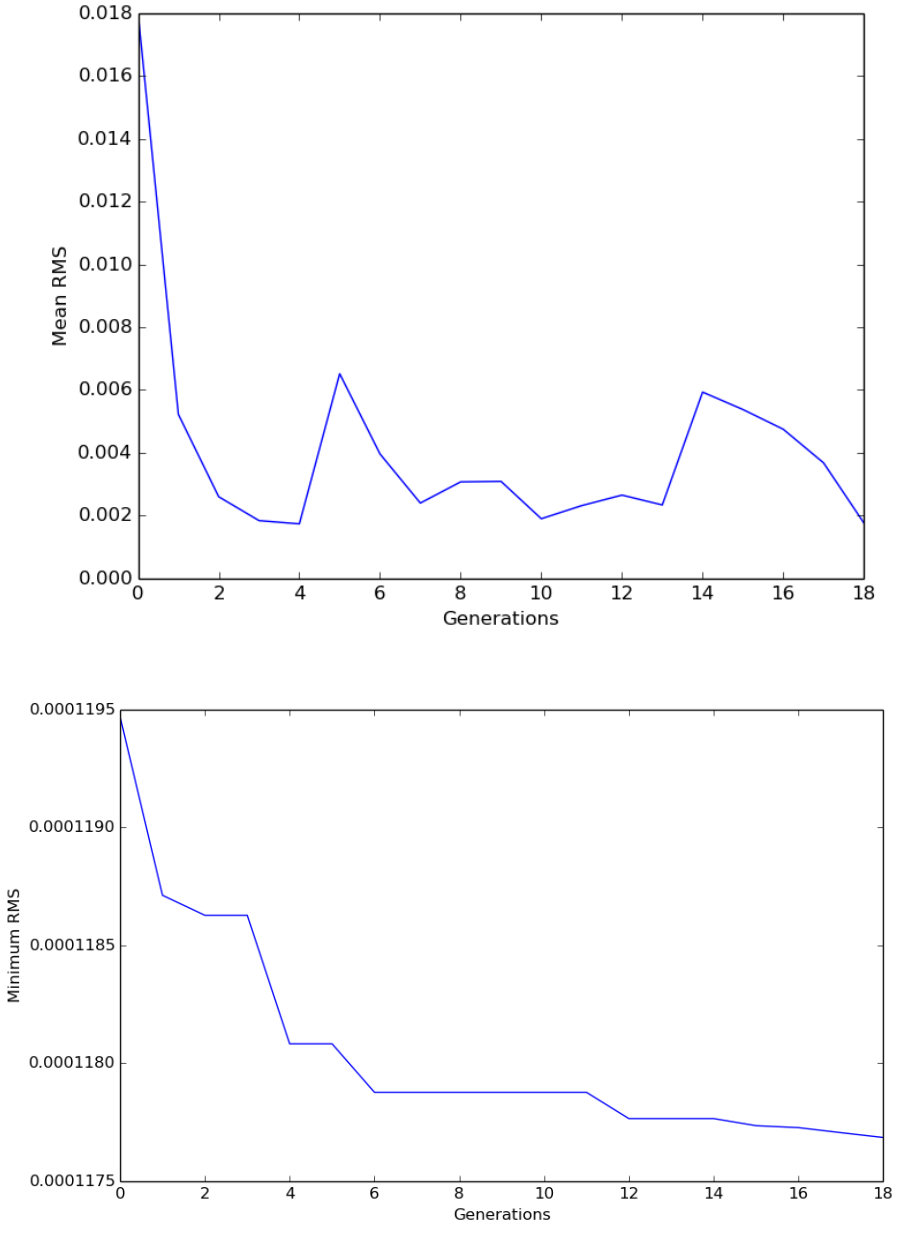
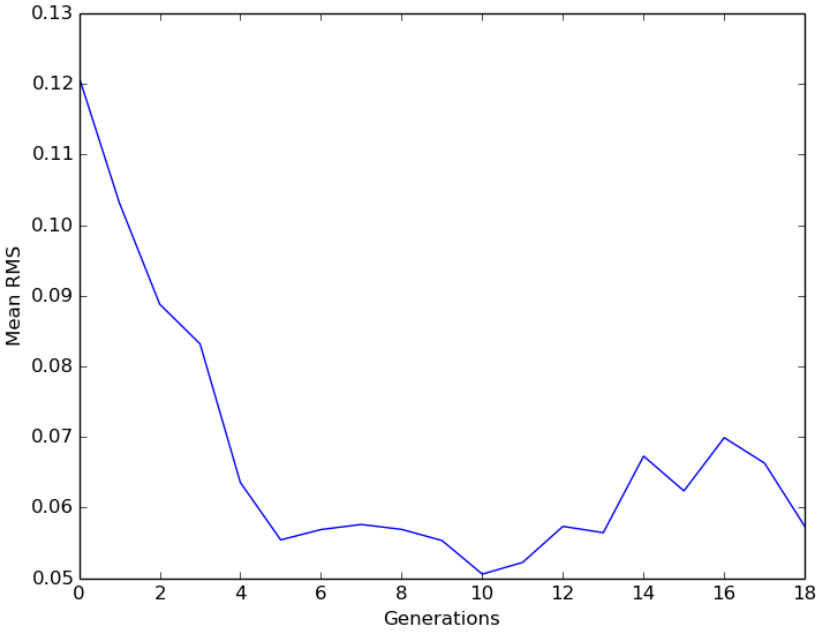


FIGURE 5.10: Average and Minimum rms values of each generation of a Membrane under 36 loads

5.2.3 Temperture Variation for model with higher RMS

The minimum value of out-of-plane displacement that is obtained is - **0.0405163469722**
The chrome(activation scheme of the forces) corresponding to this value - **[0.9912109, 0.603515, 0.5185546, 0.852539, 0.3476562]**



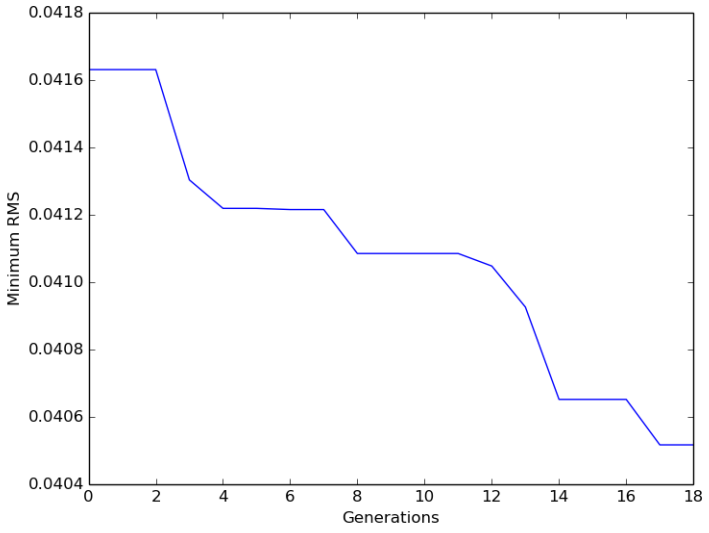


FIGURE 5.11: Average and Minimum rms values of each generation of a Membrane with temperature variation

5.3 Vibration Analysis

5.3.1 Mode Shapes

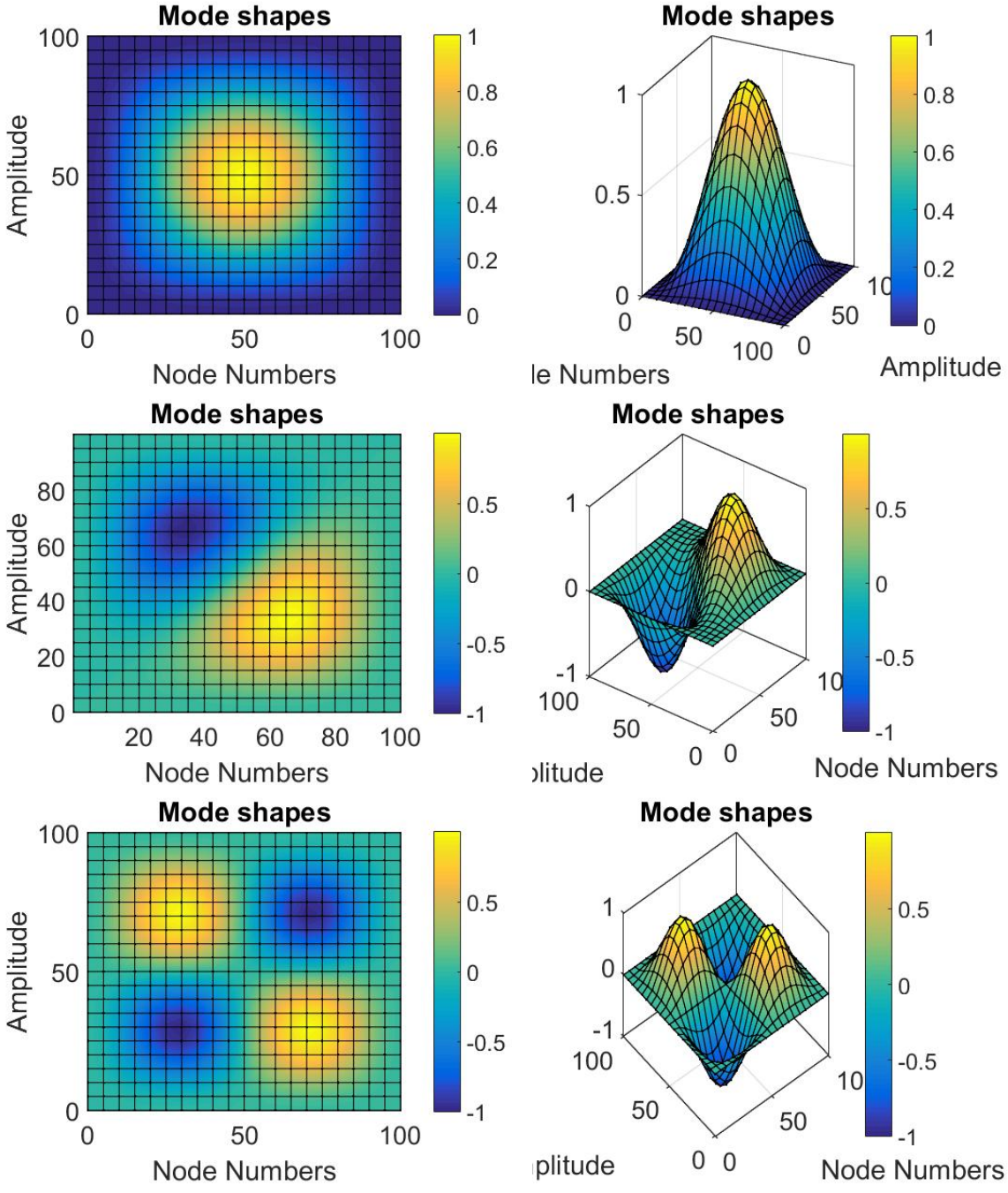


FIGURE 5.12: Mode shapes for the first,second and fourth natural frequencies

5.3.2 Wrinkle Analysis

The **Stresses** as developed in the membrane are shown below.

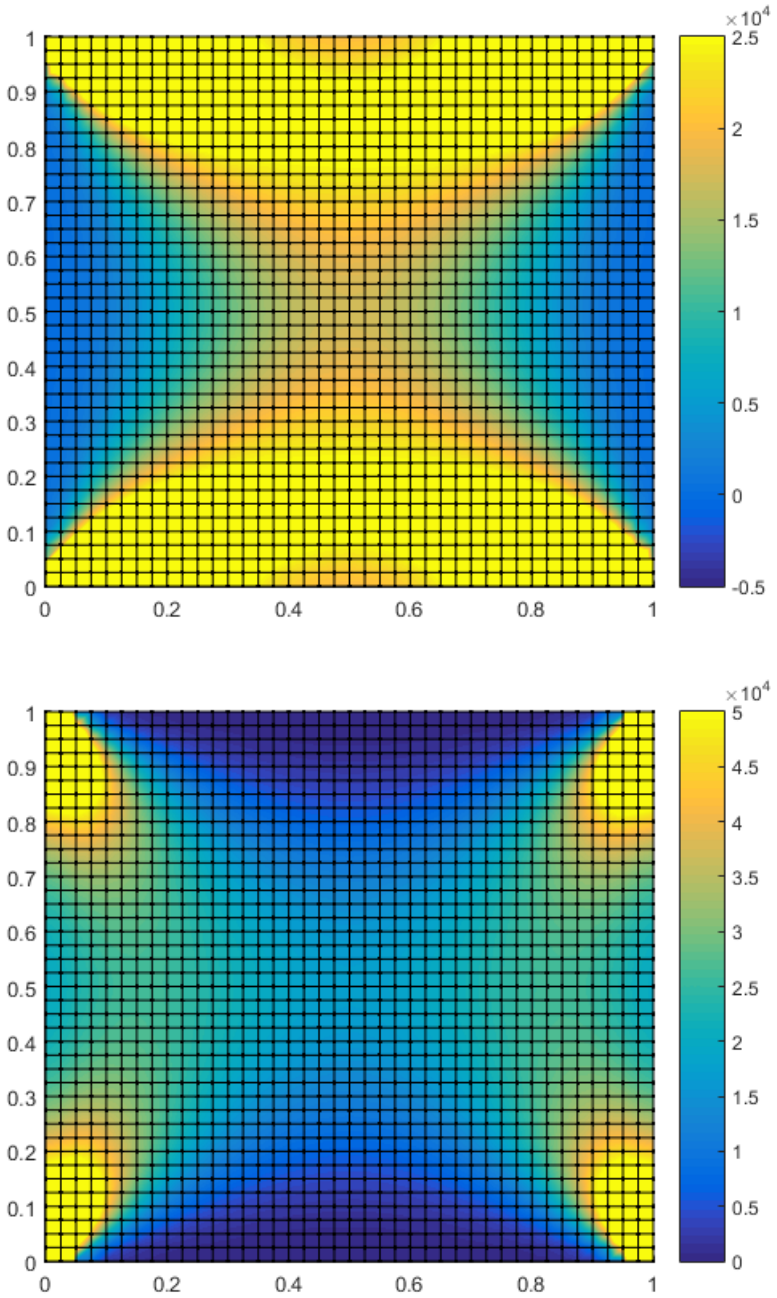


FIGURE 5.13: Stresses developed in a corner loaded Membrane with the original nodal point locations

The application of forces stretches the membrane causing the nodal points to shift from their location in the 2-D plane. These coordinates are found by adding the nodal displacements to the original point coordinates.

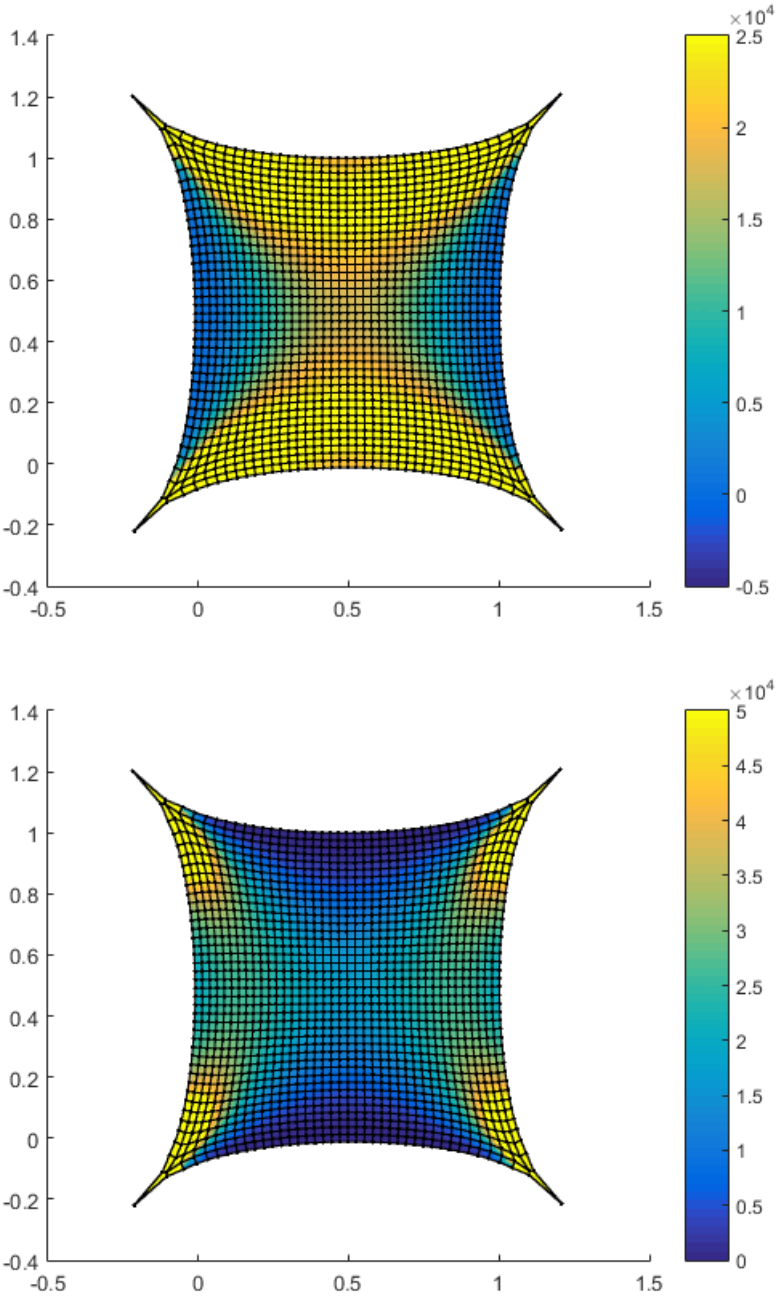


FIGURE 5.14: Stresses developed in a corner loaded Membrane by using the new nodal point locations

The location of the nodal points where wrinkles will be formed.

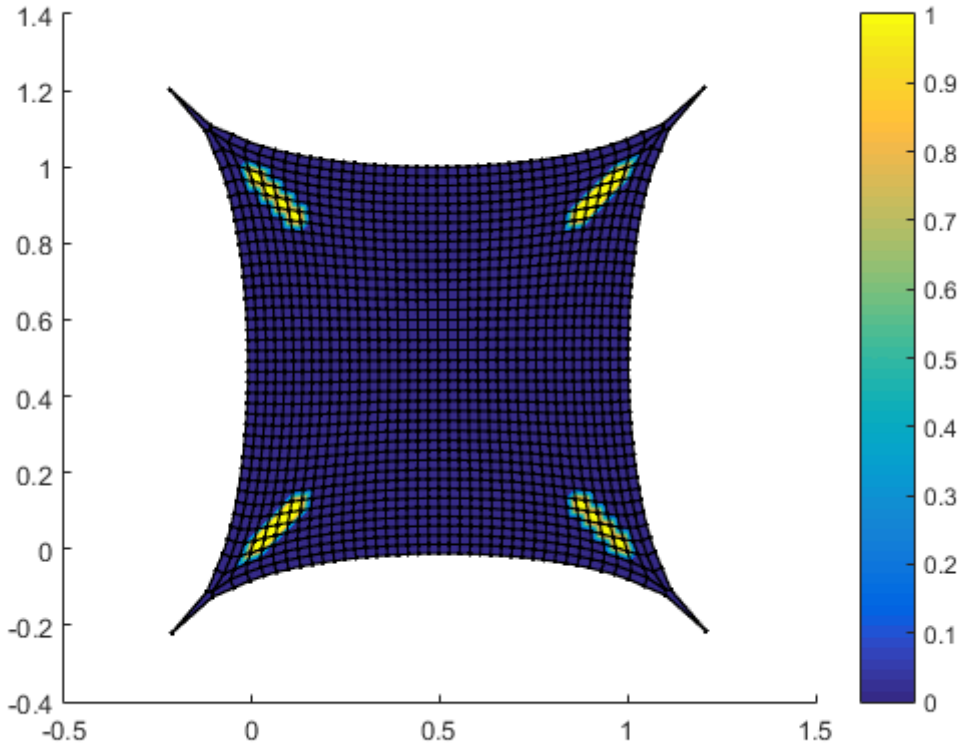


FIGURE 5.15: Points where wrinkle will be formed in a membrane under tensile corner forces

Appendix A

Code Snippets

A.1 Genetic Algorithm

```
""" A basic attempt to create a file using genetic algorithm
  Individual - A suggested solution to the problem
  Population - Collection of all individuals
"""


from operator import itemgetter, attrgetter
from att_help import *

import random
import sys
import os
import math
import re

popN = 50 # No of individuals in the population
genesPerCh = 50
max_iterations = 30
crossover_rate = 0.7
mutation_rate = 0.05
solution_found = False

min_rms = 9999999
min_chrome = []
result_file_name = 'GA_min_rms.txt'

""" Should call the abaqus code and get rms value from that"""
def getrms(chromo):
  va = 0
  for c in chromo:
    va = va + (4*c)
  return va

"""Generates random population of chromos"""
def generatePop ():
  chromos, chromo = [], []
  for _ in range(popN):
    chromo = []
    for bit in range(genesPerCh):
      chromo.append(random.randint(0,1))
    chromos.append(chromo)
  return chromos

"""Calulates fitness as a fraction of the total fitness"""
def calcFitness (errors):
  fitnessScores = []
```

```
    totalError = sum(errors)
    i = 0
    # fitness scores are a fraction of the total error
    for error in errors:
        fitnessScores.append(float(errors[i])/float(totalError))
        i += 1
    return fitnessScores

def get_force(chromos):
    forces = []
    chunks=[chromos[x:x+10] for x in range(0, len(chromos), 10)]
    for chunk in chunks:
        res = 0
        for i in range(len(chunk)):
            res = res + (math.pow(2,i)*chunk[i])
        res = float(res/1024)
        forces.append(res)

    return forces


""" Ranks the individuals according to their fitness scores """
def rankPop(chromos,iterations):
    proteins, rms_values = [], []
    i = 1
    global min_rms,min_chrome,max_rms
    no_force = [0]*genesPerCh
    result_file = open(result_file_name,'a')
    ite_info = "Iteration Number :"+str(iterations)+"\n"
    result_file.write(ite_info)

    for chromo in chromos:
        #rms = float(getrms(chromo))
        force_values = get_force(chromo)
        if chromo == no_force:
            rms = invalid_rms
        else:
            try:
                rms = float(execute(iterations,i,force_values))
            except Exception as e:
                rms = invalid_rms

        #Write the chrome and its rms value in result file
        result = "%s: %s \n"%(str(chromo),str(rms))
        result_file.write(result)

        print chromo,force_values,rms

        if rms == 0:
            global solution_found
            solution_found = True
            print '\nSOLUTION FOUND'
            print chromo
            min_rms = rms
            min_chrome = chromo
            break

        else:
            if rms<min_rms:
                min_rms = rms
                min_chrome = chromo
            rms_values.append(1/rms)
        i+=1

    result_file.close()
    fitnessScores = calcFitness(rms_values) # calc fitness scores from the erros calculated
    pairedPop = zip(chromos, rms_values, fitnessScores) # pair each chromo with its protein, ouput and fitness score
    rankedPop = sorted(pairedPop,key = itemgetter(-1), reverse = True) # sort the paired pop by ascending fitness
        score
    return rankedPop



################################################################################
""" Genetic algorithm core Functions """
```

```
def crossover (ch1, ch2):
  # at a random chiasma
  r = random.randint(0,genesPerCh)
  return ch1[:r]+ch2[r:], ch2[:r]+ch1[r:]


def mutate (ch):
  mutatedCh = []
  for i in ch:
    if random.random() < mutation_rate:
      if i == 1:
        mutatedCh.append(0)
      else:
        mutatedCh.append(1)
    else:
      mutatedCh.append(i)
  #assert mutatedCh != ch
  return mutatedCh

"""Using breed and mutate it generates two new chromos from the selected pair"""
def breed (ch1, ch2):

  newCh1, newCh2 = [], []
  if random.random() < crossover_rate: # rate dependent crossover of selected chromosomes
    newCh1, newCh2 = crossover(ch1, ch2)
  else:
    newCh1, newCh2 = ch1, ch2
  newnewCh1 = mutate (newCh1) # mutate crossovered chromos
  newnewCh2 = mutate (newCh2)

  return newnewCh1, newnewCh2

"""Fitness scores are fractions, their sum = 1. Fitter chromosomes have a larger fraction. """
def roulette (fitnessScores):
  index = 0
  cumalativeFitness = 0.0
  r = random.random()

  for i in range(len(fitnessScores)): # for each chromosome's fitness score
    cumalativeFitness += fitnessScores[i] # add each chromosome's fitness score to cumalative fitness

    if cumalativeFitness > r: # in the event of cumalative fitness becoming greater than r, return index of that
      chromo
      return i

################################################################################


""" taking a ranked population selects two of the fittest members using roulette method"""
def selectFittest (fitnessScores, rankedChromos):
  while 1 == 1: # ensure that the chromosomes selected for breeding are have different indexes in the population
    index1 = roulette (fitnessScores)
    index2 = roulette (fitnessScores)
    if index1 == index2:
      continue
    else:
      break


  ch1 = rankedChromos[index1] # select and return chromosomes for breeding
  ch2 = rankedChromos[index2]
  return ch1, ch2


""" Create new population"""
def iteratePop (rankedPop):
  fitnessScores = [ item[-1] for item in rankedPop ] # extract fitness scores from ranked population
  rankedChromos = [ item[0] for item in rankedPop ] # extract chromosomes from ranked population

  newpop = []
  newpop.extend(rankedChromos[:popN/10]) # known as elitism, conserve the best solutions to new population

  while len(newpop) < popN:
    ch1, ch2 = [], []
```

```
    ch1, ch2 = selectFittest (fitnessScores, rankedChromos) # select two of the fittest chromos

    ch1, ch2 = breed (ch1, ch2) # breed them to create two new chromosomes
    newpop.append(ch1) # and append to new population
    newpop.append(ch2)
  return newpop


def main():
  chromos = generatePop() #generate new population of random chromosomes
  iterations = 1

  # Create a result file
  result_file = open(result_file_name,'w')
  result_file.close()
  while iterations <= max_iterations and solution_found != True:
  # take the pop of random chromos and rank them based on their fitness score/proximity to target output
    rankedPop = rankPop(chromos, iterations)

    print '\nCurrent iterations:', iterations

    if solution_found != True:
    # if solution is not found iterate a new population from previous ranked population
      chromos = []
      chromos = iteratePop(rankedPop)
      iterations += 1
    else:
      break

  print '\nFinal Solution'
  print '\n Minimum rms value =',min_rms
  print '\n Chromo = ',min_chrome
  print '\n Forces = ', get_force(min_chrome)

main()
```

GA_help.py :

This file is used to run model in abaqus after updating the input file using force combinations as provided by GA_logic.py

```
import os
import subprocess

# TODO: Install visualization
import visualization
import xlsxwriter
import math

from abaqus import *
from abaqusConstants import *

invalid_rms = 999.0

class abaqusFile(object):

    def __init__(self, input_file_name, input_file_path=''):

        if input_file_path:
            path, tail = os.path.split(input_file_path)

        self.input_file = os.path.join(input_file_path, input_file_name)

    def return_rms(self, def_path):

        odbPath = def_path+".odb"
```

```
    #myViewport = session.Viewport(name='Viewport: 1',origin=(10, 10), width=150, height=100)
    rms_value = 0
    odb = visualization.openOdb(path=odbPath)
    try:
        lastFrame = odb.steps['Step-2'].frames[1]
        displacement = lastFrame.fieldOutputs['U']

        iteration_count=1
        #rms_values=[]

        endDisplacement = displacement.getSubset()
        workbook = xlsxwriter.Workbook(def_path+'.xlsx')
        worksheet = workbook.add_worksheet()

        row=0
        rms=0
        for v in endDisplacement.values:
            #print 'Position: ', v.position ,' Type: ', v.data [0],'        ', v.data[1]
            #file1.write('Node:'+str(v.nodeLabel)+'---'+str(v.data[0])+ ',' +str(v.data[1])+','+str(v.data[2])+'\n')
            disp=v.data[2]
            worksheet.write(row,0,float(v.nodeLabel))
            worksheet.write(row,1,float(disp))
            rms=rms+pow(disp,2)
            row=row+1

        rms_value=math.sqrt(rms/row)

        workbook.close()

    except Exception:
        rms_value = invalid_rms
        pass

    odb.close()


def callAbaqus(self, new_file):
    import os
    stat_info = os.stat(new_file)

    command = "abaqus j=%s int cpus=4"%new_file
    # Run abaqus through command line
    subprocess.call(command, shell=True)

    input_file_name, extension = os.path.splitext(new_file)

    # Call function return_rms to get the rms value for the specific input file
    rms_value = self.return_rms(input_file_name)
    return rms_value

def create_new_input(self, iter_count, pop_count, chromo):

    new_input_file_name = 'input_'+'iter_'+str(iter_count)+'_pop_'+str(pop_count)+'.inp'
    new_input_file = open(new_input_file_name,'w')

    count = 0       # Keeps track of which member of the chromosome to use
    checker = False
    with open(self.input_file, 'r') as f:
        previous_line = ""
        for line in f:
            try:
                if previous_line.startswith('*Cload'):
                    checker = True

                if line.startswith('** Name') and checker == True:
                    checker = False
                    count+=1

                if checker:
                    n_lin = line.split(',')
                    po = n_lin[2].split('\r')
                    aber = float(chromo[count])*float(po[0])
                    line_to_add = n_lin[0]+','+n_lin[1]+', '+str(aber)+'\r\n'
                    new_input_file.write(line_to_add)
```

```
                else:
                    new_input_file.write(line)
                previous_line = line

            except Exception as e:
                new_input_file.write(line)
                checker = False
                pass

        # Finally close the new input file
        new_input_file.close()
        return new_input_file_name

def execute(iter_count, pop_count,chromo):

    new_file = obj1.create_new_input(iter_count, pop_count, chromo)
    rms_val = obj1.callAbaqus(new_file)
    return rms_val

obj1 = abaqusFile('harshit_GA1.inp')
chromo =  [0,0,0,1,1,0,1,0,1,1,0,0,0,1,0,1]
```

A.2 Abaqus Model Postprocessing Files

A.2.1 Python module for processing .inp file

The given file is run from command-line via

```
abaqus cae noGUI=<file_name>.py
```

temp_variation.py

```
# !/usr/bin/python

import os
import subprocess

import visualization
import xlsxwriter
import math

from abaqus import *
from abaqusConstants import *

class RefineInput(object):
    """
    This class take in input file and then can refine it as
    per needed by user. It'll be containing many functions to
    change specific properties for the abaqus input file, for
    example : temperature, material properties, other boundary
    conditions, etc
    """

    def __init__(self, input_file_name, input_file_path=''):

        if input_file_path:
            path, tail = os.path.split(input_file_path)

        # TODO: Add check for input file path and make it platform independent
        # Till then, expecting input_file to be in the same directory as of this input_file_path
        self.input_file = os.path.join(input_file_path, input_file_name)
        # List to save the names of new input files created
        self.new_input_file_list = []
```

```
def create_input_file_temp(self, temp_range, load_file_number, input_file_to_change, no_of_files=1):
    """
    Number of input files created will depend upon the variation of different loading
    as well as change in input conditions and separate file will be made for each ones
    which would be later on run will different inputs.
    """

    for new_temperature in temp_range:

        # Create new input file name and append it to list
        self.temp_file_number = new_temperature
        new_input_file_name = 'input_'+'load_'+str(load_file_number)+'_temper_'+str(self.temp_file_number)+'.inp'

        # Create new input file for this particular temperature
        new_input_file = open(new_input_file_name,'w')

        # Open the original input file and start readling lines from it
        with open(input_file_to_change, 'r') as f:
            previous_line = ""
            for line in f:
                if previous_line=="*Temperature\n" and line.startswith('Set-7'):
                    line = 'Set-7, %s.\n'%new_temperature
                # Now write the new lines to the new input file
                new_input_file.write(line)
                previous_line = line

        # Finally close the new input file created after updating temperature
        new_input_file.close()
        self. new_input_file_list .append(new_input_file_name)

def create_input_file_load (self, load_range, no_of_files=1):
    """
    Create input files for load variation for different loading
    """

    self.temp_file_number = 0
    file_number = 0

    for new_load in load_range:

        # Create new input file name and append it to list
        self.load_file_number = load_range.index(new_load)+1
        new_input_file_name = 'input_'+'load_'+str(self.load_file_number)+'_temper_'+str(self.temp_file_number)+'.inp'

        # Create and open new input file for the new load
        new_input_file = open(new_input_file_name, 'w')

        # Open original input file and start reading lines from it
        with open(self. input_file, 'r') as f:
            lines_to_change_with_positive_load = [
                        'Set-3, 1, 1, 2',
                        'Set-4, 2, 2, 2'
            ]

            lines_to_change_with_negative_load = [
                        'Set-1, 1, 1, -2',
                        'Set-2, 2, 2, -2'
            ]

            for line in f:
                if line.startswith(tuple(lines_to_change_with_positive_load)):
                    line = line[:12]+'  '+str(new_load)+'.\n'
                if line.startswith(tuple(lines_to_change_with_negative_load)):
                    line = line[:12]+'  '+str(new_load*(-1))+'.\n'
                new_input_file.write(line)

        new_input_file.close()
        temp_range = range(-200, 210, 10)
        no_of_files = len(temp_range)
        input_file_to_change = new_input_file_name
        self.create_input_file_temp(temp_range, self.load_file_number, input_file_to_change, no_of_files)


def call_abaqus(self):
    """
```

```
        Make call to abaqus in noGUI mode with input file and collect the result in a
        text file for furter analysis.
        """

        result_file_name = 'result_rms.txt'
        # Create a result file
        result_file = open(result_file_name,'w')
        result_file.close()

        for input_file in self.new_input_file_list:
            import os
            stat_info = os.stat(input_file)
            if not stat_info.st_size:
                continue
            # Update the existing result file
            result_file = open(result_file_name,'a')
            command = "abaqus j=%s int cpus=4"%input_file
            # Run abaqus through command line
            subprocess.call(command, shell=True)

            input_file_name, extension = os.path.splitext(input_file)

            # Call function to write filed report
            self.writefieldreport(input_file_name)

            # Call function return_rms to get the rms value for the specific input file
            rms_value = self.return_rms(input_file_name)

            # Write rms value for current input file in rms result file
            result = "%s: %s \n"%(input_file_name,str(rms_value))
            result_file.write(result)

            result_file.close()

    def return_rms(self, def_path):
        """
        This function is copied from field_ouptput module created last year.
        Used to generate rms value from the data and show them in a text file.
        """

        odbPath = def_path+".odb"

        #myViewport = session.Viewport(name='Viewport: 1',origin=(10, 10), width=150, height=100)
        rms_value = 0
        odb = visualization.openOdb(path=odbPath)
        try:
            lastFrame = odb.steps['Step-2'].frames[1]
            displacement = lastFrame.fieldOutputs['U']

            #load = odb.rootAssembly.nodeSets['LOAD']

            iteration_count=1
            #rms_values=[]

            endDisplacement = displacement.getSubset()
            workbook = xlsxwriter.Workbook(def_path+'.xlsx')
            worksheet = workbook.add_worksheet()

            row=0
            rms=0
            for v in endDisplacement.values:
                #print 'Position: ',v.position,' Type: ',v.data[0],'  ',v.data[1]
                #file1.write('Node:'+str(v.nodeLabel)+'---'+str(v.data[0])+ ',' +str(v.data[1])+','+str(v.data[2])+'\n')
                disp=v.data[2]
                worksheet.write(row,0,float(v.nodeLabel))
                worksheet.write(row,1,float(disp))
                rms=rms+pow(disp,2)
                row=row+1

            rms_value=math.sqrt(rms/row)

            workbook.close()

        except Exception:
            pass
```

```
        odb.close()

        return rms_value

    def writefieldreport(self, def_path):
        """
        Function to write field reports for specific obd files in a text file.
        TODO: This can be expanded to accept the filed reports of variables we needed
        more explicitly.
        """

        # Take the file name of input file as we expect odb to be made from the same file
        odbPath = def_path+".odb"
        o1 = session.openOdb(name=odbPath)
        session.viewports['Viewport: 1'].setValues(displayedObject=o1)
        odb = session.odbs[odbPath]

        # Now write filed report in a spearate text file
        report_file_name = '%s_report.txt'%def_path
        try:
            session.writeFieldReport(fileName=report_file_name, append=ON,
                                     sortItem='Node Label', odb=odb, step=1, frame=1, outputPosition=NODAL,
                                     variable=(('U', NODAL), ('UR', NODAL), ('E', INTEGRATION_POINT), ('S',
                                     INTEGRATION_POINT), ('SENER', INTEGRATION_POINT), ))
        except Exception:
            pass

# Final run fot the class to Refine input
# Creating instance of class with single input file
nasa_1 = RefineInput('NASA_SCORE.inp')

# Creating input file for temp range
#temp_range = range(-200, 210, 10)
#no_of_files = len(temp_range)
#nasa_1.create_input_file_temp(temp_range, no_of_files)

# Creating multiple input files for different load ranges
load_range = range(1, 31)
 no_of_files = len(load_range)
nasa_1. create_input_file_load (load_range, no_of_files )

# Make call to abaqus to process the input files accordingly
nasa_1.call_abaqus()
```

A.3 Vibration and Wrinkle formation

A.3.1 Stiffness and Mass Matrix

The following MATLAB code is used to calculate the Global stiffness matrix by first finding the local stiffness matrix. 3-point quadrature is used for geeting the jacobian matrix. Only the important lines of code are shown.

```
% Thickness of the membrane (25 micrometer)
thickness = 0.000025;
vu=0.30;
kapa=5/6;
E=2.5*10^9; %in N/m^2
rho=1;

%formation of deeb ( Flexural Rigidity or bending stiffness )
DR= E*(thickness^3)/(12*(1.-vu*vu));
deeb=DR*[1 vu 0. ;...
         vu 1 0.  ;...
         0. 0. (1.-vu)/2] ;
```

```
% Formation of Hookes matrix
C = E/(1-vu*vu);
G = E/(2*(1+vu));
hook = C*[1 vu 0;...
          vu 1  0;...
          0 0 G];
%formation of dees

G= E/(2*(1.+vu));

dees=kapa*G*[thickness 0 ;...
             0 thickness];

%formation of D or dee (for the mass and kinetic energy term)
dee=rho*[thickness 0 0;  ...
         0 thickness^3/12 0;  ...
         0 0 thickness^3/12];


% Number of elements (Change this to get finer mesh)
NXE=20;
NYE=20;

deltax=length/NXE; %element size in x-direction
deltay=width/NYE; %element size in y-direction

total_elements = NXE*NYE;
total_nodes = (NXE+1)*(NYE+1);
DOF_node = 3; % Degree of freedom per node
nod_ele = 4; % Number of nodes per element
DOF_ele = nod_ele*DOF_node; % Degree of freedom per element
total_DOF = total_nodes*DOF_node;
dimen = 2;

% initializing  the boundry conditions
for i=1:total_nodes
    % For extreme points (where force is applied)
    if (points(i,1)==0 || points(i,1)==length) && (points(i,2)==0 || points(i,2)==width)
        for j=3:5
            boundary_values(i,j)=0;
        end
    end
end

% Current loading conditions assume equal loads applied at
% the 4 boundaries along the diagnol
% Rest all nodes have 0 force

%defining the load
theta = 0.78539816339;  % 45 degress
c1 = cos(theta);
s1 = sin(theta);

for io=1:total_elements
for i=1:nod_ele
 nd(i)=edges(io,i);
end

index = feeldof(nd,nod_ele,DOF_node);
coord=zeros(nod_ele,dimen);
for k=1:nod_ele
    for j=1:dimen
        coord(k,j)=points(edges(io,k),j);
    end
end

for ig=1:gaussB
corrX = pointb(ig,1);
wtx=weightb(ig,1);

for im=1:gaussB
corrY = pointb(im,2);
wty=weightb(im,2);
```

```
[shape,deri]=shape_func(corrX,corrY);
jacob = cal_Jacobian(nod_ele,deri,coord);

detjacob = det(jacob);
invjacob = inv(jacob);

% HELPER FUNCTIONS
% Function to calculate Shape function
function [shapeF,deri]=shape_func(corrX,corrY)
% shape functions

 shapeF(1)=0.25*(1-corrX)*(1-corrY);
 shapeF(2)=0.25*(1+corrX)*(1-corrY);
 shapeF(3)=0.25*(1+corrX)*(1+corrY);
 shapeF(4)=0.25*(1-corrX)*(1+corrY);

% derivatives

 deri(1,1)=-0.25*(1-corrY);
 deri(1,2)=0.25*(1-corrY);
 deri(1,3)=0.25*(1+corrY);
 deri(1,4)=-0.25*(1+corrY);

 deri(2,1)=-0.25*(1-corrX);
 deri(2,2)=-0.25*(1+corrX);
 deri(2,3)=0.25*(1+corrX);
 deri(2,4)=0.25*(1-corrX);
end

% Function to calculate the Jacobian Matrix
function [jacob2]=cal_Jacobian(nod_ele,deri,coords)

jacob2 = zeros(2,2);
for i=1:nod_ele
 jacob2(1,1)=jacob2(1,1)+deri(1,i)*coords(i,1);
 jacob2(1,2)=jacob2(1,2)+deri(1,i)*coords(i,2);
 jacob2(2,1)=jacob2(2,1)+deri(2,i)*coords(i,1);
 jacob2(2,2)=jacob2(2,2)+deri(2,i)*coords(i,2);
end
end

% Function to calculate input for Shape function
function [point2,weight2]=feglqd2(nglx,ngly)
%  determine the largest one between nglx and ngly

    if nglx > ngly
        ngl=nglx;
    else
        ngl=ngly;
    end

%   initialization
    point2=zeros(ngl,2);
    weight2=zeros(ngl,2);

%  find corresponding integration points and weights
 [pointx,weightx]=feglqd1(nglx);       % quadrature rule for x-axis
 [pointy,weighty]=feglqd1(ngly);       % quadrature rule for y-axis

%  quadrature for two-dimension
 for intx=1:nglx                       % quadrature in x-axis
    point2(intx,1)=pointx(intx);
    weight2(intx,1)=weightx(intx);
 end

 for inty=1:ngly                       % quadrature in y-axis
    point2(inty,2)=pointy(inty);
    weight2(inty,2)=weighty(inty);
 end
end

% Quadrature in 1-D
function [point1,weight1]=feglqd1(ngl)
%   initialization
```

```
    point1=zeros(ngl,1);
    weight1=zeros(ngl,1);

%  find corresponding integration points and weights

 if ngl==1              % 1-point quadrature rule
    point1(1)=0.0;
    weight1(1)=2.0;

 elseif ngl==2          % 2-point quadrature rule
    point1(1)=-0.577350269189626;
    point1(2)=-point1(1);
    weight1(1)=1.0;
    weight1(2)=weight1(1);

 elseif ngl==3          % 3-point quadrature rule
    point1(1)=-0.774596669241483;
    point1(2)=0.0;
    point1(3)=-point1(1);
    weight1(1)=0.555555555555556;
    weight1(2)=0.888888888888889;
    weight1(3)=weight1(1);
 end
end
```

A.3.2 Mode Shapes

```
a=size(KK);
 lll =eig(KK,MM);
[V,D]=eig(KK,MM);
D=diag(sqrt(D)*length*sqrt(rho/G)); %normalizing the frequencies
lambda=max(D)
number_of_modes=6;
[D,ii]=sort(D); %sorting D in ascending order. The order is given in  ii
 ii =ii(1:number_of_modes+1); %+1 is done to accomodate for the 1st redundat mode
V=V(:,ii); %arranginf V in accordanve with that of D using ii.
VV=V(:,1:number_of_modes+1); %this is done since the 1st node and the 2nd node are exactly the same. There fore we
       start from the 2nd mode and call it as the first mode.

grid on;
 title ('Mode shapes');
  xlabel 'Node Numbers';
   ylabel 'Amplitude';
caxis ([cmin cmax]);
patch('Faces', connec, 'Vertices', geom, 'FaceVertexCData', zz, 'Facecolor', 'interp', 'Marker', '.');

colorbar;
```

A.3.3 Wrinkle Formation

```
%to compute the strain
eps=bee*eld;

%computing the stresses
sigma=dee*eps;

    end
end

EPS(i,:)=eps; %EPS stores strains for all elements
SIGMA(i,:)=sigma; %SIGMA stores stresses for all the elements

 figure

 x_stress =SIGMA_NODES(:,1);
cmin=min(x_stress);
cmax=max(x_stress);
```

```
caxis([-0.5*10^4 2.5*10^4])

patch('Faces', edges, 'Vertices', points, 'FaceVertexCData', x_stress, 'Facecolor', 'flat', 'Marker', 'o')
patch('Faces', edges, 'Vertices', points_new, 'FaceVertexCData', x_stress, 'Facecolor', 'interp','marker', '.')

for i=1:nnd


    sigma_1=SIGMA_NODES(i,1)+SIGMA_NODES(i,2)+(((SIGMA_NODES(i,1)+SIGMA_NODES(i,2))/2)^2+(SIGMA_NODES(i,3))^2)^(1/2);

    sigma_2=SIGMA_NODES(i,1)+SIGMA_NODES(i,2)-(((SIGMA_NODES(i,1)+SIGMA_NODES(i,2))/2)^2+(SIGMA_NODES(i,3))^2)^(1/2);
  if sigma_2>=sigma_1
  [sigma_1 sigma_2]
  end

    epsalon_1=EPS_NODES(i,1)+EPS_NODES(i,2)+(((EPS_NODES(i,1)+EPS_NODES(i,2))/2)^2+(EPS_NODES(i,3)/2)^2)^(1/2);

    epsalon_2=EPS_NODES(i,1)+EPS_NODES(i,2)-(((EPS_NODES(i,1)+EPS_NODES(i,2))/2)^2+(EPS_NODES(i,3)/2)^2)^(1/2);
  if epsalon_2>=epsalon_1
  [epsalon_1 epsalon_2]
  end

  if sigma_1>0 && epsalon_2<0
      wrinkle(i)=1;
  end
  if epsalon_1<0
      wrinkle(i)=1;
  end
end

figure
```

Bibliography

[1] M Salama and CH Jenkins. Intelligent gossamer structures: a review of recent developments and future trends. *AIAA paper*, 1196:2001, 2001.

[2] SC Gajbhiye, SH Upadhyay, and SP Harsha. Finite element analysis of an inflatable torus considering air mass structural element. *Advances in Space Research*, 53(1):163–173, 2014.

[3] SC Gajbhiye, SH Upadhyay, and SP Harsha. Vibration analysis of an inflatable torus based on mode shape. *AIAA journal*, 51(6):1526–1532, 2013.

[4] Houfei Fang, E Im, UO Quijano, KW Wang, J Hill, J Moore, Jim Pearson, Changgeng Lui, and Frank Djuth. High-precision adaptive control of large reflector surface. In *Earth Science Technology Conf. 2008, USA*, 2008.

[5] Henno Allik and Thomas JR Hughes. Finite element method for piezoelectric vibration. *International journal for numerical methods in engineering*, 2(2): 151–157, 1970.

[6] Wesley Wong and Sergio Pellegrino. Wrinkled membranes ii: analytical models. *Journal of Mechanics of Materials and Structures*, 1(1):27–61, 2006.

[7] Wesley Wong and Sergio Pellegrino. Wrinkled membranes iii: numerical simulations. *Journal of Mechanics of Materials and Structures*, 1(1):63–95, 2006.

[8] CG Wang, HF Tan, XW Du, and ZM Wan. Wrinkling prediction of rectangular shell-membrane under transverse in-plane displacement. *International Journal of Solids and Structures*, 44(20):6507–6516, 2007.

[9] Xiaoyun Wang, Wanping Zheng, and Yan-Ru Hu. Active flatness control of space membrane structures using discrete boundary sma actuators. In *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*, pages 1108–1113. IEEE, 2008.

[10] Umesh A Korde. Large-displacement closed-loop control of variable area electrostatic actuation for membrane reflectors. *Journal of Intelligent Material Systems and Structures*, 2008.

[11] Jeffrey Ray Hill. *High Precision Surface Control of Flexible Space Reflectors*. PhD thesis, The Pennsylvania State University, 2011.

[12] Joseph R Blandino, John D Johnston, and Urmil K Dharamsi. Corner wrinkling of a square membrane due to symmetric mechanical loads. *Journal of Spacecraft and Rockets*, 39(5):717–724, 2002.

[13] CH Jenkins, F Haugen, and WH Spicher. Experimental measurement of wrinkling in membranes undergoing planar deformation. *Experimental Mechanics*, 38(2):147–152, 1998.

[14] Hang Shi, Bingen Yang, Mark Thomson, and Houfei Fang. *A nonlinear dynamic model and free vibration analysis of deployable mesh reflectors*. Pasadena, CA: Jet Propulsion Laboratory, National Aeronautics and Space Administration, 2011.

[15] Hang Shi, Bingen Yang, Mark Thomson, and Houfei Fang. Coupled elastic-thermal dynamics of deployable mesh reflectors. In *52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference 19th AIAA/ASME/AHS Adaptive Structures Conference 13t*, page 2001, 2011.

[16] Joseph R Blandino, John D Johnston, Jonathan J Miles, and Urmil K Dharamsi. The effect of asymmetric mechanical and thermal loading on membrane wrinkling. *AIAA Paper*, 1371:22–25, 2002.

[17] Fujun Peng, Xin-Xiang Jiang, Yan-Ru Hu, and Alfred Ng. Actuation precision control of sma actuators used for shape control of inflatable sar antenna. *Acta Astronautica*, 63(5):578–585, 2008.

[18] Fujun Peng, Xin-Xiang Jiang, Yan-Ru Hu, and Alfred Ng. Application of sma in membrane structure shape control. *Aerospace and Electronic Systems, IEEE Transactions on*, 45(1):85–93, 2009.

[19] Fujun Peng, Yan-Ru Hu, and Alfred Ng. Development of ga-based control system for active shape control of inflatable space structures. In *Control Applications, 2005. CCA 2005. Proceedings of 2005 IEEE Conference on*, pages 577–582. IEEE, 2005.

[20] Fujun Peng, Yan-Ru Hu, and Alfred Ng. Testing of inflatable-structure shape control using genetic algorithms and neural networks. *AIAA journal*, 45(7): 1771–1774, 2007.

[21] KD Dhuri and P Seshu. Multi-objective optimization of piezo actuator placement and sizing using genetic algorithm. *Journal of Sound and Vibration*, 323 (3):495–514, 2009.

[22] Ryan Orszulik, Jinjun Shan, and Michael Stachowsky. Membrane structure active flatness control using genetic algorithm with online objective reweighting. *Acta Astronautica*, 68(11):2012–2024, 2011.