Declaration of Contribution

All metaheuristic code and Abaqus scripts were written by me as well as data analysis code except for the fast Fourier transform code which was provided by Arnab Banerjee from Callaghan Innovation.

Abaqus 6.14, Matlab 2016B and Python 3 were used to do analysis and experiments.

All experiment data used in this report was collected by me.

Design Optimisation for Multi-Material 3D printing

Project Report – September 2017

Nick Waddington
Department of Engineering Science
University of Auckland

Supervised by Emilio Calius

Abstract

This project addresses some of the challenges in using metaheuristics to arrange different materials in a structure to match desired properties. The report discusses the best voxel resolution to use for getting a good match to the desired properties quickly.

It discusses the experiments done to calculate the stiffness of physical, homogeneous beams. It then discusses the metaheuristic used and how it was implemented. It talks about the experiments that were done with the metaheuristic, both to use the metaheuristic to match the natural frequencies of an existing beam, as well as spacing the natural frequencies 10Hz apart.

Finally, it talks about the results of all of the experiments. Specifically, their mode shapes, rates of convergence and resulting geometries.

Acknowledgements

I would like to acknowledge Emilio Calius, John Cater and Andrew Hall for their guidance over the whole of this project.

I would also like to thank Mark Battley and Piaras Kelly from University of Auckland and Arnab Banerjee from Callaghan Innovation for their assistance with Abaqus.

Also, I would like to thank Marlon from Callaghan Innovation for his assistance with setting up the experiments and guiding me through them.

Table of Contents

Table of Figures 6
1 Introduction 7
1.1 Aims 7
1.2 Objectives 7
1.3 Background 8
1.3.1 Multi-Material 3D Printing 8
1.3.2 Metaheuristics 8
1.3.3 Finite Element Method 8
2 Literature Review 10
3 Methodology 11
3.1 Physical Natural Frequency Experiments 11
3.2 Flexible Material Results 12
3.3 Stiff Material Results 15
3.4 Metaheuristic Experiments 17
3.5 Modal Analysis 23
3.6 Even Spread Metaheuristic Experiments 26
4.1 Discussion 32
4.1.1 Mode shapes for matching target frequencies 32
4.1.2 Rate of convergence for matching target frequencies 32
4.1.3 Resulting geometries for matching target frequencies 32
4.1.4 Rate of convergence for clustering frequencies 32
4.1.5 Resulting geometries for clustering frequencies 33
4.2 Limitations 34
4.3 Future Work 35
5 Conclusions 36
Appendices 37
Python Script 38
Matlab Functions 42
Main Matlab Script 43
References 45

Table of Figures

Figure 1: Selection of raw data with fitted exponential 13
Figure 2: Data adjusted to remove exponential decay 13
Figure 3: Fast Fourier Transform results 14
Figure 4: Stiff Material Raw Data 15
Figure 5: Selection of data - vertically adjusted 16
Figure 6: Fast Fourier Transform results 16
Figure 7: Structure of target beam 17
Figure 8: Objective Function over all iterations 19
Figure 9: Best Solution found over all iterations 19
Figure 10: Objective Function over iterations 20
Figure 11: Best Objective over all Iterations 20
Figure 12: Objective function over all iterations 21
Figure 13: Best objective over all iterations 21
Figure 14: Frequency error over all iterations 22
Figure 15: Target beam 1st natural frequency mode shape 23
Figure 16: 180 voxel solution beam 1st natural frequency mode shape 23
Figure 17: Target beam 2nd natural frequency mode shape 24
Figure 18: 180 voxel solution beam 2nd natural frequency mode shape 24
Figure 19: Target beam 3rd natural frequency mode shape 24
Figure 20: 180 voxel solution beam 3rd natural frequency mode shape 24
Figure 21: Target beam 4th natural frequency mode shape 25
Figure 22: 180 voxel solution beam 4th natural frequency mode shape 25
Figure 23: Target beam 5th natural frequency mode shape 25
Figure 24: 180 voxel solution beam 5th natural frequency mode shape 25
Figure 25: (280GPa and 0.12GPa) and (280GPa and 1.2GPa) best solution 28
Figure 26: 28GPa and 1.2GPa best solution 28
Figure 27: 2.8GPa and 1.2GPa best solution 29
Figure 28: Experiment 1 log error over iterations 29
Figure 29: Experiment 2 log error over iterations 30
Figure 30: Experiment 3 log error over iterations 30
Figure 31: Experiment 4 log error over iterations 31
Figure 32: Best 180 Voxel Match 37
Figure 33: Best 45 Voxel Match 37
Figure 34: Best 720 Voxel Match 37

1 Introduction

Multi-material 3D printing presents many possibilities in manufacturing and product design where existing materials can be combined to create new materials with interesting properties. The effects of combining different materials can be non-linear and unpredictable. This project is a part of working towards a design assistance tool to help people analyse the effects of combining different materials and incorporate them into their designs.

This project builds on previous work by Max Wang on using metaheuristics to match natural frequencies of a cantilever beam. His report demonstrated that a genetic algorithm could match frequencies to 5% accuracy in 20000 function evaluations.

This report will discuss the effects of different voxel resolutions on solution quality and speed.

1.1 Aims

This project aims to do the following:

- Determine whether a metaheuristic can be used to create a structure with natural frequencies that match specified natural frequencies.
- Determine whether a metaheuristic can be used to create a structure with natural frequencies that are closely spaced.
- Compare theoretical structures with their 3D printed counterparts.

1.2 Objectives

This project has the following objectives:

- Implement a method of evaluating the natural frequencies of a cantilever beam.
- Implement a metaheuristic and test its effectiveness on matching prescribed frequencies and clustering frequencies 10Hz apart.
- Determine material properties of the two materials that will make up the multi-material beam.
- Compare natural frequencies of multi-material 3D printed beam to theoretical frequencies

1.3 Background

1.3.1 Multi-Material 3D Printing

Multi-Material 3D printing is the process of simultaneously 3D printing with different materials. 3D printing with multiple materials has many potential applications, as described in [6], which include the ability to combine materials in such a way that they match desired properties. An example of this would be combining a stiff material and a flexible material such that it had the strength of the stiff material while retaining the flexibility of the other material.

In 3D printing, voxels are the 3D equivalent of pixels. For the purposes of this report, they are considered to be rectangular prisms with homogeneous materials.

1.3.2 Metaheuristics

A metaheuristic is the procedure of solving optimisation problems. They use a combination of local searches and some method to control the direction of the searches. They are typically used to find a good solution to an optimisation problem where an optimal solution is too difficult to find. Terms that are often used to talk about metaheuristics are solution, objective function, neighbouring solution and local optimum. A solution, with respect to metaheuristics, is a value for each of the problem variables that might represent the dimensions or materials of the problem. The objective function is a measure of how good the current solution is. For example, there might be an objective function maximizing the tensile strength of a structure or minimizing the error of the current solution from the desired properties. A neighbouring solution is a solution that is slightly different from the current solution. An example of this might be changing a variable to be slightly bigger or changing a binary variable from a 1 to a 0. A local optimum is a solution where all the neighbouring solutions have a worse objective function.

The next descent metaheuristic involves sequentially trying all neighbouring solutions and keeping the neighbourhood solution only if it improves the objective function. When a local optimum is found, the process is restarted with a randomised starting solution.

1.3.3 Finite Element Method

The finite element method is a way of breaking down an object with a complex geometry or different materials into smaller blocks that are the same material and have a simple geometry. The smaller blocks are represented by nodes and neighbours. For example, a cube element could be represented as 8 nodes with a node on each corner of the cube. The smaller blocks need to be stitched together to form the original geometry so constraints are put on the nodes to achieve this. For example, to join two cubes together we can use a “tie constraint” to set the 4 coincident nodes to remain coincident.

Shell elements are used when an object is relatively long in the x and y dimensions, thin in the z dimension, of uniform thickness and forces are primarily applied in the z dimension. They are defined by a uniform cross section and a thickness. In the case of this model, the beam is split into square shell elements, all with the same thickness.

There is a multi-material 3D beam that is split into a grid of shell elements where each element is assigned one of the two materials. Neighbouring elements have tie constraints between them and there is an encastre boundary condition (nodes have 0 displacement and moment in every direction) at all of the faces on one end of the beam.

This model can then be used to calculate the natural frequencies of the beam. The wave equation can be applied to each of the elements to give a system of equations. The eigenvalues of this system of equations can then be used to determine the natural frequencies of the beam.

Abaqus 6.14 and Python scripting have been used to automate the process of creating the elements, setting up constraints and solving for the natural frequencies of the beam.

2 Literature Review

The aim for this project was to explore using metaheuristics to arrange materials in a multi-material structure, optimising for some property. Several discrete and continuous algorithms were considered to see which algorithms performed the best. For this project, the example problem of a speaker diaphragm was used because deflections were small and linear so would be easier to analyse.

For discrete variable algorithms, it is possible to represent our object with an integer decision variable for each voxel, representing the material in that voxel. From here, a neighbourhood rule is used where there is a neighbour for each possible material in each possible voxel. Discrete problems can be solved using Tabu Search, Simulated Annealing or the Genetic Algorithm as described in reference [1]. One disadvantage of this is that the neighbourhood could get very large with higher voxel resolution so the problem could take a long time to give decent answers. These algorithms have highly varying performance depending on the problem reference [2] so it is difficult to know how they will perform for this particular problem. One possible way of overcoming the large neighbourhood is to use one the large neighbourhood search techniques discussed in reference [3] using variable depth methods or network search methods.

For continuous variable algorithms, the object can be represented as a continuous variable for groups of voxels where the variables represent the material composition of that group. Reference [5] compares the genetic algorithm, particle swarm optimisation and the differential evolution method for solving a relatively small continuous problem. For that problem, the particle swarm optimisation and the differential evolution performed well but the genetic algorithm performed poorly. There was not much literature for larger scale continuous problems so it is difficult to know how well these algorithms will perform on larger problems. Another potential issue is that the properties of a group of voxels that are 50% one material and 50% another, are highly dependent on how these materials are arranged. Reference [6] suggests that arranging the materials in a checkerboard pattern will weaken the material compared to if they were randomly arranged because of the lines of weaker material in the group.

Overall, current literature has a lot of different algorithms but none have been applied to a similar enough problem to this to give an idea of how well they will perform so experimentation is required.

3 Methodology

3.1 Physical Natural Frequency Experiments

Six sample beams were 3D printed on a Stratasys Multi-Material printer, 3 of the stiffer material, Vero Clear, and 3 of the more flexible material, RWT-EBK 250. The natural frequencies of the beams were measured by doing a fast Fourier transform on data obtained from a Keyence LK H050 Laser Displacement Sensor. The aim of this experiment was to determine the Young's Modulus of the two materials using the natural frequency and geometry of the beam.

For this experiment each of the beams were clamped at one end with the laser displacement sensor pointing at the other end of the beam because the other end had the largest displacement when oscillating. A small piece of tape was used on the end of the clear beam to give the laser an opaque surface (Figure 1). The beams were then tapped with a small allen key to get the beams to oscillate. The results were checked by looking at the displacement over time graph to see if the oscillation was distinguishable. This was repeated 5 times for each beam giving 15 total data sets for each material.

The more flexible material sagged under its own weight so the experiments were done sideways instead to reduce this effect as shown in Figure 1.

The displacement was recorded at 10 kHz (100 µs per sample) and 20 seconds of displacement data was recorded.

Figure 1: Experimental setup for RWT-EBK 250 beam and Vero Clear beam

3.2 Flexible Material Results

Figure 3 shows the raw data for the first sample taken. First, one second of the data was selected, starting at the spike in displacement (Figure 4). The selected data was shifted vertically so that the minimum displacement of the selection was 0 and fitted to a decaying exponential. An exponential fit was then subtracted from the data (Figure 5). The adjusted data was run through a fast Fourier transform (Figure 6). The highest displacement for this sample was at 32Hz. Across all samples there was a mean natural frequency of 33Hz and a standard deviation of 0.7Hz. Using the following formula, the stiffness of the beam was derived.

$$\omega = 1.875^2 \sqrt{\frac{EI}{\rho A L^4}} \text{ [7]}$$

Where ω is the circular natural frequency, $2\pi * 33$, I is the moment of inertia, ρ is the density (1120kgm^{-3}), A is the cross section area (20mm by 2mm) and L is the length (100mm). This gives us a Young's Modulus of 1170 MPa for the flexible material. It was expected that the Young's Modulus would be around 100MPa from the material data sheet so this is slightly higher than expected.

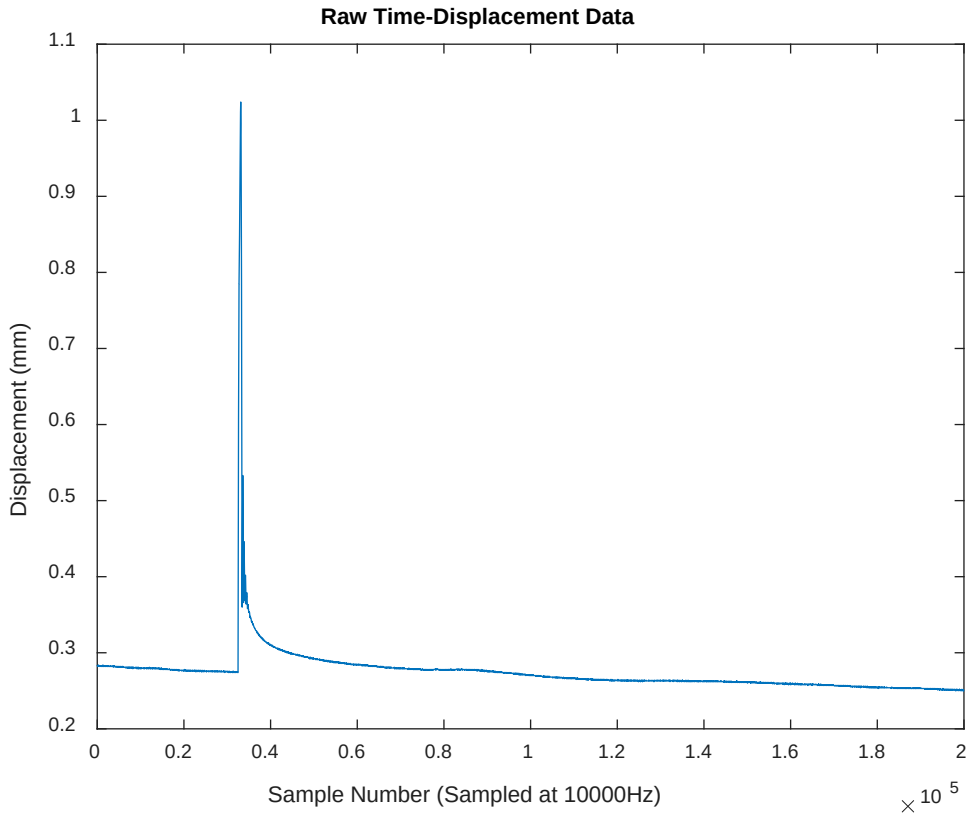


Figure 2: Raw data for flexible material

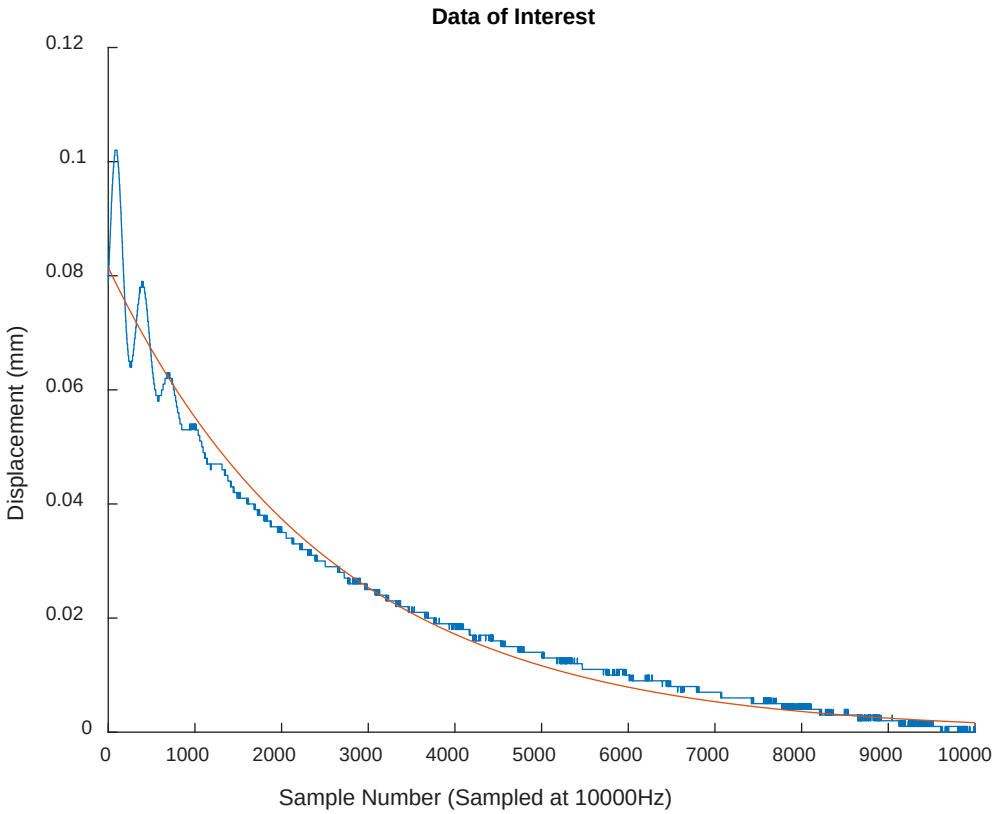


Figure 1: Selection of raw data with fitted exponential

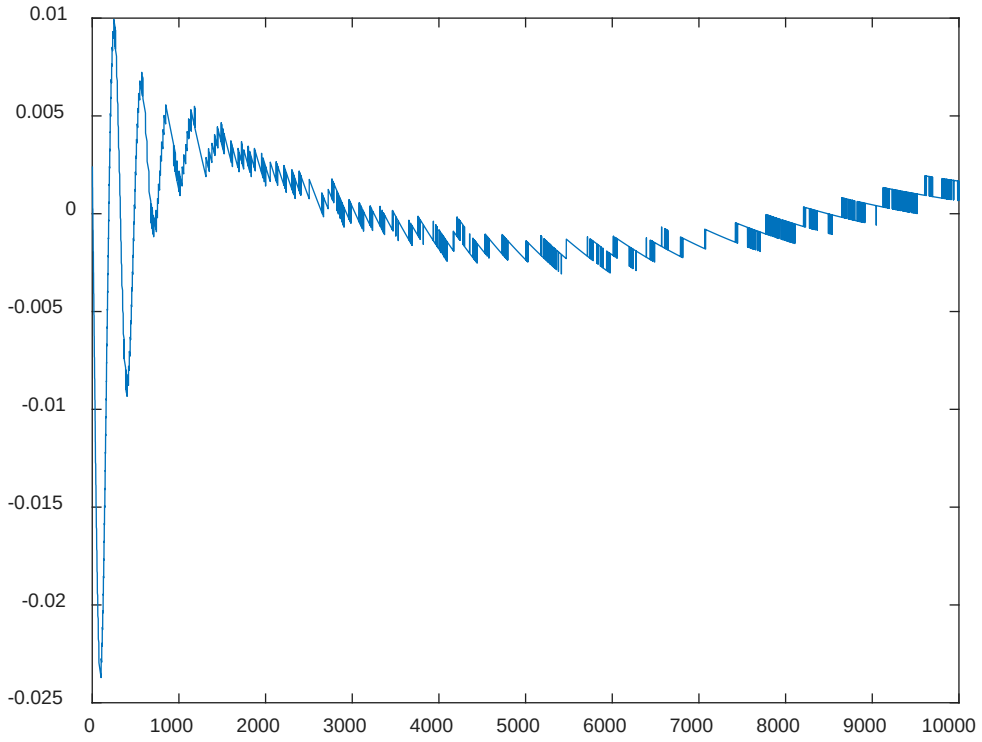


Figure 2: Data adjusted to remove exponential decay

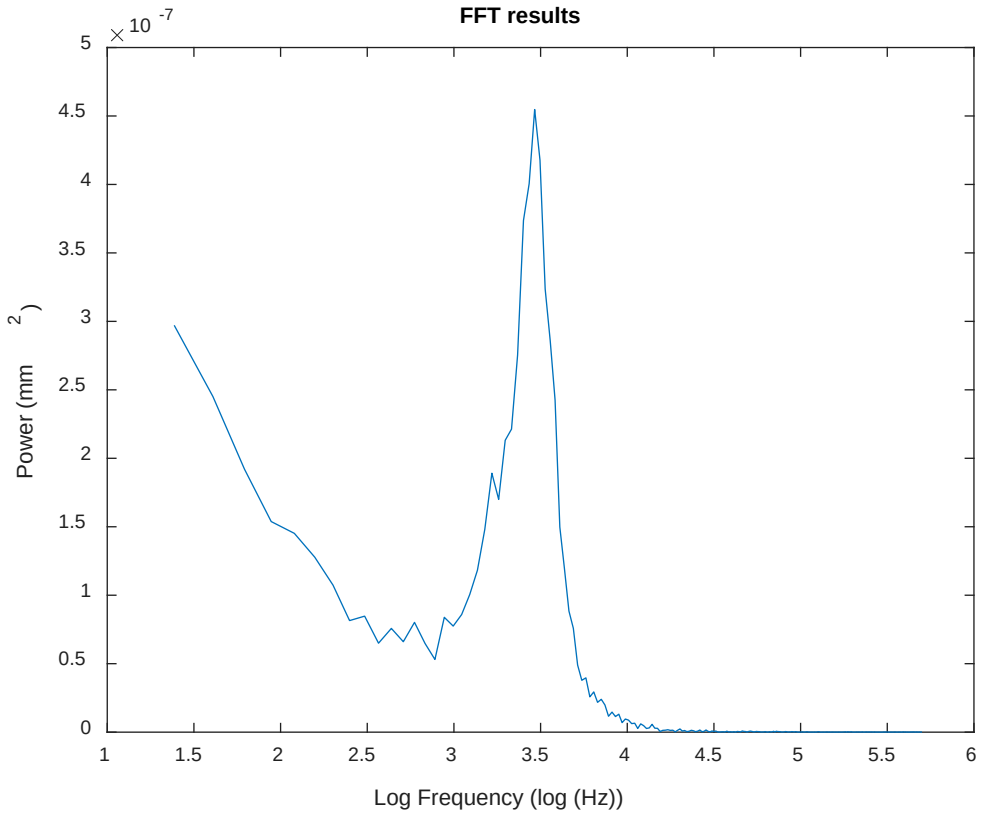


Figure 3: Fast Fourier Transform results

3.3 Stiff Material Results

Figure 7 shows the raw data set for one of the stiff material samples. Similar to the flexible material, one second of the data was selected, starting at the spike in displacement (Figure 8). The selected data was vertically adjusted so that the mean displacement of the selection was zero. The selected data was again run through a fast Fourier transform (Figure 9). There are two distinct peaks, one at 1Hz and another at 53 Hz. The second peak is used to estimate the natural frequency of the beam. From the 15 samples, 14 gave a distinct peak with a mean of 50.1 Hz and a standard deviation of 2.3. Similarly to the flexible material, we can use the analytical solution for the natural frequency of a cantilever beam to estimate the Young's Modulus of the Vero clear material at 2820 MPa.

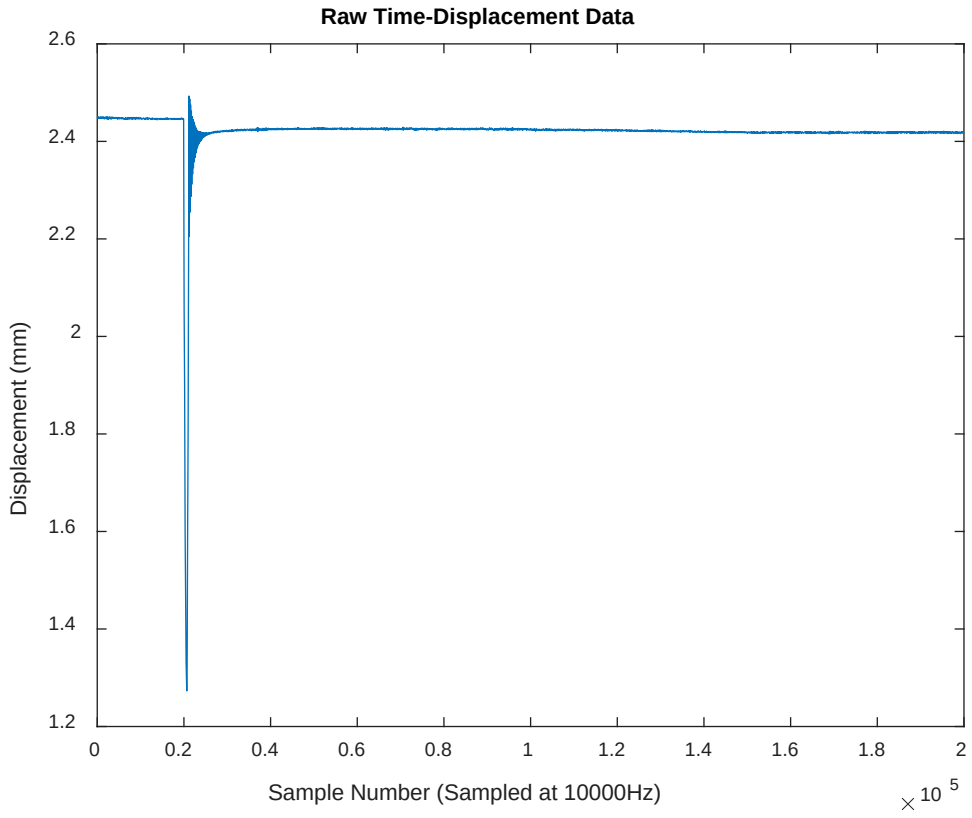


Figure 4: Stiff Material Raw Data

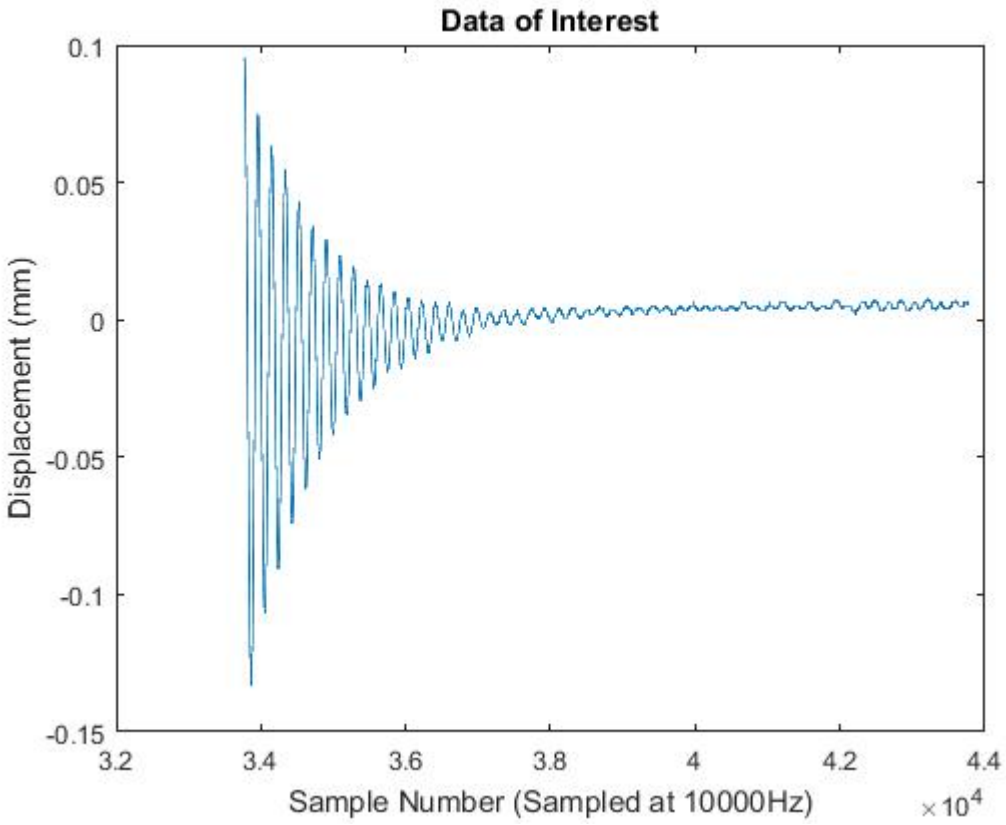


Figure 5: Selection of data - vertically adjusted

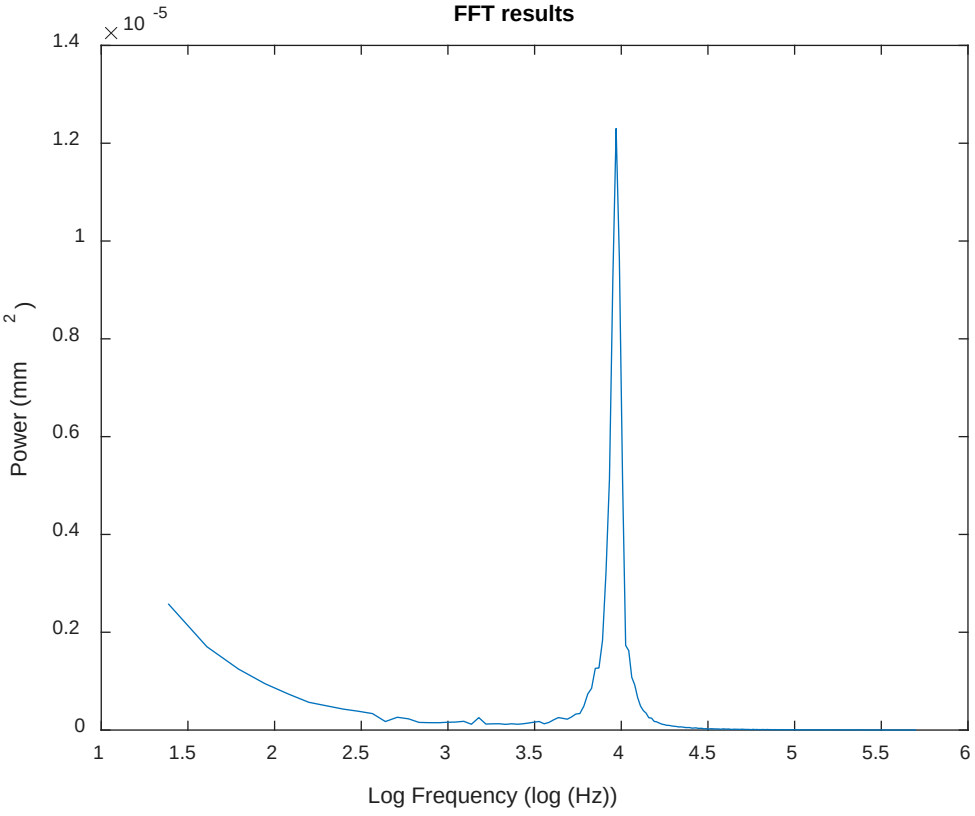
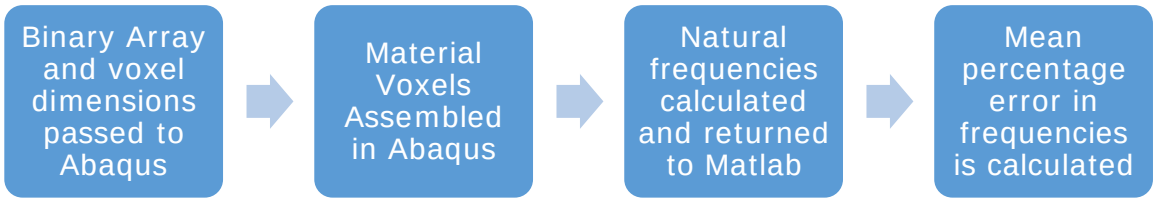


Figure 6: Fast Fourier Transform results

3.4 Metaheuristic Experiments

The aim of the metaheuristic is to arrange voxels of the stiff and flexible materials in a grid, such that the first five natural frequencies of the resulting cantilever beam match some predetermined natural frequencies. This was implemented using MATLAB 2016b and Abaqus 6.14 (See Appendix for Code). The beam had dimensions 100mm x 20mm x 2mm. The objective function was calculated as shown below.



The beam was split into m by n square voxels of equal size. The array was a binary m by n array where a 1 in cell i, j means there is a stiff material in the corresponding voxel (0 for flexible material). Voxel dimensions were calculated by dividing the length (width) of the beam by m (n). Data was passed from MATLAB to Abaqus via a python script. The voxels were assembled in Abaqus with a python script that created instances of each voxel of material as shell elements and used tie constraints to join them together. The end of the beam was fixed with encastre conditions on each of the end faces of the elements. The natural frequencies were calculated by running a natural frequency job in Abaqus, taking only the first five frequencies. These frequencies were then passed back to MATLAB by reading the Abaqus ODB file for the job from MATLAB. The objective function was taken as the mean absolute percentage error as follows, where vector x is the output frequencies and vector t is the target frequencies.

$$\text{error} = \frac{1}{5}\sum_{f=1}^{5}\left|\frac{x_f - t_f}{x_f}\right|$$

The target frequencies were obtained by finding the first five natural frequencies of a periodic structure beam (Figure 10) using the same Abaqus script as earlier with a 720 voxel model.

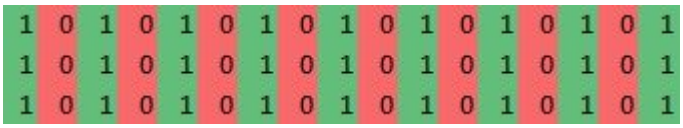


Figure 7: Structure of target beam

A next descent metaheuristic was chosen to optimise the arrangement of materials in the cantilever beam because it produced local optima in a reasonably small number of objective function evaluations.

To figure out how many voxels to split the beam into, we ran 3 experiments with the coarsest voxel size being 45 voxels (3 by 15), followed by 180 (6 by 30) and finally a 720 voxel case being the finest (12 by 60). The next smallest number of voxels while being double the previous would be 2880 voxels. The 2880 voxel Abaqus model took around 10 minutes to assemble and solve for a single set of natural frequencies so running several thousand would take an unreasonable amount of time.

Table 1 shows the mean run times and standard deviations. Note that some of the Abaqus run times were extreme outliers for the 45 and 180 voxel experiments (greater than 120 seconds) and were causing the standard deviations to go above 20. To get a more representative measure of the standard deviation of run times, the extreme outliers were omitted.

Number of voxels	Mean Abaqus Run time per function call (s)	Standard Deviation
45	13.45	0.47
180	33.60	2.19
720	134.0	16.6

Table 1: Runtimes for each iteration by number of voxels

Experiments were started with a homogeneous beam of flexible material. A next descent search was then conducted until a local optimum was found. When a local optimum was found, it was recorded and then each voxel was reassigned the stiff or flexible material with equal probability.

Experiments for each voxel size were run for 5000 iterations for the 45 voxel experiment, 23 local optima were found (figures 11 and 12 show the objective function improvement and the current best objective over all iterations). The best of these had a percentage error of 0.3%. For the 180 voxel experiment, 4 local optima were found with a best percentage error 0.03% was found (figures 13 and 14 show the objective function improvement and the current best objective over all iterations). For the 720 voxel experiment, no local optima were found but the best solution found had a best percentage error of 0.59% (figures 15 and 16 show the objective function improvement and the current best objective over all iterations). The material assignments for the lowest error solutions are shown in figures 34 to 36 in the appendices where the red voxels are the flexible material and the green voxels are the stiff material. Figure 17 shows the 180 voxel experiment convergence of the different natural frequencies to the target frequencies. As expected, for each of the 4 local optima, each of the natural frequency errors start out far from zero but gradually converge towards zero.

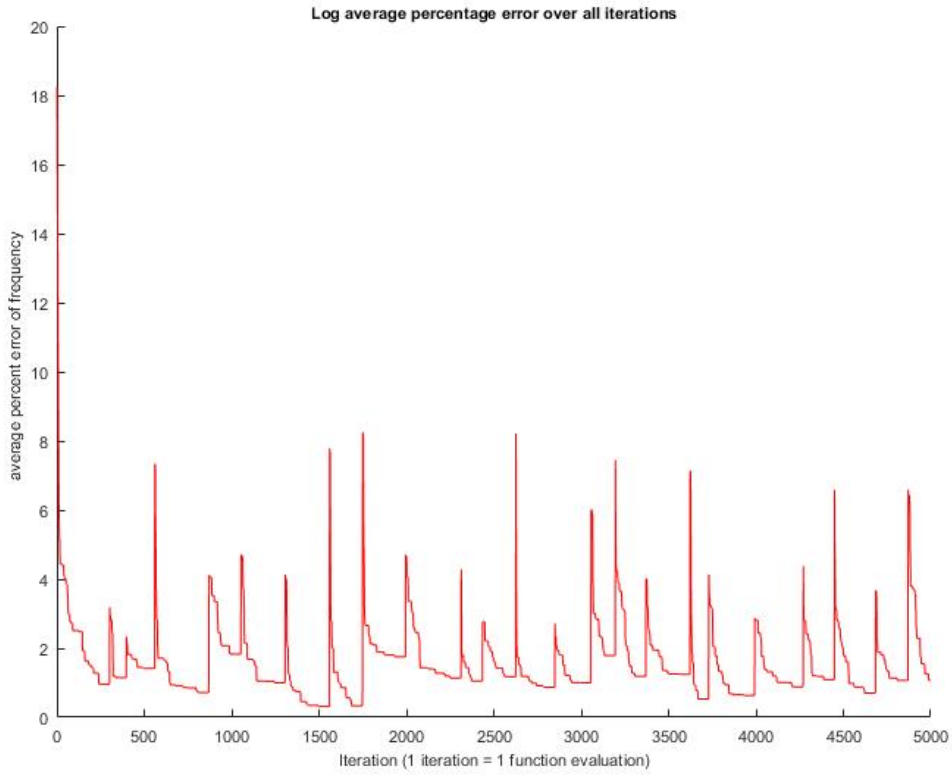


Figure 8: Objective Function over all iterations

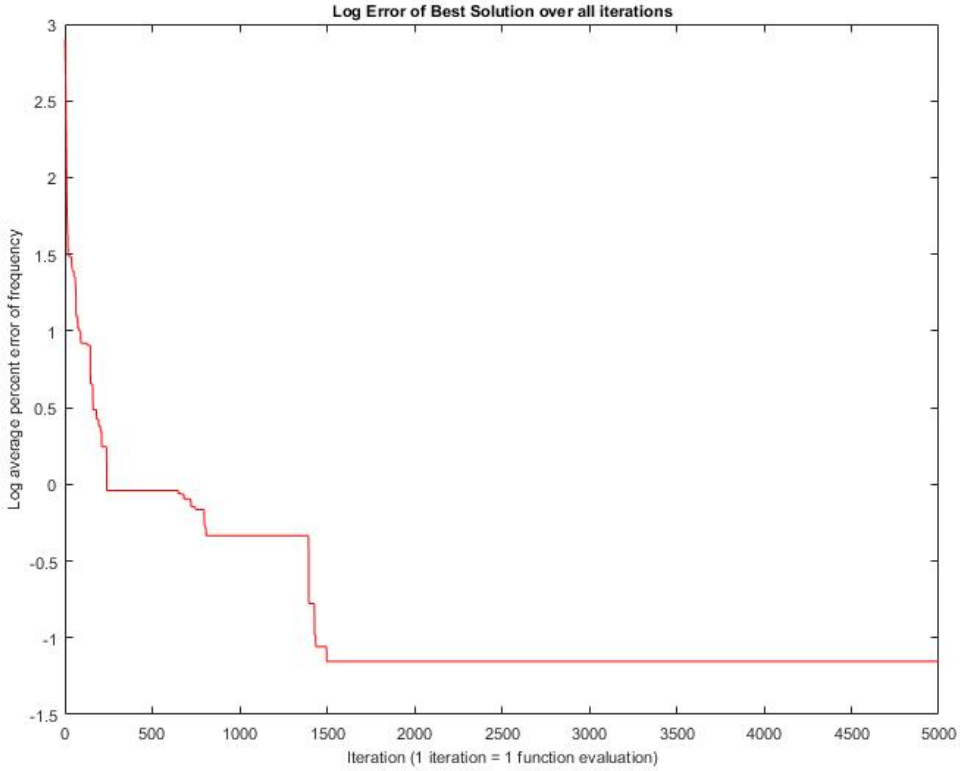


Figure 9: Best Solution found over all iterations

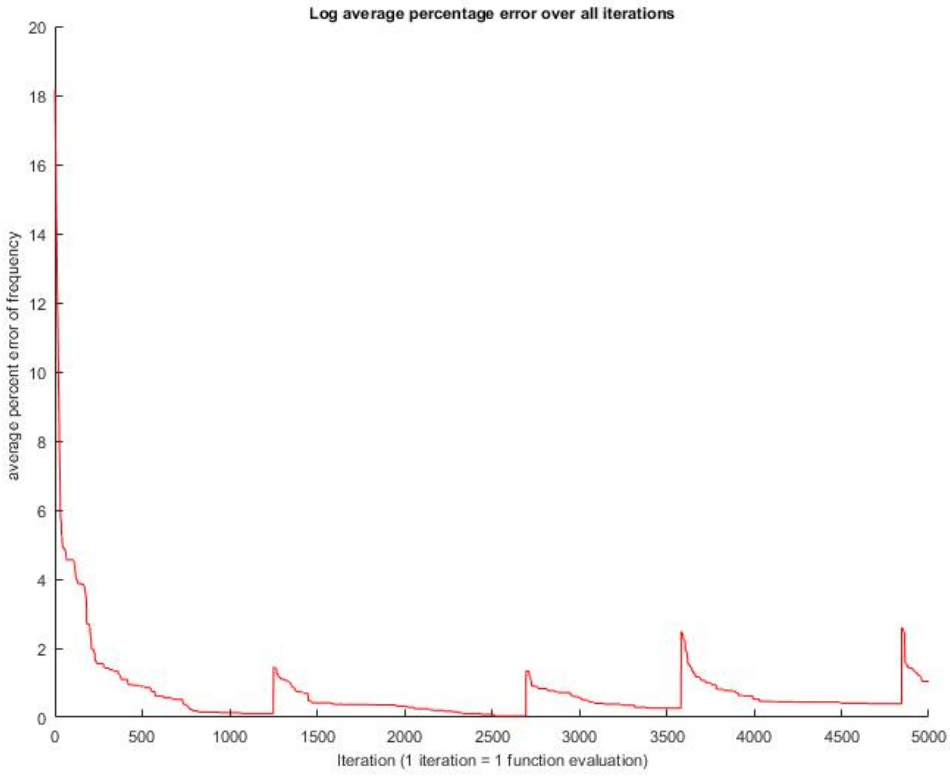


Figure 10: Objective Function over iterations

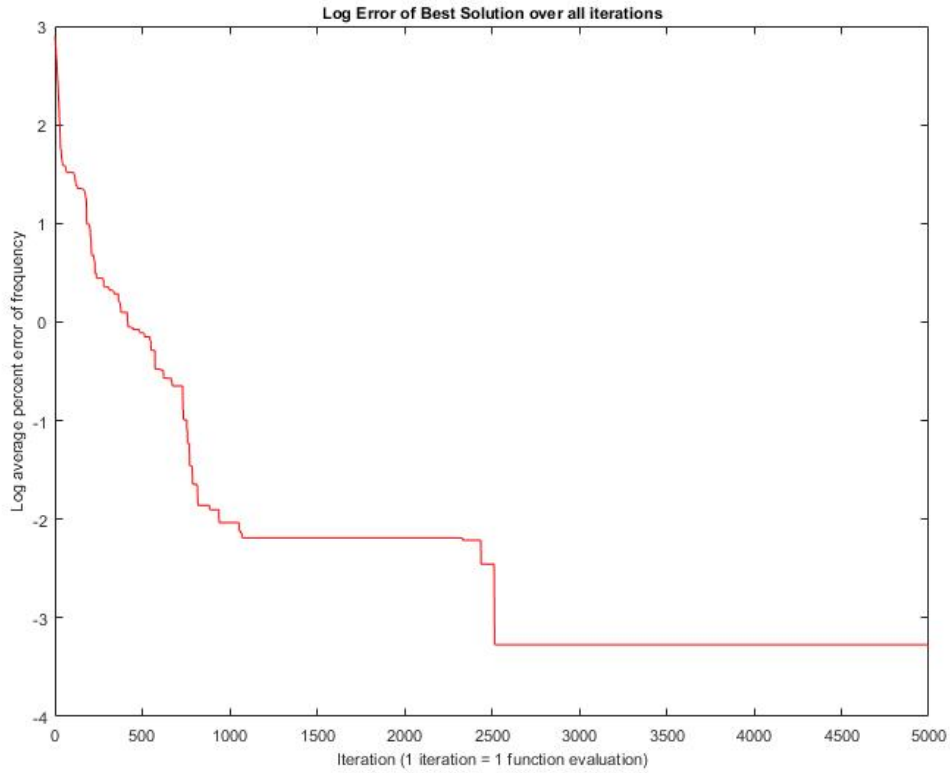


Figure 11: Best Objective over all Iterations

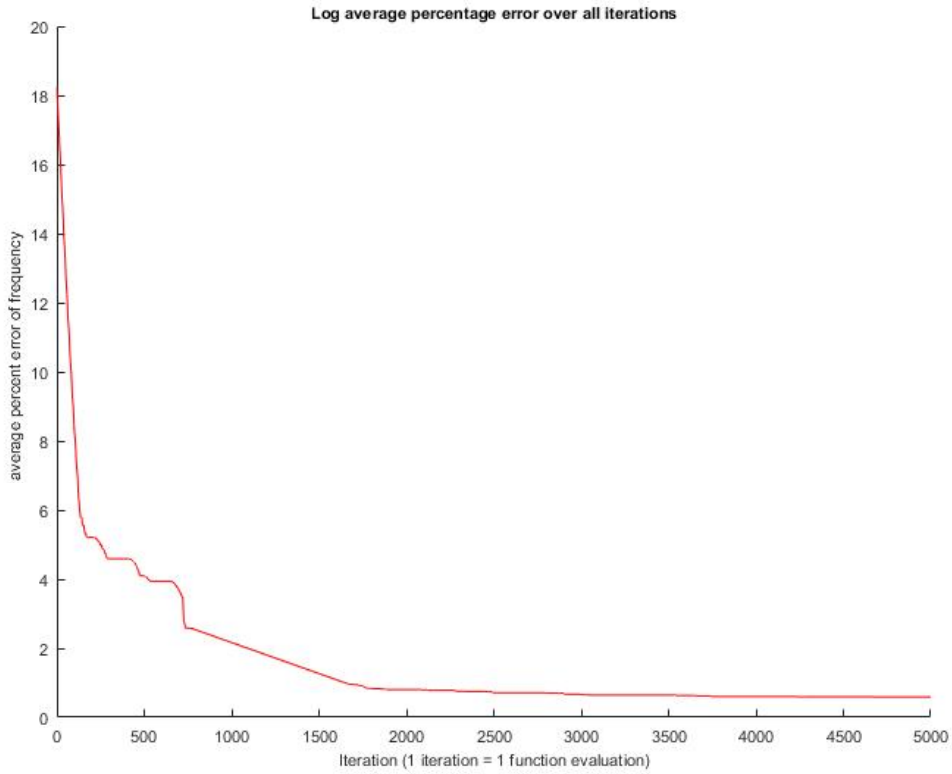


Figure 12: Objective function over all iterations

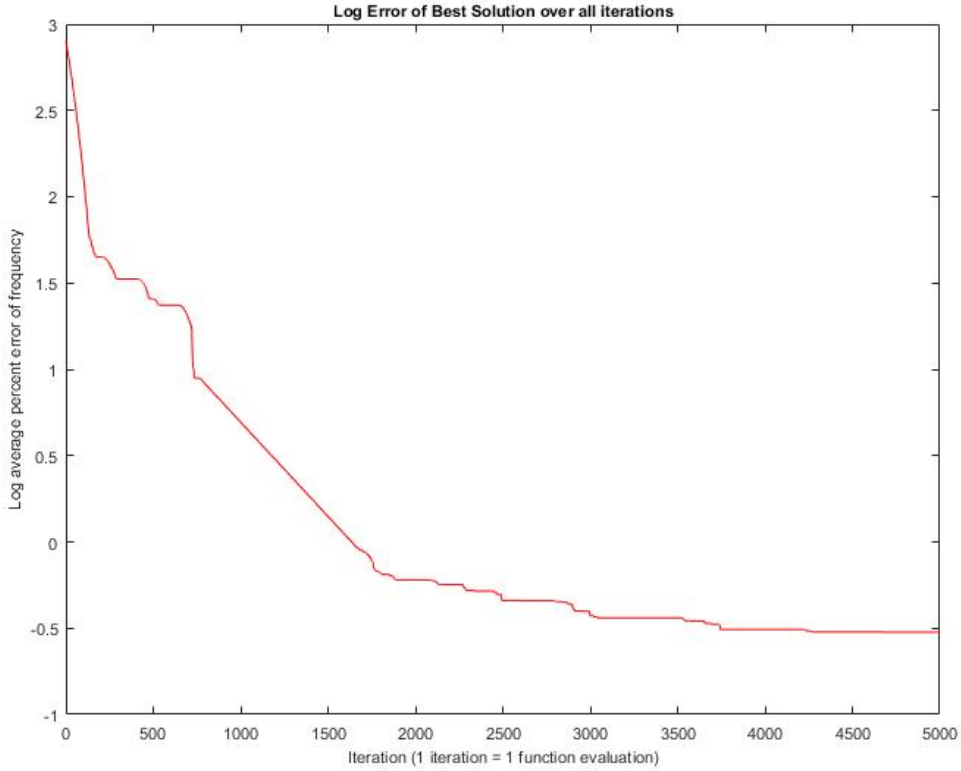


Figure 13: Best objective over all iterations

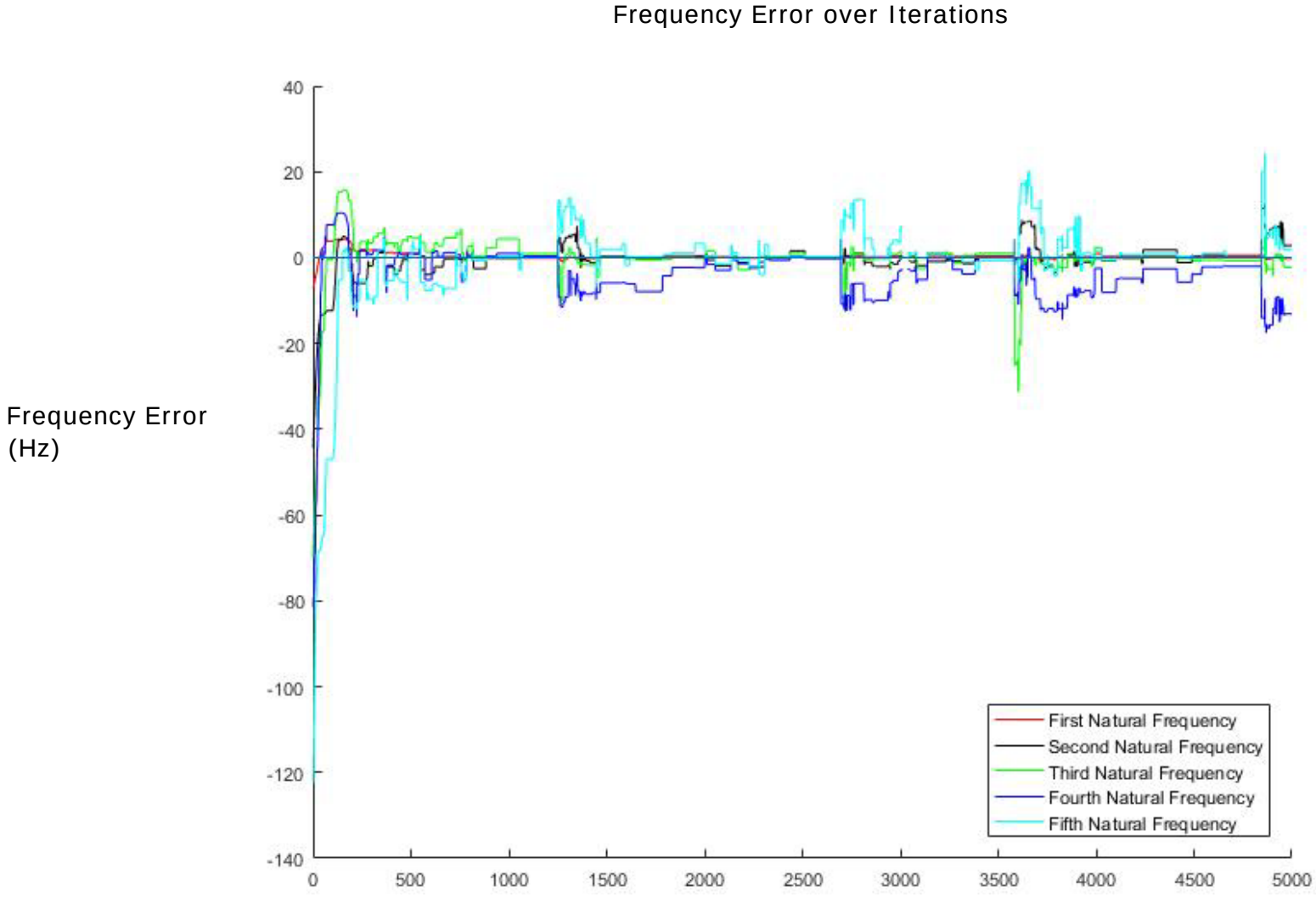


Figure 14: Frequency error over all iterations

3.5 Modal Analysis

For the three experiments, all local optima found had the same mode shapes as the target beam with some minor differences. Below are comparisons between each natural frequencies' mode shape for the target beam and the best 180 voxel solution.

The differences that can be seen from this example are some slight asymmetries in the displacements such as in figures 20 and 21. It can be observed that the target beam has a symmetrical displacement along its length but the 180 voxel beam has slightly asymmetrical displacements. Figures 24 and 25 show the same twisting mode shape but the target and result beam twist in opposite directions. This is still the same mode shape because this mode shape represents the beam alternating between twisting both ways.

The colours are a measure of displacement where blue is 0 displacement and red is maximum displacement.

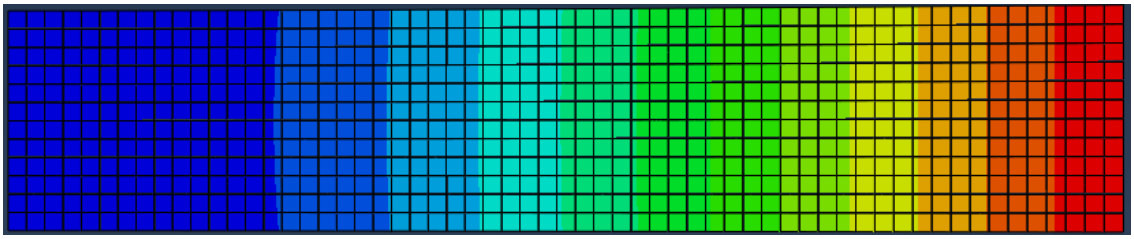


Figure 15: Target beam 1st natural frequency mode shape

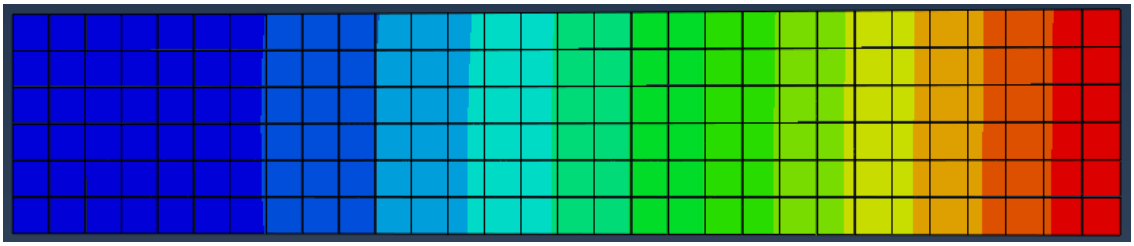


Figure 16: 180 voxel solution beam 1st natural frequency mode shape

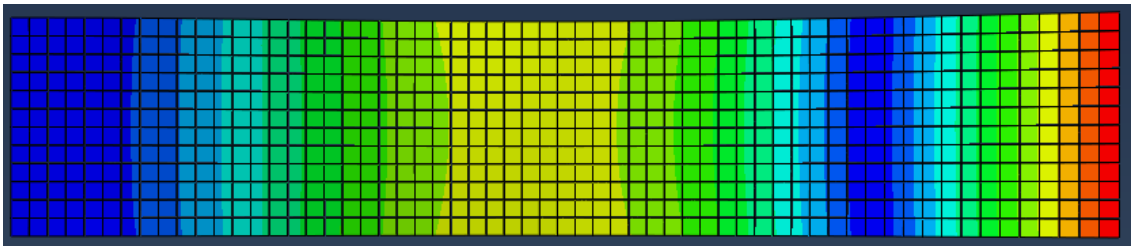


Figure 17: Target beam 2nd natural frequency mode shape

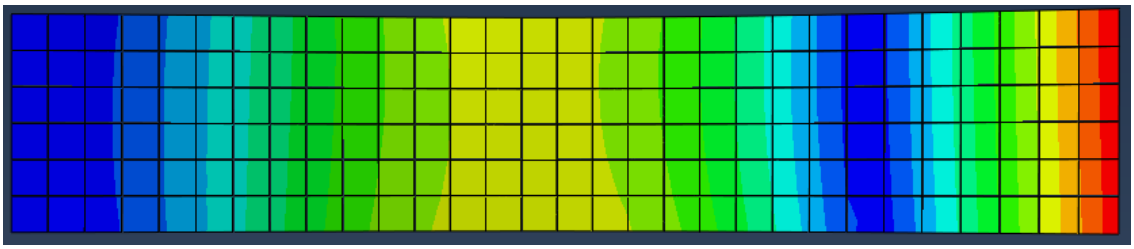


Figure 18: 180 voxel solution beam 2nd natural frequency mode shape

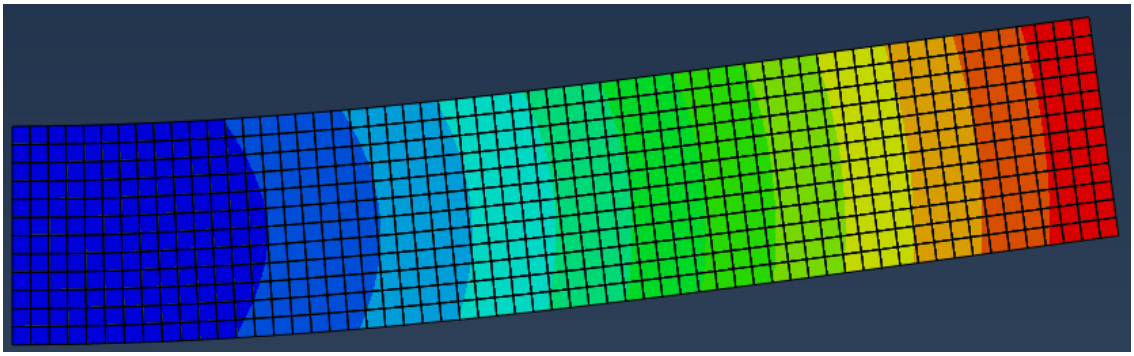


Figure 19: Target beam 3rd natural frequency mode shape

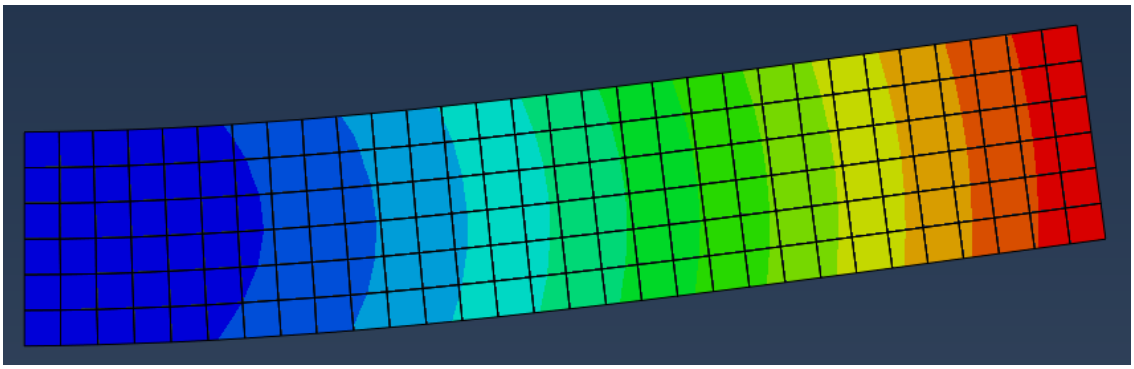


Figure 20: 180 voxel solution beam 3rd natural frequency mode shape

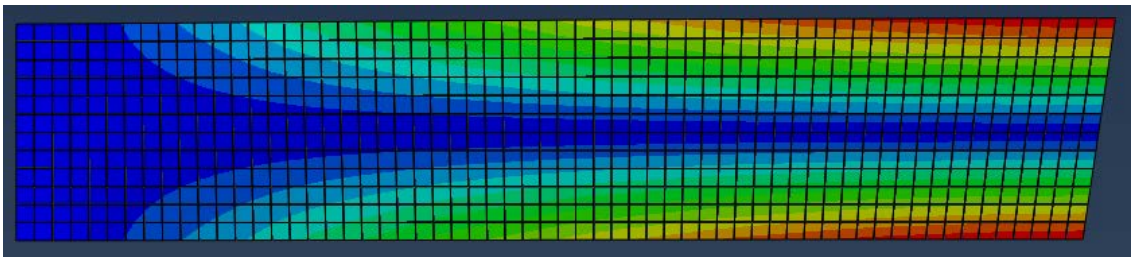


Figure 21: Target beam 4th natural frequency mode shape

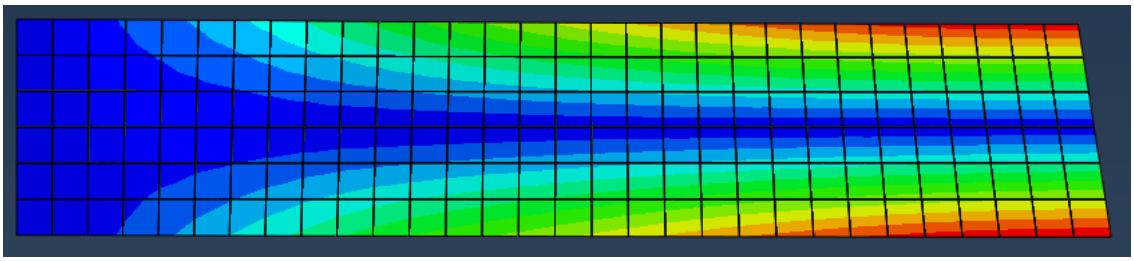


Figure 22: 180 voxel solution beam 4th natural frequency mode shape

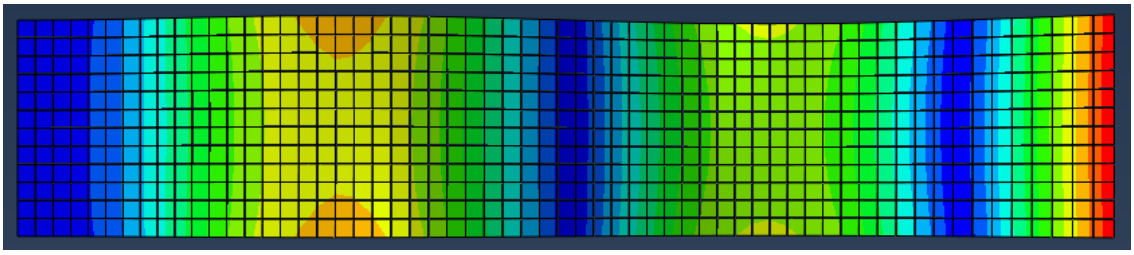


Figure 23: Target beam 5th natural frequency mode shape

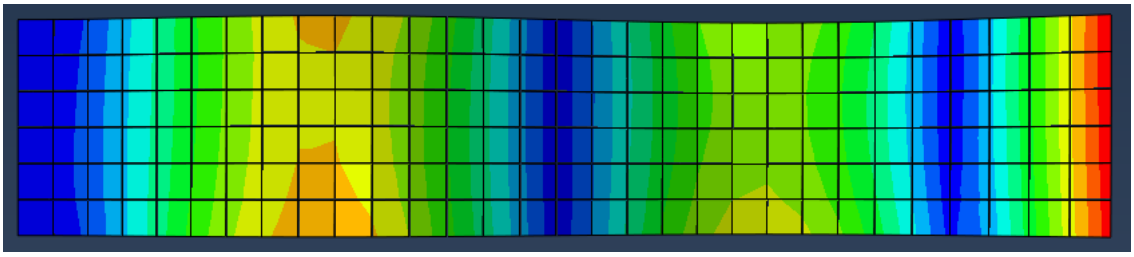


Figure 24: 180 voxel solution beam 5th natural frequency mode shape

3.6 Even Spread Metaheuristic Experiments

For the final set of experiments, the same metaheuristic was run, but with the aim of getting the first five natural frequencies 10Hz apart. This was implemented by taking the differences between consecutive natural frequencies, subtracting 10 from them and squaring the results as a measure of the errors. The sum of these errors was minimized as the objective function. The 45 voxel model was used because it ran the fastest while still giving good results.

Four experiments were done in total for this. The first used the Vero Clear and RWT-EBK 250 materials. The second and third replaced the Vero Clear with a fictitious material with 10 and 100 times the stiffness. The final experiment replaced the Vero Clear with a material 100 times as stiff and the RWT-EBK 250 with a material 1/10 of the stiffness.

Table 2 shows the frequencies and objectives of these experiments. The errors are very large for all of these experiments. The first three experiments have the closest neighbouring frequencies being 25Hz apart and the furthest being 220Hz apart. The fourth experiment has neighbouring frequencies being 40-60Hz apart

The errors go down as the Young's Moduli were made further apart so we can see that when the range between the stiffness's of the two materials increases, closer natural frequencies can be achieved.

The fourth experiment has the lowest log error at 13.2 while being the only experiment where the softer material was changed.

Figures 28-30 show the material distribution for the best solutions in these experiments where green is the stiff material and red is the other material.

Figures 31-34 show the log objective value of the best solution over all iterations for each experiment.

Stiff Material Young's Modulus	Soft Material Young's Modulus	1st Natural Frequency (Hz)	2nd Natural Frequency (Hz)	3rd Natural Frequency (Hz)	4th Natural Frequency (Hz)	5th Natural Frequency (Hz)	Log Square error of frequencies
2.8GPa	1.2GPa	41.364	239.93	367.84	393.60	627.91	16.0
28GPa	1.2GPa	41.846	239.51	364.14	451.62	609.21	15.9
280GPa	1.2GPa	42.642	241.90	340.07	461.87	609.76	15.8
280GPa	0.12GPa	13.760	75.110	110.54	151.52	197.17	13.2

Table 2: Data of optimal solutions for 10Hz spread experiment

0	0	0
1	1	0
1	1	0
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1

Figure 25: (280GPa and 0.12GPa) and (280GPa and 1.2GPa) best solution

0	0	0
1	1	0
1	1	1
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1

Figure 26: 28GPa and 1.2GPa best solution

0	0	0
1	1	0
1	1	1
0	1	0
0	0	0
0	0	0
1	1	0
1	1	1
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
0	0	0
1	1	1

Figure 27: 2.8GPa and 1.2GPa best solution

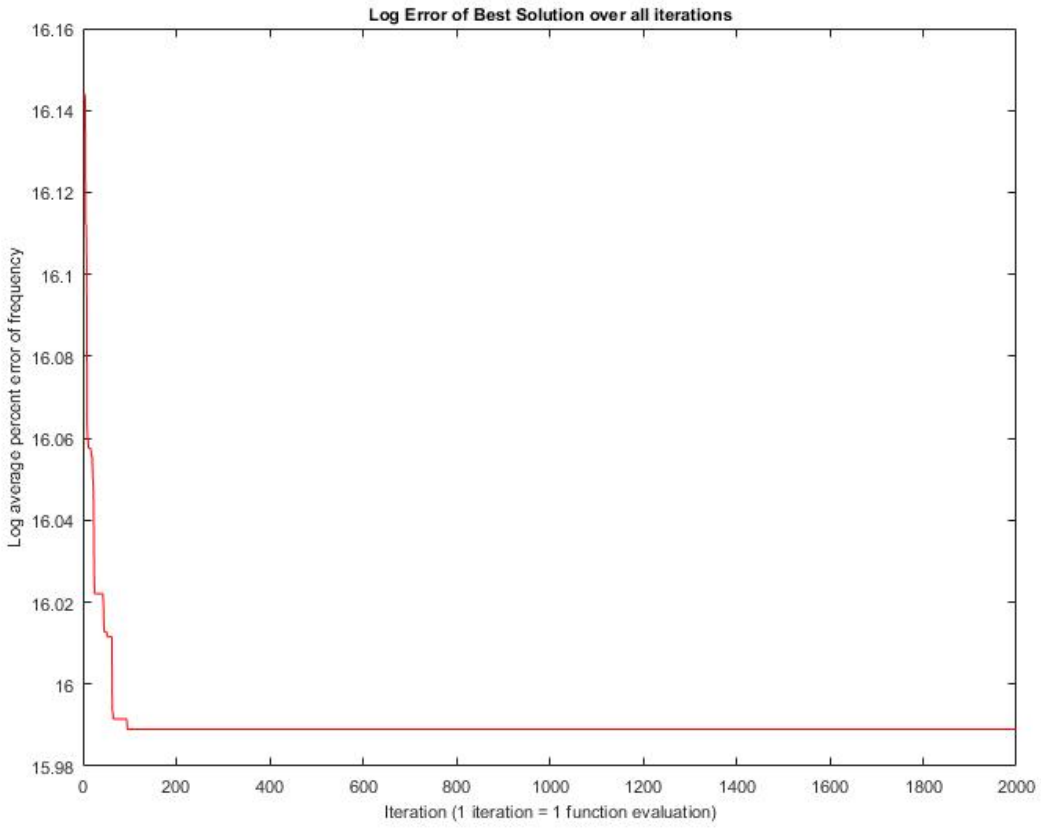


Figure 28: Experiment 1 log error over iterations

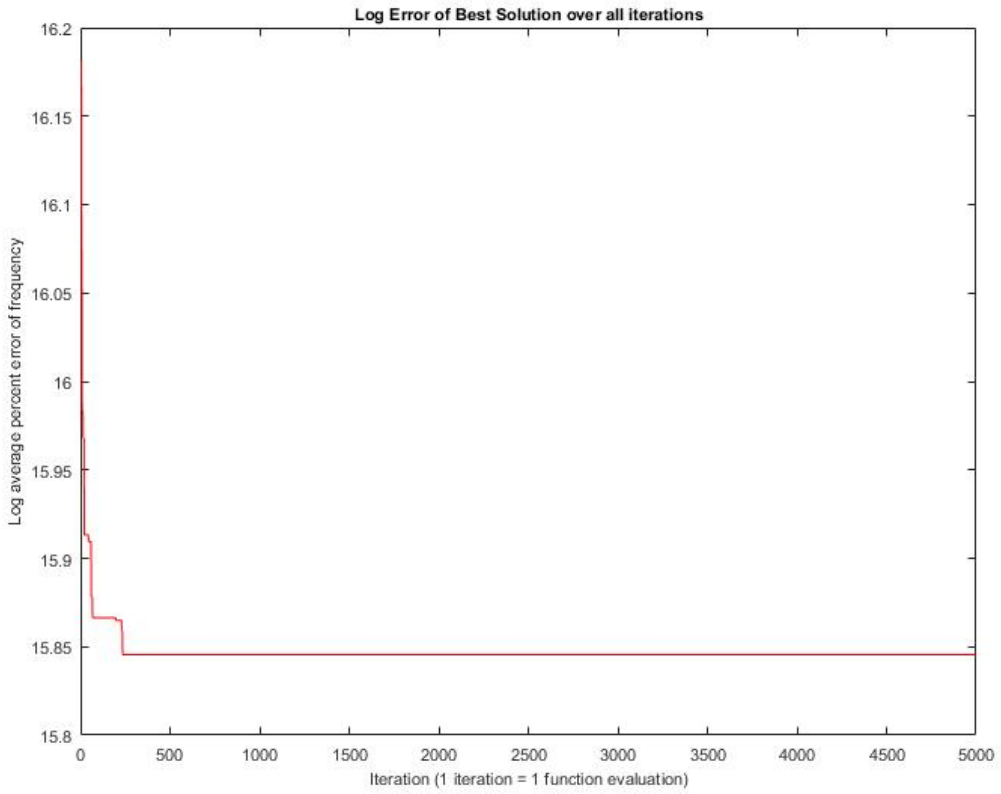


Figure 29: Experiment 2 log error over iterations

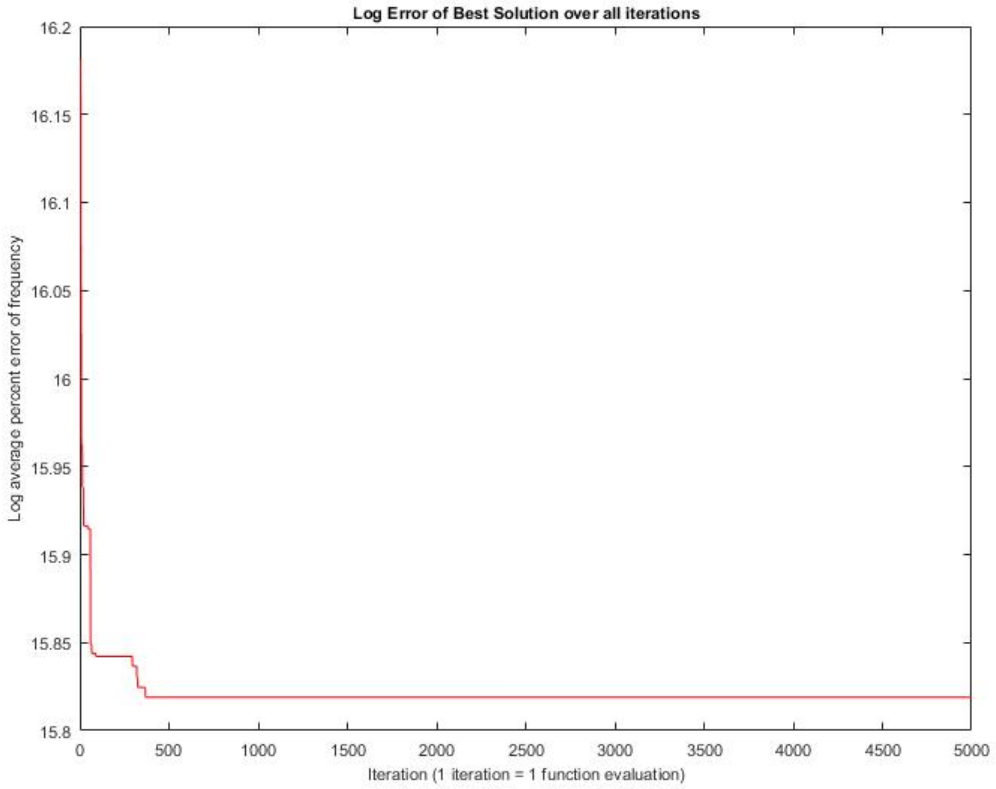


Figure 30: Experiment 3 log error over iterations

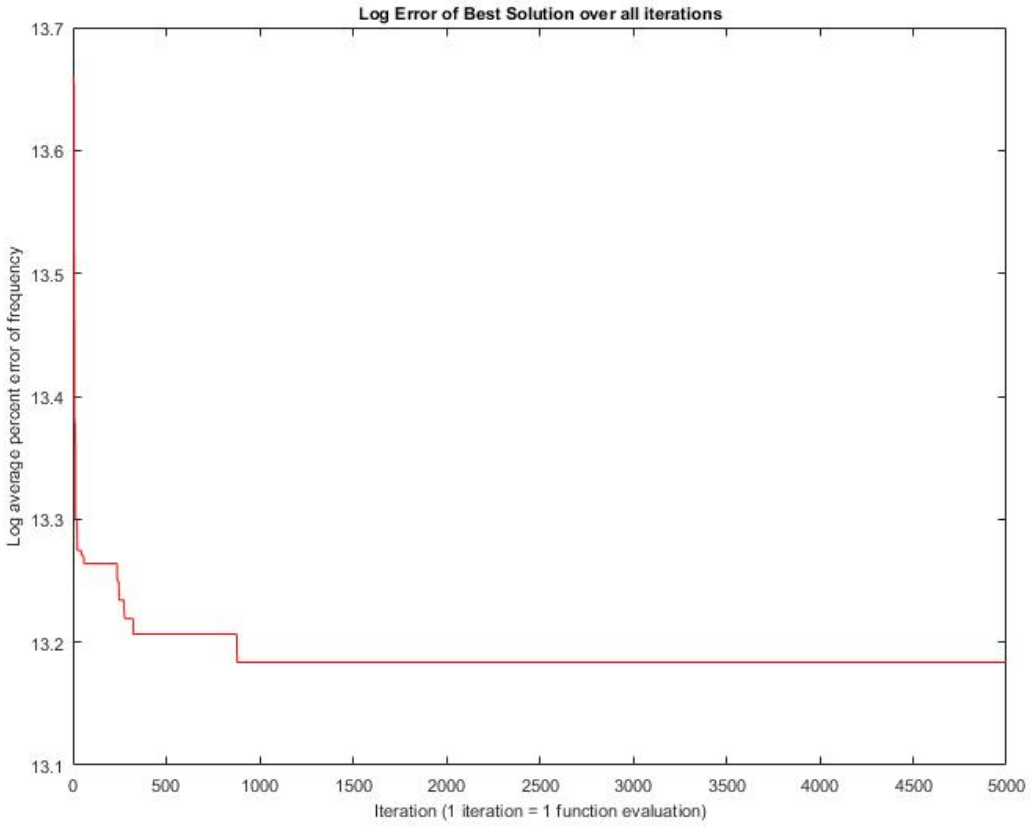


Figure 31: Experiment 4 log error over iterations

4.1 Discussion

4.1.1 Mode shapes for matching target frequencies

The mode shapes for the all of the local optima for the three experiments were the same. This indicates that for this problem, it is not possible to match the natural frequencies closely with different mode shapes. It also shows that regardless of the resolutions used, the mode shapes are the same.

4.1.2 Rate of convergence for matching target frequencies

The 45 voxel model gave 23 local optima. This was the most local optima out of all of the different resolutions showing it finds local optima the fastest. It also found its best solution in 1500 out of 5000 iterations which was the fastest out of the three resolutions.

The 180 voxel model gave the most accurate match to the target frequencies out of the different resolutions. Due to being a finer resolution than the 45 voxel model, the local optima tend to have a lower error. This is because the 180 voxel model is a less constrained version of the 45 voxel model. All of the local optima of the 45 voxel model are feasible solutions to the 180 voxel problem but could be further improved leading to lower error solutions.

The 720 voxel model performed poorly because as the number of voxels increases, the number of function evaluations required to find a local optimum also increases. This is shown in the results where the 720 voxel experiment found no local optima. This shows the accuracy gained by using a finer resolution has diminishing returns so there is no point using a model bigger than 180 voxels for this sort of problem.

4.1.3 Resulting geometries for matching target frequencies

The geometries for the beams produced by the three different resolution experiments, were roughly half and half each material with no noticeable pattern. This indicates that the positions of the voxels has much less of an effect on the natural frequency than the material assigned to them.

4.1.4 Rate of convergence for clustering frequencies

All four experiments converged to their best solution within 900 out of the 5000 iterations. The best solution does not change in the 4100 iterations that follow and the next descent heuristic works well for this model. Therefore, it is reasonable to assume that it is not possible to get much better than this and the search will remain stuck at these high error solutions if run for more iterations.

It can also be seen that the experiment where the stiffness of the soft material was lowered, the error was significantly lower. This is because lowering the stiffness soft material lowers all five natural frequencies, bringing them closer together.

4.1.5 Resulting geometries for clustering frequencies

The geometries of all four clustering experiments gave similar geometries for their best solutions (Figures 27-29). Experiment 4 with the Vero Clear and RWT-EBK 250 materials had the most voxels assigned to the stiff material. Experiment 1 with the 100 times stiffer Vero Clear and 1/10 as stiff RWT-EBK 250 had the least voxels assigned to the stiff material. This could be because as the stiff material is made stiffer, less voxels need to be assigned to achieve the same result.

The placement of the stiff voxels resembles the fifth natural frequency mode shape (Figure 25) where the stiff voxels are placed at the points of maximum displacement. This could be because the fifth natural frequency has the biggest effect on the objective function being the largest.

4.2 Limitations

The results of the frequency matching experiment were only done to match one set of frequencies. This means that these conclusions may not be applicable to other sets of frequencies and further experiments would need to be done to check that these observations are the same for different natural frequencies.

In addition to this, the quality of a match in natural frequencies is restricted by what is physically possible with the materials. For example, it is not possible to match a first natural frequency that is lower than the first natural frequency of the softer material.

4.3 Future Work

To compare the theoretical beams with their theoretical counterparts, physical multi-material beams will need to be printed. With these, the physical natural frequency experiments will need to be repeated and the natural frequencies compared with the theoretical natural frequencies.

The frequency clustering experiment did not produce satisfactory results with the current beam model. To give better results, different models will need to be tried to see if better clustering can be achieved. Possible changes include using a different shape or allowing no material to be assigned to some elements of the beam.

To further validate metaheuristics as a method for designing multi-material structures, different types of problems could also be tried. These could include trying to avoid a specified natural frequency or trying a different type of problem such as minimising weight.

Finally, since the time that the Abaqus script takes to run is the main bottleneck, it may be worth writing some dedicated finite element code for calculating the natural frequencies. Significant speed improvements could be made by removing the need to build the model, set up the problem and read from/ write to files to use the results.

5 Conclusions

In this report, the performance of a next descent metaheuristic has been demonstrated for a 45 voxel model, 180 voxel model and a 720 voxel model to match an existing periodic structure beam. When run for 5000 function evaluations, the 180 voxel model gave the lowest error although all 3 models give errors of less than 1% which is less error than the Genetic Algorithm used previously. The mode shapes remain consistent across all resolutions. However, the 45 voxel model gets to its best solution significantly faster than the 180 voxel model, so the choice between the two models depends on the error requirements of the problem.

The metaheuristic could not find a satisfactory solution for the objective of getting the natural frequencies 10Hz apart, even when fictitious materials with higher or lower stiffness's were used. It converged to a solution quickly but the solution still had a very large error. Given that the metaheuristic worked well previously, this indicates that it is not physically possible to get a beam with natural frequencies 10Hz apart with our current model.

Appendices

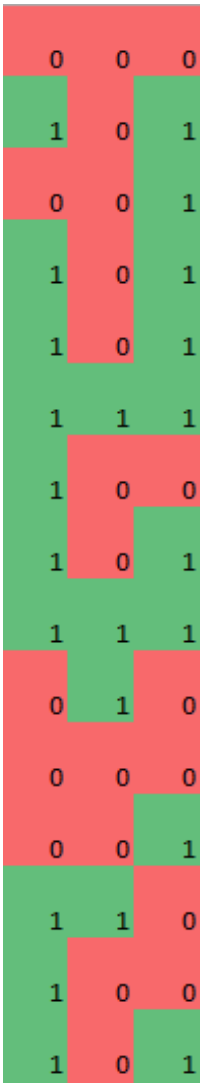


Figure 33: Best 45 Voxel Match

Figure 32: Best 180 Voxel Match

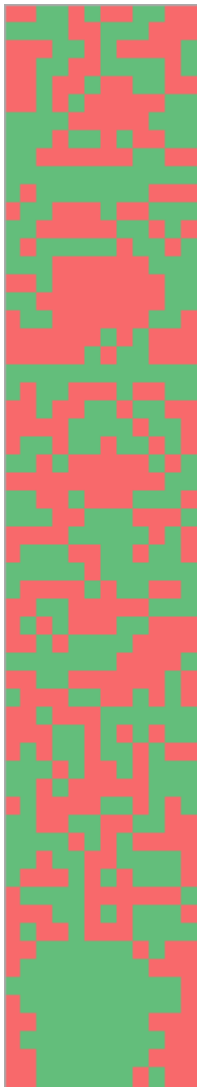


Figure 34: Best 720 Voxel Match

Python Script

```
from abaqus import *
from abaqusConstants import *
from caeModules import *
from driverUtils import executeOnCaeStartup
import os
import sys

os.chdir(r"C:\Users\nwad044\abaqus_workspace")

executeOnCaeStartup()
Mdb()



partsList = ['RubberyPart', 'StiffPart']

constraintNum = 1

integrationPoints = 5

########################################
#Change these for different geometries
########################################
voxelsPerMM = 1
execfile('Var45.py')
thickness = 0.002
width = 3
length = 15
voxelSize = 0.001/voxelsPerMM
########################################
materialArray = [[a[(j + width*i)] for j in xrange(width)] for i in
xrange(length)]



a = mdb.models['Model-1'].rootAssembly

def setupParts():
    #sketch rectangle
    s = mdb.models['Model-1'].ConstrainedSketch(name='__profile__',
sheetSize=0.005)
    g, v, d, c = s.geometry, s.vertices, s.dimensions, s.constraints
    s.sketchOptions.setValues(decimalPlaces=4)
    s.setPrimaryObject(option=STANDALONE)
    s.rectangle(point1=(0.0, 0.0), point2=(voxelSize, voxelSize))

    #make first part from sketch
    p = mdb.models['Model-1'].Part(name='StiffPart',
dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts['StiffPart']
    p.BaseShell(sketch=s)

    #make 2nd part from sketch
    p = mdb.models['Model-1'].Part(name='RubberyPart',
dimensionality=THREE_D,
        type=DEFORMABLE_BODY)
    p = mdb.models['Model-1'].parts['RubberyPart']
    p.BaseShell(sketch=s)

    #delete sketch
```

```
    s.unsetPrimaryObject()
    del mdb.models['Model-1'].sketches['__profile__']

    #make materials and sections
    mdb.models['Model-1'].Material(name='StiffMaterial')
    mdb.models['Model-1'].materials['StiffMaterial'].Density(table=((1170,
), ))
    mdb.models['Model-
1'].materials['StiffMaterial'].Elastic(table=((28200000000.0,
        0.3), ))
    mdb.models['Model-1'].HomogeneousShellSection(name='Section-1',
        preIntegrate=OFF, material='StiffMaterial', thicknessType=UNIFORM,
        thickness=thickness, thicknessField='',
idealization=NO_IDEALIZATION,
        poissonDefinition=DEFAULT, thicknessModulus=None,
temperature=GRADIENT,
        useDensity=OFF, integrationRule=SIMPSON,
numIntPts=integrationPoints)

    #make materials and sections
    mdb.models['Model-1'].Material(name='RubberyMaterial')
    mdb.models['Model-
1'].materials['RubberyMaterial'].Density(table=((1120, ), ))
    mdb.models['Model-
1'].materials['RubberyMaterial'].Elastic(table=((1160000000.0,
        0.3), ))
    mdb.models['Model-1'].HomogeneousShellSection(name='Section-2',
        preIntegrate=OFF, material='RubberyMaterial',
thicknessType=UNIFORM,
        thickness=thickness, thicknessField='',
idealization=NO_IDEALIZATION,
        poissonDefinition=DEFAULT, thicknessModulus=None,
temperature=GRADIENT,
        useDensity=OFF, integrationRule=SIMPSON,
numIntPts=integrationPoints)

    #assign sections to parts
    p = mdb.models['Model-1'].parts['StiffPart']
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
    region = p.Set(faces=faces, name='Set-1')
    p = mdb.models['Model-1'].parts['StiffPart']
    p.SectionAssignment(region=region, sectionName='Section-1', offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)
    p = mdb.models['Model-1'].parts['RubberyPart']
    f = p.faces
    faces = f.getSequenceFromMask(mask=('[#1 ]', ), )
    region = p.Set(faces=faces, name='Set-1')
    p = mdb.models['Model-1'].parts['RubberyPart']
    p.SectionAssignment(region=region, sectionName='Section-2', offset=0.0,
        offsetType=MIDDLE_SURFACE, offsetField='',
        thicknessAssignment=FROM_SECTION)

    #mesh
    p = mdb.models['Model-1'].parts['RubberyPart']
    p.seedPart(size=voxelSize, deviationFactor=0.1, minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts['RubberyPart']
    p.generateMesh()
    p1 = mdb.models['Model-1'].parts['StiffPart']
    p = mdb.models['Model-1'].parts['StiffPart']
    p.seedPart(size=voxelSize, deviationFactor=0.1, minSizeFactor=0.1)
    p = mdb.models['Model-1'].parts['StiffPart']
```

```
    p.generateMesh()

def printToTerminal(text):
    print >> sys.__stdout__, text

def getNaturalFrequency():
    #set up job
    mdb.models['Model-1'].FrequencyStep(name='Step-1', previous='Initial',
numEigen=5)
    mdb.Job(name='vibration_test45', model='Model-1', description='',
type=ANALYSIS,
        atTime=None, waitMinutes=0, waitHours=0, queue=None, memory=90,
        memoryUnits=PERCENTAGE, getMemoryFromAnalysis=True,
        explicitPrecision=SINGLE, nodalOutputPrecision=SINGLE,
echoPrint=OFF,
        modelPrint=OFF, contactPrint=OFF, historyPrint=OFF,
userSubroutine='',
        scratch='', resultsFormat=ODB, multiprocessingMode=DEFAULT,
numCpus=4,
        numDomains=4, numGPUs=0)

    mdb.jobs['vibration_test45'].submit(consistencyChecking=OFF)
    mdb.jobs['vibration_test45'].waitForCompletion()

    #read output
    ODB_name = 'C:/Users/nwad044/abaqus_workspace/vibration_test45.odb'
    odb = session.openOdb(name=ODB_name, readOnly=False)
    frames = odb.steps['Step-1'].frames
    return [frames[1].frequency, frames[2].frequency, frames[3].frequency,
frames[4].frequency, frames[5].frequency]

def tieConstraintsI(part1, part2, num):
    s1 = a.instances[part2].edges
    side1Edges1 = s1.getSequenceFromMask(mask=('[#2 ]', ), )
    region1=a.Surface(side1Edges=side1Edges1, name='m_Surf-' + str(num))
    s1 = a.instances[part1].edges
    side1Edges1 = s1.getSequenceFromMask(mask=('[#8 ]', ), )
    region2=a.Surface(side1Edges=side1Edges1, name='s_Surf-' + str(num))
    mdb.models['Model-1'].Tie(name='Constraint-' + str(num),
master=region1, slave=region2,
        positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
thickness=ON)

def tieConstraintsJ(part1, part2, num):
    s1 = a.instances[part1].edges
    side1Edges1 = s1.getSequenceFromMask(mask=('[#1 ]', ), )
    region1=a.Surface(side1Edges=side1Edges1, name='m_Surf-' + str(num))
    s1 = a.instances[part2].edges
    side1Edges1 = s1.getSequenceFromMask(mask=('[#4 ]', ), )
    region2=a.Surface(side1Edges=side1Edges1, name='s_Surf-' + str(num))
    mdb.models['Model-1'].Tie(name='Constraint-' + str(num),
master=region1, slave=region2,
        positionToleranceMethod=COMPUTED, adjust=ON, tieRotations=ON,
thickness=ON)


print "Setting up parts..."
setupParts()

partObjects = []
for i in range(len(partsList)):
    partObjects.append(mdb.models['Model-1'].parts[partsList[i]])
```

```
print "Assembling..."

instanceList = []


a.DatumCsysByDefault(CARTESIAN)

for i in range(0,length):
    print(i)
    for j in range(0,width):
        partName = 'part_' + str(i) + ',' + str(j)
        #instance = a.Instance(name=partName, part=partObjects[0],
dependent=ON)
        instance = a.Instance(name=partName,
part=partObjects[materialArray[i][j]], dependent=ON)
        instanceList.append(instance)
        instance.translate(vector=(voxelSize*i, voxelSize*j, 0))


        if(i > 0):
            prevI = 'part_' + str(i-1) + ',' + str(j)
            tieConstraintsI(partName, prevI, prevI + '-' + partName)

        if(j > 0):
            prevJ = 'part_' + str(i) + ',' + str(j-1)
            tieConstraintsJ(partName, prevJ, prevJ + '-' + partName)


        if(i == 0):
            e1 = a.instances[partName].edges
            edges1 = e1.getSequenceFromMask(mask=('[#8 ]', ), )
            region = regionToolset.Region(edges=edges1)
            mdb.models['Model-1'].EncastreBC(name='BC-' + str(j),
createStepName='Initial',
                region=region, localCsys=None)



f = getNaturalFrequency()

printToTerminal('#Frequency' + str(f[0]) + ',' + str(f[1]) + ',' +
str(f[2]) + ',' + str(f[3]) + ',' + str(f[4]))

print f

print "done"
```

Matlab Functions

```
function [ frequency ] = GetNaturalFrequency(materialArrayUnprocessed,
voxelsPerMM)

sz = size(materialArrayUnprocessed);
l = sz(1);
w = sz(2);
materialArray = zeros(l*w,1);

for i = 1:l
    for j = 1:w
        materialArray((j-1) + w*(i-1) + 1) = materialArrayUnprocessed(i,j);
    end
end

%mo='script';
mo='noGUI';

%Make python file with variables
delete('Var45.py');
fid = fopen('Var45.py', 'w');

fprintf(fid,'voxelsPerMM = %i', voxelsPerMM);
fprintf(fid,'\na = [');

fprintf(fid,'%i',materialArray(1));
for i = 2:length(materialArray)
    fprintf(fid,', %i',materialArray(i));
end


fprintf(fid,']');
fclose(fid);

[~, output] = system(['abaqus cae ',mo,'=2d_explicit_align_standard.py']);

output

index = strfind(output, '#Frequency') + 10;

frequency = str2double(strsplit(output(index:length(output)), ','));

end



function [ obj , frequencyArray] = objective( arr, target, vpmm )
%RMS Error of natural frequency from target

frequencyArray = GetNaturalFrequency(arr, vpmm);
obj = (sum(abs(frequencyArray-target)./target))/5;

end
```

Main Matlab Script

```
voxelsPerMM = 0.15;

l = 15;
w = 3;

target = [40.36, 252.73, 385.74, 403.19, 707.98];

arr = zeros(l,w);
% arr = randi([0,1],[l,w]);

bestArr = arr;
currentBest = arr;

numIterations = 5000;
bestObj = 9*ones(numIterations,1);
tm = zeros(numIterations,1);
freqArray = zeros(numIterations, 5);
lastI = l*w;

clear objective

tic
[obj, freq] = objective(arr, target, voxelsPerMM);
bestObj(1) = obj;
tm(1) = toc;
freqArray(1,:) = freq;

numLocalOptima = 1;
bestInd = 1;

newLocalSearch = false;

for i = 2:numIterations

    if(mod(i,l*w) == mod(lastI,l*w)) %found local optimum
        tm(i:end) = tm(i-1);
        bestObj(i:end) = bestObj(i-1);

        dlmwrite(['localOptimum45_', num2str(numLocalOptima),'.txt'], arr);

        numLocalOptima = numLocalOptima + 1;

        newLocalSearch = true;
        arr = randi([0,1],[l,w]);
    end

    arr = arr';
    arr(mod(i-2,l*w)+1) = 1 - arr(mod(i-2,l*w)+1);
    arr = arr';

    [obj, freq] = objective(arr, target, voxelsPerMM);

    bestObj(i) = obj;
    freqArray(i,:) = freq;

    if(bestObj(i) < bestObj(i-1) || newLocalSearch)
        newLocalSearch = false;

        lastI = i;
        currentBest = arr;
```

```
        if(bestObj(i) <= min(bestObj))
            bestInd = numLocalOptima;
            bestArr = arr;
            dlmwrite('best45.txt', bestArr);
        end
        dlmwrite('objectiveHistory45.txt', [(1:numIterations)', tm,
bestObj, freqArray]);

    else
        arr = currentBest;

        bestObj(i) = bestObj(i-1);
    end

    fprintf('%d - %g\n', i, bestObj(i));

    tm(i) = toc;
end

dlmwrite('objectiveHistory45.txt', [(1:numIterations)', tm, bestObj,
freqArray]);

plot((1:numIterations)',bestObj)
```

References

1. BoussaïD I, Lepagnot J, and Siarry P. 2013. A survey on optimization metaheuristics. *Inf. Sci.* 237 (July 2013), 82-117.
2. Arostegui M. A. Kadipasaoglu S. N. Khumawala B. M. An empirical comparison of Tabu Search, Simulated Annealing, and Genetic Algorithms for facilities location problems, In International Journal of Production Economics, Volume 103, Issue 2, 2006, Pages 742-754, ISSN 0925-5273
3. Ahuja R. K. Ergun O, Orlin J. B. Punnen J. P. A survey of very large-scale neighborhood search techniques, In Discrete Applied Mathematics, Volume 123, Issues 1–3, 2002, Pages 75-102, ISSN 0166-218X.
4. Banitalebi A. Aziz M. I. A. and Aziz Z. A. 2016. A self-adaptive binary differential evolution algorithm for large scale binary optimization problems. *Inf. Sci.* 367, C (November 2016), 487-511.
5. Panduro M. A., Brizuela C. A., Balderas L. I., and Acosta D. A., "A comparison of genetic algorithms, particle swarm optimization and the differential evolution method for the design of scannable circular antenna arrays," *Progress In Electromagnetics Research B*, Vol. 13, 171-186, 2009.
6. Lipson, H, Kurman, M, February 2013, Fabricated: The new world of 3D printing, Wiley
7. Free Vibration of a Cantilever Beam, [Online], Indian Institute of Technology, Available: http://iitg.vlab.co.in/?sub=62&brch=175&sim=1080&cnt=1, [September 2017]