A Python Based Environment For Structural Topology Optimisation

John Hutcheson

May 28, 2019

1 Introduction

This document gives an overview of the topology optimisation environment created as part of the Weir design optimisation PhD. This optimiser was created to enable new optimisation algorithms to be applied to structural topology optimisation problems.

A number of topology optimisation environments have been described in the literature. Most notably we have a topology optimisation environment for simple 2D beams, based on the SIMP technique. This was first proposed by Sigmund [6] then refined later by Andreassen [2]. Following this, a 2D MATLAB implementation of the level set method has been created by Otomori et. al. [5]. Whilst these programs are useful for understanding the techniques, it is not useful for solving real engineering problems as only very simple structures can be optimised.

Zho et. al [7] has provided a 3D topology optimisation environment in ABAQUS which uses python scripts to implement the BESO method. This environment allows topology optimisation of 3D components, however it is limited to the BESO method only.

This paper presents two optimisers which have been created to provide a customisable topology optimisation environment. The first is a 2D environment, only suitable for compliance based optimisation problems, while the second takes advantage of a commercial FEA solver and can be used for real world topology optimisation problems.

The second optimiser is based around a central Python script and ABAQUS CAE is used as the FEA solver. This configuration allows freedom in terms of the optimisation algorithms which can be implemented, and use of a commercial FEA solver means that the techniques can be applied to real engineering components. Furthermore effort has been made to optimise the program for parallel

computing in order to fully utilise the computational resources available for this project.

Section 2 gives an overview of the solver and the optimisation methodology that has been implemented. Section 3 gives a more detailed explanation of how the Python scripts work. Finally section 4 presents current issues and suggested improvements for the solver.

2 An Overview of NETO

2.0.1 Artificial Neural Network

A fully connected feed forward neural network is used to calculate the sensitivities using the NETO method. The network architecture can be seen in the image below.

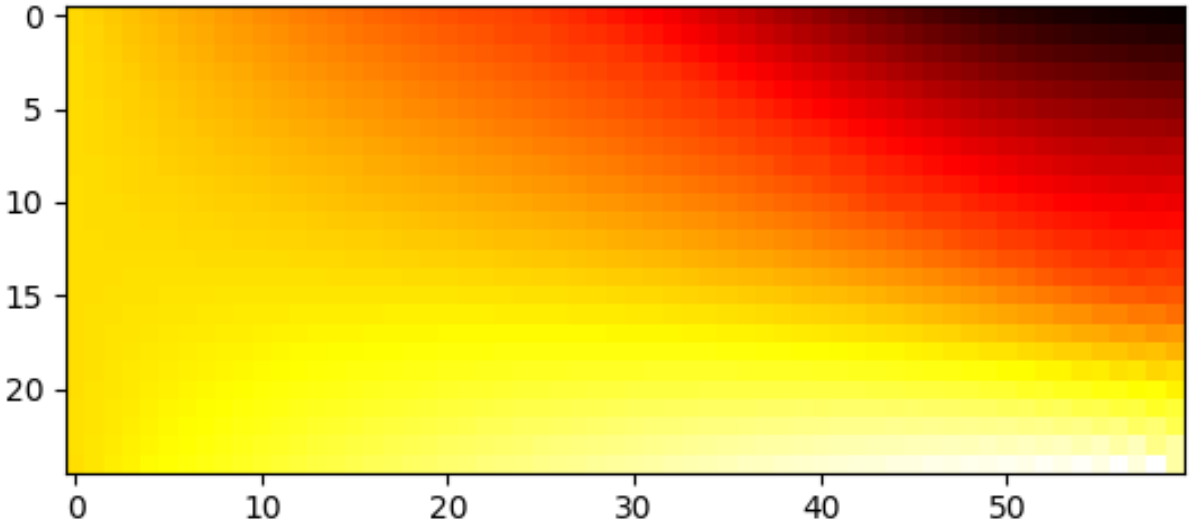
The input layer is made of 10 nodes. Each node takes design response information from the FEA analysis. The first 8 nodes take the nodal displacements of each element (2 displacements, x and y, on an element with 4 nodes gives 8 displacements). The 9th node takes the strain energy of the element, and the 10th node takes the element material density. Each of these inputs are normalised to the values that were obtained of the initial analysis of the component with a volume fraction of 1.0. The hidden layer contains 10 fully connected nodes using the sigmoid activation function. The output layer is made of one node, which outputs the sensitivity for the element. The output is fully connected to the hidden layer. The output node uses the ReLU activation function. When the network is used to generate the sensitivity for every element in the design domain, the distribution of sensitivities throughout the component can be plotted for visualisation purposes. The plot below shows an arbitrary sensitivity distribution for a 2D rectangular beam structure.

2.0.2 CMA-ES

The CMA-ES strategy was used to evolve the weights and biases of the network. CMA-ES works by evolving a distribution rather than individual parameters themselves. The strategy is initialised with a mean of 0 and a standard deviation of 0.01. A population of 18 is used for each generation, and 100 generations are run in order to arrive at a satisfactory sensitivity distribution.

The objective of the strategy is to minimise the strain energy of the structure. The total strain energy of the structure. Strain energy information is obtained from the FEA analysis. The strategy evolves a (120 x 1) array of parameters. These correspond to 110 connection weights, and 10 bias values.

Figure 1: Arbitrary sensitivity distribution for a rectangular beam element.)



3 An Overview of Python Only Optimiser

First, a python only environment was created in order to allow new algorithms to be tested on simple problems. This script was created using [2] as the starting point. There is a python version of this code available on the DTU website [1]. This script contained the Python based FEA solver necessary to evaluate each structure produced by the algorithm. This script was heavily modified in order to implement the NETO algorithm. The Keras library was used for building the neural networks, and the DEAP library was used for the CMA-ES used to evolve the weights and biases. Parallel processing was not implemented in the 2D version of the script. The code for the 2D optimisation environment can be found on Github [4]. This optimiser is contained within a single python script and can be used to test new algorithms on simple linear compliance based problems. This script implements the method described by [3] et. al. and this should be reviewed to understand how the script works.

4 An Overview of Python / ABAQUS Optimiser

4.1 Python and ABAQUS Optimisation Environment

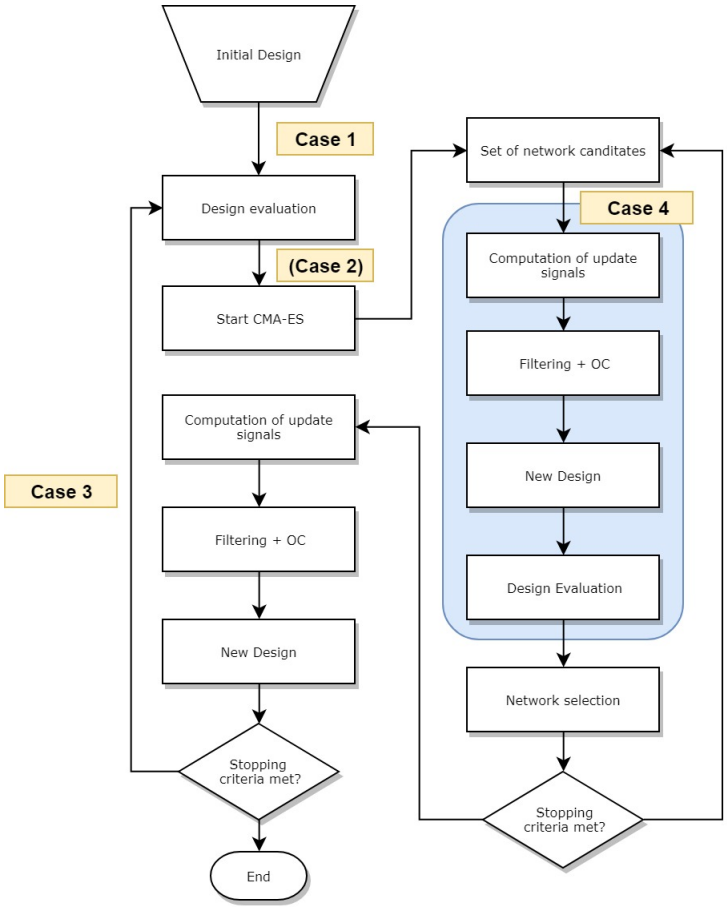
This optimisation environment is based on the work by Zuo et al. [7] however the scripts have been heavily modified and re purposed for the new environment. The main difference is that Zuo's scripts must be run within Abaqus, whereas the optimiser proposed in this paper can be run outside ABAQUS. This means that there is no limitation in terms of the python libraries available, and also it means that parallel processing can be utilised.

4.1.1 SIMP NETO Method

The SIMP-NETO method of topology optimisation has been implemented in the python script provided. The SIMP-NETO technique is based on the well established SIMP approach to topology optimisation however instead of the derivatives being calculated, they are approximated using a neural network. The neural network is trained through the process of neuroevolution. This approach means that the SIMP technique can be applied to non-linear topology optimisation problems in which the derivatives are hard to calculate, for example, stress based optimisation problems. Further details of the SIMP-NETO approach can be found in the literature [3]. Flowcharts showing the SIMP and SIMP-NETO processes can be seen in the figures below.

The flowchart below shows the SIMP-NETO method of topology optimisation. The boxes on the left hand side of the chart show the outer loop, which is identical to the SIMP method. The boxes on the right shows the neuroevolution loop, which replaces the calculation of sensitivities in the standard SIMP method.

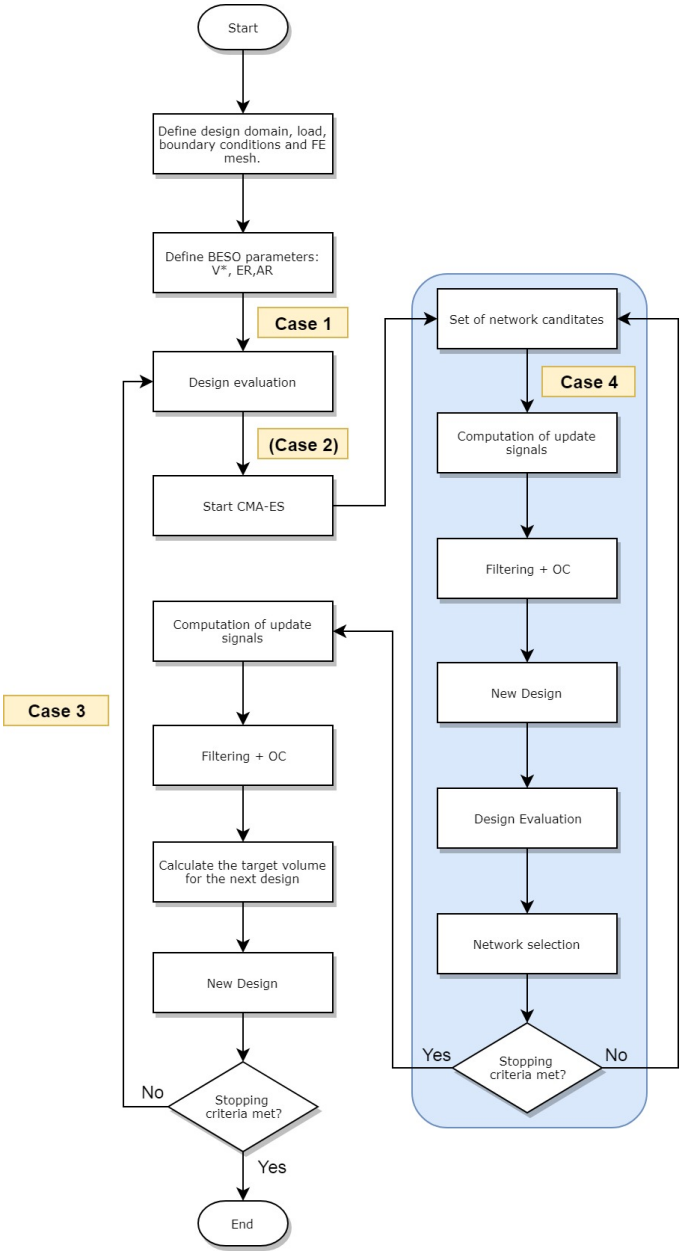
Figure 2: Flowchart for SIMP-NETO topology optimisation procedure.)



4.1.2 BESO NETO Method

As an alternative to the SIMP-NETO approach, a method based on the BESO method of topology optimisation was tested. This method is is essentially the BESO method, a discrete method of topology optimisation, however instead of derivatives being calculated, a neural network is used to approximate them, in the same way as SIMP-NETO. This can be seen on the right hand side of the flowchart below. The sensitivities are then used in the same way as in the BESO method - to determine which elements should remain solid, and which should be voids. This script can be found in this project's Github repository.[4]

Figure 3: Flowchart for BESO-NETO topology optimisation procedure.)



4.2 File Structure

This section explains the file structure used by the program. The file structure is particularly important as it ensures that the structural configuration and all relevant data is stored and labelled at each iteration of the optimisation process. It is also used to separate data accessed by the cores during parallel processing. A number of cases have been used to tell the evaluate function what stage the optimisation process is in, and to what folders results should be saved to. The cases and how they are used will be explained below.

Before the optimisation is run, the only folder present in the working directory is the 'original' folder. This folder contains all the python scripts and FEA models required to run the optimisation. When the optimisation is run for the first time, the 'original' folder is copied and saved to a folder called 'Eval_0' and an initial analysis of the FEA model is carried out. This is known as 'Case 1'.

During the CMA-ES the optimiser copies the 'Eval_X' folder and saves it in a folder named according to the process number from the operating system. This is done to ensure that each process is working from its own folder to avoid data being shared between processes. This is known as 'Case 4'.

When the CMA-ES loop has finished in the iteration, a final run is done to evaluate the fittest member of the population and to save its structure and associated data. This is known as 'Case 3'.

For iterations one onwards (iterations start at zero), the starting structure is initialised by copying the structure from the final folder as opposed to the original folder, as this contains the fittest structural configuration and design response values from the previous iteration. This is known as 'Case 2'.

Figures 2 and 3 indicate where each of the cases are used in both processes.

4.3 Python Scripts for ABAQUS

A number of Python scripts have been created for the pre and post processing operations as part of the FEA analysis. This section explains what each of these scripts is for.

4.3.1 GenerateInp4.py / GenerateInp2.py

This script is used to update the FEA model with a new material density distribution, then write an input file for the FEA analysis. The new material density distribution is read in from a csv file stored in the working directory. This file is called 'elDenNew.csv'. 'GenerateInp4.py' is used for the

SIMP-NETO process and 'GenerateInp42.py' for the BESO NETO process.

4.3.2 GenerateInp5.py / GenerateInp3.py

This script is used for pre-processing and is identical to 'generateInp4.py'/'generateInp2.py' however this script contains a function for pre-filtering the mesh. The pre-filtering is only needs to be run once at the start of the optimisation process, hence why this script is only used in 'Case 1'. Additionally, as this script is only run at the very start of the program, no element density distribution is read in and used to update the structure. Instead the optimiser creates a list of ones corresponding to every element within the part. 'GenerateInp5.py' is used for the SIMP-NETO process and 'GenerateInp3.py' for the BESO NETO process.

4.3.3 GetResults.py

This script is used to extract the design response information of a structure after the analysis has been run. Primarily this is run to extract the objective function value for the structure. This is currently the total elemental strain energy of the structure. The objective function value is saved to a file 'obj.csv' so that it can be accessed by the main python script outwith ABAQUS. The script opens the ODB and extracts the nodal displacements, density and strain energy for each element. This information is then saved to a file called 'LSF.csv'. This information is required when the initial design volume is first analysed, as it provides an input to the neural network. This script is identical for both the SIMP and BESO NETO processes.

4.4 Parallel Processing

The 2D python only optimisation environment runs on one core only. The python / ABAQUS optimiser has been structured so that it can run in parallel. This is done through use of Python's multiprocessing module. Conveniently, DEAP has multiprocessing functionality built in which enables straightforward setup of a multiprocessing pool.

When multiprocessing is enabled with DEAP, the population is grouped into a process pool. Function evaluations are split between the cores, and as each core finishes a job it takes a new one from the pool. In order for this to work the evaluate function must be self contained and it must be possible for this to be processed independently of any shared variables.

In order to achieve this, it was necessary to run a separate instance of Keras on each core, as one

instance of Tensorflow cannot be shared across processes. This is why a new network is initialised each time the function evaluation is called.

Furthermore, in order for ABAQUS to work in multiple instances, it is necessary for a new instance of ABAQUS to be opened for each different process. This can be problematic when running the program on a large number of cores as each instance of ABAQUS requires a license and the department only has 22 ABAQUS CAE licenses.

4.5 Sensitivity Filtering

It is well known that sensitivity filtering is required in topology optimisation in order to prevent checker boarding and ensure a minimum member thickness. Sensitivity filtering has been implemented in both the 2D and 3D optimisation environments. The filtering of the 2D environment is identical to that of Andreassen et. al. [2] and the 3D ABAQUS based optimiser is identical to that of Zuo et. al [7].

Note: as of the time of writing, sensitivity filtering in the 3D environment was not working correctly and needs attention.

5 Current Issues and Suggested Improvements

5.1 Python Only Solver

5.1.1 Parallel Processing

The main issue with the Python only environment is that it is quite slow, as each function evaluation is done in series. This environment would lend itself well to parallel processing. It is advised that parallel processing is implemented in the same way as it is in the Python / ABAQUS solver.

5.2 Python / ABAQUS Solver

This section presents the current problems with the Python / ABAQUS solver and suggest potential improvements

5.2.1 Sensitivity filtering not working

In the current version of the code, the sensitivity filtering does not seem to be working. The sensitivity filtering implementation is identical to that in [7], and full details of the filtering scheme can be found in this paper. A pre-filtering step is required to initialise the filtering scheme for the initial design volume. This is only done once at the start of the script. It should be noted that this pre-filtering step can take some time therefore patience is advised for this step.

5.2.2 Visualisation of final result

Another suggested area for improvement is the visualisation of the optimised geometry for the SIMP-NETO implementation. Currently it is very hard to visualise the optimised geometry with this method. This is because the density is a continuous variable , and as there is a new section for each different density value, we typically end up with a huge number of sections. In order to visualise the structure we need to delete the elements which correspond to low densities. With a large number of sections, it can be hard to determine which should be turned off, in order to view the final structure. A suggested improvement is to add a post processing step which removes all section elements below a user specified density threshold from the CAE model. The user will then be left with a mesh composed of only the solid elements.

References

[1] Topology optimization codes written in python - topopt. Available at `http://www.topopt.mek.dtu.dk/Apps-and-software/Topology-optimization-codes-written-in-Python`.

[2] E. Andreassen, A. Clausen, M. Schevenels, B. S. Lazarov, and O. Sigmund. Efficient topology optimization in matlab using 88 lines of code. *Structural and Multidisciplinary Optimization*, 43(1):1–16, Jan 2011.

[3] N. Aulig and M. Olhofer. Neuro-evolutionary topology optimization of structures by utilizing local state features. In *GECCO*, 2014.

[4] J. A. Hutcheson. Python topology optimisation environment. Available at `https://github.com/strath-ace-labs/smart-ml/tree/master/neuroevolution`.

[5] M. Otomori, T. Yamada, K. Izui, and S. Nishiwaki. Matlab code for a level set-based topology optimization method using a reaction diffusion equation. *Structural and Multidisciplinary Optimization*, 51(5):1159–1172, May 2015.

[6] O. Sigmund. A 99 line topology optimization code written in matlab. *Structural and Multidisciplinary Optimization*, 21(2):120–127, Apr 2001.

[7] Z. H. Zuo and Y. M. Xie. A simple and compact python code for complex 3d topology optimization. *Advances in Engineering Software*, 85:1 – 11, 2015.