The Development of an Advanced Composite Structure Using Evolutionary Design Methods

David van Wyk

Submission Approved for Examination

_____ _____

Supervisor Date

Dedication

This is dedicated to my mom and dad, who have supported me through thick and thin. Thank you for being there for me, and for all that you have done.

Acknowledgements

I would like to thank my supervisor, Dr David Jonson, for all his guidance and help through the course of this work. I also would like to thank Graeme Kidson and all the others at DUT who helped me.

Abstract

The development of an evolutionary optimisation method and its application to the design of an advanced composite structure is discussed in this study.

Composite materials are increasingly being used in various fields, and so optimisation of such structures would be advantageous. From among the various methods available, one particular method, known as Evolutionary Structural Optimisation (ESO), is shown here. ESO is an empirical method, based on the concept of removing and adding material from a structure, in order to create an optimum shape. The objective of the research is to create an ESO method, utilising MSC.Patran/Nastran, to optimise composite structures. The creation of the ESO algorithm is shown, and the results of the development of the ESO algorithm are presented.

A tailfin of an aircraft was used as an application example. The aim was to reduce weight and create an optimised design for manufacture. The criterion for the analyses undertaken was stress based. Two models of the tailfin are used to demonstrate the effectiveness of the developed ESO algorithm. The results of this research are presented in the study.

List of Figures

List of Tables

List of Symbols

Introduction to Composite Theory

V_f, V_m	Volume Fractions
σ_1, σ_2, τ_{12}	Stress Components
ε_1, ε_2, γ_{12}	Strain Components
E_{11}, E_{22}	Elastic Moduli
G_{12}	Shear Modulus
ν_{12}, ν_{21}	Poisson's Ratios
θ	Angle of fibre orientation
z_k	Laminate layer coordinate
t_k	Laminate layer thickness

Introduction to Optimisation

X	Design Vector
f(X)	Objective function
$g_j(X)$	Inequality constraints
$l_j(X)$	Equality Constraints
$[J_i]$	Hessian Matrix

Development of the ESO Algorithm

σ_T	Threshold Stress
i	Iteration number
RR_i	Rejection Ratio
RR_0	Original Rejection Ratio
ER	Evolutionary Rate
σ_{max}	Maximum Allowable Stress

σ_{fail}	Failure Stress
σ_{el}	Element Stress

Contents

Chapter 1

Introduction

1.1 Introduction

The modern world is fast-paced and highly competitive which demands new and better products in a seemingly insatiable market. At the same time, there is pressure to ensure that the product costs less, is more efficient and capable and has lower environmental impact, among other requirements. In order to keep up with demand, designers are resorting to various methods. Among these are the use of more advanced materials, and the use of optimisation techniques.

One of the key guiding principles of designing structures is that a design is considered complete, not when nothing more can be added, but when nothing more can be taken away. This is one of the underlying principles of optimisation. With the drive towards the use of minimum resources for modern designs, motivated by factors as diverse as cost and ecological impact, optimisation is increasingly being used to create practical designs. With the availability of powerful desktop computers, optimisation has become much easier to implement. Coupling the Finite Element Method (FEM) with optimisation routines has opened up vast new areas to optimisation methods.

At the same time, non-traditional materials for design are increasingly being used. Foremost among these is the use of composites in various forms. The use of carbon-fibre for many applications is now seen as de rigueur. These include applications as diverse as military aircraft, racing cars and golf club shafts. Their usage has increased as better manufacturing methods for

these materials are created and new applications are explored. With their excellent material properties, the use of composites looks set to expand even further into many more areas of everyday life.

With these two developments, it was foreseeable that they would be brought together. This thesis aims to do that and presents a method for optimizing a composite structure. The development of an algorithm for optimizing a composite structure will be shown, after which the application of the algorithm to a sample composite problem will be demonstrated.

1.2 Thesis Objectives

The thesis presented here describes the development of an Evolutionary Structural Optimisation (ESO) algorithm. This algorithm was to be created for use with MSC.Patran/Nastran and is designed to optimise a given composite structure, with specified loads and boundary conditions, such that mass is removed from the structure, while retaining the required level of structural strength.

The developed algorithm was applied to a test structure, and the results for the optimisation are presented. The final design is also shown for the structure.

1.3 Thesis Layout

Chapter One introduces the thesis and topic. The aims and objectives for the research into an ESO method for advanced composite structures are outlined.

The following three chapters present some of the basic theories in the various fields related to this thesis, that being composite materials, optimisation and the Finite Element Method.

The fifth chapter gives a literature review of the state of ESO and the various developments in the field. The development of the ESO algorithm is presented here, with examples presented, and problems that were encountered and solved. The final ESO algorithm is shown, with flowchart and explanation. The results of some simple optimisation examples are shown.

The application of the developed ESO algorithm to a real-world design problem is shown in Chapter Six. The design of an aircraft tailfin is used to demonstrate the application of the algorithm. Two models are presented, with different numbers of finite elements, and the results are shown from the optimisation routine. The results of the ESO optimisation study are compared and analysed. A final design is developed and shown.

Chapter Seven presents a conclusion and recommendations for future work.

Chapter 2

Introduction to Composite Theory

2.1 Introduction

The next three chapters will introduce the main areas covered in this research, and present an overview of the basic ideas and theories in each area. In this research, three areas were covered - optimisation, composite materials, and the finite element method. All three will be used to develop the final research required to achieve the specified objectives.

In the past few years, the use of composite materials has increased dramatically in many different fields [1]. Composites are being used in automotive, aeronautical, naval and civil engineering structures [2]. In 1939, the first continuous glass strands were created, which started the modern revolution in composite materials. In the last 60 years there has been a rapid advance in the development of composite materials, spurred by their increasing use in industry due to their advantageous properties [3]. This has led to much work being done, both practical and theoretical, in the development of composite material theory. In this chapter, the basic equations for composite materials are shown, and are extended to laminated composites. A brief look at composite failure is also shown.

2.2 Fundamentals of Composite Materials

Composite materials can be defined as a material with two or more distinct phases, where one phase acts as reinforcement for a second phase. There are two types of composites, particulate

and fibrous. Particulate composites consist of reinforcement particles suspended in a matrix. A general example of this is concrete. Fibrous composites consist of strands of material in a matrix material. They can either be continuous (long strands) or discontinuous (chopped strands) [4].

2.2.1 Fibre Types

There is a wide range of fibre types available for use in composites. These range from the simple, such as glass to the complex, such as ceramic. Glass fibre strands are created by drawing molten glass through tiny orifices in a gravity fed tank. These fibres are then bundled together. There are two common glass fibre reinforcements, E-glass, which is used for strength applications and S-2 glass, which is used for structural applications that require high strength and stability under extreme conditions. Ceramic fibres come in a variety of forms. Among the earliest created was boron, which was superseded by later developments. Alumina fibres are created using a slurry which is spun out to produce a yarn, which is then subjected to controlled heating. This type retains its strength at high temperatures. There are other ceramic fibres such as Silicon Carbide which are used for specific applications. Aramid is an organic fibre, first traded by the du Pont Company under the trade name Kevlar. Aramid fibres tend to have high tensile strength, but an average Young's Modulus. Carbon fibre covers both carbon and graphite reinforcements. Graphite contains more than 99% carbon content, while carbon ranges from 80 to 95% but for the purpose of this section, carbon fibre will refer to both. The carbon filaments are created from a precursor, either polyacrylonitrile (PAN), pitch or rayon. The precursor undergoes a process that involves controlled pyrolysis, which is chemical decomposition by heat. The fibre properties are determined by the manufacturing process, with strength increasing as the manufacturing temperature is increased.

The table below gives a summary of some of the fibre material properties, with a comparison to some common metals:

Material	Young's Modulus(GPa)	Tensile Strength (MPa)	Poisson's Ratio ν	Density ρ (g/cm^3)
Steel	200	1724	0.32	7.8
Aluminium	69	483	0.33	2.7
Titanium	91	758	0.36	4.5
E-Glass	69	3450	0.22	2.58
S-2 Glass	86.8	4585	0.23	2.46
Boron	385	3799	0.21	2.6
Alumina	379	1585	0.25	3.95
Kevlar 49	124	3620	0.34	1.44
Carbon High Strength	295	5600	0.2	1.74
Carbon High Modulus	690	3300	0.2	2.17

Table 2.1: Properties of Fibre Material [4]

2.2.2 Matrix Materials

In a composite, the matrix is the binding material that gives the composite its structure. It supports, protects and distributes the load to the reinforcement fibres. The matrix enables the redistribution of the load paths as the fibres break. The matrix material tends to have lower material properties than the reinforcement material - stiffness, strength, and density are diminished. There are various types of matrix material - these include metals, ceramics and polymers.

Metals used include copper, aluminium and titanium. The reasons for choosing metal matrices include the ability to reach high temperatures, strength and toughness. However, metal matrices are susceptible to corrosion and degradation between the matrix and the reinforcement.

Ceramic matrix materials have low density and are capable of handling high temperatures. Typical materials are carbon, silicon carbide and silicon nitride. However they are also brittle, making them susceptible to flaws and cracks.

Polymers as matrix materials are by far the most common type used. They can be divided into two types, thermoplastics and thermosets. Thermoplastics can be reshaped with heat and pressure, as they soften when their temperature is increased. Thermosets do not soften upon reheating. Thermosets tend to have better mechanical properties than thermoplastics, as well as increased corrosion resistance, thermal stability and toughness, but thermoplastics are suited for high volume, low cost processing. Problems with polymer matrices include a limited temperature range, susceptibility to corrosion from environmental factors, and thermal factors such as differences between coefficients of thermal expansion between matrix and reinforcement.

2.2.3 Composite Formats

There are various ways of forming a composite. Some of the more common methods are:

- Unidirectional Lamina - This is the simplest form of lay-up. The reinforcement fibres all lie orientated in the same direction. The lamina is strongest in the direction of the fibres, with the strength in the other directions dependent on the matrix material and matrix/fibre bond. The properties of a unidirectional lamina are orthotropic, but if the lamina is thick enough, the properties in the transverse plane i.e. perpendicular to the fibres, may be isotropic, in which case the laminate is transversely isotropic.

- Woven fabric lamina - The flexible fibre reinforcements can be woven into a cloth fabric, which can then be shaped and set with the matrix material. There are a variety of weave patterns available.

- Laminates - These are created by stacking layers of lamina, with varying fibre angles. The properties of the laminate vary with the thickness and orientations of the lamina chosen.

- Chopped Fibre - In chopped fibre composites, the reinforcement fibres are cut into short lengths. These short strand composites can be used in moulding processes, to create com-

plex shapes. These have many of the same advantages of continuous strand composites, such as stiffness and strength.

2.3 Mechanics of Composite Materials

2.3.1 Fibre Volume Fraction

The fibre volume is the fraction of fibre to resin, by volume. This fraction is used to determine the strength and stiffness properties of a composite laminate. The fibre provides the stiffness and strength, and so higher fibre content is desirable. The fraction, expressed as a percentage, ranges from 20% to 70%, depending on the application and manufacturing process. A fraction of over 70% is disadvantageous, as it means the fibre will typically not be able to wet out fully, meaning that the mechanical properties could be reduced by a significant margin.

2.3.2 Rule of Mixtures

The simplest method for determining the properties of a unidirectional composite is called the rule of mixtures. This is done by combining the individual properties of the materials used based on the volume fractions of each. If a composite has a fibre volume fraction V_f and a matrix volume fraction V_m, then the following equation must be satisfied:

$$V_f + V_m = 1 \tag{2.1}$$

Any property, p, of the composite can be estimated as

$$p_f V_f + p_m V_m = p \tag{2.2}$$

where p_f and p_m are the constituent properties of the fibre and matrix. This can be seen in the simple case of density, where the density of the composite is given by

$$\rho_f V_f + \rho_m V m = \rho_c \tag{2.3}$$

where ρ_c is the composite density. For values such as the Young's Moduli, the rule of mixtures can be applied in a similar method. For the longitudinal stiffness, E_1, the composite could be represented as a pair of springs connected in parallel [5]. With ε_m, the strain in the matrix and ε_f, the strain in the fibre, being equal, using the rule of mixtures gives

$$E_f V_f + E_m V_m = E_1 \tag{2.4}$$

For transverse stiffness E_2, the composite can be represented as a pair of springs in series. This gives the total strain in the composite as the sum of ε_m and ε_f, and so the rule of mixtures gives

$$\frac{V_f}{E_f} + \frac{V_m}{E_m} = \frac{1}{E_2} \tag{2.5}$$

2.3.3 Basic Stress-Strain Relationships

The properties of composite materials make them very attractive materials for use in engineering applications. However, it must be remembered that the properties are direction sensitive [6]. A unidirectional material is much stronger in the direction of the fibres, while transverse properties will be lower and depend on the matrix and the interface between the fibre and the matrix. Taking into account the directional effects, the stress-strain relationships are given by:

$$\varepsilon_1 = \frac{\sigma_1}{E_{11}} - \nu_{21}\frac{\sigma_2}{E_{22}} \tag{2.6}$$

$$\varepsilon_2 = \frac{\sigma_2}{E_{22}} - \nu_{12}\frac{\sigma_1}{E_{11}} \tag{2.7}$$

$$\gamma_{12} = \frac{\tau_{12}}{G_{12}} \tag{2.8}$$

where ε_1 is the strain in the longitudinal direction, ε_2 is the strain in the transverse direction, γ_{12} is the shear strain, ν_{12} is the major Poisson's Ratio, which provides the strain in the transverse direction subject to strain applied in the longitudinal direction, ν_{21} is the minor Poisson's Ratio, the converse of ν_{12}, E_{11} is the elastic modulus in the longitudinal direction, E_{22} is the elastic modulus in the transverse direction, and G_{12} is the shear modulus in the 1-2

plane. The relationship between ν_{12} and ν_{21} is given by:

$$\frac{\nu_{12}}{E_{11}} = \frac{\nu_{21}}{E_{22}} \tag{2.9}$$

This shows that ν_{12} is greater than ν_{21} as expected, since E_{11} is greater than E_{22}. Rearranging these equations to get stresses in term of strain, the equations become:

$$\sigma_1 = \frac{E_{11}\varepsilon_1}{1 - \nu_{12}\nu_{21}} + \frac{\nu_{21}E_{11}\varepsilon_2}{1 - \nu_{12}\nu_{21}} \tag{2.10}$$

$$\sigma_2 = \frac{E_{22}\varepsilon_2}{1 - \nu_{12}\nu_{21}} + \frac{\nu_{12}E_{22}\varepsilon_1}{1 - \nu_{12}\nu_{21}} \tag{2.11}$$

Rewriting 2.6, 2.7 and 2.8 in matrix form it becomes:

$$\varepsilon_{12} = S\sigma_{12} \tag{2.12}$$

where $\varepsilon_{12} = \begin{bmatrix} \varepsilon_1 & \varepsilon_2 & \gamma_{12} \end{bmatrix}^T$ and $\sigma_{12} = \begin{bmatrix} \sigma_1 & \sigma_2 & \tau_{12} \end{bmatrix}^T$. S is known as the compliance matrix and is given by:

$$S = \begin{bmatrix} \frac{1}{E_{11}} & \frac{-\nu_{21}}{E_{22}} & 0 \\ \frac{-\nu_{12}}{E_{11}} & \frac{1}{E_{22}} & 0 \\ 0 & 0 & \frac{1}{G_{12}} \end{bmatrix}$$

Writing 2.10, 2.11 and 2.8 in matrix form gives:

$$\sigma_{12} = Q\varepsilon_{12} \tag{2.13}$$

Where Q is given by:

$$Q = \begin{bmatrix} \frac{E_{11}}{1-\nu_{12}\nu_{21}} & \frac{E_{11}\nu_{21}}{1-\nu_{12}\nu_{21}} & 0 \\ \frac{E_{22}\nu_{12}}{1-\nu_{12}\nu_{21}} & \frac{E_{22}}{1-\nu_{12}\nu_{21}} & 0 \\ 0 & 0 & G_{12} \end{bmatrix}$$

It can be seen that $Q = S^{-1}$ using matrix properties.

2.3.4 Lamina Loading

Laminates typically consist of multiple layers, often orientated at differing angles. Considering a single layer with the principal axes rotated through angle θ, as shown in Figure 2-1.

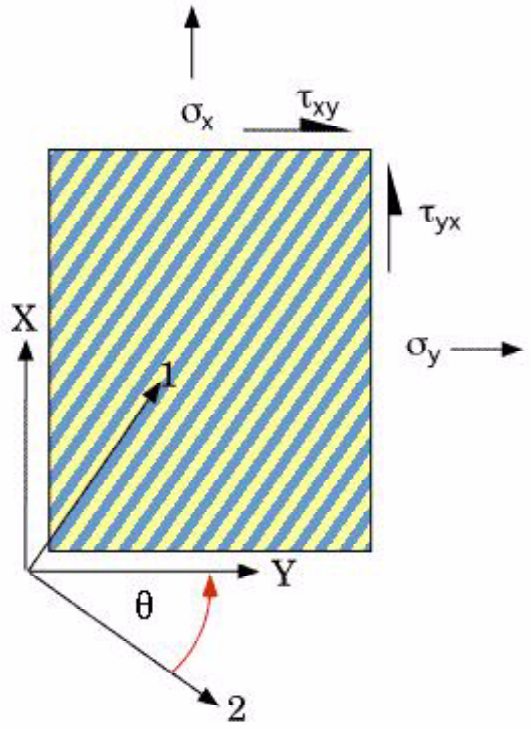


Figure 2-1: Rotation of axes through angle θ in a unidirectional lamina

Relating the stresses and strains in the x and y directions with the principal axes of the laminate, the stresses are expressed as:

$$\sigma_{12} = T\sigma_{xy} \tag{2.14}$$

and

$$\bar{\varepsilon}_{12} = T\bar{\varepsilon}_{xy} \tag{2.15}$$

where $\bar{\varepsilon}_{12} = \begin{bmatrix} \varepsilon_1 & \varepsilon_2 & \frac{1}{2}\gamma_{12} \end{bmatrix}^T$ and $\bar{\varepsilon}_{xy} = \begin{bmatrix} \varepsilon_x & \varepsilon_y & \frac{1}{2}\gamma_{xy} \end{bmatrix}^T$, and T is the transformation

matrix, given by:

$$T = \begin{bmatrix} m^2 & n^2 & 2mn \\ n^2 & m^2 & -2mn \\ -mn & mn & (m^2 - n^2) \end{bmatrix}$$

where $m = \cos(\theta)$ and $n = \sin(\theta)$. Translating the above equations and combining gives:

$$\sigma_{xy} = \bar{Q}\varepsilon_{xy} \tag{2.16}$$

The matrix Q, known as the transformed stiffness matrix, is given by:

$$\bar{Q}_{11} = Q_{11}m^4 + 2(Q_{12} + 2Q_{33})n^2m^2 + Q_{22}n^4$$

$$\bar{Q}_{22} = Q_{11}n^4 + 2(Q_{12} + 2Q_{33})n^2m^2 + Q_{22}m^4$$

$$\bar{Q}_{12} = (Q_{11} + Q_{22} - 4Q_{33})n^2m^2 + Q_{12}(m^4 + n^4)$$

$$\bar{Q}_{13} = (Q_{11} - Q_{12} - 2Q_{33})nm^3 + (Q_{12} - Q_{22} + 2Q_{33})mn^3$$

$$\bar{Q}_{23} = (Q_{11} - Q_{12} - 2Q_{33})mn^3 + (Q_{12} - Q_{22} + 2Q_{33})m^3n$$

$$\bar{Q}_{33} = (Q_{11} + Q_{22} - 2Q_{33} - 2Q_{12})n^2m^2 + Q_{33}(m^4 + n^4)$$

Inverting 2.16 gives strains in terms of stresses, that is:

$$\varepsilon_{xy} = \bar{S}\sigma_{xy} \tag{2.17}$$

where $\bar{S}$ is the transformed compliance matrix and the components of which are given by:

$$\bar{S}_{11} = S_{11}m^4 + 2(S_{12} + 2S_{33})n^2m^2 + S_{22}n^4$$

$$\bar{S}_{22} = S_{11}n^4 + 2(S_{12} + S_{33})n^2m^2 + S_{22}m^4$$

$$\bar{S}_{12} = (S_{11} + S_{22} - S_{33})n^2m^2 + S_{12}(m^4 + n^4)$$

$$\bar{S}_{13} = (2S_{11} - 2S_{12} - S_{33})nm^3 + (2S_{12} - 2S_{22} + S_{33})mn^3$$

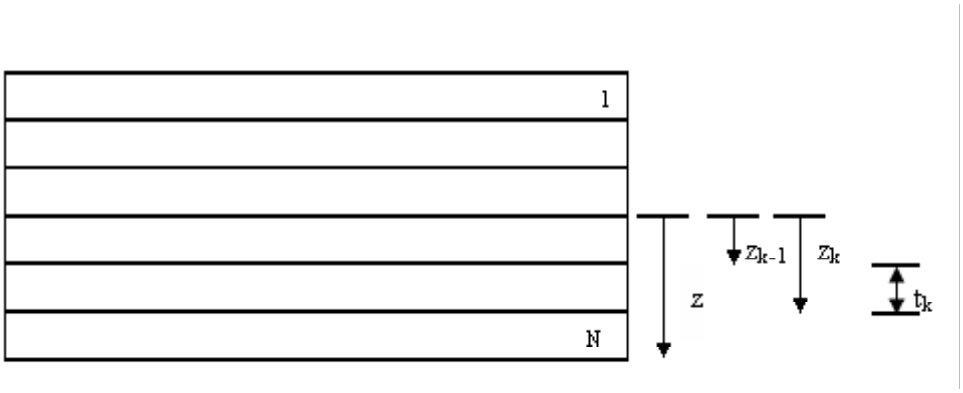


Figure 2-2: Layering of plies in the laminate

$$\bar{S}_{23} = (2S_{11} - 2S_{12} - S_{33})mn^3 + (2S_{12} - 2S_{22} + S_{33})m^3n$$

$$\bar{S}_{33} = 2(2S_{11} + 2S_{22} - S_{33} - 4S_{12})n^2m^2 + S_{33}(m^4 + n^4)$$

2.4 Lamination Theory

The equations describing the linear elastic response of a laminated composite subjected to in-plane loads and bending moments are shown below.

2.4.1 Notation

The laminate is arranged such that the z-direction is taken to be perpendicular to the plane of the laminate and positive in the downward direction. The origin is located at the midplane of the laminated plate, centred between top and bottom surfaces. The individual layers, or lamina, are numbered 1 to N from top to bottom. The layers each have an orientation θ_k. The thickness of each layer is given by $t_k = z_k - z_{k-1}$, where z_k is the z-coordinate of the bottom of layer k, and $z_{k-1}$the top. This is shown in Figure 2-2

2.4.2 Assumptions for Laminate Theory

Laminate theory is based on certain assumptions:

1. The laminate consists of perfectly bonded layers, also known as laminae.

2. Each layer is a homogenous material, with known properties.

3. Individual layers can vary in property, being either isotropic, orthotropic or transversely isotropic.

4. Each layer is in a state of plane stress.

5. The laminate deforms according to the following Kirchoff assumptions:

   - Normals to the midplane remain straight and normal to the midplane after deformation.
   - Normals to the midplane do not change length.

2.4.3 Strain-Displacement Relationships

Taking the Kirchoff assumption that the normals remain straight and normal to the deformed midplane gives that the shear strains γ_{zx} and γ_{zy} be zero. Further, the assumption that the normals to the midplane do not change length gives that the z-displacement of the midplane be a function of x and y only, that is $w = w(x, y)$. This then gives that $\varepsilon_z = 0$.

Considering the angle α, assuming the angles are small, this gives

$$\tan\alpha = \frac{\partial w}{\partial x} \cong \alpha \tag{2.18}$$

The displacement of an arbitrary point in the x-direction is given by the sum of the midplane displacement, u^0, plus the displacement due to the rotation of the normal about the midplane. This gives

$$u = u^0 - z\tan\alpha = u^0 - z\alpha = u^0 - z\frac{\partial w}{\partial x} \tag{2.19}$$

Similarly in the y-direction, with displacement v,it gives

$$v = v^0 - z\frac{\partial w}{\partial y} \tag{2.20}$$

Since the normals do not change length, w, the plate deflection, is independent of z and is given by

$$w(x, y) = w^0(x, y) \tag{2.21}$$

where the superscript 0 refers to the midplane. The above equations, when combined with the strain-displacement equations, give:

$$\varepsilon_x = \frac{\partial u}{\partial x} = \frac{\partial u^0}{\partial x} - z\frac{\partial^2 w}{\partial x^2} = \varepsilon_x^0 + z\kappa_x$$

$$\varepsilon_y = \frac{\partial u}{\partial y} = \frac{\partial u^0}{\partial y} - z\frac{\partial^2 w}{\partial y^2} = \varepsilon_y^0 + z\kappa_y \tag{2.22}$$

$$\gamma_{xy} = \left(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}\right) = \frac{\partial u^0}{\partial x} + \frac{\partial u^0}{\partial y} - 2z\frac{\partial^2 w}{\partial x \partial y} = \gamma_{xy}^0 + z\kappa_{xy}$$

where κ is the defined as the curvature and is given by:

$$\kappa_x = -\frac{\partial^2 w}{\partial x^2}$$

$$\kappa_y = -\frac{\partial^2 w}{\partial y^2} \tag{2.23}$$

$$\kappa_{xy} = -2\frac{\partial^2 w}{\partial x \partial y}$$

Combining the equations into matrix form gives:

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{Bmatrix} \varepsilon_x^0 \\ \varepsilon_y^0 \\ \gamma_{xy}^0 \end{Bmatrix} + z \begin{Bmatrix} \kappa_x \\ \kappa_y \\ \kappa_{xy} \end{Bmatrix} \tag{2.24}$$

which can be written as:

$$\{\varepsilon\}_x = \{\varepsilon\}_x^0 + z\{\kappa\}_x \tag{2.25}$$

this gives the total strains $\{\varepsilon\}_x$ at any location in the laminate in terms of the midplane strains, $\{\varepsilon\}_x^0$, and the curvatures, $\{\kappa\}_x$. This is shown in Figure 2-3.

2.4.4 Laminate Stresses

The stresses at any z-location can be obtained by substituting the strains, as shown in 2.25, into 2.16, which gives:

$$\{\sigma\}^k = [\bar{Q}]^k \left\{\varepsilon^0\right\} + [\bar{Q}]^k z\{\kappa\} \tag{2.26}$$

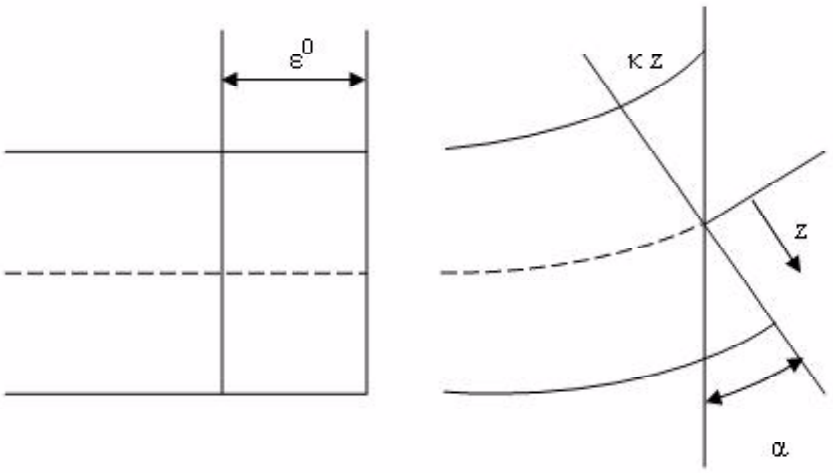


Figure 2-3: The two components of laminate deformation: ε^0, the in-plane deformation, and κz, the bending deformation

which is the stresses in the kth layer of the laminate. Note that the reduced stiffness matrix, $\left[\bar{Q}\right]^k$, varies with the fibre orientation of each layer. It can also be seen that $\{\varepsilon^0\}$ and $\{\kappa\}$ are independent of z.

2.4.5 In-Plane Forces and Moments

For the in-plane forces per unit length, these are defined as the through-thickness integrals of the planar stresses in the laminate. This gives

$$N_x = \int_{-H}^{H} \sigma_x dz$$

$$N_y = \int_{-H}^{H} \sigma_y dz \tag{2.27}$$

$$N_{xy} = \int_{-H}^{H} \tau_{xy} dz$$

Combining them gives

$$\{N\} = \int_{-H}^{H} \{\sigma\}\, dz \tag{2.28}$$

Substituting 2.26 into the above equation gives

$$\{N\} = \int_{-H}^{H} \left[\bar{Q}\right]^k \left\{\varepsilon^0\right\} dz + \int_{-H}^{H} \left[\bar{Q}\right]^k \{\kappa\}\, z\, dz \tag{2.29}$$

Since $\{\varepsilon^0\}$ and $\{\kappa\}$ are independent of z, the equation can be rewritten as

$$\{N\} = \sum_{k=1}^{n} \left(\int_{z_{k-1}}^{z_k} \left[\bar{Q}\right]^k dz \right) \left\{\varepsilon^0\right\} + \sum_{k=1}^{n} \left(\int_{z_{k-1}}^{z_k} \left[\bar{Q}\right]^k z\, dz \right) \{\kappa\} \tag{2.30}$$

since the integral over the laminate thickness can be replaced by a summation of the integrals over the thicknesses of the individual layers. This can be written as

$$\{N\} = [A] \left\{\varepsilon^0\right\} + [B] \{\kappa\} \tag{2.31}$$

and since the stiffness, $\left[\bar{Q}\right]^k$, is constant in each layer, $[A]$ and $[B]$can be defined as

$$[A] = \sum_{k=1}^{N} \left[\bar{Q}\right]^k (z_k - z_{k-1}) \tag{2.32}$$

$$[B] = \frac{1}{2} \sum_{k=1}^{N} \left[\bar{Q}\right]^k \left(z_k^2 - z_{k-1}^2\right) \tag{2.33}$$

where $[A]$ represents the in-plane stiffness and $[B]$ represents the bending-stretching coupling.

For the moments per unit length, the definition is the integral of the differential force multiplied by the moment arm, integrated over the laminate thickness. This gives

$$\begin{aligned} M_x &= \int_{-H}^{H} \sigma_x z\, dz \\ M_y &= \int_{-H}^{H} \sigma_y z\, dz \\ M_{xy} &= \int_{-H}^{H} \tau_{xy} z\, dz \end{aligned} \tag{2.34}$$

As before, it can be condensed to

$$\{M\} = \int_{-H}^{H} \{\sigma\}\, z\, dz \tag{2.35}$$

Substituting 2.26 gives

$$\{M\} = [B]\left\{\varepsilon^0\right\} + [D]\{\kappa\} \tag{2.36}$$

where $[D]$, the bending stiffness matrix, is given by

$$[D] = \frac{1}{3}\sum_{k=1}^{N}\left[\bar{Q}\right]^k\left(z_k^3 - z_{k-1}^3\right) \tag{2.37}$$

These equations, 2.31 and 2.36 can be combined to give

$$\begin{Bmatrix} N \\ M \end{Bmatrix} = \begin{bmatrix} A & B \\ B & D \end{bmatrix}\begin{Bmatrix} \varepsilon^0 \\ \kappa \end{Bmatrix} \tag{2.38}$$

This shows the coupling between the bending and stretching response, reflected in the $[B]$ matrix.

2.5 Composite Failure

In composite materials, failure is a difficult term to define. A typical composite laminate will sustain multiple local failures before the final large- scale rupture into multiple parts. Local failure is referred to as damage, and the process of damage accumulation is termed damage mechanics. Composite materials can fail in various ways. Failure can be split into failure at the matrix/fibre interface, or micro, level, and failure at the laminate level [4]. At the matrix/fibre level, there can be failure of the fibre or the matrix, if the maximum allowable tensile stress for either component is exceeded. Fibre/matrix debonding can occur, and can be accompanied by fibre pullout if fibre fracture occurs. If the composite is under compressive loading, fibre kinking could occur. At the level of laminate, failure could occur with transverse cracks appearing in planes parallel to the fibres, or fibre-dominated failures perpendicular to the fibre. These failures are the result of micro-level failures. Delamination is the failure of the composite between two laminae, where the composite splits between the two layers.

2.5.1 Laminate Failure

In the development of a mathematical model for laminate failure, certain assumptions are made. It is assumed that all laminae are homogeneous, orthotropic materials, with known strength properties in the principal material directions. Also, the shear strength in the plane of the fibres is independent of the sign of the shear stress in the principal material directions, as shear stress, positive and negative, are identical in terms of the stress parallel and perpendicular to the composite fibres. It must be noted that the sign independence of the shear strength only holds for the principal axes. For off-axis fibre orientations, the shear strengths may vary across a wide range. For the failure theories that are shown here, it is assumed that the laminate fails when the first ply fails. However, this does not always mean that the composite structure has failed. In a typical laminate, there may be multiple localised failures before total failure occurs.

2.5.2 Maximum Stress Theory

This failure criterion operates under the assumption that failure occurs when a component of stress reaches it maximum limiting value, no matter what the other stresses may be. This can be written as

$$\begin{aligned} X_c < \sigma_1 < X_T \\ Y_c < \sigma_2 < Y_T \\ Z_c < \sigma_3 < Z_T \\ |\tau_{23}| < Q \\ |\tau_{31}| < R \\ |\tau_{12}| < S \end{aligned} \tag{2.39}$$

where X is the ultimate normal strength in the fibre direction, Y and Z the ultimate normal strengths in the transverse directions, with C and T indicating compression and tension, And Q, R, and S are the ultimate shear stresses. In order for this criterion to be applied, the allowable limits must be determined using testing.

2.5.3 Maximum Strain Theory

In a similar method to the Maximum Stress Theory, strain can be used as a criterion to determine failure. This can be represented by

$$\begin{aligned} \varepsilon_1^C < \varepsilon_1 < \varepsilon_1^T \\ \varepsilon_2^C < \varepsilon_2 < \varepsilon_2^T \\ \varepsilon_3^C < \varepsilon_3 < \varepsilon_3^T \\ |\gamma_{12}| < \Gamma_{12} \\ |\gamma_{13}| < \Gamma_{13} \\ |\gamma_{23}| < \Gamma_{23} \end{aligned} \tag{2.40}$$

where Γ_{ij} is the maximum allowable shear strains.

2.5.4 Tsai-Hill Criterion

There are a variety of failure criteria that utilise quadratic functions of the stresses to provide better correlation between theoretical and experimental results. This criterion utilises Hill's anisotropic plasticity theory and applies it to the failure of homogenous, isotropic materials. The failure equation is given by

$$(G+H)\sigma_1^2+(F+H)\sigma_2^2+(F+G)\sigma_3^2-2H\sigma_1\sigma_2-2G\sigma_1\sigma_3-2F\sigma_2\sigma_3+2N\tau_{12}^2+2M\tau_{13}^2+2L\tau_{23}^2=1 \tag{2.41}$$

where F, G, H, L, M and N are material strength parameters. The values of F, G, H, L, M and N are expressed in terms of the failure stresses, and can be derived from some basic assumptions. Assuming pure shear stress loading, with $\tau_{12} \neq 0$, and with a corresponding shear strength S, and all other stress equal to zero, it becomes

$$2N = \frac{1}{S^2} \tag{2.42}$$

Similarly,

$$2L = \frac{1}{Q^2} \tag{2.43}$$

$$2M = \frac{1}{R^2} \tag{2.44}$$

F, G and H are derived from simultaneous equations obtained from 2.41, using a similar method, and this gives

$$2F = -\frac{1}{X^2} + \frac{1}{Y^2} + \frac{1}{Z^2} \tag{2.45}$$

$$2G = \frac{1}{X^2} - \frac{1}{Y^2} - \frac{1}{Z^2} \tag{2.46}$$

$$2H = \frac{1}{X^2} + \frac{1}{Y^2} - \frac{1}{Z^2} \tag{2.47}$$

Chapter 3

Introduction to the Finite Element Method

3.1 Introduction

The Finite Element Method is an important tool for the modern engineer, providing, as it does, a technique to analyse complex structures efficiently and effectively. This method grew out of earlier techniques and has been quickly developed into a viable means of getting results, with its integration with the modern computer. This means that an engineer can take a complex component and model it, and obtain the required results. In this section the derivation of the Finite Element Method, and the modelling of composite materials using this technique, is shown.

3.2 History of the Finite Element Method

The Finite Element Method (FEM) arose out of the need to accurately analyse complex structures. Before FEM, classical (continuum) methods were used to solve stress analysis problems. These methods had developed over many decades, and provided displacement and stress results using various methods. However, in order to use the classical methods, many assumptions needed to be made. These included simplification of structures into two or one dimensions, the use of isotropic materials and so on. As structures grew more complex, the equations to model

them became more complex, and so more sophisticated mathematical techniques were required to solve them. Often this resulted in inaccurate solutions. The typical design process then became a vastly simplified model which was used as a starting point to the design process. A prototype would then be built and tested, and the final design would be developed from that.

An alternative method was needed, and numerical methods started to be increasingly used to find close approximate solutions. One of the more commonly used was the general finite difference scheme [7], [8]. However, this method, although capable of solving complex structures, was hampered by problems with irregular or unusual boundary conditions. The finite element method grew out of the finite difference scheme, and became widely adopted, first in the aerospace industry to study stresses, and then in other industries as its uses grew. It has been expanded to dynamic structures, heat transfer, and fluid flow.

The finite element method can be explained as the breaking up of the structure to be analysed into a finite number of parts, or elements, then analysing each part separately and the reassembling the structure to obtain the final results. In order to do this, nodes are used to link the elements together. This can be considered a piecewise polynomial interpolation. This means that a set of simultaneous algebraic equations is generated. With more complex structures, the equations become more numerous and complicated, hence the requirement and use of computers to solve them.

3.3 Basic Operation of the Finite Element Method

The basic principle of finite element analysis is the discretisation of a structure into elements. This reduces the problem to a finite number of unknowns. From this, the final solution can be defined in terms of assumed approximating functions for each of the elements. These functions, also known as interpolation functions, are defined at the nodes of the element, which are usually placed on the element boundary, although it is possible to have interior nodes as well. The nodal values now become the new unknowns for the mathematical representation of the structure. The next step is the selection of the interpolation functions. These functions need to fulfil certain criteria, and are typically chosen so that the field variables, or its derivatives, are continuous

across the element boundaries. The degree of approximation depends on size and the number of elements, as well as the interpolation functions chosen.

Finite Element Analysis is now a computer based technique, which typically implements the following steps:

1. The first step is to break down the structure into elements. There are a number of different types of elements, and the selection of which to use is based on the type of problem being solved.

2. The interpolation functions need to be chosen and applied to the structure, using the nodes assigned.

3. The properties for the individual elements are expressed using matrix equations calculated using the interpolation functions.

4. The matrix equations for the element properties are assembled to give the overall system equations. This enables the overall behaviour of the structure to be analysed.

5. The system equations are then solved to give the final results.

The finite element method offers many advantages in solving structural problems. One of the principal advantages is that it can handle irregular shapes without difficulty. Also, the materials used can be anisotropic or non-homogenous. Another advantage is that the model can be meshed (divided into elements) with different sized elements, which is useful when there are locations of high stress, so those areas would have a greater number of elements. Any type of load and boundary condition can be modelled, being replaced by equivalent nodal loads and boundary conditions.

3.4 Theoretical Formulation

3.4.1 Continuum Problems

Continuum problems are based on the idea that all processes are given by field values that are defined at every point in space. The independent variables in continuum problems are

the coordinates of time and space. Examples of continuum problems are those that involve temperature, electromagnetic fields, stress and displacement. All these problems arise from properties in Nature that are given by partial differential equations and specified boundary conditions.

Continuum problems are sometimes called boundary value problems, since their solution is often wanted for a particular region specified by a boundary, on which boundary conditions are imposed. Boundaries are either open or closed. Open boundaries extend to infinity, and no boundary conditions are specified on the part at infinity [9]. Closed boundaries are those where conditions affecting the solution of the problem are specified everywhere.

3.4.2 Problem Statement

Consider some domain D bounded by the surface Σ. Let ϕ be a scalar function defined in the interior of D such that the behaviour of ϕ in D is given by

$$L\left(\phi\right) - f = 0 \tag{3.1}$$

where f is a known scalar function of the independent variables and L is a linear or nonlinear differential operator. It can be assumed that the physical parameters in the differential operator are known constants or functions. In n dimensions, second order differential operators can typically be reduced, using suitable transformations, to the form

$$L\left(\right) = \sum_{i=1}^{n} A_i \frac{\partial^2 \left(\right)}{\partial x_i^2} + \sum_{i=1}^{n} B_i \frac{\partial \left(\right)}{\partial x_i} + \left(\right) C + D \tag{3.2}$$

where Ai, Bi, C, and D may be functions. The operator as given in 3.2 may be linear if Ai, Bi, C, and D are functions only of the independent variables $(x_1, x_2, ..., x_n)$ and quasi-linear if Ai, Bi, C, and D are functions of x_i and the dependent parameter, as well as first derivatives of the dependent parameter. An operator is linear if

$$L(f + g) = L(f) + L(g) \tag{3.3}$$

3.4.3 Methods for Solving Continuum Problems

It can be seen that the problem is finding the unknown function ϕ that satisfies 3.1, as well as the boundary conditions specified for surface Σ. Solutions to this problem range from completely analytical to completely numerical. Some methods are listed below:

Exact Solutions

- Separation of variables
- Similarity solutions
- Fourier/Laplace Transformations

Approximate Solutions

- Perturbation
- Power Series
- Probability Schemes
- Finite difference method
- Ritz Method
- Finite element method

3.4.4 The Variational Approach

Continuum problems often have different differential and variational formulations. However these formulations are equivalent, as will be shown below. The differential equation formulation involves the integration of a differential equation (or system of equations) subject to given boundary conditions. For the variational formulation, the problem entails finding the unknown function (or functions) that either extremise or make stationary a functional such as $I(\phi)$ or a system of functionals subject to the same boundary conditions. These two formulations are equivalent since the functions that satisfy the differential equations and the boundary conditions also extremise or make stationary the system of functionals.

The variational formulation has advantages in the obtaining of an approximate solution, over the differential formulation. These are:

1. The functional, which may represent some actual physical quantity in the problem, contains derivatives of a lower order than that of the differential operator. This means that an approximate solution can be sought in a larger class of functions.

2. There may be what is known as reciprocal variational formulations, where one functional must be minimised and the other must be maximised.

3. Complicated boundary conditions can be treated as natural boundary conditions.

4. The existence of a solution can be proven using the calculus of variations.

When applicable, this approach can be very useful, however a variational statement for the continuum problem must be formulated, which means that the continuum problem must be posed in a variational form. The variational methods are some of the oldest methods of obtaining solutions to continuum problems.

3.4.5 The Ritz Method and the Finite Element Method

The Ritz method can be considered a predecessor to the finite element method. The Ritz method is one of the general methods for obtaining approximate solutions to variational form problems. This method consists of assuming the form of the unknown solution, in terms of trial functions with unknown parameters, which are adjustable. The method involves substituting the trial functions into the functional and then expressing the functional in terms of the adjustable parameters. The functional is then differentiated with respect to each parameter, and the resulting equations are all set equal to zero. For n parameters, there will be n simultaneous equations to be solved for those parameters. The choice of the trial functions will determine the accuracy of the approximate solution. The exact solution is given if it is contained in the set of trial solutions. It can be shown that the approximation improves in accuracy as the number of trial functions and adjustable parameters is increased. The process of including more and more trial functions leads to a series of approximate solutions that converges to the true solution.

The finite element method can be considered as a subset of the Ritz method, with the interpolation method involved satisfying certain requirements. The main difference is that the trial functions in the finite element method are not defined over the whole solution domain. They also only have to satisfy certain continuity conditions, not the boundary conditions. In the Ritz method the functions are defined over the whole solution domain, so it can only be used for relatively simple geometric shapes. In the finite element method, these functions have the same limitations, but they apply only to the element directly. Since elements can be assembled to create complex geometric shapes, the finite element method is far more versatile.

3.4.6 General Definition of an Element

The mathematical definition of a finite element needs to be generalised so that it is expressed in less physical terms. Elements are only interconnected at node points on the boundaries of elements. Generally, in solid mechanics, elements do not deform or change shape. Instead, they are defined as regions where a displacement field exists. The nodes are located in space where the displacement or its derivatives are known or sought. The finite element mesh can be interpreted as a spatial subdivision rather than a material subdivision .

Once the finite element mesh for the solution domain has been created, the behaviour of the unknown field value over each element can be approximated. This is done using a continuous function, expressed in terms of the nodal values of its derivatives, up to a certain degree. These functions are the interpolation functions. The collection of the interpolation functions over the whole solution domain gives a piecewise approximation to the unknown field variable.

3.4.7 Element Equations from the Variational Principle

The task is to find the solution that makes the functional stationary, by determining the nodal values of ϕ. In order to achieve this, it is required that

$$\partial I\left(\phi\right) - \sum_{i=1}^{n} \frac{\partial I}{\partial \phi_i} \delta\phi_i = 0 \tag{3.4}$$

where n is the number of discrete values of ϕ assigned to the solution domain. Since the $\delta\phi_i$'s are independent, 3.4 can be satisfied only if

$$\frac{\partial I}{\partial \phi_i} = 0, \quad i = 1, 2, ..., n \tag{3.5}$$

The functional $I(\phi)$ may be written as a sum of individual functionals that are defined for all elements of the assemblage, i.e.

$$I(\phi) = \sum_{e=1}^{M} I^{(e)} \left(\phi^{(e)}\right) \tag{3.6}$$

where M is the total number of elements and (e) denotes an element. From 3.6, it can be seen that

$$\delta I = \sum_{e=1}^{M} \delta I^{(e)} = 0 \tag{3.7}$$

where the variation of $I^{(e)}$ is only with respect to the nodal values associated with element (e). 3.7 shows that

$$\left\{\frac{\partial I^{(e)}}{\partial \phi}\right\} = \frac{\partial I}{\partial \phi_j} = 0, \quad j = 1, 2, ..., r \tag{3.8}$$

where r is the number of nodes for element (e). 3.8 gives a system of r equations that describe the behaviour of element (e). Equation 3.8 can be written in alternate form as

$$\left\{\frac{\partial I^{(e)}}{\partial \phi}\right\} = [K]^{(e)} \{\phi\}^{(e)} - \{F\}^{(e)} = \{0\} \tag{3.9}$$

where $[K]^{(e)}$ is the square matrix of constant stiffness coefficients, $\{\phi\}^{(e)}$ is the column vector of nodal values, and $\{F\}$ is the vector of resultant nodal forces [10]. The complete set of equations can be written in symbolic form as

$$\frac{\partial I}{\partial \phi_i} = \sum_{e=1}^{m} \frac{\partial I^{(e)}}{\partial \phi_i} = 0, \quad i = 1, 2, ..., n \tag{3.10}$$

or as

$$\left\{\frac{\partial I}{\partial \phi}\right\} = \{0\} \tag{3.11}$$

To solve the problem, the set of n equations is solved simultaneously for the n nodal values of ϕ . If there are q nodes in the solution domain where ϕ is specified by the given boundary

conditions, there will be $n-q$ equations to be solved for the $n-q$ unknowns.

3.4.8 Interpolation Function Requirements

The approximate solutions converge to the final, correct result as an increasing number of elements are used. This is known as refining the mesh. There are three conditions that define the process of mesh refinement:

1. Elements must be made smaller so that every point in the solution domain can always be inside an element, no matter how small the element may be.

2. All the previous meshes must be contained in the refined meshes.

3. The form of the interpolation functions must remain unchanged throughout the process of refining the mesh.

In order to guarantee that there is monotonic convergence as described, and to make the assembly of the individual equations meaningful, the interpolation functions $N^{(e)}$ in the expressions

$$\phi^{(e)}\left[N^{(e)}\right]\{\phi\}^{(e)}, \quad e=1,2,..,M \tag{3.12}$$

are required, where $[N^{(e)}]$ is the row vector of the interpolation functions, that are functions of the coordinate value of the nodes and $\{\phi\}^{(e)}$ is the column vector. An important concept is the degree of continuity of a field variable at element interfaces. If the field variable is continuous at the element interfaces it is said to have C^0 continuity. For field variables that are continuous across element boundaries for their first derivatives, there is C^1 continuity, if second derivatives are continuous, there is C^2 continuity etc. The functions that appear in the integrals in the element equations contain derivatives up to the $(r+1)$th order. For convergence, 3.12 is specified to match the following requirements:

1. Compatability requirements - At element interfaces there must be C^r continuity. This means that at the element interfaces, the field variable ϕ, and any of its partial derivatives, up to one order less than the highest order derivative appearing in $I(\phi)$, must be continuous. Elements whose interpolation functions satisfy this are known as compatible elements.

2. Completeness requirement - Within an element there must be C^{r+1} continuity. All uniform states of ϕ and its partial derivatives up to one order less than the highest order derivative appearing in $I(\phi)$ should have representation in $\phi^{(e)}$ when, in the limit, the element size shrinks to zero. Elements with interpolation functions complying with this are known as complete elements.

These requirements were derived by Felippa and Clough [11], and confirmed by Oliviera [12]. These two requirements hold no matter whether the element equations were derived using the Variational, Galerkian or other methods. Finally, there is the requirement that the field variable representation within an element, and the polynomial representation for that element, remain unchanged under a linear transform from one Cartesian coordinate system to another. Polynomial representations that exhibit such invariant properties are said to possess geometric isotropy.

3.4.9 Domain Discretisation

The finite element solution starts off with discretising the continuum, i.e. dividing it up into a series of elements. There are a variety of elements available for use. Typically only one type is used, but other types can be added in depending on the structure being modelled. The mesh is often graded i.e. the nodes are not equally spaced. This is useful when the boundary is irregular, and in areas of complex geometry, while smooth regions would have a more regular mesh. High stress areas should also have a finer mesh, so that the results are accurate. Care must be taken to ensure that elements are not distorted i.e. the ratio of the element's smallest dimension to its largest should be near unity. Long narrow elements may introduce directional biases, which would lead to incorrect results. Convergence testing is also a useful tool. This involves the creation of several different meshes for the same structure, with a varying number of elements. By comparing the results, it is possible to see if enough elements are being used to model the structure and obtain the correct solution.

3.4.10 Basic Element Shapes

Given a continuum or solution domain of any shape, it can be accurately modelled using a range of simple shaped finite elements. There are a range of shapes available, from one dimensional bars, to three dimensional blocks.

In one dimension, with one independent variable, the elements are line segments. The typical number of nodes is two, but it can be increased depending on nodal variables, interpolation functions and continuity. A typical use of one dimensional elements is frame analysis in solid mechanics.

For two dimensions, there are a number of basic shapes. The most common is the 3- node triangular element. This is a useful shape as most two dimensional shapes can be approximately constructed using triangles, without much distortion. Another useful shape is the 4-node rectangle, which can be easily created using most finite element programs.

Extending to three dimensions, the variety of shapes available for use is increased. The most common and useful shape is the 4-node tetrahedron. Similarly to the triangle in two dimensions, the tetrahedron can be used to create most solid shapes accurately. The right-hand prism is also common. Hexahedrons are also used, and can be constructed from five tetrahedra. Curved elements, known as isoparametric elements, are also used. They are useful for approximating curved boundaries with a minimum number of elements. All of the above elements have higher order, multiple node versions, which are useful in obtaining more accurate results.

3.5 Computer Implementation of Finite Element Analysis

Most finite element analysis is done currently using computers. There are a variety of programs available, but most operate in a similar manner. Generally, the program consists of a front-end user interface, typically a graphical interface, and the solver. The basic steps are:

1. Preprocessing - The user inputs the geometry, materials, loads and boundary conditions. The mesh is created at this stage. The user must take care that the mesh represents the structure correctly. The choice of elements must be made at this stage, ranging from one

dimensional bar elements to three dimensional solid elements, depending on the structure and results needed. Care must be taken to avoid such things as skewed elements, which would have a deleterious impact on the results. This step takes some experience and knowledge, so that the final model represents the structure the user needs to analyse.

2. Analysis - The model is then passed to the solver program, where the system equations are generated from the model. The program then solves the equations to give the results.

3. Postprocessing - The results are then passed back to the front-end interface, and the results can be displayed.

3.6 Composites and Finite Element Analysis

Composite materials present certain challenges in order to be modelled corectly and efficiently using the finite element method [13]. With a layered composite laminate, it would be possible to model each layer separately using three dimensional elements, with each layer of bricks modelling one layer of the laminate. However, this is highly inefficient for more than a few layers, as the required number of elements would be very large, and would consequently lead to long analysis times. Also, as the laminae are very thin, it could lead to ill-conditioned sets of equations. In most analyses, it is convenient to use plate or shell elements to model the laminate.

If the shell is sufficiently thin, shell theory provides a suitable representation of the behaviour. It must be assumed that there is no stress or strain in the direction normal to the plane of the midsurface. Due to difficulties with the classical Kirchhoff thin shell theory, which has problems with elements other than simple rectangular element geometries, other shell theories are used, such as Mindlin theory. These allow for the derivation of bending strains from the transverse shear strains, given by the equation

$$\varepsilon_{x'x'} = \frac{\partial u'}{\partial x'} + z' \frac{\partial \theta_{y'}}{\partial x'} \tag{3.13}$$

where the prime denotes coordinates in the local coordinate system in the plane, and normal to the mid-thickness layer. The other bending strains can be similarly derived. The derivation also

allows the transverse displacements w' and the rotations $\theta_{x'}$ and $\theta_{y'}$ to be derived independently, so simplifying the shape function needed. This means that standard shape functions of the form

$$\mathbf{w'} = \begin{bmatrix} N_1 & N_2 & N_3 & . & . & N_m \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ . \\ . \\ . \\ w_m \end{bmatrix} \tag{3.14}$$

and

$$\boldsymbol{\theta}_{x'} = \begin{bmatrix} N_1 & N_2 & N_3 & . & . & N_m \end{bmatrix} \begin{bmatrix} \theta_{x1} \\ \theta_{x2} \\ . \\ . \\ . \\ \theta_{xm} \end{bmatrix} \tag{3.15}$$

can be used, where N_i is the standard simple shape function used for many two dimensional membrane plate and brick elements.

A more complex form of shell element is the solid shell. This can be described as an element with the geometrical definition of a solid element but the behaviour of a shell element. This element may have a varying number of nodes, but only two elements through the thickness. Each node is restricted to only translational freedoms. Stretching of the element is given by the average value of a pair of nodes on the top and bottom, and the difference between the nodes gives the mid-plane rotation.

3.6.1 Modelling of Composite Laminates

Using laminate theory, as seen in Chapter 2, the strain can be given in terms of mid-plane strain ε_0 and curvature κ . It can be assumed that the strain in the plate varies in a linear

fashion with the local through-thickness. Since the material properties may vary from layer to layer in the laminate, the variation of the stresses in the laminate is more complex than the those of the strains. As seen before, the laminate stiffness properties are given by

$$\begin{bmatrix} N \\ M \end{bmatrix} = \begin{bmatrix} A & B \\ B & D \end{bmatrix} \begin{bmatrix} \varepsilon_0 \\ \kappa \end{bmatrix} \tag{3.16}$$

Using the average values of stresses gives the the in-plane loads N and the linear variation gives the moment M. For shell elements that have transverse shear strains included, the laminate stiffness matrix becomes

$$\begin{bmatrix} N \\ M \\ Q \end{bmatrix} = \begin{bmatrix} A & B & 0 \\ B & D & 0 \\ 0 & 0 & A_s \end{bmatrix} \begin{bmatrix} \varepsilon_0 \\ \kappa \\ \gamma \end{bmatrix} \tag{3.17}$$

where Q is the matrix of the transverse shear forces, A_s is the matrix of the laminate transverse material properties and γ is the transverse shear strains.

Chapter 4

Introduction to Optimisation

4.1 Introduction

Optimisation is not a new field - indeed, some of the most famous names in Mathematics have been involved, such as Euler, Newton, Galileo Galilei, and Lagrange, among many others [14]. However, it was only in the middle of the twentieth century that optimisation really expanded, driven by the development of computers. The discussion in this chapter presents a brief overview of various methods of solving optimisation problems.

4.2 Optimisation Methods

4.2.1 Statement of an Optimisation Problem

An optimisation problem can be stated as follows:

$$\text{Find } X = \left\{ \begin{array}{c} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{array} \right\} \text{ which minimises } f(X) \tag{4.1}$$

subject to the constraints

$$
\begin{aligned}
g_j(X) &\leq 0, \quad j = 1, 2, ...m \\
l_j(X) &= 0, \quad j = 1, 2, ...p
\end{aligned}
$$

where X is an n-dimensional vector, known as the design vector, $f(X)$ is known as the objective function, and $g_j(X)$ and $l_j(X)$ are known as the inequality and equality constraints [15]. The number of variables n and the number of constraints m and p need not be related. The problem stated in Equation 4.1 is known as a constrained optimisation problem. Some optimisation problems that have no constraints, which may be stated as:

$$
\text{Find } X = \left\{ \begin{array}{c} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{array} \right\} \text{ which minimises } f(X) \tag{4.2}
$$

In this case the problem is known as an unconstrained optimisation problem.

4.2.2 Unconstrained optimisation

There are a variety of methods for solving unconstrained problems, broadly divided into two groups, namely direct search methods and descent methods [16]. Direct search methods are generally only suitable for simple problems that have a small number of variables. This method requires only the objective function and no partial derivatives, and hence are generally less efficient than descent methods. Descent methods incorporate first, and possibly second, derivatives of the objective function in addition to the function itself. This makes them more efficient as more information is available for the search method.

Unconstrained optimisation methods are iterative - this means they proceed from an initial trial solution and move toward the optimal result in a sequence of steps. This means that an initial point X_i is required to start the process. The various methods available differ in how

they generate the next point, X_{i+1}, and in testing the point X_{i+1} for optimality.

4.2.3 Direct search methods

In direct search methods, the techniques range from random searching, to the more rationalised approaches such as the simplex method. The random search is the simplest of the methods available, and involves taking a wide range of values and substituting them in to find the correct solution. This method can be extended in the random walk method, which enables the method to move toward the optimal result. This is done by setting the new approximation with

$$X_{i+1} = X_i + \lambda u_i \tag{4.3}$$

where λ is a prescribed scalar step length and u_i is a unit random vector generated for the ith stage. The new value X_{i+1} is compared with the previous X_i and proceeds if it is a better solution, otherwise a new random vector is generated. The step length is made smaller after each successful iteration. This process proceeds until the step length becomes smaller than the minimum allowable step length ε.

The univariate method [15] involves changing only one variable at a time in the objective function. This reduces the problem to a one-dimensional optimisation problem, which can be solved with a wide range of simple techniques. This solution gives a new starting point, and the search then proceeds in a new direction. This is done by changing any of the other variables that were fixed in the previous iterations. This method is easy to implement, but care must be taken in regard to the result oscillating as it approaches the optimum point.

Pattern search methods, such as Hooke and Jeeves, and the Powell method [15], utilise the observation that the univariate approach creates a pattern direction in the direction of the optimum result. The Hooke and Jeeves technique is a two step method, in that it moves twice per iteration. The first step, the exploratory move, analyses the local behaviour of the objective function, and the second step, the pattern move, takes into account the pattern direction. The Powell method utilises conjugate directions to enhance the efficiency of the optimisation. The simplex technique utilises the simplex, which is a geometric figure formed from $n + 1$ points in

n-dimensional space. Thus, for two dimensions, the simplex is a triangle, in three dimensions it is a tetrahedron, and so on. The basic idea is to compare the values of the objective function at the corners of the simplex, and move the simplex toward the optimum result in an iterative process.

4.2.4 Descent Methods

In the various descent methods, the gradient of the objective function is given by

$$\underset{n\times 1}{\nabla f} = \begin{Bmatrix} \partial f/\partial x_1 \\ \partial f/\partial x_2 \\ . \\ . \\ . \\ \partial f/\partial x_n \end{Bmatrix} \tag{4.4}$$

This gradient represents the direction of steepest ascent, with the negative of the vector representing the direction of steepest descent. In general, the gradient varies from point to point. The evaluation of the gradient vector requires the computation of the partial derivatives $\partial f/\partial x_i, i = 1, 2, ...n$. This can sometimes be impractical, such as when the calculation requires a large amount of time, or is impossible. The objective function would then be approximated using, for example, the finite difference formula.

The steepest descent method [15] is the simplest of the gradient-based methods. The gradient is calculated, and used to determine the search direction, and hence the next point in the process. While this technique is useful, it is limited by the fact that the steepest descent direction is a local property, and so it may not find the absolute optimum point. The Fletcher-Reeves method improves on the steepest descent method by utilising the conjugate gradient technique to obtain a better search direction.

Newton's method [17] is based on using the Taylor's series expansion of the objective $f(X)$

at $X = X_i$, which gives

$$f(X) = f(X_i) + \nabla f_i^T (X - X_i) + \frac{1}{2}(X - X_i)^T [J_i\}(X - X_i) \tag{4.5}$$

where $[J_i]$ is the matrix of second partial derivatives of f evaluated at the point X_i and is known as the Hessian matrix. By setting the partial derivatives of 4.5 to zero, we get

$$\frac{\partial f(X)}{\partial x_i} = 0, \quad j = 1, 2, ...n \tag{4.6}$$

Combining 4.5 and 4.6 we get

$$\nabla f = \nabla f_i + [J_i]^{-1}(X - X_i) = 0 \tag{4.7}$$

If $[J]^{-1}$ is non-singular, then 4.7 can be used to obtain a better solution by

$$X_{i+1} = X_i - [J]^{-1} \nabla f_i \tag{4.8}$$

Newton's method works particularly well when the design vector is close to the optimal point, converging rapidly on the solution.

The Marquardt method [15] was introduced to combine the steepest descent method, which works well far from the optimal point, and Newton's method, which works well close into the optimum. The Hessian matrix $[J_i]$ is modified to

$$[\bar{J}_i] = [J_i] + \alpha_i [I] \tag{4.9}$$

where $[I]$ is the identity matrix, and α is a positive constant. For large values of α_i, the search pattern becomes the steepest descent method. As α_i decreases, so the characteristics of the search method approaches Newton's method. So in the Marquardt method, as the search proceeds, so the value of α_i is slowly decreased, increasing the efficiency of the search.

4.2.5 Constrained Optimisation

Problems that involve constraints usually represent a more realistic problem. There are a number of potential scenarios that occur when constraints are introduced. The first is that the constraints have no effect on the optimum point. However, this can be difficult to determine, and so it is usually assumed that this is not the case, and that the constraints have an effect on the optimal point. The optimal point could occur on the boundary defined by the constraints. There is the possibility that the constraints could introduce multiple local optimal points. The optimisation method must be able to find the optimum in all of these situations.

Constrained problems can be solved using either direct or indirect methods. In direct methods, the constraints are handled explicitly, while in indirect methods, the constrained problem is solved as a series of unconstrained minimisation problems.

The random search method for unconstrained problems can be adapted to those with constraints. The random values used must fall within the constraints imposed on the problem. Similarly, the random walk method can be adapted to constrained problems.

The simplex method can be extended to constrained problems of the type

$$\text{Minimise } f(X) \tag{4.10}$$

subject to

$$g_j(X) \leq 0 \quad j = 1, 2, ..., m$$

$$x_i^{(l)} \leq x_i \leq x_i^{(u)} \quad i = 1, 2, ..., n$$

This is called the complex method and utilises a sequence of geometric figures, each having $k \geq n + 1$ vertices in an n-dimensional space, called a complex. These are gradually moved in towards the optimum point by removing the worst vertex and replacing it with a vertex that is closer to the optimal. The solution is considered to have converged when the complex shrinks to a specified small size.

Sequential linear programming (SLP) [15] works by creating a series of linear approximations of the original problem. The linear problems are solved using the simplex method to find the

next step in the solution. The objective function and constraints are linearised and the problem is reformulated as

$$\text{Minimise } f(X_i) + \nabla f_i^T (X - X_i) \tag{4.11}$$

subject to

$$g_j(X_i) + \nabla g_j(X_i)^T (X - X_i) \leq 0, \quad j = 1, 2, ..., m$$

$$h_k(X_i) + \nabla h_k(X_i)^T (X - X_i) = 0, \quad k = 1, 2, ..., p$$

This is solved for the solution X_{i+1}. If $g_j(X_{i+1}) \leq \varepsilon$ for $j = 1, 2, ..., m$ and $|h_k(X_{i+1})| \leq \varepsilon$ for $k = 1, 2, ..., p$ where ε is a prescribed small positive tolerance, then all the original constraints have been met, and the solution is taken as $X_{opt} \approx X_{i+1}$. If the constraints are violated, the most violated needs to be found and the constraint relinearised about the point X_{i+1} and this new constraint added to the previous linear programming problem. The cycle then starts again, with new iteration number $i = i + 1$. SLP is efficient, especially for nearly linear objective and constraint problems.

The method of feasible directions [17] covers a variety of techniques. The basic premise is that of finding a better point X_{i+1}using

$$X_{i+1} = X_i + \lambda S_i \tag{4.12}$$

where λ is the step length, S_i is the direction of movement, and X_i is the starting point. S_i is found such that a small move in that direction does not violate any of the given constraints, and the value of the objective function is decreased in that direction. This is repeated in an iterative manner until no direction can be found that satisfies the previous conditions. A direction S_i is usable and feasible direction if

$$\frac{d}{d\lambda} f\left(X_i + \lambda S\right)|_{\lambda=0} = S^T \nabla f(X_i) < 0 \tag{4.13}$$

$$\frac{d}{d\lambda} g_j\left(X_i + \lambda S\right)|_{\lambda=0} = S^T \nabla g_j(X_i) \leq 0 \tag{4.14}$$

4.2.6 Heuristic Methods

Heuristic methods are intuitive methods that are practical and quick [15]. They are usually based on simple rules and common sense. There are a number of different methods, of which most are based on observations of the natural world. While it is not guaranteed that an optimum can be found, a good heuristic method will arrive at the best solution available in the time allocated.

Among the heuristic methods available is Simulated Annealing [15], which, as the name indicates is based on principle of the annealing of metals. The method drives the problem to a solution in a similar method to how controlled cooling drives a heated solid to proper solidification with a highly ordered crystalline state, with the lowest internal energy. The optimisation is stated as

$$\text{Minimise } f(X) \tag{4.15}$$

subject to

$$x_i^{(l)} \leq x_i \leq x_i^{(u)}, \quad i = 1, 2, ..., n$$

The algorithm undergoes a series of random moves, each along one coordinate direction. The generated points are accepted or rejected according to the metropolis criterion. This gives $X_i^+ = X_i + \Delta X$, $f_i^+ = f(X_i^+)$ and $\Delta f = f_i^+ - f_i$. If $\Delta f \leq 0$ then X_{i+1} is set equal to X, otherwise the new point is accepted with a probability of

$$P(\Delta f) = e^{-\Delta f / kT} \tag{4.16}$$

where k is Boltzmann's constant and T is a parameter called temperature. The algorithm starts with a high value of T and it is reduced once f has reached a stable value. This starts the cycle again and this continues until a sufficiently low temperature is arrived at and at which no more improvement in the objective function can be reached.

Genetic Algorithms are another popular heuristic method, where a random selection of optimal solutions is evolved towards the best solution, using the principal of survival of the fittest [15]. These algorithms are useful for problems that combine mixed continuous-discrete

variables, and non-convex and discontinuous design spaces. Typical optimisation methods are inefficient and often do not find the absolute optimum and instead find a local solution close to the starting point. Genetic algorithms work well in these situations, and can find the optimal solution with a high degree of probability. These algorithms utilise the principles of genetics and natural selection, with reproduction, genetic crossover and genetic mutation providing the driving force for the algorithms. Instead of a single trial solution, the algorithm starts with a population of solution points. These points are represented by binary strings, which can represent both discrete and continuous variables. The objective function is rewritten as an unconstrained problem with a penalty function, given by

$$\text{Minimise } f(X) + R \sum_{j=1}^{m} \Phi(g_j(X)) \tag{4.17}$$

subject to

$$x_i^{(l)} \leq x_i \leq x_i^{(u)}, \quad i = 1, 2, ..., n$$

where Φ is the penalty function given by

$$\Phi(Z) = \langle Z \rangle^2 \tag{4.18}$$

where

$$\langle Z \rangle = \begin{cases} Z & \text{if } Z > 0 \\ 0 & \text{if } Z \leq 0 \end{cases}$$

and R is a constant, known as the penalty parameter. Following this, the minimisation of $f(X)$ is achieved by the maximisation of the fitness function $F(X)$, defined as

$$F(X) = F_{\max} - \left((f(X) + R \sum_{j=1}^{m} \Phi(g_j(X)) \right) = F_{\max} - f'(X) \tag{4.19}$$

where $F_{\max}$ is chosen to be greater than the largest value of $f'(X)$ in the population. In the algorithm, those solutions that haver the highest fitness are selected for the next step, genetic crossover. For this step, two binary strings are chosen and combined to create two new offspring. These new strings are added back into the population. Finally, a mutation operator is applied to the newly generated strings, which has a preset probability of altering the string. This helps

prevent the loss of critical solution information.

Chapter 5

Development of the ESO Algorithm

5.1 Introduction

This chapter will deal with the creation of the ESO Algorithm, starting with the general developmental history of Evolutionary Structural Optimisation. This provides the basis for the development of the ESO algorithm for the application presented here. The algorithm is developed through a series of demonstration models, and gradually expanded to include the use of composite materials.

5.2 A History of Evolutionary Structural Optimisation

In 1993, Y.M. Xie and G.P. Steven presented a new optimisation technique that they termed Evolutionary Structural Optimisation [18]. This technique allied the finite element method and shape optimisation. The idea was based on the structural adaptation found in nature, such as bone formation, shells, and trees. The central principle was the removal of unnecessary material from the given structure, over a successive number of iterations. This initial version of ESO was very basic, only removing material from the given structure, and used the stresses in the elements as the determining criterion for the removal. Material that was at a low stress level was removed, eventually leaving the optimised structure. This was followed by what was known as Additive ESO [19], which involved the addition of material to a minimal structure, by generally adding material to high stress areas. This could be thought of as growing an optimal

structure from some minimum, typically a basic shape connecting the loads to the boundary conditions. This was quickly developed further, with the removal and addition of material in a single algorithm, now termed Bidirectional ESO [20], as the structure could both grow or shrink, depending on the results. These algorithms all utilised the stresses in the elements as the determining criterion.

The ESO algorithm proved to be highly adaptable to many different optimisation requirements. One of the early adaptations of ESO was the use of stiffness as a determining criterion for the algorithm [21]. In this instance, the structure was analysed, and the elements assigned individual sensitivity numbers, based on the effect they had on the overall stiffness of the structure. Those that affected the stiffness the least would then be removed from the structure. This was useful in structures where the displacement was critical.

A further development was the creation of an ESO algorithm for heat transfer across a structure [22]. For problems like this, a thermal model is created and analysed. The criterion for removal and addition is the heat transfer across an element. In general, those elements with low heat transfer would be removed. The use of ESO algorithms in this field proved especially useful, as typical heat transfer optimisation is a complex mathematical problem.

In a similar way, ESO algorithms were created for fluid dynamics. This involved the reduction in drag of a structure placed into fluid flow. This was done by smoothing out the structure, to create a more hydrodynamic shape.

When ESO was originally developed, it remained a purely empirical method. When compared to other, more rigorous numerical methods, the results for the ESO algorithm were found to be similar. However, there was no mathematical basis to the algorithm to explain its validity. Much work went into proving the legitimacy of ESO as a viable technique [23].

At the present time, ESO is being extended and improved, and is finding greater usage in the engineering field [24]. This is due to its simplicity and usability, and its ability to provide valid qualitative results in minimal time. It is being used to solve many different and complex problems by designers and engineers in their attempts to create optimal designs [25]. This makes ESO a useful tool in the modern designer's collection.

5.3 The Development of the ESO Algorithm

As described previously, the goal was the creation of an ESO algorithm that could be applied to the model of a composite structure. The basis for the development was a simple ESO algorithm similar in operation to that presented by Xie and Steven [18]. This simple basis would help eliminate any uncertainties and biases that may have occurred when using other developed ESO algorithms. The algorithm was developed in a series of steps, starting with simple two dimensional shapes, and moving to more complex, three dimensional models. Each step was used to develop a particular aspect of the algorithm, until the algorithm was ready to be applied to the final test structure. The basic ESO algorithm is shown, and then the developmental models are presented, with each particular adaptation shown. Finally, the fully developed final algorithm is presented.

5.3.1 The Basic ESO Algorithm

The ESO algorithm that was used as a starting point for the development may be described as follows:

1. The design domain is discretised into elements. The normal conditions for meshing a solid structure apply here, such as more elements in higher stress areas. There is no specific need to ensure that the elements are the same size.

2. The elements are assigned their material properties.

3. The structure is then analysed under the loading conditions using MSC.Nastran and the stresses are exported to the algorithm

4. The removal threshold value for the removal of elements is determined. The value is based on the maximum stress of the structure, and the number of iterations the optimisation routine has gone through. Hence the removal threshold value is given by:

$$\sigma_T = RR_i \times \sigma_{\max} \tag{5.1}$$

where RR_i is the current rejection ratio for iteration i and $\sigma_{\max}$ is the maximum value of stress in the structure. The value of the rejection ratio is given by:

$$RR_i = RR_0 + (i-1) \times ER, \quad i = 1, 2, 3, ... \tag{5.2}$$

where RR_0 is the initial rejection ratio, ER is the evolutionary rate, and i is the iteration number.The values of RR_0 and ER are based on experience with the algorithm.

5. With the removal threshold value, elements are selected for removal. All elements with a stress value lower than the removal threshold value will be removed from the model.

6. The algorithm then modifies the structure, removing material based on the above criteria. The structure is then resubmitted for analysis.

7. The cycle comes to a halt when the removal threshold value reaches a predetermined value.

5.3.2 Two-Dimensional Models

The first step in the development of the algorithm was the application to two-dimensional models. This step was used to develop the basic algorithm programs and routines. Various concepts were tested and discarded at this stage. Due to the simplicity of the model, it made it simpler to observe the effects that occurred as changes were made to the algorithm, as the analysis times were rapid.

Once the basic algorithm, as shown previously, was working, the first adaptations were made. At this stage the process of removal was altered. Instead of removing the element from the model, the elements were modified, their material properties altered so that they had a minimal effect on the structure. This technique was described by Rozvany [26], and is achieved by modifying the element properties so that they have no further effect on the load-carrying capacity of the structure. It was suggested that the element properties, namely the Young's and Shear Moduli, be modified to a value of 10^{-6} times smaller than their original values. This alteration provided certain benefits. This method solved the problem of 'islanding', where elements may be isolated after the removal process, or cases where loads and boundary conditions

become detached from each other. This would cause the failure of the finite element analysis due to numerical instability. Using the modification method, instead of removal, prevented this from occurring, This would mean that islands could not form, nor boundary conditions be detached from the loads.

A further development incorporated the use of non-modifiable material. This was done for areas where boundary conditions and loads were attached. This prevented any problems that may have occurred, had the element to which the boundary condition or load was attached been modified. If it had been modified, the results from the finite element analysis would have become invalid, since the loads and boundary conditions would be applied to an element that had been effectively removed from the model.

A further consequence was the ability to add material back to the structure. This was done in two ways. The first was for areas of high stress concentration. If the algorithm found an area where the stress level had been raised such that approached failure, as specified by the user, the algorithm would attempt to add material around the overstressed elements. This was done by finding all modified elements that shared a common node with that element, and modifying them back to the original material. This can be seen in the diagram below.

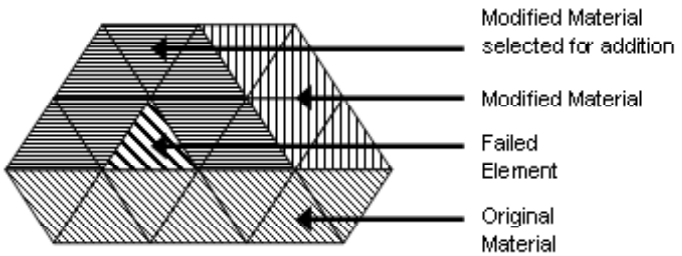


Figure 5-1: Addition of Material

The other addition method was analysing the modified material directly, and finding any that had risen sharply in their stress values. Those modified elements that had shown such behaviour were then added back into the structure as a precaution.

The example used for develeopment was a flate plate, with a single point load, and constrained around three of the edges. The model was given orthotropic properties, matching a typical composite material. This model was based on a paper presented by Walker [27], which presented a simple self-design technique. Two versions of the model were used, one with 256 rectangular plate elements, and a second with 1024 elements. The elements were assigned orthotropic properties, given by:

$$E_{11} = 75 \text{ GPa}$$

$$E_{22} = 45 \text{ GPa}$$

$$\nu = 0.22$$

$$G_{12} = 4 \text{ GPa}$$

$$G_{23} = 2 \text{ GPa}$$

$$G_{13} = 4 \text{ GPa}$$

The modified materials were given by:

$$E_{11} = 75 \text{ kPa}$$

$$E_{22} = 45 \text{ kPa}$$

$$\nu = 0.22$$

$$G_{12} = 4 \text{ kPa}$$

$$G_{23} = 2 \text{ kPa}$$

$$G_{13} = 4 \text{ kPa}$$

The plate was loaded with a single point load of 20000 N perpendicular to the plate, located $\frac{3}{4}$ of the way from the bottom and the left side. The plate was clamped along the left, top and right sides, leaving the bottom edge free. These loads and boundary conditions can be seen in the diagrams below.

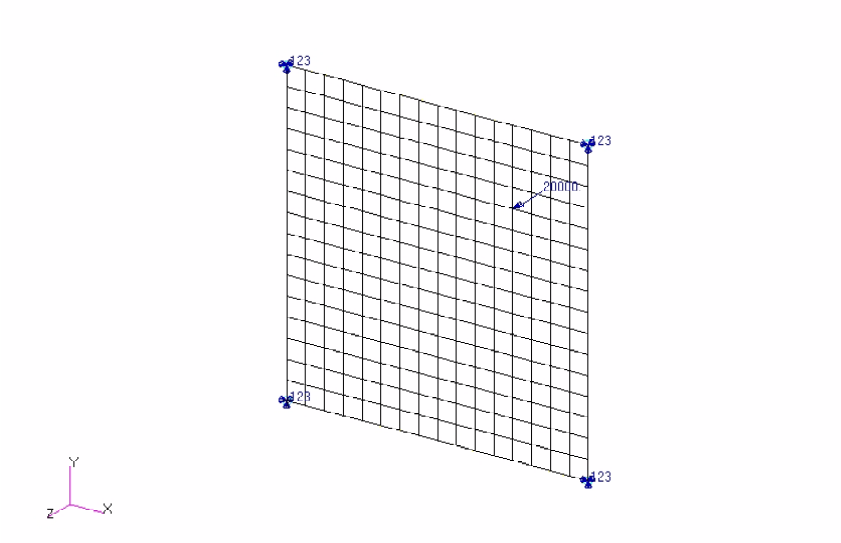


Figure 5-2: 256 Element Flat Plate Model

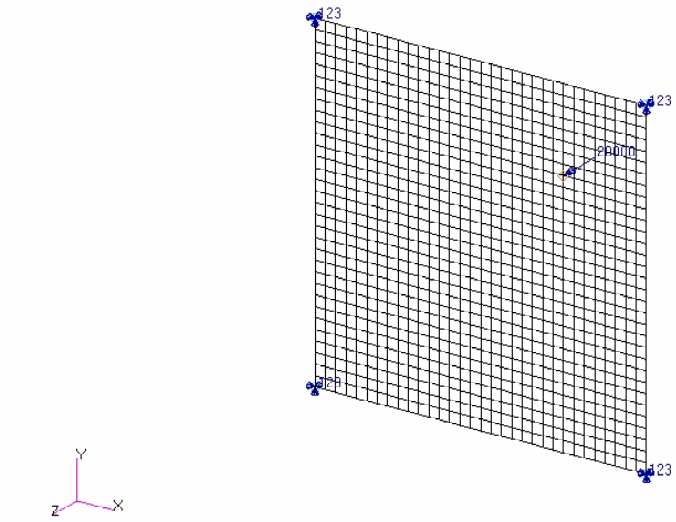


Figure 5-3: 1024 Element Flat Plate Model

The model was run for various combinations of ER and RR_0. The values are given in Table 5.1.After the models were run, the results were plotted, and con be found in Appendix A, consist graphs of comparisons between the maximum and removal stresses, and the area remaining in the model, which refers to the combination of original and non-modifiable material still left in the model, and the number of iterations. The results are shown in table 5.2 for comparison.

Model Number	Model	RR_0	ER
1	256 Elements	0.5	0.25
2	256 Elements	1	0.5
3	256 Elements	2	1
4	1024 Elements	0.5	0.25
5	1024 Elements	1	0.5
6	1024 Elements	2	1

Table 5.1: ER and RR for 2D Plate

Model Number	No of Iterations	Area Remaining (%)
1	100	29
2	50	27.3
3	27	26.2
4	87	19.7
5	44	15.2
6	20	18.3

Table 5.2: 2D Model Results

The analysis halted when the stress had reach a specified level of 50 MPa. The results of the optimisation process are shown in Table 5.2 and the diagrams below.In the diagrams, the white is the non-modifiable areas, red is the original material, and green is the modified material.

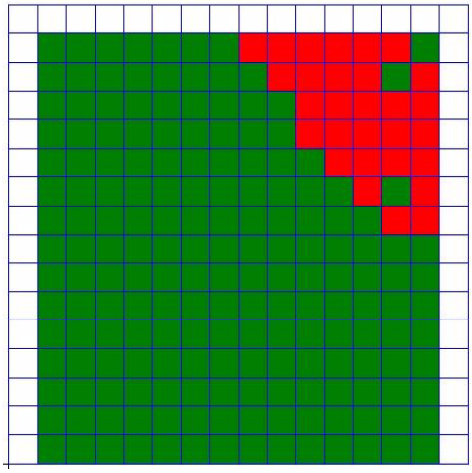


Figure 5-4: 2D Model 1 Results

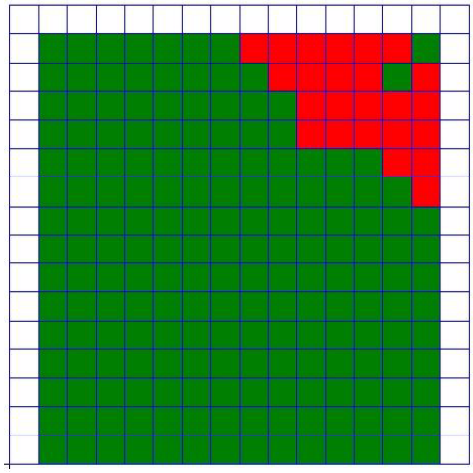


Figure 5-5: 2D Model 2 Results

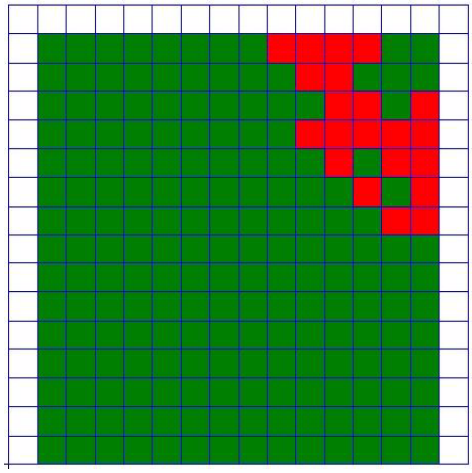


Figure 5-6: 2D Model 3 Results

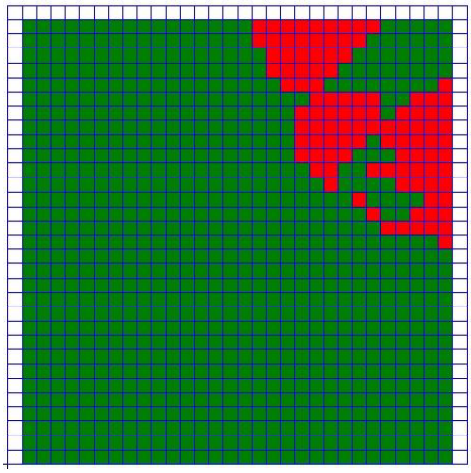


Figure 5-7: 2D Model 4 Results

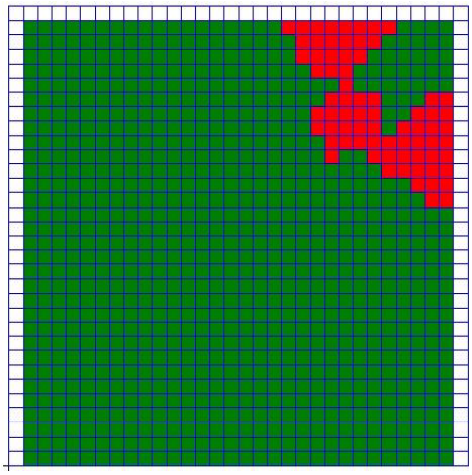


Figure 5-8: 2D Model 5 Results

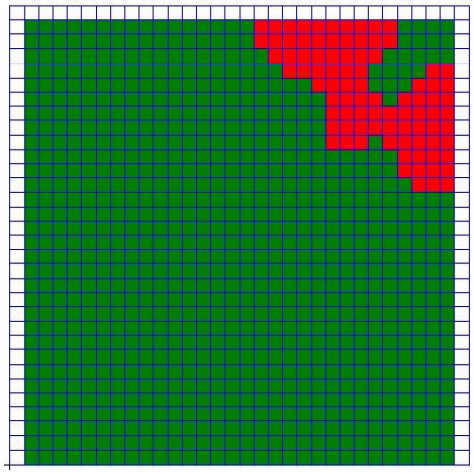


Figure 5-9: 2D Model 6 Results

These models also show a characteristic of the ESO algorithm - with higher density meshes, the algorithm becomes more efficient as it can remove more material. This will be discussed further in a later section.

5.3.3 The Application of the ESO Algorithm to Composite Structures

Before the ESO algorithm was developed further, the application of the ESO algorithm to three dimensional composite structures had to be considered, as the aim was the optimisation of composite sandwich structures. Sandwich structures provide a method of creating efficient products, ranging from skis to turbine blades. The sandwich structure consists of two exterior

layers, known as the skins, and the internal core. The separation of the skins by the core, which is typically of a low density material, increases the moment of inertia of the beam/panel with little increase in weight. This helps create a light, stiff and strong structure. Under loading, the core and skins have different roles to play. The diagram below illustrates the different tensile and shear loads that occur in a sandwich structure. As can be seen, the skins tend to be dominated by the tensile and compressive stresses, while the core is dominated by the shear stresses.

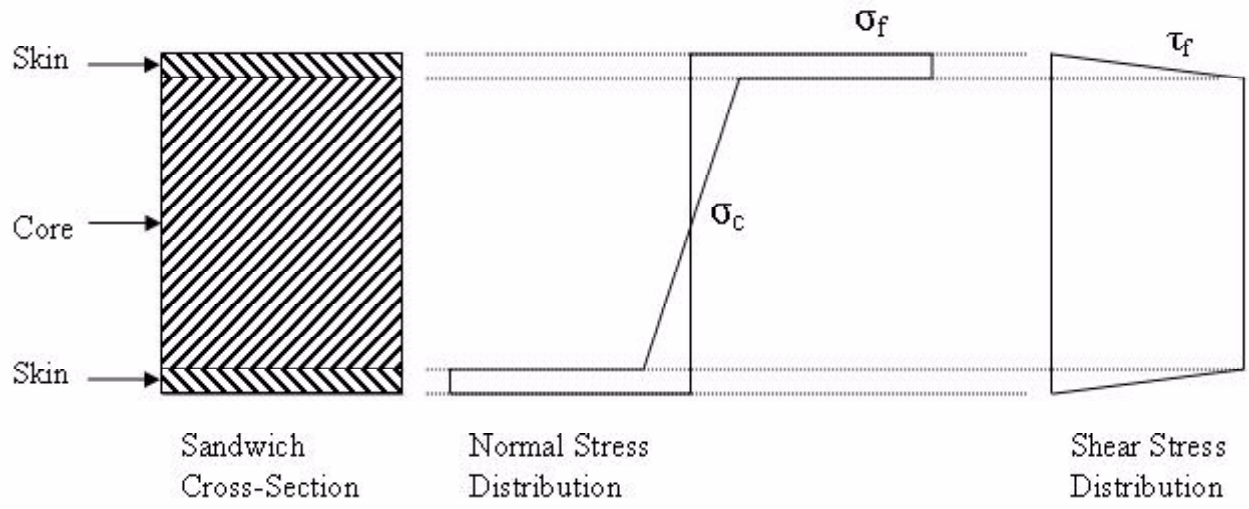


Figure 5-10: Approximate stress distribution in a sandwich structure

The aim is to produce a composite sandwich structure of minimal core volume. To model this for the algorithm, the structure is modelled using the following steps:

1. The geometry for the sandwich structure is created.

2. The core is modelled using 3D solid elements. These are assigned idealised material properties, dominated by the strain moduli. This assumption is based on the ability of composite materials to be designed to fit the application. This enables the algorithm to easily work with the composite material.

3. The skins are then created, by generating layers of 2D shell elements on the free faces

of the 3D core elements. These elements are assigned orthotropic material properties to match the composite material from which they would be constructed. These skins are not altered during the optimisation process. This is important, for example, for structures where aerodynamics are important. The skin carries and distributes the load, allowing the ateration of the core, including the removal of material directly under the load.

4. Loads and boundary conditions are applied.

The model was then ready to be optimised.

5.3.4 Three-Dimensional Models

Following the successful testing on the two-dimensional plates, the algorithm was adapted to three dimensions. This was done using a sequence of models, that by the end of which the algorithm was ready to be applied to the test case.

The L-Block

This model was used to extend the algorithm into three dimensions. It was a simple L-shape, clamped on the base, and a number of point loads on the upper surface. This can be seen in the diagram below. The model was meshed with hexahedral 3D elements, with a skin of 2D rectangular plate elements. The green elements are modifiable material and red elements are non-modifiable.

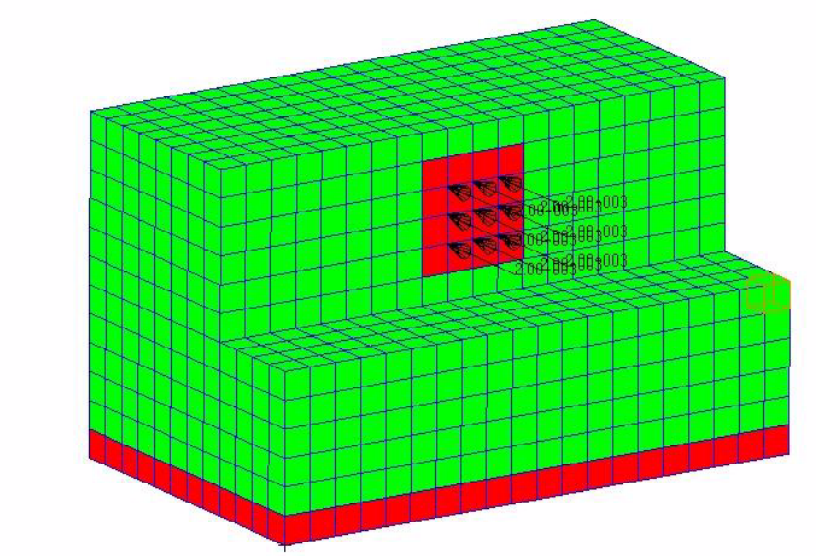


Figure 5-11: L-Block Model

The model was given the following material properties:

$$E_{11} = 1.81 \text{ GPa}$$

$$E_{22} = 1.03 \text{ GPa}$$

$$E_{33} = 1.81 \text{ GPa}$$

$$\nu = 0.28$$

$$G_{12} = 7.13 \text{ GPa}$$

$$G_{23} = 5 \text{ GPa}$$

$$G_{13} = 7.13 \text{ GPa}$$

The modified material was given Young and Shear modulus values set to 10^{-6} times the original value.

The block was run for a range of values for RR_0 and ER, which are given in Table 5.3 below.

Model	RR_0	ER	Iterations	Volume Remaining (%)
1	0.5	0.25	144	14.08
2	1	0.5	87	14.25
3	2	1	42	13.83

Table 5.3: Initial RR and ER values for L-Block Model

A selection of the resulting models are shown below, showing the changes from the original shape. The models show only the non-modified and non-removable material.

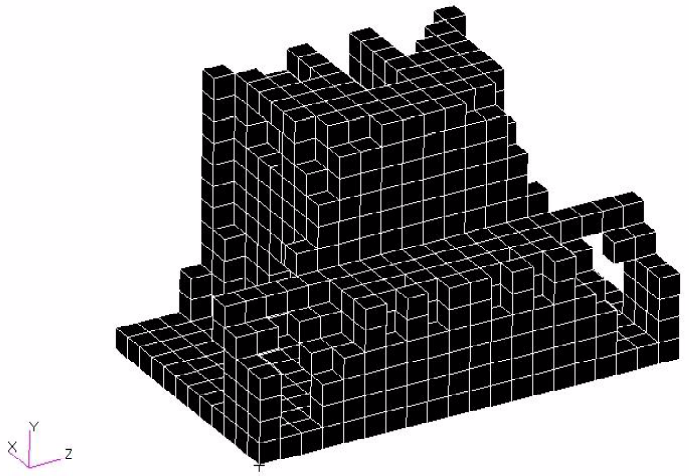


Figure 5-12: L-Block Model 1 after 30 Iterations

.

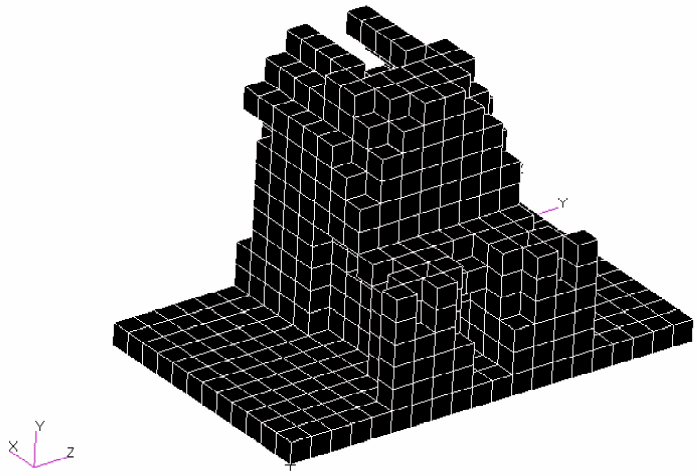


Figure 5-13: L-Block Model 1 after 60 Iterations

.

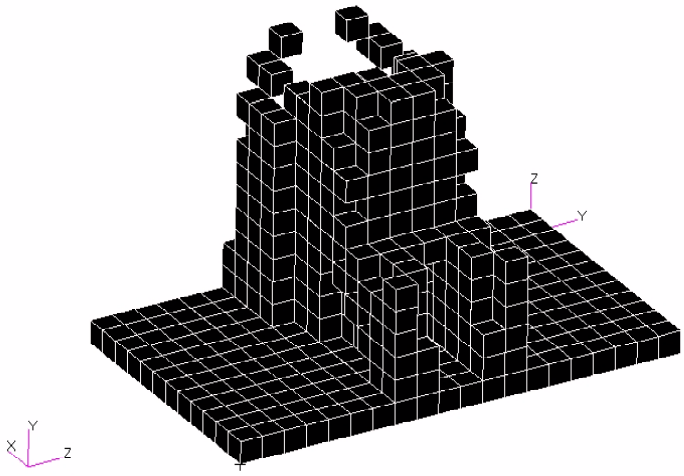


Figure 5-14: L-Block Model 1 after 90 Iterations

.

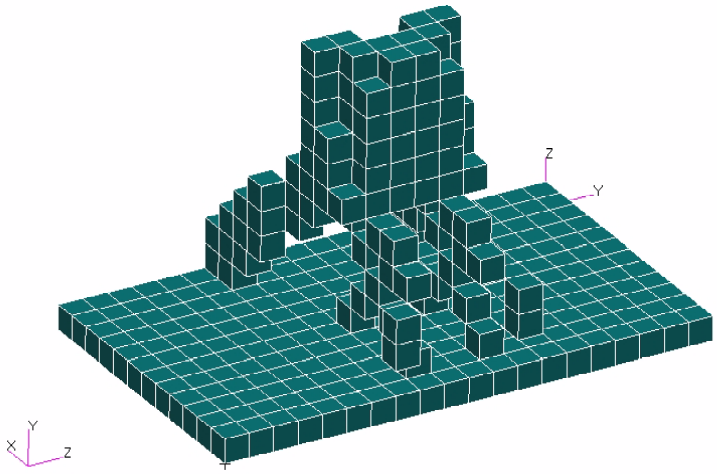


Figure 5-15: L-Block Model 1 after 120 Iterations

.

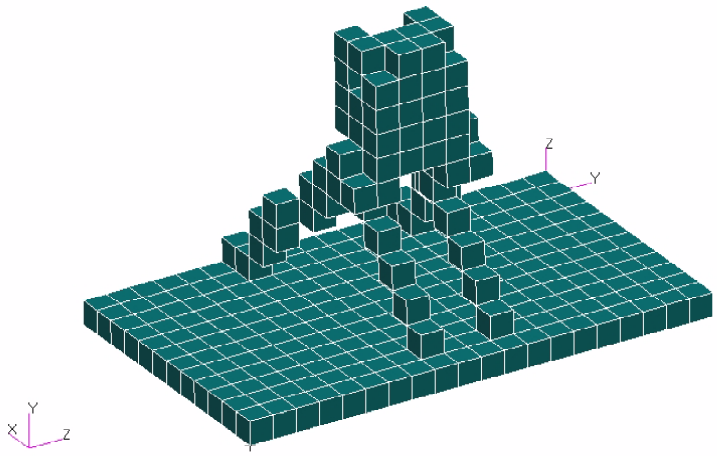


Figure 5-16: L-Block Model 1 after 144 Iterations

.

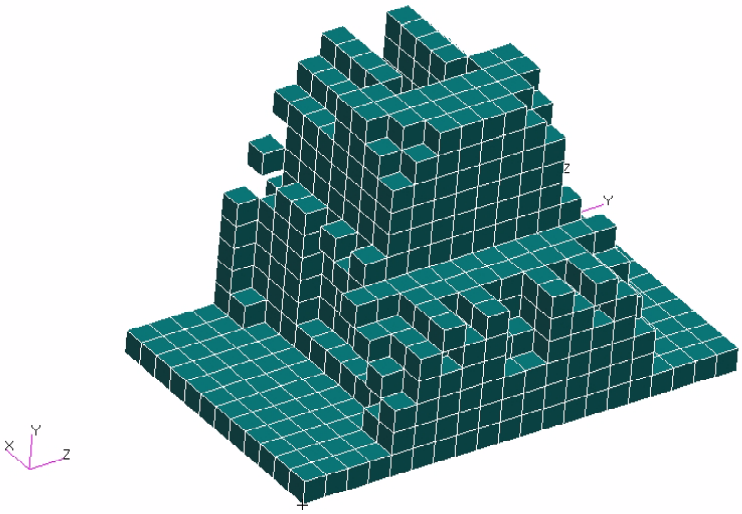


Figure 5-17: L-Block Model 2 after 20 Iterations

..

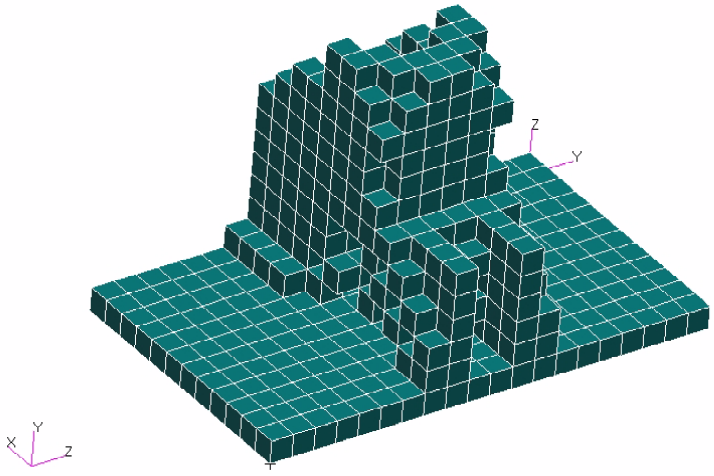


Figure 5-18: L-Block Model 2 after 40 Iterations

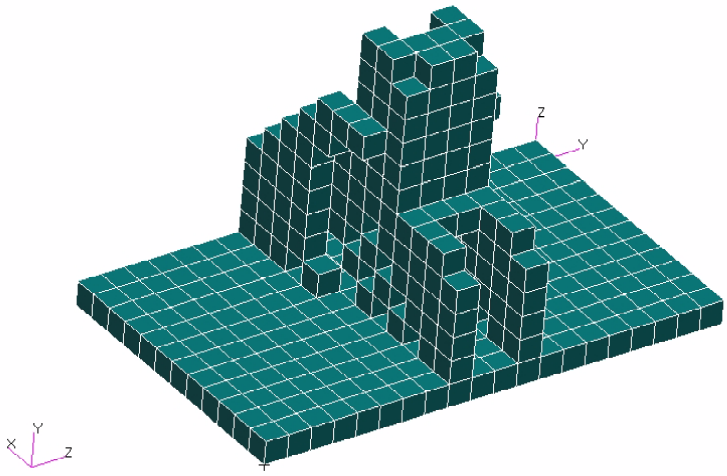


Figure 5-19: L-Block Model 2 after 60 Iterations

.

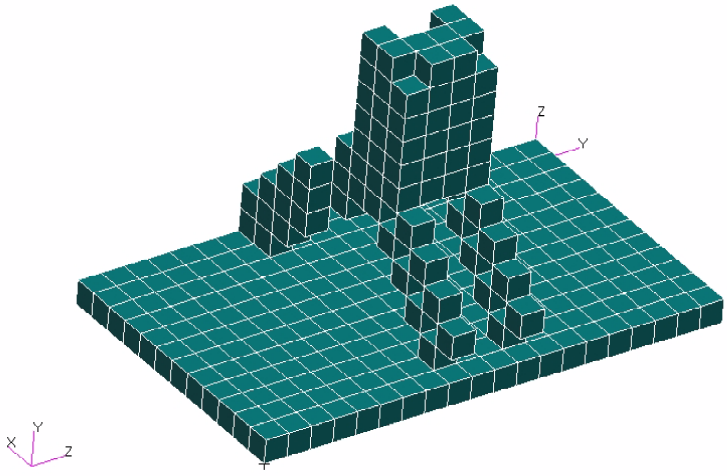


Figure 5-20: L-Block Model 2 after 88 Iterations

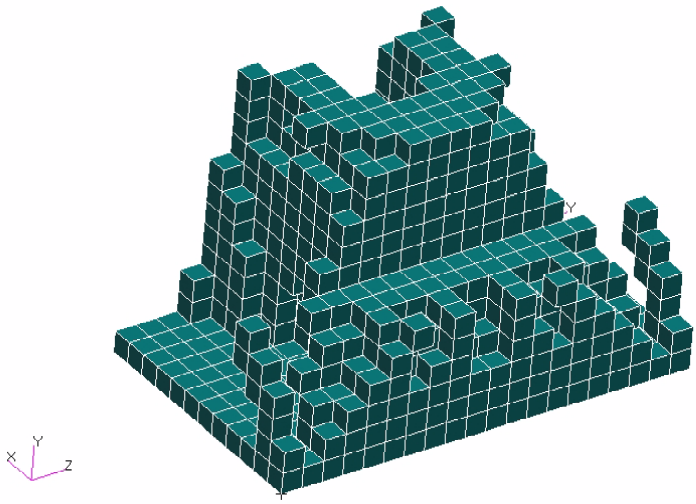


Figure 5-21: L-Block Model 3 after 9 Iterations

.

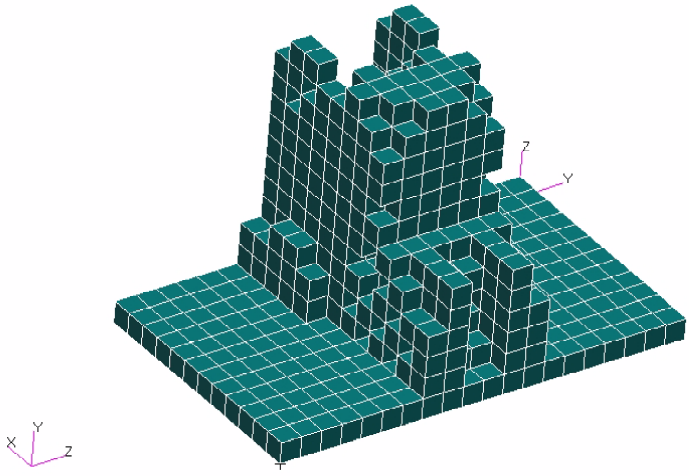


Figure 5-22: L-Block Model 3 after 18 Iterations

.

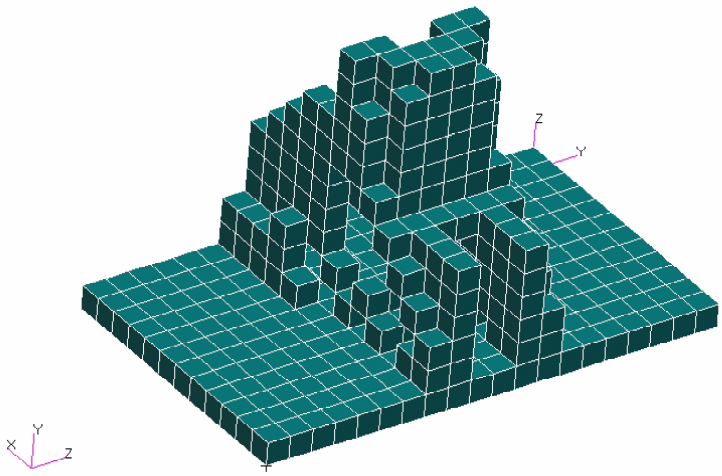


Figure 5-23: L-Block Model 3 after 27 Iterations

.

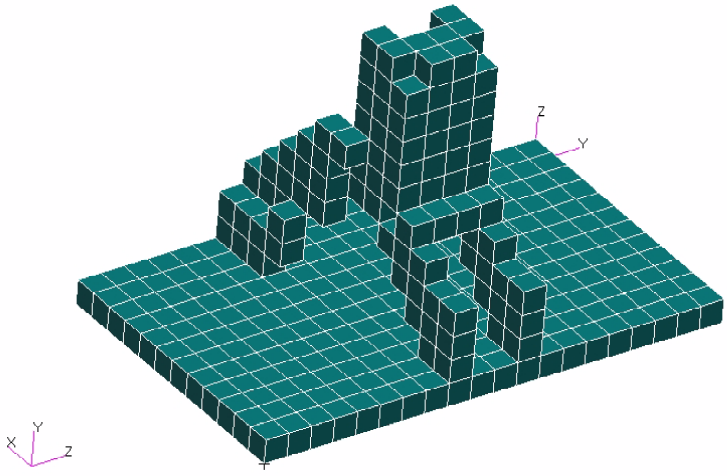


Figure 5-24: L-Block Model 3 after 36 Iterations

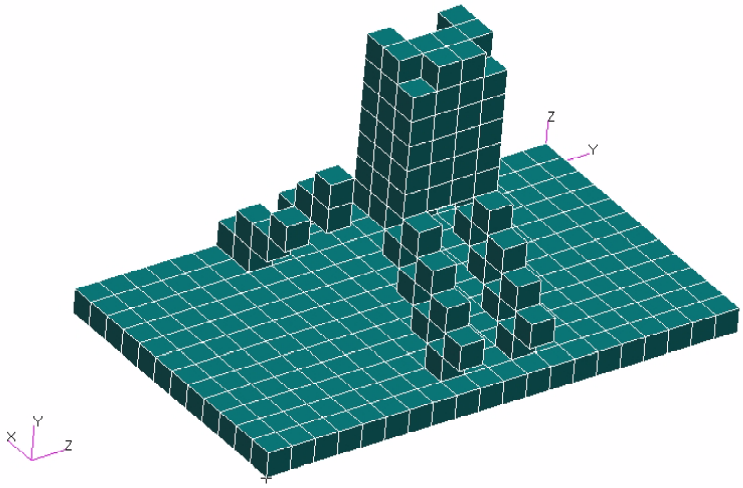


Figure 5-25: L-Block Model 3 after 43 Iterations

.

The graphs showing changes in volume and stress can be seen in Appendix A.

For the given loading conditions, the resulting shapes consist of the area where the load is applied, and connections to the front and rear edges of the bottom plate. Looking at the resulting shape from the side, it can be seen it resembles a triangular truss, with the bottom points anchored on the plate, and the top point where the load is applied. This is an effective

shape in dealing with loading and boundary conditions such as those specified, and so the final resulting models can be understood in those terms.

Sandwich structure

With the ESO algorithm adapted to three dimensions, work proceeded with the application to sandwich structures. As the test structure would be modelled as using tetrahedral elements, this basic model of a sandwich structure was used to adapt the algorithm to use tetrahedral elements, as well as test the method of modelling composite structures as previously outlined.

In order to do this, a model was created and meshed twice, once with hexahedral elements and once with tetrahedral elements. The same loads and boundary conditions were applied to both models. The models are shown below.

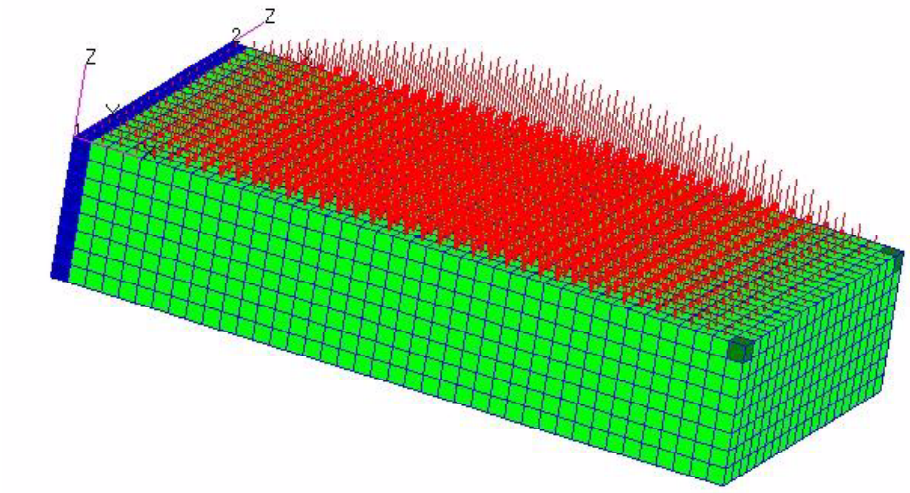


Figure 5-26: Sandwich Model - Hexahedral Elements

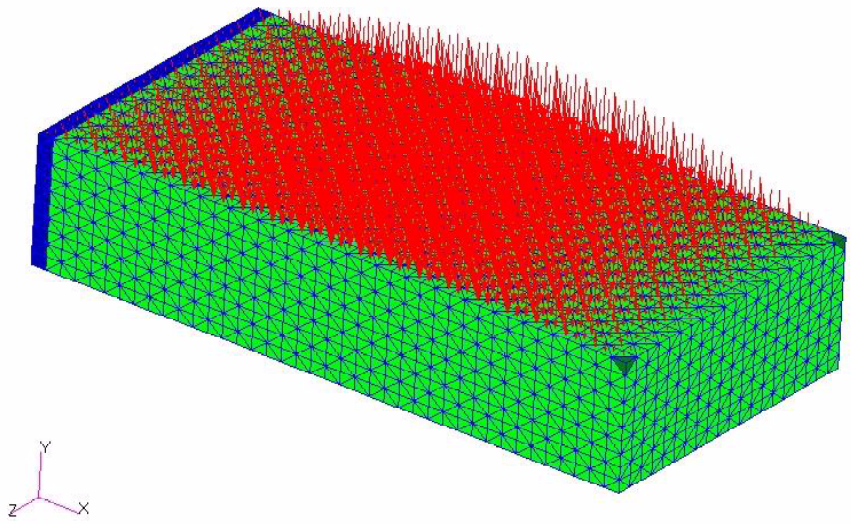


Figure 5-27: Sandwich Structure - Tetrahedral Elements

.

The load is a pressure load, distributed over upper surface, with a parabolic profile, with the maximum load in the centre. The skin elements were given the following material properties:

$$E_{11} = 75 \text{ GPa}$$

$$E_{22} = 45 \text{ GPa}$$

$$\nu = 0.22$$

$$G_{12} = 4 \text{ GPa}$$

$$G_{23} = 2 \text{ GPa}$$

$$G_{13} = 4 \text{ GPa}$$

The solid elements were given the following material properties:

$$E_{11} = 1.81 \text{ GPa}$$

$$E_{22} = 1.03 \text{ GPa}$$

$$E_{33} = 1.81 \text{ GPa}$$

$$\nu = 0.28$$

$$G_{12} = 7.13 \text{ GPa}$$

$$G_{23} = 5 \text{ GPa}$$

$$G_{13} = 7.13 \text{ GPa}$$

The modified material was given Young and Shear modulus values set to 10^{-6} times the original value. The models were run for different values of RR_0 and ER, as shown in Table 5.4 below. The resulting models are shown in the diagrams below. The models show only the non-modified and non-removable material.

Model No	Elements	RR_0	ER	Iterations	Volume Remaining (%)
1	Hex	0.5	0.25	136	29.03
2	Hex	1	0.5	74	32.22
3	Hex	2	1	28	33.47
4	Tetra	0.5	0.25	82	38.43
5	Tetra	1	0.5	52	31.27
6	Tetra	2	1	26	37.19

Table 5.4: Initial RR and ER values for Sandwich Model

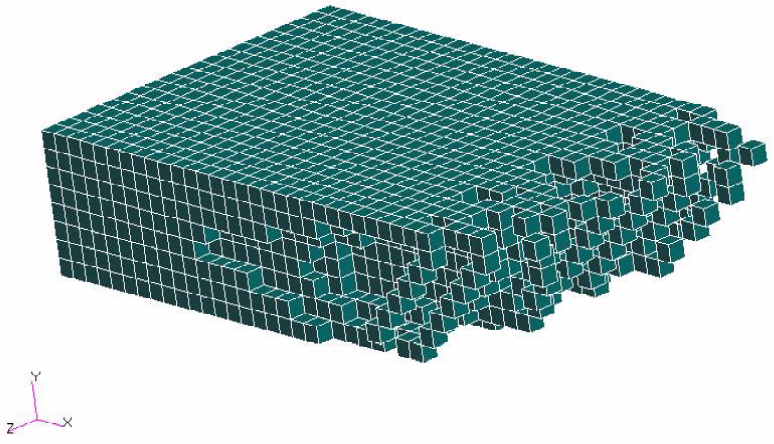


Figure 5-28: Model 1 of the Hexahedral Sandwich Structure after 50 Iterations

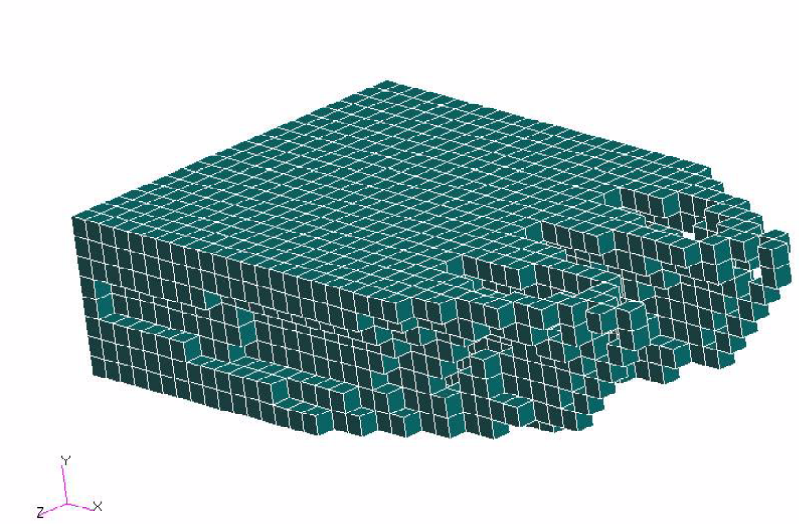


Figure 5-29: Model 1 of the Hexahedral Sandwich Structure after 90 Iterations

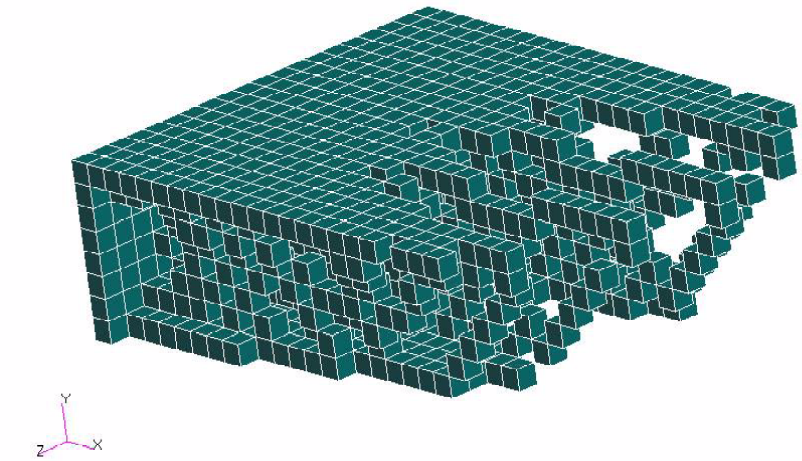


Figure 5-30: Model 1 of the Hexahedral Sandwich Structure after 136 Iterations

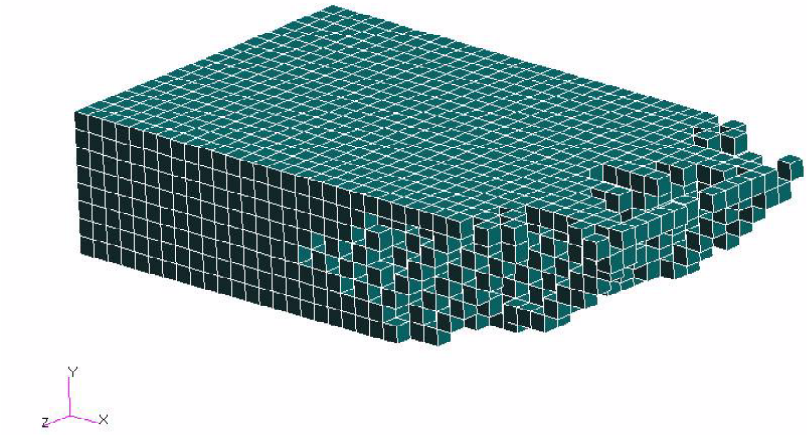


Figure 5-32: Model 2 of the Hexahedral Sandwich Structure after 21 Iterations

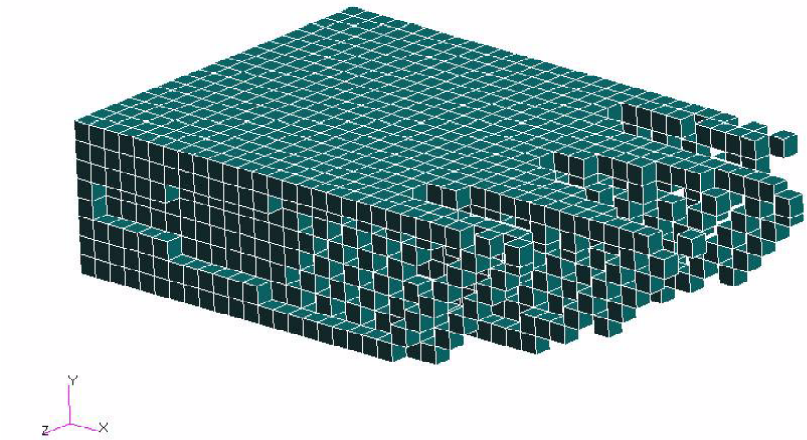


Figure 5-33: Model 2 of the Hexahedral Sandwich Structure after 49 Iterations

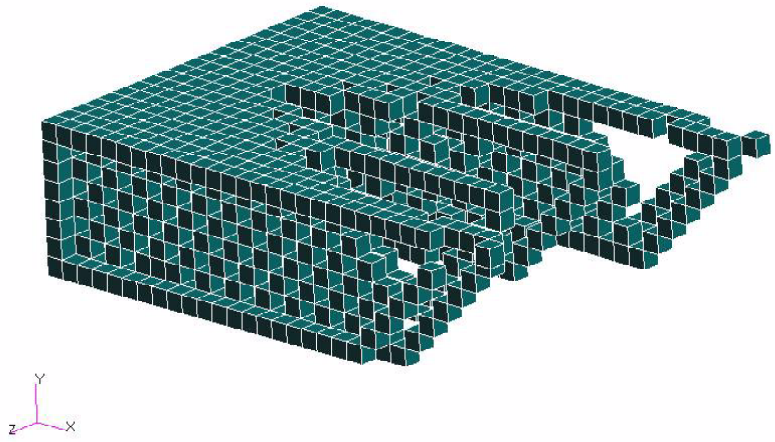


Figure 5-34: Model 2 of the Hexahedral Sandwich Structure after 75 Iterations

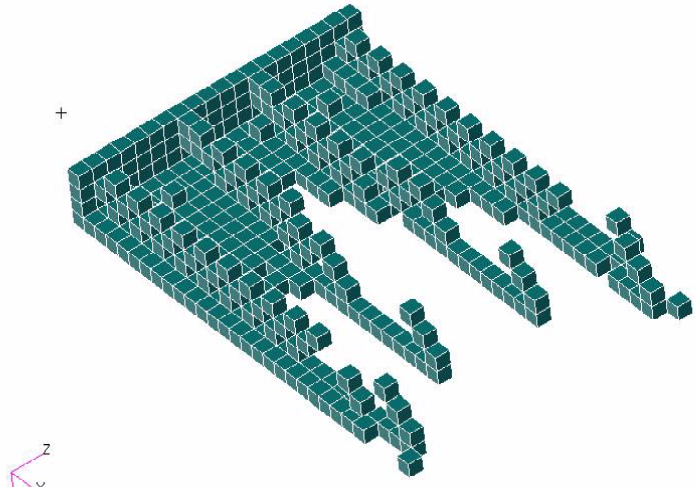


Figure 5-35: Model 2 of the Hexahedral Sandwich Structure after 75 Iterations, showing internal detailing

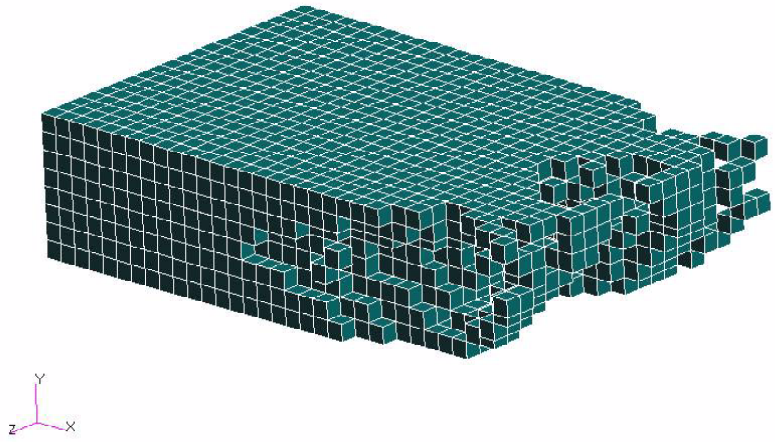


Figure 5-36: Model 3 of the Hexahedral Sandwich Structure after 12 Iterations

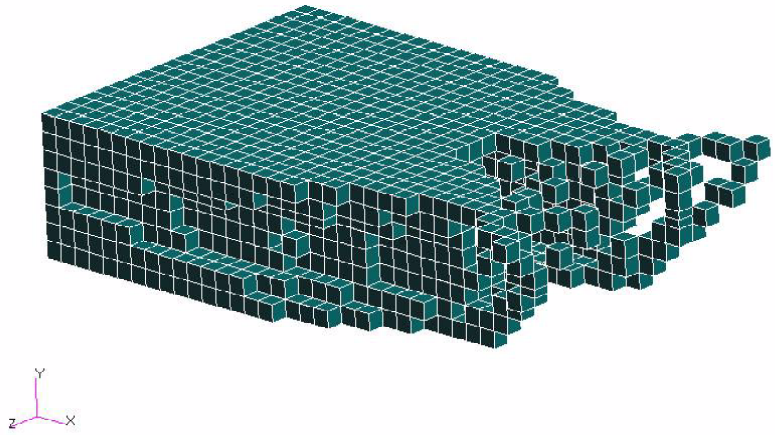


Figure 5-37: Model 3 of the Hexahedral Sandwich Structure after 24 Iterations

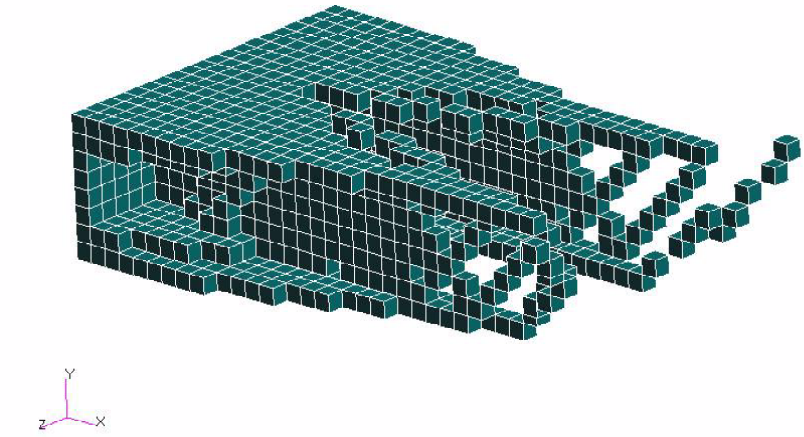


Figure 5-38: Model 3 of the Hexahedral Sandwich Structure after 28 Iterations

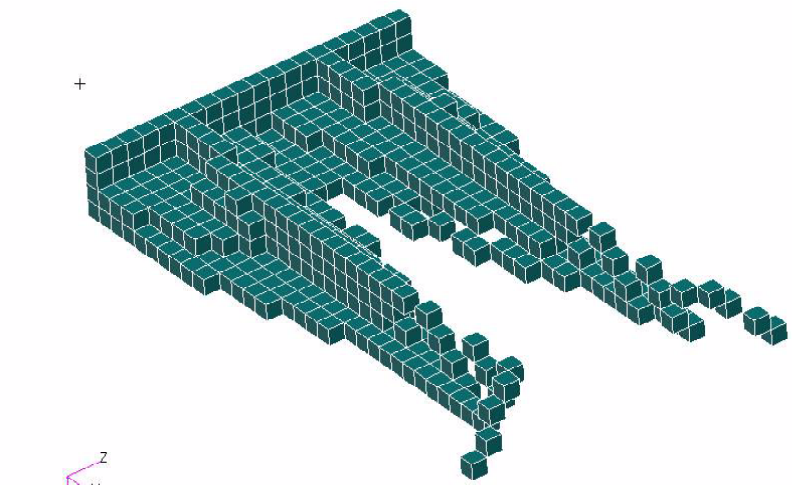


Figure 5-39: Model 3 of the Hexahedral Sandwich Structure after 28 Iterations, showing internal detailing

...

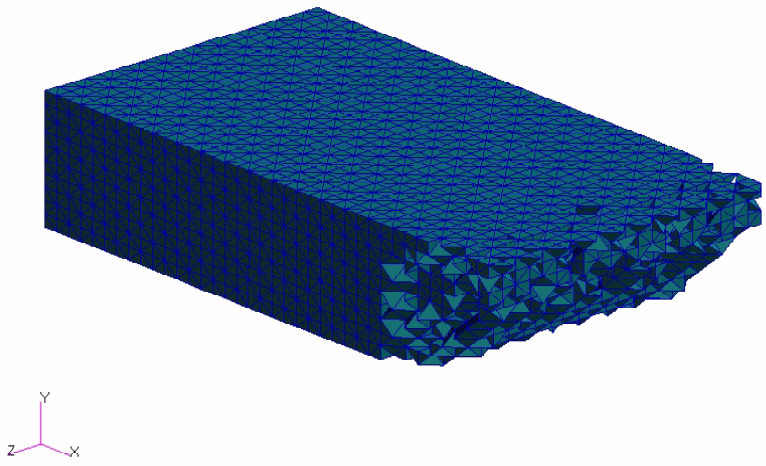


Figure 5-40: Model 4 of the Tetrahedral Sandwich Structure after 12 Iterations

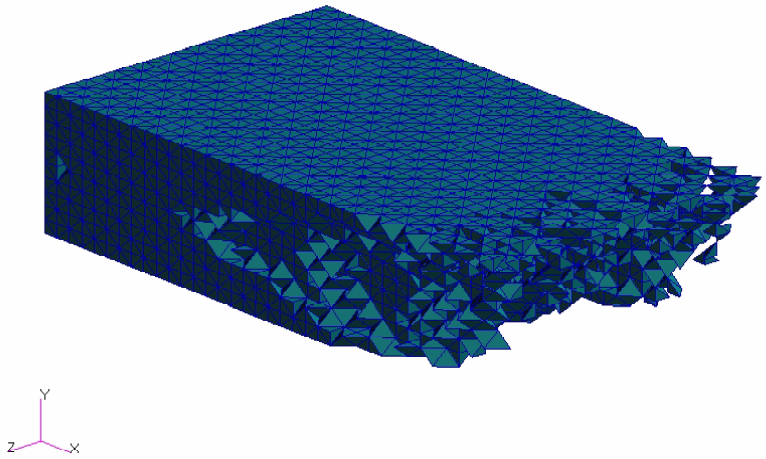


Figure 5-41: Model 4 of the Tetrahedral Sandwich Structure after 44 Iterations

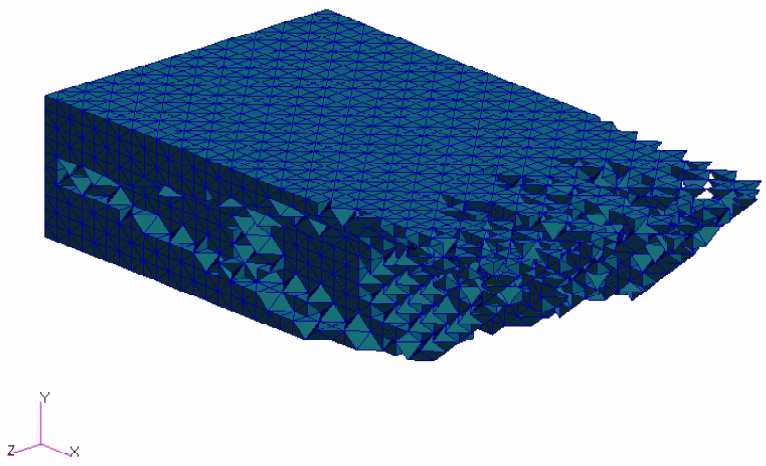


Figure 5-42: Model 4 of the Tetrahedral Sandwich Structure after 67 Iterations

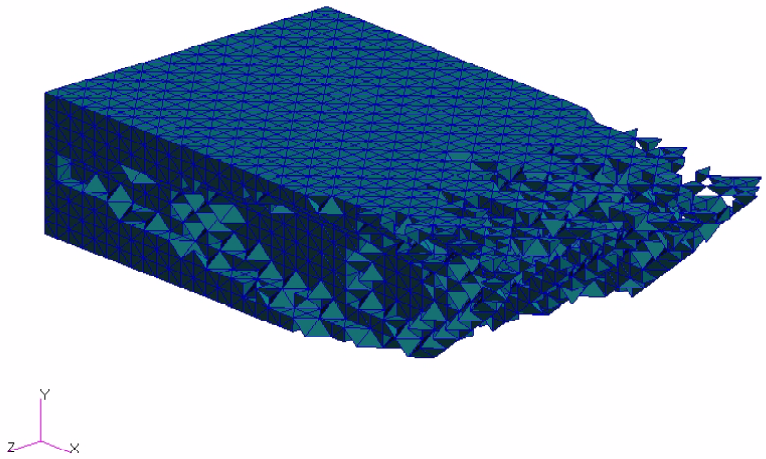


Figure 5-43: Model 4 of the Tetrahedral Sandwich Structure after 79 Iterations

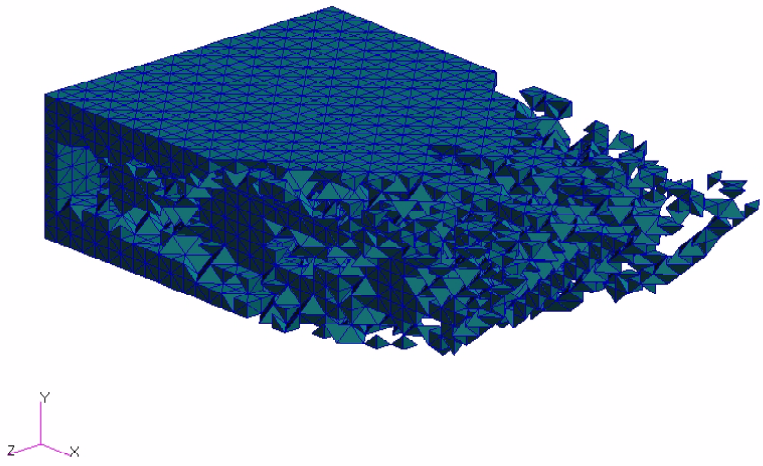


Figure 5-44: Model 4 of the Tetrahedral Sandwich Structure after 81 Iterations

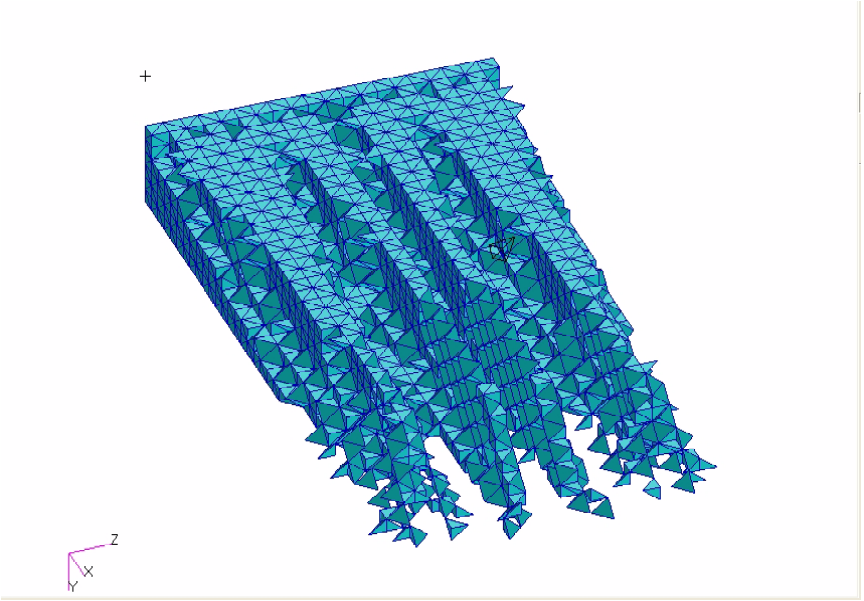


Figure 5-45: Model 4 of the Tetrahedral Sandwich Structure after 79 Iterations, showing internal details

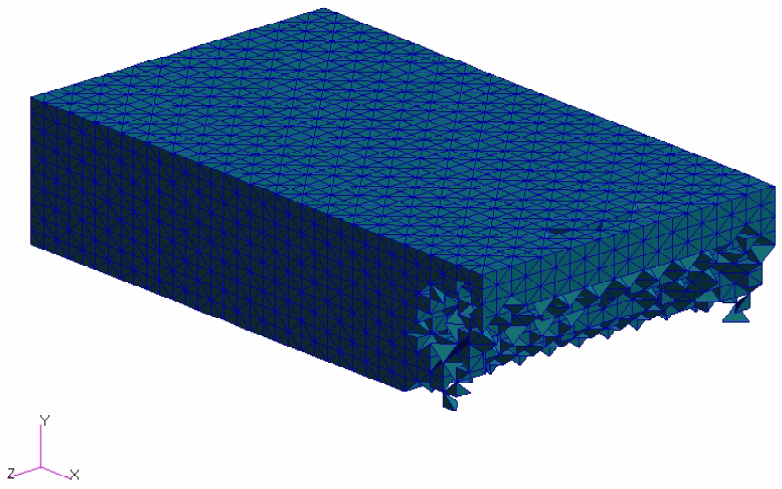


Figure 5-46: Model 5 of the Tetrahedral Sandwich Structure after 5 Iterations

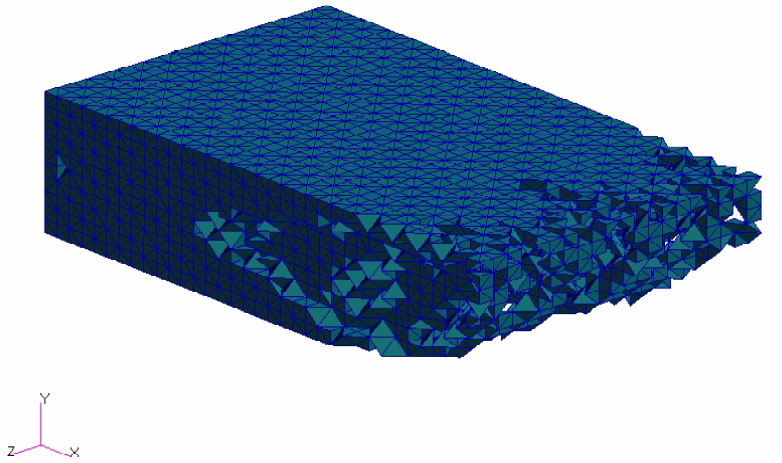


Figure 5-47: Model 5 of the Tetrahedral Sandwich Structure after 22 Iterations

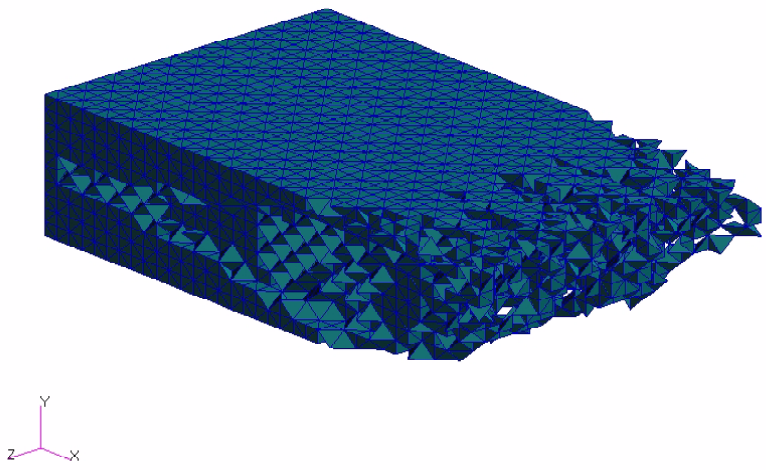


Figure 5-48: Model 5 of the Tetrahedral Sandwich Structure after 34 Iterations

.

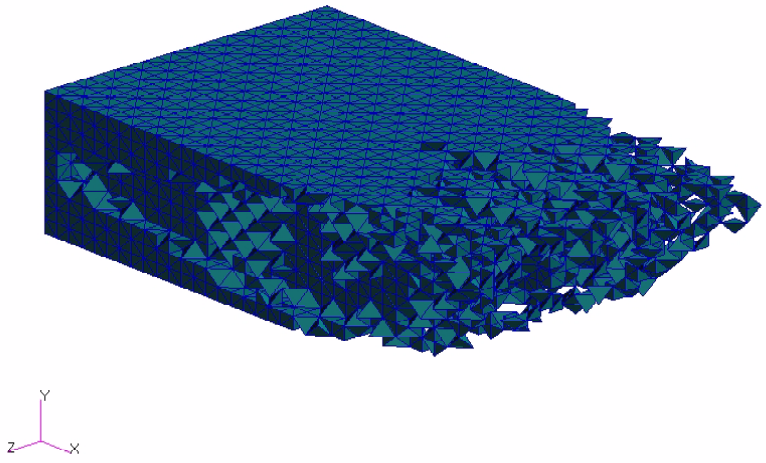


Figure 5-49: Model 5 of the Tetrahedral Sandwich Structure after 43 Iterations

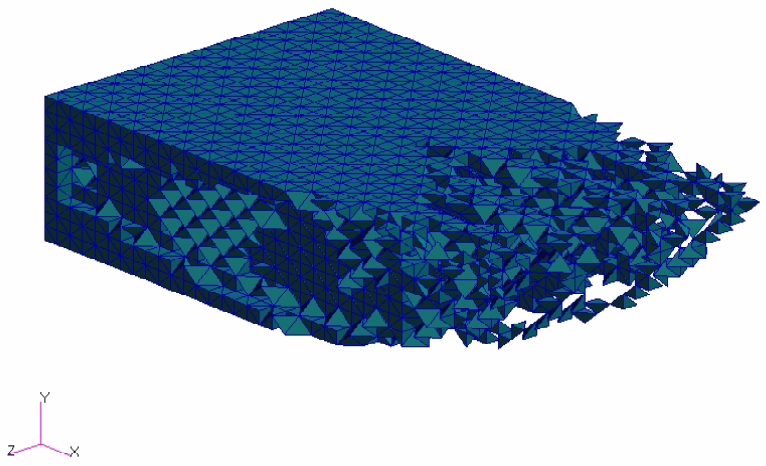


Figure 5-50: Model 5 of the Tetrahedral Sandwich Structure after 49 Iterations

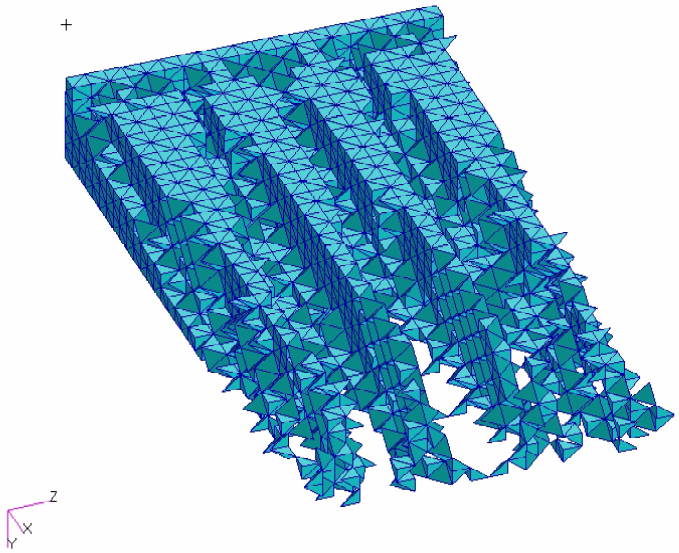


Figure 5-51: Model 5 of the Tetrahedral Sandwich Structure after 43 Iterations, showing internal details

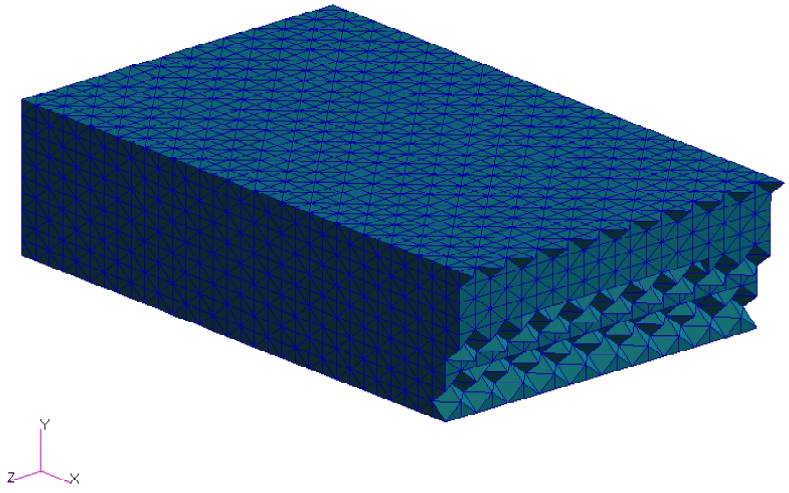


Figure 5-52: Model 6 of the Tetrahedral Sandwich Structure after 2 Iterations

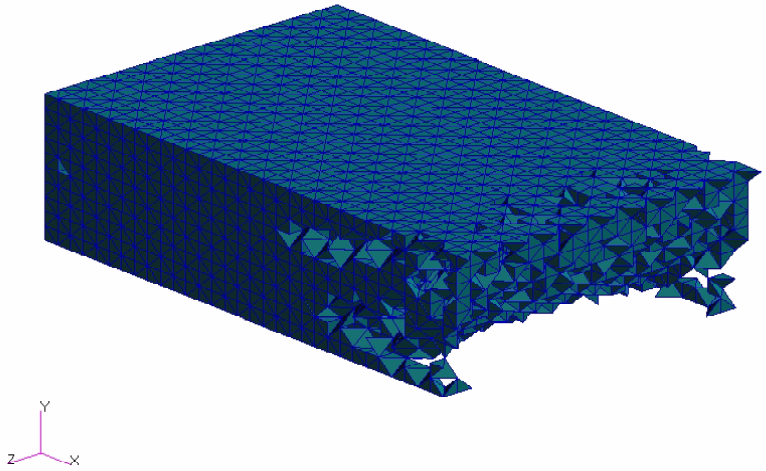


Figure 5-53: Model 6 of the Tetrahedral Sandwich Structure after 11 Iterations

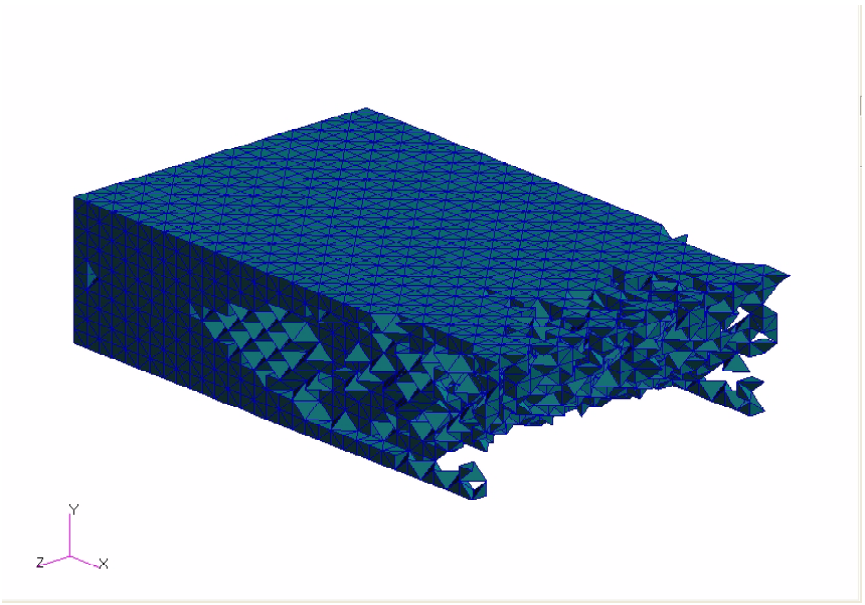


Figure 5-54: Model 6 of the Tetrahedral Sandwich Structure after 17 Iterations

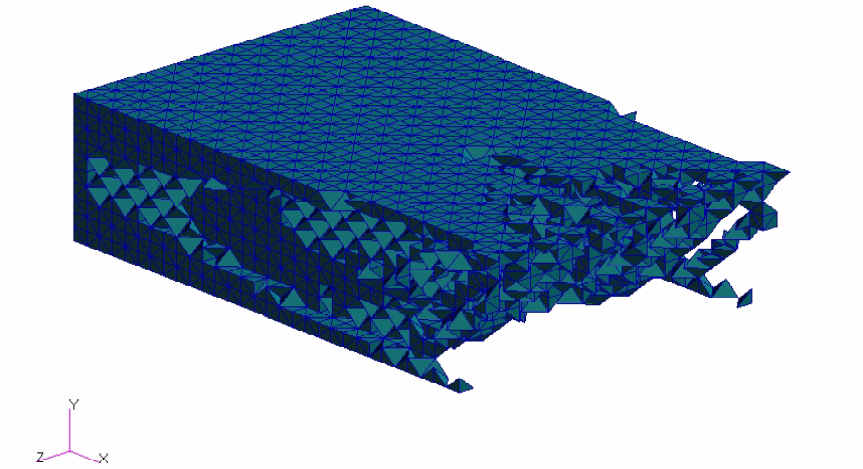


Figure 5-55: Model 6 of the Tetrahedral Sandwich Structure after 25 Iterations

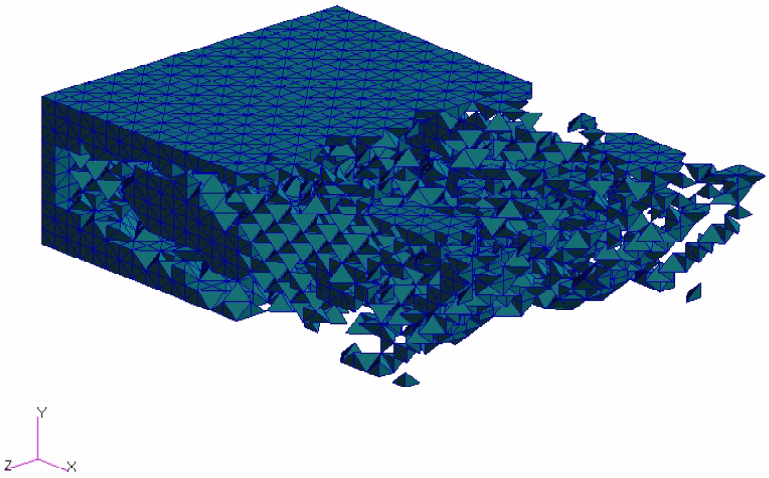


Figure 5-1: Figure 5-56: Model 6 of the Tetrahedral Sandwich Structure after 26 Iterations

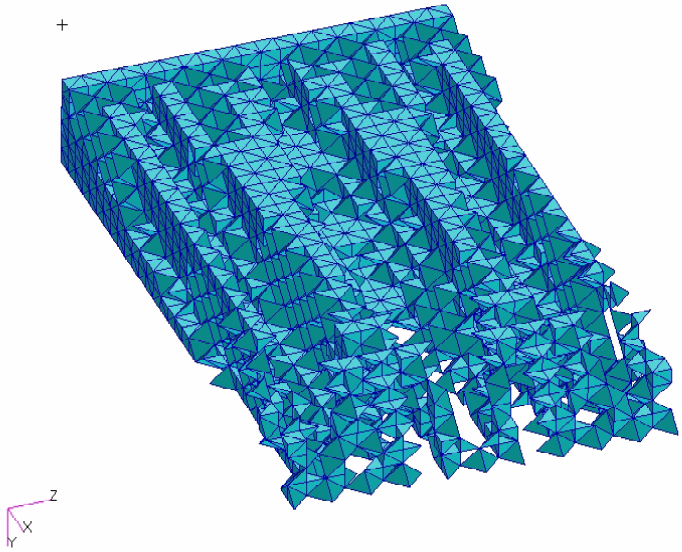


Figure 5-57: Model 6 of the Tetrahedral Sandwich Structure after 25 Iterations, showing internal details

The graphs of change in volume and stress change are in Appendix A.

The outcome of the optimisation of the two structures, while not easy to compare, do show similarities. These results show the development of a series of I-beams in the sandwich structure for both the hexahedral and tetrahedral structures. This was to be expected as the I-beam provides a good combination of properties for the carrying of loads. These similarities show that the algorithm was successfully adapted to use tetrahedral elements.

The Cantilever Beam

The cantilever beam was used to examine the effects of varying element sizes and the results such changes had on the final outcome. As was seen in the two-dimensional model, as the element size was reduced, so could the final result be refined and more volume saved. The beam was modelled as a sandwich structure and modelled with varying size tetrahedral elements. Three models were created, with 9632, 31648 and 74395 elements. These were given identical properties, loads and boundary conditions, which are shown in the diagram below.

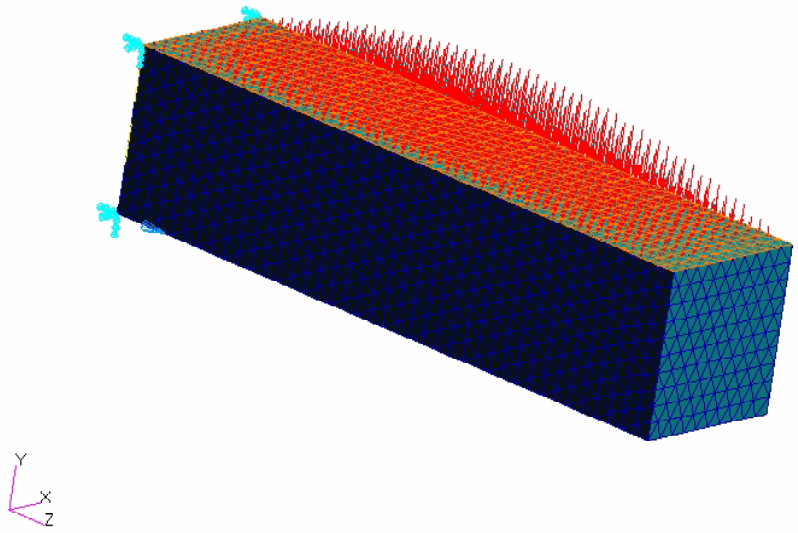


Figure 5-58: Cantilever Beam - Loads and Boundary Conditions

The load was a pressure load with a parabolic profile, distributed over the upper surface.

The skin elements were given the following material properties:

$$E_{11} = 75 \text{ GPa}$$

$$E_{22} = 45 \text{ GPa}$$

$$\nu = 0.22$$

$$G_{12} = 4 \text{ GPa}$$

$$G_{23} = 2 \text{ GPa}$$

$$G_{13} = 4 \text{ GPa}$$

The solid elements were given the following material properties:

$$E_{11} = 1.81 \text{ GPa}$$

$$E_{22} = 1.03 \text{ GPa}$$

$$E_{33} = 1.81 \text{ GPa}$$

$$\nu = 0.28$$

$$G_{12} = 7.13 \text{ GPa}$$

$$G_{23} = 5 \text{ GPa}$$

$$G_{13} = 7.13 \text{ GPa}$$

The modified material was given Young and Shear modulus values set to 10^{-6} times the original value.

These three models were given initialisation values of RR_0 equal to 1 and ER equal to 0.5. The results are shown in Table 5.5 below.

This shows that changing the element size can have an effect, improving the final results. This can be useful, especially where volume is a critical factor. The downside is an increase

Model	Iterations	Volume Remaining
9632 Elements	95	47.9
31648 Elements	76	47.55
74395 Elements	73	39.6

Table 5.5: The Effect of Varying Element Sizes

in analysis runtime. This was noted by Abolbashari and Keshavarzmanesh[24], who also noted there was a trade off between weight loss and computational time. On a Pentium 4 running at 3.2 GHz, with 2 Gigabytes RAM, the difference in analysis times for each model is significant. For the 9632 element model, each finite element analysis took one minute, for the 31648 model, an analysis took five minutes, and for the 74395 element model, an analysis took thirty minutes. The change in times for the ESO algorithm to operate was negligible. This means that the design must decide if the increase in efficiency is worth the increased amount of time to achieve the final results.

5.3.5 The Developed ESO Algorithm

As the models were developed, analysed and optimised, the ESO algorithm was refined and extended. The final ESO algorithm is presented here.

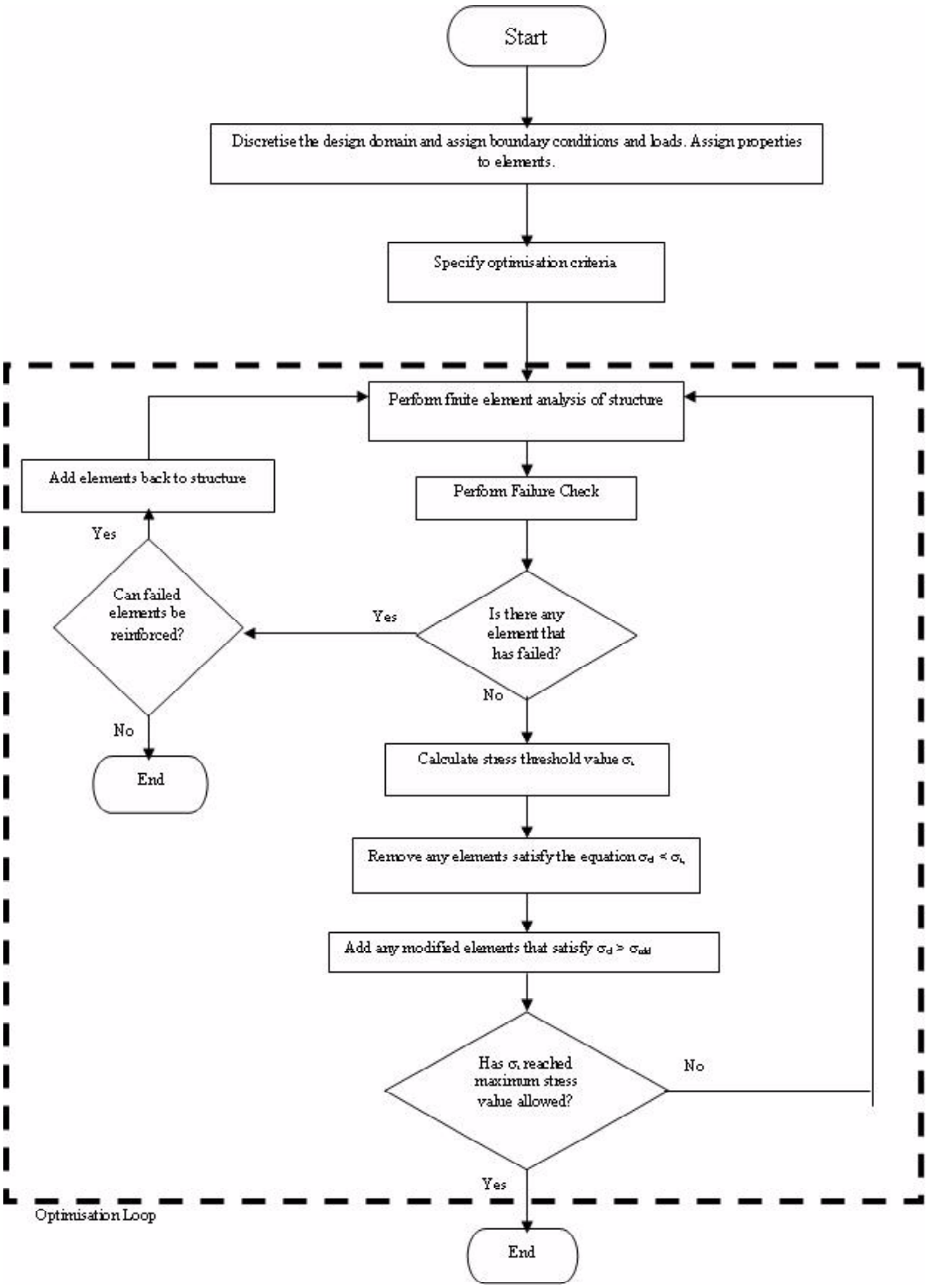


Figure 5-59: The ESO Algorithm Flowchart

The first is the design domain is discretised into 10 node tetrahedral elements.The elements are then assigned their material properties. As noted before, these properties are either removable or non-removable.The next step is the asignment of optimisation criteria. After this, the program enters the optimisation loop, as shown in Figure 5-59. The structure is then analysed under the loading conditions using a Finite Element Analysis program and the stresses are exported to the algorithm A point of concern is the possibility of localised stress concentrations as material is removed. These stress concentrations could lead to failure of the material if it is not addressed. In order to achieve this, a failure check is done on all the elements to ensure that there are none that have failed. The stresses are checked against a predetermined failure value i.e. if

$$\sigma_{el} > \sigma_{fail} \tag{5.3}$$

where σ_{el} is the element stress and σ_{fail} is the failure stress, then the element is considered to have failed. If elements have failed, the algorithm attempts to modify the structure to redistribute the stress, by adding material around the failed element. This is done by adding back all modified elements that are connected to the failed element via a common node, as can be seen in the diagram below.

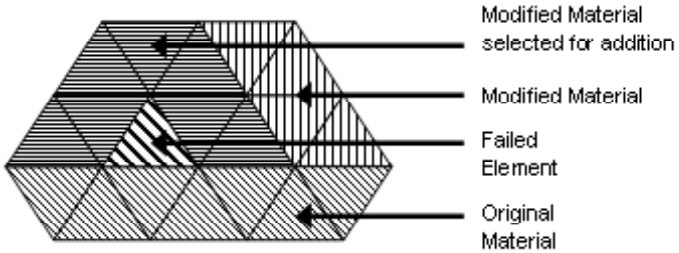


Figure 5-60: Addition of Material

If failure has occurred and material has been added back, the algorithm will send the modified structure back for analysis and start the cycle again. If no new material can be added, the structure has encountered a critical problem. The algorithm will halt the cycle and inform the user, letting the user determine what course of action to take.After the failure check, the removal threshold value for the removal of elements is determined. The value is based on the

maximum stress of the structure. The removal threshold value is given by:

$$\sigma_T = RR_i \times \sigma_{\max} \tag{5.4}$$

where RR_i is the current rejection ratio for iteration i and $\sigma_{\max}$ is the maximum value of stress in the structure. The value of the rejection ratio is given by:

$$RR_i = RR_0 + (i-1) \times ER, \quad i = 1, 2, 3, ... \tag{5.5}$$

where RR_0 is the initial rejection ratio, ER is the evolutionary rate, and i is the iteration number. With the removal threshold value, elements are selected for removal. All elements with a stress value lower than the removal threshold value will have their properties modified .i.e.

$$E_{i+1} = E \text{ and } G_{i+1} = G \text{ if } \sigma_{el} > \sigma_T \tag{5.6}$$

$$E_{i+1} = E_s \text{ and } G_{i+1} = G_s \text{ if } \sigma_{el} < \sigma_T \tag{5.7}$$

where E is the is the modulus of elasticity of the stiff material, E_s is the reduced modulus of elasticity for the removed material, and E_{i+1} is the modulus of elasticity of the element for iteration $i+1$, and similarly for G, the shear modulus. For the addition of elements, the algorithm uses a preset addition threshold value. The algorithm selects elements to add from the modified material, if the element's stress is greater than the addition threshold value i.e.

$$E_{i+1} = E \text{ and } G_{i+1} = G \text{ if } \sigma_{el} > \sigma_{add} \tag{5.8}$$

$$E_{i+1} = E_s \text{ and } G_{i+1} = G_s \text{ if } \sigma_{el} < \sigma_{add} \tag{5.9}$$

where σ_{add} is the addition threshold value. The algorithm then modifies the structure, adjusting element properties to effectively remove or add them, based on the above criteria. The structure is then resubmitted for analysis. The next step as seen in Figure 5-59 is the test to see if σ_T has reached the maximum allowable value. If it has then the optimisation routine comes to a halt, otherwise the loop cycles again.

Operation of the ESO Algorithm

The algorithm was written into code using Compaq Visual FORTRAN V6.6a and was designed to work with MSC.Patran/Nastran 2005r2. The code for the algorithm is contained in Appendix B. The program consisted of a master control program, which ran the entire process, and sub-programs which performed specific functions. Data for the program, such as what model files to use, and algorithm data such as RR_0, ER, and the various limits were written into a readable file. The program was designed to run without any further user input, unless it came upon a problem such as failure, as was described previously. The program stores the results of each iteration for later inspection. Results such as stress limits, elements removed and added and any others required are written to separate files for further examination. Once the algorithm has been completed, the resulting model files can be imported back into MSC.Patran for further analysis.

Chapter 6

The Application of the ESO Algorithm

6.1 Introduction

This chapter will present the application of the developed ESO algorithm to the sample problem chosen, the tailfin of an Uninhabited Aerial Vehicle (UCAV). This involved the creation of the computer model, with applicable loads and boundary conditions, as well as specified material properties. Two models were created, one with approximately 100,000 elements, and the other with 300,000 elements. The first model was used to give a rough guide to the expected shape, and the latter model was used for a more accurate result. The algorithm was then ready to be applied to the model, and the final optimised shape was obtained.

6.2 The UCAV Tailfin

The sample problem chosen was the tailfin of a UCAV, an unmanned aerial vehicle.

6.2.1 The UCAV Tailfin

The UCAV under consideration employs 2 tailfins. These tailfins are all-moving, with no other aerodynamic aids. This means that the entire tailfin moves when required to alter the UCAV's aerodynamics. This is done by means of the control shaft connecting the tailfin to the UCAV.

The tailfin pivots around this control shaft as required. The tailfin would be manufactured out of advanced composite materials, principally carbon fibre, and would consist of skins and an internal structure to give it strength.

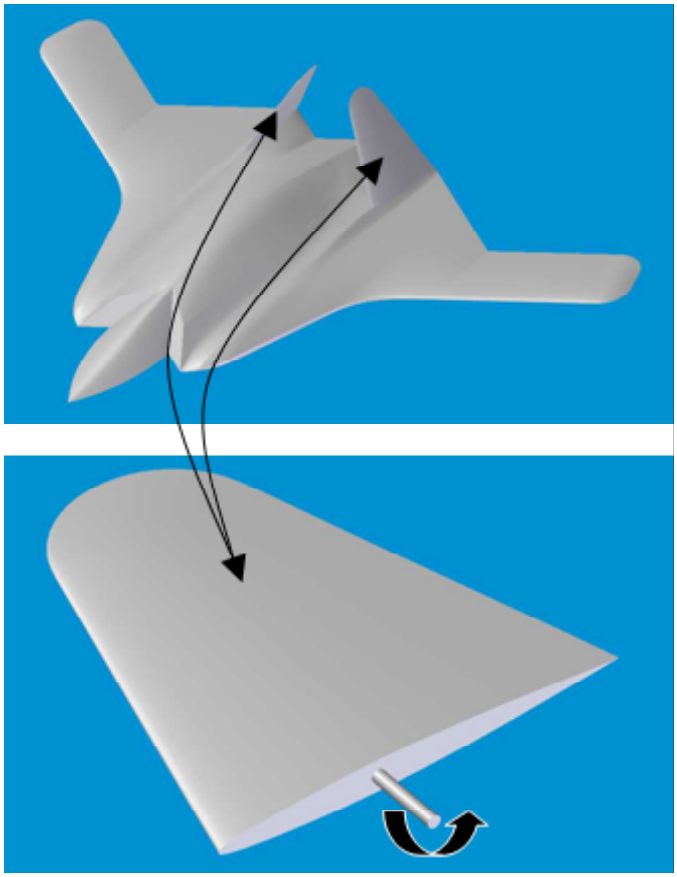


Figure 6-1: The UCAV and Tailfin

6.2.2 Description of Optimisation Problem

The optimal design of the UCAV tailfin is considered for the case where the tailfin is subjected to aerodynamic loading. The aim of the optimisation problem is to produce an optimal structural design, that of a tailfin with minimum internal volume whilst maintaining structural integrity. The objective is to optimise the tailfin, and create a final design from the results. The goals of the optimisation is to remove excess weight while retaining a structure that can carry the given loading conditions. In order to achieve this, the tailfin will be modelled using the finite element method, and the developed ESO algorithm will be applied.

6.2.3 Computer Modelling of the UCAV Tailfin

For the analysis and optimisation of the UCAV Tailfin, the internal core would be optimised, with the external geometry set by design and aerodynamics. This model would be placed under an aerodynamic load, with boundary conditions simulating the control shaft.

Element Properties

The tailfin was modelled as a combination of 3D tetrahedral elements, representing the core, and 2D shell elements representing the skins. The skin was given orthortopic material properties, matching the properties of the carbon fibre from which it would be constructed. The skin was then given the properties:

$$E_{11} = 75 \text{ GPa}$$

$$E_{22} = 45 \text{ GPa}$$

$$\nu = 0.22$$

$$G_{12} = 4 \text{ GPa}$$

$$G_{23} = 2 \text{ GPa}$$

$$G_{13} = 4 \text{ GPa}$$

In order to lower the stress gradient on the skin due to the multipoint constraints that were applied, the skin was given a varying thickness across its surface. It was thickest where the constraints were attached. and stepped down to a minimum thickness near the edges. This can be seen in the diagram below, where red represents the thickest section, green a medium thickness, and white representing the minimum thickness.

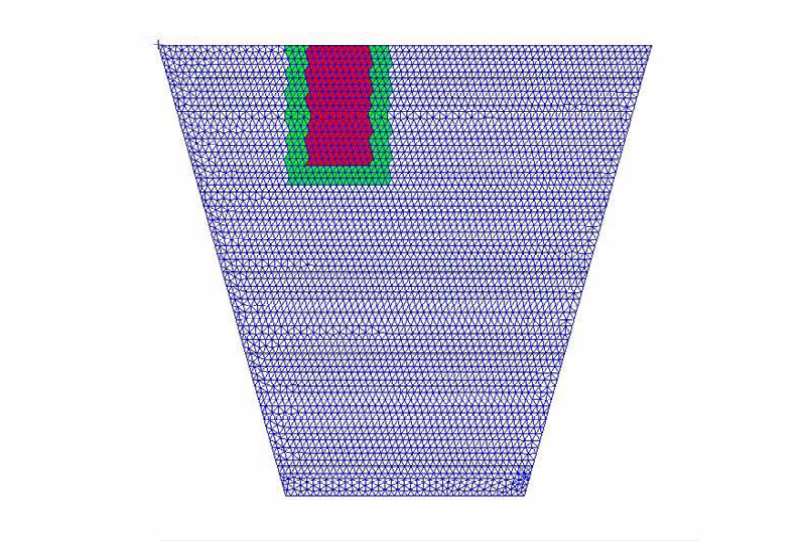


Figure 6-2: Skin Thickess for UCAV Tailfin

The core was modelled using a tailored material. This material did not represent a real, available core material, but rather represented an ideal material. This assumption was based on the ability to tailor composite materials to the required application. The properties were specified such that shear strength was dominant. This is due to the tailfin being comparable with a sandwich structure. This means that the skins would be carrying tensile and compressive loads, and the core would principally be carrying shear loads. The properties were then specified as:

$$E_{11} = 14 \text{ GPa}$$

$$E_{22} = 14 \text{ GPa}$$

$$E_{33} = 14 \text{ GPa}$$

$$\nu = 0.8$$

$$G_{12} = 30 \text{ GPa}$$

$$G_{23} = 30 \text{ GPa}$$

$$G_{13} = 30 \text{ GPa}$$

The modified materials were given by:

$$E_{11} = 14 \text{ kPa}$$

$$E_{22} = 14 \text{ kPa}$$

$$E_{33} = 14 \text{ kPa}$$

$$\nu = 0.8$$

$$G_{12} = 30 \text{ kPa}$$

$$G_{23} = 30 \text{ kPa}$$

$$G_{13} = 30 \text{ kPa}$$

Tailfin Loading

An aerodynamic pressure load was applied to the tailfin model, and was calculated using lift and the location of the mean centre of aerodynamic pressure. The pressure loading was applied to the upper surface of the wing. The mean centre of aerodynamic pressure was taken as 25% along the mean aerodynamic chord. The diagram below shows the load distribution on the upper surface of the wing.

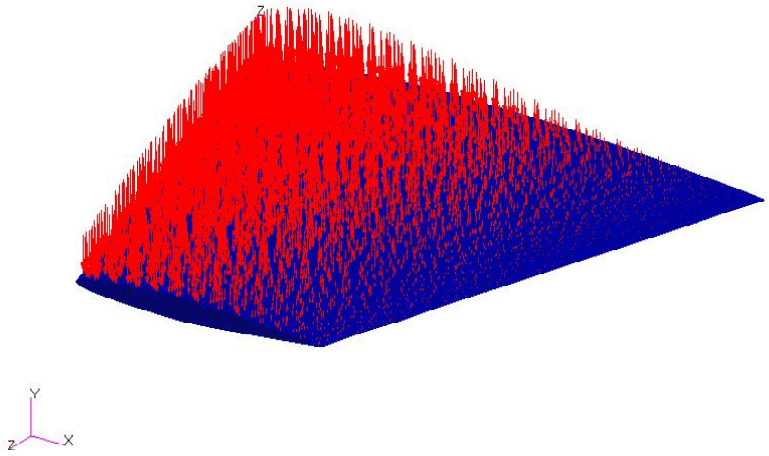


Figure 6-3: Loading on UCAV Tailfin Upper Surface

Boundary Conditions for the UCAV Tailfin

The model of the tailfin was clamped on nodes where the control shaft was attached to the tailfin. In addition, rigid multipoint constraints were applied, connecting the skin to the clamped nodes. This provided a suitable connection between the skin and boundary conditions.

Mesh Creation for the UCAV Tailfin

It was decided to create two separate models for the UCAV tailfin, with different numbers of elements for each model. The low element count model would be used to guide the process, and develop the initial set of conditions for the optimisation process, and the high element count model would be used to provide the final results for the optimisation process. The low element model consisted of approximately 100,000 elements, while the high element model had approximately 300,000 elements. The model was created using the specified geometry of the UCAV tailfin to create the core, following which the skin elements were created on the surface of the core.

6.3 Optimisation Results for the UCAV Tailfin

The results for the two sets of optimisation undertaken are presented here. As noted before, the 100,000 element model was used as a test to obtain the best values of ER and RR_0 for the optimisation of the full 300,000 element model. This was done as the run time for the finite element method portion of the optimisation process was much shorter for the smaller model, which meant that the results could be obtained more quickly, and hence a greater variety of values could be tested. The results were inspected and the best values chosen for use in the larger model. Finally, from the results of the optimisation of the larger model were used in the creation of a final model for the internal structure of the tailfin.

6.3.1 Results for 100,000 Element Model

Based on the experience gained in the development of the ESO algorithm, values were chosen for ER and RR_0, as seen in Table 6.1 below.

Model	RR_0	ER	Iterations	Fail
1	1	0.5	40	No
2	2	1	17	No
3	5	1	18	Yes

Table 6.1: Initial RR and ER Values for 100,000 Element Model

The first two sets of values were similar to values chosen in the development process of the algorithm. The final set was used to determine the effect of setting a high value for RR_0, in order to try and reduce the run time for the total analysis. The models were run until the preset maximum stress was reached. The number of iterations is shown in the table, and whether the optimisation was considered to have failed or not. The failure was determined by the response of the model and based on how the model performed over the various iterations. The progress of the first two models is shown in the diagrams below, and the graphs of the stress change is shown in the appendix.

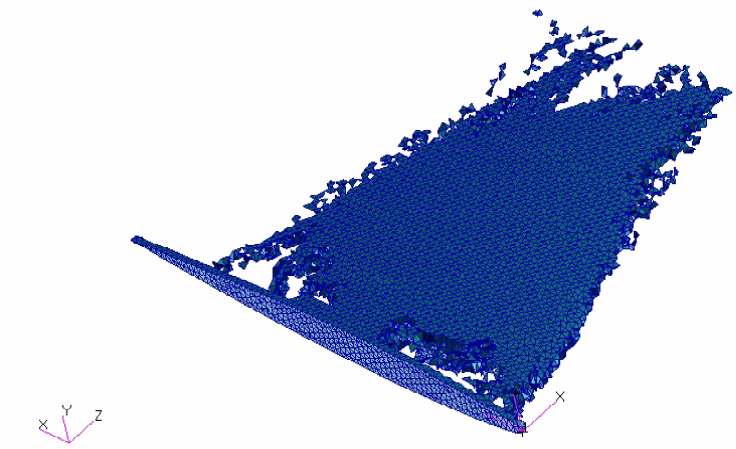


Figure 6-4: Tailfin Model 1 after 11 Iterations

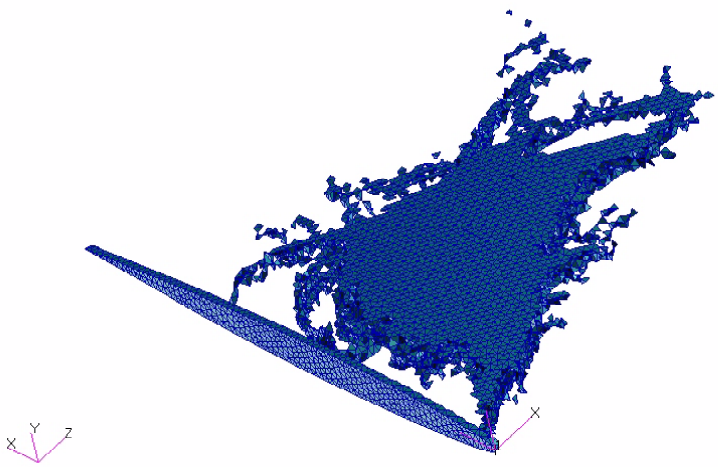


Figure 6-5: Tailfin Model 1 after 22 Iterations

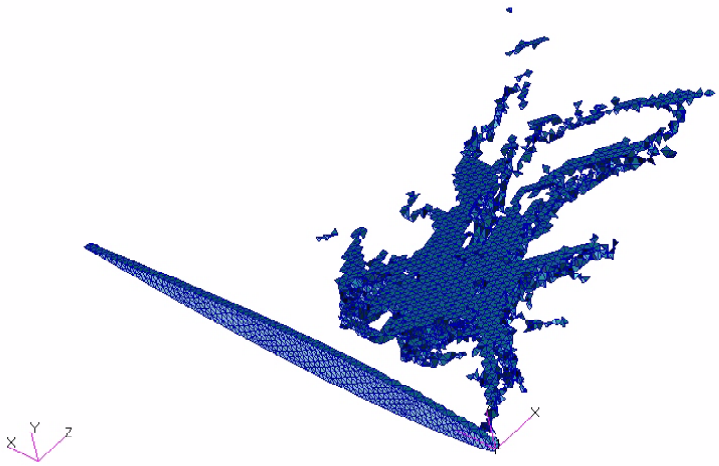


Figure 6-6: Tailfin Model 1 after 34 Iterations

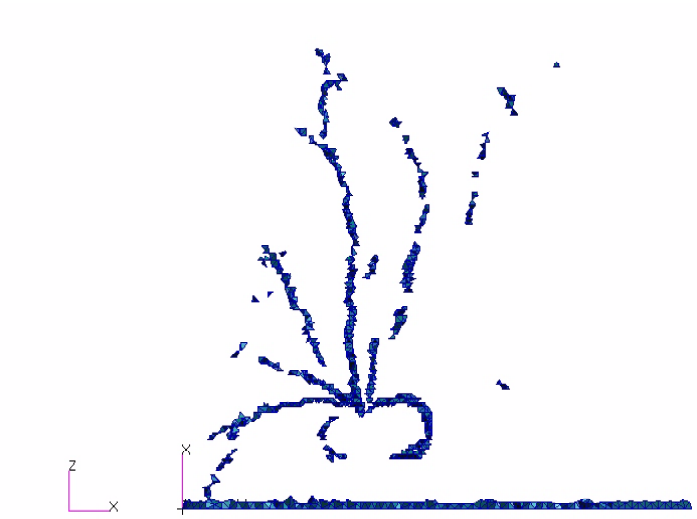


Figure 6-7: Tailfin Model 1 after 34 Iterations, showing Internal Details

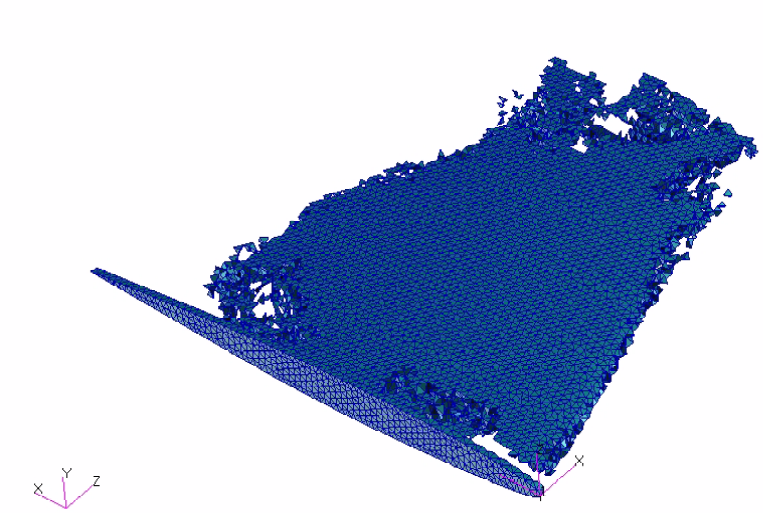


Figure 6-8: Tailfin Model 2 after 5 Iterations

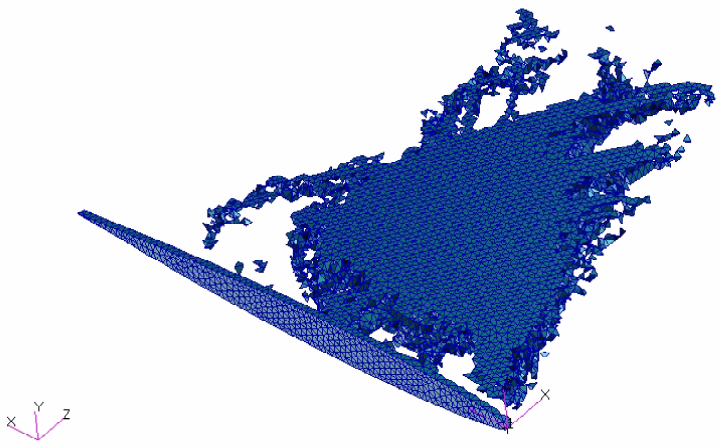


Figure 6-9: Tailfin Model 2 after 10 Iterations

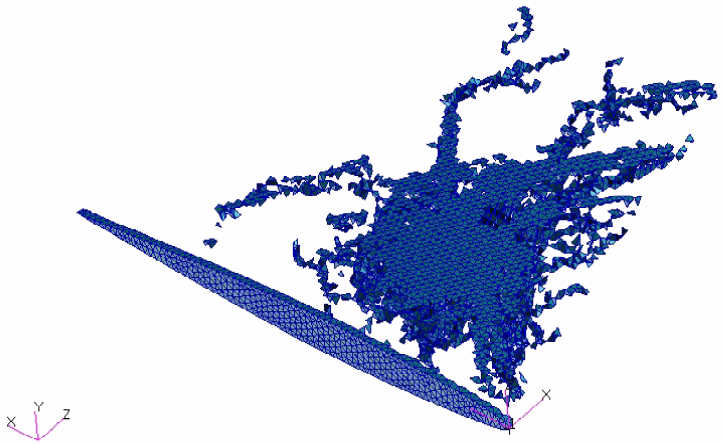


Figure 6-10: Tailfin Model 2 after 15 Iterations

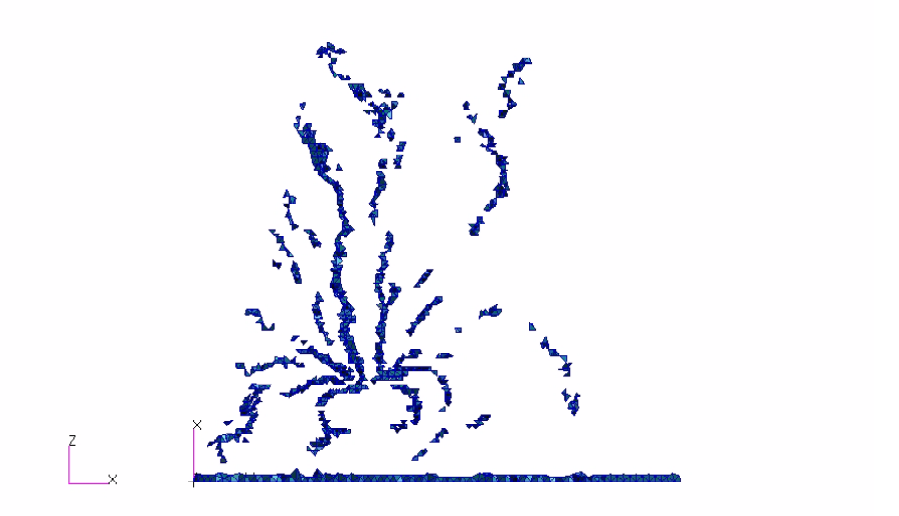


Figure 6-11: Tailfin Model 2 after 15 Iterations, showing Internal Details

The similarities between the two models can be seen. The cross-sections show the development of a series of curved webs running through the tailfin. These webs act much like I-beams, and was to be expected, based on the loads and boundary conditions inposed.

The final model was determined to have failed, based on the response of the model over the successive iterations. In this model, the maximum stress rose sharply after the first iteration, and continued to climb over the remainder of the iterations. This can be explained by the results for the first iteration. In this case, too much material was removed from the model, due to the high RR_0, principally from the centre of the core. This can be seen in Figure 6-12 below, which shows a cross-section across the centre of the tailfin, showing the lack of material.

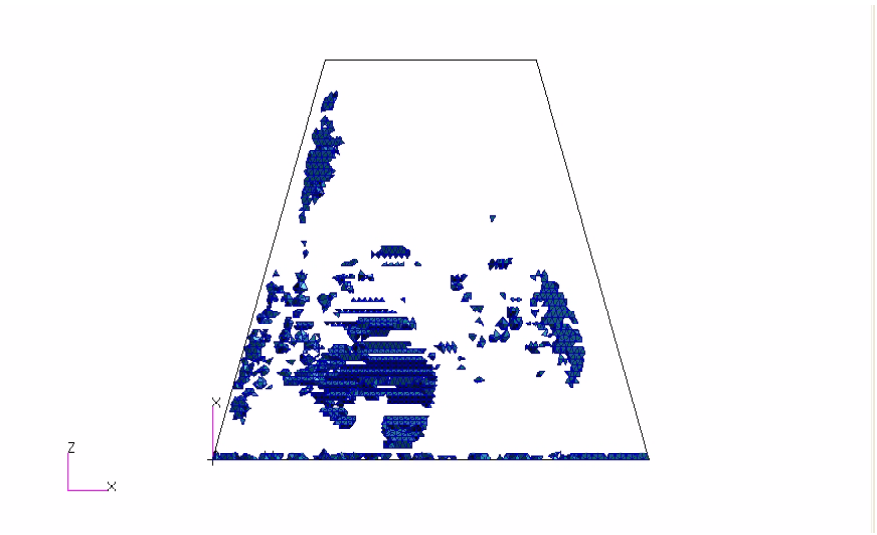


Figure 6-12: Tailfin Model 3 after 1 Iteration, showing Internal Details.

This led to a poor response from the model over the successive iterations, which could not be corrected by the failure mechanisms in the algorithm This was due to the fact that the model loads were now being carried primarily by the skin. This failure shows the effect of an incorrectly selected RR_o value.

6.3.2 Results for the 300,000 Element Model

With the initial analysis completed using the 100,000 element model, the 300,000 element model was then analysed and optimised. For the intial RR_0 and ER values, the values from the second 100,000 element model were used. In order to obtain a finer set of results the value of ER was varied. These values, and the number of iterations, are shown in Table 6.2 below.

Model	RR_0	ER	Iterations
4	2	2	6
5	2	1	12
6	2	0.5	21

Table 6.2: Initial RR and ER values for 300,000 Element Model

The development of the models can be seen in the diagrams below, and the graphs showing the change in stress can be seen in the appendix.

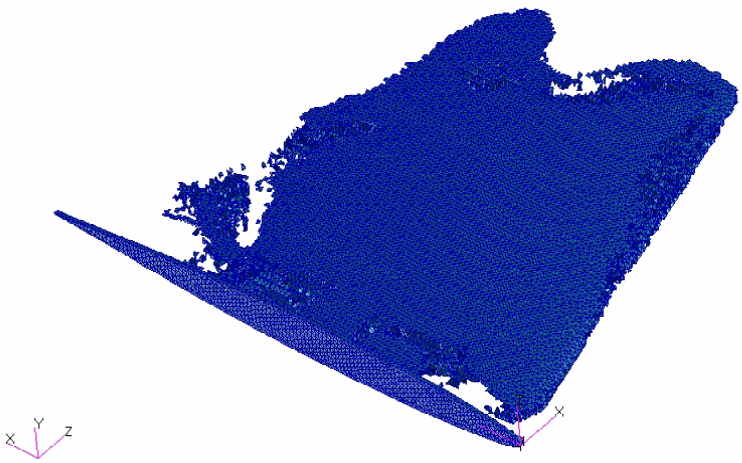


Figure 6-13: Tailfin Model 4 after 2 Iterations

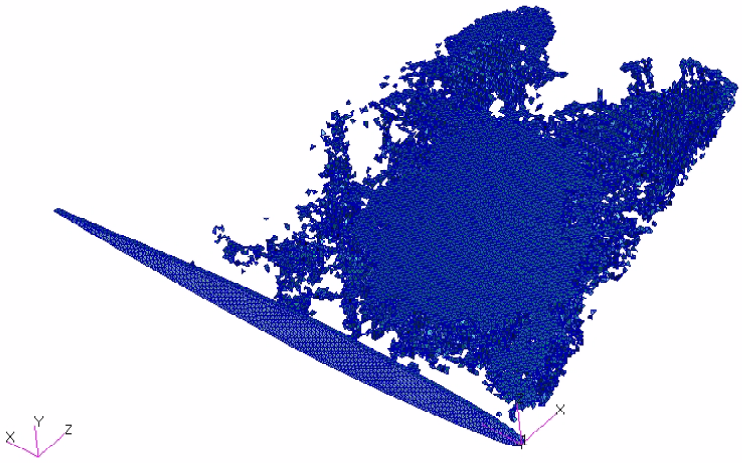


Figure 6-14: Tailfin Model 4 after 4 Iterations

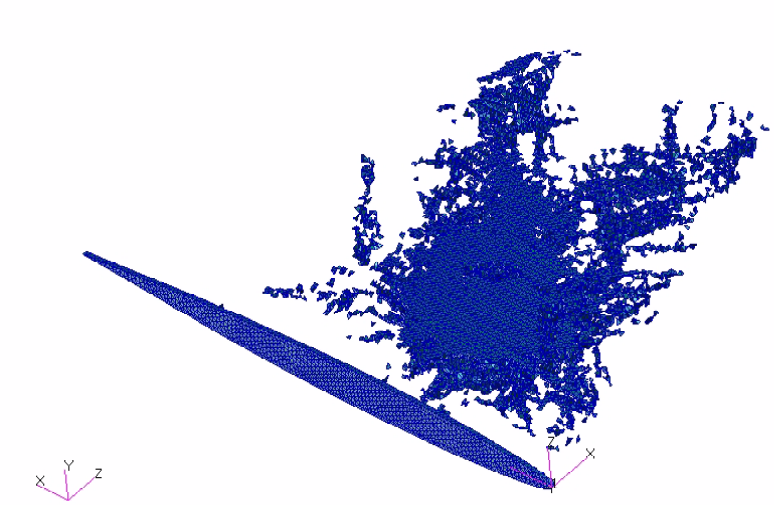


Figure 6-15: Tailfin Model 4 after 6 Iterations

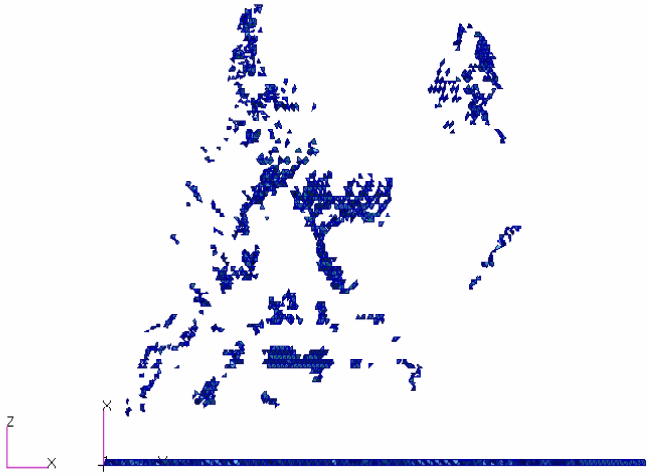


Figure 6-16: Tailfin Model 4 after 6 Iterations, showing Internal Details

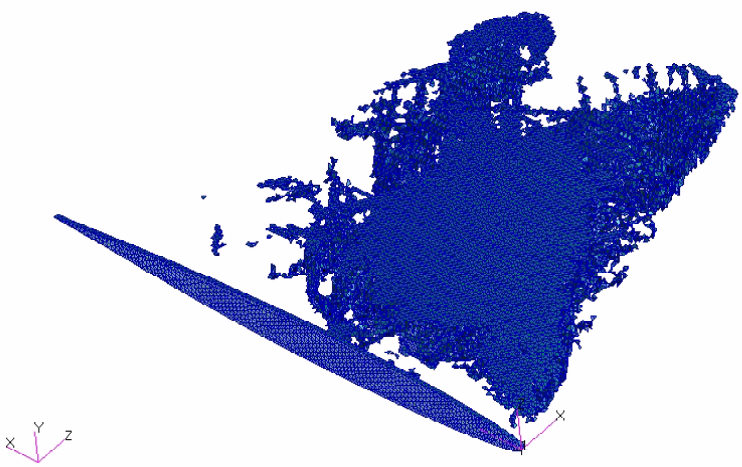


Figure 6-17: Tailfin Model 5 after 4 Iterations

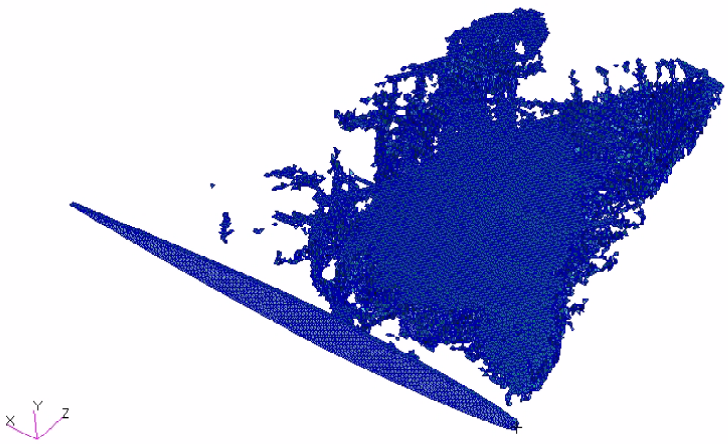


Figure 6-18: Tailfin Model 4 after 8 Iterations

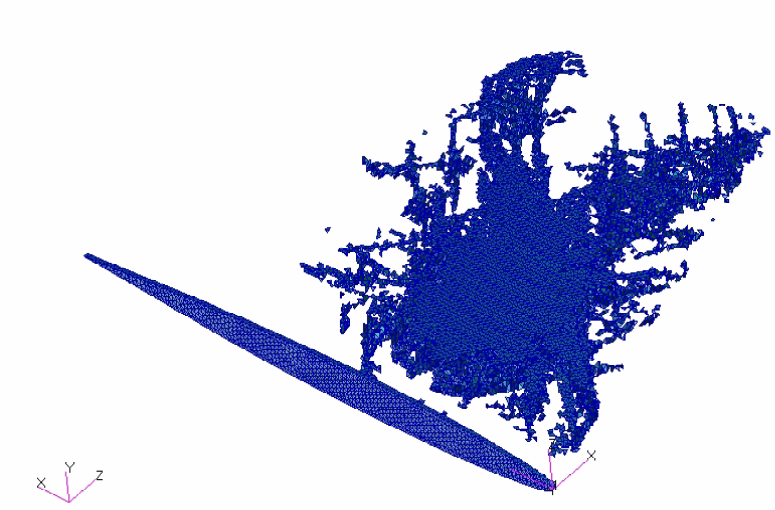


Figure 6-19: Tailfin Model 5 after 11 Iterations

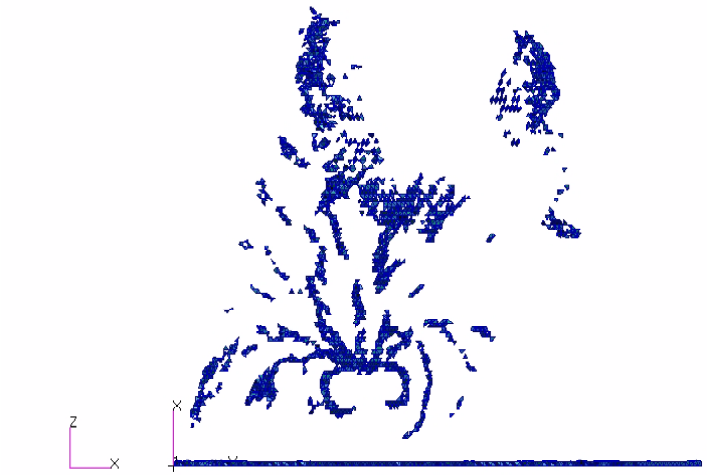


Figure 6-20: Tailfin Model 5 after 11 Iterations, showing Internal Details

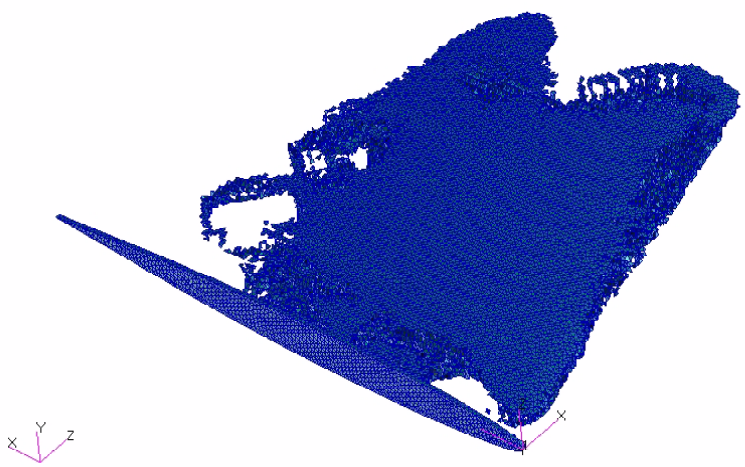


Figure 6-21: Tailfin Model 6 after 7 Iterations

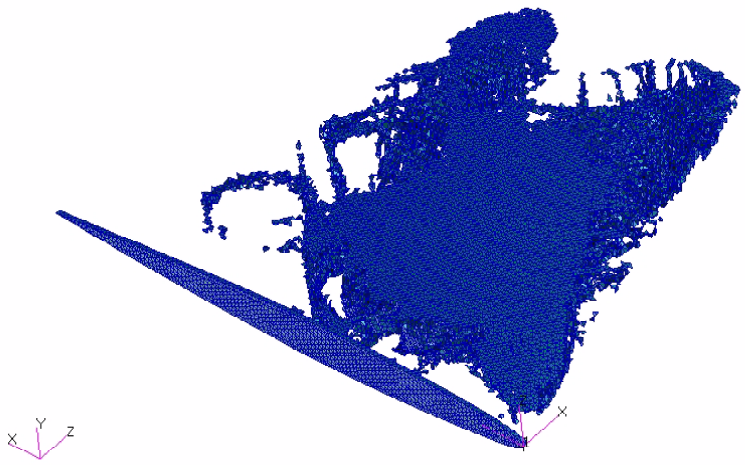


Figure 6-22: Tailfin Model 6 after 14 Iterations

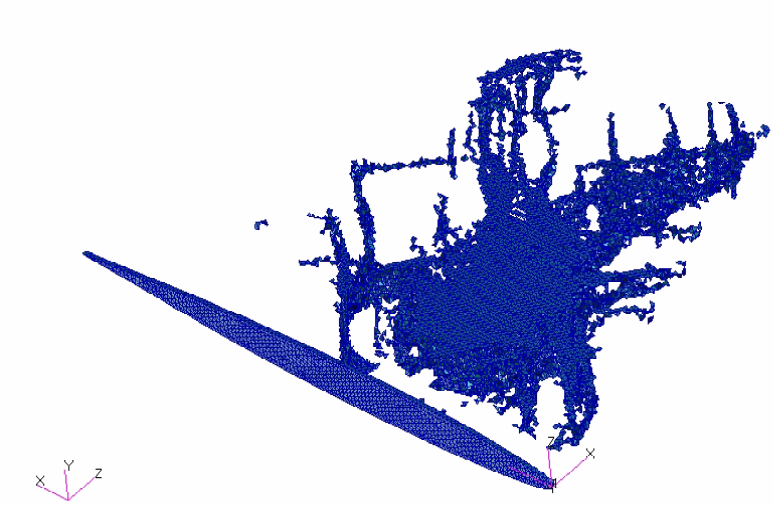


Figure 6-23: Tailfin Model 6 after 20 Iterations

Figure 6-24: Tailfin Model 6 after 20 Iterations, showing Internal Details

The results of the first model are not ideal, as it can be seen that a similar problem has emerged to the results of the first 100,000 element model, in that too much material has been removed from the structure. As for the second and third models, the results are similar to the ones obtained from the 100,000 element model, namely a series of curved webs, acting as I-beams.

6.3.3 Final Model

For the development of the final design, the results of the various iterations were inspected and compared. The model was developed with particular reference to the second 300,000 element model, most particularly iterations 10 and 11, as it was felt these represented the results most clearly, giving a good combination of volume saving and structural integrity. The model consists of a series of curved I-beams spreading through the tail, as can be seen in the diagram below.

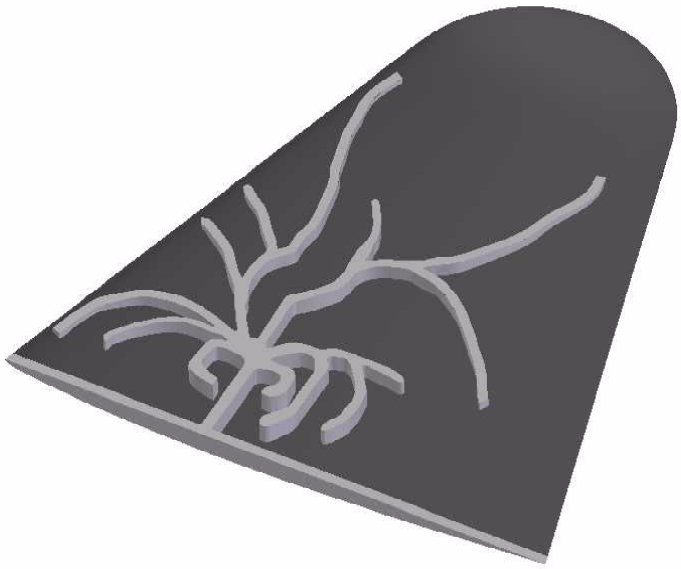


Figure 6-25: Final Design for Internal Stucture

6.4 Discussion

The goal of this work was the development of an ESO algorithm for use with advanced composite structures. In order to demonstrate this, the tailfin of a UCAV was chosen to show the effectiveness of the method. The developed ESO algorithm was successfully applied to the UCAV tailfin. From these results, a design for the internal structure of the tailfin was developed.

It is impotant to note that the results of the ESO technique show here were qualitative, not quantitative results. These results would be used perhaps as a starting point for a more comprehensive design process. The technique allows for the designer to obtain the general shape of an optimal structure for given loads and conditions. This allows for a large amout of time

to be saved at the start of the design process, allowing for a shorter design schedule, or a more comprehensive final design.

6.5 Conclusion

The successful application of the developed ESO algorithm to the test struture of the UCAV tailfin was shown here. The results were then used to create a tailfin design, which would be used as a starting point for a more rigorous optimisation process.

Chapter 7

Conclusion

An Evolutionary Structural Optimisation technique was developed for the use of advanced composite structures and was demonstrated using the example of the tailfin of a UCAV. The combination of the finite element method and the ESO algorithm provides the developed technique with a wide range of abilities , with the finite element method providing the ability to analyse complex structures, and the ESO algorithm providing the abilty to optimise a structure efficiently. This combination provides a versatile and effective method to optimise advanced composite structures.

The development of the algorithm was based on the simple ESO algorithm first presented by Xie and Steven. Form this basis it was developed unsing a series of test models, to expand into an algorithm suitable for use in the optimisation of the test composite structure. From the original simple algorithm, the technique was expanded from two dimensions to 3, from handling isotropic materials to orthotropic materials. This development ensured that the developed algorithm was filled the required specifications, and that the results of the final test optimisation would be valid.

The finite element model of the UCAV tailfin was created, and the developed ESO algorithm was applied to it. The optimisation proved successful, and the results were used to develop a design for the internal structure of the tailfin. This design was a a series of webs, shaped in a 'palm-leaf' design. These results reflect that the tailfin acts as a sandwich structure, and

the I-beams are a efficient shape to carry such loads. This shape is in contrast to the typical internal structure of ribs and spars. The result can be compared to that of Jordan [28]. the study there presented a comparison between the standard rib and spar design, and a web of I-beams for the internal structure of the aerodynamic surface, in this case the wing and canard of a different UCAV. It was found that the web displayed superior properties, with a saving in mass. Comparing the web of Jordan, with the web here, it can be seen that the ESO web is much more complex. However, the principle remains the same, and so it can be seen that the work confirms that of Jordan. The results from the UCAV tailfin show that the developed ESO algorithm is a viable method for the optimisation of advanced composite structures

The ESO algorithm developed for the application to advanced composite structures is a simple but effective tool for a designer. The technique is not without problems which could be addressed in the future. The ESO algorithm is sensitive to the selection of the RR_o and ER values. With incorrect algorithm initialisation, the initial error is compounded over the course of the optimisation, as can be seen in the case of the third 100,000 element tailfin model. This can be corrected with experience in the techniqe, or by the use of a smaller model to test the effects of varying the values. The algorithm is sensitive to changes in the maximum stress in the structure, and can be disrupted if localised stress concentrations occur. This could be corrected by ensuring localised stresses are not taken into account in the determination of σ_t, the threshold stress. In general, the ESO algorithm is effective, obtaining results in a short amount of time. This technique could be compared to a more formal optimisation method, to obtain a measure of its efficiency in comparison to standard techniques.

The behaviour of the algorithm did vary, depending on the model, but the following generalisations could be drawn. The volume removed every step over the successive iterations gradually decreased as the structure became more uniformly stressed. This can be compared to the work of Savas *et al* [25], who used an ESO algorithm to create structures with a unifrom stress distribution. There tended to be no asymptote, as eventually too much material would be removed, and the stress distribution in the structure would change, leading to changes in the material removed. This point, which can be seen in the stress graphs of the structures, can be considered an end point, following which the results are no longer valid. The algorithm is fairly

adaptable, able to handle varying problems without significant changes required. The limit is the structure's ability to be modelled accurately using the finite element method, and the validity of the output results. Provided these conditions are satisfied, the developed algorithm could be used in a wide variety problems.

The ESO algorithm created for this work is very easy to modify, and can be developed further to provide better optimisation results. The sample problem here consisted of only one load case. It would be advantageous to extend the ability of the algorithm to handle multiple load cases. A further adaptation could be the use of deflection as a criteria. either in determining the end point or failure. The modification of material could be modified, so that the material has several different options, corresponding to different construction options. The principal restriction on the further development of the algorithm is the ability of the finite element analysis to provide the required results. If the results for a particular problem are available, there is little reason why the ESO algorithm could not be developed to take advantage of those results.

Bibliography

[1] Bannister, M. (2001) Challenges for Composites into the Next Millenium - a Reinforcement Perspective. *Composites: Part A*.

[2] Staab, G.H. (1999) *Laminar Composites*. Butterworth-Heinemann, Boston.

[3] Jones, R.M. (1975) *Mechanics of Composite Materials*. McGraw-Hill, New York.

[4] Herakovich, C.T. (1998) *Mechanics of Fibrous Composites*. John Wiley and Sons.

[5] Gurdal, Z., Haftka, R.T. and Hajela, P. (1999) *Design and Optimisation of Laminated Composite Materials*. John Wiley and Sons, New York.

[6] Middleton, H.D. (1990) *Composite Materials in Aircraft Structures*. Longman Scientific and Technical, England.

[7] Forsythe, G.E. and Wasow, W.R. (1960) *Finite Difference Methods for Partial Differential Equations*. John Wiley and Sons, New York.

[8] Richtmeyer, R.D. and Morton, K.W. (1967) *Difference Methods for Initial Value Problems*. Wiley-Interscience, New York, 2nd edn.

[9] Oden, J.T. and Reddy, J.N. (1976) *An Introduction to the Mathematical Theory of Finite Elements*. John Wiley and Sons, New York.

[10] Zienkiewicz, O.C. (1977) *The Finite Element Method*. McGraw-Hill Book Company, New York, 3 edn.

[11] Felippa, C.A. and Clough, R.W. (1970) The finite element method in solid mechanics. vol. 2, pp. 210–252, SIAM-AMS Proceedings, American Mathematical Society, Providence, R.I.

[12] Oliviera, E.R.A. (1968) Theoretical foundations of the finite element method. *Int. J. Solids Struct.*, **4**, 929–952.

[13] Matthews, F.L., Davies, G.A.O., Hitchings, D. and Soutis, C. (2000) *Finite Element Modelling of Composite Materials and Structures*. Woodhead Publishing Limited, Cambridge.

[14] Zhao, Z. (1991) *Shape Design Sensitivity Analysis and Optimisation Using the Boundary Element Method*. Springer-Verlag, Berlin.

[15] Rao, S. (1996) *Engineering Optimisation: Theory and Practice*. John Wiley and Sons, 3rd edn.

[16] Querin, O.M. (1997) *Evolutionary Structural Optimisation: Stress Based Formulation and Implementation*. Ph.D. thesis, Department of Aeronautical Engineering, University of Sydney.

[17] Fletcher, R. (1987) *Practical Methods of Optimization*. John Wiley and Sons, 2nd edn.

[18] Xie, Y.M. and Steven, G.P. (1993) A Simple Evolutionary Procedure for Structural Optimisation. *Computers and Structures*, **49**.

[19] van Gemert, R.J. (1996), Additive evolutionary structural optimisation.

[20] Young, V., Querin, O.M., Steven, G.P. and Xie, Y.M. (1999) 3-d and multiple load case bi-directional evolutionary structural optimisation (beso)'. *Struct. Optimisation*, **18**, 183–192.

[21] Chu, D.N., Xie, Y.M., Hira, A. and Steven, G.P. (1997) On various aspects of evolutionary structural optimization for problems with stiffness constraints. *Finite Elements in Analysis and Design*, **24**, 197–212.

[22] Li, Q., Steven, G.P., Querin, O.M. and Xie, Y.M. (1999) Shape and topology design for heat conduction by evolutionary structural optimization. *International Journal of Heat and Mass Transfer*, **42**, 3361–3371.

[23] Tanskanen, P. (2002) The Evolutionary Structural Optimisation Method: Theoretical Aspects. *Comput. Meth. Appl. Mech. Eng.*, **191**.

[24] Abolbashari, M.H. and Keshavarzmanesh, S. (2005) On Various Aspects of Application of the Evolutionary Structural Optimisation Method for 2D and 3D Continuum Structures. *Finite Elements in Analysis and Design*.

[25] Savas, S., Ulker, M. and Saka, M.P. (2003) Evolutionary topological design of three dimensional structures. *Proceedings of the 9th International Conference on Civil and Structural Engineering Computing*, no. 132.

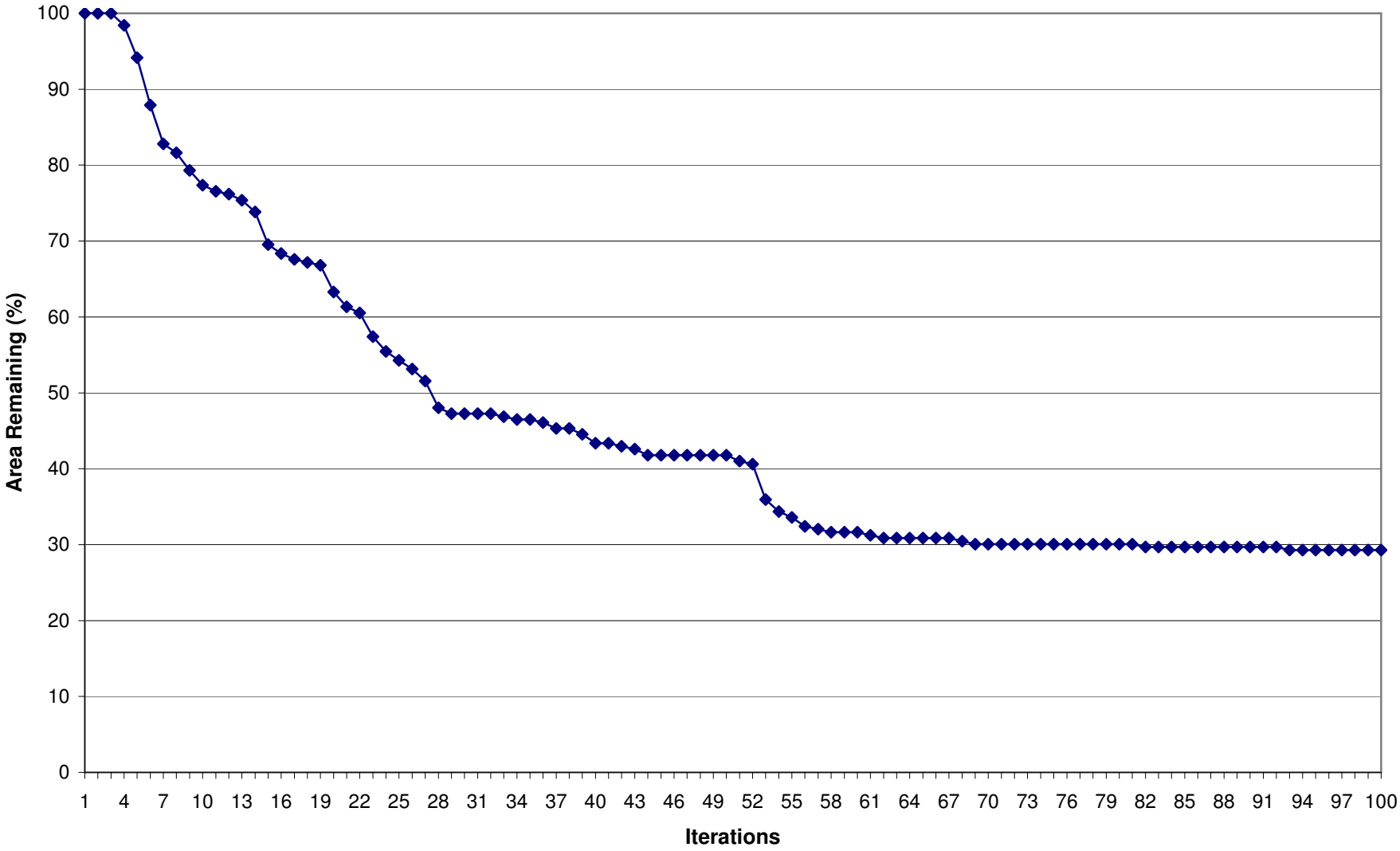
[26] Rozvany, G.I.N and Querin, O.M. Present Limitations and Possible Improvements in SERA (Sequential Element Rejection and Admissions) methods in Topology Optimisation.

[27] Walker, M. and Smith, R.E. (2003) A simple self-design methodology for laminated composite structures to minimize mass. *Advances in Engineering Software*, **34**.

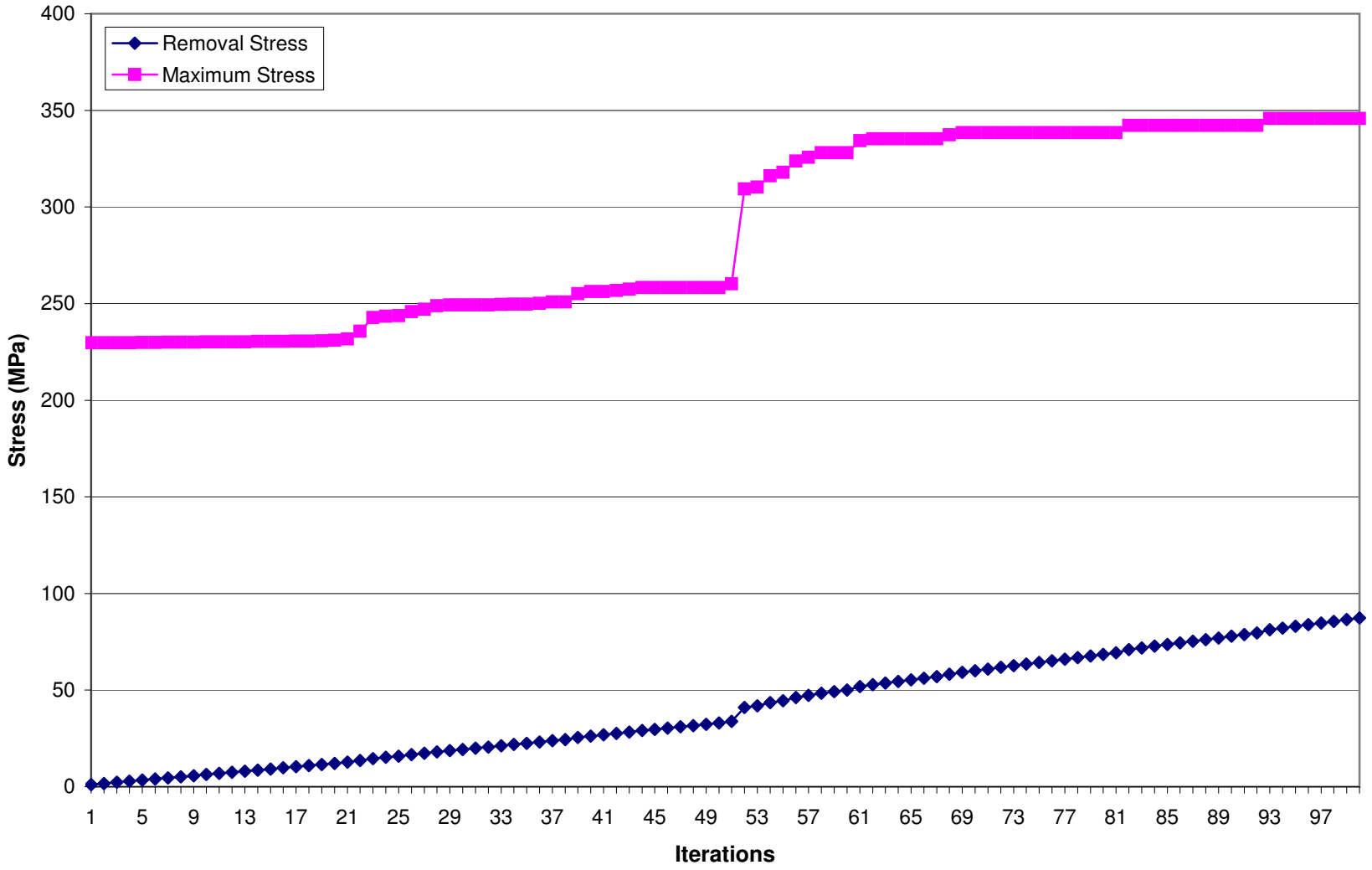
[28] Jordan, K. (2005) *Masters Thesis*. Master's thesis, Durban University of Technology.

Appendix A

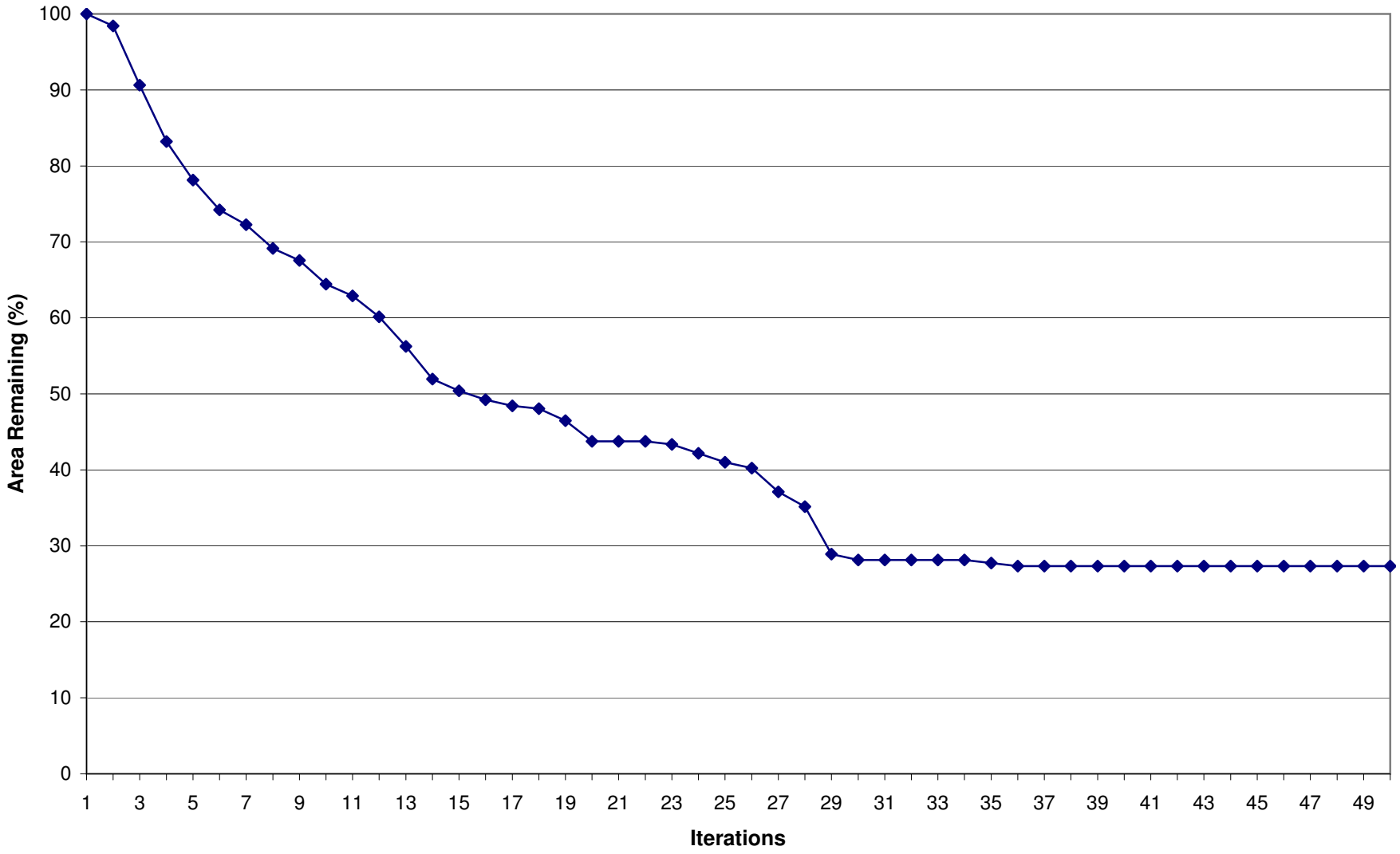
Graphs



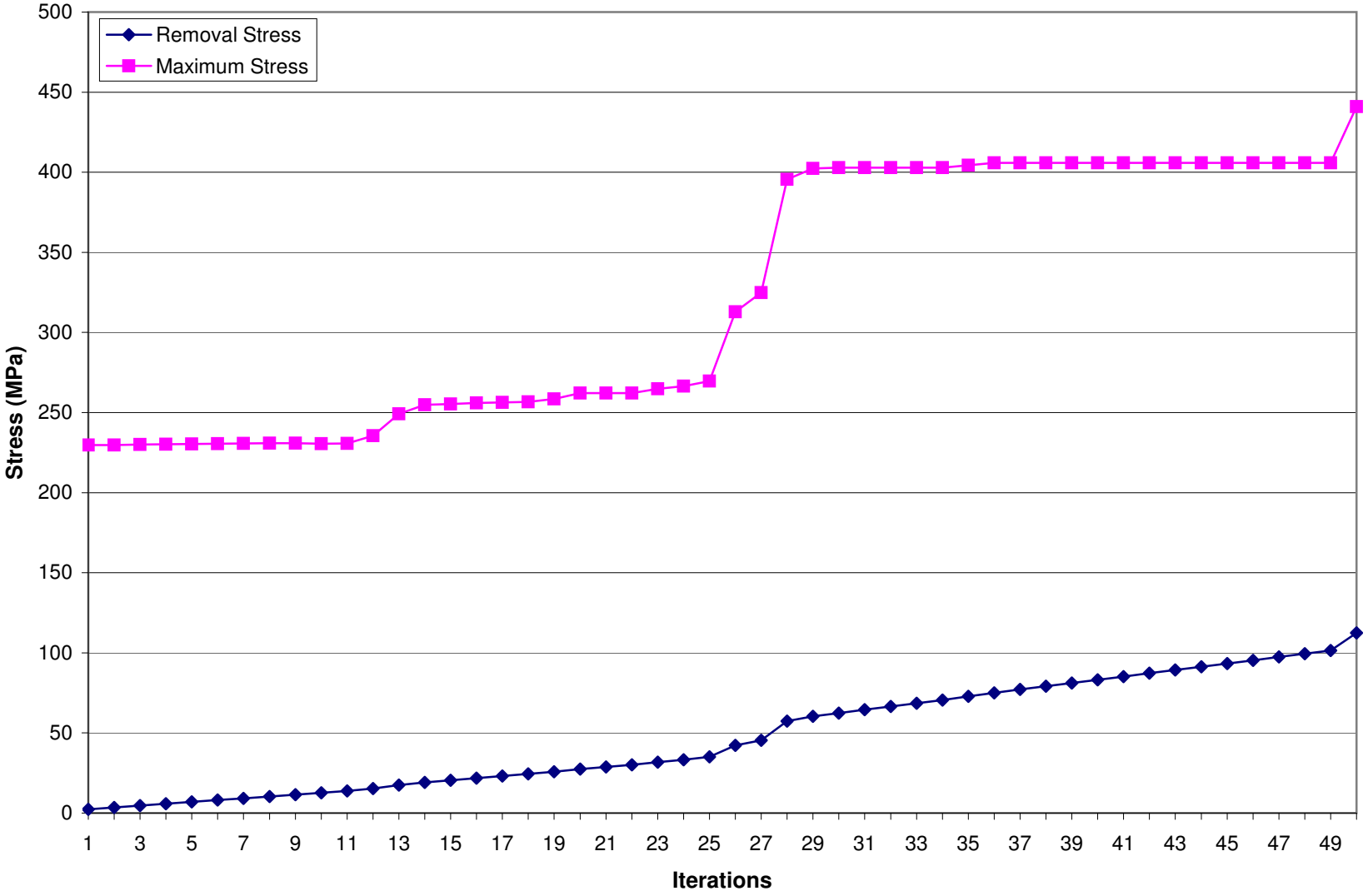
Graph A-1: 2D Model 1 - Area Remaining



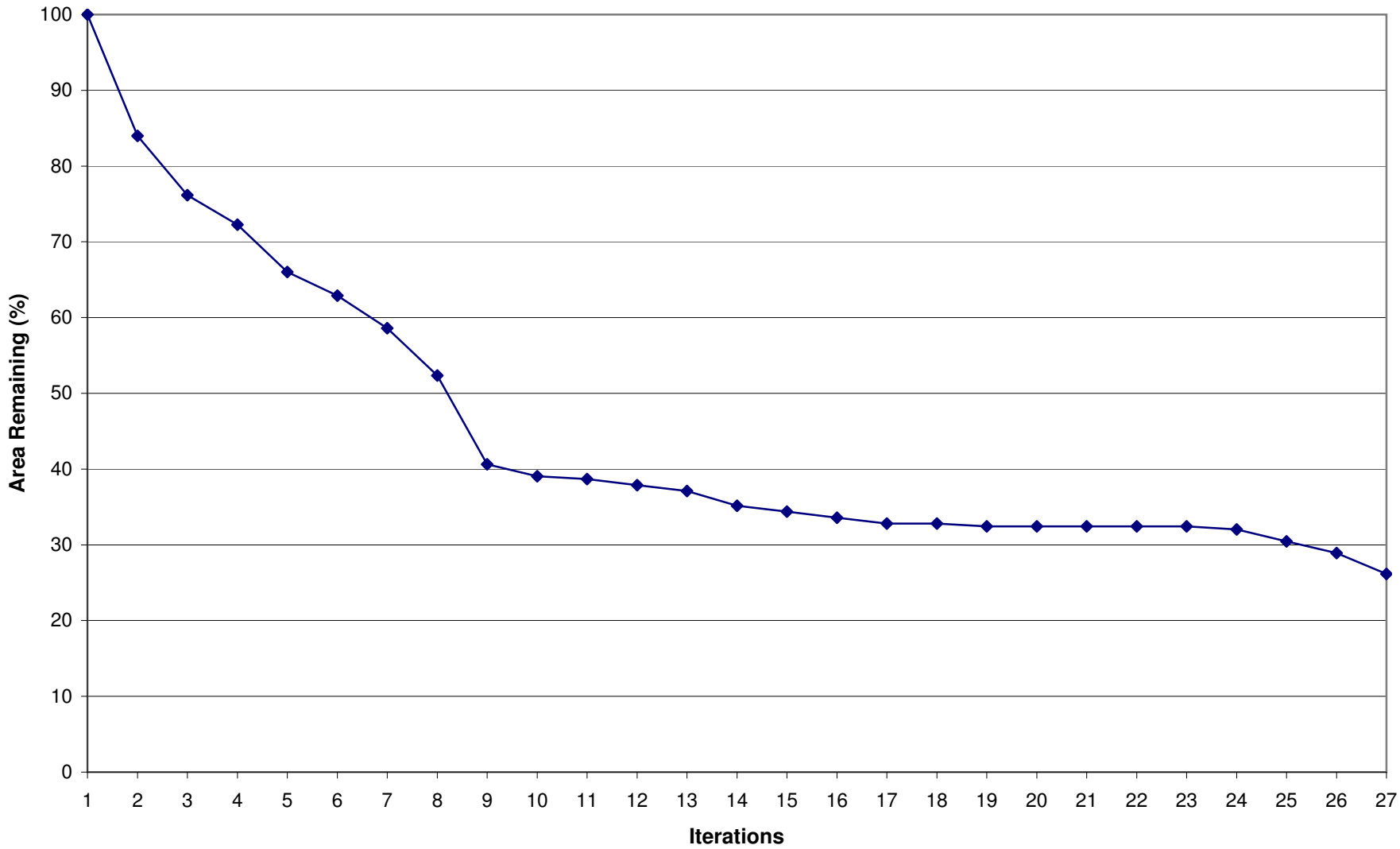
Graph A-2: 2D Model 1 - Stress Results



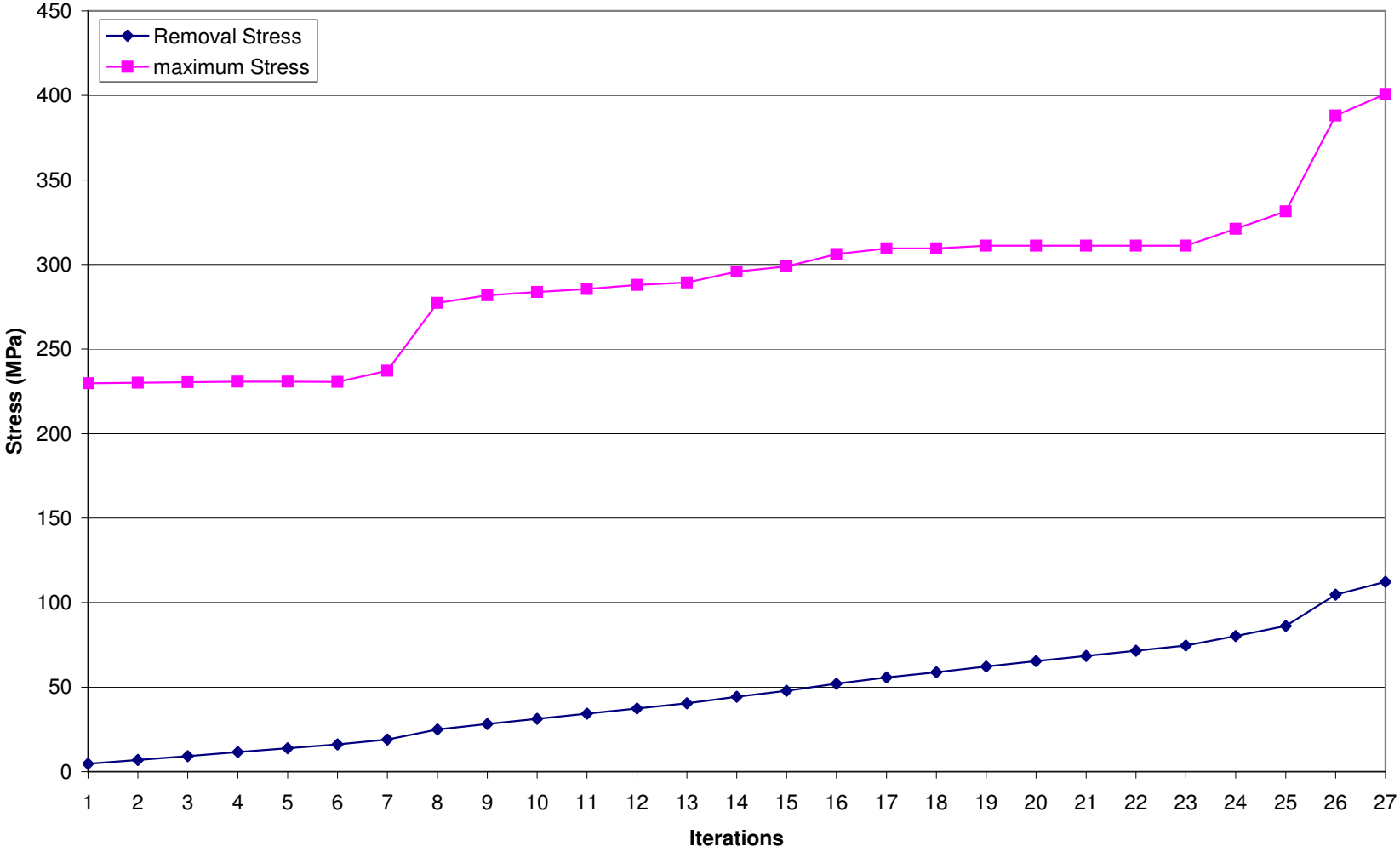
Graph A-3: 2D Model 2 - Area Remaining



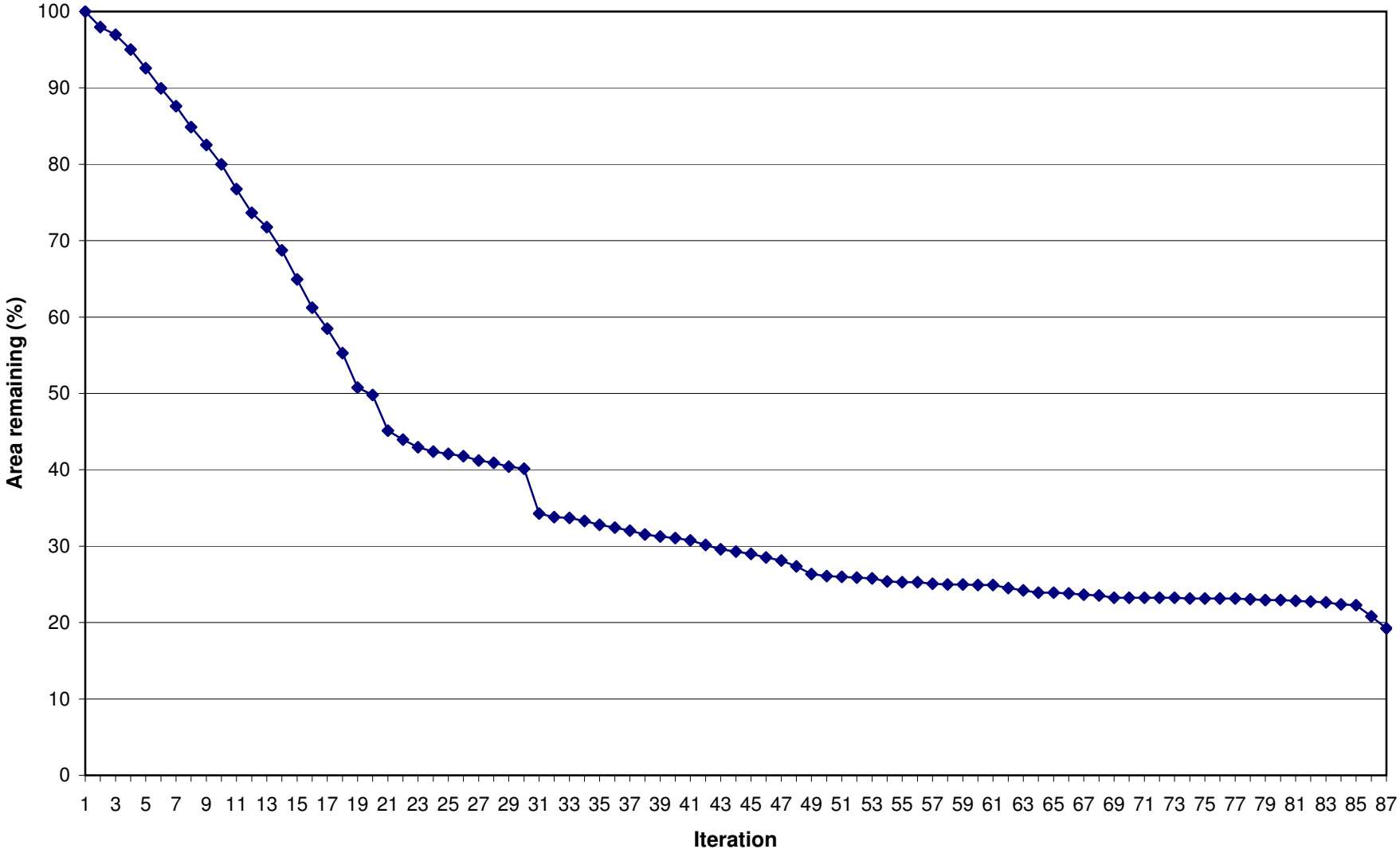
Graph A-4: 2D Model 2 - Stress Results



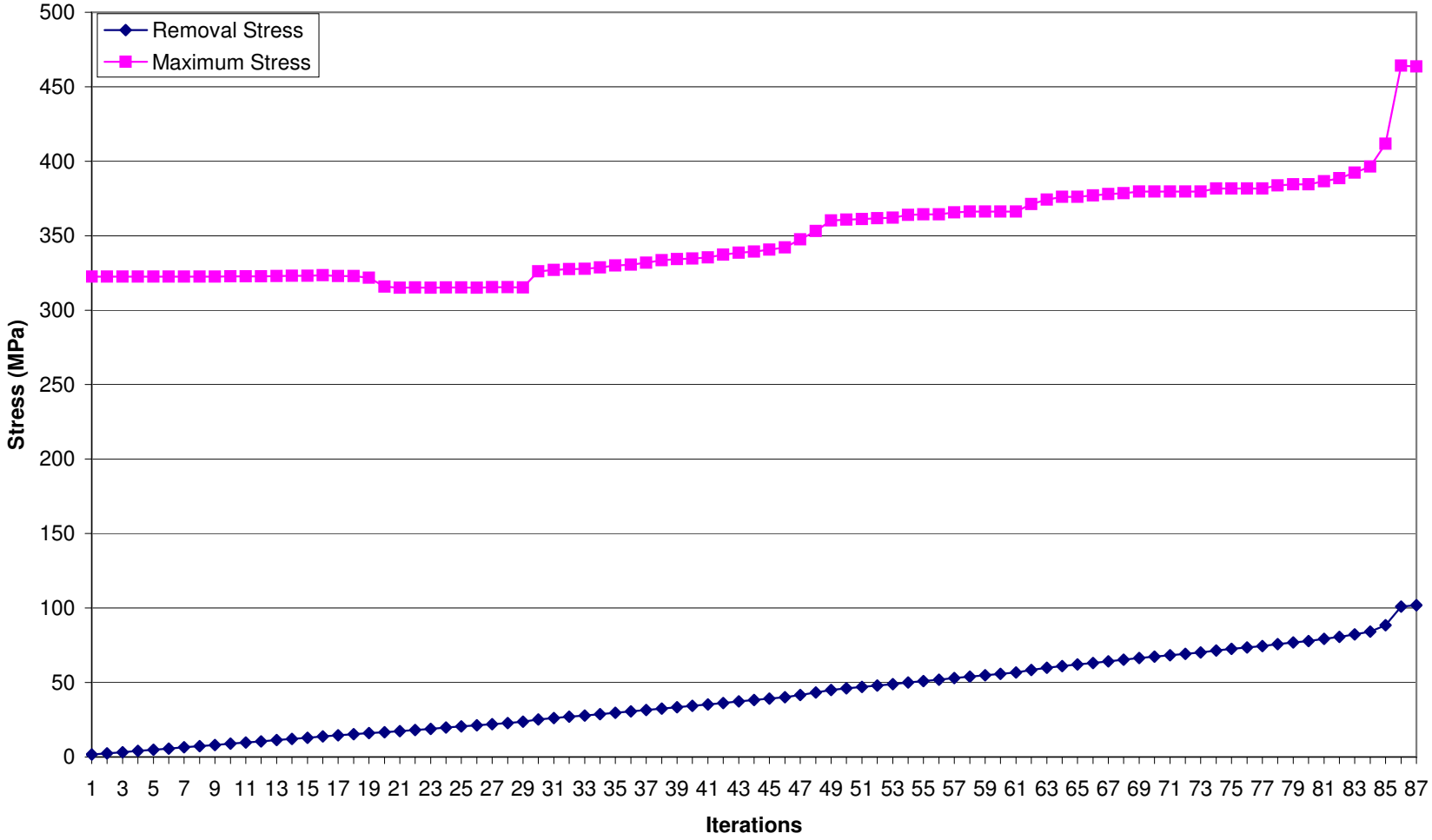
Graph A-5: 2D Model 3 - Area Remaining



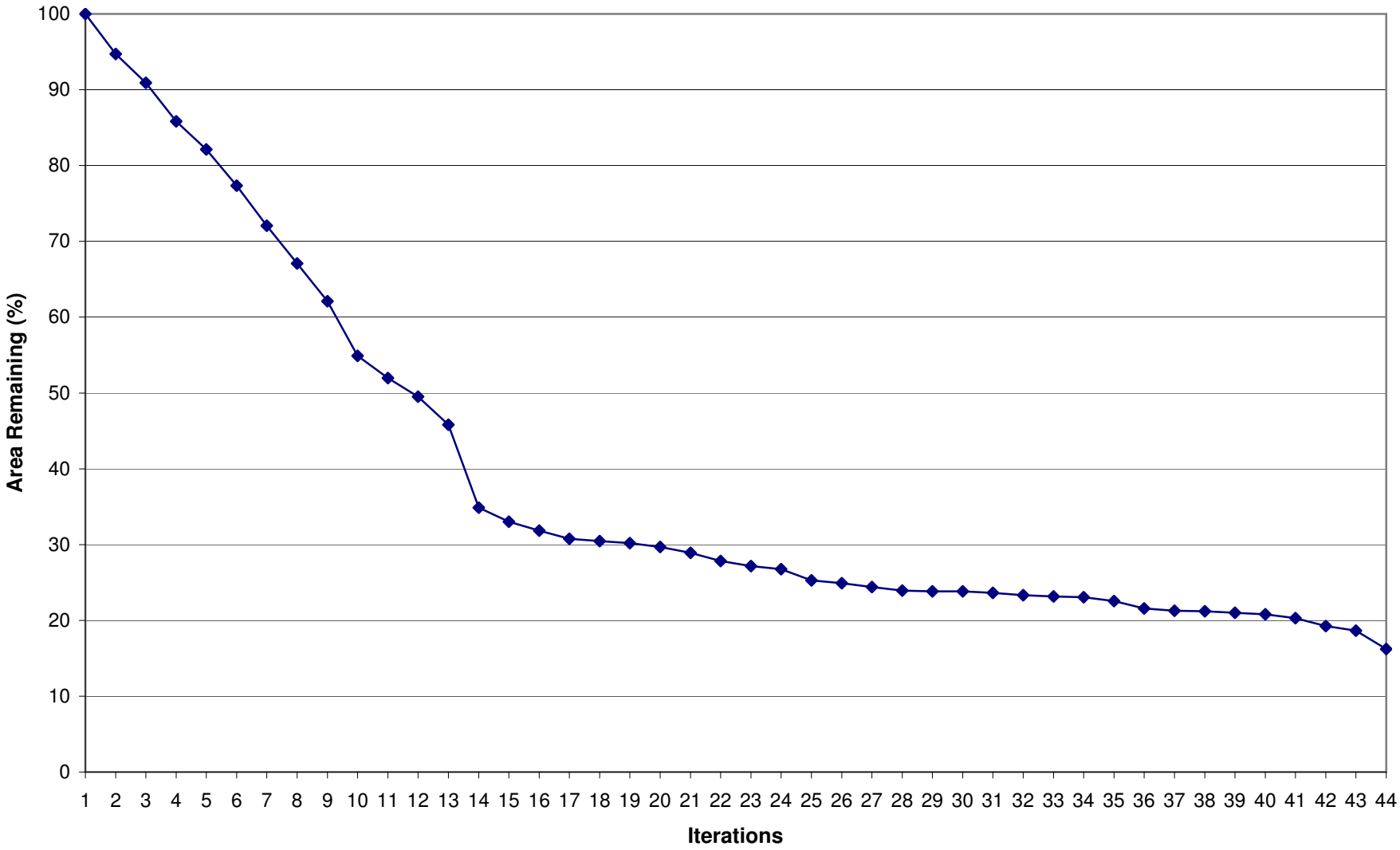
Graph A-6: 2D Model 3 - Stress Results



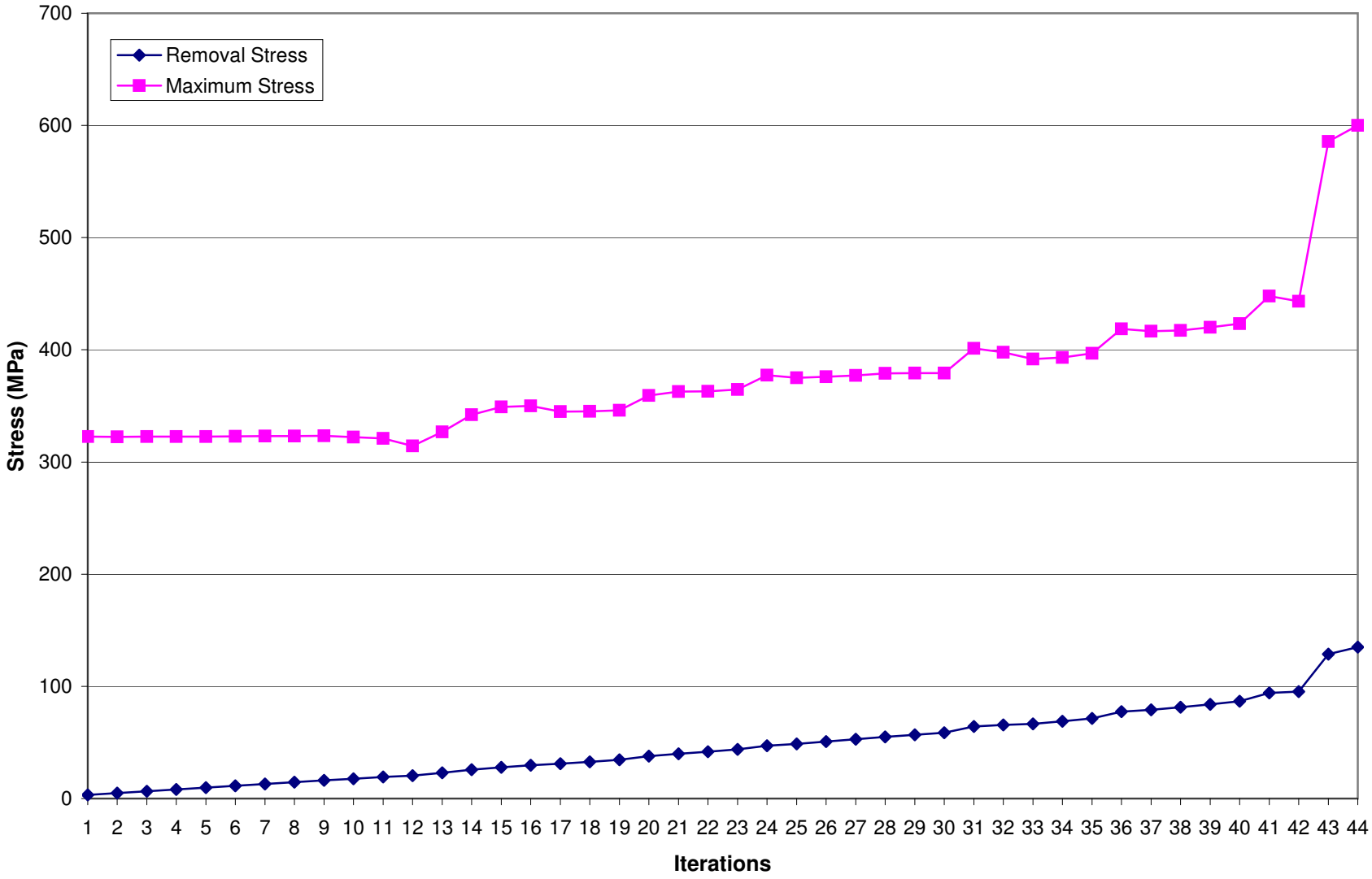
Graph A-7: 2D Model 4 - Area Remaining



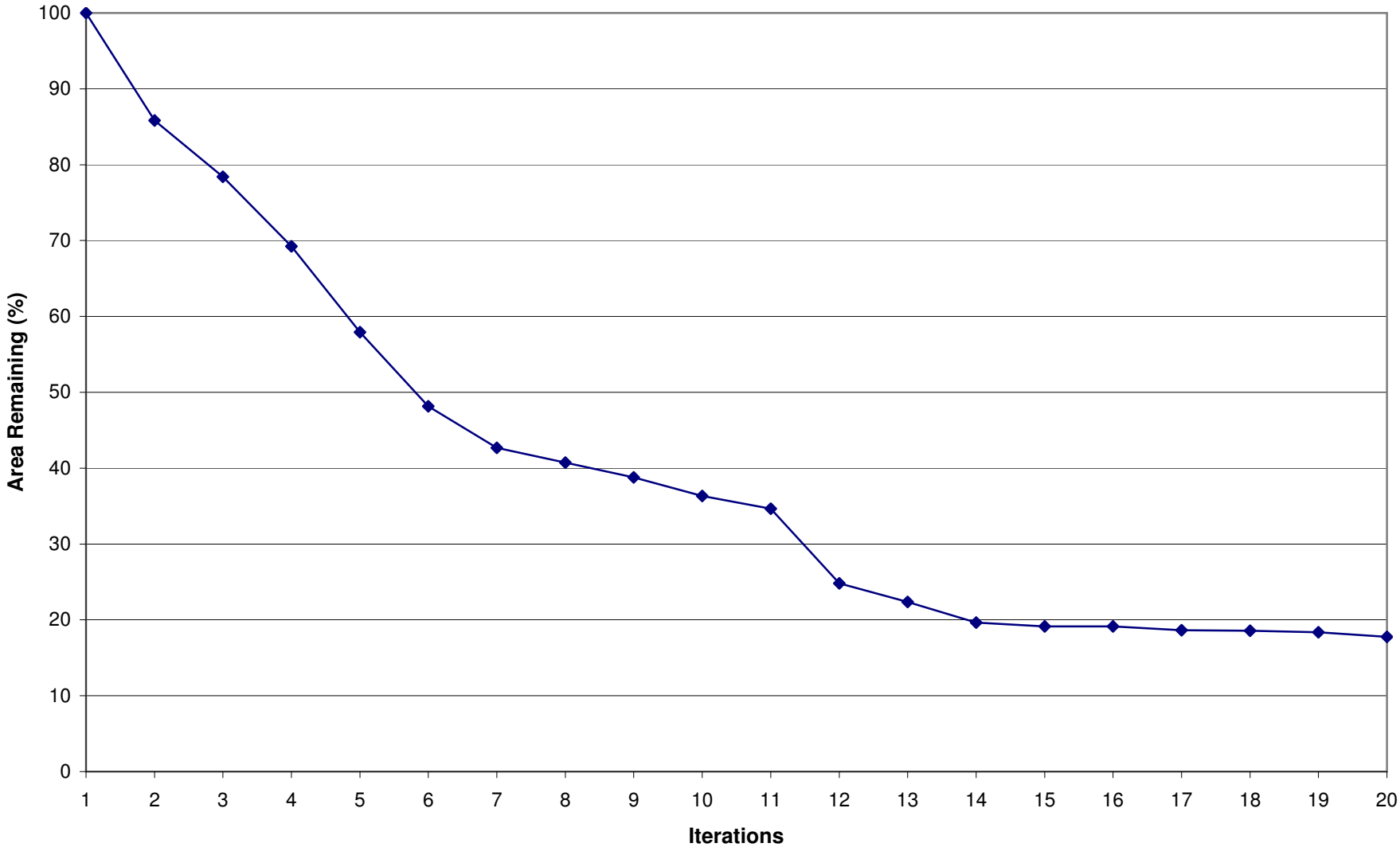
Graph A-8: 2D Model 4 - Stress Results



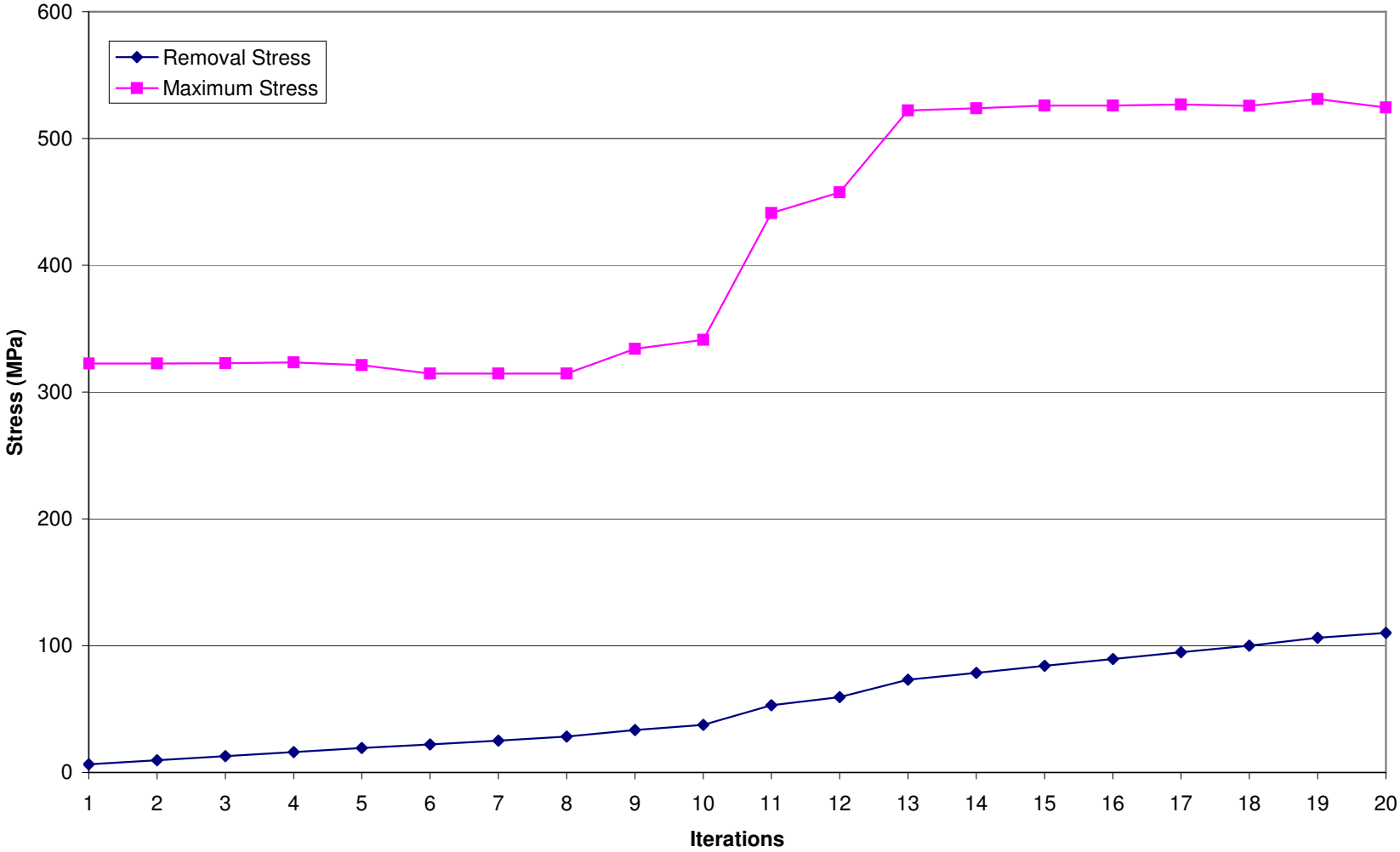
Graph A-9: 2D Model 5 - Area Remaining



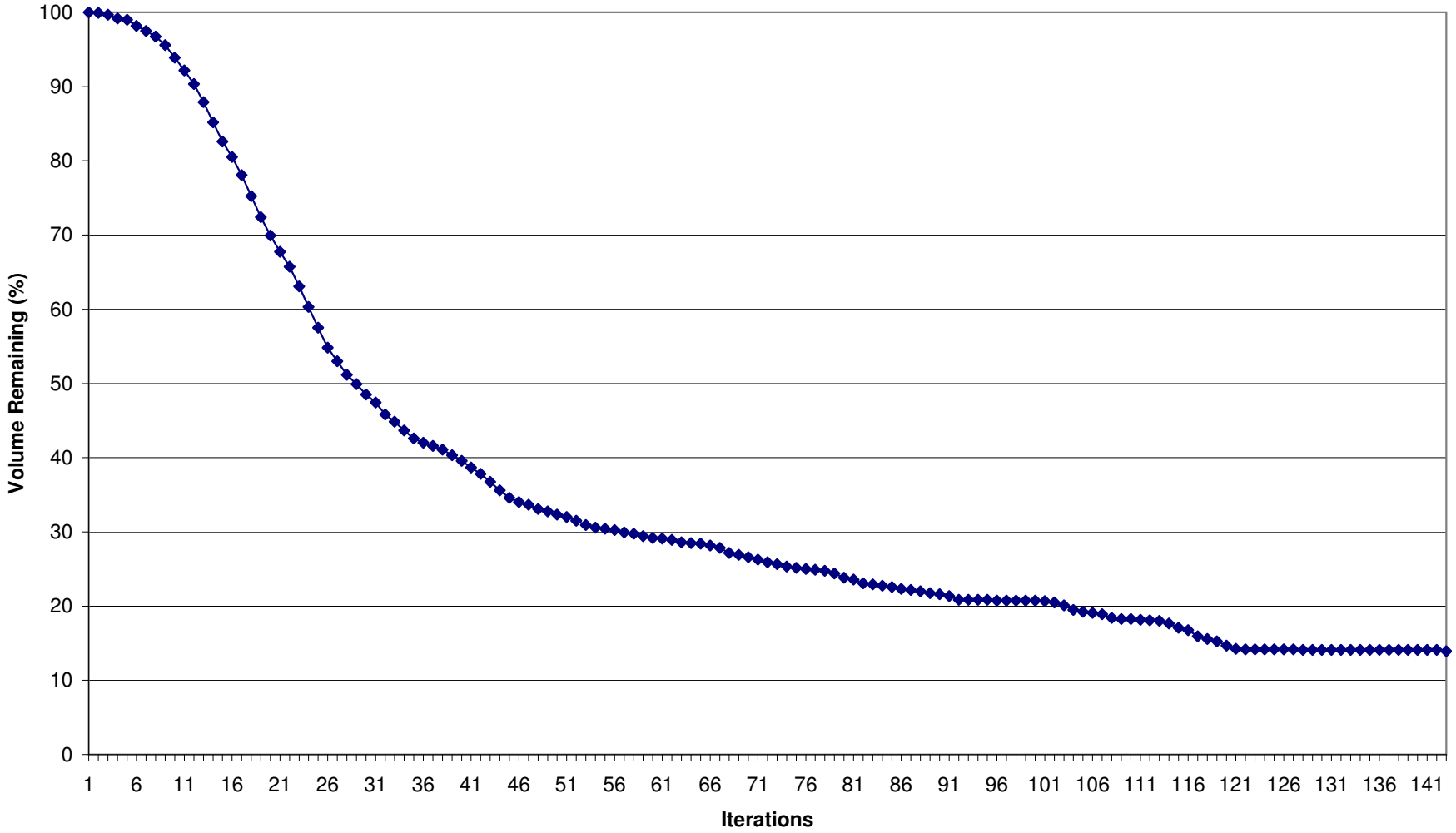
Graph A-10: 2D Model 5 - Stress Results



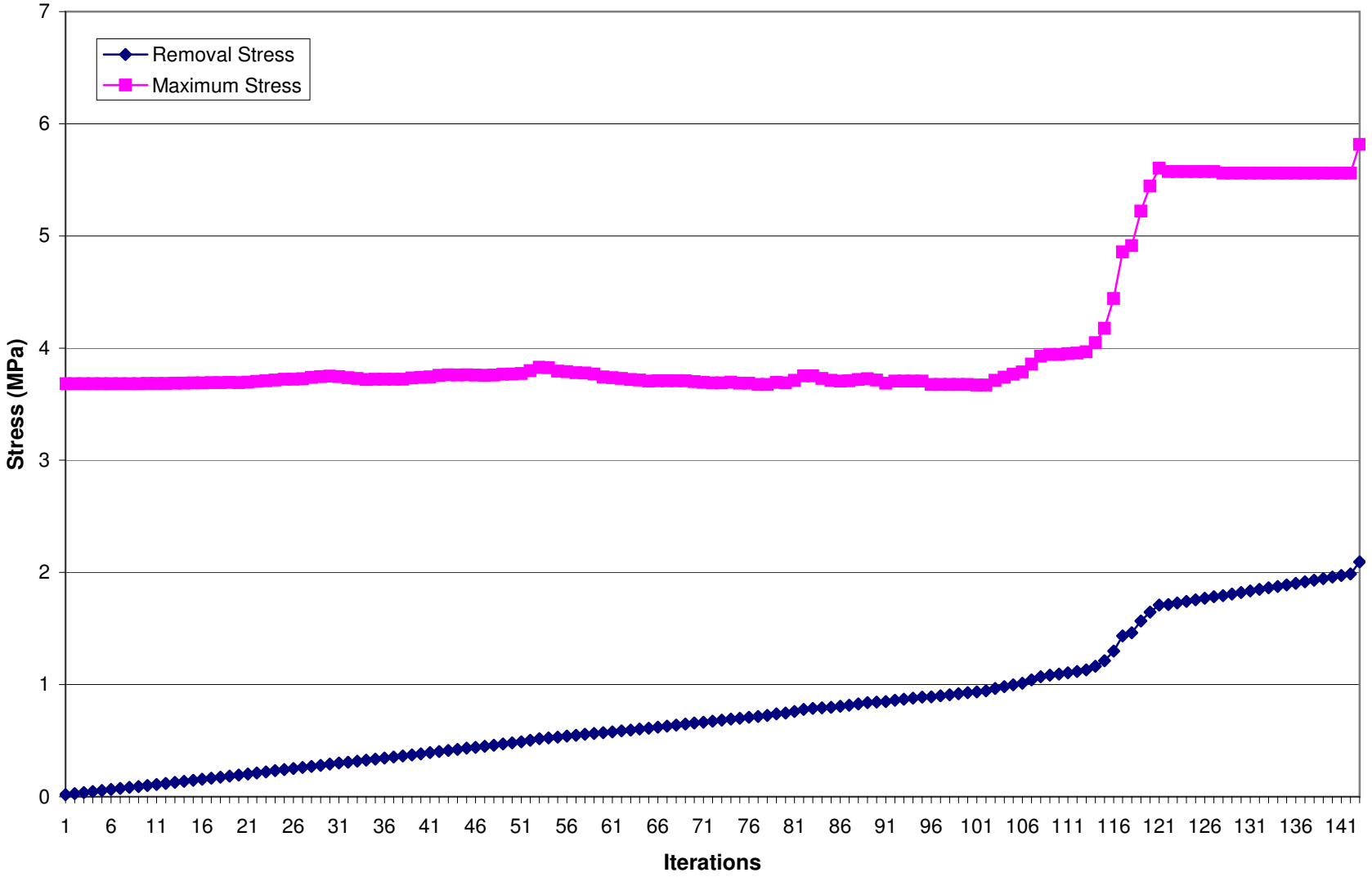
Graph A-11: 2D Model 6 - Area Remaining



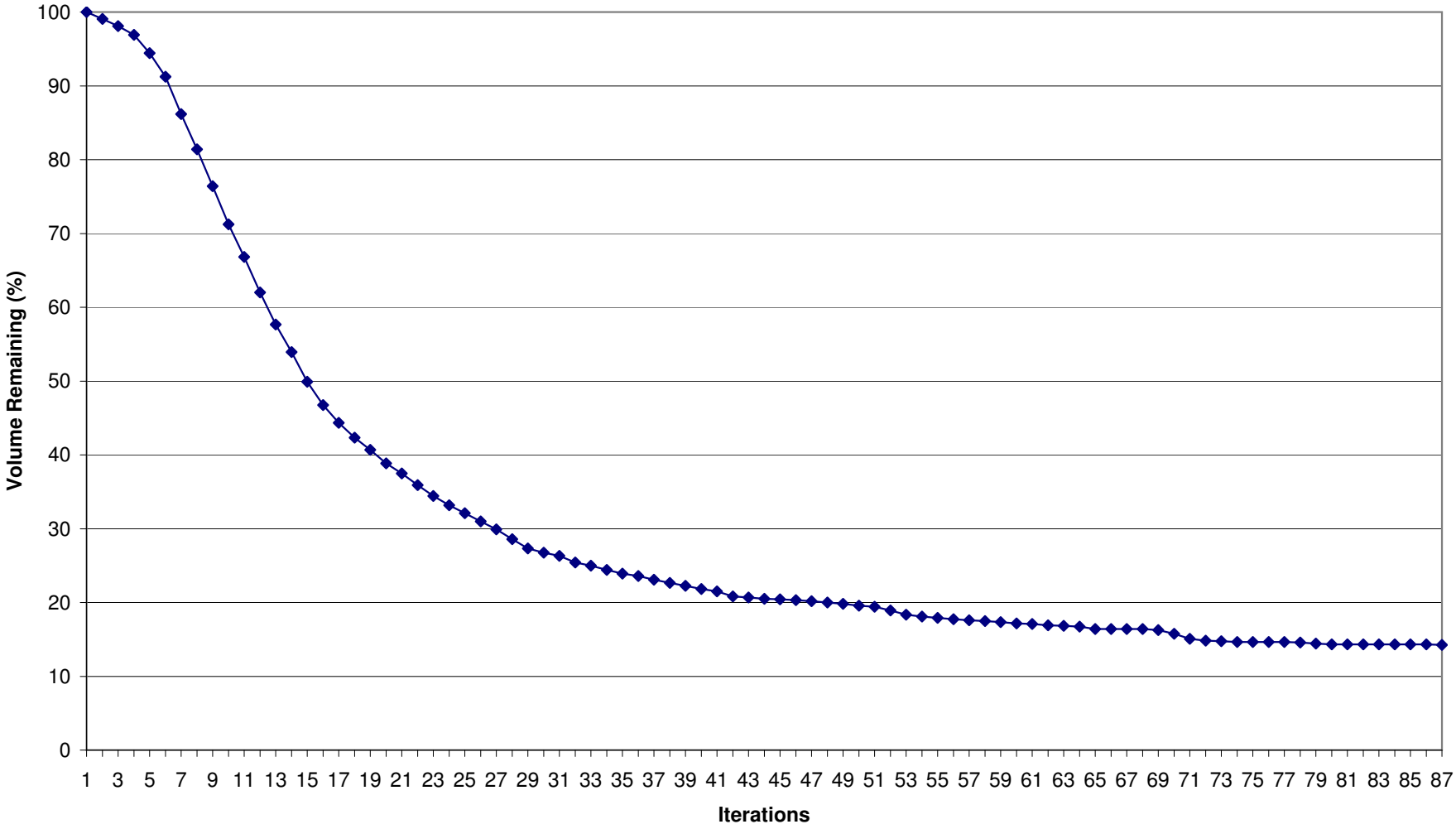
Graph A-12: 2D Model 6 - Stress Results



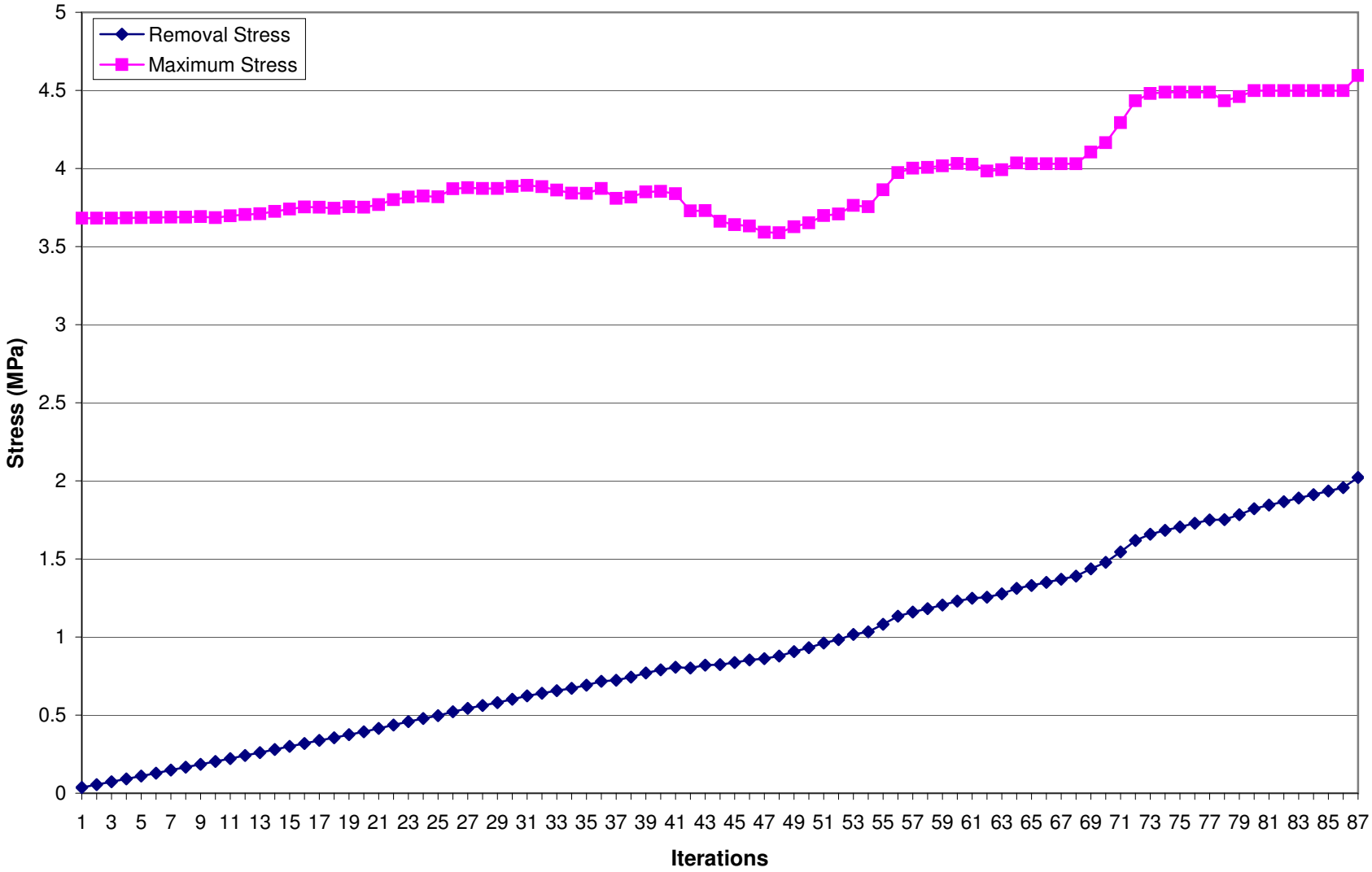
Graph A-13: L-Block Model 1 - Volume Remaining



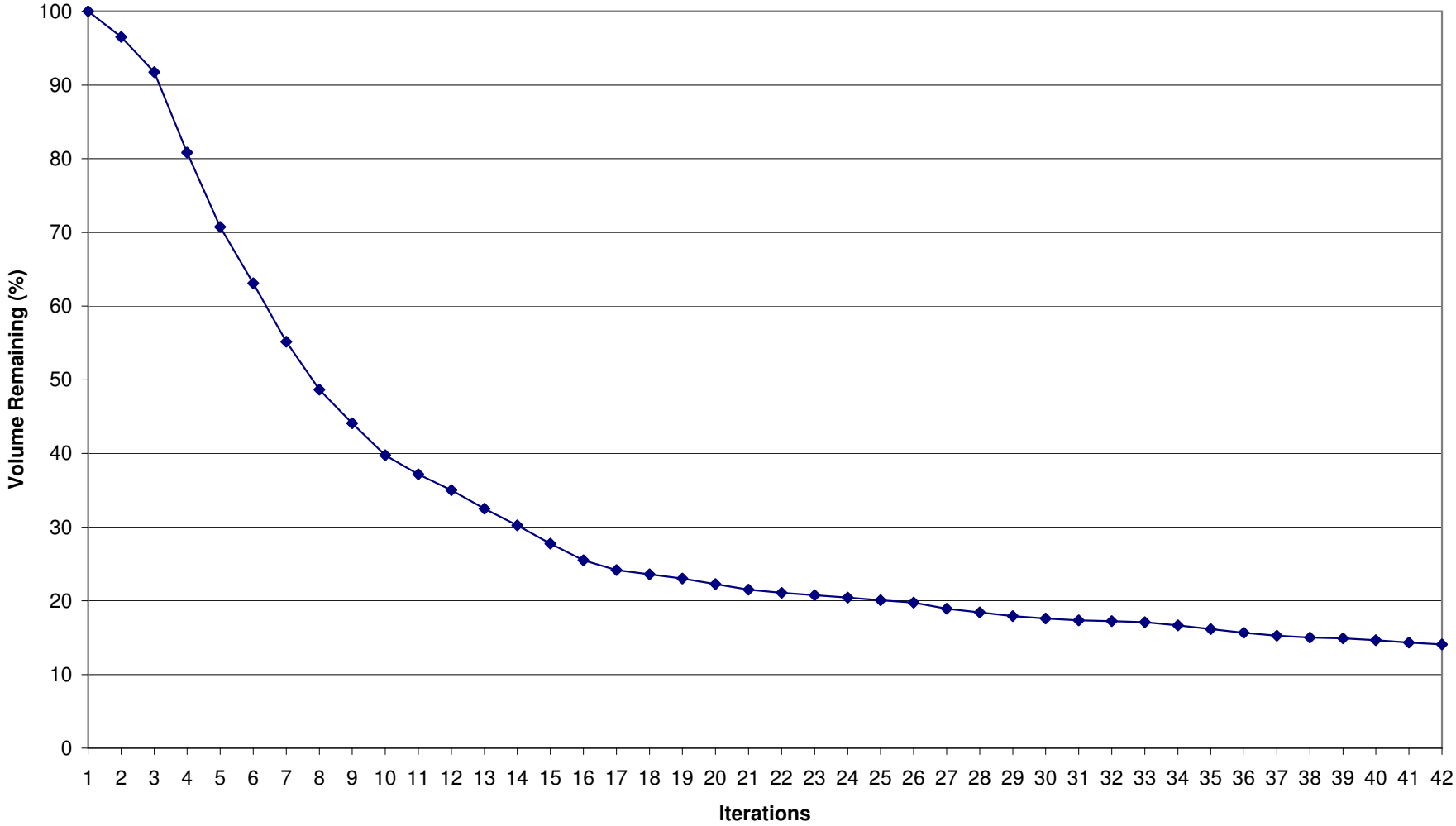
Graph A-14: L-Block Model 1 - Stress Results



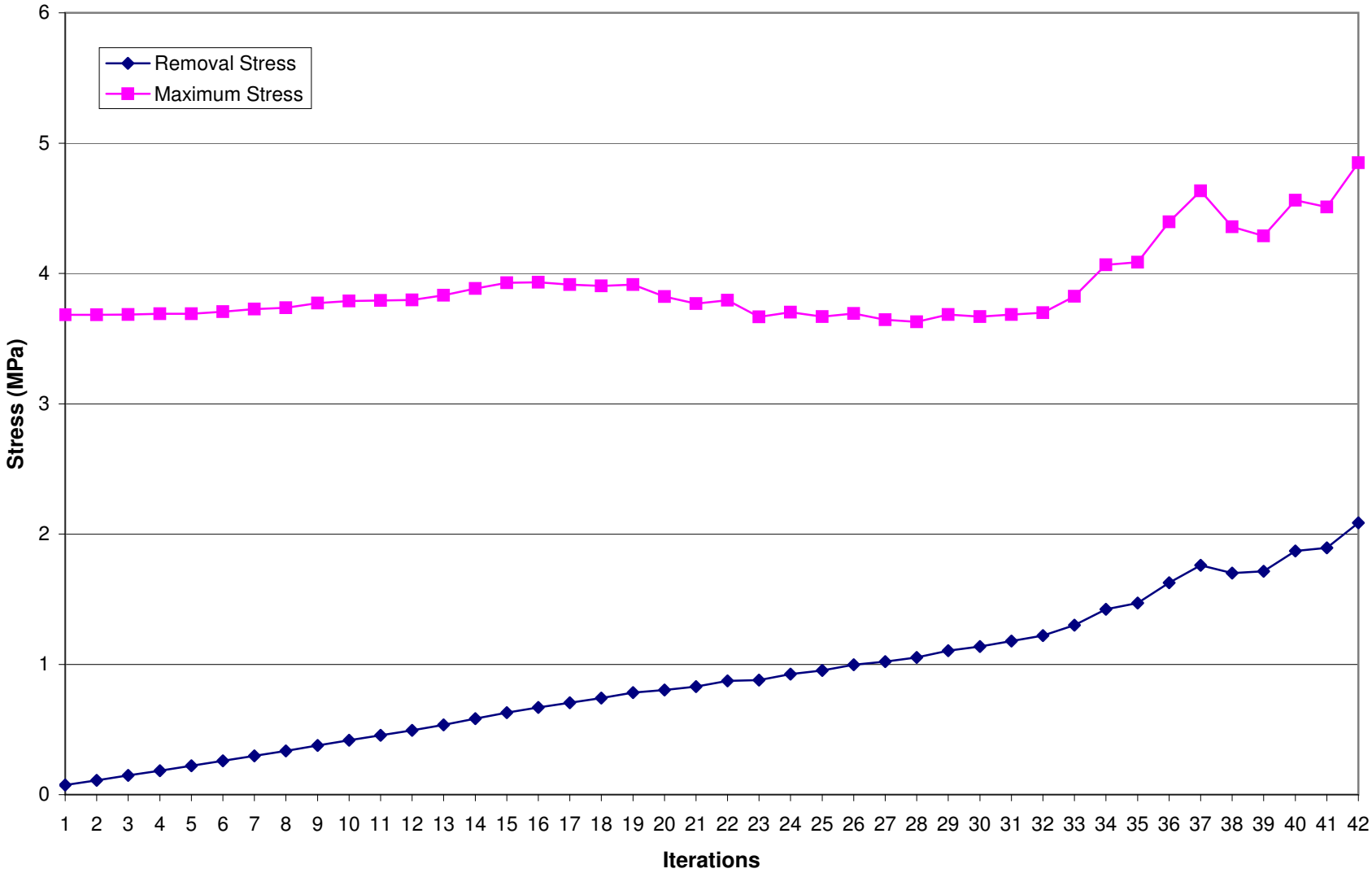
Graph A-15: L-Block Model 2 - Volume Remaining



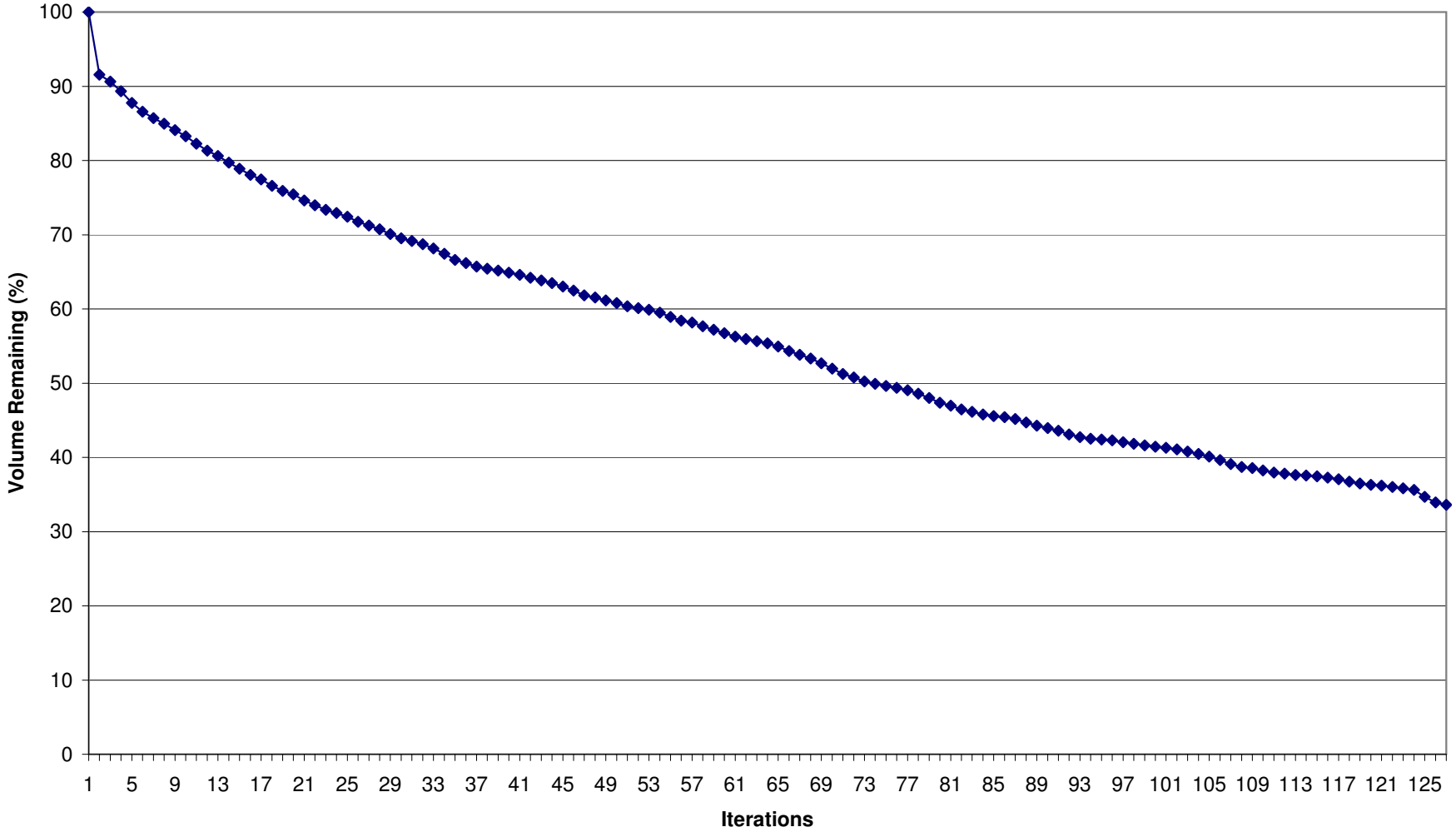
Graph A-16: L-Block Model 2 - Stress Results



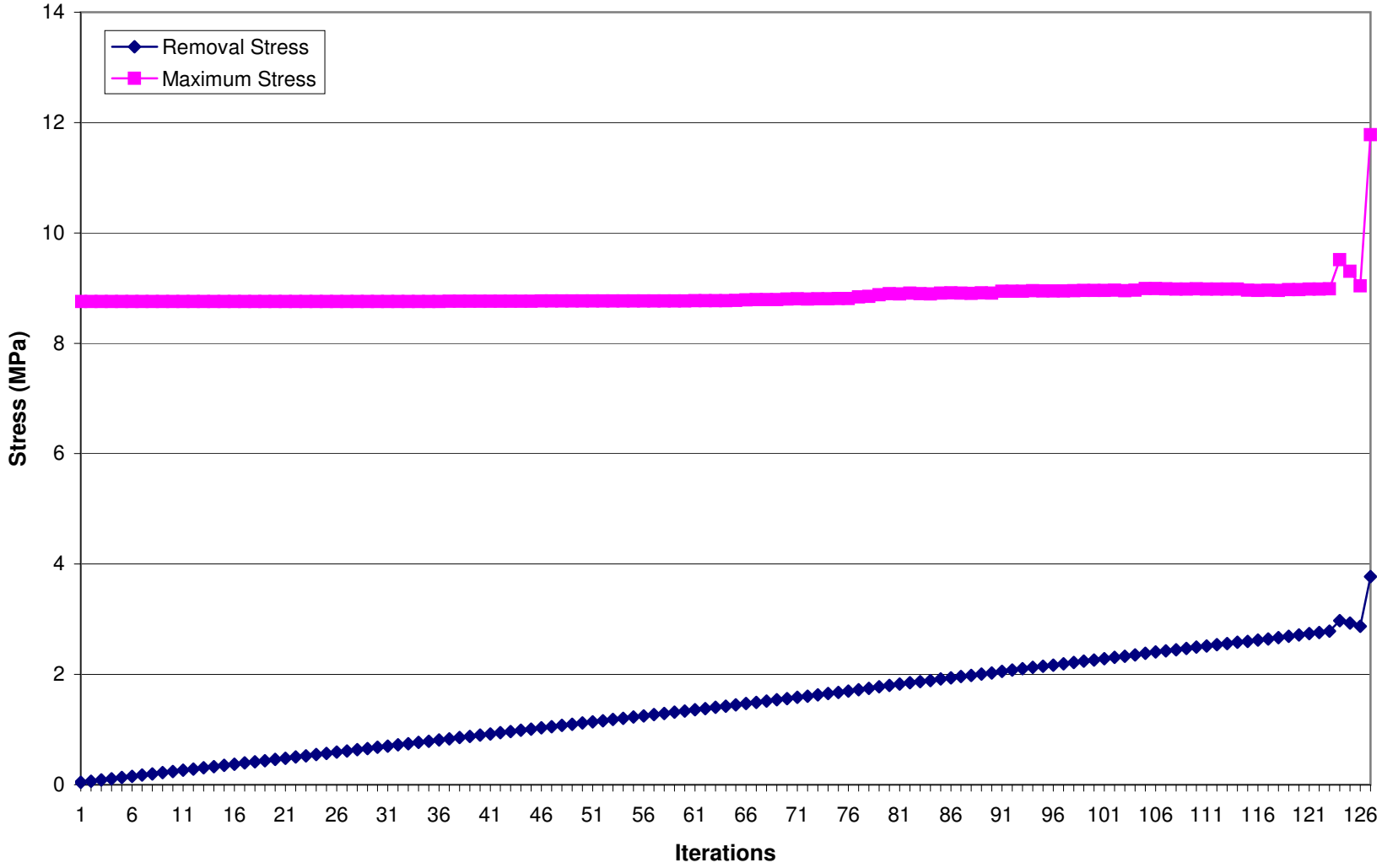
Graph A-17: L-Block Model 3 - Volume Remaining



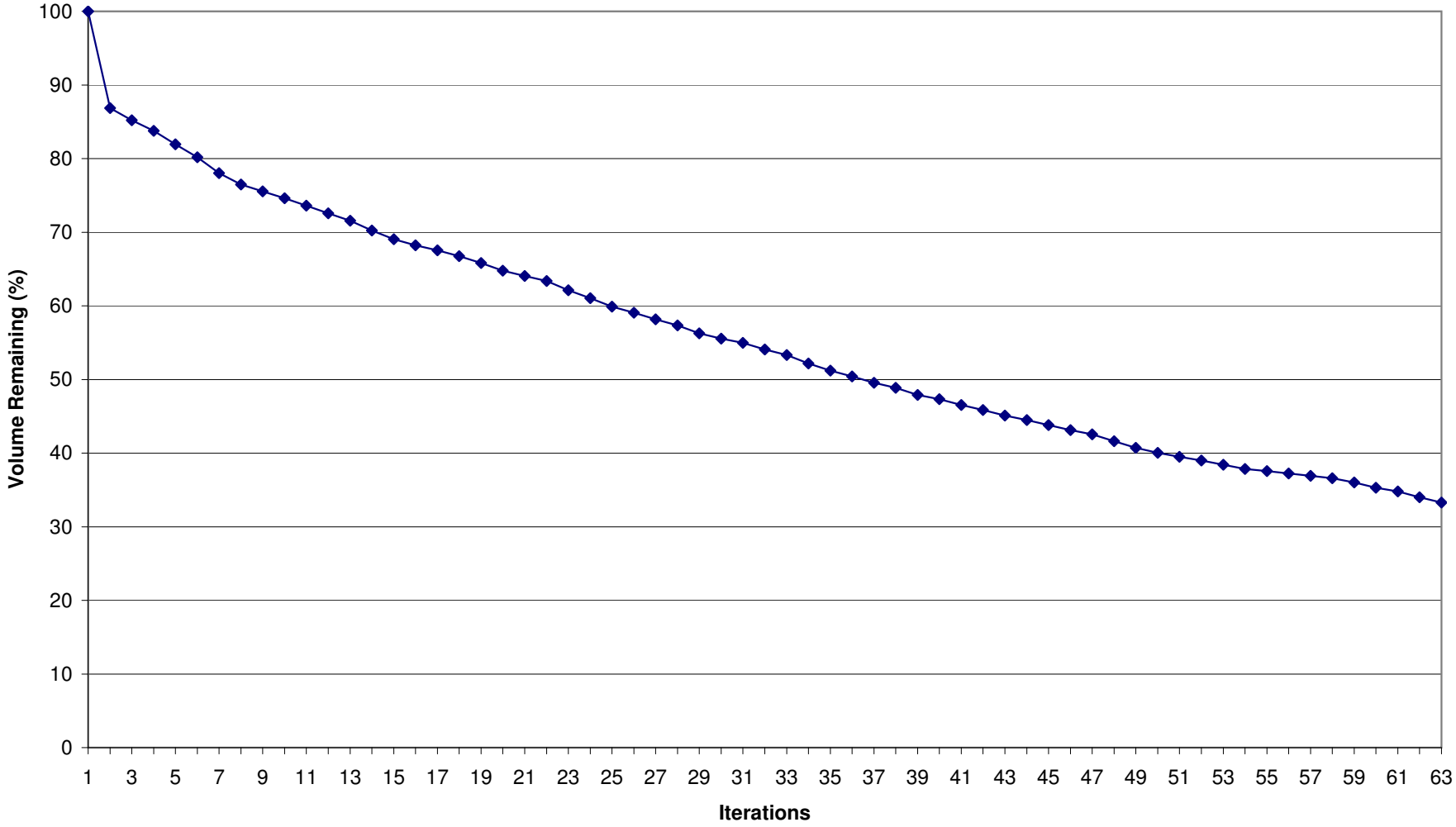
Graph A-18: L-Block Model 3 - Stress Results



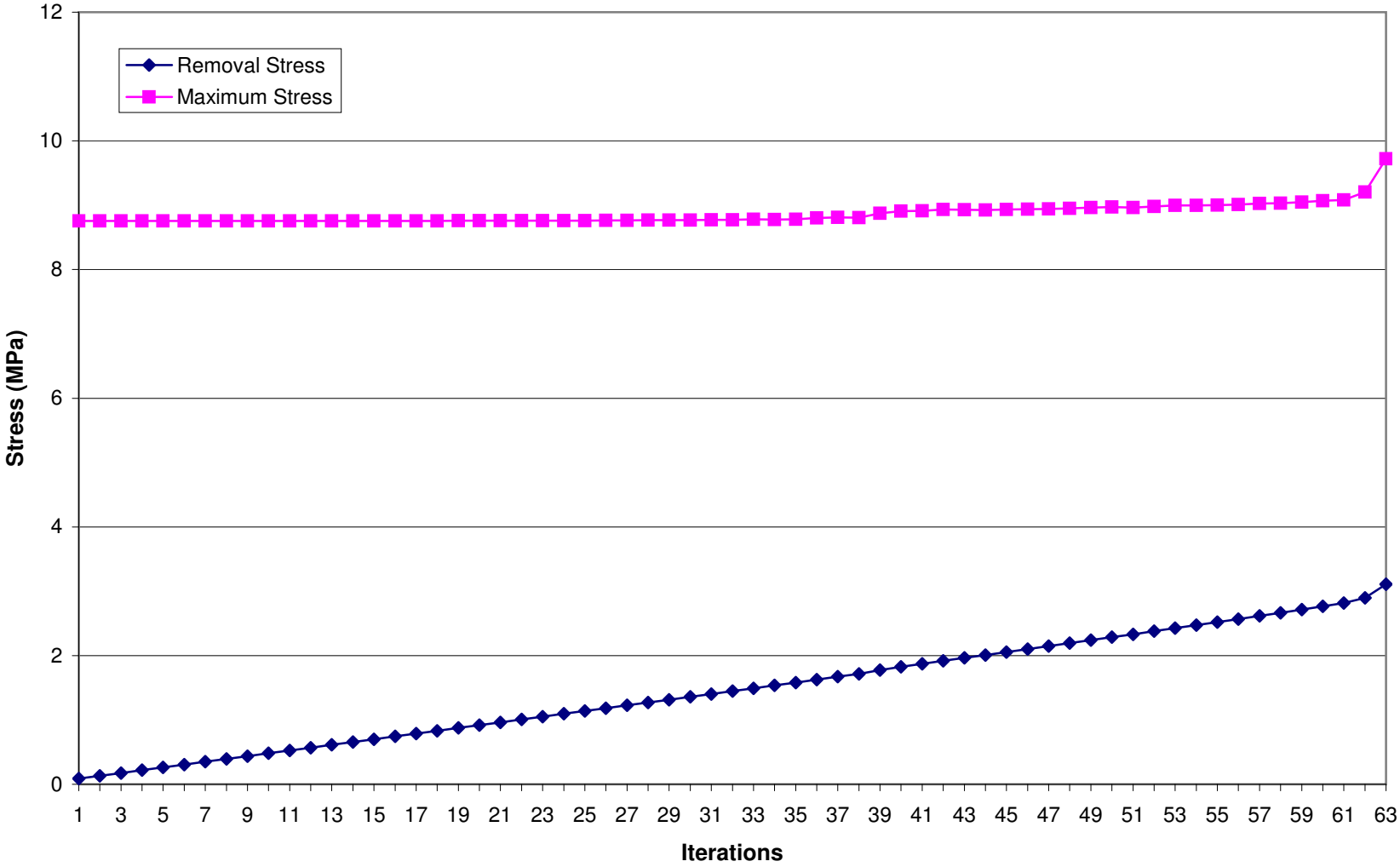
Graph A-19: Sandwich Hexahedral Model 1 - Volume Remaining



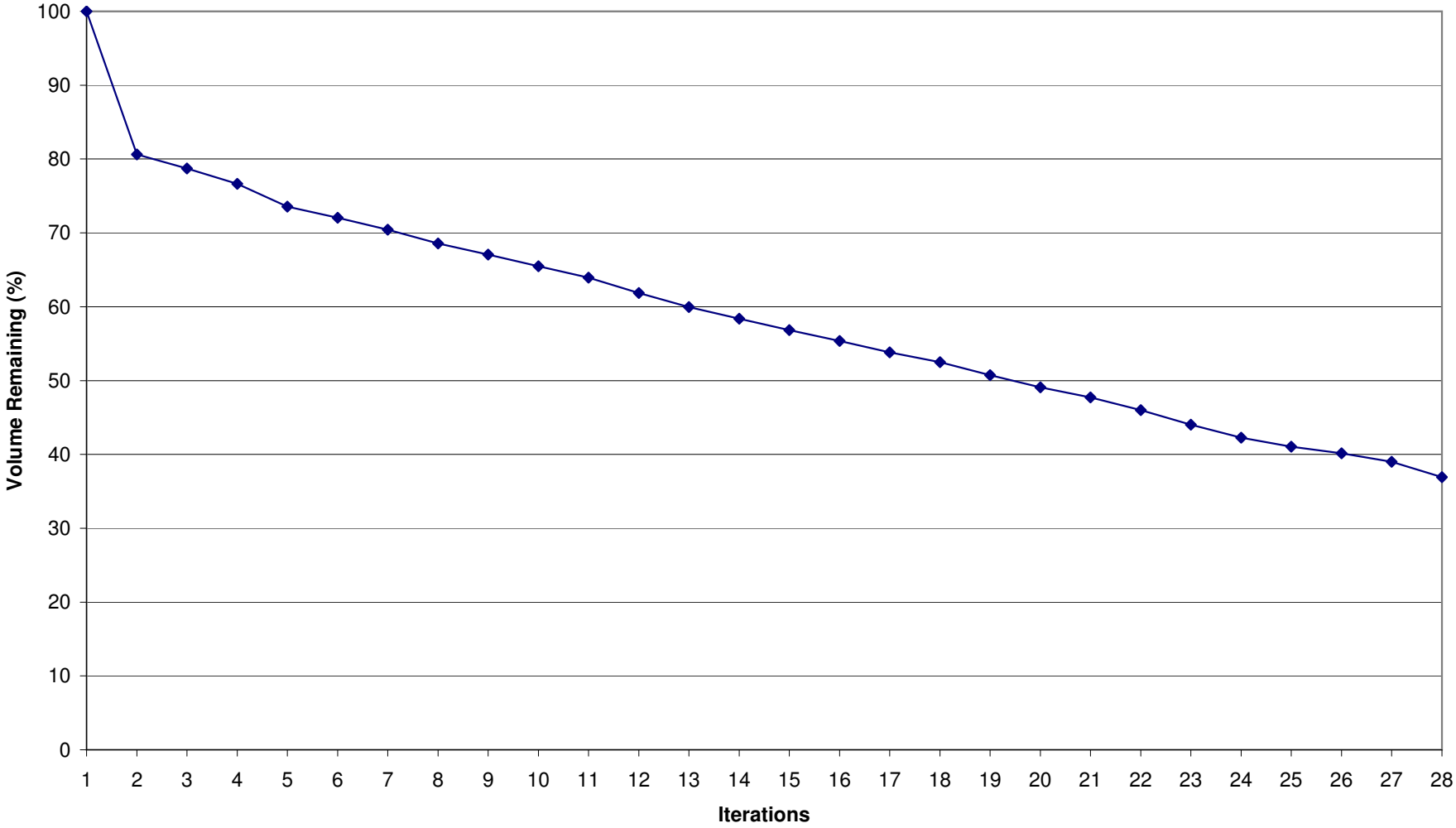
Graph A-20: Sandwich Hexahedral Model 1 - Stress Results



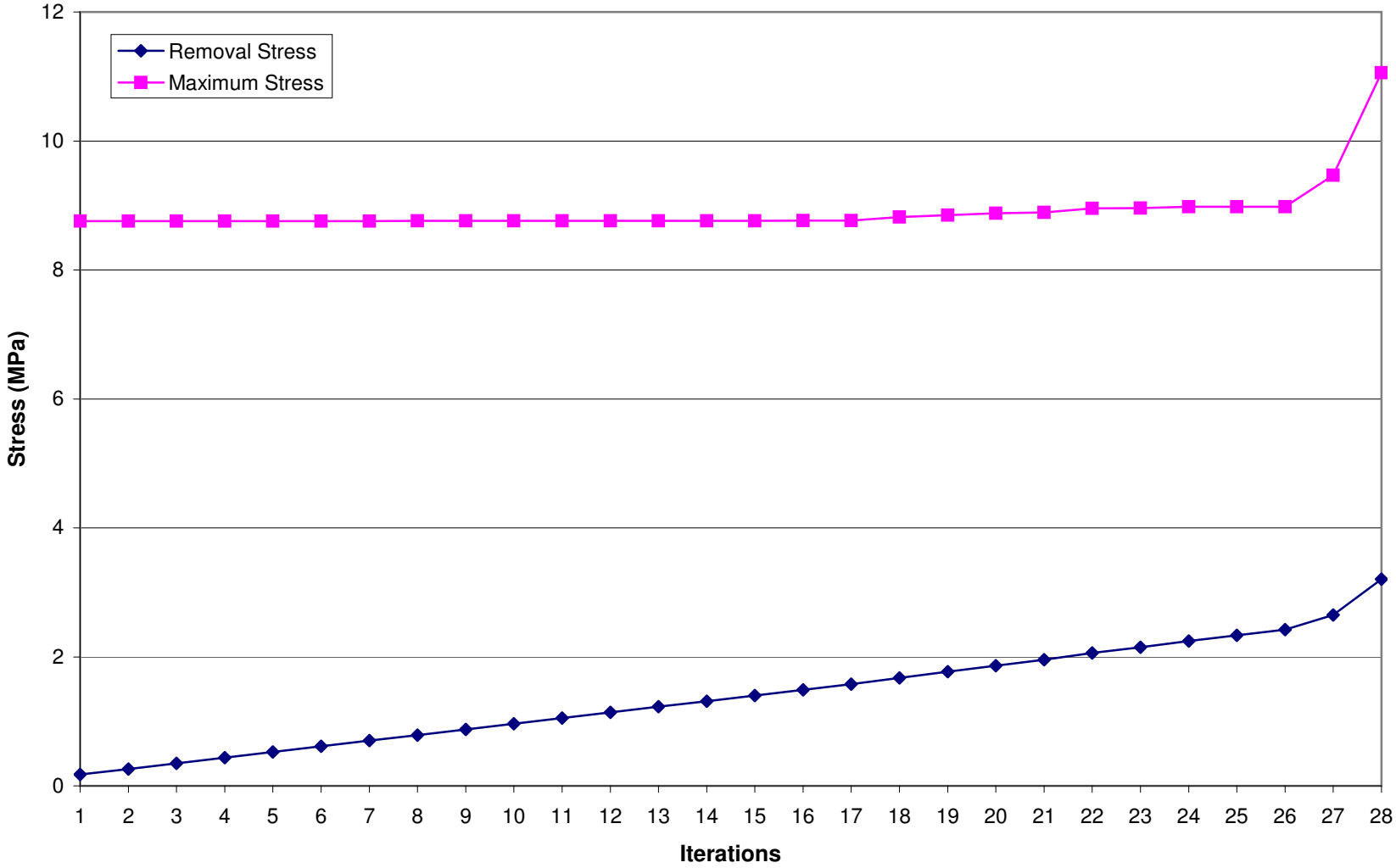
Graph A-21: Sandwich Hexahedral Model 2 - Volume Remaining



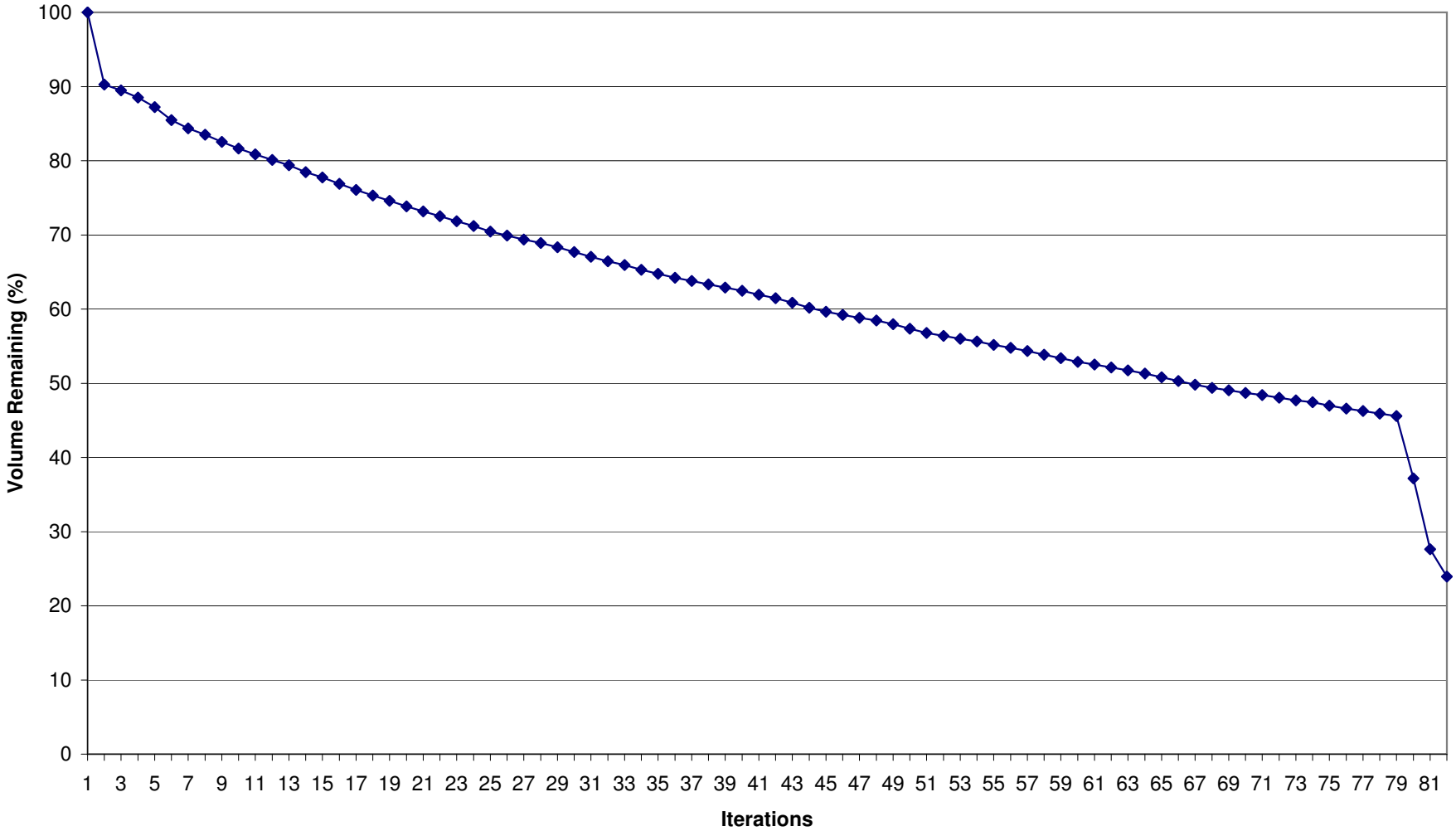
Graph A-22: Sandwich Hexahedral Model 2 - Stress Results



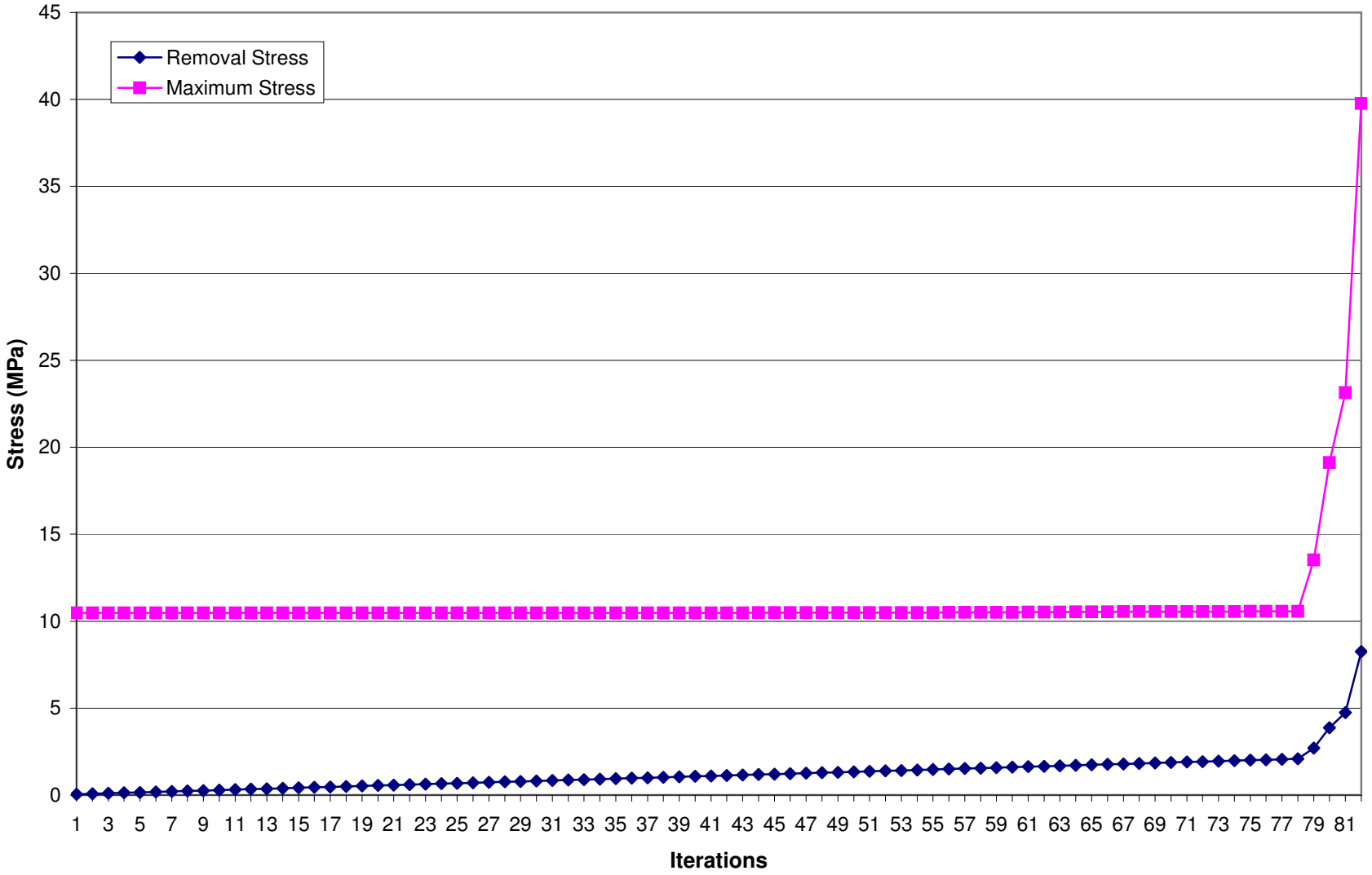
Graph A-23: Sandwich Hexahedral Model 3 - Volume Remaining



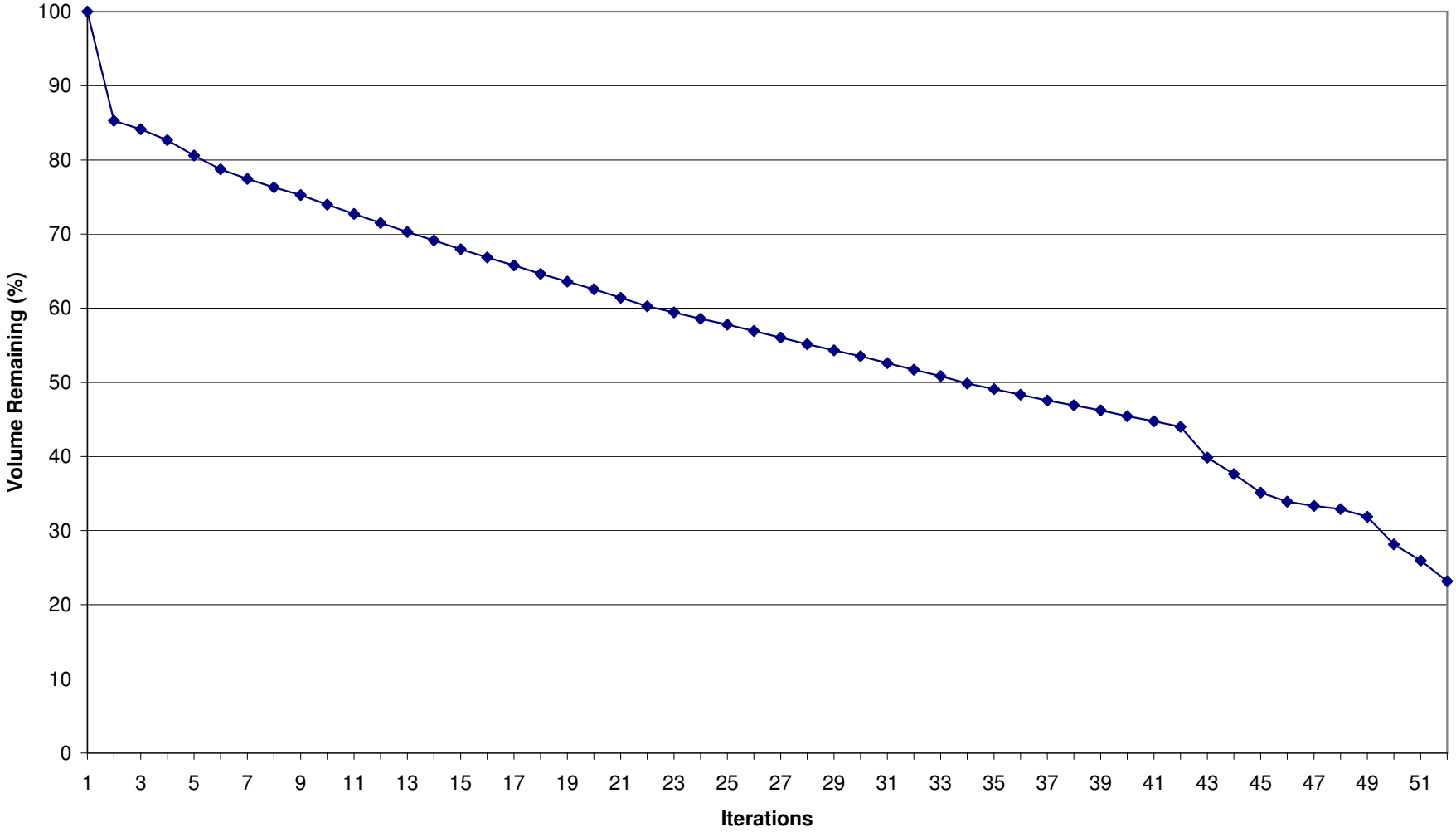
Graph A-24: Sandwich Hexahedral Model 3 - Stress Results



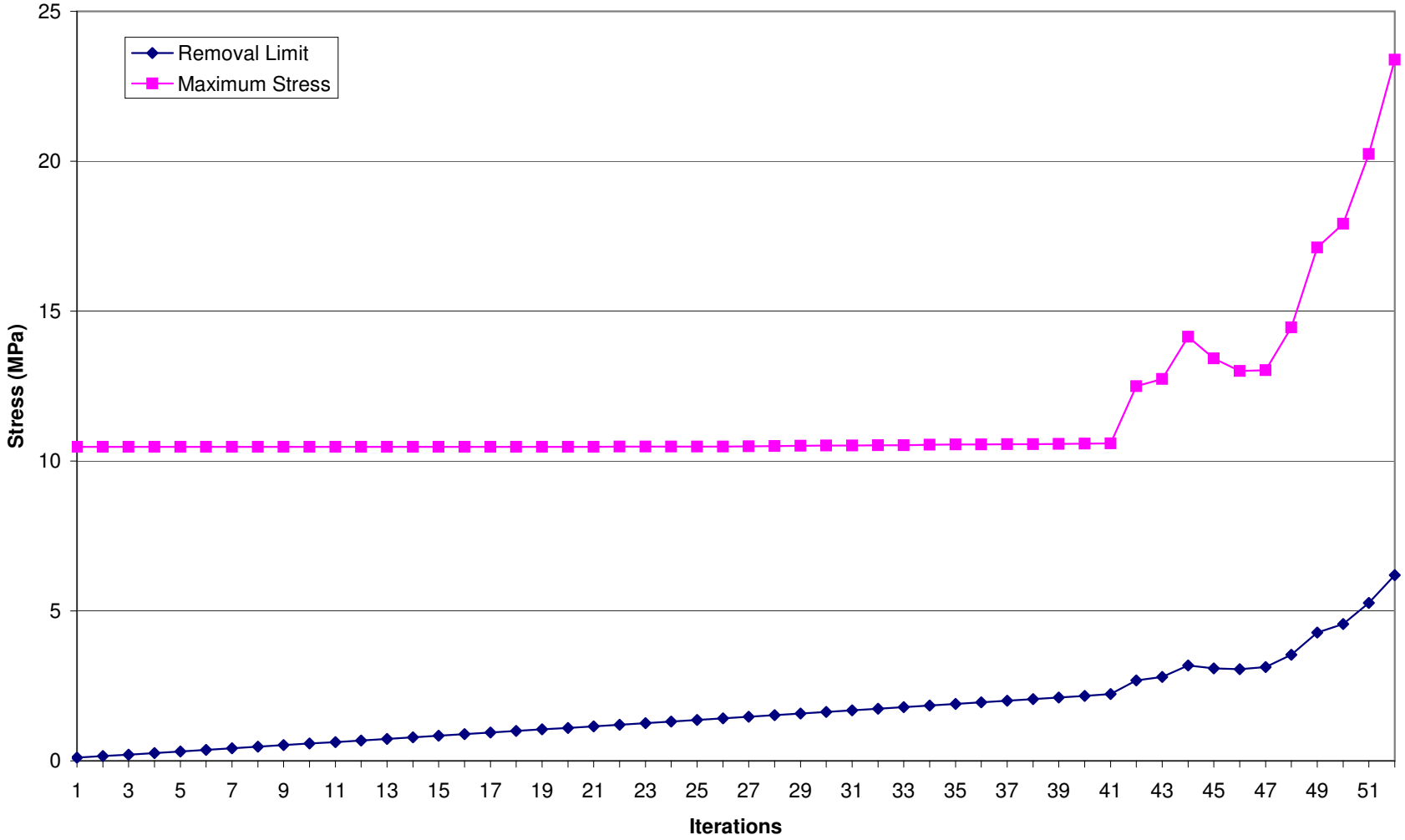
Graph A-25: Sandwich Tetrahedral Model 4 - Volume Remaining



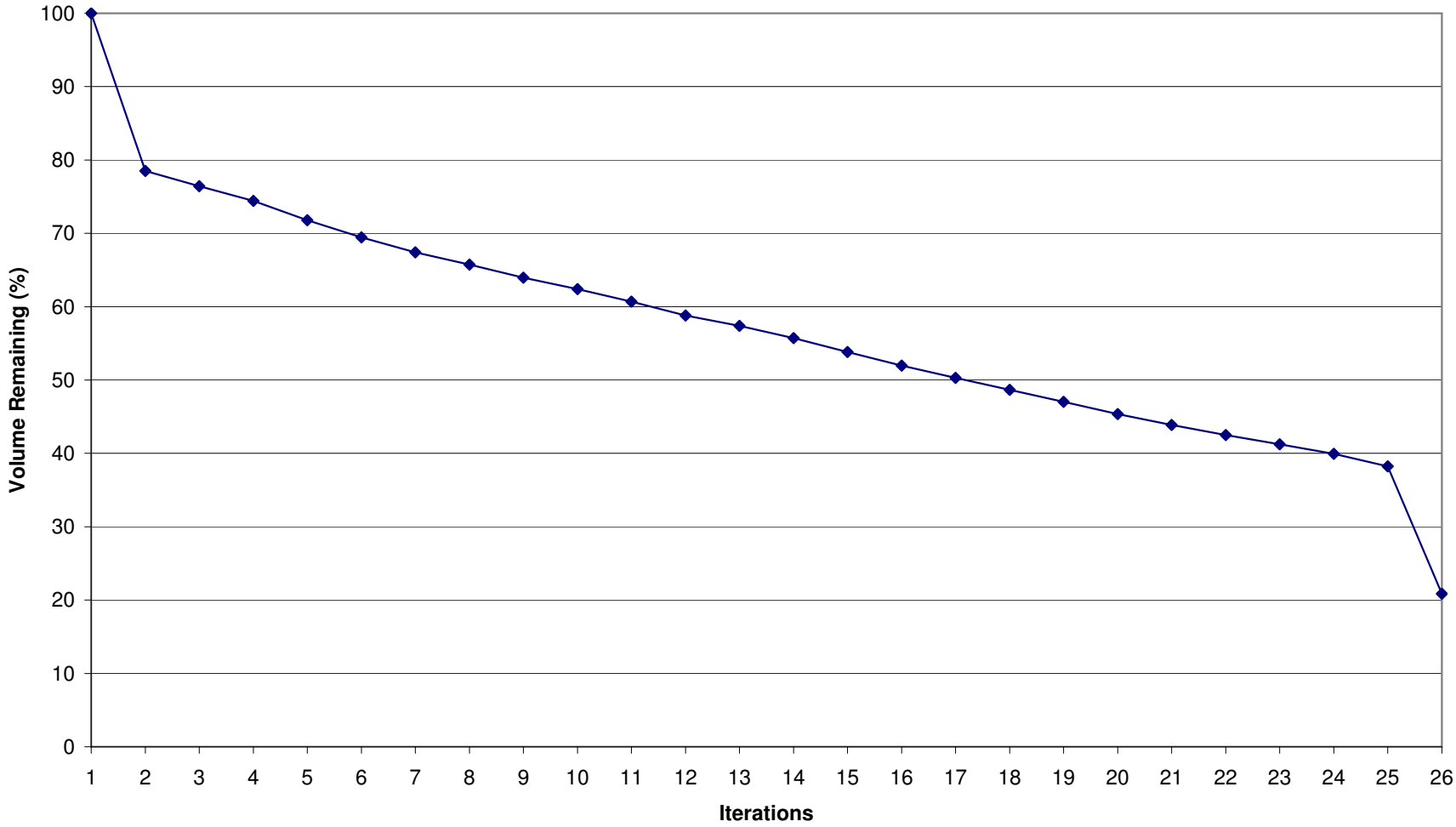
Graph A-26: Sandwich Tetrahedral Model 4 - Stress Results



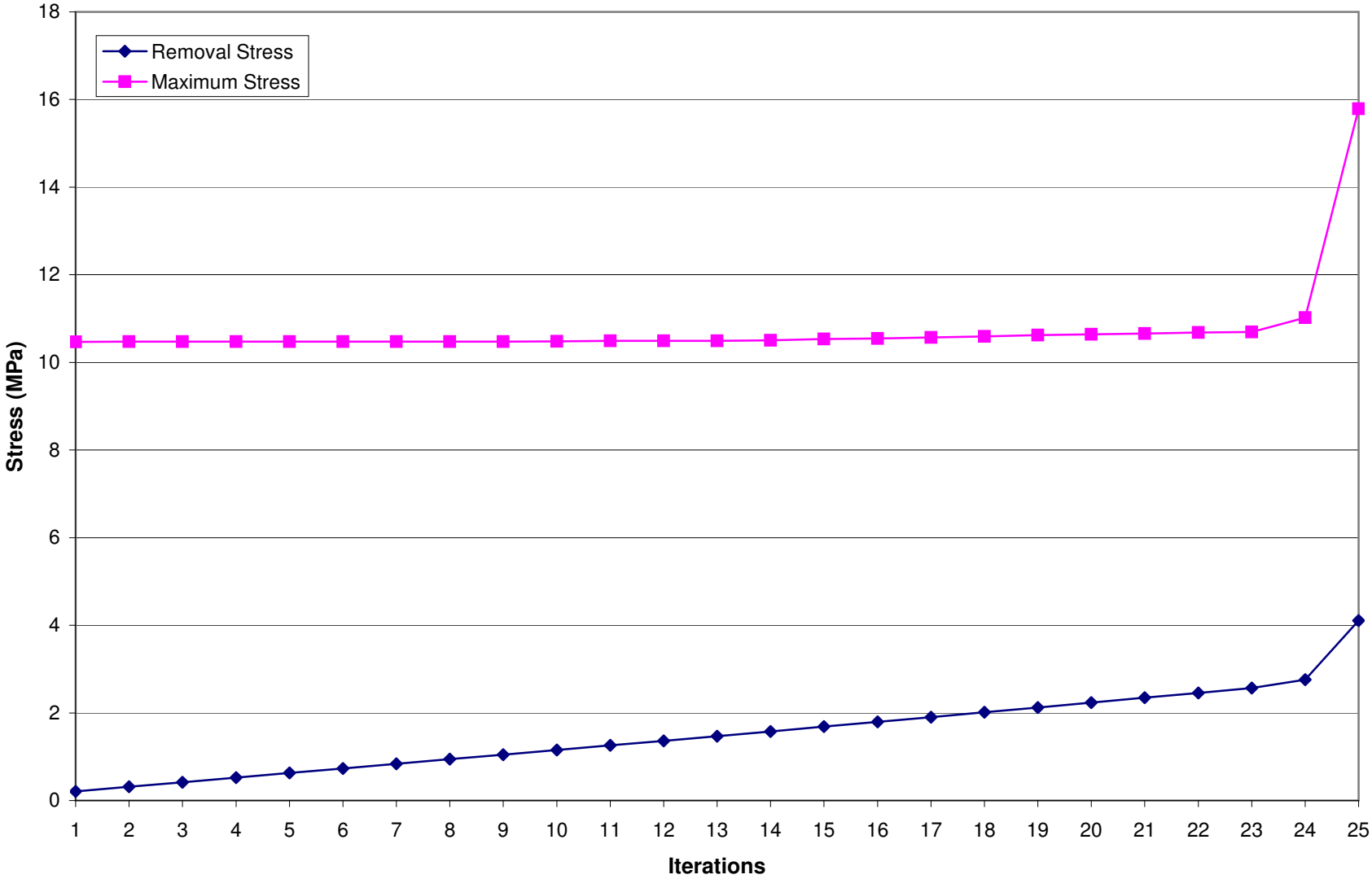
Graph A-27: Sandwich Tetrahedral Model 5 - Volume Remaining



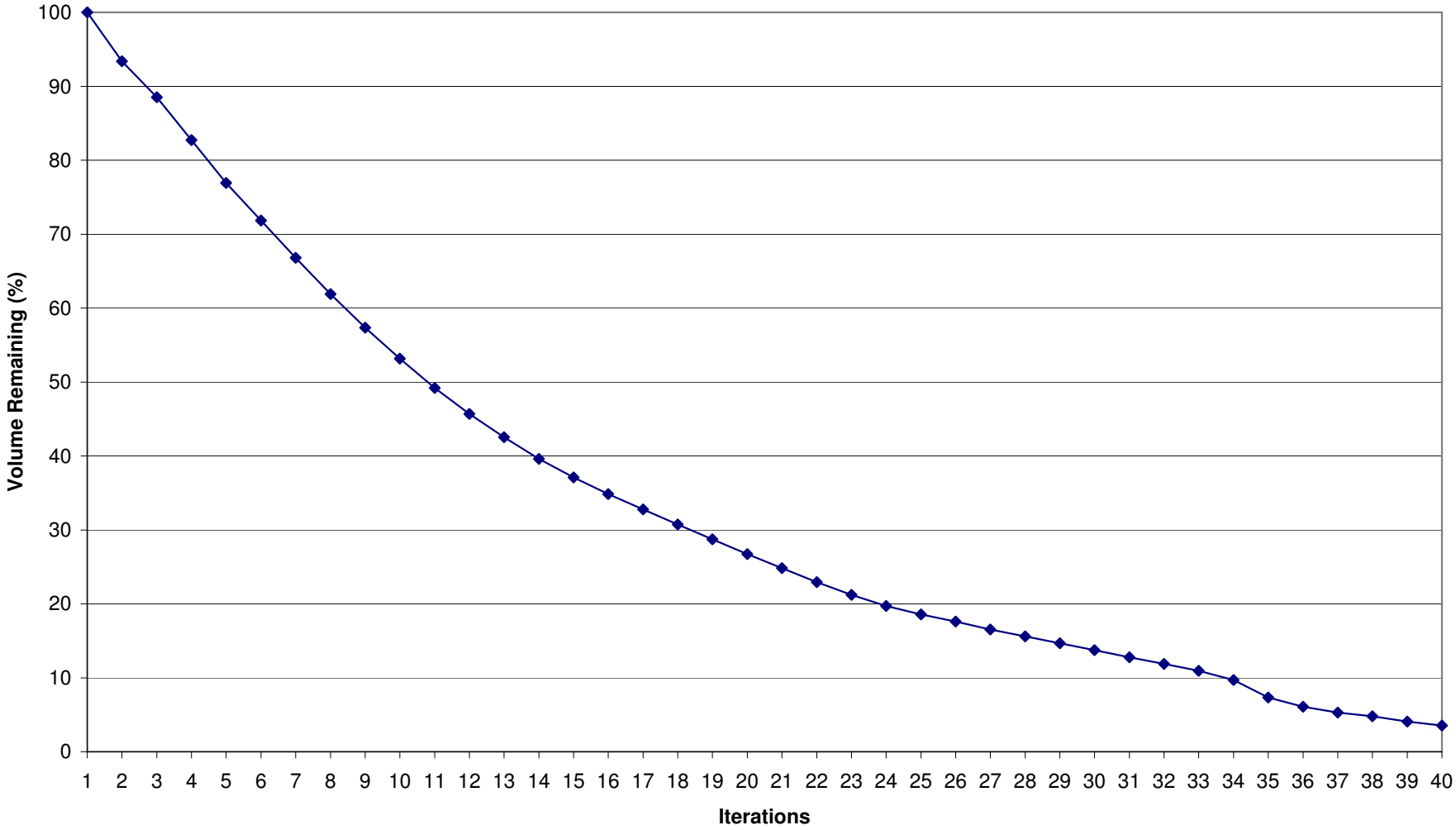
Graph A-28: Sandwich Tetrahedral Model 5 - Stress Results



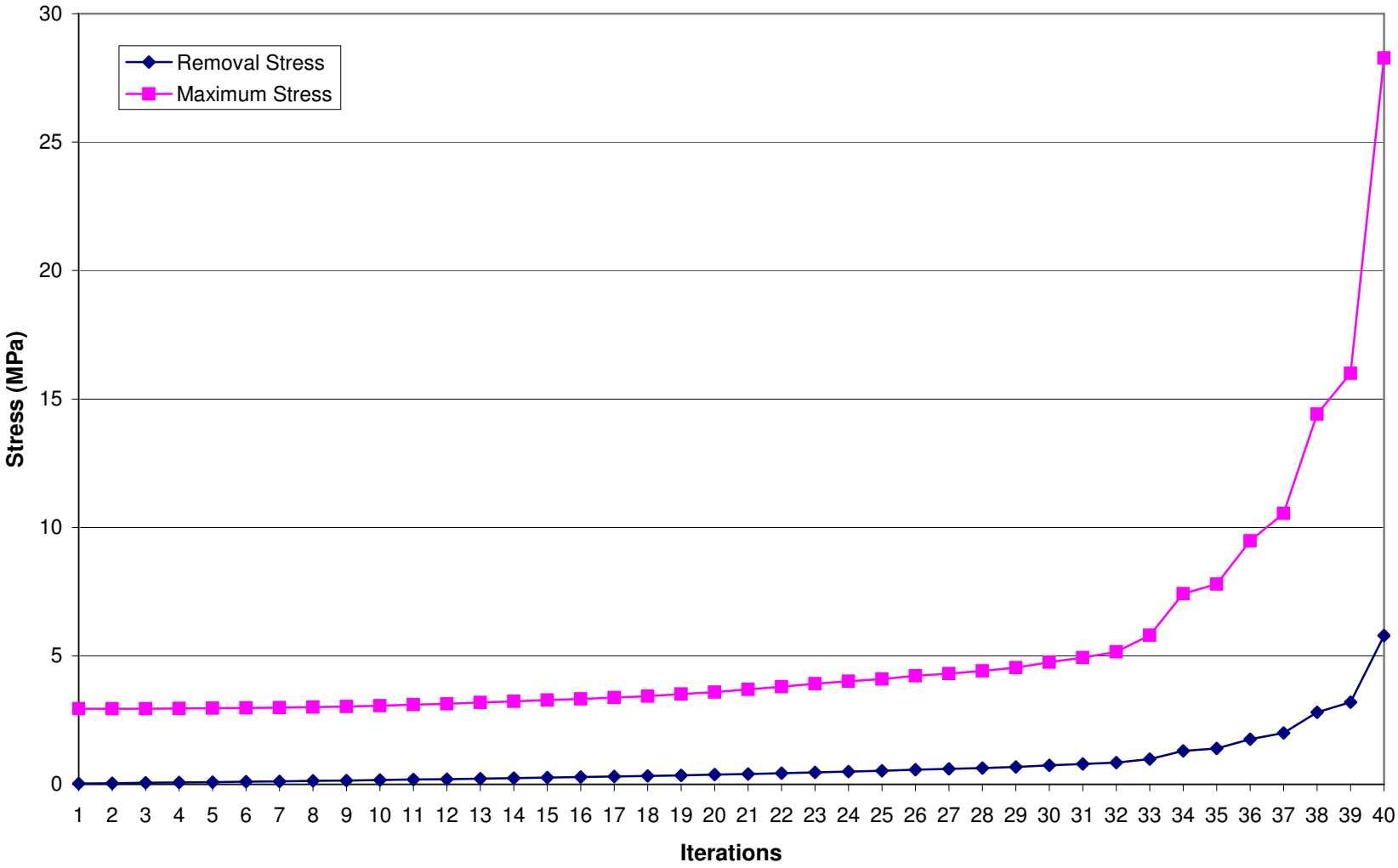
Graph A-29: Sandwich Tetrahedral Model 6 - Volume Remaining



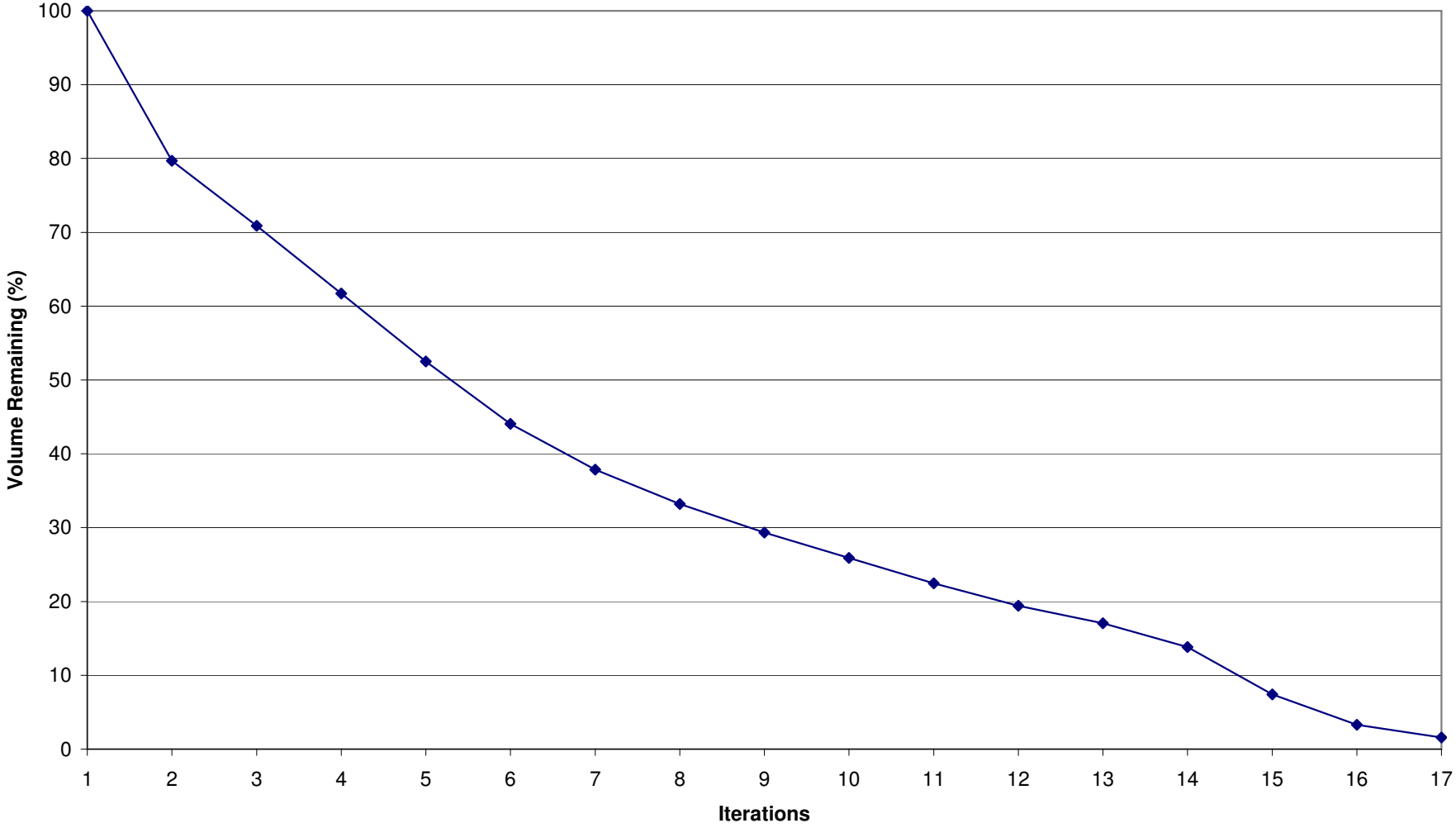
Graph A-30: Sandwich Tetrahedral Model 6 - Stress Results



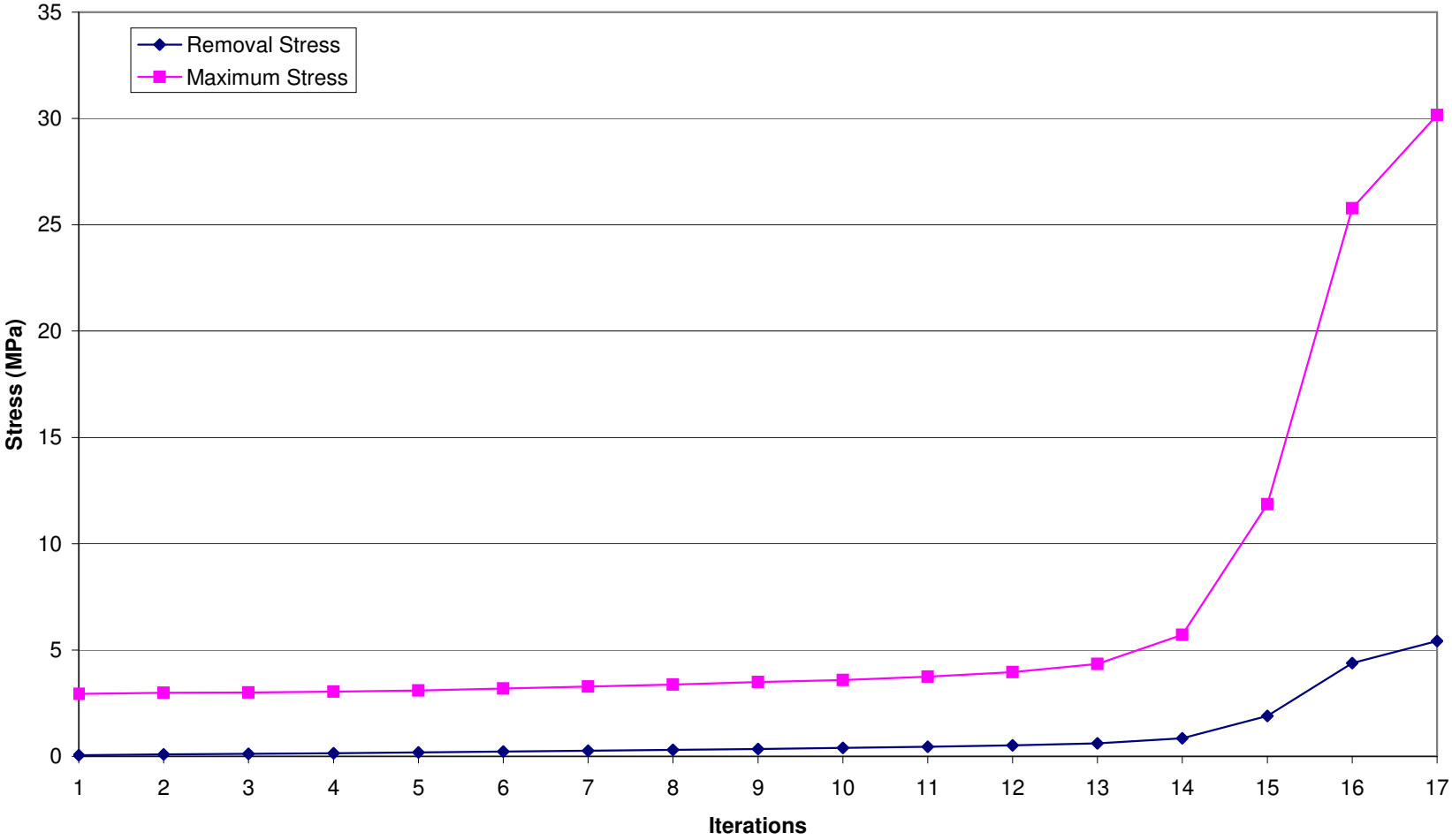
Graph A-31: Tailfin Model 1 - Volume Remaining



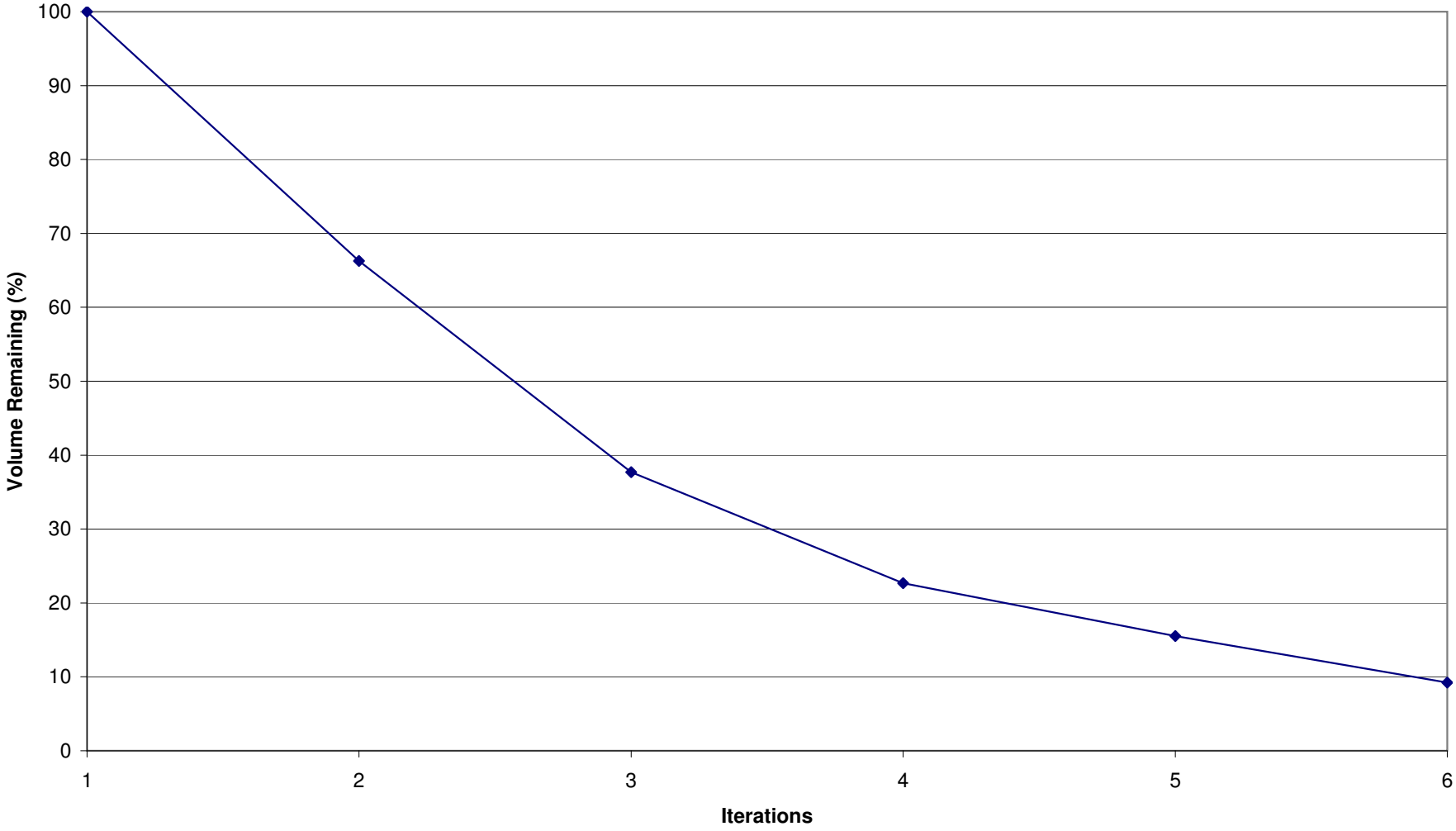
Graph A-32: Tailfin Model 1 - Stress Results



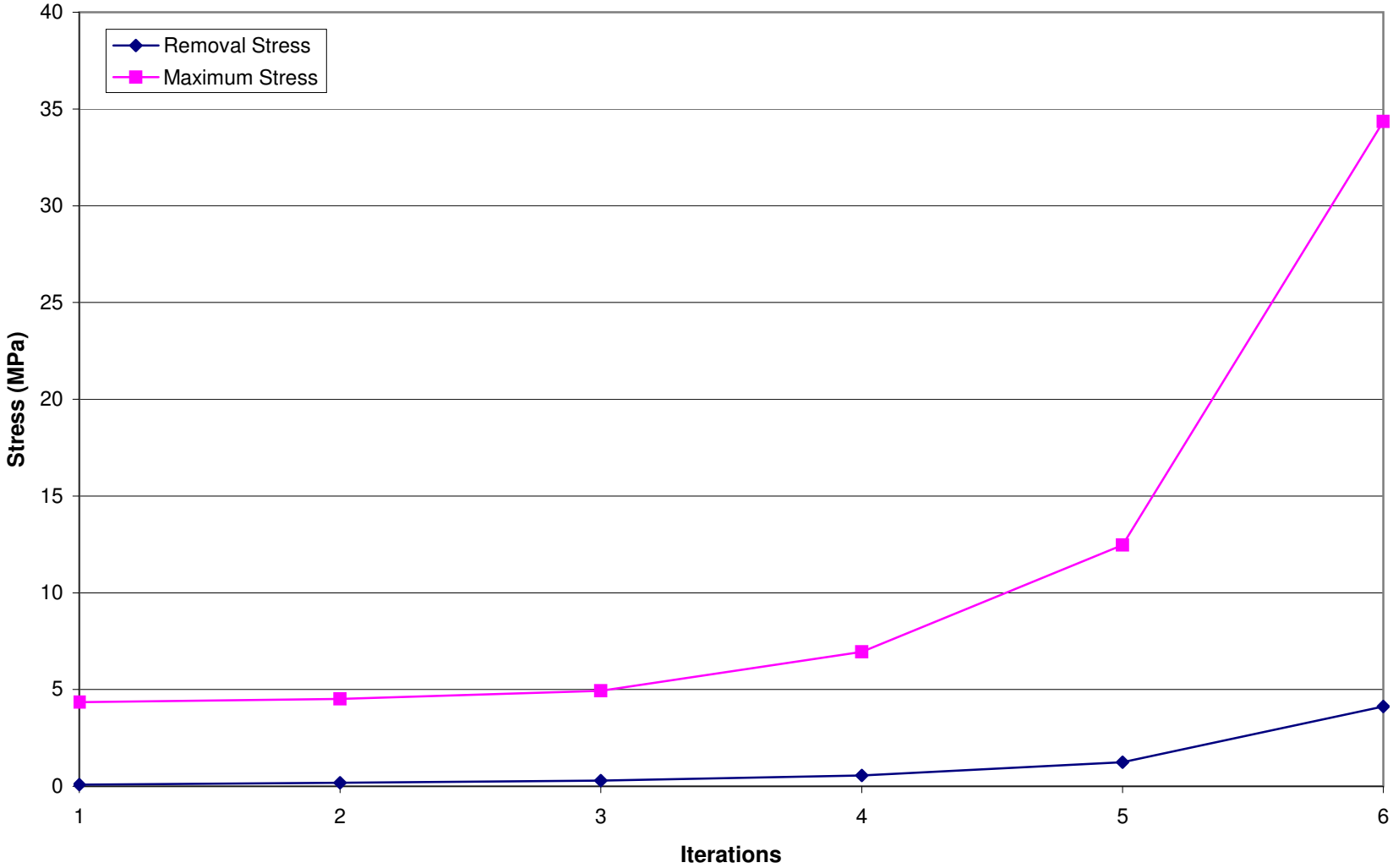
Graph A-33: Tailfin Model 2 - Volume Remaining



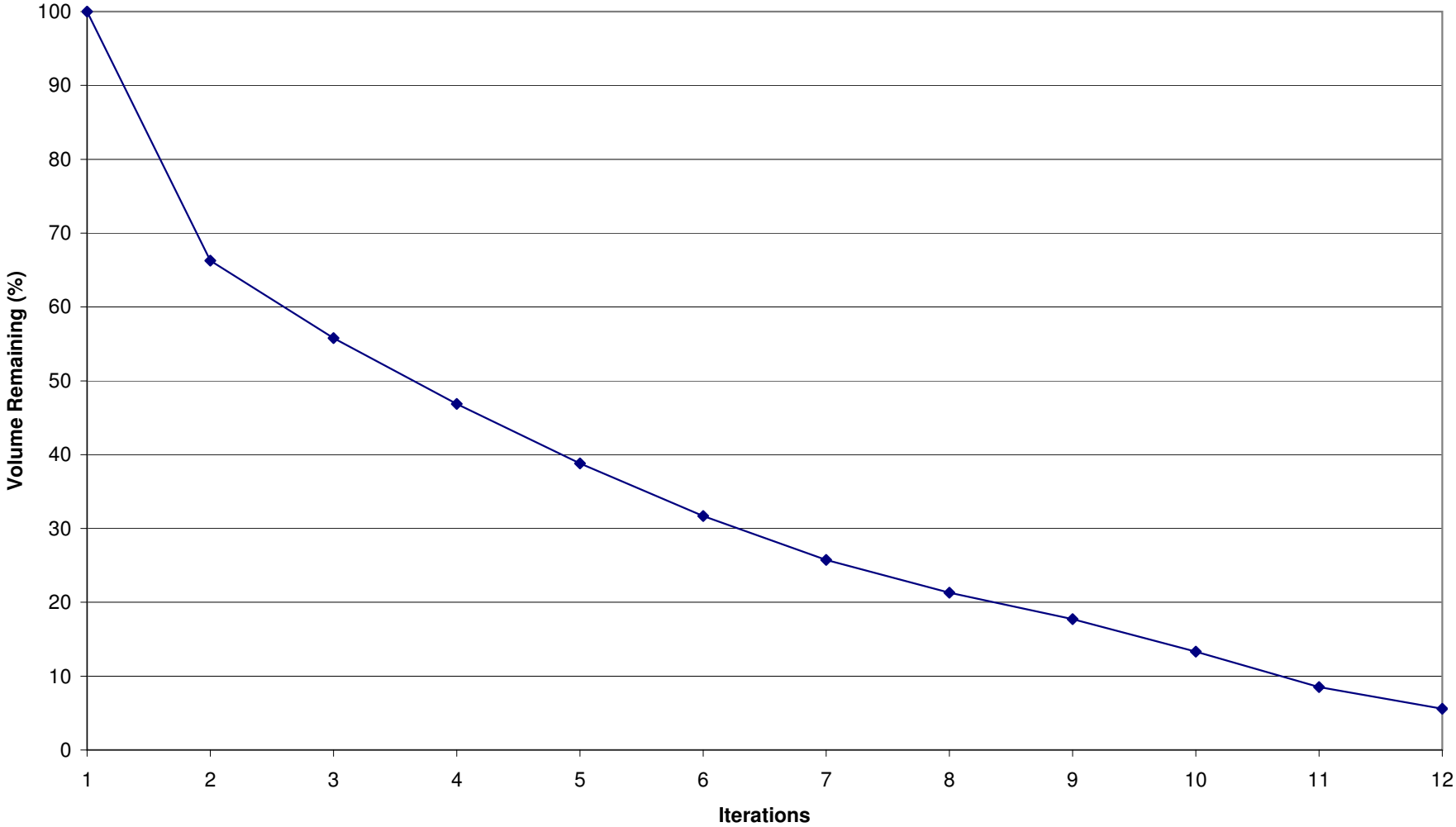
Graph A-34: Tailfin Model 2 - Stress Results



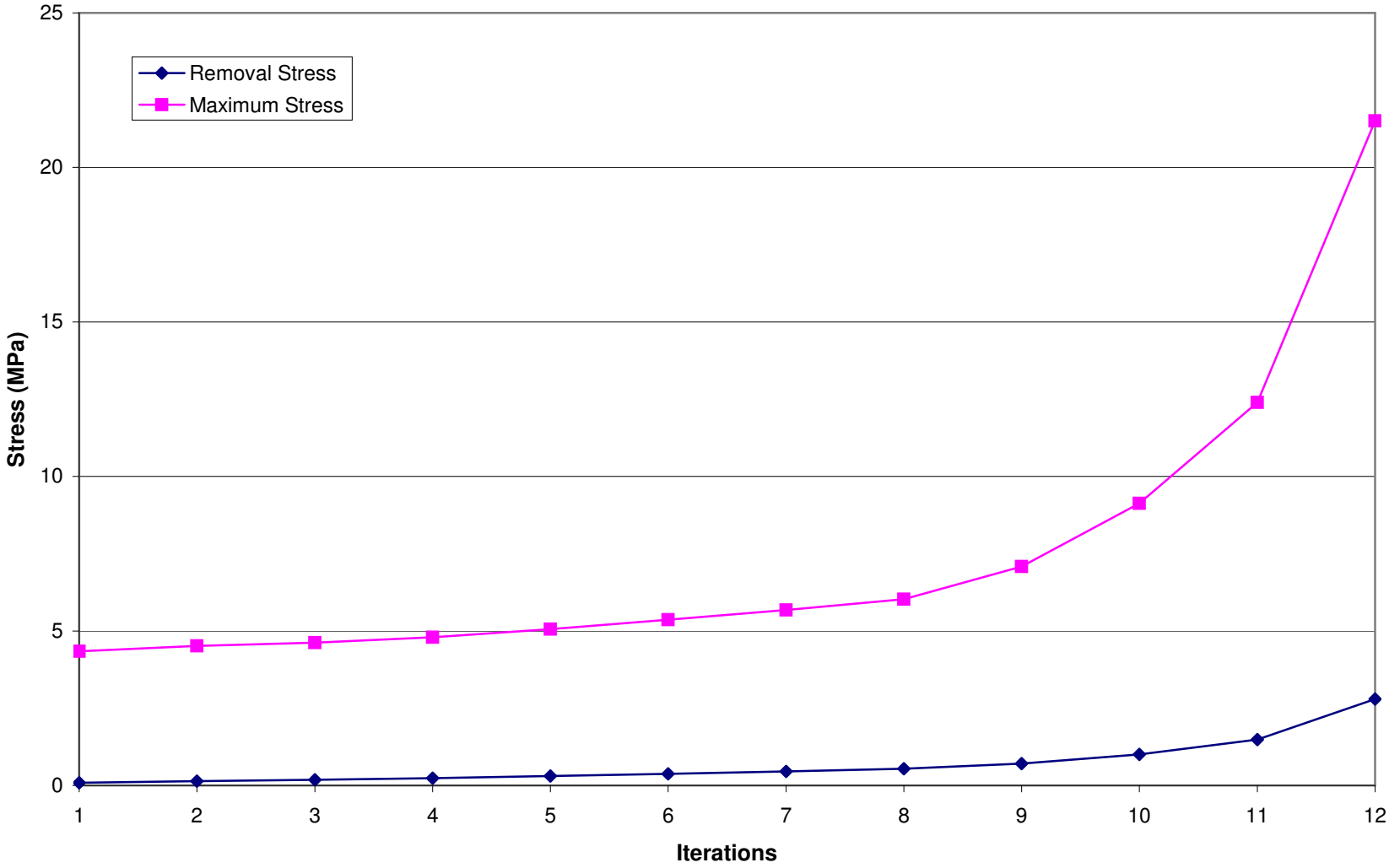
Graph A-35: Tailfin Model 4 - Volume Remaining



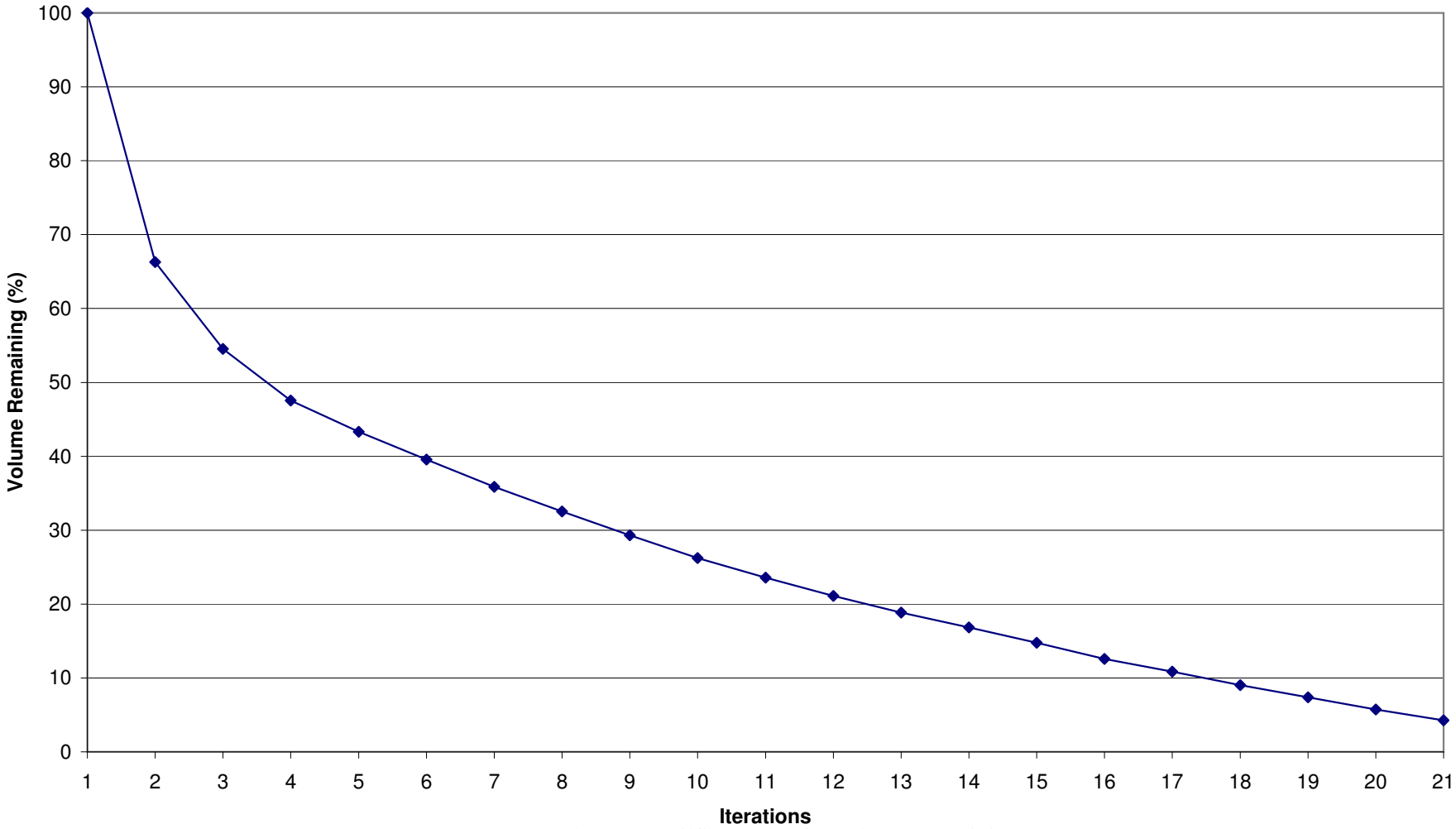
Graph A-36: Tailfin Model 4 - Stress Results



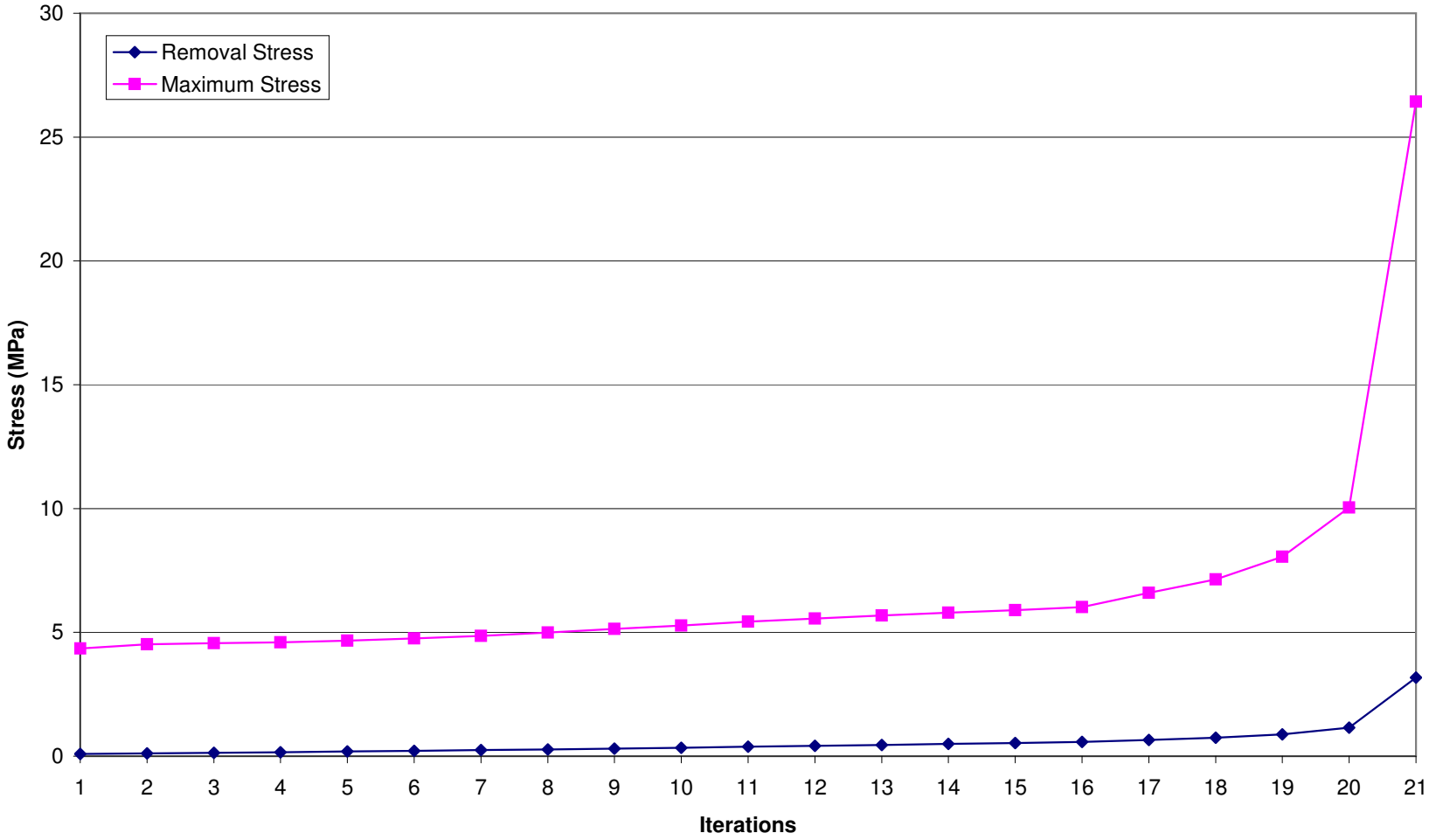
Graph A-37: Tailfin Model 5 - Volume Remaining



Graph A-38: Tailfin Model 5 - Stress Results



Graph A-39: Tailfin Model 6 - Volume Remaining



Graph A-40: Tailfin Model 6 - Stress Results

Appendix B

Computer Programs

```
!*****************************************************************************
!
!  PROGRAM: OptTest
!
!  PURPOSE: Run analysis and optimisation of model. This is the Master Control
Program
!
!*****************************************************************************

      program OptTest
      USE DFLIB
      implicit none

      ! Variables
      CHARACTER(20) bdffile
      CHARACTER(20) f06file
      CHARACTER(20) xdbfile
      CHARACTER(100) line1
      CHARACTER(80) line2
      CHARACTER(80) line3
      CHARACTER(80) line4
      REAL remlimit
      REAL stepremlimit
      INTEGER elemrem
      INTEGER elemadd
      INTEGER y
      INTEGER fail
      INTEGER failtrip
      INTEGER stresstrip
      LOGICAL(4) result1
      REAL time_begin, time_end

      ! Body of OptTest
      CALL CPU_TIME ( time_begin )

      OPEN(100, FILE = 'rundata.dat')
      READ(100, *) bdffile
      READ(100, *)
      READ(100, *) f06file
      READ(100, *)
      READ(100, *)
      READ(100, *) remlimit
      READ(100, *)
      READ(100, *) xdbfile
      CLOSE(100)
```

```
      OPEN(1, FILE = 'elemremlist.dat', status = 'replace')
      OPEN(2, FILE = 'elemaddlist.dat', status = 'replace')
      OPEN(300, FILE = 'remlist.dat', status = 'replace')

      WRITE(line1, 10) 'c:\msc.software\msc.nastran\bin\nastran ', bdffile
10    FORMAT(A40, A16)

      y = 1
      ESO: DO

       print*, 'This is iteration', y

       OPEN(200, FILE = 'loopcount.dat', status = 'replace')
       WRITE(200, *) y
       CLOSE(200)

       result1 = systemqq('clear.bat') !prepare

       result1 = systemqq(line1) !analysis

       WRITE(line2, 20) 'copy ', xdbfile, y, '_loop.xdb.bak' !Backup xdb
20  FORMAT(A5, A15, ' ', I3, A13)
       result1 = systemqq(line2)

       WRITE(line3, 30) 'copy ', bdffile, y, '_loop.bdf.bak' !Backup bdf
30     FORMAT(A5, A15, ' ', I3, A13)
       result1 = systemqq(line3)

       WRITE(line4, 40) 'copy ', f06file, y, '_loop.f06.bak' !Backup f06
40     FORMAT(A5, A15, ' ', I3, A13)!
       result1 = systemqq(line4)

       result1 = systemqq('FailCheck.exe') !Failcheck
       OPEN(6, FILE = 'failchk.dat')
       READ(6, *) fail
       CLOSE(6)
       IF (fail .eq. 0) THEN
        result1 = systemqq('FailReplace_1.exe')
        OPEN(8, FILE = 'failtrip.dat')
        READ(8, *) failtrip
        CLOSE(8)
        IF (failtrip .eq. 0) THEN
         print*, 'Model failed - failure stress reached in non-modifiable region'
         pause
         exit
        ELSE
```

```
            result1 = systemqq('FailFileAdd_1.exe')
            OPEN(7, FILE = 'failaddelem.dat') !Write add elements to file
            DO WHILE (.TRUE.)
             READ(7, *, END=100) elemadd
             IF (elemadd .eq. 0) cycle
             WRITE(2, *) y, elemadd, 'Fail_replace'
            END DO
100    CLOSE(7)
            y = y + 1
            cycle ESO
           END IF
          ELSE
           print*, 'No failure detected'
      END IF

          result1 = systemqq('StressLimit_1.exe')
          OPEN(3, FILE = 'remlimit.dat')
          READ(3, *) stepremlimit
          READ(3, *) stresstrip
          CLOSE(3)
          print*, ' Stress limit    is ', stepremlimit
          WRITE(300, *) y, stepremlimit

          result1 = systemqq('ESO_1.exe')

          result1 = systemqq('FileRem_1.exe')
          result1 = systemqq('FileAdd_1.exe')

          elemrem = 0
          OPEN(4, FILE = 'remelem.dat') !Write rem elements to file
          DO WHILE (.TRUE.)
           READ(4, *, END=150) elemrem
           IF ((elemrem .eq. 0) .or. (elemrem .lt. 0)) cycle
           WRITE(1, *) y, elemrem, 'ESORem'
          END DO
150  CLOSE(4)

          OPEN(5, FILE = 'addelem.dat') !Write add elements to file
          DO WHILE (.TRUE.)
           READ(5, *, END=200) elemadd
           IF (elemadd .eq. 0) cycle
           WRITE(2, *) y, elemadd, 'ESOAdd'
          END DO
200  CLOSE(5)

         IF (stresstrip .eq. 1)  THEN
```

```
        print*, 'Stress limit reached'
        pause
        exit
ELSE
        print*, 'Stress limit not reached'
END IF

      y = y + 1

      END DO ESO

      CALL CPU_TIME ( time_end )
      PRINT *, 'Time of operation was ', time_end - time_begin, ' seconds'
      pause

      end program OptTest
```

```
!*****************************************************************************
!  PROGRAM: ESO_1
!
!  PURPOSE:  Analysis Program
!
!*****************************************************************************

      program ESO_1

      ! Variables
      INTEGER count
      INTEGER count1
      INTEGER count2
      INTEGER count3
      INTEGER, ALLOCATABLE, DIMENSION(:) :: remlist
      INTEGER, ALLOCATABLE, DIMENSION(:) :: elemremlist
      INTEGER, ALLOCATABLE, DIMENSION(:) :: elemaddlist
      INTEGER, ALLOCATABLE, DIMENSION(:) :: addlist
      INTEGER :: elemno
      INTEGER elemrem
      INTEGER elemadd
      INTEGER, ALLOCATABLE, DIMENSION(:,:) :: ellistcomp
      REAL, ALLOCATABLE, DIMENSION(:,:) :: remelemlist
      REAL, ALLOCATABLE, DIMENSION(:,:) :: addelemlist
      REAL, ALLOCATABLE, DIMENSION(:,:) :: stresslist
      CHARACTER(len=150) :: line1
      CHARACTER(len=150) :: line2
      CHARACTER(len=150) :: line3
      CHARACTER(len=150) :: nline
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER i
      INTEGER j
      INTEGER x
      INTEGER y
      INTEGER z
      INTEGER cnrn
      INTEGER nodes(6)
      INTEGER elno
      INTEGER elmat
      CHARACTER(len=10) :: tofind
      CHARACTER(len=10) :: tofind2
      REAL :: stresslimit
      REAL :: elstress
```

```
      CHARACTER(30) bdffile
      CHARACTER(30) bdfnewfile
      CHARACTER(30) f06file
      REAL addlimit
      REAL remlimit
      REAL time_begin, time_end

      ! Body of ESO_1

      !!!!!!!!!!!!!!!!!!!!!! File Open
      CALL CPU_TIME ( time_begin )
      OPEN( 100, FILE = 'rundata.dat')
      READ(100, *) bdffile
      READ(100, *) bdfnewfile
      READ(100, *) f06file
      READ(100, *)
      READ(100, *) addlimit

      OPEN( 1, FILE = bdffile)
      OPEN( 2, FILE = f06file)
      OPEN( 3, FILE = 'remlimit.dat')
      OPEN( 4, FILE = 'addelem.dat', status='replace')
      OPEN( 5, FILE = 'remelem.dat', status='replace')

      READ(3, *) remlimit

      elemno = 1
      tofind = 'CTETRA'
      strlen= LEN(TRIM(tofind))

      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind))

      DO WHILE (.TRUE.)
       READ(1, '(a)', END = 100) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
       IF ((indx .eq. 1) .or. (indx2 .eq. 1) )THEN
        elemno = elemno + 1
       END IF
      END DO

100 CLOSE(1)

      allocate(remlist(elemno))
      allocate(elemremlist(elemno))
```

```
          allocate(elemaddlist(elemno))
          allocate(addlist(elemno))
          allocate(ellistcomp(elemno,2))
          allocate(remelemlist(elemno,2))
          allocate(addelemlist(elemno,2))
          allocate(stresslist(elemno,2))

          remlist = 0
          elemremlist = 0
          elemaddlist = 0
          addlist = 0
          ellistcomp = 0
          remelemlist = 0
          addelemlist = 0
          stresslist = 0

          OPEN( 1, FILE = bdffile)

          y = 1
          tofind = 'CTETRA'
          strlen= LEN(TRIM(tofind))

          tofind2 = 'CHEXA'
          strlen2 = LEN(TRIM(tofind))

          DO WHILE (.TRUE.)
           READ(1, '(a)', END = 109) line1
           indx = INDEX(line1, TRIM(tofind))
           indx2 = INDEX(line1, TRIM(tofind2))
           IF ((indx .eq. 1) .or. (indx2 .eq. 1) )THEN
            READ(line1(strlen+1:), *) elno, elmat, nodes(1), nodes(2), nodes(3), nodes(4),
nodes(5), nodes(6)
            READ( 1, '(a)') nline
            ellistcomp(y , 1) = elno
            ellistcomp(y , 2) = elmat
            y = y + 1
           END IF
          END DO

109 CLOSE(1)

          !Input results

          z = 1
          tofind = 'GRID'
          strlen = LEN(TRIM(tofind))
```

```
          tofind2 = 'CENTER'
          strlen2 = LEN(TRIM(tofind2))

          DO WHILE (.TRUE.)
           READ(2, '(a)', END=200) line2
           indx = INDEX(line2, trim(tofind))
           IF (indx .eq. 24) THEN
            READ(line2, 110) elno
110   FORMAT(T5, I7)
             loop: DO WHILE (.TRUE.)
              READ(2, '(a)') line3
              indx2 = INDEX(line3, TRIM(tofind2))
              IF (indx2 .eq. 18) THEN
               READ(line3, 111) elstress
111                FORMAT(T118, E18.8)
              END IF
              IF (elstress /= 0) exit loop
             END DO loop
             stresslist(z, 1) = elno
             stresslist(z, 2) = elstress
             z = z + 1
             elstress = 0
           END IF
          END DO

200 CLOSE(2)

          y = 1
          DO count1 = 1, elemno
           IF (ellistcomp(count1, 1) .eq. 0) THEN
            exit
    ElSE
           IF (ellistcomp(count1, 2) .eq. 1) THEN
           remlist(y) = ellistcomp(count1, 1)
     y = y + 1
           END IF
          END IF
          END DO

          DO count2 = 1, elemno
           IF (remlist(count2) .eq. 0) exit
           remelemlist(count2, 1) = remlist(count2)
           DO count3 = 1, elemno
            IF (stresslist(count3, 1) .eq. remlist(count2)) THEN
             remelemlist(count2, 2) = stresslist(count3, 2)
            END IF
```

```
        END DO
       END DO

CALL Elemremoval(remelemlist, elemno, elemremlist, remlimit)

       y = 1
       DO count1 = 1, elemno
        IF (ellistcomp(count1, 1) .eq. 0) THEN
         exit
 ELSE
         IF (ellistcomp(count1, 2) .eq. 2) THEN
          addlist(y) = ellistcomp(count1, 1)
          y = y + 1
  END IF
        END IF
       END DO


       DO count2 = 1, elemno
       IF (addlist(count2) .eq. 0) exit
        addelemlist(count2, 1) = addlist(count2)
        DO count3 = 1, elemno
         IF (stresslist(count3, 1) .eq. addlist(count2)) THEN
          addelemlist(count2, 2) = stresslist(count3, 2)
         END IF
        END DO
       END DO

       !ESO
        CALL Elemaddition(addelemlist, elemno, elemaddlist, addlimit)

       !!!!!!!!!!!!!!!!!!!!!!!!!!!! Output of results
       remfile: DO count = 1, elemno
        IF (elemremlist(count) .eq. 0 ) exit remfile
        WRITE(5,*) elemremlist(count)
END DO remfile

addfile: DO count = 1, elemno
        IF (elemaddlist(count) .eq. 0 ) exit addfile
        WRITE(4,*) elemaddlist(count)
END DO addfile

       CALL CPU_TIME ( time_end )
       PRINT *, 'Time for ESO was ', time_end - time_begin, ' seconds'

       end program ESO_1
```

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      SUBROUTINE elemremoval(array, arrayno, arrayrem, limit)
implicit none

      INTEGER :: arrayno
      INTEGER, DIMENSION(arrayno) :: arrayrem
      REAL, DIMENSION (arrayno , 2), INTENT(INOUT) :: array
      INTEGER :: ctta
      INTEGER :: arrayremstress
      INTEGER :: z
      INTEGER :: count1
      REAL :: limit

      arrayrem = 0
      z = 1
      loop: DO ctta = 1, arrayno
IF (array( ctta , 2) .lt. limit) THEN
        arrayrem(z) = array( ctta, 1)
        z = z + 1
       END IF
      END DO loop

      END SUBROUTINE elemremoval

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      SUBROUTINE elemaddition(list, no, addlist, limit)
implicit none

      INTEGER :: no
      REAL, DIMENSION (no,2), INTENT(INOUT) :: list
      INTEGER, DIMENSION(no) :: addlist
      INTEGER :: count
      INTEGER :: x
      REAL :: limit

      addlist = 0
      x = 1
      loop: DO count = 1, no
IF (list(count , 2) .gt. limit) THEN
        addlist(x) = list(count, 1)
```

```
        x = x + 1
END IF
      END DO loop

      END SUBROUTINE elemaddition

      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
!*********************************************************************
!
!  PROGRAM: StressLimit_1
!
!  PURPOSE:  Calculate the stress limit for The optimisation
!
!*********************************************************************

      program StressLimit_1
      implicit none


      INTEGER count1
      INTEGER count2
      INTEGER count3
      INTEGER count4
      INTEGER, ALLOCATABLE, DIMENSION(:) :: avglist
      INTEGER :: elemno
      INTEGER, ALLOCATABLE, DIMENSION (:, :) :: ellistcomp
      REAL, ALLOCATABLE, DIMENSION(:, :) :: elemlist
      REAL, ALLOCATABLE, DIMENSION(:, :) :: stresslist
      CHARACTER(len=150) :: line1
      CHARACTER(len=150) :: line2
      CHARACTER(len=150) :: line3
      CHARACTER(len=150) :: nline
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER y
      INTEGER z
      INTEGER nodes(6)
      INTEGER elno
      INTEGER elmat
      INTEGER trip
      CHARACTER(len=10) :: tofind
      CHARACTER(len=10) :: tofind2
      REAL :: stresslimit
      REAL :: elstress
      CHARACTER(30) bdffile
      CHARACTER(30) f06file
      REAL sumstress
      REAL avgstress
      REAL rem
      REAL time_begin, time_end
      REAL RR0
```

```
      REAL ER
      INTEGER LC
      REAL stressmax

      ! Body of StressLimit_1

      !!!!!!!!!!!!!!!!!!!!!! File Open
      CALL CPU_TIME ( time_begin )
      OPEN( 100, FILE = 'rundata.dat')
      READ(100, *) bdffile
      READ(100, *)
      READ(100, *) f06file
      READ(100, *) elemno
      READ(100, *)
      READ(100, *) stresslimit
      READ(100, *)
      READ(100, *)
      READ(100, *) RR0
      READ(100, *) ER

      OPEN(200, FILE = 'loopcount.dat')
      READ(200, *) LC

      OPEN( 1, FILE = bdffile)
      OPEN( 2, FILE = f06file)

      !Input nodeconn

      elemno = 0
      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))

      DO WHILE (.TRUE.)
       READ(1, '(a)', END = 10) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
       IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
        elemno = elemno + 1
       END IF
      END DO

10    CLOSE(1)

      allocate(avglist(elemno))
```

```
          allocate(ellistcomp(elemno,2))
          allocate(elemlist(elemno,2))
          allocate(stresslist(elemno,2))

          OPEN( 1, FILE = bdffile)

          y = 1
          tofind = 'CTETRA'
          strlen = LEN(TRIM(tofind))
          tofind2 = 'CHEXA'
          strlen2 = LEN(TRIM(tofind2))

          DO WHILE (.TRUE.)
           READ(1, '(a)', END = 100) line1
           indx = INDEX(line1, TRIM(tofind))
           indx2 = INDEX(line1, TRIM(tofind2))
           IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
            READ(line1(strlen+1:), *) elno, elmat, nodes(1), nodes(2), nodes(3), nodes(4),
nodes(5), nodes(6)
            READ( 1, '(a)') nline
            !READ( nline, *) nodes(7), nodes(8), nodes(9), nodes(10)
            ellistcomp(y , 1) = elno
            ellistcomp(y , 2) = elmat
            y = y + 1
           END IF
          END DO

100 CLOSE(1)

          !Input results

          z = 1
          tofind = 'GRID'
          strlen = LEN(TRIM(tofind))
          tofind2 = 'CENTER'
          strlen2 = LEN(TRIM(tofind2))

          DO WHILE (.TRUE.)
           READ(2, '(a)', END=200) line2
           indx = INDEX(line2, trim(tofind))
           IF (indx .eq. 24) THEN
            READ(line2, 110) elno
110   FORMAT(T5, I7)
            loop: DO WHILE (.TRUE.)
             READ(2, '(a)') line3
             indx2 = INDEX(line3, TRIM(tofind2))
```

```
            IF (indx2 .eq. 18) THEN
             READ(line3, 111) elstress
111               FORMAT(T118, E18.8)
            END IF
            IF (elstress /= 0) exit loop
           END DO loop
           stresslist(z, 1) = elno
           stresslist(z, 2) = elstress
            z = z + 1
           elstress = 0
          END IF
         END DO

200 CLOSE(2)

         y = 1
         DO count1 = 1, elemno
          IF (ellistcomp(count1, 1) .eq. 0) THEN
           exit
    ElSE
           IF ((ellistcomp(count1, 2) .eq. 1) .or. (ellistcomp(count1, 2) .eq. 3)) THEN
           avglist(y) = ellistcomp(count1, 1)
     y = y + 1
           END IF
          END IF
         END DO

         DO count2 = 1, elemno
          IF (avglist(count2) .eq. 0) exit
          elemlist(count2, 1) = avglist(count2)
          DO count3 = 1, elemno
           IF (stresslist(count3, 1) .eq. avglist(count2)) THEN
            elemlist(count2, 2) = stresslist(count3, 2)
           END IF
          END DO
         END DO

   stressmax = elemlist(1,1)
   DO count4 = 2, elemno
          IF (elemlist(count4, 2) .gt. stressmax) THEN
           stressmax = elemlist(count4, 2)
          END IF
         END DO

         rem = (RR0 + (LC-1)*ER)*stressmax
         IF (rem .gt. stresslimit) THEN
```

```
        trip = 1
       ELSE
        trip = 0
       END IF

       OPEN(3, FILE = 'remlimit.dat', status = 'replace')
WRITE(3, *) rem
WRITE(3, *) trip
CLOSE(3)

CALL CPU_TIME ( time_end )
PRINT *, 'Time for StressLimit was ', time_end - time_begin, ' seconds'

end program StressLimit_1
```

```
!*****************************************************************************
!
!  PROGRAM: FailCheck
!
!  PURPOSE:  Determines Failure for the Optimisation Process
!
!*****************************************************************************

      program FailCheck

      implicit none

      ! Variables
      INTEGER :: elemno
      REAL, ALLOCATABLE, DIMENSION(:,:) :: stresslist
      CHARACTER(len=150) :: line2
      CHARACTER(len=150) :: line3
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER y
      INTEGER z
      INTEGER elno
      INTEGER fail
      CHARACTER(len=10) :: tofind
      CHARACTER(len=10) :: tofind2
      REAL :: elstress
      CHARACTER(30) f06file
      REAL faillimit

      ! Body of FailCheck
      OPEN( 100, FILE = 'rundata.dat')

      READ(100, *)
      READ(100, *)
      READ(100, *) f06file
      READ(100, *) elemno
      READ(100, *)
      READ(100, *)
      READ(100, *) faillimit

      OPEN( 1, FILE = 'failchk.dat', status = 'replace')
      OPEN( 2, FILE = f06file)
      !Input results
```

```
          elemno = 1
          tofind = 'GRID'
          strlen = LEN(TRIM(tofind))
          tofind2 = 'CENTER'
          strlen2 = LEN(TRIM(tofind2))

          DO WHILE (.TRUE.)
           READ(2, '(a)', END=100) line2
           indx = INDEX(line2, trim(tofind))
           IF (indx .eq. 24) THEN
            elemno = elemno + 1
           END IF
          END DO

100 CLOSE(2)

          allocate(stresslist(elemno,2))

          OPEN( 2, FILE = f06file)

          z = 1
          tofind = 'GRID'
          strlen = LEN(TRIM(tofind))
          tofind2 = 'CENTER'
          strlen2 = LEN(TRIM(tofind2))

          DO WHILE (.TRUE.)
           READ(2, '(a)', END=200) line2
           indx = INDEX(line2, trim(tofind))
           IF (indx .eq. 24) THEN
            READ(line2, 110) elno
110   FORMAT(T9, I3)
            loop: DO WHILE (.TRUE.)
             READ(2, '(a)') line3
             indx2 = INDEX(line3, TRIM(tofind2))
             IF (indx2 .eq. 18) THEN
              READ(line3, 111) elstress
111               FORMAT(T118, E18.8)
             END IF
             IF (elstress /= 0) exit loop
            END DO loop
            stresslist(z, 1) = elno
            stresslist(z, 2) = elstress
            z = z + 1
            elstress = 0
           END IF
```

```
        END DO

200 CLOSE(2)

        fail = 1
        DO y = 1, elemno
         IF (stresslist(y, 2) .gt. faillimit) THEN
          fail = 0
    exit
         END IF
   END DO

        WRITE(1, *) fail
        close(1)
        end program FailCheck
```

```
!*****************************************************************************
!
!  PROGRAM: FailReplace_1
!
!  PURPOSE:  Select elements for re-addition after failure
!
!*****************************************************************************

      program FailReplace_1

      implicit none
      ! Variables
      INTEGER count
      INTEGER, ALLOCATABLE, DIMENSION(:) :: tempnode
      INTEGER, ALLOCATABLE, DIMENSION(:) :: failnode
      INTEGER, ALLOCATABLE, DIMENSION(:) :: tempfaillist
      INTEGER, ALLOCATABLE, DIMENSION(:) :: failaddlist
      INTEGER, ALLOCATABLE, DIMENSION(:) :: tempadd
      INTEGER, ALLOCATABLE, DIMENSION(:) :: faillist
      INTEGER :: elemno
      INTEGER, ALLOCATABLE, DIMENSION (:, :) :: ellistcomp
      REAL, ALLOCATABLE, DIMENSION(:, :) :: stresslist
      CHARACTER(len=150) :: line1
      CHARACTER(len=150) :: line2
      CHARACTER(len=150) :: line3
      CHARACTER(len=150) :: nline
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER r
      INTEGER s
      INTEGER t
      INTEGER x
      INTEGER y
      INTEGER z
      INTEGER nodest(10)
      INTEGER nodesh(8)
      INTEGER elno
      INTEGER elmat
      CHARACTER(len=10) :: tofind
      CHARACTER(len=10) :: tofind2
      REAL :: faillimit
      REAL :: elstress
      CHARACTER(30) bdffile
      CHARACTER(30) f06file
```

```
      REAL time_begin, time_end
      REAL sumfailadd

      ! Body of FailReplace_1

      !!!!!!!!!!!!!!!!!!!!!!! File Open
      CALL CPU_TIME ( time_begin )
      OPEN( 100, FILE = 'rundata.dat')

      READ(100, *) bdffile
      READ(100, *)
      READ(100, *) f06file
      READ(100, *) elemno
      READ(100, *)
      READ(100, *)
      READ(100, *) faillimit

      OPEN( 1, FILE = bdffile)
      OPEN( 2, FILE = f06file)

      !Input nodeconn

      elemno = 1
      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen = LEN(TRIM(tofind2))


      DO WHILE (.TRUE.)
       READ(1, '(a)', END = 100) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
       IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
        elemno = elemno + 1
       END IF
      END DO

100   CLOSE(1)

      allocate(tempnode(elemno))
      allocate(failnode(elemno))
      allocate(tempfaillist(elemno))
      allocate(failaddlist(elemno))
      allocate(tempadd(elemno))
      allocate(faillist(elemno))
```

```
      allocate(ellistcomp(elemno,12))
      allocate(stresslist(elemno,2))

      !Input results

      z = 1
      tofind = 'GRID'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CENTER'
      strlen2 = LEN(TRIM(tofind2))

      DO WHILE (.TRUE.)
       READ(2, '(a)', END=200) line2
       indx = INDEX(line2, trim(tofind))
       IF (indx .eq. 24) THEN
        READ(line2, 110) elno
110   FORMAT(T5, I7)
        loop: DO WHILE (.TRUE.)
         READ(2, '(a)') line3
         indx2 = INDEX(line3, TRIM(tofind2))
         IF (indx2 .eq. 18) THEN
          READ(line3, 111) elstress
111             FORMAT(T118, E18.8)
         END IF
         IF (elstress /= 0) exit loop
        END DO loop
        stresslist(z, 1) = elno
        stresslist(z, 2) = elstress
        z = z + 1
        elstress = 0
       END IF
      END DO

200 CLOSE(2)

      OPEN( 1, FILE = bdffile)

      y = 1
      tofind = 'CTETRA'
      strlen= LEN(TRIM(tofind))

      DO WHILE (.TRUE.)
       READ(1, '(a)', END = 300) line1
       indx = INDEX(line1, TRIM(tofind))
       IF (indx .eq. 1) THEN
        READ(line1(strlen+1:), *) elno, elmat, nodest(1), nodest(2), nodest(3),
```

```
nodest(4), nodest(5), nodest(6)
            READ( 1, '(a)') nline
            READ( nline, *) nodest(7), nodest(8), nodest(9), nodest(10)
            ellistcomp(y , 1) = elno
            ellistcomp(y , 2) = nodest(1)
            ellistcomp(y , 3) = nodest(2)
            ellistcomp(y , 4) = nodest(3)
            ellistcomp(y , 5) = nodest(4)
            ellistcomp(y , 6) = nodest(5)
            ellistcomp(y , 7) = nodest(6)
            ellistcomp(y , 8) = nodest(7)
            ellistcomp(y , 9) = nodest(8)
            ellistcomp(y , 10) = nodest(9)
            ellistcomp(y , 11) = nodest(10)
            ellistcomp(y , 12) = elmat
            y = y + 1
           END IF
          END DO

300 CLOSE(1)

          r = 1
          DO y = 1, elemno
           IF (stresslist(y,1) .eq. 0) exit
           IF (stresslist(y,2) .gt. faillimit) THEN
            tempfaillist(r) = stresslist(y,1)
            r = r + 1
           END IF
          END DO

          s = 1
          loop1: DO x = 1, elemno
           IF (tempfaillist(x) .eq. 0) exit loop1
           DO y = 1, elemno
            IF (ellistcomp(y,1) .eq. 0) cycle loop1
            IF (ellistcomp(y,1) .eq. tempfaillist(x) .and. (ellistcomp(y,12) /= 2)) THEN
             faillist(s) = tempfaillist(x)
             s = s + 1
            END IF
           END DO
          END DO loop1

          DO z = 1, elemno
           IF (ellistcomp(z,1) .eq. faillist(1)) THEN
            tempnode(1) = ellistcomp(z,2)
      CALL combine(tempnode, failnode, elemno)
```

```
 END IF
END DO

loop2: DO x = 1, elemno
 IF (faillist(x) .eq. 0) exit loop2
 DO y = 1, elemno
  IF (ellistcomp(y,1) .eq. 0) cycle loop2
  IF (ellistcomp(y,1) .eq. faillist(x)) THEN
   tempnode(1) = ellistcomp(y,2)
   tempnode(2) = ellistcomp(y,3)
   tempnode(3) = ellistcomp(y,4)
   tempnode(4) = ellistcomp(y,5)
   CALL combine(tempnode, failnode, elemno)
  END IF
 END DO
END DO loop2

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(1)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(2)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(3)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
```

```
        END DO
       END DO
       CALL combine(tempadd, failaddlist, elemno)

       DO y = 1, elemno
        DO z = 2,11
         IF (ellistcomp(y,z) .eq. failnode(4)) THEN
          IF (ellistcomp(y,12) .eq. 2) THEN
           tempadd(1) = ellistcomp(y,1)
          END IF
         END IF
        END DO
       END DO
       CALL combine(tempadd, failaddlist, elemno)

       DO x = 1, elemno
t = 1
        IF (failnode(x) .eq. 0) exit
        DO y = 1, elemno
         DO z = 2,11
          IF (ellistcomp(y,z) .eq. failnode(x)) THEN
           IF (ellistcomp(y,12) .eq. 2) THEN
            tempadd(t) = ellistcomp(y,1)
                  t = t + 1
           END IF
          END IF
         END DO
        END DO
        CALL combine(tempadd, failaddlist, elemno)
       END DO

       deallocate(ellistcomp)

       allocate(ellistcomp(elemno, 10))

       OPEN( 1, FILE = bdffile)

       y = 1
       tofind = 'CHEXA'
       strlen= LEN(TRIM(tofind))

       DO WHILE (.TRUE.)
        READ(1, '(a)', END = 400) line1
        indx = INDEX(line1, TRIM(tofind))
        IF (indx .eq. 1) THEN
         READ(line1(strlen+1:), *) elno, elmat, nodesh(1), nodesh(2), nodesh(3),
```

```
nodesh(4), nodesh(5), nodesh(6)
            READ( 1, '(a)') nline
            READ( nline, *) nodesh(7), nodesh(8)
            ellistcomp(y , 1) = elno
            ellistcomp(y , 2) = nodesh(1)
            ellistcomp(y , 3) = nodesh(2)
            ellistcomp(y , 4) = nodesh(3)
            ellistcomp(y , 5) = nodesh(4)
            ellistcomp(y , 6) = nodesh(5)
            ellistcomp(y , 7) = nodesh(6)
            ellistcomp(y , 8) = nodesh(7)
            ellistcomp(y , 9) = nodesh(8)
            ellistcomp(y , 10) = elmat
            y = y + 1
           END IF
          END DO

400 CLOSE(1)

          faillist = 0
          failnode = 0
          tempnode = 0
          tempadd = 0
          s = 1
          loop3: DO x = 1, elemno
           IF (tempfaillist(x) .eq. 0) exit loop3
           DO y = 1, elemno
            IF (ellistcomp(y,1) .eq. 0) cycle loop3
            IF (ellistcomp(y,1) .eq. tempfaillist(x) .and. (ellistcomp(y,10) /= 2)) THEN
             faillist(s) = tempfaillist(x)
             s = s + 1
            END IF
           END DO
          END DO loop3

          DO z = 1, elemno
           IF (ellistcomp(z,1) .eq. faillist(1)) THEN
            tempnode(1) = ellistcomp(z,2)
      CALL combine(tempnode, failnode, elemno)
           END IF
          END DO

          loop4: DO x = 1, elemno
           IF (faillist(x) .eq. 0) exit loop4
           DO y = 1, elemno
            IF (ellistcomp(y,1) .eq. 0) cycle loop4
```

```
  IF (ellistcomp(y,1) .eq. faillist(x)) THEN
   tempnode(1) = ellistcomp(y,2)
   tempnode(2) = ellistcomp(y,3)
   tempnode(3) = ellistcomp(y,4)
   tempnode(4) = ellistcomp(y,5)
   tempnode(5) = ellistcomp(y,6)
   tempnode(6) = ellistcomp(y,7)
   tempnode(7) = ellistcomp(y,8)
   tempnode(8) = ellistcomp(y,9)
   CALL combine(tempnode, failnode, elemno)
  END IF
 END DO
END DO loop4

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(1)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(2)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(3)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)
```

```
DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(4)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(5)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(6)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
 DO z = 2,11
  IF (ellistcomp(y,z) .eq. failnode(7)) THEN
   IF (ellistcomp(y,12) .eq. 2) THEN
    tempadd(1) = ellistcomp(y,1)
   END IF
  END IF
 END DO
END DO
CALL combine(tempadd, failaddlist, elemno)

DO y = 1, elemno
```

```
        DO z = 2,11
         IF (ellistcomp(y,z) .eq. failnode(8)) THEN
          IF (ellistcomp(y,12) .eq. 2) THEN
           tempadd(1) = ellistcomp(y,1)
          END IF
         END IF
        END DO
       END DO
       CALL combine(tempadd, failaddlist, elemno)

       DO x = 1, elemno
t = 1
        IF (failnode(x) .eq. 0) exit
        DO y = 1, elemno
         DO z = 2,11
          IF (ellistcomp(y,z) .eq. failnode(x)) THEN
           IF (ellistcomp(y,10) .eq. 2) THEN
            tempadd(t) = ellistcomp(y,1)
                  t = t + 1
           END IF
          END IF
         END DO
        END DO
        CALL combine(tempadd, failaddlist, elemno)
       END DO

       sumfailadd = SUM(failaddlist)

       IF (sumfailadd /= 0) THEN
        OPEN( 3, FILE = 'failtrip.dat', status='replace')
        WRITE(3, *) 1
        CLOSE(3)
        OPEN( 4, FILE = 'failaddelem.dat', status='replace')
        fw: DO count = 1, elemno
         IF (failaddlist(count) .eq. 0) exit fw
         WRITE(4, *) failaddlist(count)
        END DO fw
        CLOSE(4)
       ELSE
        OPEN( 3, FILE = 'failtrip.dat', status='replace')
        WRITE(3, *) 0
        CLOSE(3)
        OPEN( 4, FILE = 'failaddelem.dat', status='replace')
        CLOSE(4)
       END IF
```

```
      end program FailReplace_1

      !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      SUBROUTINE Combine(add, orig, no)
      implicit none

      INTEGER no
      INTEGER, DIMENSION(no) ::  add
      INTEGER, DIMENSION(no) ::  orig
      INTEGER, DIMENSION(no) ::  temp
      INTEGER count1
      INTEGER count2
      INTEGER w
      INTEGER z

      z = sum(orig)
      IF (z .eq. 0) THEN
       orig = add
END IF

      loop1: DO count1 = 1,no
       IF (add(count1) /= 0) THEN
        IF (add(count1) < orig(1)) THEN
         temp(1) = add(count1)
         DO w = 2, no
          temp(w) = orig(w-1)
         END DO
  orig = temp
       END IF
       loop2: DO count2 = 2,no
        IF (add(count1) .eq. orig(count2)) cycle loop1

        IF ((add(count1) < orig(count2)) .and. (add(count1) > orig(count2-1))) THEN

              temp(count2) = add(count1)
              DO w = 1, (count2 - 1)
               temp(w) = orig(w)
              END DO
              DO w = (count2 + 1), no
               temp(w) = orig(w - 1)
              END DO
              orig = temp
              cycle loop1
        END IF

        IF ((add(count1) > orig(count2-1)) .and. (orig(count2) .eq. 0)) THEN
```

```
                IF (orig(count2-1) .eq. 0) cycle loop1
                temp(count2) = add(count1)
                DO w = 1, (count2 - 1)
                 temp(w) = orig(w)
                END DO
                DO w = (count2 + 1), no
                 temp(w) = orig(w - 1)
                END DO
          orig = temp
                cycle loop1
          END IF

END DO loop2
       END IF
      END DO loop1

      END SUBROUTINE Combine
```

```
!*****************************************************************************
!
!  PROGRAM: FailFileAdd_1
!
!  PURPOSE:  Add elements back to model after failure
!
!*****************************************************************************

      program FailFileAdd_1
      USE DFLIB
      implicit none

      INTEGER, ALLOCATABLE, DIMENSION(:) :: elemaddlist
      INTEGER elemadd
      CHARACTER(len=150) :: line1
      CHARACTER(len=150) :: nline
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER check
      INTEGER z
      INTEGER y
      INTEGER nodes(6)
      INTEGER elno
      INTEGER elmat
      CHARACTER(len=10) :: tofind
      CHARACTER(len=6) :: ident
      CHARACTER(len=10) :: tofind2
      CHARACTER(len=6) :: ident2
      LOGICAL(4) result1
      CHARACTER(30) bdffile
      CHARACTER(30) bdfnewfile
      CHARACTER(100) line
      REAL time_begin, time_end
      REAL remlimit
      INTEGER elemno

      ! Body of FailFileAdd_1
      CALL CPU_TIME ( time_begin )

      OPEN( 2, FILE = 'failaddelem.dat')
      OPEN( 100, FILE = 'rundata.dat')

      READ(100, *) bdffile
      READ(100, *) bdfnewfile
```

```
      OPEN( 1, FILE = bdffile)
      OPEN( 5, FILE = bdfnewfile, status='replace')

      elemno = 0
      tofind = 'CTETRA'
      strlen= LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))

      DO WHILE (.TRUE.)
       READ(1, '(a)', END = 10) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
       IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
        elemno = elemno + 1
       END IF
      END DO

10 CLOSE(1)

      allocate(elemaddlist(elemno))

      elemaddlist = 0
      z = 1
      DO WHILE (.TRUE.)
       READ(2, 100, END=101) elemadd
100  FORMAT(T7, I7)
       IF (elemadd .eq. 0) cycle
       elemaddlist(z) = elemadd
       z = z + 1
      END DO

101 CLOSE(2)

      WRITE(line, 110) 'copy ', bdfnewfile, bdffile
110 FORMAT(A6, A24, A20)

      OPEN( 1, FILE = bdffile)

      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))
```

```
          DO WHILE (.TRUE.)
           READ(1, '(a)', END=300) line1
           indx = INDEX(line1, TRIM(tofind))
           indx2 = INDEX(line1, TRIM(tofind2))
           IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
            READ( line1, *) ident, elno, elmat, nodes(1), nodes(2), nodes(3), nodes(4),
nodes(5), nodes(6)
            READ(1, '(a)') nline
            !READ( nline, *) nodes(7), nodes(8), nodes(9), nodes(10)
            check = 0
            DO y = 1, elemno
             IF (elno .eq. elemaddlist(y)) THEN
              check = 1
      END IF
            END DO
            IF (check .eq. 1) THEN
       WRITE(5, 201) ident, elno, 1, nodes(1), nodes(2), nodes(3), nodes(4), nodes(5),
nodes(6)
201     FORMAT(A6, T9, I6, T18, I1, T26, I6, T34, I6, T42, I6, T50, I6, T58, I6, T66, I6)
             WRITE(5, 202) nline
202          FORMAT(A)
            ELSE
             WRITE(5, 203) line1
203          FORMAT(A)
             WRITE(5, 204) nline
204     FORMAT(A)
      END IF
          ELSE
           WRITE(5, 205) line1
205   FORMAT(A)
          END IF
         END DO
300 CLOSE(1)
         CLOSE(5)

         result1 = systemqq(line)

         CALL CPU_TIME ( time_end )
         PRINT *, 'Time for FailFileAdd was ', time_end - time_begin, ' seconds'

         end program FailFileAdd_1
```

```
!*****************************************************************************
!
!  PROGRAM: FileRem_1
!
!  PURPOSE:  Remove Elements from Model
!
!*****************************************************************************

      program FileRem_1

      USE DFLIB
      implicit none

! Variables
      INTEGER, ALLOCATABLE, DIMENSION(:) :: elemremlist
      INTEGER :: elemno
      INTEGER elemrem
      CHARACTER(len=150) :: line1
      CHARACTER(len=150) :: nline
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER y
      INTEGER z
      INTEGER check
      INTEGER nodes(6)
      INTEGER elno
      INTEGER elmat
      CHARACTER(len=10) :: tofind
      CHARACTER(len=6) :: ident
      CHARACTER(len=10) :: tofind2
      CHARACTER(len=6) :: ident2
      LOGICAL(4) result1
      CHARACTER(30) bdffile
      CHARACTER(30) bdfnewfile
      CHARACTER(100) line
      REAL time_begin, time_end


      ! Body of FileRem_1
      CALL CPU_TIME ( time_begin )

      OPEN( 2, FILE = 'remelem.dat')
      OPEN( 100, FILE = 'rundata.dat')
```

```
      READ(100, *) bdffile
      READ(100, *) bdfnewfile

      OPEN( 1, FILE = bdffile)
      OPEN( 5, FILE = bdfnewfile, status='replace')

      elemno = 1
      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))
      DO WHILE (.TRUE.)
       READ(1, '(a)', END=10) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
       IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
        elemno = elemno + 1
       END IF
      END DO

10  CLOSE(1)

      allocate(elemremlist(elemno))

      elemremlist = 0
      z = 1
      DO WHILE (.TRUE.)
       READ(2, 100, END=101) elemrem
100  FORMAT(T7, I7)
       IF (elemrem .eq. 0) cycle
       elemremlist(z) = elemrem
       z = z + 1
      END DO

101 CLOSE(2)

      OPEN( 1, FILE = bdffile)

      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))
      DO WHILE (.TRUE.)
       READ(1, '(a)', END=300) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
```

```
           IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
            READ( line1, *) ident, elno, elmat, nodes(1), nodes(2), nodes(3), nodes(4),
nodes(5), nodes(6)
            READ(1, '(a)') nline
            !READ( nline, *) nodes(7), nodes(8), nodes(9), nodes(10)
            check = 0
            DO y = 1, elemno
             IF (elemremlist(y) .eq. elno) THEN
                   check = 1
      END IF
            END DO
            IF (check .eq. 1) THEN
       WRITE(5, 201) ident, elno, 2, nodes(1), nodes(2), nodes(3), nodes(4), nodes(5),
nodes(6)
201     FORMAT(A6, T10, I6, T18, I1, T26, I6, T34, I6, T42, I6, T50, I6, T58, I6, T66,
I6)
             WRITE(5, 202) nline
202          FORMAT(A)
            ELSE
             WRITE(5, 203) line1
203          FORMAT(A)
             WRITE(5, 204) nline
204     FORMAT(A)
      END IF
          ELSE
           WRITE(5, 205) line1
205   FORMAT(A)
          END IF
         END DO
300 CLOSE(1)
         CLOSE(5)


!         WRITE(line, 310) 'copy ', bdfnewfile, bdffile
!310 FORMAT(A6, A24, A20)!
!         result1 = systemqq(line)

         CALL CPU_TIME ( time_end )
         PRINT *, 'Time for FileRem was ', time_end - time_begin, ' seconds'

         end program FileRem_1
```

```
!*****************************************************************************
!
!  PROGRAM: FileAdd_1
!
!  PURPOSE:  Add Elements to Model.
!
!*****************************************************************************

      program FileAdd_1
      USE DFLIB
      implicit none

      ! Variables
      INTEGER, ALLOCATABLE, DIMENSION(:) :: elemaddlist
      INTEGER :: elemno
      INTEGER elemadd
      CHARACTER(len=150) :: line1
      CHARACTER(len=150) :: nline
      INTEGER indx
      INTEGER strlen
      INTEGER indx2
      INTEGER strlen2
      INTEGER check
      INTEGER y
      INTEGER z
      INTEGER nodes(6)
      INTEGER elno
      INTEGER elmat
      CHARACTER(len=10) :: tofind
      CHARACTER(len=6) :: ident
      CHARACTER(len=10) :: tofind2
      CHARACTER(len=6) :: ident2
      LOGICAL(4) result1
      CHARACTER(30) bdffile
      CHARACTER(30) bdfnewfile
      REAL addlimit
      CHARACTER(100) line
      REAL time_begin, time_end

      ! Body of FileAdd_1
      CALL CPU_TIME ( time_begin )

      OPEN( 2, FILE = 'addelem.dat')
      OPEN( 100, FILE = 'rundata.dat')

      READ(100, *) bdffile
```

```
      READ(100, *) bdfnewfile

      OPEN( 1, FILE = bdffile)
      OPEN( 5, FILE = bdfnewfile, status='replace')

      elemno = 0
      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))
      DO WHILE (.TRUE.)
       READ(1, '(a)', END=10) line1
       indx = INDEX(line1, TRIM(tofind))
       indx2 = INDEX(line1, TRIM(tofind2))
       IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
        elemno = elemno + 1
       END IF
      END DO

10    CLOSE(1)

      allocate(elemaddlist(elemno))

      elemaddlist = 0
      z = 1
      DO WHILE (.TRUE.)
       READ(2, 100, END=101) elemadd
100  FORMAT(T5, I10)
   IF (elemadd .eq. 0) cycle
       elemaddlist(z) = elemadd
       z = z + 1
      END DO

101 CLOSE(2)

      WRITE(line, 110) 'copy ', bdfnewfile, bdffile
110 FORMAT(A6, A24, A20)

      OPEN( 1, FILE = bdffile)

      tofind = 'CTETRA'
      strlen = LEN(TRIM(tofind))
      tofind2 = 'CHEXA'
      strlen2 = LEN(TRIM(tofind2))
      DO WHILE (.TRUE.)
       READ(1, '(a)', END=300) line1
```

```
            indx = INDEX(line1, TRIM(tofind))
            indx2 = INDEX(line1, TRIM(tofind2))
            IF ((indx .eq. 1) .or. (indx2 .eq. 1)) THEN
             READ( line1, *) ident, elno, elmat, nodes(1), nodes(2), nodes(3), nodes(4),
nodes(5), nodes(6)
             READ(1, '(a)') nline
             !READ( nline, *) nodes(7), nodes(8), nodes(9), nodes(10)
             check = 0
             DO y = 1, elemno
              IF (elno .eq. elemaddlist(y)) THEN
               check = 1
      END IF
             END DO
             IF (check .eq. 1) THEN
               print*, elno
                    WRITE(5, 201) ident, elno, 1, nodes(1), nodes(2), nodes(3), nodes(4),
nodes(5), nodes(6)
201     FORMAT(A6, T9, I6, T18, I1, T26, I6, T34, I6, T42, I6, T50, I6, T58, I6, T66, I6)
              WRITE(5, 202) nline
202           FORMAT(A)
             ELSE
              WRITE(5, 203) line1
203           FORMAT(A)
              WRITE(5, 204) nline
204     FORMAT(A)
      END IF
           ELSE
            WRITE(5, 205) line1
205   FORMAT(A)
           END IF
          END DO
300 CLOSE(1)
          CLOSE(5)

          result1 = systemqq(line)
          CALL CPU_TIME ( time_end )
          PRINT *, 'Time for FileAdd was ', time_end - time_begin, ' seconds'
          end program FileAdd_1
```