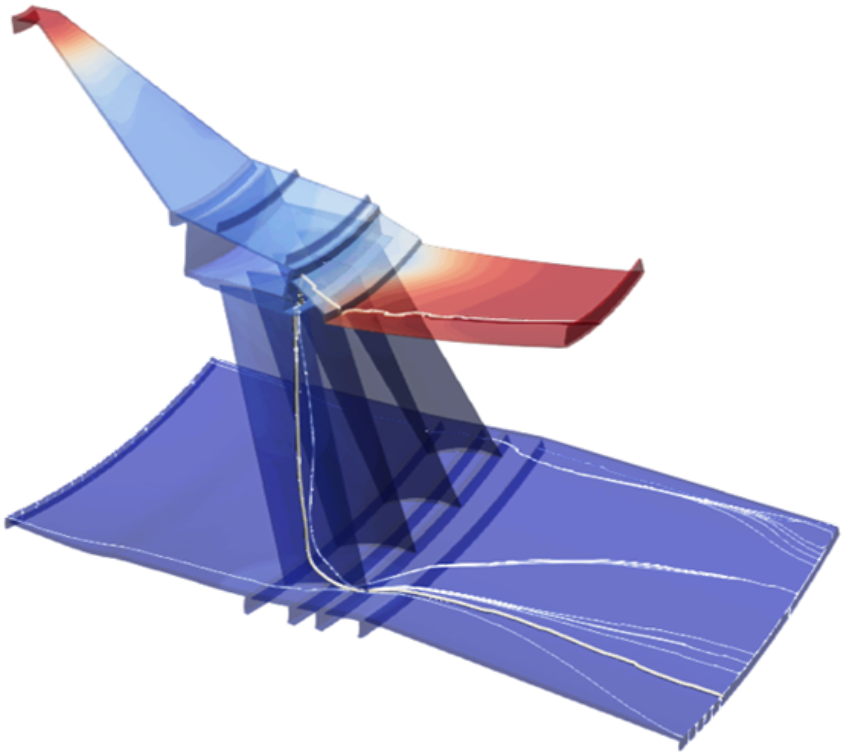


Load Path Visualization in Engine Structures

Master's thesis in Applied Mechanics

RAM JAYANATH AINIKKATTIL
PONKUMAR SANTHOSH ELANGO

DEPARTMENT OF INDUSTRIAL AND MATERIALS SCIENCE - IMSX30

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Load Path Visualization in Engine Structures

Master's thesis in Applied Mechanics, MSc (MPAME)
28 June 2023

Ram Jayanath Ainikkattil
ramj@chalmers.se

Ponkumar Santhosh Elango
ponkumar@chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Industrial and Material Science
Division of Material and Computational Mechanics
Chalmers University of Technology
Gothenburg, Sweden 2023

Load Path Visualization in Engine Structures

RAM JAYANATH AINIKKATTIL
PONKUMAR SANTHOSH ELANGO

Supervisor: Visakha Raja, GKN Aerospace Sweden AB
Examiner: Ragnar Larsson, Department of Industrial and Material Science, Chalmers University of Technology

Master's Thesis 2023
Department of Industrial and Materials Science
Division of Material and Computational Mechanics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Load path visualized on GKN EleFanT structure.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Load Path Visualization in Engine Structures

Ram Jayanath Ainikkattil, Ponkumar Santhosh Elango

Thursday 29$^{\text{th}}$ June, 2023

Abstract

From an engineering perspective, it is crucial to employ lightweight and cost-effective components in structures for optimal design, sustainability, and structural efficiency. The pursuit of such design has always been a challenge for mechanical engineers, and this is especially true in the case of the aerospace industry, where safety and function cannot be compromised. Though there are many ways to achieve this goal, the potential to use structural analysis and load path visualization for the same purpose is explored in the current work. The possibility of using the insights gained from the load path analysis to bring about changes in the structure for achieving desirable mechanical properties forms the objective of the current work. A relatively new concept of U^* index is used, which eliminates the complications involved in the conventional stress-based approach to predict the trajectory of load transfer through the structure. The thesis aims to improve previous research in the field by introducing distributed loading boundary conditions and principal load path extraction. The finite element calculations required for the U^* index are performed in ANSYS MAPDL using a script formulated specifically for the purpose. The post-processing to visualize the load paths through the structure is done in ParaView. Several case studies are done to visualize the load paths through 2D and 3D structures. Two design criteria, namely the uniformity and continuity curves, are studied for the principal load paths in each of these cases. The work also aims to investigate the effectiveness of the U^* index method in predicting the transfer of loads through structures when boundary conditions of the domain are altered. The challenges and restrictions that come with using the U^* index approach are also discussed in detail, thus paving the way for future study and research in the field.

Keywords: U^* index, Load path visualization, Principal load path, ANSYS, ParaView.

Acknowledgement

We would like to express our heartfelt gratitude to everyone who has contributed to the successful completion of this master thesis work. Without the support and encouragement of everyone involved, this work would have remained incomplete.

We would like to thank Dr Visakha Raja for allowing us the opportunity to work on this exciting assignment and whose guidance and mentorship have been the significant factor in completing the work on time. His unwavering positive energy has helped us tackle the challenges faced during the thesis work.

We also want to express our sincere gratitude to our examiner, Dr Ragnar Larsson, for the consistent suggestions and technical inputs, without which this work would not have been a success. Executing this thesis under his supervision has been a great learning experience.

We want to thank Jonathan Muistama and Rajesh Ramesh, whose works provided the platform upon which the current thesis was built. Their support and guidance have been indispensable throughout the execution of this thesis.

Finally, we would like to thank our family and friends for being our constant support and anchor throughout every phase of the thesis work. Their love and encouragement have been our source of strength throughout the course of the thesis work.

Ram Jayanath Ainikkattil
Ponkumar Santhosh Elango

List of Figures

List of Tables

List of Acronyms

2D Two Dimensional

3D Three Dimensional

APDL Ansys Parametric Design Language

FEM Finite Element Method

MAPDL Mechanical Ansys Parametric Design Language

VTK Visualization Toolkit

Table of Contents

1 Introduction

This section provides background information for the thesis work, an introduction to the U^* index concept, previous works in this field by GKN, the general objectives and aims of the current work, its limitations, and the ethical considerations taken into account for this project.

1.1 Project Background

Aerospace engineers have long been tasked with the difficult challenge of crafting lightweight, cost-effective components that do not compromise on strength or safety. Though material selection is undoubtedly a critical factor in determining mechanical properties, the expenses associated with design and manufacturing can be exorbitant. As such, structural analysis plays a critical role in achieving the desired outcome. By gaining valuable insights from this analytical process, engineers can make informed decisions regarding design modifications during product development, ultimately striking a delicate balance between cost and function.

In structural design, it is crucial to identify how the loads imposed on the structure are transferred to the supports because this could provide information critical for the design and optimization of the structure. Load path analysis provides valuable insights into the structural behaviour of a component when subjected to static loading conditions, such as critical areas of load transfer, the stiffest regions in the structure and the scope for design optimization. Therefore, it can be regarded as a technique for assessing the strength of a structure[1]. Since the path taken by the load through the structure is not readily apparent [2], methods to visualize the load paths are needed. There have been numerous studies done to visualize load paths. Tamijani et al. [3] proposed load function method to visualize load paths. The conventional methods used to identify the load paths are based on the stress induced in the structure when loaded under static conditions. While this approach helps in understanding the internal stiffness distribution in the structure, the presence of stress concentration in the structure affects the load paths and could thus be misleading. This necessitates a load path visualization technique that circumvents this problem. The inception of the concept of using a non-dimensional index that represents the internal stiffness distribution of a structure was introduced by Shinobu et al. [4].

The earliest work on load transfer was done by Shinobu et al. [4] in 1995, wherein a non-dimensional parameter E^* was proposed to express the degree of connectivity between the loading point and any arbitrary point in the structure. Later on, in 2003, Hoshino et al. [5] proposed a new parameter U^* to find load paths in mechanical engineering components. The concept of U^* has been modified by researchers to expand its application to orthotropic materials [6], nonlinear elastic materials [7] and to elements having rotational degrees of freedom [6] [8]. Further, the U^* index method also finds application in vibration control problems [5] [9].

Various studies have been done by researches to highlight the important applications of load path in engineering design, the work done by Zhanguang et al. [10] predicts the multi-axial fatigue life of metal with the help of load paths coupled with principles of machine learning and image recognition. Another work done by Qing-Yun et al. [11] also employs the sensitivity of load path in predicting multi axial fatigue life of metals. Load paths have been useful in

design and optimization based on study done by Shengjie et al. [12], in this study functionally graded lattice structures are optimized using reinforcements predicted based on load paths.

The U^* index method has been used in various applications over the past two decades. Wang et al. [9] adopted the method to demonstrate the load transfer in truck cab structure during frontal collision and Tadashi et al [13] made use of U^* index method to show the load transfer through structures joined together by spot welds.

The U^* index is a normalized quantity used to represent the internal stiffness distribution in a system. The calculation of U^* index is done through the Finite Element Method, which divides the elastic continuum into a finite number of elements. The U^* values are then calculated for each node in the system, and the streamlines through the U^* contours are obtained from the corresponding gradients. By following the streamline with the highest gradient of U^*, the stiffest path through the structure can be identified, also referred to as the principal load path. This is based on the assumption that the stiffest path in the structure carries the highest load.The concept of principal load path could serve as a means to enhance the structural stiffness of structures without using complex methods such as topology optimization [14].

1.2 Related Works

The master thesis work at GKN Aerospace AB by Rajesh [15] successfully developed a program to calculate the U^* index using ANSYS Mechanical APDL [16], the commercial CAE software employed by GKN Aerospace. The load path visualization was done using the opensource software ParaView. However, the computation time was deemed high for the practical viability of the approach. This drawback was overcome in the paper presented by Jonathan and Oscar. [17] by incorporating the inspection loading method proposed by Sakurai et al. [18]. This approach showed a significant reduction in the computation time, thus making it possible to adopt it practically into a commercial environment. Despite the reduced computation time, the applicability of the U^* index was limited to point loading. Therefore, the load paths due to distributed boundary loading could not be visualized using the U^* method.

1.3 Aim and Ojectives

This project aims to build upon the research in load path visualization using the U^* index method, conducted by Ramesh [15] and Jonathan et al. [17] at GKN Aerospace AB. The current work is aimed at achieving a number of specific tasks as listed:

1. Calculate the U^* index for structures subjected to distributed loading boundary conditions using the method proposed by Sakurai et al. [19] and develop a script in ANSYS Mechanical APDL to perform the necessary finite element calculations.

2. Using the APDL script, perform load path analysis on simple 2D and 3D structures. Conduct multiple case studies of load transfer on these structures by applying different boundary conditions.

3. Develop a post-processing routine for the load path visualization in ParaView and a Python script to extract the principal load path, which is considered the stiffest path through the structure domain.

4. Study two design parameters, uniformity and continuity, of the principal load paths in each case study and discuss the findings.

5. Finally, the study is extended to a complex engine structure geometry to visualize the load paths and study how changing the boundary conditions altre the load transfer path. Extract uniformity and continuity curves for the load paths.

1.4 Limitations

1. The U^* index theory has a major limitation - it can only be used for static structural analysis, and the structures must be analyzed within the elastic domain of the material, meaning that Hooke's Law must be applicable. These drawbacks also restrict the current thesis.

2. The materials evaluated in this thesis work are linear isotropic materials. Material orthotropy or anisotropy is not considered in the current thesis work.

3. The structures analyzed in this thesis are modelled using finite elements with only translational degrees of freedom. The rotational degrees of freedom, which are helpful for analyzing thin-walled structures, are not considered.

4. In this study, the 3D models are meshed using first-order tetrahedral elements, a computationally expensive process resulting in longer solution times.

5. The research findings presented through finite element analysis have not been experimentally validated. Therefore, to apply these findings in a real-world environment, they must be validated experimentally.

1.5 Ethical Considerations

1. Extensive research for this project was conducted using published papers, research works, and reference books to guarantee the utmost precision and openness.

2. Care has been taken to give credit for the ideas and works of other researchers using proper citations and by attaining the relevant permissions to use copyright material and proprietary software used at GKN.

3. The current thesis work needs to be experimentally validated. The technological readiness of the work is considered low. Any adaptation of the work into a real-world scenario must be performed with due consideration.

2 Theory

This section discusses the theoretical concepts that serve as the foundation of load path calculation and visualization. The fundamental concepts of finite element method, strain energy, U index and load streamlines, helpful for the study, are described and illustrated in detail.*

2.1 Finite Element Method

The study of the physical world around us is constantly done by scientists and engineers alike. Mathematical models of the physical world are constructed using differential equations [20], and these models represent the physical system to varying degrees of accuracy. Engineers aim to solve such mathematical models to obtain valuable insights into the functions of the physical world. In solid mechanics, the study of the mechanical behaviour of engineering systems plays an important role. The insight gained from such studies helps to improve the design, predict failure, improve safety and reduce cost. These differential equations are challenging or often impossible to solve by analytical approaches. This necessitates the use of numerical techniques that will yield approximate solutions to those differential equations but at the same time provide a close enough prediction from an engineering standpoint. One among those various numerical techniques is the finite element method, which has been widely adapted for structural analysis. The Finite Element Method solves the differential equation in its finite dimensional weak form, allowing the system to be discretized into smaller, simpler subdomains called elements, over which the solution is piece-wise continuous. This approach reduces the complexity of the solution process. The weak form to the linear elasticity problem is given by equation 2.1.1.

Find $u \in \mathbb{U}$ such that

$$a(\boldsymbol{u}, \delta\boldsymbol{u}) = l(\delta\boldsymbol{u}) \quad \forall \delta\boldsymbol{u} \in \mathbb{U}^0 \tag{2.1.1}$$

where

$$\begin{aligned} a(\boldsymbol{v}, \boldsymbol{w}) &= \int_\Omega \boldsymbol{\epsilon}[\boldsymbol{w}] : \mathbf{E} : \boldsymbol{\epsilon}[\boldsymbol{v}] \mathrm{d}\Omega \\ l(\boldsymbol{v}) &= \int_\Omega \boldsymbol{v} \cdot \boldsymbol{b} \, \mathrm{d}\Omega + \int_{\Gamma_\mathrm{N}} \boldsymbol{v} \cdot \boldsymbol{t}_\mathrm{p} \mathrm{d}\Gamma \\ \mathbb{U} &= \{\boldsymbol{v} \text{ sufficiently regular, } \boldsymbol{v} = \boldsymbol{u}_\mathrm{D} \text{ on } \Gamma_\mathrm{D}\} \\ \mathbb{U}^0 &= \{\boldsymbol{v} \text{ sufficiently regular, } \boldsymbol{v} = \mathbf{0} \text{ on } \Gamma_\mathrm{D}\} \end{aligned}$$

2.2 Strain Energy

The concept used in the computation of the U^* index is strain energy. When an elastic body is deformed, work done on the material is stored as potential energy. This potential energy of the system is called strain energy. The strain energy is the direct consequence of the stiffness of the system. The ability of the system to resist deformation and revert to the original configuration of stable equilibrium enables it to do work in the form of stored strain energy.

The amount of strain energy stored within a system depends on the material's elastic properties, the magnitude of deformation, and the system's geometry. For a Finite Element dis-

cretized system, the strain energy can be calculated using the equation 2.2.1.

$$U = \frac{1}{2}\boldsymbol{d}^T \boldsymbol{K} \boldsymbol{d} \tag{2.2.1}$$

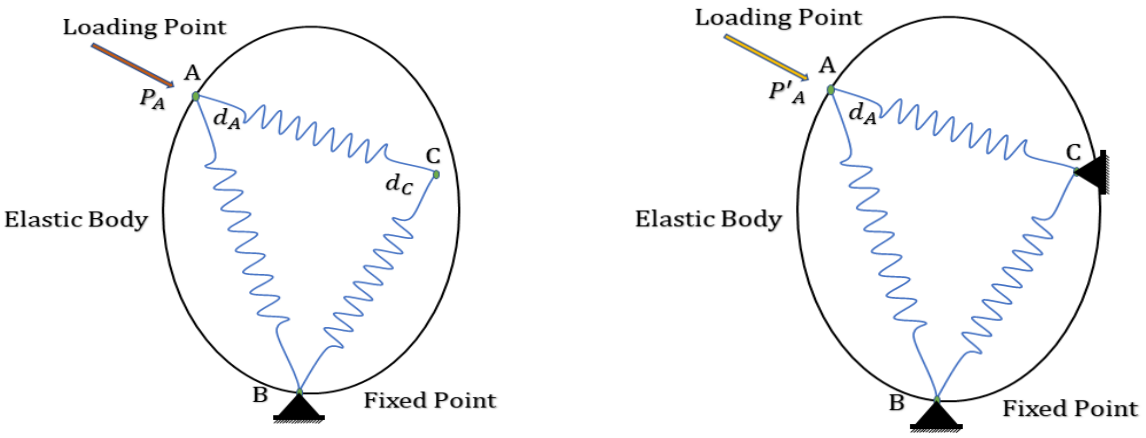
where U is the strain energy, $\boldsymbol{K}$ is the stiffness matrix, and $\boldsymbol{d}$ is the nodal displacement for the system. In engineering, strain energy is a crucial factor in designing of structures. The area under the material's stress-strain curve up to the deformation point denotes the strain energy. The U^* index is calculated using the concept that a stiffer system develops more strain energy than a less stiff system when subjected to similar deformation.

2.3 U^* Index and its Calculation

The visualization of the load path performed in this thesis is based on the theory of U^* index. The approach used in the computation of U^* index is described in this section.The U^* index is a mathematical index used to represent the internal stiffness distribution of a structure under the applied loads and constraints. It is a non-dimensional value ranging from 0 to 1. U^* index values are not unique to a particular structure or component, as its value depends on the applied loads and boundary conditions. Computation of the U^* index is performed for each node in the system. Based on these values, the U^* index contours are plotted across the domain. These contours help to understand how the streamlines representing the load paths pass through the mechanical system. The computation of the U^* index has been done in two different approaches, namely the direct approach and the inspection loading approach.

2.3.1 Direct Method

The direct method for computation of U^* index is one of the more straightforward approaches. Even though this method is computationally expensive, the method provides an overview of the theoretical background that could help in understanding how this technique could be used in visualizing the stiffness distribution in a system.



(a) Original Configuration (b) Modified Configuration

Figure 2.1: Direct method calculation of U^*

Consider the elastic body shown in Figure 2.1a. Points A and B represent the point of loading and the point of fixed support in the structure respectively. The U^* index is to be computed at every node in the system that is not A or B. These nodes referred to in this work as points/nodes of interest. $\boldsymbol{P}_A$ denotes the load applied to point A and $\boldsymbol{d}_A$ denotes the displacement at point A due to the applied load. The stiffness between these points are represented by linear elastic springs that connect them. This linear system is then solved to find the displacements $\boldsymbol{d}_A$ and $\boldsymbol{d}_C$ of points A and C, respectively. The strain energy of this system can be calculated using the following equation.

$$U = \frac{1}{2}\boldsymbol{P}\boldsymbol{d} \tag{2.3.1}$$

$\boldsymbol{P}$ and $\boldsymbol{d}$ are the applied mechanical load and the resulting nodal displacement, respectively. The analysis in this thesis is done within the material's elastic limit and for linear kinematics. Hence, a linear stress-strain relation and linear strain-displacement relation give rise to a linear Finite Element system of equations as follows. The applied load $\boldsymbol{P}$ can be calculated using the equation 2.3.2.

$$\boldsymbol{P} = \boldsymbol{K}\boldsymbol{d} \tag{2.3.2}$$

Where $\boldsymbol{K}$ is the stiffness and $\boldsymbol{d}$ is the nodal displacement of the system. Since we are interested in the stiffness between the points A, B and C, the equation 2.3.2 can be condensed to the degrees of freedom in nodes A, B and C as follows.

$$\begin{bmatrix} \boldsymbol{P}_A \\ \boldsymbol{P}_B \\ \boldsymbol{P}_C \end{bmatrix} = \begin{bmatrix} \boldsymbol{K}_{AA} & \boldsymbol{K}_{AB} & \boldsymbol{K}_{AC} \\ \boldsymbol{K}_{BA} & \boldsymbol{K}_{BB} & \boldsymbol{K}_{BC} \\ \boldsymbol{K}_{CA} & \boldsymbol{K}_{CB} & \boldsymbol{K}_{CC} \end{bmatrix} \begin{bmatrix} \boldsymbol{d}_A \\ \boldsymbol{d}_B \\ \boldsymbol{d}_C \end{bmatrix} \tag{2.3.3}$$

For the original system in Figure 2.1a, the load P_A applied at point A can be expressed as:

$$\boldsymbol{P}_A = \boldsymbol{K}_{AA}\boldsymbol{d}_A + \boldsymbol{K}_{AB}\boldsymbol{d}_B + \boldsymbol{K}_{AC}\boldsymbol{d}_C \tag{2.3.4}$$

Applying the boundary condition i.e, $\boldsymbol{d}_B = 0$, the equation 2.3.4 is further simplified as:

$$\boldsymbol{P}_A = \boldsymbol{K}_{AA}\boldsymbol{d}_A + \boldsymbol{K}_{AC}\boldsymbol{d}_C \tag{2.3.5}$$

The strain energy, U, of the original system can be calculated as shown in equation 2.3.6.

$$U = \frac{1}{2}\boldsymbol{P}_A\boldsymbol{d}_A = \frac{1}{2}(\boldsymbol{K}_{AA}\boldsymbol{d}_A + \boldsymbol{K}_{AC}\boldsymbol{d}_C)\boldsymbol{d}_A \tag{2.3.6}$$

Consider the modified system shown in Figure 2.1b. The displacement $\boldsymbol{d}_A$ obtained from the original system is applied to point A. Point C fixed. The strain energy of this modified system can be calculated using the equation 2.3.7.

$$U' = \frac{1}{2}\boldsymbol{P}'_A\boldsymbol{d}_A \tag{2.3.7}$$

From equation 2.3.3, the strain energy of the modified system can be written as:

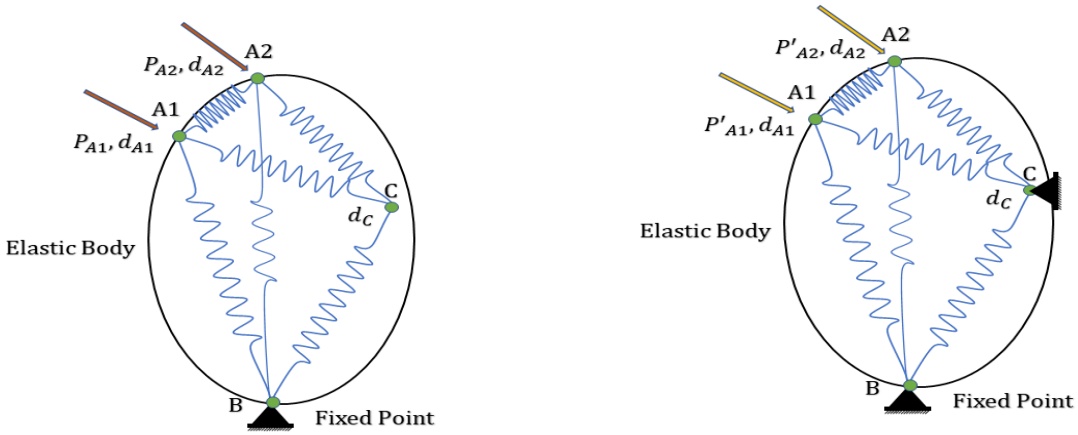
$$U' = \frac{1}{2}(\boldsymbol{K}_{AA}\boldsymbol{d}_A)\boldsymbol{d}_A \tag{2.3.8}$$

The of U^* index for the point C can then be computed using the equation 2.3.9.

$$U^* = 1 - \frac{U}{U'} = 1 - \frac{(\boldsymbol{K}_{AA}\boldsymbol{d}_A + \boldsymbol{K}_{AC}\boldsymbol{d}_C)\boldsymbol{d}_A}{(\boldsymbol{K}_{AA}\boldsymbol{d}_A)\boldsymbol{d}_A} \tag{2.3.9}$$

With the help of the finite element method, the U^* index values are calculated for each node present in the system. This index computed using the strain energy provides insight into the internal stiffness distribution of the system.

2.3.2 Direct Method - Multiple Loading



(a) Original Configuration

(b) Modified Configuration

Figure 2.2: Direct method calculation of U^* for multiple loading points.

The method for computing the U* index for multiple loading points and distributed loading is described in this section [19]. The computation of the U* index for a structure loaded at multiple points and with distributed loads can be derived from the equations presented in the section for direct method computation of the U* index.

Consider the elastic body shown in Figure 2.2a. Loads PA1 and PA2 are applied on points A1 and A2, respectively. Since the analysis is done within the elastic limit, the force and displacement relation of the system in matrix form can be written as shown in equation 2.3.10.

$$\begin{Bmatrix} \boldsymbol{p}_{A1} \\ \boldsymbol{p}_{A2} \\ \boldsymbol{p}_B \\ \boldsymbol{p}_C \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}_{A1A1} & \boldsymbol{K}_{A1A2} & \boldsymbol{K}_{A1B} & \boldsymbol{K}_{A1C} \\ \boldsymbol{K}_{A2A1} & \boldsymbol{K}_{A2A2} & \boldsymbol{K}_{A2B} & \boldsymbol{K}_{A2C} \\ \boldsymbol{K}_{BA1} & \boldsymbol{K}_{BA2} & \boldsymbol{K}_{BB} & \boldsymbol{K}_{BC} \\ \boldsymbol{K}_{CA1} & \boldsymbol{K}_{CA2} & \boldsymbol{K}_{CB} & \boldsymbol{K}_{CC} \end{bmatrix} \begin{Bmatrix} \boldsymbol{d}_{A1} \\ \boldsymbol{d}_{A2} \\ \boldsymbol{d}_B \\ \boldsymbol{d}_C \end{Bmatrix} \tag{2.3.10}$$

The work done by the applied loads is stored as the strain energy of the system. This strain energy, U, can be calculated using the equation 2.3.11.

$$\begin{aligned} U =& \frac{1}{2}\boldsymbol{p}_{A1} \cdot \boldsymbol{d}_{A1} + \frac{1}{2}\boldsymbol{p}_{A2} \cdot \boldsymbol{d}_{A2} \\ =& \frac{1}{2}\left\{(\boldsymbol{K}_{A1A1}\boldsymbol{d}_{A1} + \boldsymbol{K}_{A1A2}\boldsymbol{d}_{A2} + \boldsymbol{K}_{A1C}\boldsymbol{d}_C) \cdot \boldsymbol{d}_{A1}\right. \\ & \left. + (\boldsymbol{K}_{A2A1}\boldsymbol{d}_{A1} + \boldsymbol{K}_{A2A2}\boldsymbol{d}_{A2} + \boldsymbol{K}_{A2C}\boldsymbol{d}_C) \cdot \boldsymbol{d}_{A2}\right\} \end{aligned} \tag{2.3.11}$$

Now consider the modified system shown in Figure 2.2b. Point C is constrained, and the nodal displacements $dA1$ and $dA2$ obtained from the original system are applied to points $A1$ and $A2$, respectively. The constrained point C results in a higher energy requirement to achieve the same displacements at points $A1$ and $A2$ compared to the original system. This strain energy of the modified system, U', is computed using the equation 2.3.12

$$\begin{aligned} U' =& \frac{1}{2}\boldsymbol{p}'_{A1} \cdot \boldsymbol{d}_{A1} + \frac{1}{2}\boldsymbol{p}'_{A2} \cdot \boldsymbol{d}_{A2} \\ =& \frac{1}{2}\left\{(\boldsymbol{K}_{A1A1}\boldsymbol{d}_{A1} + \boldsymbol{K}_{A1A2}\boldsymbol{d}_{A2}) \cdot \boldsymbol{d}_{A1}\right. \\ & \left. + (\boldsymbol{K}_{A2A1}\boldsymbol{d}_{A1} + \boldsymbol{K}_{A2A2}\boldsymbol{d}_{A2}) \cdot \boldsymbol{d}_{A2}\right\} \end{aligned} \tag{2.3.12}$$

From equations 2.3.11 and 2.3.12, the relation between the strain energies of the original and the modified system can be written as shown in equation 2.3.13.

$$U' = U - \frac{1}{2}\left\{(\boldsymbol{K}_{A1C}\boldsymbol{d}_C) \cdot \boldsymbol{d}_{A1} + (\boldsymbol{K}_{A2C}\boldsymbol{d}_C) \cdot \boldsymbol{d}_{A2}\right\} \tag{2.3.13}$$

The U^* index of point C is calculated as shown in Equation 2.3.14. This value represents the stiffness of the path between the load application points and point C.

$$\begin{aligned} U^* &= 1 - \left(\frac{U'}{U}\right)^{-1} \\ &= \left(1 - \frac{2U}{(\boldsymbol{K}_{A1C}\boldsymbol{d}_C) \cdot \boldsymbol{d}_{A1} + (\boldsymbol{K}_{A2C}\boldsymbol{d}_C) \cdot \boldsymbol{d}_{A2}}\right)^{-1} \end{aligned} \tag{2.3.14}$$

Equations 2.3.10 to 2.3.14 provide a way to calculate the U^* index for multiple loading points. This method can be expanded to include distributed loading conditions.
Consider an elastic body which is loaded at n points. The force-displacement relation for the system can be written as shown in Equation 2.3.15.

$$\begin{Bmatrix} \boldsymbol{p}_{A1} \\ \boldsymbol{p}_{A2} \\ \vdots \\ \boldsymbol{p}_{An} \\ \boldsymbol{p}_B \\ \boldsymbol{p}_C \end{Bmatrix} = \begin{bmatrix} \boldsymbol{K}_{A1A1} & \boldsymbol{K}_{A1A2} & \cdots & \boldsymbol{K}_{A1An} & \boldsymbol{K}_{A1B} & \boldsymbol{K}_{A1C} \\ \boldsymbol{K}_{A2A1} & \boldsymbol{K}_{A2A2} & \cdots & \boldsymbol{K}_{A2An} & \boldsymbol{K}_{A2B} & \boldsymbol{K}_{A2C} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \boldsymbol{K}_{AnA1} & \boldsymbol{K}_{AnA2} & \cdots & \boldsymbol{K}_{AnAn} & \boldsymbol{K}_{AnB} & \boldsymbol{K}_{AnC} \\ \boldsymbol{K}_{BA1} & \boldsymbol{K}_{BA2} & \cdots & \boldsymbol{K}_{BAn} & \boldsymbol{K}_{BB} & \boldsymbol{K}_{BC} \\ \boldsymbol{K}_{CA1} & \boldsymbol{K}_{CA2} & \cdots & \boldsymbol{K}_{CAn} & \boldsymbol{K}_{CB} & \boldsymbol{K}_{CC} \end{bmatrix} \begin{Bmatrix} \boldsymbol{d}_{A1} \\ \boldsymbol{d}_{A2} \\ \vdots \\ \boldsymbol{d}_{An} \\ \boldsymbol{d}_B \\ \boldsymbol{d}_C \end{Bmatrix} \tag{2.3.15}$$

Equation 2.3.16 shows how the strain energy of the system is computed when it is loaded at multiple points. In this equation, the indices j and i refer to the row and column of the global stiffness matrix.

$$
\begin{aligned}
U &= \frac{1}{2} p_{A1} \cdot \boldsymbol{d}_{A1} + \frac{1}{2} p_{A2} \cdot \boldsymbol{d}_{A2} + \cdots + \frac{1}{2} \boldsymbol{p}_{An} \cdot \boldsymbol{d}_{An} \\
&= \frac{1}{2} \sum_{j=1}^{n} \left\{ \sum_{i=1}^{n} \left(\boldsymbol{K}_{AjAi} \boldsymbol{d}_{Ai} + \boldsymbol{K}_{AjC} \boldsymbol{d}_{C} \right) \cdot \boldsymbol{d}_{Aj} \right\}
\end{aligned}
\tag{2.3.16}
$$

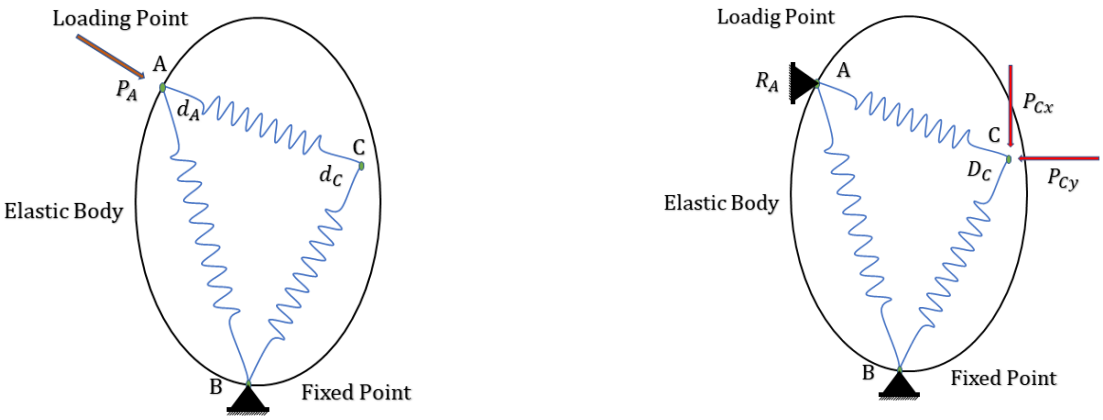
Equation 2.3.17 shows how the modified system's strain energy is computed when loaded at multiple points. In this equation, the indices j and i refer to the row and column of the global stiffness matrix. In the modified system, point C is constrained in all directions.

$$
\begin{aligned}
U' &= \frac{1}{2} p'_{A1} \cdot d_{A1} + \frac{1}{2} p_{A2} \cdot d_{A2} + \cdots + \frac{1}{2} p_{An'} \cdot \boldsymbol{d}_{An} \\
&= \frac{1}{2} \sum_{j=1}^{n} \left\{ \sum_{i=1}^{n} \left(\boldsymbol{K}_{AjAi} \cdot \boldsymbol{d}_{Ai} \right) \cdot \boldsymbol{d}_{Aj} \right\}
\end{aligned}
\tag{2.3.17}
$$

To determine the U^* index value for point C in the system, equation 2.3.18 is utilized. The index i represents the number of loading points in the original system. This equation is used to compute the U^* index for distributed loads. A finite element routine is developed using an APDL script to calculate the U^* index values for each point in the analysis domain.

$$
U^* = \left(1 - \frac{2U}{\sum_{i=1}^{n} \left(\boldsymbol{K}_{AiC} \boldsymbol{d}_{C} \right) \cdot \boldsymbol{d}_{Ai}} \right)^{-1}
\tag{2.3.18}
$$

2.3.3 Inspection Loading Method



(a) Original Configuration

(b) Modified Configuration

Figure 2.3: Inspection loading method calculation of U^*

The inspection loading method for computing the U* index is described in this section. Sakurai et al. [18] devised this alternative method for computing the U* index as a means to resolve the issue of high computational time associated with the direct method.

The inspection loading method is a variation of the direct method wherein the computation of U^* index does not require a change of displacement boundary conditions for every desired

node. Instead, the loading points in the original load case are fixed and linearly independent trial/inspection loads are applied on the node of interest. Thus, only the force boundary conditions are changed every time and not the displacement boundary conditions. Commercial Finite Element software like ANSYS APDL do not recompute the global stiffness matrix unless the geometric/displacement boundary conditions are changed. Thus the computation time is significantly reduced for a very large finite element system. Moreover, the condensed stiffness matrices connecting the loading points and the points of interest do not change for a particular structure, and can therefore be saved and reused for different loading conditions.

Consider the elastic body shown in Figure 2.3a. The strain energy of the system when a load is applied to point A is computed similarly using equation 2.3.6 as in the direct method. The U* index for the system can be computed using Equation 2.3.19.

$$U^* = 1 - \frac{U}{U'} = 1 - \frac{(K_{AA}d_A + K_{AC}d_C)}{(K_{AA}d_A)} = (1 - \frac{2U}{(K_{AC}d_C)d_A})^{-1} \tag{2.3.19}$$

where U represents the strain energy of the original system, and d_c and d_a are the displacements of points A and C, respectively, when the load P_A is applied to the original system. The term K_{AC} denotes the stiffness between points A and C. This stiffness can be calculated with the help of inspection loads.

Consider the modified system shown in Figure 2.3b. Point A is constrained in all degrees of freedom. Two inspection loads, P_{CX} and P_{CY}, are applied at point C. Equation 2.3.20 can be used to express the stiffness K_{AC} based on the displacements at point C and the reaction forces generated at point A.

$$[\boldsymbol{R_A}] = [\boldsymbol{K_{AC}}][\boldsymbol{D_C}] \tag{2.3.20}$$

where R_A is a vector containing the reaction forces generated at point A due to the inspection loads, and D_C is a vector containing the corresponding displacements of point C. Each column in these vectors represents the x, y and z components of the reaction forces and displacements.

Two sets of inspection loads are used if a system has nodes with two degrees of freedom. For a 3D structure with nodes having three degrees of freedom, three sets of inspection loads are applied. Equations 2.3.21 and 2.3.22 display the arrangement of the reaction vector R_A and displacement vector D_C for a 3D body with three degrees of freedom per node.

$$[\boldsymbol{R_A}] = \begin{bmatrix} \boldsymbol{r_{A1}} & \boldsymbol{r_{A2}} & \boldsymbol{r_{A3}} \end{bmatrix} \tag{2.3.21}$$

$$[\boldsymbol{D_C}] = \begin{bmatrix} \boldsymbol{d_{C1}} & \boldsymbol{d_{C2}} & \boldsymbol{d_{C3}} \end{bmatrix} \tag{2.3.22}$$

where $\boldsymbol{r_A}$ and $\boldsymbol{d_C}$ are column vectors with a size of 3x1, the finite element problem is solved to determine displacement and reaction force values, which are then used to calculate the stiffness value, K_{AC}, using equation 2.3.23.

$$[\boldsymbol{K_{AC}}] = [\boldsymbol{R_A}][\boldsymbol{D_C}]^{-1} \tag{2.3.23}$$

For each node in the system, the computation is performed to extract the U^* distribution in the structure. The inspection method involves more calculations compared to the direct

method. If a system has two degrees of freedom per node, the number of calculations required is twice as much, and for three degrees of freedom per node, it is three times as much. However, the inspection loading method does not need to recompute the global stiffness matrix in every load step. The FEM software can reuse the initially computed global stiffness matrix for all subsequent calculations, which significantly reduces the overall computation time. This time reduction is especially evident for complex systems with a large number of elements and nodes.

2.4 Load Path Visualization using U^* index

2.4.1 U^* Gradient and Streamlines

In mathematics, gradient of a scalar function represents the direction and rate of fastest increase of that function. Essentially, it is a vector that indicates the direction of the function's steepest increase at a given point.

For instance, consider a scalar valued function $f(x, y)$ defined over a domain Ω in two-dimensional space. The gradient of f at an arbitrary point $(x_1, y_1) \in \Omega$ is a two-dimensional vector $(\partial f/\partial x, \partial f/\partial y)$, where $\partial f/\partial x$ and $\partial f/\partial y$ are the partial derivatives of f with respect to x and y respectively, evaluated at (x_1, y_1).

The gradient is an important concept in many areas of mathematics, physics, and engineering. It is used to solve optimization problems, to calculate strains, velocities, heat flux and so on in physical systems, and to analyze the behaviour of functions and equations. In structural engineering, the concept of gradients plays a vital role in determining load paths through the given structure. The overall load is carried through the path with the highest stiffness values, Shinobu et al. [4], which is a critical factor in designing and constructing any structural system. One way to assess the internal stiffness distribution of a structure is through the use of the U^* index, which is used to identify the path with the highest U^* index values and consequently, the stiffest path. To accurately express the load path, a decay vector is introduced by Wang et al, [21] in equation 2.4.1 to trace the points with the highest stiffness values, providing a clear picture of the structural forces at play.

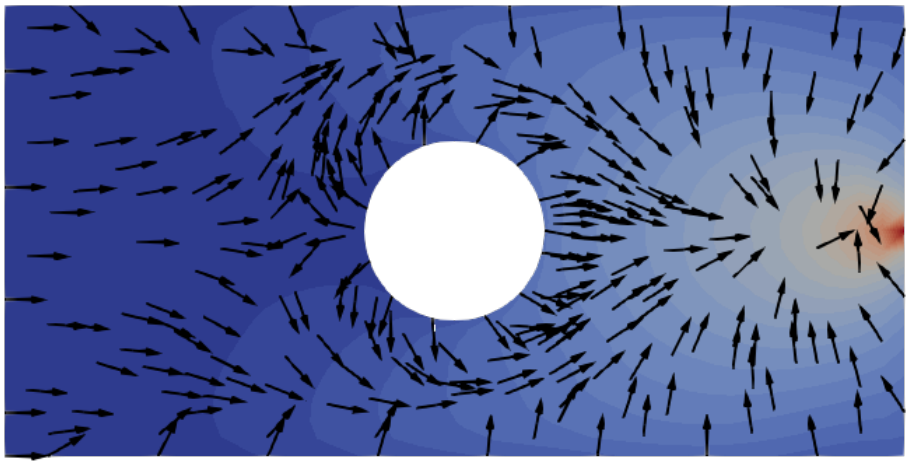


Figure 2.4: U^* contour & gradient field on plate with a hole

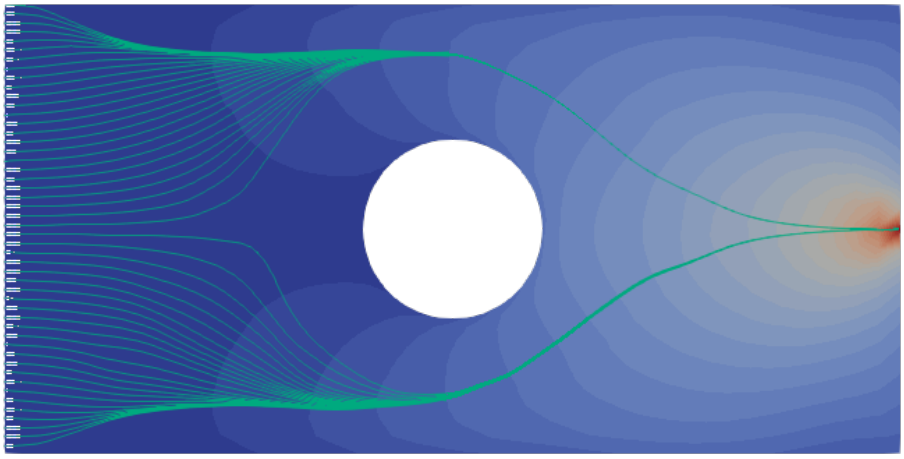


Figure 2.5: Streamlines through plate with a hole structure

$$\lambda = -grad(U^*) \tag{2.4.1}$$

Figure 2.4 represents the U^* index contour and the corresponding gradient vector field for a plate with hole structure subject to fixed boundary condition on the left edge and point loading condition on the right edge. The orientation of the black arrows denote the direction of highest change in U^* index values.

The field of fluid mechanics involves the study of fluids in motion, and one of the fundamental concepts in this field is that of streamlines. Streamlines are imaginary lines that are used to depict the movement of fluid particles in a flow field. They are an essential tool for visualizing fluid flow behaviour, and they are defined as lines that are tangent to the velocity vector of the fluid at every point in the flow. In other words, streamlines are lines that run parallel to the direction of fluid flow at all times.

The concept of streamlines could be applied to the current work to represent the flow of load through the structure. The streamlines are plotted for the decay vector field computed using the equation 2.4.1. They are obtained by tangentially connecting the decay vectors starting from the seeding points. Figure 2.5 shows the streamlines for the aforementioned system.

2.4.2 Principal load path

The principal load path in the structure is determined as the streamline through the highest gradient of the U^* index. In other words, the path through the lowest decay vector magnitude is considered the principal load path. This path is considered to transfer the highest portion of the load from the point of application to the supports. For the plate with a hole system discussed earlier, the thick white lines in figure 2.6 denote the principal load paths.

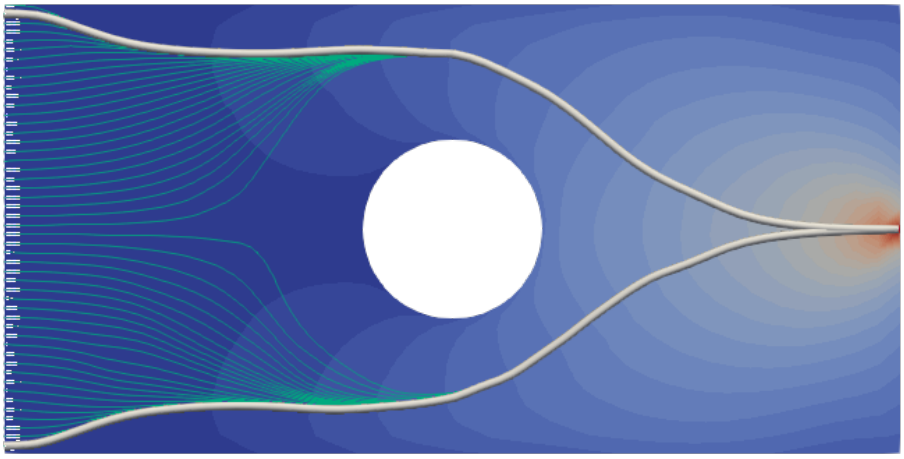


Figure 2.6: Principal load paths through the plate with hole structure

2.5 Uniformity and Continuity Plots

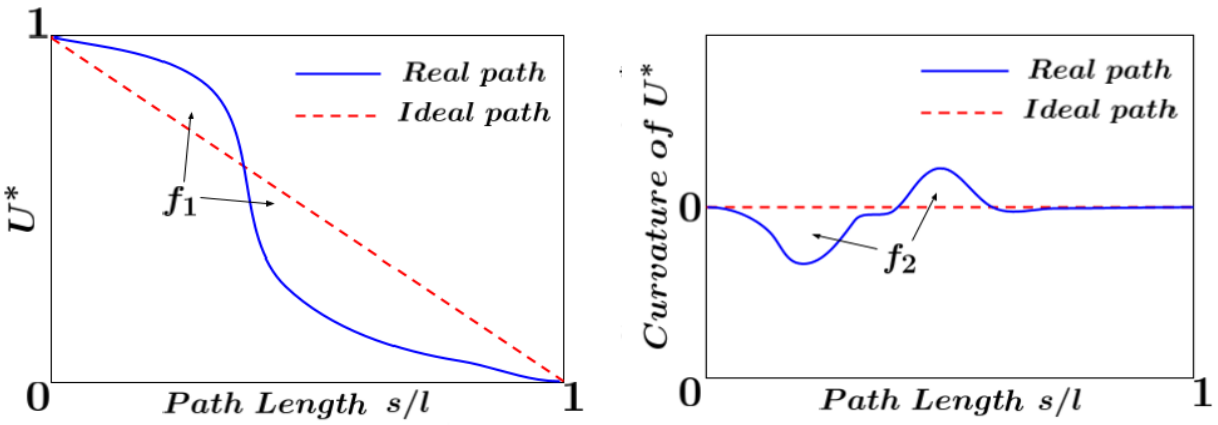
The principal streamlines, also called the principal load paths of the structure can be useful in two ways. The first is identification of regions of significance in the structure under a certain boundary condition to determine if the presence of any notches or cracks would affect the load carrying capacity of the structure.

The second is the applicability of load paths as a design criteria to optimise the structure for stiffness distribution. Previous studies [22] have demonstrated similarities between optimisation using sensitivity analysis and optimisation based on load path properties as design criteria. This quantitative use of load paths as design criteria requires the following three conditions:

- *Uniformity*: minimization of area between ideal and actual curves in Uniformity plot ($f1$ in figure 2.7a)
- *Continuity*: minimization of area between ideal and actual curves in Continuity plot ($f2$ in figure 2.7b)
- *Path Consistency*: minimization of area between load paths $S1$ and $S2$, where $S1$ is the load path between a loading point and a support point, and $S2$ is the load path between the same set of points but with the support and loading boundary conditions reversed

The current work limits itself to the uniformity and continuity criteria. The idea behind these two criteria is as follows:

1. The uniformity plot represents the distribution of U* values along the length of the load path, where s is the coordinate along the load path and l is the total length of the load path. It is represented by the solid blue line in figure 2.7a. The point of loading is taken as the seeding point of the load path, which explains the curve going from U^*=1 to U^*=0. The goal is to optimise the structure such that the stiffness is uniformly distributed along the load path, as represented by the dotted red line.



(a) Uniformity plot

(b) Continuity plot

Figure 2.7: Uniformity and Continuity plots

2. The continuity plot represents the distribution of curvature along the uniformity plot, as represented by the solid blue line in figure 2.7b. The goal is to optimise the structure such that there are no sudden changes in stiffness along the load path. The sudden changes are represented by the curvature [23] of the uniformity plot, given by formula 2.5.1.

$$\kappa = \frac{\frac{d^2U^*}{ds^2}}{\left[1 + \left(\frac{dU^*}{ds}\right)^2\right]^{3/2}} \tag{2.5.1}$$

3 Software

This section discusses the software that are used in the thesis work, mainly how these software are used in combination during the pre-processing, solution and post-processing stages of the load path analysis.

3.1 Ansys Parametric Design Language (APDL)

Mechanical APDL (MAPDL) is a popular finite element analysis program capable of performing a wide range of simulations including structural analysis, thermal analysis, electromagnetic analysis and vibro-acoustics. Most of the widely used commercial finite element software do not come with an in-built module that allows for visualization of load paths in a structure. However, Mechanical APDL offers users the flexibility to customize finite element simulations according to their specific needs using its APDL scripting feature. Ansys Parametric Design Language, also known as APDL is a powerful scripting language that runs Mechanical APDL. With scripting, users can interact with Ansys Mechanical Solver to solve complex finite element problems or compute user-defined results. In this project, the scripting capability of APDL was utilized to compute the U^* index values. An APDL script was developed to perform the finite element calculations required for U^* index, and the result files needed for post-processing were written using the same script. The current work makes use of Mechanical APDL 2022 R1 for the structural simulations.

3.2 MATLAB

MATLAB is a powerful high-level programming language that has the capability to perform complex numeric calculations, data manipulations and data visualization. In this project, MATLAB is used for the generation of *.vtk* format files and for other post-processing tasks. The output files obtained from Ansys Mechanical Solver are manipulated using MATLAB to generate a Visualization Tool Kit (.vtk) file. The mesh information, such as the nodal connectivity, nodal coordinates, element type and U^* index data, are manipulated and arranged in MATLAB to generate the *.vtk* file, which is used as the input file for the ParaView software to generate the load paths in the structure. Thus, MATLAB acts as a bridge between Mechanical APDL and ParaView. MALAB version R2016b is used for the thesis.

3.3 ParaView

ParaView is a competent tool designed to assist in analyzing and visualizing large scientific datasets. It is an open-source software that is easily accessible and can be used on various platforms, making it an excellent choice for researchers and scientists. With its wide variety of features and a user-friendly interface, ParaView is a powerful software for any individual or organization looking to gain insights into complex data.

The current thesis work utilizes the ParaView software for post-processing solution results, such as visualization of U^* contours and propagation of load paths through the structure. ParaView uses the Visualization Toolkit (VTK), a high level object oriented library, to provide visualization and data processing capabilities in a Graphical User Interface. The software

uses a pipeline-based architecture that allows for sequential manipulation of data to arrive at the desired result. The manipulation of data is done using the various filters built-into the software, and those relevant to the current work are discussed in detail under section 4.2.4. Figure 3.1 shows an example pipeline browser for load path visualization in a 2D structure.

Another major advantage of the ParaView software is its ability to automate post-processing tasks and to perform batch processing using Python. The 'Start/Stop Trace' tool that comes with the software allows users to record a set of actions in GUI and generate the corresponding Python script for the same. This facilitates faster implementation of automation routines in the software, and avoids the need for in-depth knowledge of the source code. This feature is particularly useful in the current work for the identification of the principal load path from the generated set of streamlines. The present thesis work uses ParaView version 5.4.1.

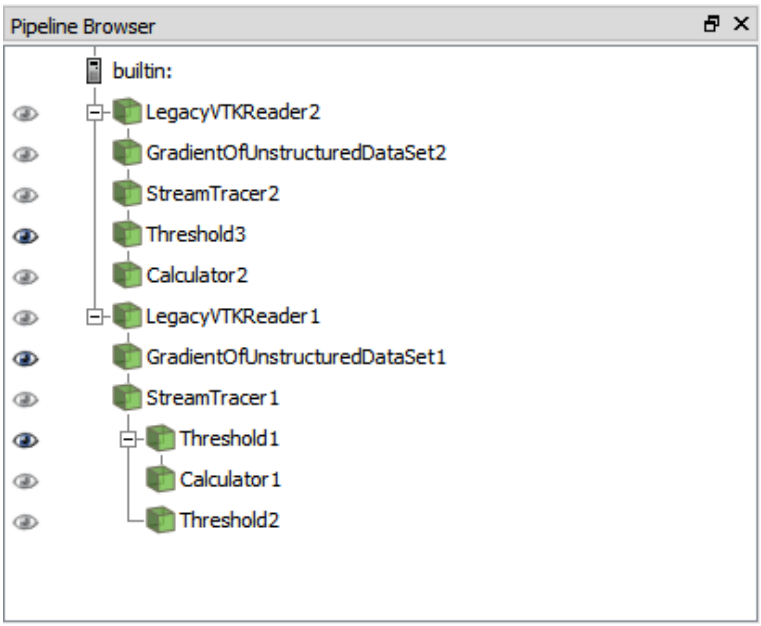


Figure 3.1: Pipeline browser example

4 Methodology

4.1 Solution Approach

The previous iterations of the study on U* index and load path visualization done at GKN Aerospace were focused on mechanical structures under point loading and single support conditions. The current study aims to build on the previous works by incorporating distributed loading, multiple loading interfaces and multiple supports for the computation of U^*. In addition, it also aims to identify the so called principal load path or the path of highest stiffness from the U^* index data. The concept of U^* index is used to determine the path of load transfer from the points of loading to the supports. The U^* contours are shown across the structure in figure 5.1 along with the corresponding streamlines. The principal streamlines or the principal load paths are depicted using thick lines.
The analysis is done using user-defined scripts in ANSYS Mechanical APDL, and the contours and load paths are visualized in ParaView. The data transfer from MAPDL to ParaView is done using a VTK file generated with the help of Matlab. In order to test the functionality of the developed APDL scripts, the analysis is first performed on a simple 2D structure such as a plate with a hole as shown in figure 4.5, under different boundary conditions. Later, the same script is modified to analyse 3D geometries such as the aero engine structure shown in Figure 4.11. As a means to validate the concept of U^* index, a correlation study between the point of maximum reaction force along the constrained boundary and the seed point of the principal load path is done. The sequential set of actions performed for the computation of U* index and for the visualisation of load paths are schematically explained in the following subsection.

4.2 Process Flow

The process flow for visualising load paths consists of three phases.

- Computation of U^* using APDL scripts
- Generation of VTK file using MATLAB
- Generation of streamlines and principal load path in Parview

4.2.1 U^* computation using APDL scripts

The first phase employs user-defined scripts in APDL to compute U^* data at the Finite Element nodes and then output the U^* solution data and the mesh data into text files. The computation of the U^* index involves looping over the desired set of nodes, and changing the boundary conditions in every such loop. This is done by the use of load steps, wherein each load step pertains to a loop in U^* computation. This process therefore requires the use of **/POST26**, APDL's time history post-processor to execute the aforementioned loops in separate load steps. The flow in this phase is different for the direct method and the inspection loading method, as depicted in figures 4.1 to 4.3.

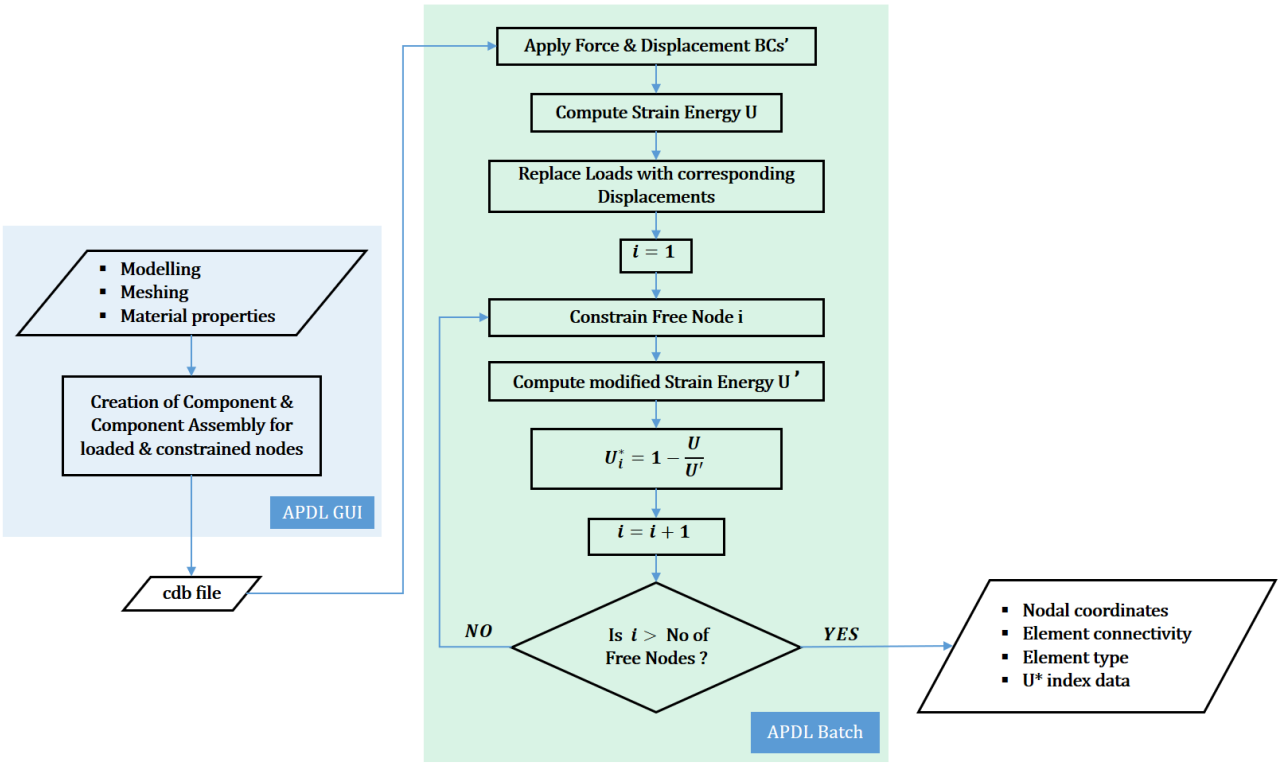


Figure 4.1: Process flow for U^* computation using the Direct Method

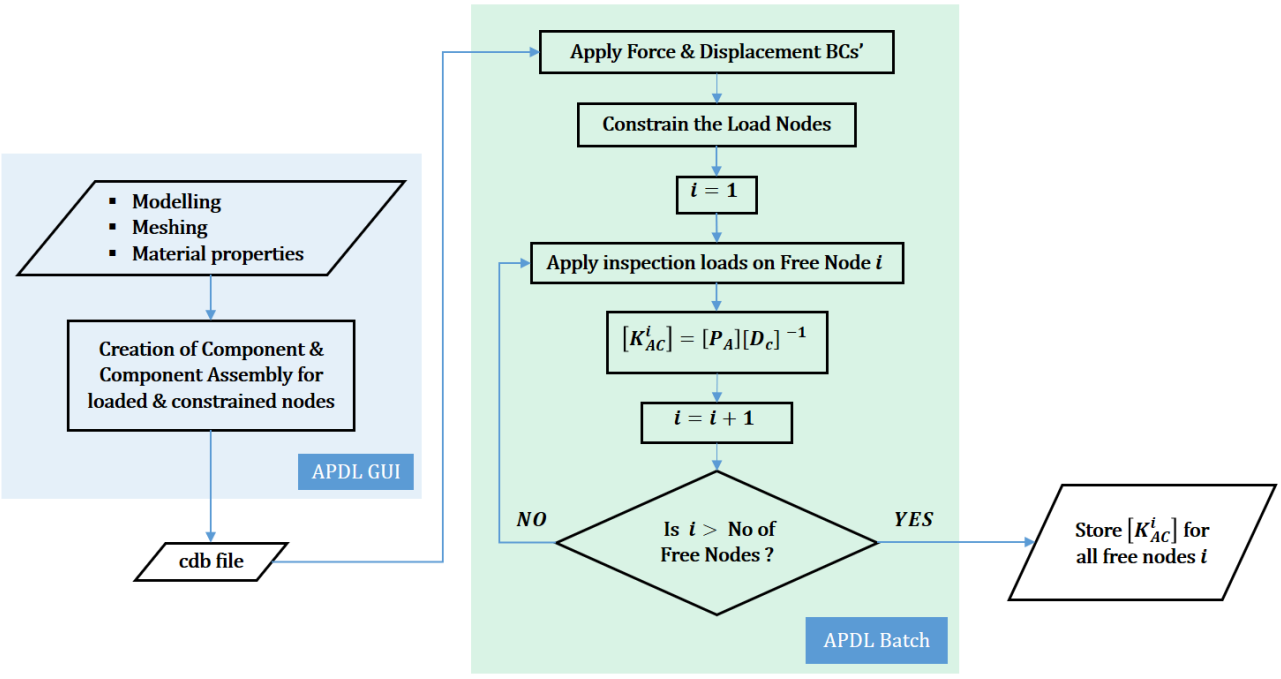


Figure 4.2: Process flow for K_{AC} extraction using Inspection Load Method

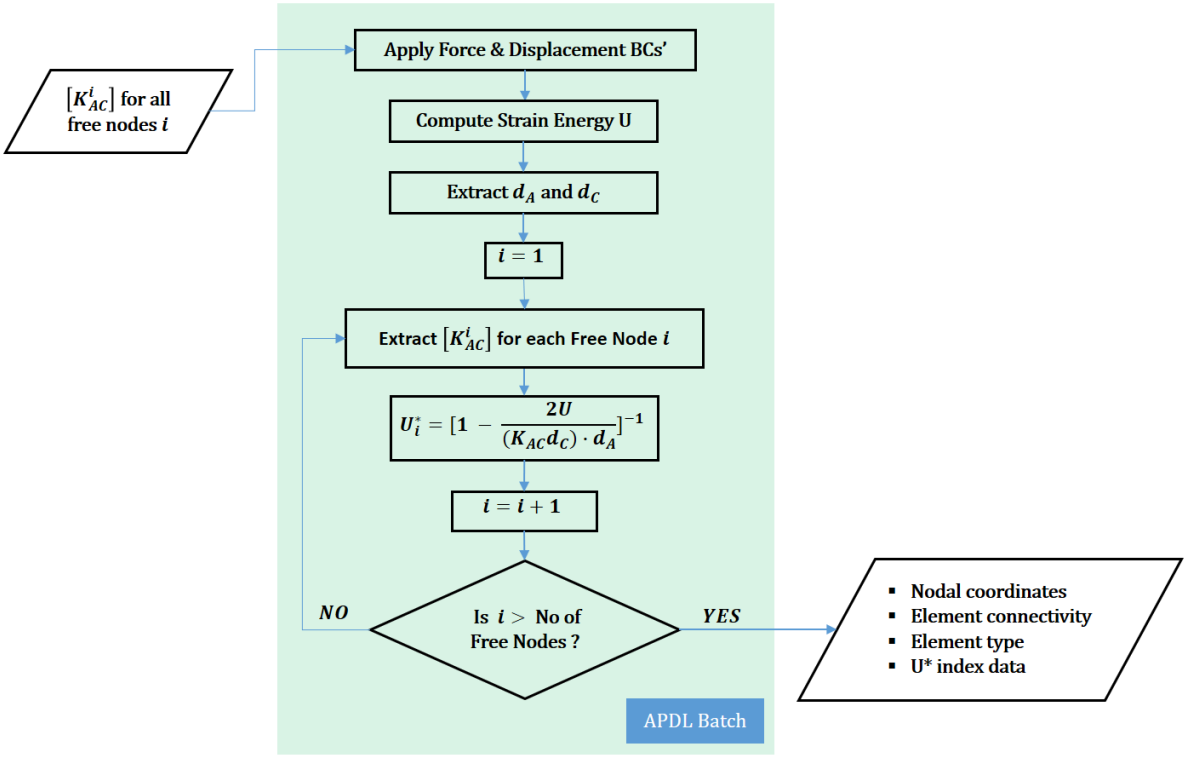


Figure 4.3: Process flow for U^* computation in Inspection Load Method

As explained in section 2.3.3, the condensed stiffness matrices K_{AC} connecting the loading points and the point of interest are independent of the force boundary condition on the structure. Therefore, the inspection load method allows re-usability of the stiffness matrices for calculating the U^* distribution under different loading conditions. An APDL script is used to extract the K_{AC} matrices for all points of interest in the system, as shown in figure 4.2. The matrices are exported in MAPDL specific native format and stored for a particular structure. Though the extraction of stiffness matrices is computationally expensive for the initial simulation run, the exported file can be reused for any number of load cases on the structure as long as the displacement boundary conditions remain the same, as shown in figure 4.3. This allows significant reduction in computation time compared to the direct method.

4.2.2 Generation of VTK file using MATLAB

The second involves conversion of APDL output data to VTK format for visualisation in open source software, Paraview. This is done through data processing in MATLAB. The MATLAB script takes as input, the U^* data file 'U_star.txt', and the mesh related data files 'element_type.txt', 'ELIST.txt' and 'NLIST.txt' to generate the '.vtk' file. The Legacy Format is chosen for the creation of VTK file, as it is the simplest of the available formats. More details about the Legacy format can be found in The Visualization Toolkit (4th ed.) [24].

4.2.3 Generation of streamlines and principal load path in Parview

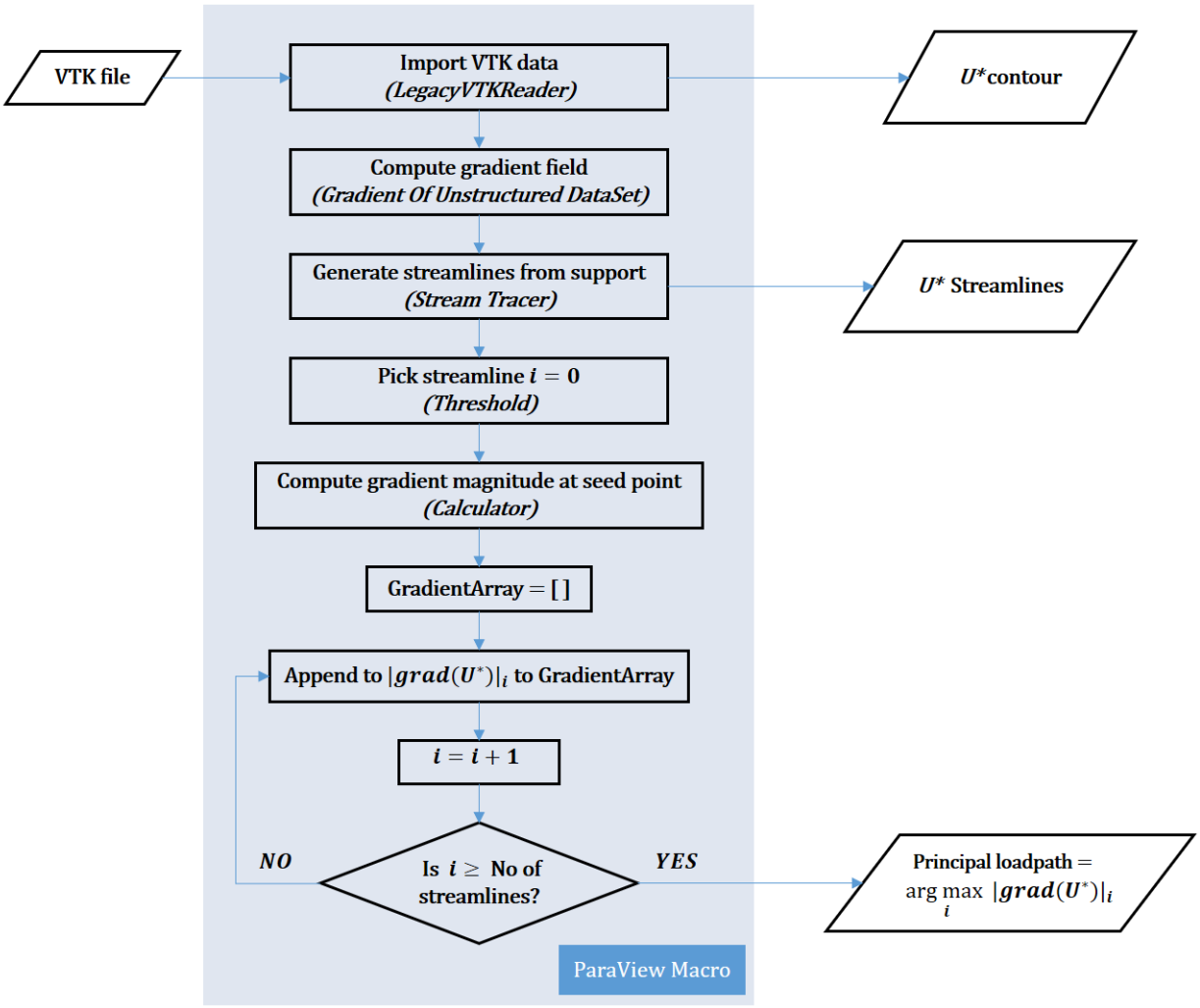


Figure 4.4: Process flow for generating streamlines and principal load path in ParaView

The final phase involves the use of Paraview to visualise the streamlines, and implementation of Python scripts to automate post-processing in Paraview. The Legacy format 'VTK' file is first imported into ParaView using the Legacy VTK Reader, and the various filters available in the software are applied successively to transform the input data into streamlines and the principal load path. The process flow for generation of streamlines and the principal load path is given in figure 4.4 which is followed by brief description of the pipeline browser and filters used.

4.2.4 Filters

The filters used in the current work in order to visualize load paths are as follows.

1. **Gradient Of Unstructured DataSet** filter calculates the local gradients for all cells in the domain and then averages out the gradients to obtain a continuous field. In this

case, the gradient field is required to determine streamlines through the U^* contour.

2. **Extract Surface**: This filter is used when visualizing load paths through thin walled 3D structures. The streamlines are generated based on the gradient of U^* field in the domain, and are initiated from user-defined seed points. In the case of thin-walled structures, the Stream Tracer filter might not yield the desired streamlines connecting the point of loading and point of support. This could be attributed to the gradient vectors not lying in the plane of membrane, thus leading to short streamlines that terminate at the surface of the membrane. In order to circumvent this problem, the U^* index data from the 3D continuum is interpolated onto the surfaces using the Extract Surface filter to compute streamlines on surfaces.

3. **Stream Tracer** filter generates streamlines from seed points in a given vector field. The seed points could be provided using a 'High Resolution Line Source' or a 'Point Source'. Various attributes including the maximum length of the streamline, minimum and maximum step length, and surface streamlines are used to control the generated streamline.

4. **Extract Selection** filter is used to extract selected mesh entities such as cells or points as separate objects in the pipeline browser. This enables the user to pick nodes on the face of the support to act as seed points for the streamlines.

5. **Stream Tracer With Custom Source** filter takes seed points from the created Extract Selection object. This is particularly useful in the case of 3D structures when seeding points are not easy to pick using a High Resolution Line source or Point source.

6. **Threshold** filter slices portions of the input dataset using a specified range on available scalar attributes. When applied on streamlines, it allows for selection of one streamline at a time using Seed ID attribute in the process of determining the principal load path.

7. **Calculator** filter operates on existing arrays to create a new array. The present work requires the magnitude of gradient to be calculated at the seed points in order to determine the principal load path, and the Calculator filter is employed for the same.

8. **Tube** filter constructs tubes around input lines for improved visibility. This is used to highlight the principal load paths when plotted with the rest of the streamlines.

4.3 Load Path Visualization in 2D structure

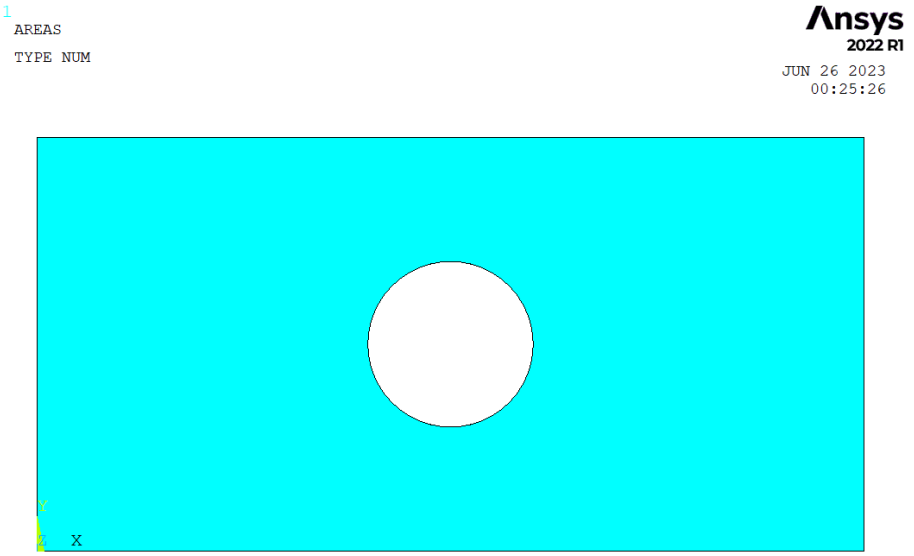


Figure 4.5: Rectangular plate with hole

A 2D plate with a hole, as shown in figure 4.5, is chosen for the load path visualization. This geometry was chosen as the load paths could be verified and validated using the results presented in the paper [21] . The simple nature of this geometry meant that the computation time is less and the MAPDL script could be modified and debugged for errors relatively quickly.

The plate's length, breadth and thickness are taken as 2m, 1m and 0.005m, respectively. The hole diameter is 0.4m. The material is chosen as steel with Young's Modulus value of 210GPa and Poisson's ratio of 0.3. This plate is used for the many cases of study done in the following subsections. In all such cases, the load paths are identified for static structural analysis under plane stress assumptions.

4.3.1 Analysis Procedure

In this thesis work, the load path analysis is broken down into three main stages: pre-processing, solution, and post-processing. This process is consistently applied to all analysis cases. The following section will provide a step-by-step explanation of each stage.

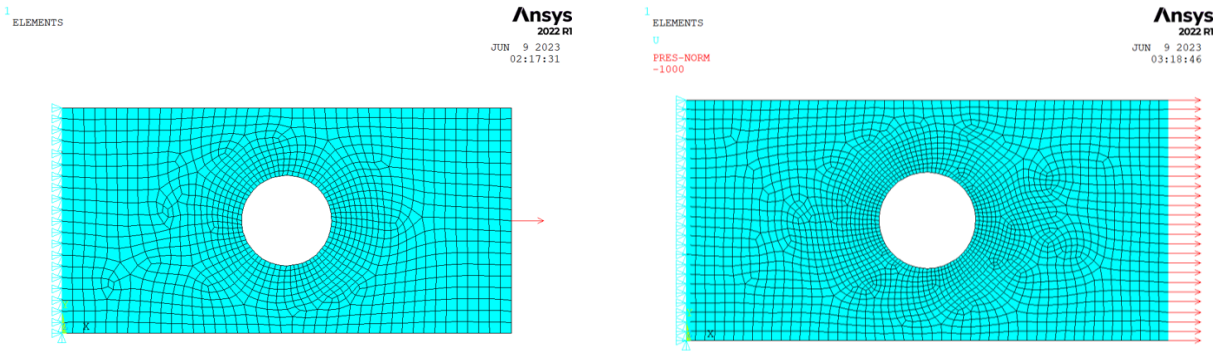
- The geometry cleanup is performed to get rid of construction lines, sliver faces and cad penetrations and intersections.
- Appropriate element type is chosen to fit the nature of the geometry and analysis.
- The geometry is meshed with the relevant element size. The loading and support points are identified in the active mesh. MAPDL GUI is used for this.
- The mesh data is exported as CBD file.

- The U^* computation is performed in MAPDL using the developed script.
- The output of the load path analysis from MAPDL is extracted. The output files contain the element connectivity, nodal coordinates, element type and U^* index data.
- Using the output from MAPDL, a VTK file is generated using a script developed in MATLAB.
- The VTK file serves as the input to ParaView. The U^* gradients are computed, and the load paths are generated as streamlines.
- Python script is used to automate the process of extracting streamline data to identify the principal streamlines. For the principal streamline the uniformity and continuity graphs are plotted.

4.3.2 Point Loading

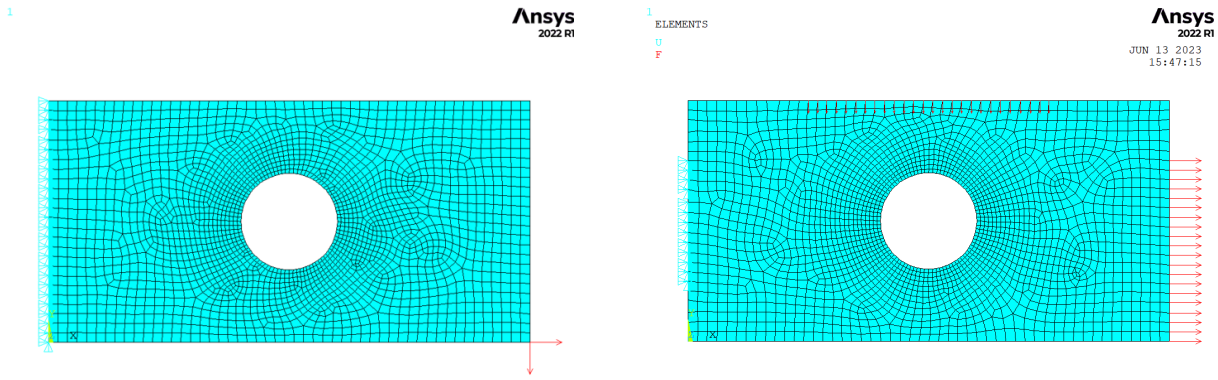
At GKN, Rajesh, Jonathan, and Oscar [15] [17] conducted load path analysis work, which has now been replicated on a plate with a hole structure (as depicted in Figure 4.6a). This replication serves as a first step to comprehend the complexities and challenges involved in using the U^* index method and APDL scripting for load path calculation. Additionally, this initial step aims to understand the post-processing aspect of load path analysis, which involves using ParaView software to plot load paths and visualize the principal load path that may not be immediately apparent.

A point load of 1000N is applied to the plate. The loading is done in the positive x-axis direction. The left-hand side edge of the domain is fixed. The loading and boundary conditions are shown in figure 4.6a. The direct method of U^* index computation is used here. 4 noded quadrilateral elements are used to mesh the component. A total of 216 nodes are present in the structure, and a coarse mesh is used for ease of analysis.



(a) Point Load (b) Distributed Load

Figure 4.6: Boundary conditions applied on plate with hole structure.



(a) Point Bending Load (b) Multiple Distributed Loading Interfaces

Figure 4.7: Boundary conditions applied on plate with hole structure.

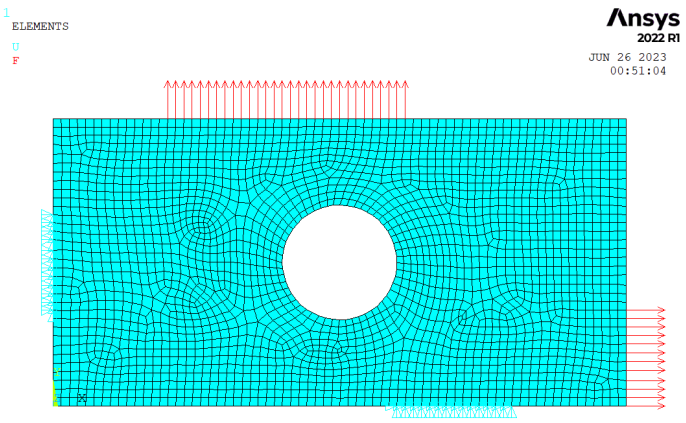


Figure 4.8: Multiple Supports and Loading Interfaces

4.3.3 Distributed Loading

As part of the current thesis work, distributed loading has been included for load path analysis. The APDL script has been modified to accommodate distributed loading, and the theory section discusses the equations used for such loads.

A load of 1000N per node is applied on the right side boundary of the plate with a hole structure shown in Figure 4.6b. The left edge of the boundary is fixed, the U^* computation is performed in MAPDL, and the load paths thus generated are plotted in ParaView. The Uniformity and Continuity plots for the principal load paths are plotted.

4.3.4 Distributed Loading at Multiple Interface

A case study is done to understand the nature of the load transfer when distributed loads are applied at multiple locations.

As shown in the Figure 4.7b, distributed loads are applied on the top and right boundary of the plate with hole structure. The left boundary is fixed. The U^* computation is performed

in MAPDL, and the load paths thus generated are plotted in ParaView. The Uniformity and Continuity plots for the principal load paths are plotted.

4.3.5 Distributed Loading at Multiple Interface and with Multiple Constraints

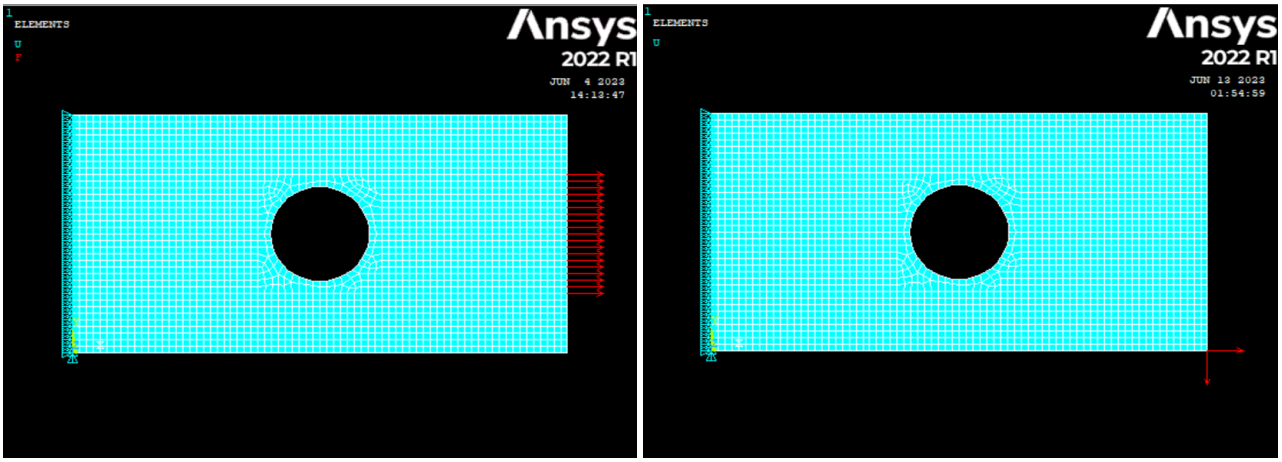
In order to understand how the load is transferred when distributed loads are applied at multiple locations with multiple supports, a second case study is conducted.

The diagram referenced as Figure 4.8 shows distributed loads on the plate with a hole structure's top and right borders. The left and bottom border is fixed, as shown in Figure 4.8. The computation for U^* is done in MAPDL, and the resulting load paths are visualized in ParaView. The Uniformity and Continuity plots display the primary load paths.

4.4 Validation of load paths from U^* index

4.4.1 Case study to validate load path analysis

Two case studies are done on the plate with a hole structure to confirm that the principal load path is the stiffest in the structure and transfers majority of the load from the loading boundary to the supports. The boundary conditions used in the case study are illustrated in Figure 4.9. For the study-1, the left edge of the plate is fixed, and the right edge is loaded, as shown in Figure 4.9a. In the study-2, a point bending load is applied, the boundary conditions applied are shown in Figure 4.9b.



(a) Case study-1.

(b) Case study-2.

Figure 4.9: Boundary conditions applied for the two case study.

The entire domain of the plate is divided into 32 separate areas, as shown in Figure 4.10. Initially, the entire domain area is assigned a plate thickness of 5mm and meshed using first-order quadrilateral elements. The load path analysis is performed on this structure, and the load paths are extracted. The maximum deflection produced in the plate is also noted. Once the load paths through the structure are established, specific segmented areas in the plate are strategically chosen, and the thickness of these segments is increased from 5mm to 10mm. The area segments are chosen so that some are areas through which the principal load path

passes. The boundary conditions of the original plate are then applied to the system. The system is then solved and the maximum deflection produced is noted. A comparison of the maximum deflection produced in the plate for different cases is made.

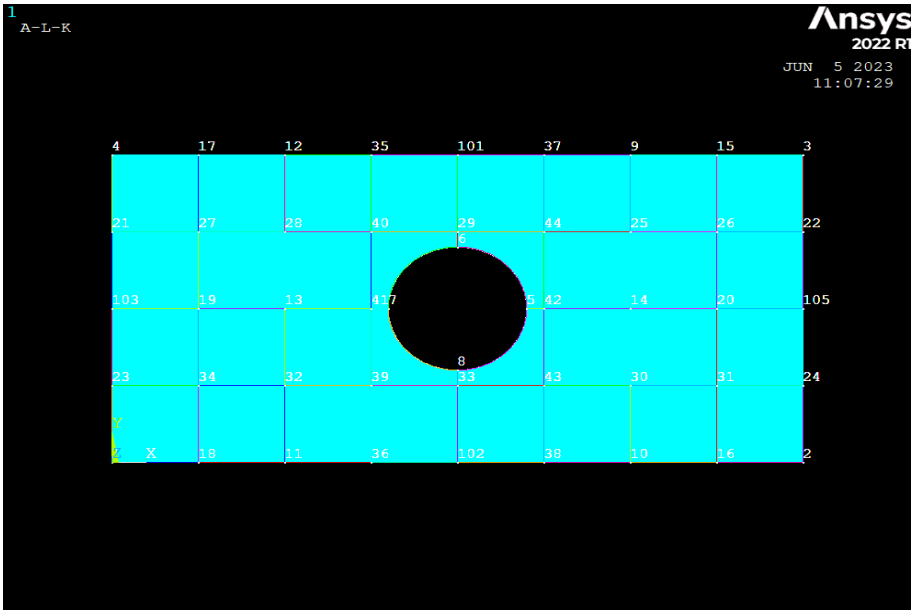


Figure 4.10: The domain of the plate split into 32 area segments.

Four separate cases are analysed on the two study cases shown in Figure 4.9 plate domain. These studies are done to show that the most rigid route through the structure is the principal load path. To reduce deflection in the system, it is more effective to stiffen the regions of the plate that the principal load path passes through, rather than stiffening a region outside of it.

4.4.2 Correlation of load paths with regions of highest reaction

In a static equilibrium, the loads/forces applied on different regions of the body should be balanced by reaction forces arising at the support. The load paths are said to be paths or trajectories that transmit the highest share of loads through the structure for a given loading configuration. The hypothesis behind the U^* index method is that the path with the highest stiffness carries the highest load. As a means to test this hypothesis, the reaction traction at the support boundary is calculated and plotted along the boundary coordinates to check if there is indeed a correlation between the regions of highest reaction and the point of intersection of the principal load path with the support boundary. This correlation if achieved would serve as further validation for the use of U^* index in finding load paths through structures. The formula for calculating the traction from FE nodal reaction forces is described below.
Let $\Gamma_c \subset \partial\Omega$ be the constrained boundary where the reaction traction is to be measured. The nodal reaction forces are given by the vector $\underline{f}_c$, and the traction $\underline{t}_c$ is to be computed.

$$\underline{f}_c = \int_{\Gamma_c} \underline{N}_c^{\mathrm{T}}\, \underline{t}_c \, \mathrm{d}\Gamma \tag{4.4.1}$$

Upon inserting the Least squares approximation $\underline{t}_c(x) = \underline{N}_c(x)\,\underline{b}$ where $\underline{b}$ is the nodal traction vector, the equation 4.4.1 becomes 4.4.2.

$$\underline{f}_c = \int_{\Gamma_c} \underline{N}_c^{\mathrm{T}}\,\underline{N}_c\,\mathrm{d}\Gamma\,\underline{b} = \underline{M}\,\underline{b} \tag{4.4.2}$$

4.5 Load Path Visualization in 3D structure

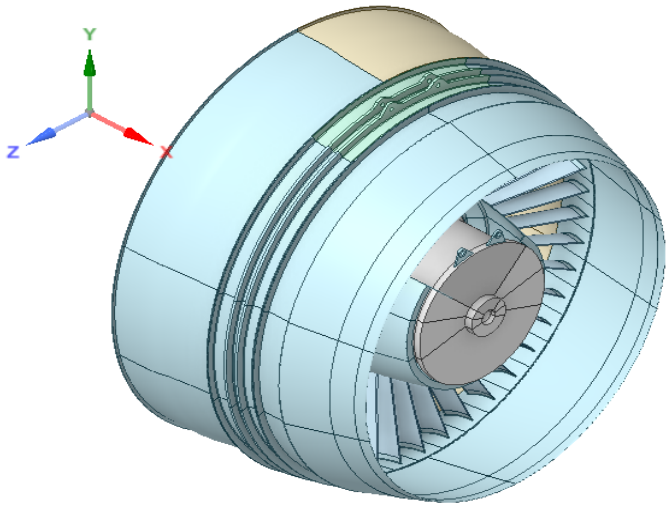


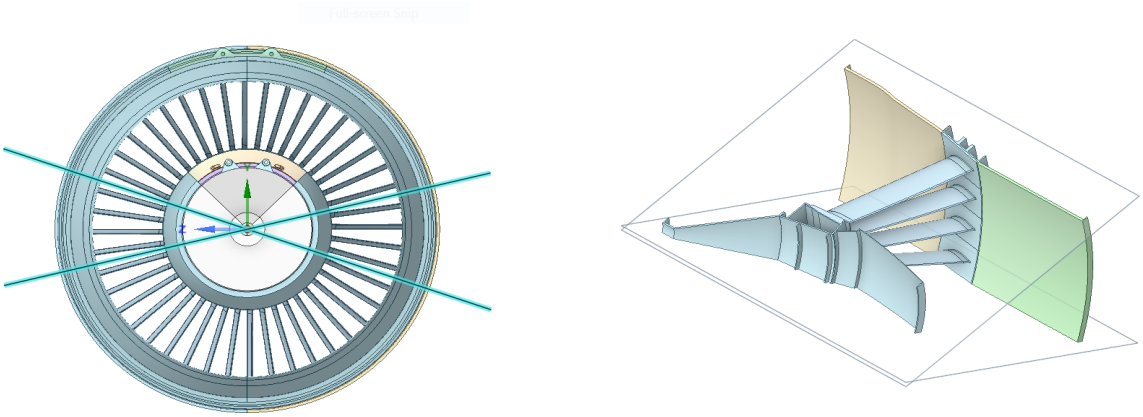
Figure 4.11: EleFanT - Electric Fan Thruster.

The Electric Fan Thruster, EleFanT 4.11, is the engine structure used for load path analysis in the current thesis work. Due to the complex nature of the geometry and the high computational time associated with U^* index computation, the geometry is cut into a section with four guide veins. This section fairly represents the nature of the load transfer in the primary structure when loaded under similar conditions. The cut section of the primary geometry is shown in Figure 4.12b. Blue lines in Figure 4.12a represent the planes used to cut the geometry.

The simplified geometry is meshed using 3D first-order tetrahedral elements (Linear Elements) as shown in Figure 4.13b. Several loading scenarios are studied to realize which guide veins are active participants in the load transfer. Depending on the nature of the loads, different guide veins are expected to be active in transferring loads.

The SpaceClaim software is used to perform the necessary geometry cleanup and perform sectioning to simplify the geometry. The cleaned-up geometry is imported into ANSYS Workbench for meshing. A $10mm$ element size is used to mesh the geometry, and structural steel is used for the material. The geometry is meshed with first-order tetrahedral elements, as shown in Figure 4.13b. The relevantly named selections are made in the workbench to identify the loaded and fixed nodes. Using the 'write input' option, a '.cdb' file is written, used as the

input file for ANSYS APDL mechanical solver. The CDB file contains the mesh information, such as the element connectivity and nodal coordinates.

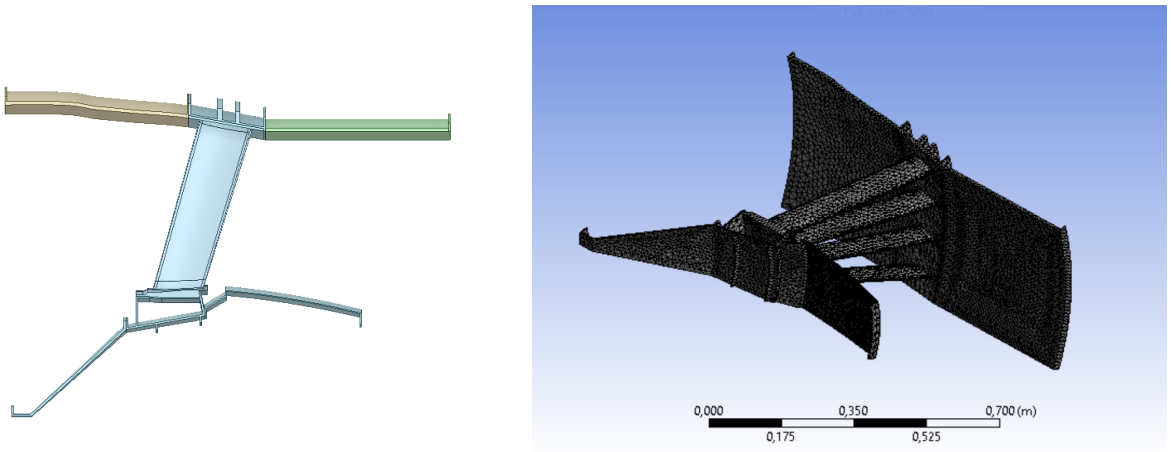


(a) Cut Planes. (b) Cut Section.

Figure 4.12: EleFanT geometry simplified.

The CDB file is imported into the ANSYS APDL batch solver for U^* computation. The output files from the batch solver, namely the element connectivity, nodal coordinates, element type and U^* index values, are used to create a VTK file in MATLAB. The generated VTK file acts as the input file for post-processing in ParaView.

The ParaView software extracts the load paths and visualizes the load transfer for several load cases.



(a) Simplified 3D Model. (b) First order tetrahedral mesh.

Figure 4.13: The EleFanT cad used for load path visualization.

5 Results and Discussions

This section discusses the load path analysis results of the various load cases studied in the current thesis work. Certain inferences are drawn from the results of 2D and 3D analysis. The challenges faced in load path visualization and the potential solutions are discussed in detail.

5.1 Load Path Visualization in 2D Structure

A plate with a hole structure is analyzed under different loading and boundary conditions to study the nature of the load paths through the structure.

5.1.1 Point Loading

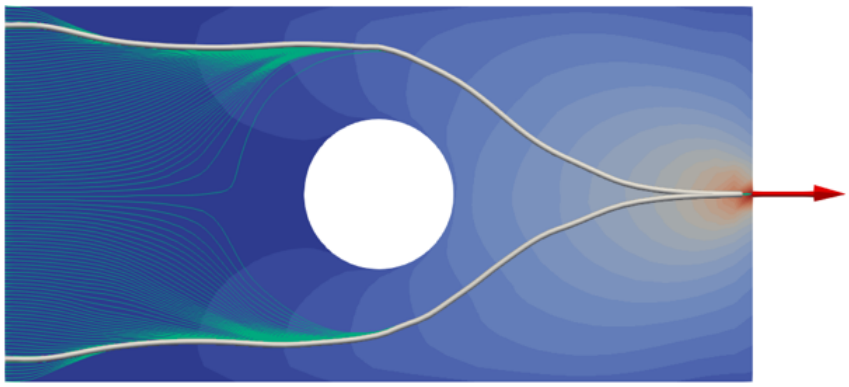
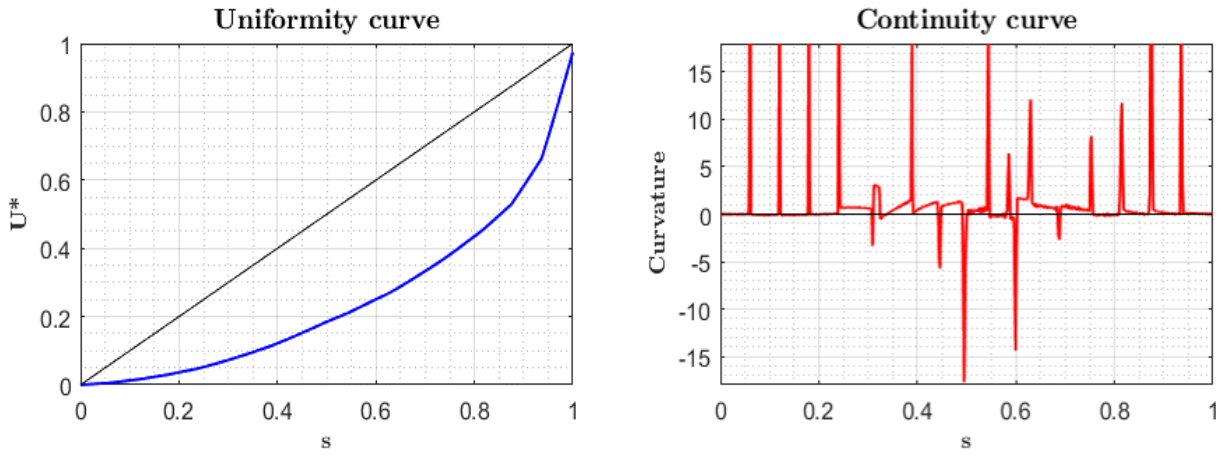


Figure 5.1: Load path in a plate with hole structure under point loading

In order to review GKN's previous load path visualization work, an APDL script was developed to analyze a basic plate with a hole structure. The U^* index method was used to conduct load path analysis for a point-loading scenario as shown in Figure 5.1. The fixed boundary has a U^* index value of 0, while the loading point has a U^* index value of 1.

The fixed boundary is chosen as the seeding edge for the initiation of streamlines as it has a large enough dimension to enable a wide spread for seeding points. It can be observed that the streamlines move through the plate, starting from the fixed boundary and reaching the loading point. The stream tracer functionality of ParaView plots the streamlines by connecting the U^* gradient vectors tangentially starting from the seed points. Notably, each streamline is distinct and does not intersect with any others.

The two streamlines highlighted using thick white tubes represent the principal load paths, i.e. these are the paths through the structure that transfer the highest amount of load, consequently, the stiffest paths. By definition, the principal load paths are represented by the streamlines that pass through the highest gradient of the U^* index, hence the stiffest route.



(a) Uniformity Curve

(b) Continuity Curve

Figure 5.2: Unfitted Uniformity and Continuity curve for the Principal Streamline

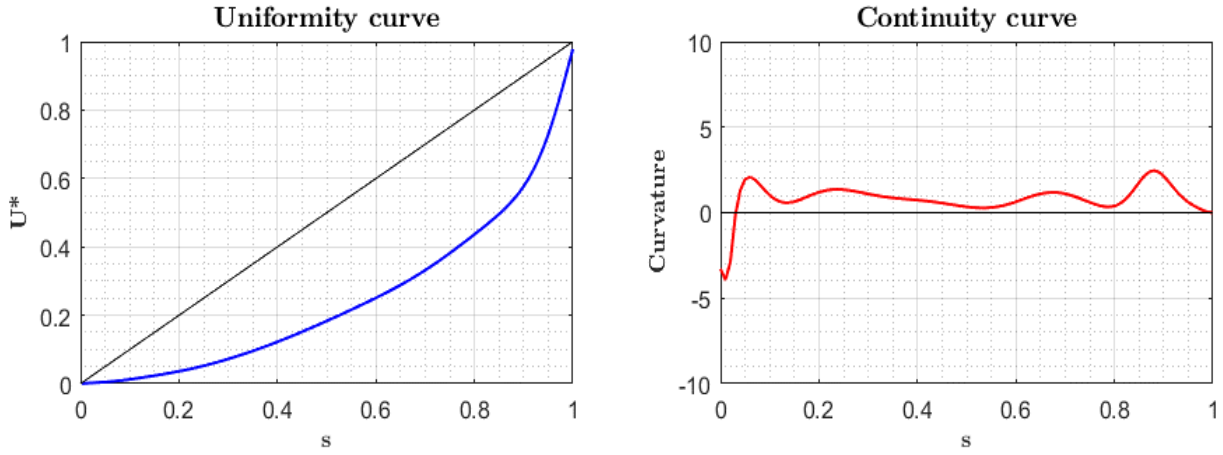
The two design criteria used to assess the uniformity of stiffness distribution and the abrupt changes in the stiffness along the principal load path are expressed using the uniformity and the continuity curves plotted in Figure 5.3. The Uniformity curves for the principal streamline is shown in the Figure 5.3a. The load paths generated by ParaView are made up of numerous small line segments stitched together to make up the entire streamline. The cumulative length of these line segments at each step is divided by the entire length of the streamline and this normalized length is represented by the horizontal axis of the uniformity plot. The U^* index value is represented on the vertical axis.

The ideal or the desired condition is to have a uniform distribution of the stiffness throughout the load path, which would yield a uniformity plot that is a straight line that goes from a U^* index value of 0 to 1, as indicated by the solid black line in Figure 5.3a. The curved blue line is the uniformity plot of the principal load path for the plate with the hole structure loaded with a point load. From the plot, it is fair to conclude that the stiffness distribution in the route of load transfer is not optimal. The structure could be stiffened further. The area between the ideal and original uniformity curve could be reduced to optimize that structure.

It is to be noted that the uniformity curve generated from ParaView's streamline data was not smooth and required least squares fitting onto a curve of sufficient polynomial order. This is especially important for generating the continuity curve, which is highly sensitive to changes in direction of the uniformity curve. Figure 5.2 shows the uniformity and continuity plots generated using unfitted data from ParaView. Figure 5.3 shows the plots after the uniformity curve was fitted onto a polynomial of order 12. The fluctuations in the uniformity and continuity data arise because of the numerical search algorithm that ParaView uses to trace the load paths.

Since the stiffest path transfers majority of the loads, it would be a fair assumption that the magnitude of reaction forces at the fixed support at these points should also be high. Based on this, a study of the reaction traction along the support is shown in Figure 5.4. The vertical axis of the plot represents the magnitude of the traction along support and the horizontal axis represents the y-coordinate of the support edge, which in this case is the left edge of the

plate. It could be observed from the figure that there is a noticeable kink at the end points of the graph with unnatural peaks. This is a characteristic of the least squares fitting of the traction values, and could therefore be neglected. Thus, it could be said that the highest traction occurs at the end points of the fixed support.These reaction values are fairly close to the principal load path initiation points on the support edge of the plate. Thus validating the premises that the principal load paths represent a path of maximum load transfer.



(a) Uniformity Curve

(b) Continuity Curve

Figure 5.3: Uniformity and Continuity curve for Principal Streamline
Polynomial order = 12

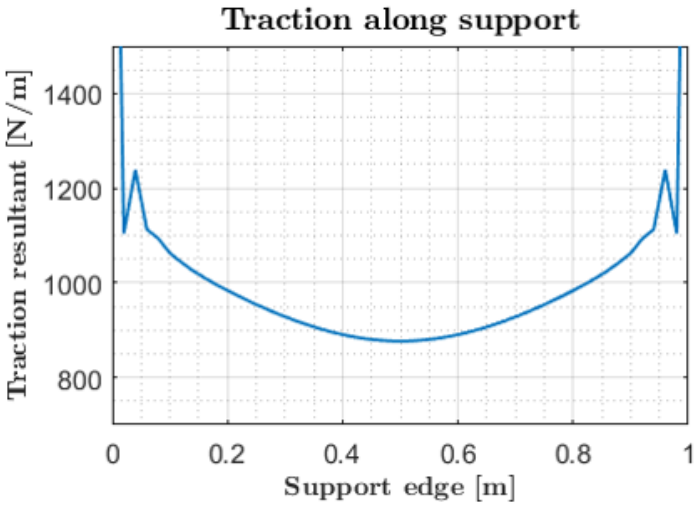


Figure 5.4: Reaction traction along fixed support

5.1.2 Distributed Loading

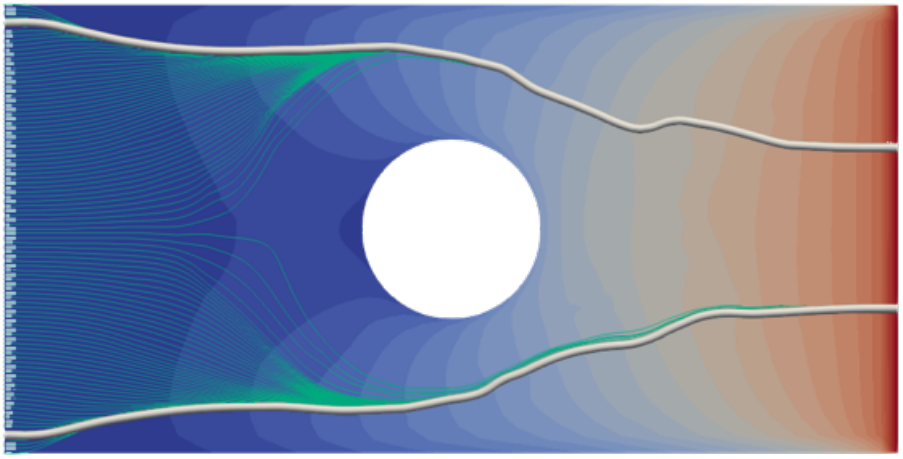


Figure 5.5: Load path in a plate with hole structure under distributed loading.

The load path visualization when a distributed load is applied to a structure is one of the specific goals of this thesis work. To achieve this, a plate with a hole structure is used for the load path analysis. The load paths through the structure can be seen in Figure 5.5 when a distributed load is applied on the right edge of the plate while the left edge is fixed.

The green-coloured streamlines which are supposed to be representative of the flow of load through the structure do not travel from the entire support edge to the entire loading edge, but only to two select regions on the loading edge. Hence it could be said that the U^* index method does not provide an accurate prediction for the entire flow of load through the structure. However, it could be observed that the principal load paths denoted by thick white tubes are found to be in good correlation of the highest reaction regions on the constrained edge as shown in figure 5.6. As a further step in validating the applicability of U^* index in determination of principal load paths, a case study is performed in the following section 5.1.6.

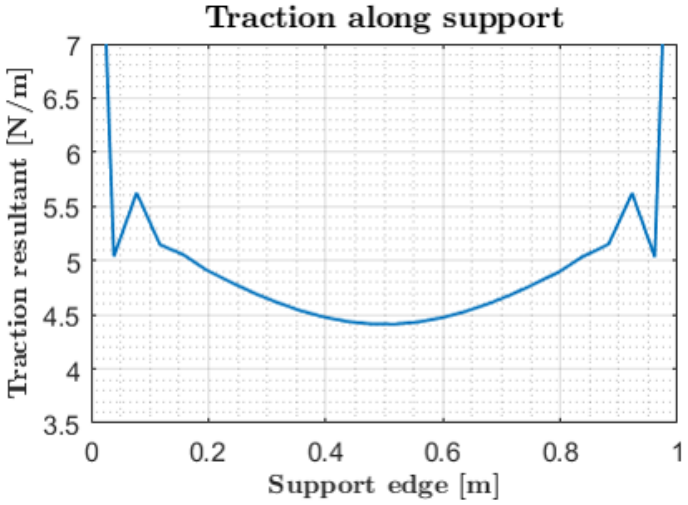
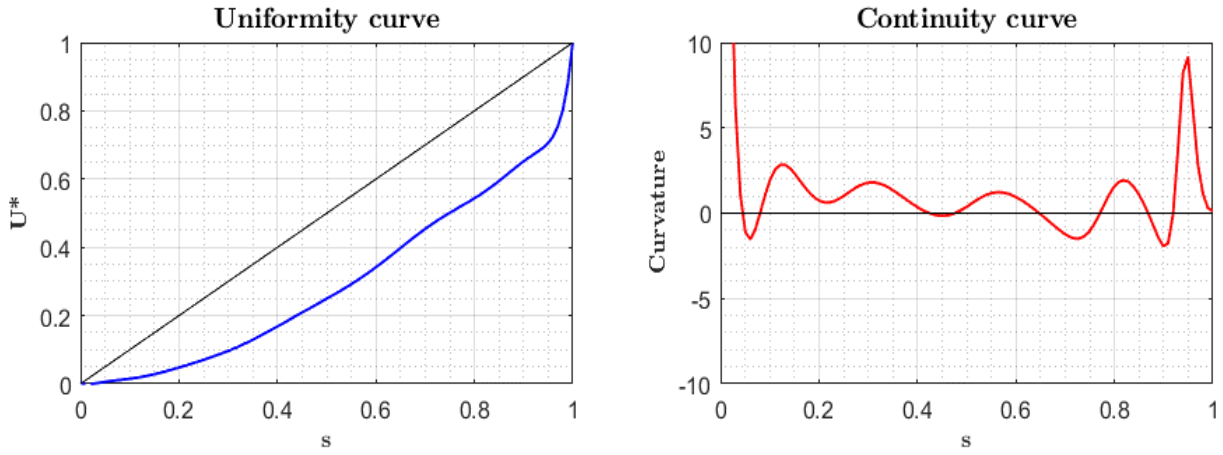


Figure 5.6: Reaction traction along fixed support

The uniformity and continuity curves plotted in figure 5.7 suggest a non-uniform stiffness distribution along the load path. The uniformity curve for the load path in figure 5.7a lies below the ideal uniformity curve given by the straight black line and therefore suggests a necessity for increasing the stiffness.



(a) Uniformity Curve

(b) Continuity Curve.

Figure 5.7: Uniformity and Continuity curve for Principal Streamline.

5.1.3 Point Bending Load

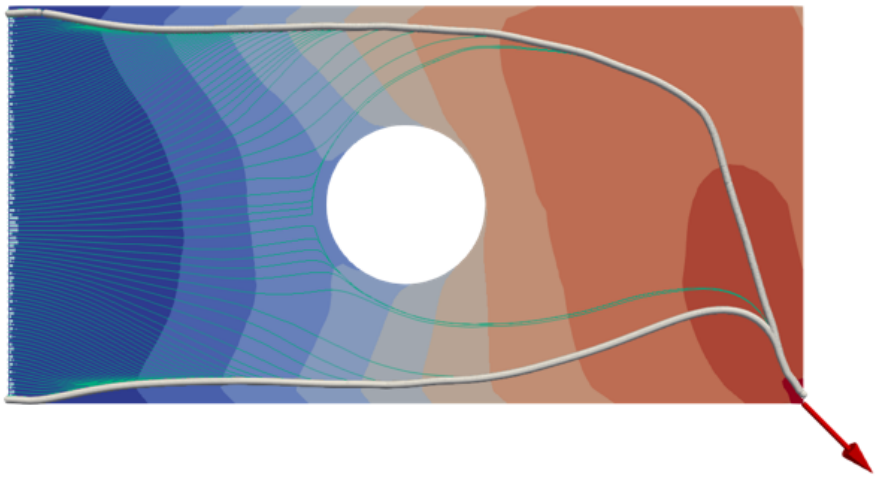
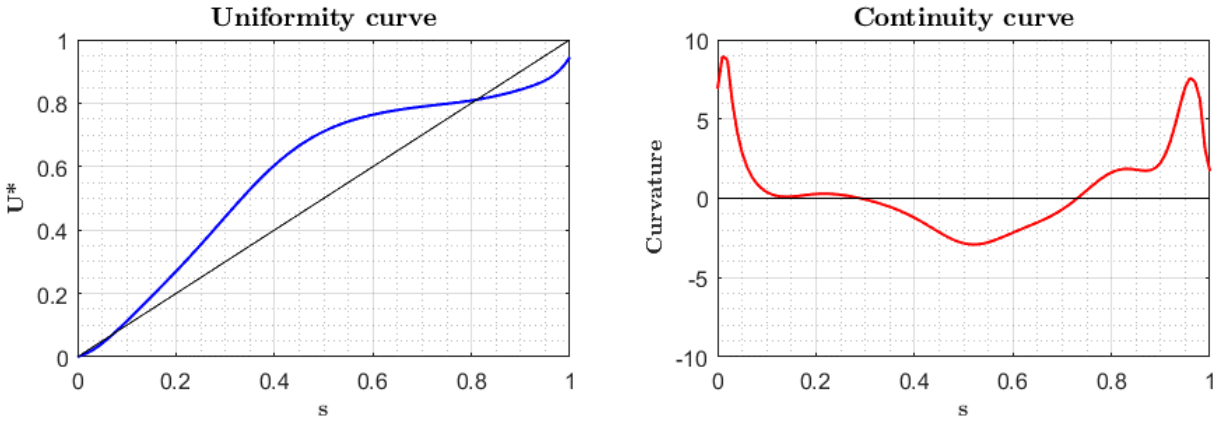


Figure 5.8: Load path in a plate with hole structure under point bending load

Both the loading cases studied previously were symmetric in terms of boundary conditions and were also similar in the nature of loading (tensile). Moreover, the uniformity curves obtained from those cases were both below the ideal line, suggesting a necessity for increase in stiffness. However, different kinds of results are required for better insight into the uniformity and continuity of load paths. Aimed at achieving a difference in pattern, this load case therefore

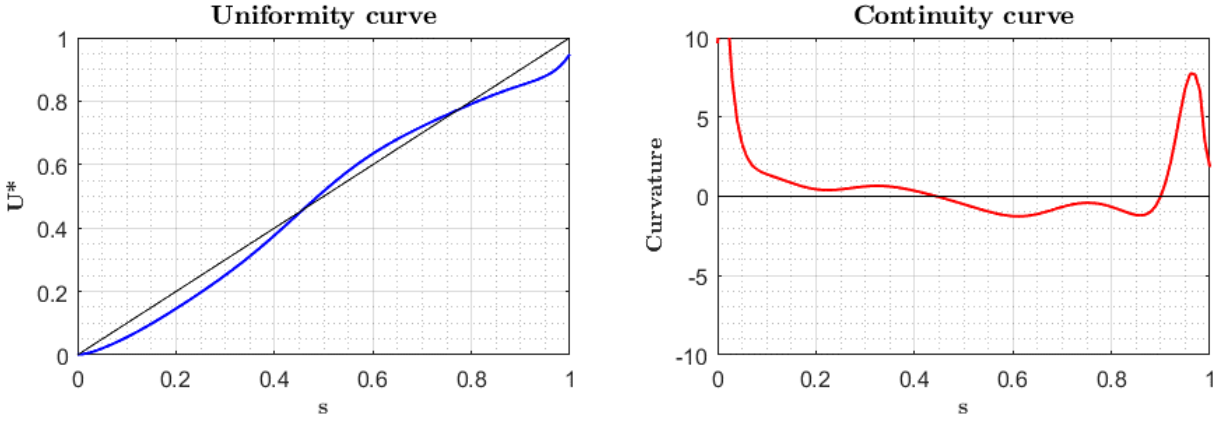
employs an unsymmetric boundary condition with a bending load. This loading yields an unsymmetric U^* contour with a first principal load path and a second principal load path.

The first principal load path, i.e. the streamline with the highest gradient of U^*, connects the loading point with the top corner of the support edge as shown in figure 5.8. The highest reaction along the support edge occurs at the top corner at $y = 1$ in figure 5.11. It can therefore be concluded that there is a good correlation between the first principal load path and the highest reaction region along the support.



(a) Uniformity Curve (b) Continuity Curve.

Figure 5.9: Uniformity and Continuity curve for Principal Streamline 1



(a) Uniformity Curve (b) Continuity Curve.

Figure 5.10: Uniformity and Continuity curve for Principal Streamline 2

The same can be said about the second principal load path, which in figure 5.8 is represented by the thick white tube from the loading point to the bottom corner of the constrained edge. A case study has also been performed for this system in section 5.1.6 demonstrating the significance of the first and second principal load paths and their impact on maximum deformation in the structure. The case study concludes that the first principal load path is the stiffest against deformation, followed by the second principal load path. Furthermore, the uniformity curves in figures 5.9 and 5.10 show that the first principal load path has a higher stiffness than the second, as inferred from the area between the actual and ideal curves. This also support the conclusions from the case study.

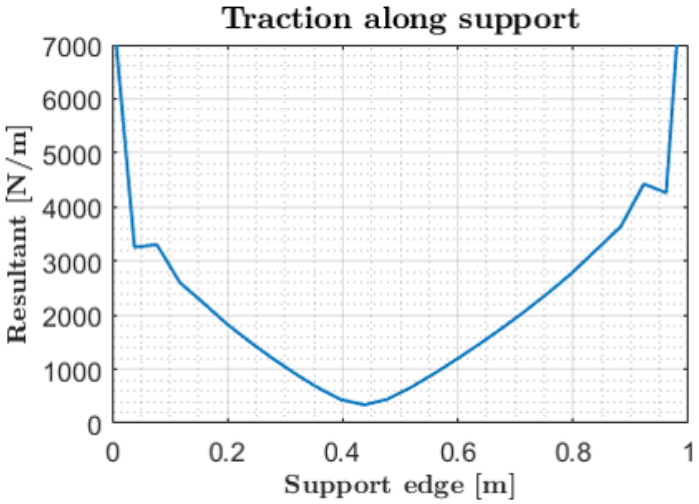


Figure 5.11: Reaction traction along fixed support

5.1.4 Distributed Loading at Multiple Interface

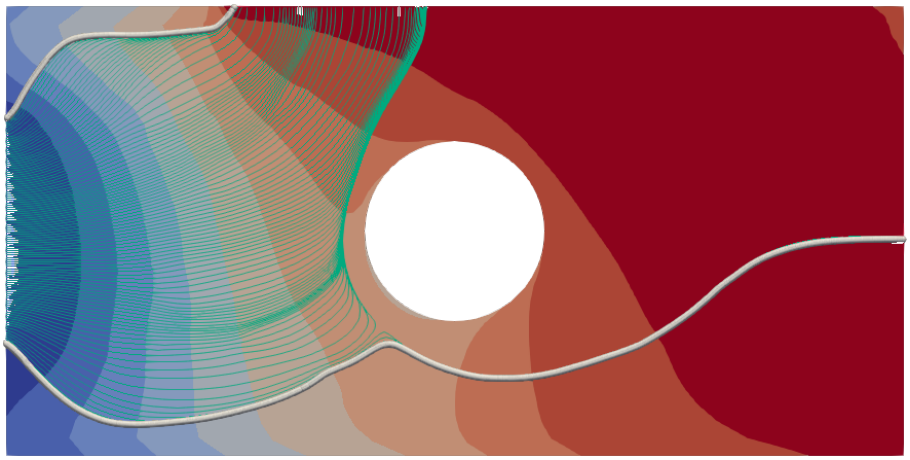


Figure 5.12: Load path in a plate with hole structure under single support and distributed loading at multiple interfaces

In order to learn how the load paths through the structure would vary when multiple fixed boundaries and multiple loading interfaces are present in the structure, a case study is done

on the plate with a hole structure shown in Figure 5.12. The nature of the loading and the fixtures are shown in Figure 4.7b.

The U^* index values are calculated for all the nodes present in the structure, and the contour of U^* values are plotted across the structure domain. The maximum U^* index values are present at the loading interfaces, and the minimum value of 0 is present at the supports or the fixed boundaries. The green-coloured streamlines represent the load paths propagating through the structure shown in Figure 5.12. The two solid white streamlines are the principal load paths of the system. These lines represent the stiffest path through the geometry and transfer the maximum load from the loading points to the support.

The first principal load path at the top of the support boundary has the highest reaction traction value, as shown in Figure 5.13. The second principal load path at the bottom of the support boundary has the second-highest reaction traction value. Based on the traction values, it can be concluded that the two principal load paths transfer the highest amount of loads to the supports.

The uniformity and continuity curves which could be used as a design guide for optimizing the structure, are plotted in Figure 5.14 and 5.15. In the case of the first principal load path, the uniformity curve indicates higher stiffness values initially followed by a decrease in stiffness midway through its trajectory as shown in Figure 5.14a. In the case of the second principal load path, the uniformity curve shows higher than optimum stiffness values throughout its trajectory as shown in Figure 5.15a, thus indicating the scope to optimize the structure by reducing the stiffness and consequently reducing the area between the real and the ideal curves. The continuity curves 5.14b 5.15b in both cases show significant curvature changes, which also need to be reduced in the context of optimization.

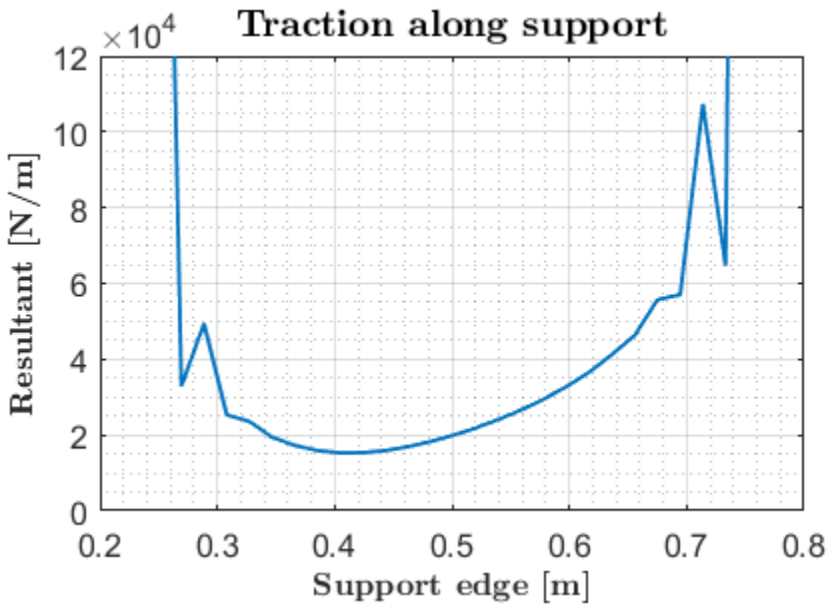
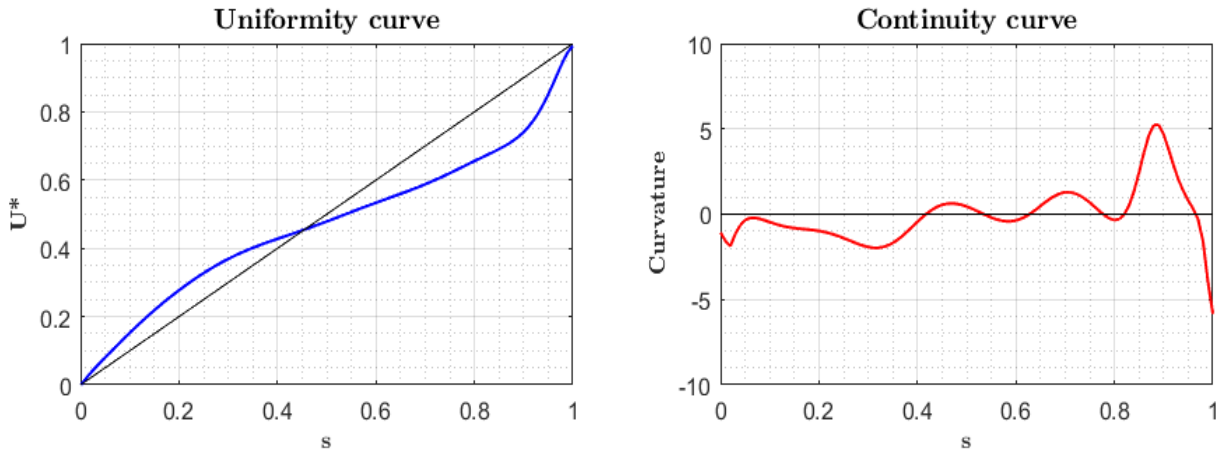
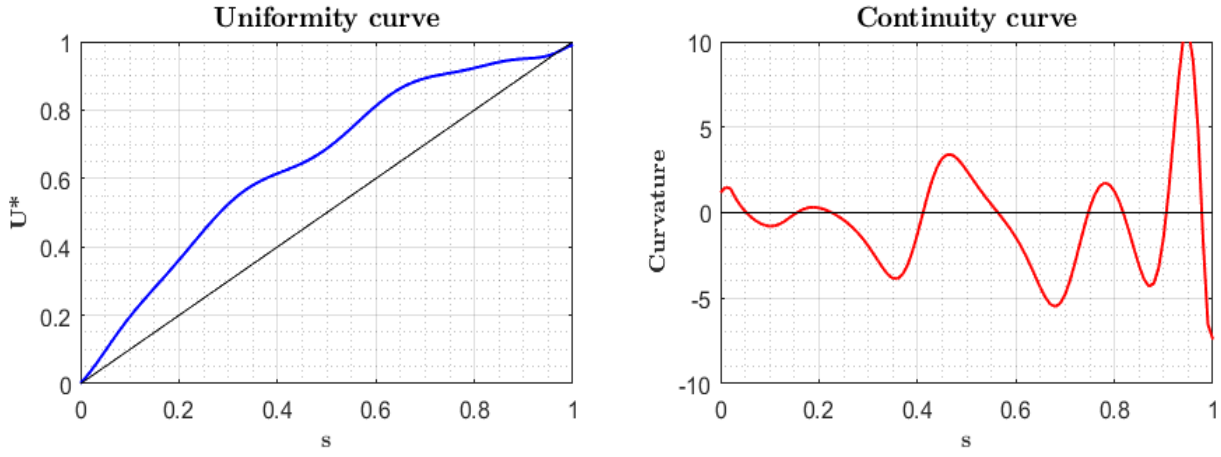


Figure 5.13: Reaction traction along the fixed boundary



(a) Uniformity Curve. (b) Continuity Curve.

Figure 5.14: Uniformity and Continuity curve for Principal Streamline 1



(a) Uniformity Curve. (b) Continuity Curve.

Figure 5.15: Uniformity and Continuity curve for Principal Streamline 2

5.1.5 Distributed Loading at Multiple Interface and with Multiple Constraints

In order to learn how the load paths through the structure would vary when multiple fixed boundaries and multiple loading interfaces are present in the structure, a case study is done on the plate with a hole structure shown in Figure 5.16. The nature of the loading and the fixtures are shown in Figure 4.8.

The streamline initiation point is of great importance. As per the concept of load transfer by the U^* index method, the load is transferred from the points of applied load all the way to the supports. If the streamlines are initiated from the loading boundary, a large number of initiation points have to be chosen to get all the streamlines that propagate to the supports. However, suppose the initiation points for the streamlines are chosen as the fixed boundary, which has the lowest U^* index value. In that case, all the streamlines thus generated propagate to the loading boundary, which gives a clear picture of how loads are transferred between the

loading points and the supports. Therefore the supports are chosen as the streamline initiation point.

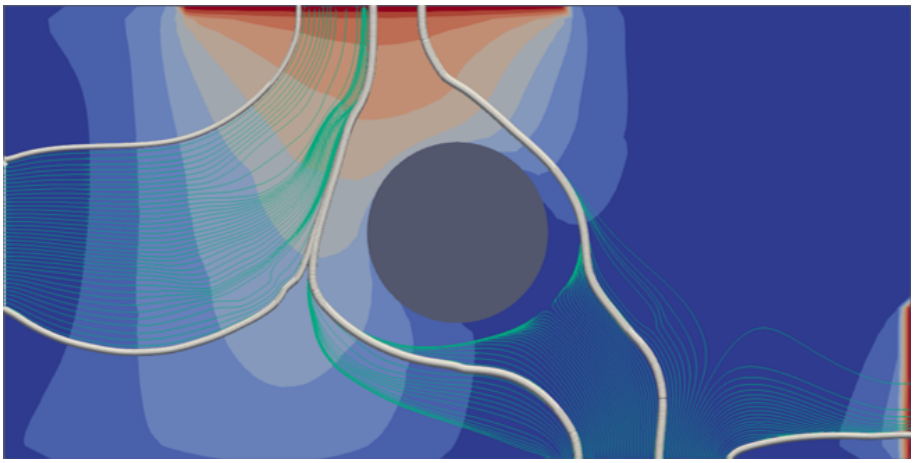


Figure 5.16: Load streamlines for multiple interface case.

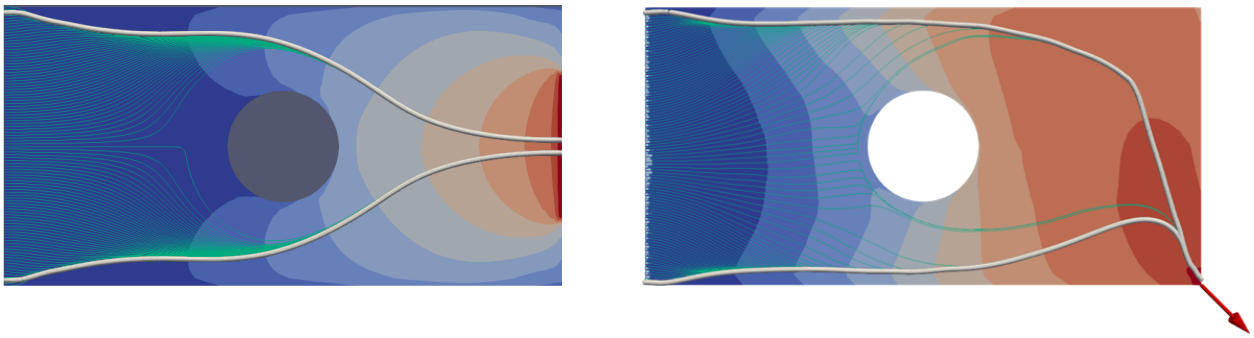
The load paths thus generated are depicted as green-coloured streamlines passing through the structure as shown in Figure 5.16. The solid white lines represent the principal streamlines of the system. However, it can be observed that the load streamlines from the top loading boundary propagate to both the supports, but the load streamlines from the right edge loading boundary do not travel to the fixed support at the left edge of the plate. This is because the streamlines do not intersect each other. Mathematically if the streamlines intersect, it would suggest that at the point of intersection, the U^* gradients will have two different values, which is not physically possible. Thus, this can be considered a limitation of the U^* index method. The load paths predicted in cases of multiple loading and multiple support conditions may not be accurate.

In order to further illustrate the limitation of the U^* index method, the net reaction forces at the supports in the x-axis and y-axis direction are compared in the table 1. It can be observed that a higher percentage of reaction force in the x-axis direction is generated at the left edge support even though no load paths from the loading boundary at the right edge go to that support. This indicates that the load paths predicted by U^* index method in cases with multiple loading and support boundary conditions could be misleading.

Total Reaction	Fx (N)	Fy (N)
Left Support	-6258	-12269
Bottom Support	-5742	-17731

Table 1: Total reaction force at the support boundaries of the plate.

5.1.6 Case study to validate load path analysis



(a) Study-1 Load Path (b) Study-2 Load Path

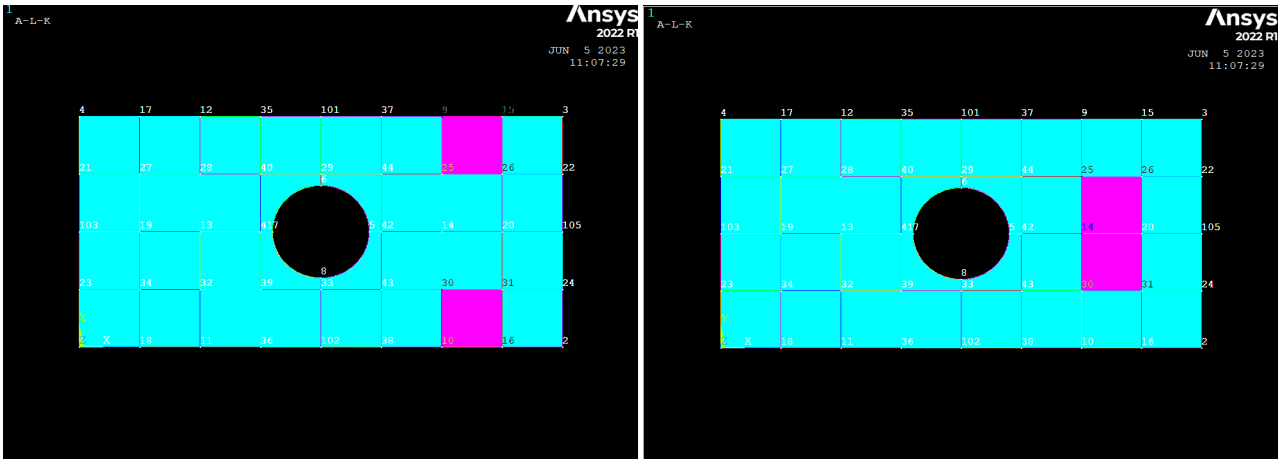
Figure 5.17: The principal load path for the two study cases.

Study − 1

Figure 5.17a shows the load paths through the plate structure for the boundary conditions established in Figure 4.9a. The load streamlines in Figure 5.17a are represented by the green lines initiated from the fixed boundary on the left edge of the plate to the loading boundary at the right edge. The principal load path representing the maximum transfer of load through the structure is highlighted in white solid lines shown in Figure 5.17a.

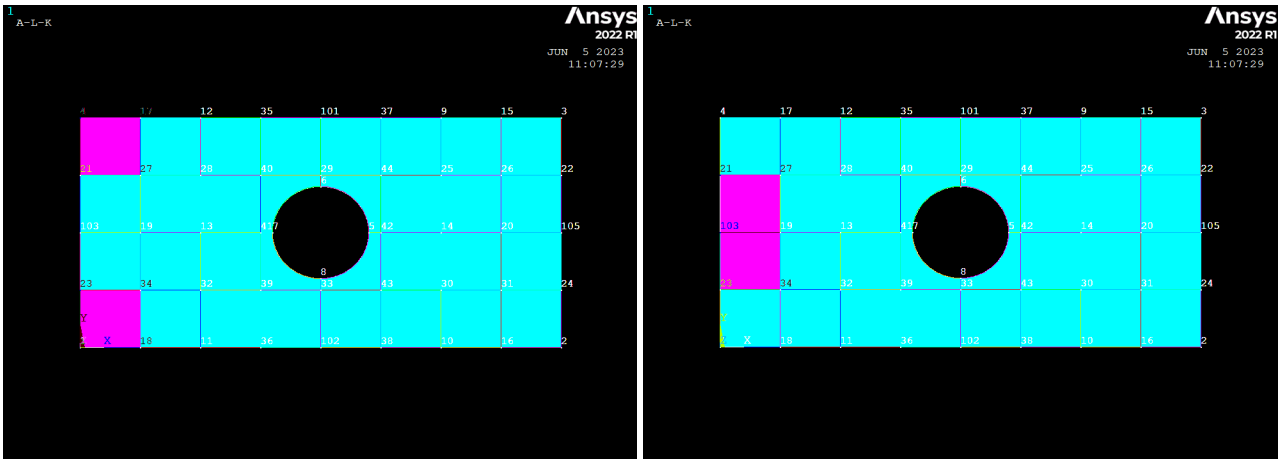
The plate domain is subjected to four different case studies. Figures 5.18a and 5.19a show the two cases where the highlighted areas lie on the principal load. Meanwhile, the other two cases, depicted in Figures 5.18b and 5.19b, show areas outside the primary load path.

The primary route that carries the maximum load in a structure is known as the principal load path. In order to minimize the bending or deformation in the system, it is most efficient to strengthen the specific areas of the plate that lie along the principal load path. Reinforcing a different region outside of it may not have the same impact on reducing deflection.



(a) Case-1 area selected. (b) Case-2 area selected.

Figure 5.18: Area chosen for two cases in study-1.



(a) Case-3 area selected.

(b) Case-4 area selected.

Figure 5.19: Area chosen for two cases in study-1.

All five load cases are analysed, and the maximum deflection induced in the system is tabulated as shown in the table 2. In the original case, the entire plate domain has a uniform thickness of 5mm. The maximum deflection induced in the system is 0.52 mm.

Figure 5.18a displays Case-1, where certain areas are highlighted with a thickness of 10mm while the rest of the domain has a thickness of 5mm. The highlighted area in case-1 does not pass through the principal load path. This increase in thickness in specific area segments significantly enhances the overall stiffness of the structure, and the system's maximum deflection is reduced from 0.52mm to 0.509mm. There is a 2.1% decrease in the overall deflection.

In Figure 5.18b, the highlighted area passes through the principal load path and has a 10mm thickness. The remainder of the domain has a 5mm thickness. The maximum deflection produced in the system is reduced from 0.52mm to 0.494mm. The reduction in the overall deflection is 5%. It is evident that reinforcing areas through which the principal load path passes significantly improve the system's stiffness more than reinforcing other regions.

Sl.No.	Case	Max. Deflection (mm)
1	Original	0.520
2	Case-1	0.509
3	Case-2	0.494
4	Case-3	0.505
5	Case-4	0.510

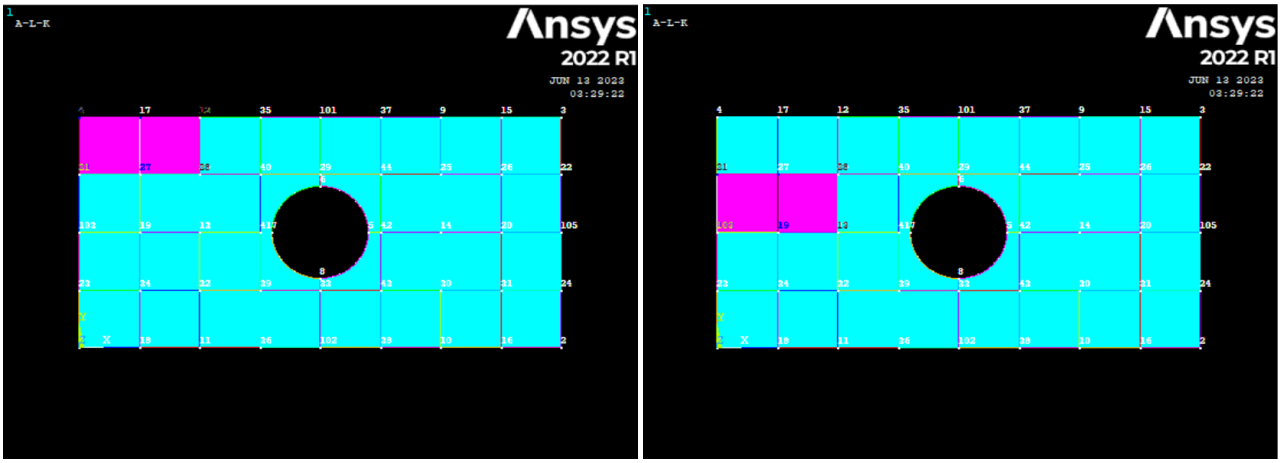
Table 2: Maximum deflection induced for different cases in study-1.

In Figure 5.19a, Case-3 displays highlighted areas that are 10mm thick, while the rest of the domain measures 5mm in thickness. These areas are situated in the region where the principal load path passes, resulting in a significant increase in stiffness for the entire structure. The maximum deflection of the system is reduced from 0.52mm to 0.505mm, representing a 2.9% decrease in overall deflection. On the other hand, Case-4, depicted in Figure 5.19b, features reinforced areas outside the principal load path. This reduces the maximum deflection from 0.52mm to 0.51mm, a 1.9% reduction in the overall value.

An important observation here is that the principal load paths are the stiffest path through the structure. The maximum deflection produced in the system for the same loading conditions is significantly reduced when the stiffness along the principal load path is increased. This implies that principal load paths could be used as a visual tool to optimize structures to achieve desirable mechanical properties.

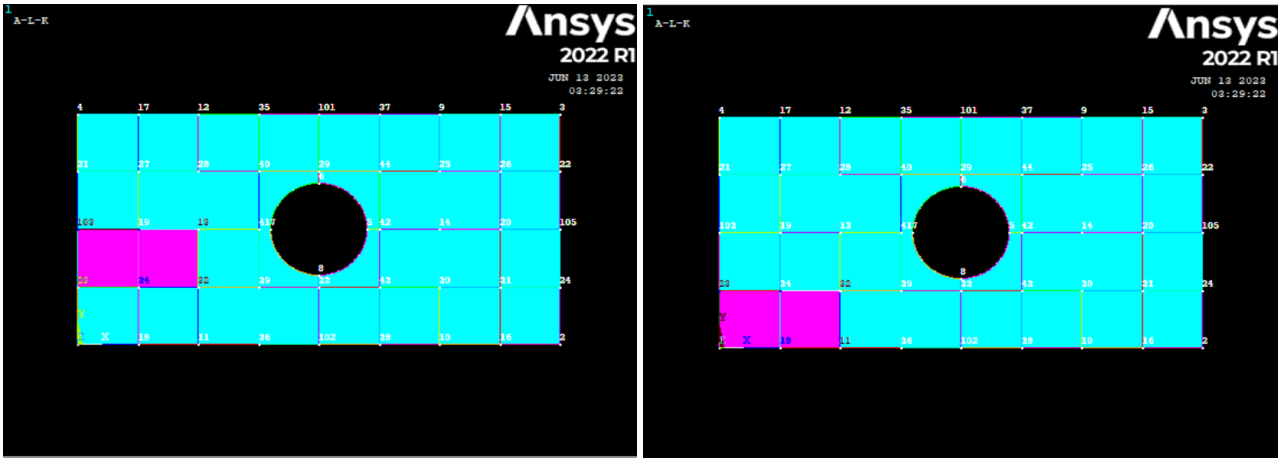
Study – 2

Four different cases are analysed in study-2 as well, shown in Figure 5.20 and 5.21. In case-1 and case-4, the highlighted areas pass through the principal load path, whereas the areas in case-2 and case-3 are not in the principal load path. The highlighted areas in all these cases are reinforced by altering the plate thickness from 5mm to 10mm. The maximum deflection induced in the structure is tabulated as shown 3. Based on deflection values in each case, it is evident that reinforcing areas through which the principal load path passes result in a much stiffer geometry. case-1 shows the lowest deflection value and is 12.26% lower than the original plate, followed by case-4 with a reduction of 9% in maximum deflection. Cases 2 and 3 show very close deflection values with reductions of 2.6% and 1.9% in displacement, respectively.



(a) case 1 for study-2 (b) case 2 for study-2

Figure 5.20: Area chosen for two cases in study-2.



(a) case 1 for study-2 (b) case 2 for study-2

Figure 5.21: Area chosen for two cases in study-2.

Sl.No.	Case	Max. Deflection (mm)
1	Original	0.0155
2	Case-1	0.0136
3	Case-2	0.0151
4	Case-3	0.0152
5	Case-4	0.0141

Table 3: Maximum deflection induced for different cases in study-2.

The two studies on the plate structure subjected to different loading conditions indicate that the stiffest path in the system is the principal load path, which carries a significant portion of the load transfer. By reinforcing the region through which the principal load path passes, the stiffness of the structure is greatly enhanced, resulting in a reduction in overall deflection. However, the stiffening effect is less efficient when reinforcements are added outside the principal load path. Understanding the load paths through the structure offers valuable insights into the distribution of stiffness within the system. While more research into the concept of load path using the U^* index theory is needed to make it practical, these load paths generated using the theory could provide a means to modify the stiffness of the structure in strategic locations to enhance overall stiffness without using more complex optimization methods like topology optimization.

5.2 Load Path Visualization in 3D Structure

The load path analysis is performed on complex 3D structures. The geometry chosen for the analysis is the Electric Fan Thruster (EleFanT).

5.2.1 Load Case: Multiple Loading Interface

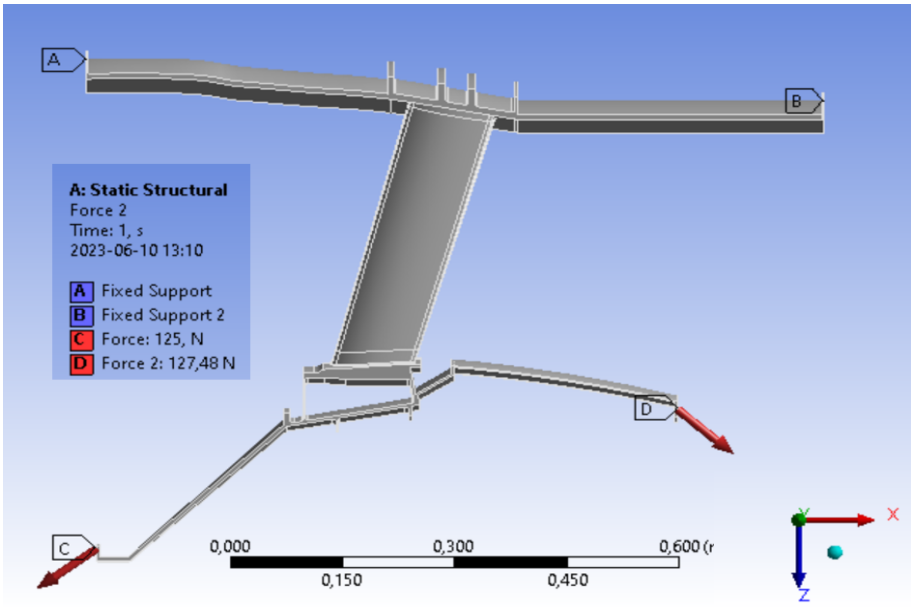


Figure 5.22: The boundary conditions applied on the EleFanT geometry.

The boundary conditions applied to the geometry are shown in Figure 5.22. Faces A and B are fixed. A force of 100N in the negative x-axis direction and 75N in the z-direction is distributed across face C. Meanwhile, A force of 95N in the x-axis direction and 85N in the z-direction is distributed across face D.

The U^* contours obtained from the load paths analysis are shown in Figure 5.23. The identification and plotting of streamlines in this case is done on the surface of the 3D structure, as mentioned already in section 4.2.4 of Methodology. This is due to the tendency of the streamlines to halt at surfaces of thin-walled structures, and could be attributed to the direction of gradients not aligned with the plane of membrane. The U^* index peaks at the loading boundaries, which can be identified by the red regions. On the other hand, the fixed faces are represented by the blue regions and have the lowest U^* index values. According to the U^* index theory, the load paths seek the highest value of U^* gradient when initiated from the support boundaries. The load path analysis focuses on studying how the loads are transferred from the loading point to the supports; hence in this case study, the load path initiation points are taken at the supports. This would allow the streamlines to propagate through the structure by seeking the highest U^* gradient in the immediate vicinity.

In the case of complex geometry such as the one used in the current analysis, the ParaView software stream tracer function is not able to generate the entire streamline path in one single go. The streamlines from the supports propagate a certain distance and stop. It is observed that the load streamlines come to a halt, usually when there are sudden direction changes in the geometry. This problem is resolved to a significant extent by initiating the streamlines again from the tail end of the previous streamline.

However, in certain cases, the re-initiation of streamlines from the tail of the previous stream-

line also proves to be ineffective. One such observed case is when the streamline travels to the center of a contour line and halts. Oftentimes, the contours generated by the U^* index method in such geometries are complex and involve local extremum points, represented by circular contour lines. In such extremum points, the gradient is not unique and has the same magnitude of change in all directions. One such instance of local maxima is shown in figure 5.25 where the streamlines do not propagate beyond the circular contour line, although higher U^* values can be observed on the guide veins. One possibility is that the algorithm in ParaView is not equipped handle such complex contours, in which case an alternate platform is required. It could also be possible that this is one of the shortcomings of the U^* index method, and further research is required in this field to shed light on this issue.

Figure 5.24b shows the uniformity plot of the streamline from the point of support to the point of loading. It highlights the different regions through which the streamline travels through the structure. It can be seen that the uniformity curve lies well below the ideal line for a significant part of the path from the point of support. The curve then experiences sudden change in U^* values followed by a smooth curve again. This is indicative of a very unevenly distributed stiffness in the structure with respect to the current loading configuration. The steep rise in the uniformity curve, denoted by the solid black line, occurs at the highlighted region in figure 5.24a. This region could therefore be inferred as the stiffest part of the structure.

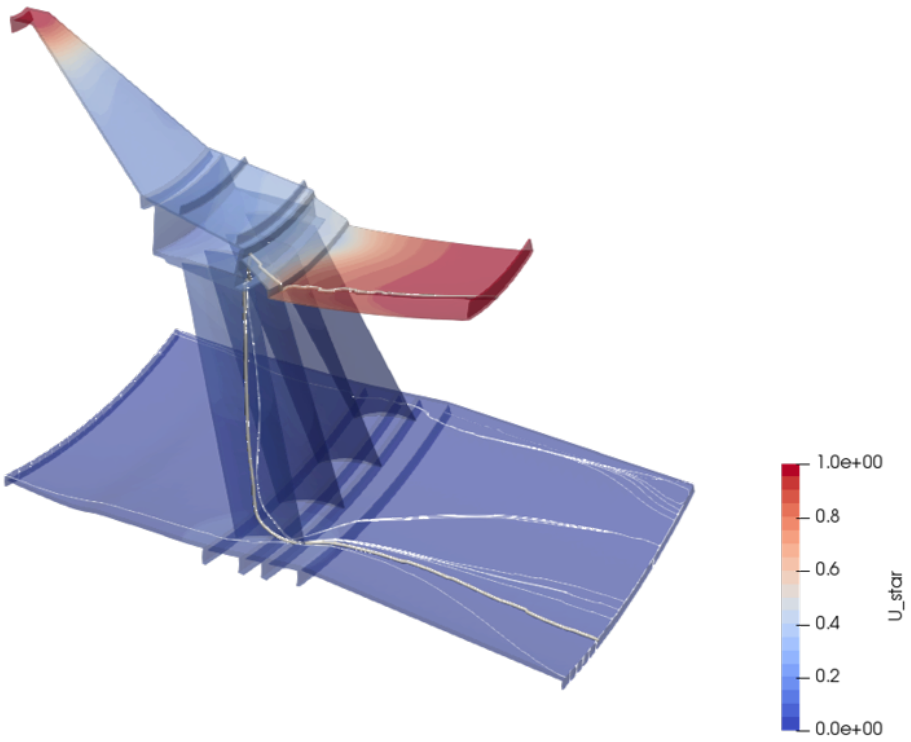
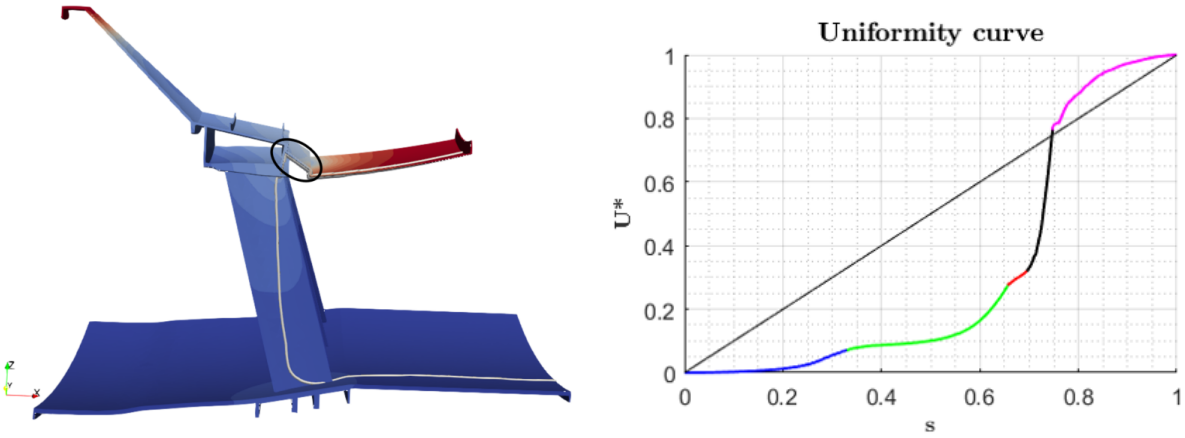


Figure 5.23: Streamlines through the 3D Structure



(a) Load path in the 3D structure (b) Uniformity curve of the load path

Figure 5.24: Uniformity curve and corresponding regions in the structure

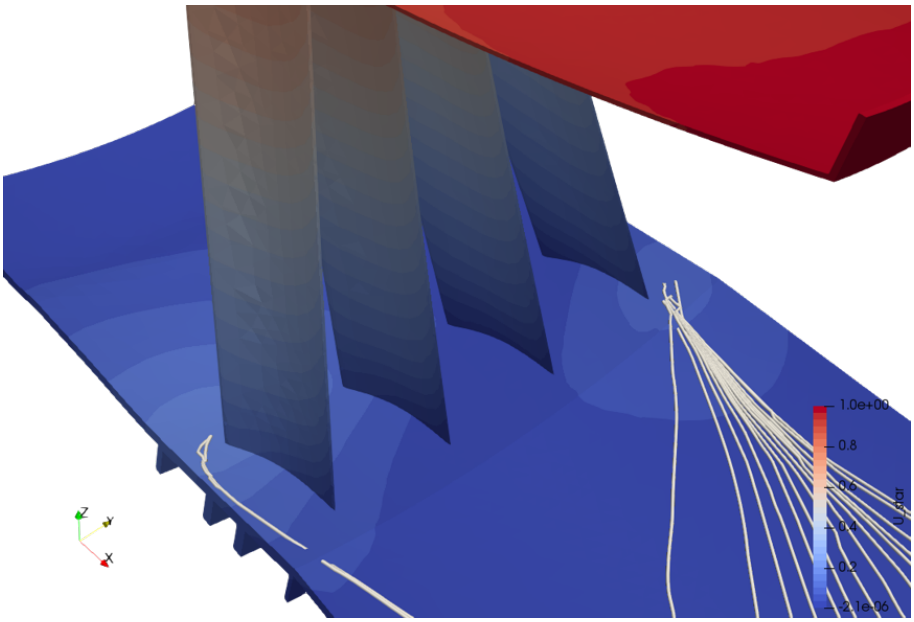


Figure 5.25: Local maxima in U^* contour

6 Conclusion

The specific aims of the current thesis work have been successfully implemented. The results obtained are explained and discussed in detail. The use of the U^* index method to visualize the load paths through a structure has the advantage that it does not rely on the stress induced in the system. Thus stress concentrations do not influence the load paths.

The load path visualization on 2D structures reveals several vital observations. Due to the simple nature of the geometry, the number of elements present in the mesh is significantly less, and the computation time is less compared to complex 3D structures. The cases studied in the current work highlight how the load paths are influenced by the loading and boundary conditions. When compared with the principal load path, the traction magnitude shows that the highest reaction happens in close proximity to the area where the principal load path meets the supports. The validation study further consolidates that the principal load paths represent the paths of high relative stiffness in the structure. When such load trajectories are reinforced, the overall stiffness of the geometry is enhanced, as evidenced by the lower deflection produced in the system.

The load path visualization on 3D structures poses several challenges. Due to the complex nature of the geometry, a relatively large number of elements are required to represent the structure for finite element analysis. The U^* index method is computationally expensive. The engine structure analyzed in the current work requires an overall runtime of 14hrs in the solution phase. The inspection method shows significant time reduction and has the advantage that if the loading and support boundaries remain the same, the geometry can be analyzed for different load cases by extracting the stiffness matrix of the finite element system from the initial analysis run. Thus the subsequent load cases can be analyzed quickly.

The load path visualization in ParaView for complex structures is challenging. The sudden geometric changes in the structure make the visualization routine difficult. Since the functions built into ParaView are not aimed explicitly at visualizing load paths, the sudden geometric changes make the streamlines stop in their path abruptly. The streamlines need to be initiated multiple times to enable them to propagate from the supports to the loading boundary. In some cases, the load path is only partially generated despite multiple initiations. This is undoubtedly a limitation of the post-processing routine, and a better alternative needs to be explored, which might give the user better control over the load path initiation process and the trajectory seeking based on the U^* gradients.

Another observation based on the case studies done in the current work is that multiple loading interfaces cannot be reliably predicted using the U^* index method. This is mainly because the U^* index method, by definition, assigns the maximum value to the loading boundaries and does not take into account the magnitude or direction of the loading. The load paths generated for multiple loading interface cases could be misleading. This problem could be overcome by assigning the loads one at a time and superimposing the generated streamlines to get a general understanding of the load transfer. This approach certainly needs further investigation and is out of the scope of the current thesis work.

7 Future Work

Due to the time constraints involved in the current thesis work, certain studies could not be executed successfully. These can be implemented and further explored in future work.

1. All the cases studied in the present work do not consider the rotational degrees of freedom of nodes. Using shell elements with all six degrees of freedom could potentially make the computation less expensive and resolve the problem of load paths not travelling through the structure. In this case, the mid-surface geometry of the structure could be used, which is much less complex than the whole 3D geometry.

2. The current analysis is limited to elastic isotropic materials. Research has been done to use the U^* index method on orthotropic materials, which could expand the use of load path visualization to composite materials.

3. The current work has not explored the practical applicability of uniformity and continuity plots. These plots could potentially be used for topological optimization. This new avenue needs to be further explored in future work.

4. Other post processing software need to be explored to overcome the challenges faced in complex geometric structures where the load paths do not propagate through the entire geometry. A better post processing routine that enable fine control over the propagation of streamlines and that can account for the sudden geometric changes in the structure.

References

[1] S Zhao, N Wu, and Q Wang. "Novel Damage Detection Tool Based on Load Path Analysis Using Ustar (U*)." In: *IEEE Access, Access, IEEE* 8 (2020), pp. 82607–82616. ISSN: 2169-3536. URL: https://search.ebscohost.com/login.aspx?direct=true&db=edseee&AN=edseee.9076083&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[2] Kun Marhadi and Satchi Venkataraman. "Comparison of quantitative and qualitative information provided by different structural load path definitions". In: *International Journal for Simulation and Multidisciplinary Design Optimization* 3.3 (2009), pp. 384–400.

[3] Ali Y. Tamijani et al. "Load paths visualization in plane elasticity using load function method". In: *International Journal of Solids and Structures* 135 (2018), pp. 99–109. ISSN: 0020-7683. DOI: https://doi.org/10.1016/j.ijsolstr.2017.11.013.

[4] Manabu Shinobu et al. "Transferred load and its course in passenger car bodies". In: *JSAE Review* 16.2 (1995), pp. 145–150. ISSN: 0389-4304. DOI: https://doi.org/10.1016/0389-4304(95)00010-5. URL: https://www.sciencedirect.com/science/article/pii/0389430495000105.

[5] Hiroaki Hoshino, Toshiaki Sakurai, and Kunihiro Takahashi. "Vibration reduction in the cabins of heavy-duty trucks using the theory of load transfer paths". In: *JSAE Review* 24.2 (2003), pp. 165–171. ISSN: 0389-4304. DOI: https://doi.org/10.1016/S0389-4304(03)00005-5. URL: https://www.sciencedirect.com/science/article/pii/S0389430403000055.

[6] Qingguo Wang et al. "High efficient load paths analysis with U* index generated by deep learning". In: *Computer Methods in Applied Mechanics and Engineering* 344 (2019), pp. 499–511. ISSN: 0045-7825. DOI: https://doi.org/10.1016/j.cma.2018.10.012. URL: https://www.sciencedirect.com/science/article/pii/S0045782518305097.

[7] Khashayar Pejhan et al. "Extension of U* Index Theory to Nonlinear Case of Load Transfer Analysis". In: Nov. 2015, V009T12A008. DOI: 10.1115/IMECE2015-51175.

[8] Shengjie Zhao et al. "Load Path Visualization Using U* Index and Principal Load Path Determination in Thin-Walled Structures". In: *Facta Universitatis Series Mechanical Engineering* (Jan. 2022). DOI: 10.22190/FUME211105006Z.

[9] Yasuhisa Okano et al. "Load Path Analysis of Vehicle Body Structures under Eigenmode Deformation of Bending Vibration". In: *SAE Technical Paper 2009-01-0770* (2009). DOI: https://doi.org/10.4271/2009-01-0770.

[10] Zhanguang Zheng et al. "Multiaxial fatigue life prediction of metals considering loading paths by image recognition and machine learning". In: *Engineering Failure Analysis* 143 (2023), p. 106851. ISSN: 1350-6307. DOI: https://doi.org/10.1016/j.engfailanal.2022.106851. URL: https://www.sciencedirect.com/science/article/pii/S1350630722008184.

[11] Qing-Yun Deng et al. "Load path sensitivity and multiaxial fatigue life prediction of metals under non-proportional loadings". In: *International Journal of Fatigue* 166 (2023), p. 107281. ISSN: 0142-1123. DOI: https://doi.org/10.1016/j.ijfatigue.2022.107281.

[12] Shengjie Zhao et al. "Design and optimization of graded lattice structures with load path-oriented reinforcement". In: *Materials Design* 227 (2023), p. 111776. ISSN: 0264-1275. DOI: https://doi.org/10.1016/j.matdes.2023.111776.

[13] Tadashi Naito, Hirokazu Kobayashi, and Yuta Urushiyama. "Application of Load Path Index U* for Evaluation of Sheet Steel Joint with Spot Welds". In: *SAE Technical Paper 2012-01-0534* (2012). DOI: https://doi.org/10.4271/2012-01-0534.

[14] Shengjie Zhao et al. "LOAD PATH VISUALIZATION USING U* INDEX AND PRINCIPAL LOAD PATH DETERMINATION IN THIN-WALLED STRUCTURES". In: *Facta Universitatis Series Mechanical Engineering* (Jan. 2022). DOI: 10.22190/FUME211105006Z.

[15] Rajesh Ramesh. "Load Path Visualization in Aero-engine Structures Using U* Index Method". In: *(Dissertation)* (2021), p. 68. URL: https://hdl.handle.net/20.500.12380/302575.

[16] ANSYS Inc. "ANSYS Mechanical APDL Documentation. Version 2022 R1". In: *ANSYS Inc.* (2022).

[17] Oscar Johansson and Jonathan Muistama. "Development Integration of Load Path Visualization With the U* Index Method : Applications In Aerospace Product Development". In: *(Dissertation)* (2022), p. 104. URL: http://urn.kb.se/resolve?urn=urn:nbn:se:bth-22988.

[18] Toshiaki Sakurai et al. "Reduction of Calculation Time for Load Path U* Analysis of Structures". In: *Journal of Solid Mechanics and Materials Engineering* 1.11 (2007), pp. 1322–1330. DOI: http://dx.doi.org/10.1299/jmmp.1.1322.

[19] Toshiaki Sakurai et al. "Load Path U* Analysis of Structures under Multiple Loading Conditions". In: *Transactions of the Japan Society of Mechanical Engineers Series A* 73 (Feb. 2007), pp. 195–200. DOI: 10.1299/kikaia.73.195.

[20] Niels Saabye Ottosen and Hans Petersson. *Introduction to the finite element method.* Prentice Hall, 1992. ISBN: 0134738772. URL: https://search.ebscohost.com/login.aspx?direct=true&db=cat09075a&AN=clpc.oai.edge.chalmers.folio.ebsco.com.fs00001000.629a4b00.0b55.4dbd.a9aa.508c7b47d930&site=eds-live&scope=site&authtype=guest&custid=s3911979&groupid=main&profile=eds.

[21] Enyang Wang et al. "Load Transfer in Truck Cab Structures under Initial Phase of Frontal Collision". In: *Int. J. Vehicle Structures Systems* 2.2 (2010), pp. 60–68. DOI: http://dx.doi.org/10.4273/ijvss.2.2.03.

[22] Masato Kamimura et al. "Structural Optimization and Load Paths". In: *Transactions of the Japan Society of Mechanical Engineers Series A* 74 (Jan. 2008), pp. 6–12. DOI: 10.1299/kikaia.74.6.

[23] R. Smith and R. Minton. *Calculus.* McGraw-Hill Publishing, 2011. ISBN: 9780077418670. URL: https://books.google.se/books?id=rAVLAgAAQBAJ.

[24] Will Schroeder, Ken Martin, and Bill Lorensen. *The Visualization Toolkit (4th ed.)* Kitware, 2006. ISBN: 978-1-930934-19-1.

8 Appendix

8.0.1 APDL Script for K_{AC} extraction - Inspection Load Method

```
/NOPR   ! Suppress printing of UNDO process
/PMACRO ! Echo following commands to log
FINISH  ! Make sure we are at BEGIN level
/CLEAR,NOSTART  ! Clear model since no SAVE found

!/FILNAME,Plate_with_Hole_Dist_Insp,0
!/CWD,'P:\nobackup\KBEIMP\Ram&Santhosh\Plate_With_Hole\Trial13'
/TITLE,Plate_with_Hole_Dist_Insp

CDREAD,DB, /project/nobackup/KBEIMP/Ram_Santhosh/ElefantTrial1/Elefant, cdb
DOF_PER_NODE = 2    ! 2D problem
/REPLOT

!--------------------------------------------------------------------------


! DEFINE NODE SETS
NSEL, ALL
CM, TOTNODES, NODE
NSEL, U,,, FIXNODES
NSEL, U,,, LOADNODES
CM, FREENODES, NODE

!--------------------------------------------------------------------------

! GET THE NODE COUNTS
NSEL, S,,, FIXNODES
*GET, NUM_FIX_NODES, NODE, 0, COUNT
NSEL,S,,, LOADNODES
*GET, NUM_LOAD_NODES, NODE, 0, COUNT
NSEL,S,,, FREENODES
*GET, NUM_FREE_NODES, NODE, 0, COUNT
NSEL,S,,, TOTNODES
*GET, NUM_TOT_NODES, NODE, 0, COUNT

!--------------------------------------------------------------------------

! EXTRACT NODE NUMBERS IN A VECTOR
NSEL, S,,, FIXNODES
*VGET, FIX_NODES_LIST, NODE, 0 ,NLIST
NSEL,S,,, LOADNODES
*VGET, LOAD_NODES_LIST, NODE, 0 ,NLIST
```

```
NSEL,S,,, FREENODES
*VGET, FREE_NODES_LIST, NODE, 0 ,NLIST

!-------------------- Apply original boundary conditions --------------------

/PREP7   ! Enter the preprocessor
ALLSEL
D,FIXNODES,ALL,ALL                          ! All dofs of the fix nodes to zero
F,LOADNODES,FX,10000

!--------------------------- Solve the problem ---------------------------
/SOLU ! Enter the solution
SOLVE

!---------------------- Calculate the strain Energy ----------------------

/POST1
SET,
/POST26
ENERSOL,2,SENE,,STRAINENERGY

!kommer från Time history variables

FILE,,   !kan nog behöva ändra här
/UI,COLL,1
NUMVAR,200
SOLU,191,NCMIT
STORE,MERGE
FILLDATA,191,,,,1,1
REALVAR,191,191

*GET,U_VALUE,VARI,2, REAL,1  ! Store the Strain energy value

!Exporting U-value
*cfopen,U_value.csv
*VWRITE,U_VALUE
(F15.13)
*cfclose
!--------------STORING THE DISPALECMENT d_a AND d_c------------------

*DIM, LOADNODE_DISP,ARRAY,NUM_LOAD_NODES,2
*DO, I, 1,NUM_LOAD_NODES
    *GET,LOADNODE_DISP(I,1),NODE,LOAD_NODES_LIST(I),U,X
    *GET,LOADNODE_DISP(I,2),NODE,LOAD_NODES_LIST(I),U,Y
*ENDDO
```

```
*DIM, FREENODE_DISP,ARRAY,NUM_FREE_NODES,2
*DO, I, 1,NUM_FREE_NODES
    *GET,FREENODE_DISP(I,1),NODE,FREE_NODES_LIST(I),U,X
    *GET,FREENODE_DISP(I,2),NODE,FREE_NODES_LIST(I),U,Y
*ENDDO

*MWRITE,LOADNODE_DISP,A_da.csv                    ! COMMA ISSUE RESOLVED
(F18.14,F18.14)

*MWRITE,FREENODE_DISP,A_dc.csv                    ! COMMA ISSUE RESOLVED
(F18.14,F18.14)

!---------------------- Do the looping ----------------------
/PREP7 ! Enter the processorer

!Fixing the loading point and and supporting points
ALLSEL
FDELE,ALL
D,FIXNODES,ALL,ALL
D,LOADNODES,ALL,ALL


*DO,I,1,NUM_FREE_NODES*2
    ALLSEL
    *IF,I,LE,NUM_FREE_NODES,THEN
        F,FREE_NODES_LIST(I),FX,-15000!
        F,FREE_NODES_LIST(I),FY,-20000!
    *ELSEIF,I,GT,NUM_FREE_NODES
        F,FREE_NODES_LIST(I-NUM_FREE_NODES),FX,-17500!
        F,FREE_NODES_LIST(I-NUM_FREE_NODES),FY,-22500!
    *ENDIF

    ALLSEL
    LSWRITE,I        ! Write loadstep
    FDELE,ALL !DELETING ALL FORCES FOR NEW LOOP
*ENDDO


!-------------------- SOLVE ALL THE SETS ------------------------
/SOLU
ALLSEL
LSSOLVE,1,NUM_FREE_NODES*2 !solving each load step


!----------------- CALCULATE STRAIN ENERGY FOR ALL SET ---------------------
```

```
!DET ÄR HÄR JAG VÄLJER REAKTIONKRAFTERNA


/post1
SET,
*DIM, DC,ARRAY,NUM_FREE_NODES,4
*DO,I,1,NUM_FREE_NODES*2

    *IF,I,LE,NUM_FREE_NODES,THEN
        SET,,, ,,, ,I
        *GET,DC(I,1),NODE,FREE_NODES_LIST(I),U,X
        *GET,DC(I,2),NODE,FREE_NODES_LIST(I),U,Y
    *ELSEIF,I,GT,NUM_FREE_NODES
        SET,,, ,,, ,I
        *GET,DC(I-NUM_FREE_NODES,3),NODE,FREE_NODES_LIST(I-NUM_FREE_NODES),U,X
        *GET,DC(I-NUM_FREE_NODES,4),NODE,FREE_NODES_LIST(I-NUM_FREE_NODES),U,Y
    *ENDIF

*ENDDO

*MWRITE,DC,AA_DC.csv                    ! COMMA ISSUE RESOLVED
(F18.14,F18.14,F18.14,F18.14)

!------------------------------------------------------------------------

/POST26
!!!!!!!!!!!!!Samma problem som för 2 dof kan inte byta namn i varje loop
!när jag exporterar datan
FILE,,
/UI,COLL,1
NUMVAR,200
SOLU,191,NCMIT
STORE,MERGE
FILLDATA,191,,,,1,1
REALVAR,191,191


! -------------- EXTRACTING REACTION FORCES at node set 'A' --------------

! Extract x component of reaction forces
*DIM,RF_X,ARRAY,NUM_FREE_NODES*DOF_PER_NODE,NUM_LOAD_NODES

*DO,I,1,NUM_LOAD_NODES

    RFORCE,2,LOAD_NODES_LIST(I),F,X,FX_2
```

```
    STORE,MERGE

    VGET,RF_X(1,I),2

*ENDDO


! Extract y component of reaction forces
*DIM,RF_Y,ARRAY,NUM_FREE_NODES*DOF_PER_NODE,NUM_LOAD_NODES

*DO,I,1,NUM_LOAD_NODES

    RFORCE,3,LOAD_NODES_LIST(I),F,Y,FY_3
    STORE,MERGE

    VGET,RF_Y(1,I),3

*ENDDO

!------------------------------------------------------------------------------

*MWRITE,FREE_NODES_LIST,A_freenode_number,txt ! FOR U* COMPUTATION IN MATLAB
(F10.0)


! -------------------------- PRINT REACTION FORCES ------------------------

! Save time history variables to file
*MWRITE,RF_X,A_reaction_x,txt
(F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,...
F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4)

! Save time history variables to file
*MWRITE,RF_Y,A_reaction_y,txt
(F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,...
F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4,F15.4)


!---------------------------- DATA PROCESSING ----------------------------


ALLSEL
*GET,ELEM_NO_MIN,ELEM,0,NUM,MIN
*GET,ELEM_NO_MAX,ELEM,0,NUM,MAX
*GET,ELEM_COUNT,ELEM,0,COUNT
```

```
*DIM,PA,ARRAY,2*NUM_LOAD_NODES,2
*DIM,DC_EACH_FREE_NODE,ARRAY,2,2


! each column of K_AC_FREENODES forms K_AC for a freenode C
*DMAT,K_AC_FREENODES,D,ALLOC,NUM_LOAD_NODES*DOF_PER_NODE**2,NUM_FREE_NODES


*DO,I,1,NUM_FREE_NODES

    *DO,J,1,NUM_LOAD_NODES

        ! 1st load step for each node
        *SET,PA(2*J-1,1),RF_X(I,J)
        *SET,PA(2*J,1)  ,RF_Y(I,J)

        ! 2nd load step for each node
        *SET,PA(2*J-1,2),RF_X(NUM_FREE_NODES+I,J)
        *SET,PA(2*J,2)  ,RF_Y(NUM_FREE_NODES+I,J)

    *ENDDO

    *SET,DC_EACH_FREE_NODE(1,1),DC(I,1)
    *SET,DC_EACH_FREE_NODE(1,2),DC(I,3)
    *SET,DC_EACH_FREE_NODE(2,1),DC(I,2)
    *SET,DC_EACH_FREE_NODE(2,2),DC(I,4)

    *MOPER,INVDC,DC_EACH_FREE_NODE,INVERT
    *MOPER,K_AC,PA,MULT,INVDC

    *DO,J,1,DOF_PER_NODE
        *DO,K,1,NUM_LOAD_NODES*DOF_PER_NODE
            ROWIND = (J-1)*NUM_LOAD_NODES*DOF_PER_NODE+K
            K_AC_FREENODES(ROWIND,I) = K_AC(K,J)
        *ENDDO
    *ENDDO

*ENDDO


SAVE ! SAVES DATABASE TO JOBNAME.DB
```

```
!-------------------- EXPORTING DATA ---------------------
*cfopen,element_type.txt
! list element types
*GET,net,ETYP,1,NUM,COUNT
*DO,i,1,net
    *GET,ETname,ETYP,i,ATTR,ENAM
    *VWRITE,'ET ',i,' ',ETname
    (A3, F4.0, A1, F4.0)
*ENDDO
*cfclose

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! format output for nodes, elements and stress listings
/PAGE, 1E9,, 1E9,,  ! disable headers
/FORMAT, , ,14,5, , ! fix floating point format
/HEADER, off, off, off, off, on, off ! disable summaries

/POST26
/OUTPUT,NLIST,txt
NLIST,,,,COORD ! print nodes w. coordinates
/OUTPUT

/OUTPUT,ELIST,txt
ELIST ! print element connectivity table
/OUTPUT

!/POST1
!/OUTPUT,DISP,txt
!PRNSOL,U ! print all displacements
!/OUTPUT

! save K_AC matrices
*EXPORT,K_AC_FREENODES,MAT,K_AC_FREENODES

/DELETE,,rst
```

8.0.2 APDL Script for U^* computation - Inspection Load Method

```
/NOPR   ! Suppress printing of UNDO process
/PMACRO ! Echo following commands to log
FINISH  ! Make sure we are at BEGIN level
/CLEAR,NOSTART  ! Clear model since no SAVE found

!/FILNAME,Tet_Elefant,0
!/CWD,'P:\nobackup\KBEIMP\Ram&Santhosh\Plate_With_Hole\Trial13'
/TITLE,Tet_Elefant

!*DIM,CDBFILE,STRING,32
!*DIM,PWD,STRING,150
!*DIM,FULLPATH,STRING,150
!/INQUIRE,PWD,DIRECTORY
!CDBFILE(1) = 'Plate_with_Hole_distributed'
!*SET,FULLPATH(1),'%PWD(1)%/%CDBFILE(1)%'

!CDREAD,DB,'%FULLPATH(1)%',CDB
CDREAD,DB, <path>/Elefant, cdb
DOF_PER_NODE = 3    ! 3D problem
/REPLOT

!------------------------------------------------------------------------

! DEFINE NODE SETS
NSEL, ALL
CM, TOTNODES, NODE
NSEL, U,,, FFACE_1
NSEL, U,,, FFACE_2
NSEL, U,,, LFACE_1
NSEL, U,,, LFACE_2
CM, FREENODES, NODE

!------------------------------------------------------------------------

! GET THE NODE COUNTS
NSEL, S,,, FFACE_1
NSEL, A,,, FFACE_2
*GET, NUM_FIX_NODES, NODE, 0, COUNT
NSEL,S,,, LFACE_1
NSEL,A,,, LFACE_2
*GET, NUM_LOAD_NODES, NODE, 0, COUNT
NSEL,S,,, FREENODES
*GET, NUM_FREE_NODES, NODE, 0, COUNT
```

```
NSEL,S,,, TOTNODES
*GET, NUM_TOT_NODES, NODE, 0, COUNT

!--------------------------------------------------------------------------

! EXTRACT NODE NUMBERS IN A VECTOR
NSEL, S,,, FFACE_1
NSEL, A,,, FFACE_2
*VGET, FIX_NODES_LIST, NODE, 0 ,NLIST
NSEL,S,,, LFACE_1
NSEL,A,,, LFACE_2
*VGET, LOAD_NODES_LIST, NODE, 0 ,NLIST
NSEL,S,,, FREENODES
*VGET, FREE_NODES_LIST, NODE, 0 ,NLIST

!--------------- Apply original boundary conditions ----------------

/PREP7   ! Enter the preprocessor
ALLSEL
D,FFACE_1,ALL,ALL                        ! All dofs of the fix nodes to zero
D,FFACE_2,ALL,ALL

F,LFACE_1,FX,100/17
F,LFACE_1,FZ,125/17

F,LFACE_2,FX,100/53
F,LFACE_2,FZ,110/53

!---------------------- Solve the problem ----------------------
/SOLU ! Enter the solution
SOLVE

!---------------------- Calculate the strain Energy ----------------------

/POST1
SET,
/POST26
ENERSOL,2,SENE,,STRAINENERGY

!kommer från Time history variables

FILE,,   !kan nog behöva ändra här
/UI,COLL,1
NUMVAR,200
SOLU,191,NCMIT
```

```
STORE,MERGE
FILLDATA,191,,,,1,1
REALVAR,191,191

*GET,U_VALUE,VARI,2, REAL,1  ! Store the Strain energy value

!Exporting U-value
*cfopen,U_value.csv
*VWRITE,U_VALUE
(F15.13)
*cfclose
!--------------STORING THE DISPALECMENT d_a AND d_c------------------

*DIM, LOADNODE_DISP,ARRAY,NUM_LOAD_NODES,DOF_PER_NODE
*DO, I, 1,NUM_LOAD_NODES
    *GET,LOADNODE_DISP(I,1),NODE,LOAD_NODES_LIST(I),U,X
    *GET,LOADNODE_DISP(I,2),NODE,LOAD_NODES_LIST(I),U,Y
    *GET,LOADNODE_DISP(I,3),NODE,LOAD_NODES_LIST(I),U,Z
*ENDDO

*DIM, FREENODE_DISP,ARRAY,NUM_FREE_NODES,DOF_PER_NODE
*DO, I, 1,NUM_FREE_NODES
    *GET,FREENODE_DISP(I,1),NODE,FREE_NODES_LIST(I),U,X
    *GET,FREENODE_DISP(I,2),NODE,FREE_NODES_LIST(I),U,Y
    *GET,FREENODE_DISP(I,3),NODE,FREE_NODES_LIST(I),U,Z
*ENDDO

*MWRITE,LOADNODE_DISP,A_da.csv                  ! COMMA ISSUE RESOLVED
(F18.14,F18.14)

*MWRITE,FREENODE_DISP,A_dc.csv                  ! COMMA ISSUE RESOLVED
(F18.14,F18.14)

! ------------------------ COMPUTING U_STAR -------------------------

*DIM,U_STAR,ARRAY,NUM_TOT_NODES,2

! assign node numbers
*DO,I,1,NUM_TOT_NODES
    *SET,U_STAR(I,1),I
*ENDDO

! assign U*=1 for load nodes
*DO,I,1,NUM_LOAD_NODES
    *SET,U_STAR(LOAD_NODES_LIST(I),2),1
```

```
*ENDDO

! assign U*=0 for fixed nodes
*DO,I,1,NUM_FIX_NODES
    *SET,U_STAR(FIX_NODES_LIST(I),2),0
*ENDDO


*DIM,LOADNODE_DISP_VEC,ARRAY,1,DOF_PER_NODE*NUM_LOAD_NODES
*DO,I,1,NUM_LOAD_NODES
    *SET,LOADNODE_DISP_VEC(1,3*I-2),LOADNODE_DISP(I,1)
    *SET,LOADNODE_DISP_VEC(1,3*I-1),LOADNODE_DISP(I,2)
    *SET,LOADNODE_DISP_VEC(1,3*I)  ,LOADNODE_DISP(I,3)
*ENDDO

*DIM,PA,ARRAY,DOF_PER_NODE*NUM_LOAD_NODES,DOF_PER_NODE
*DIM,DC_EACH_FREE_NODE,ARRAY,DOF_PER_NODE,DOF_PER_NODE
*DIM,FREENODE_DISP_EACH_NODE,ARRAY,DOF_PER_NODE,1


! import the K_AC matrices created using ComputeKAC.inp
*DMAT,K_AC_FREENODES,D,IMPORT,MAT,<path>/K_AC_FREENODES
*DIM,K_AC,ARRAY,NUM_LOAD_NODES*DOF_PER_NODE,DOF_PER_NODE


*DO,I,1,NUM_FREE_NODES

    *DO,J,1,DOF_PER_NODE
        *DO,K,1,NUM_LOAD_NODES*DOF_PER_NODE
            ROWIND = (J-1)*NUM_LOAD_NODES*DOF_PER_NODE+K
            K_AC(K,J) = K_AC_FREENODES(ROWIND,I)
        *ENDDO
    *ENDDO

    *SET,FREENODE_DISP_EACH_NODE(1,1),FREENODE_DISP(I,1)
    *SET,FREENODE_DISP_EACH_NODE(2,1),FREENODE_DISP(I,2)
    *SET,FREENODE_DISP_EACH_NODE(3,1),FREENODE_DISP(I,3)

    *MOPER,PROD1,K_AC,MULT,FREENODE_DISP_EACH_NODE
    *MOPER,DENOM,LOADNODE_DISP_VEC,MULT,PROD1
    *SET,DVAR1,DENOM(1,1)

    *SET,U_STAR(FREE_NODES_LIST(I),2),1/(1 - 2*U_VALUE/DVAR1)

*ENDDO
```

```
STATUS, U_STAR

SAVE ! SAVES DATABASE TO JOBNAME.DB


!-------------------- EXPORTING DATA --------------------
*cfopen,element_type.txt
! list element types
*GET,net,ETYP,1,NUM,COUNT
*DO,i,1,net
    *GET,ETname,ETYP,i,ATTR,ENAM
    *VWRITE,'ET ',i,' ',ETname
    (A3, F4.0, A1, F4.0)
*ENDDO
*cfclose


*MWRITE,U_STAR,U_STAR,txt
(F10.0,F10.6,F10.6,F10.6,F10.6)
!6 digit precision. if you want more precision, increase'x' in F10.x


!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! format output for nodes, elements and stress listings
/PAGE, 1E9,, 1E9,,  ! disable headers
/FORMAT, , ,14,5, , ! fix floating point format
/HEADER, off, off, off, off, on, off ! disable summaries

/POST26
/OUTPUT,NLIST,txt
NLIST,,,,COORD ! print nodes w. coordinates
/OUTPUT

/OUTPUT,ELIST,txt
ELIST ! print element connectivity table
/OUTPUT

!/POST1
!/OUTPUT,DISP,txt
!PRNSOL,U ! print all displacements
!/OUTPUT

!/DELETE,,rst
```

8.0.3 Python Macro for ParaView

```
#-------------------------------------------------------------------------
# INPUT TO BE MODIFIED FOR EVERY PROBLEM
num_streamlines = 50
ParentFolder = '</path>'
VTKFileName  = 'Plate_with_hole_bending.vtk'
#-------------------------------------------------------------------------
# DIRECTIONS:
#
# Change streamTracer1.SeedType.Point1 to change the streamTracer Line Source
# File 'streamline_<I>.csv' contains data for the <I>th streamline #COMMENTED
# File 'streamline_i.csv' temporarily stores point data for finding the
  highest gradient streamline, i.e. the principal streamline
# Point data for the principal streamline stored in 'principal_streamline_<P>
  .csv'
#-------------------------------------------------------------------------

#### import csv reader
import csv

#### import numerical library
#import numpy as np

#### import the simple module from the paraview
from paraview.simple import *
#### disable automatic camera reset on 'Show'
paraview.simple._DisableFirstRenderCameraReset()

# create a new 'Legacy VTK Reader'
plate_with_holevtk = LegacyVTKReader(FileNames=[ParentFolder+VTKFileName])

# get active view
renderView1 = GetActiveViewOrCreate('RenderView')
# uncomment following to set a specific view size
# renderView1.ViewSize = [1233, 807]

# show data in view
plate_with_holevtkDisplay = Show(plate_with_holevtk, renderView1)
# trace defaults for the display properties.
plate_with_holevtkDisplay.Representation = 'Surface'

# reset view to fit data
renderView1.ResetCamera()
```

```
#changing interaction mode based on data extents
renderView1.InteractionMode = '2D'
renderView1.CameraPosition = [1.0, 0.5, 10000.0]
renderView1.CameraFocalPoint = [1.0, 0.5, 0.0]

# show color bar/color legend
plate_with_holevtkDisplay.SetScalarBarVisibility(renderView1, True)

# update the view to ensure updated data information
renderView1.Update()

# get color transfer function/color map for 'U_star'
u_starLUT = GetColorTransferFunction('U_star')
u_starLUT.ScalarRangeInitialized = 1.0

# Properties modified on u_starLUT
u_starLUT.NumberOfTableValues = 10

# create a new 'Gradient Of Unstructured DataSet'
gradientOfUnstructuredDataSet1 = GradientOfUnstructuredDataSet(Input=...
plate_with_holevtk)

# show data in view
gradientOfUnstructuredDataSet1Display = Show(gradientOfUnstructuredDataSet1,...
renderView1)
# trace defaults for the display properties.
gradientOfUnstructuredDataSet1Display.Representation = 'Surface'

# hide data in view
Hide(plate_with_holevtk, renderView1)

# show color bar/color legend
gradientOfUnstructuredDataSet1Display.SetScalarBarVisibility(renderView1, ...
True)

# update the view to ensure updated data information
renderView1.Update()

# set scalar coloring
ColorBy(gradientOfUnstructuredDataSet1Display, ('POINTS', 'Gradients', ...
'Magnitude'))

# Hide the scalar bar for this color map if no visible data is colored by it.
HideScalarBarIfNotNeeded(u_starLUT, renderView1)
```

```
# rescale color and/or opacity maps used to include current data range
gradientOfUnstructuredDataSet1Display.RescaleTransferFunctionToDataRange...
(True, False)

# show color bar/color legend
gradientOfUnstructuredDataSet1Display.SetScalarBarVisibility(renderView1, ...
True)

# get color transfer function/color map for 'Gradients'
gradientsLUT = GetColorTransferFunction('Gradients')
gradientsLUT.RGBPoints = [0.0037388897989653245, 0.231373, 0.298039, ...
0.752941, 1.5768434242596319, 0.865003, 0.865003, 0.865003, ...
3.1499479587202983, 0.705882, 0.0156863, 0.14902]
gradientsLUT.ScalarRangeInitialized = 1.0

# hide data in view
Hide(gradientOfUnstructuredDataSet1, renderView1)

# set active source
SetActiveSource(plate_with_holevtk)

# show data in view
plate_with_holevtkDisplay = Show(plate_with_holevtk, renderView1)

# show color bar/color legend
plate_with_holevtkDisplay.SetScalarBarVisibility(renderView1, True)

# reset view to fit data
renderView1.ResetCamera()

# set active source
SetActiveSource(gradientOfUnstructuredDataSet1)

# create a new 'Stream Tracer'
streamTracer1 = StreamTracer(Input=gradientOfUnstructuredDataSet1,
    SeedType='High Resolution Line Source')

# Properties modified on streamTracer1.SeedType
streamTracer1.SeedType.Point1 = [0.0, 0.01, 0.0]

# Properties modified on streamTracer1.SeedType
streamTracer1.SeedType.Point2 = [0.0, 0.49, 0.0]
streamTracer1.SeedType.Resolution = num_streamlines-1

# Properties modified on streamTracer1
```

```
streamTracer1.InitialStepLength = 0.01
streamTracer1.MaximumStepLength = 0.02
streamTracer1.MaximumSteps = 20000
streamTracer1.MaximumStreamlineLength = 200.0

# show data in view
streamTracer1Display = Show(streamTracer1, renderView1)
# trace defaults for the display properties.
streamTracer1Display.Representation = 'Surface'

# hide data in view
Hide(gradientOfUnstructuredDataSet1, renderView1)

# show color bar/color legend
streamTracer1Display.SetScalarBarVisibility(renderView1, True)

# update the view to ensure updated data information
renderView1.Update()

# Hide the scalar bar for this color map if no visible data is colored by it.
HideScalarBarIfNotNeeded(u_starLUT, renderView1)

# create a new 'Threshold'
threshold1 = Threshold(Input=streamTracer1)

# Properties modified on threshold1
threshold1.Scalars = ['CELLS', 'SeedIds']
threshold1.ThresholdRange = [0.0, 0.0]

# show data in view
threshold1Display = Show(threshold1, renderView1)
# trace defaults for the display properties.
threshold1Display.Representation = 'Surface'

# hide data in view
Hide(streamTracer1, renderView1)

# show color bar/color legend
threshold1Display.SetScalarBarVisibility(renderView1, True)

# update the view to ensure updated data information
renderView1.Update()

# destroy renderView1
Delete(renderView1)
```

```
del renderView1

# Create a new 'SpreadSheet View'
spreadSheetView1 = CreateView('SpreadSheetView')
spreadSheetView1.ColumnToSort = ''
spreadSheetView1.BlockSize = 1024L
# uncomment following to set a specific view size
# spreadSheetView1.ViewSize = [400, 400]

# get layout
layout1 = GetLayout()

# place view in the layout
layout1.AssignView(0, spreadSheetView1)

# show data in view
threshold1Display = Show(threshold1, spreadSheetView1)

# Looping through seed points
FileName     = 'streamline_'
Extension    = '.csv'
#for i in range(num_streamlines):
#
#    # Properties modified on threshold1
#    threshold1.ThresholdRange = [i, i]
#
#    # update the view to ensure updated data information
#    spreadSheetView1.Update()
#
#    # export view
#    ExportView(ParentFolder+FileName+str(i)+Extension, view=spreadSheetView1)

#-----------------------------------------------------------------------------

# set active source
SetActiveSource(threshold1)

# create a new 'Calculator'
calculator1 = Calculator(Input=threshold1)

# Properties modified on calculator1
calculator1.ResultArrayName = 'GradMag'
calculator1.Function = 'mag(Gradients)'

# Initializing streamline length vector
```

```
streamlineGrad = []

for i in range(num_streamlines):

    # Properties modified on threshold1
    threshold1.ThresholdRange = [i, i]

    # update the view to ensure updated data information
    spreadSheetView1.Update()

    # export view
    ExportView(ParentFolder+'streamline_i.csv', view=spreadSheetView1)

    # reading csv data
    with open(ParentFolder+'streamline_i.csv','r') as file:
        csv_data = csv.reader(file)
        csv_list = list(csv_data)

    # storing streamline lengths
    streamlineGrad.append(float(csv_list[1][2]))

principalStreamline = streamlineGrad.index(max(streamlineGrad))

# Properties modified on threshold1
threshold1.ThresholdRange = [principalStreamline, principalStreamline]

# show data in view
threshold1Display = Show(threshold1, spreadSheetView1)

# update the view to ensure updated data information
spreadSheetView1.Update()

# export view
ExportView(ParentFolder+'principal_streamline_'+str(principalStreamline)+...
'.csv', view=spreadSheetView1)
```

8.0.4 MATLAB function to generate VTK file

```
clear
close all
clc

fileId1 = fopen('Plate_with_hole_DL_MS.vtk','w'); %open file for writing; ...
discard existing contents
fprintf(fileId1,'# vtk DataFile Version 2.0 \n');
fprintf(fileId1,'Unstructured Grid - Plate with hole 2D \n');

%read data
nlist = readtable('NLIST.txt'); % get the locations of the nodes
u_star = readtable('U_STAR.txt'); % U* data % 'Delimiter',' ', ...
'MultipleDelimsAsOne', true
etable = readtable('ELIST.txt'); % get the element-nodes connectivity
element_type = fgets(fopen('element_type.txt'));
element_type = str2num(element_type(9:end));

%convert it to array
nlist = table2array(nlist);
u_star = table2array(u_star);
etable = int64(table2array(etable));
etable(:,2:6) = []; %removing useless columns from ELIST
nodes_per_element = length(etable(1,:))-1 ;

etable(:,all(etable == 0))=[]; % remove all the zero columns
etable=etable-1; % because the indexing in paraview starts from 0 not from 1
etable_vtk = etable ;
etable_vtk(:,1) = nodes_per_element; % insert the number of nodes for ...
element in the first column

if  element_type == 181 || element_type == 182
    cell_type = 9;
elseif element_type == 183
    cell_type = 23;
elseif element_type == 185
    cell_type = 12;
elseif element_type == 186
    cell_type = 25;
elseif element_type == 187
    cell_type = 24;
end
% if nodes_per_element == 20
%     cell_type = 25;
```

```
% end
%%


fprintf(fileId1,'ASCII \n');
fprintf(fileId1,'DATASET UNSTRUCTURED_GRID \n');
% Define points
fprintf(fileId1,'\nPOINTS %d float \n', length(nlist(:,1)));
for i =1 :length(nlist(:,1))
    fprintf(fileId1,'%d ',nlist(i,[2:4]));
    fprintf(fileId1,'\n ');
end
% define cells
fprintf(fileId1,'\nCELLS %d %d \n', length(etable(:,1)), ...
size(etable,1)*size(etable,2));

for i =1 :length(etable_vtk(:,1))
    fprintf(fileId1, '%d ', etable_vtk(i,:));
    % order is changed because of the format
    fprintf(fileId1,'\n ');
end
%%
% define cell data
fprintf(fileId1,'\nCELL_TYPES %d \n',length(etable(:,1)));
for i =1 :length(etable(:,1))
    fprintf(fileId1,'%d \n', cell_type );
end

fprintf(fileId1,'POINT_DATA %d \n',length(nlist(:,1)));
fprintf(fileId1,'SCALARS U_star float 1 \n');
fprintf(fileId1,'LOOKUP_TABLE default \n');
for i= 1:length(nlist(:,1))
    fprintf(fileId1,'%d \n', u_star(i,2) );
end

figure
plot3(nlist(:,2),nlist(:,3),nlist(:,4),'*')
hold on
% text(nlist(:,2),nlist(:,3),nlist(:,4),string(nlist(:,1)))
% axis equal
```

8.0.5 MATLAB script for Uniformity and Continuity plots

```
close all
clear
clc

M = dlmread('principal_streamline_82.csv',',',1,1);
XYZ = M(:,9:11);
Ustar = M(:,13);

lengthInc = zeros(length(XYZ),1);
lengthInc(2:end) = sqrt( sum( (XYZ(2:end,:)-XYZ(1:end-1,:)).^2 ,2) );
streamlineCoord = zeros(size(lengthInc));
for i=1:length(streamlineCoord)
    streamlineCoord(i) = sum(lengthInc(1:i));
end

s = streamlineCoord/streamlineCoord(end); %normalised streamline coordinate

%% Polynomial fitting
% Making a sufficient order polynomial fit for curvature values
p = polyfit(s,Ustar,12);
sfit = linspace(0,1,101);
Ustarfit = polyval(p,sfit);


%% Uniformity plot
fig1 = figure(1);
% yyaxis left
plot(sfit,Ustarfit,'-b','LineWidth',1.4)
ylim([0,1])

hold on
plot([0,1],[0,1],'-k','LineWidth',0.1)

set(gca,'FontSize',12)
fig1.Units = 'centimeters';
fig1.Position(3:4) = [12 8];
grid on
grid minor
title('\textbf{Uniformity curve}', 'FontSize',14, 'Interpreter','latex')
xlabel('\textbf{s}', 'FontSize',12, 'Interpreter','latex')
ylabel('\textbf{U*}', 'FontSize',12, 'Interpreter','latex')
```

```
%% Continuity plot

%-----------------------------
%%% ALSO WORKS %%%
% dydx  = gradient(y,x);
% d2ydx2 = gradient(dy,x);
% num   = d2ydx2;
% denom = ( 1+dydx.^2 ).^(3/2)
%-----------------------------

ds  = gradient(sfit);
dds = gradient(ds);
dy  = gradient(Ustarfit);
ddy = gradient(dy);
num   = ds .* ddy - dds .* dy;
denom = ds .* ds + dy .* dy;
denom = denom.^(3/2);
curvUstar = num ./ denom;
% curvUstar(denom < 0) = NaN;

% figure(1)
fig2 = figure(2);
% yyaxis right
plot(sfit,curvUstar,'-r','LineWidth',1.4)
ylim([-10 10])

hold on
plot([0,1],[0,0],'-k','LineWidth',0.1)

set(gca,'FontSize',12)
fig2.Units = 'centimeters';
fig2.Position(3:4) = [12 8];
grid on
grid minor
title('\textbf{Continuity curve}', 'FontSize',14, 'Interpreter','latex')
xlabel('\textbf{s}', 'FontSize',12, 'Interpreter','latex')
ylabel('\textbf{Curvature}', 'FontSize',12, 'Interpreter','latex')
```

8.0.6 MATLAB script for computing reaction traction

```
Order=1;

NLIST = readtable('NLIST.txt');
NLIST = table2array(NLIST);
NLIST = NLIST(:,2:3);

% RF_SUPPORT = readtable('RF_SUPPORT.txt');
% RF_SUPPORT = table2array(RF_SUPPORT);
RF_SUPPORT = dlmread('RF_SUPPORT.txt');
% Use dlmread if the lines above don't work
RF_SUPPORT(:,1) = NLIST(RF_SUPPORT(:,1),2);
RF_SUPPORT(:,4) = sqrt(RF_SUPPORT(:,2).^2+RF_SUPPORT(:,3).^2); % resultant
RF_SUPPORT = sortrows(RF_SUPPORT,1);


%% Computing resultant traction

M = zeros(length(RF_SUPPORT));
nel = (length(RF_SUPPORT)-1)/Order;
for e=1:nel
    indx = ((e-1)*Order+1):(e*Order+1);
    xe = RF_SUPPORT(indx,1);
    Me = constructM(xe);
    M(indx,indx) = M(indx,indx) + Me;
end

trX = M\RF_SUPPORT(:,2);
trY = M\RF_SUPPORT(:,3);
trR = sqrt(trX.^2 + trY.^2);

%% Plotting traction along support

fig5 = figure(5);
plot( RF_SUPPORT(:,1) , trX , 'Linewidth',1.4)
set(gca,'FontSize',12)
fig5.Units = 'centimeters';
fig5.Position(3:4) = [12 8];
grid on
grid minor
title('\textbf{Traction along support}', 'FontSize',14, 'Interpreter','latex')
xlabel('\textbf{Support edge [m]}', 'FontSize',12, 'Interpreter','latex')
ylabel('\textbf{Traction x-dir [N/m]}', 'FontSize',12, 'Interpreter','latex')
```

```
fig6 = figure(6);
plot( RF_SUPPORT(:,1) , trY , 'Linewidth',1.4)
set(gca,'FontSize',12)
fig6.Units = 'centimeters';
fig6.Position(3:4) = [12 8];
grid on
grid minor
title('\textbf{Traction along support}', 'FontSize',14, 'Interpreter','latex')
xlabel('\textbf{Support edge [m]}', 'FontSize',12, 'Interpreter','latex')
ylabel('\textbf{Traction y-dir [N/m]}', 'FontSize',12, 'Interpreter','latex')

fig7 = figure(7);
plot( RF_SUPPORT(:,1) , trR , 'Linewidth',1.4)
set(gca,'FontSize',12)
fig7.Units = 'centimeters';
fig7.Position(3:4) = [12 8];
grid on
grid minor
title('\textbf{Traction along support}', 'FontSize',14, 'Interpreter','latex')
xlabel('\textbf{Support edge [m]}', 'FontSize',12, 'Interpreter','latex')
ylabel('\textbf{Resultant [N/m]}', 'FontSize',12, 'Interpreter','latex')


%% Functions

function Me = constructM(xe)
Order = length(xe)-1;
NNPE = Order+1;
[q_pt,q_wt] = define_quadRule(6);

Me = zeros(NNPE);
for k=1:length(q_pt)
    N  = zeros(1,NNPE);
    for i=1:NNPE
        N(1,i) = phi(i,q_pt(k),Order);
    end
    Me = Me + q_wt(k)*(N'*N)* (xe(2)-xe(1))/2;
end
end


function val = phi(node,xi,Order)
val=1;
for j=1:Order+1
    if j~=node
```

```
        val = val*(xi-xi_at_node(j,Order))/...
        (xi_at_node(node,Order)-xi_at_node(j,Order));
    end
end
end


function xi = xi_at_node(N,Order)
if N<=Order+1
    xi = -1 + 2*(N-1)/Order;
else
    error('Node count limited to %d per elem', Order+1)
end
end


function [quad_points,quad_weight] = define_quadRule(Q)
% Q = Number of quadrature points

quadRule = Q;

EPS = 3.0e-16;
x1= -1; x2= 1;

m=(Q+1)/2;
xm=0.5*(x2+x1);
xl=0.5*(x2-x1);
for i=1:m
   z=cos(3.141592653*(i-0.25)/(Q+0.5));
   % Starting with the above approximation to the ith root, we enter
   % the main loop of refinements by Newton's method
   z1 = z + 2*EPS;
   while abs(z-z1) > EPS
      p1 = 1.0;
      p2 = 0.0;
      for j=1:Q
         p3 = p2;
         p2 = p1;
         p1 = ((2.0*j-1.0)*z*p2-(j-1.0)*p3)/j;
      end
      % p1 is now the desired Legendre polynomial. We next compute pp,
      % its derivative, by a standard relation involving also p2, the
      % polynomial of one lower order
      pp = Q*(z*p1-p2)/(z*z-1.0);
      z1 = z;
```

```
      z = z1-p1/pp; % Newtons method
   end
   % Scale the root to the desired interval, and put in its
   % symmetric counterpart
   quad_points(i) = double (xm-xl*z);
   quad_points(Q+1-i) =double(xm+xl*z);
   % Compute the weigh and ist symmetric counterpart
   quad_weight(i) = double (2.0*xl/((1.0-z*z)*pp*pp));
   quad_weight(Q+1-i) = quad_weight(i);
end
end
```

DEPARTMENT OF INDUSTRIAL AND MATERIAL SCIENCE
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY