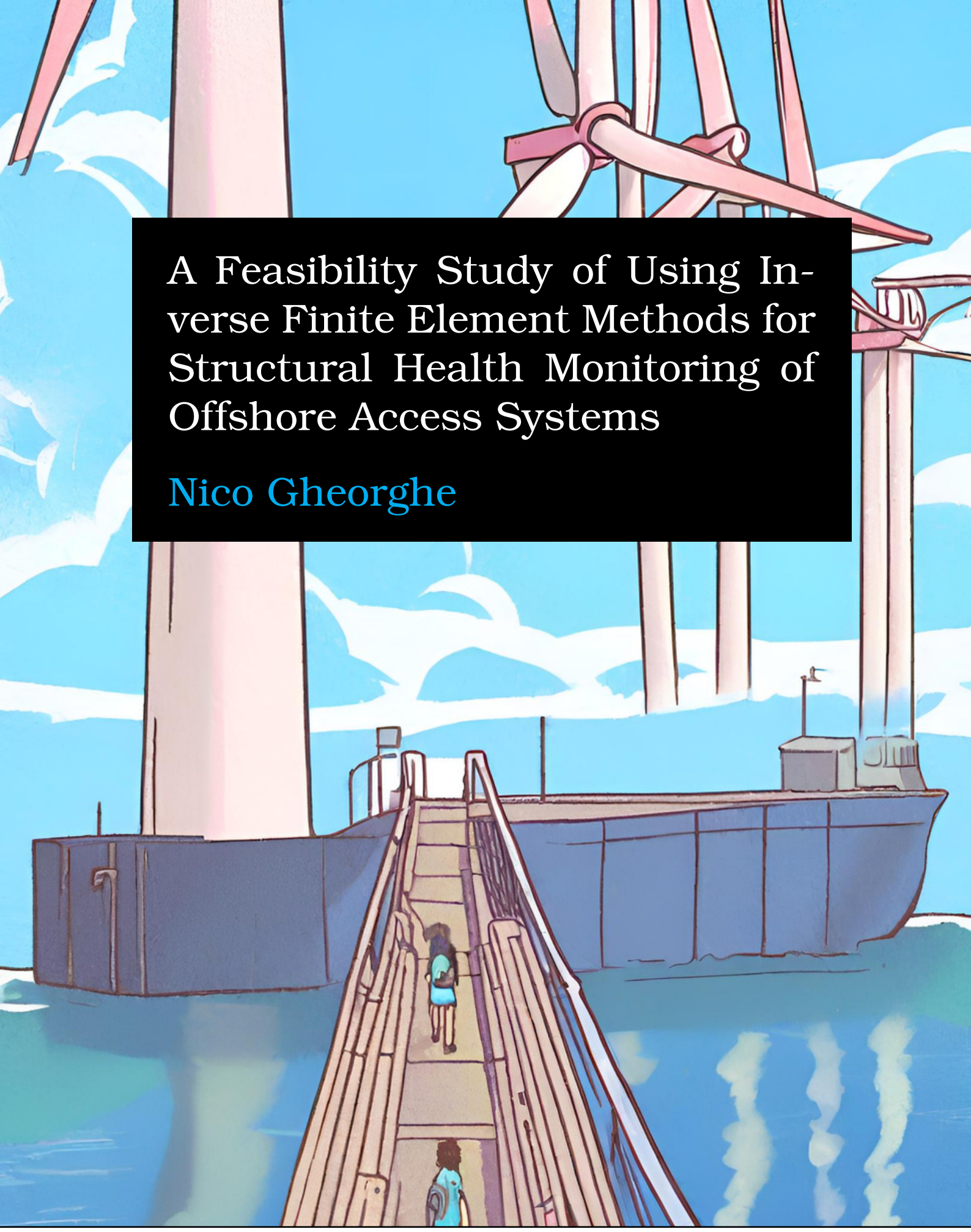


A Feasibility Study of Using Inverse Finite Element Methods for Structural Health Monitoring of Offshore Access Systems

Nico Gheorghe

TU Delft Delft University of Technology

AMPELMANN

A Feasibility Study of Using Inverse Finite Element Methods for Structural Health Monitoring of Offshore Access Systems

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering
at Delft University of Technology

Nico Gheorghe

22/02/2024

Faculty of Aerospace Engineering · Delft University of Technology

Cover page image generated through AI on canva.com

Delft University of Technology
Faculty of Aerospace Engineering
Department of Aerospace Structures and Materials

GRADUATION COMMITTEE

Dated: 22/02/2024

Chair holder:

Dr.ir. R. Groves. Chair

Committee members:

Dr.ir. N. Yue. Supervisor

Ir. L. van Veen. Company Supervisor

Dr.ir. S.P. Castro. Examiner

Dr.ir. L. Phalavan. Examiner

Acknowledgments

While the report reads me as the only author, the list of people who contributed to my morale, motivation and well-being, and thus to the good development of this project, does not seem to end.

I want to thank Siebert and Sai, my first contact with Ampelmann. Thank you for entrusting me with this amazing project. I want to thank all my colleagues here, especially those from the Concepts and Structural teams. Thank you for all the interesting discussions, feedback and patience.

My deepest gratitude goes to all my supervisors. From the company's side, Elsy and Lisanne. Thank you for offering me the support I needed in some rather rough patches and for all the advice, both professional and personal. From the university's side, Nan. Thank you for taking the time with me. You are a great educator and a truly inspiring researcher.

I got the chance to make the most amazing friends here in Delft. Alex, Andrada, Crina, Tudor and Octavian, thank you for being my home away from home and my continuous source of motivation over the past 5+ years. Your hard work and efforts inspired mine. Ishita, Irina and Naishu, thank you for being my comfort and always providing me with the best laughs. Martino, Michal, Nikita and Oscar, thank you for always pushing me and inspiring me to do the best I can. You kept pushing my targets higher and higher. Xenia, thank you for being my safe space and my commute companion. Andrei and Ioana, thank you for helping me broaden my technical skills outside Aerospace and helping me gain confidence in them. Your patience and kindness was greatly appreciated. Horia, thank you for always having a kind word for me. You made the tough days a bit lighter.

Tanuj, thank you for your continuous affection and endearment. You were both my thesis buddy and my biggest supporter. Thanks for putting up with me during the raccoon phases.

All my love and affection goes to the people back home. Many thanks to my highschool friends: Cristiana, Cristi and Mihaita. You guys always put a smile on my face. Thanks to my childhood friends, Vlad and Radu, for always making me feel strong. Thanks to my best friend, Andrada, for always believing in me and keeping me accountable. You are the first one to pull me back when I stray.

But the most important and precious of all, I want to thank my family. Many thanks to my grandparents for showering me with their love and care. Thanks to my aunt, Consuela, for being a true role model regarding discipline around work. All my gratitude goes to my parents for all their efforts, sacrifice and trust in me. Thank you for having all the confidence in me and my abilities even when I was full of doubts.

Delft, University of Technology — Nico Gheorghe

Abstract

Over the past decade, the offshore industry, particularly the wind energy sector, has exhibited continuous growth, emphasizing the escalating significance of sustainability within this domain. The current work seeks to enhance this aspect by exploring offshore access systems featuring composite gangways.

Composite offshore structures however pose challenges due to the involved complex damage mechanisms and the need for novel maintenance procedures, introducing uncertainties concerning their operation. A Structural Health Monitoring (SHM) system is proposed for increasing confidence in the safe operation of the new composite gangway.

The suggested SHM system relies on Inverse Finite Element Methods (iFEM) deflection reconstruction using Fiber Optics (FO) strain data. The gangway design is simplified to a U-shaped beam geometry under bending load, modeled using Inverse Quadrilateral Shell 4 Points (IQS4) elements. Its performance was assessed using mock strain data generated numerically through Finite Element Methods (FEM) software.

Deflection reconstruction using both tri-axial and uni-axial strain measurements was investigated, revealing that uni-axial measurements can be sufficient for the current application. The sensing network was streamlined by focusing on line configurations along the length of the beam, leveraging the capabilities of FO sensors.

The introduction of strainless inverse elements highlighted the limitations of strain pre-extrapolation with Smoothing Element Analysis (SEA) for such a geometry. Modeling guidelines and their effect on improving the robustness of SEA are explored. A strain sensing network using four uni-axial sensing lines is found to offer a sufficiently accurate deflection reconstruction for the application.

Table of Contents

Acknowledgments iii

Abstract iv

Nomenclature viii

List of Acronyms . . . viii

1 Introduction 1

I Literature Study 2

2 Literature Study Starting Point 3

3 Background 5

4 Composites 7

4.1 General Definitions . . . 7

4.2 Damage Mechanisms of Composites . . . 8

4.3 Composites Application in Offshore . . . 8

5 Operations 10

5.1 Operational Environment . . . 10

5.2 Operational Loads . . . 11

5.3 Gangway Operational Incidents . . . 12

6 Maintenance 14

6.1 Maintenance Strategies . . . 14

6.2 Current Practices . . . 15

6.3 Structural Health Monitoring . . . 15

7 Discussion on gangway SHM goals **17**
7.1 Sensors . . . 17
7.2 Operational Focus . . . 18
7.3 Tracked Behaviour . . . 18
7.4 Methods for Tracking Deflection . . . 19
7.4.1 Numerical Integration . . . 20
7.4.2 Linear Continuous Basis Functions . . . 21
7.4.3 Inverse Finite Element Methods . . . 22
7.4.4 Concluding Discussion . . . 23

8 Inverse Finite Element Methods **24**
8.1 Data Smoothing . . . 24
8.2 Damage Identification . . . 25

9 Research Proposal **28**

II Thesis Work **30**

10 Methodology **31**
10.1 iFEM . . . 31
10.1.1 General Functional . . . 31
10.1.2 Variational Approach . . . 32
10.2 Inverse Element Formulation . . . 33
10.2.1 Element Overview . . . 33
10.2.2 Shape Functions . . . 35
10.2.3 Strain Location . . . 36
10.2.4 Matrix Formulation . . . 37
10.2.5 Coordinate System Transformation . . . 38
10.2.6 Boundary Conditions . . . 39
10.3 Smoothed Element Analysis . . . 40
10.3.1 Overview . . . 40
10.3.2 Quadrilateral SEA Element Formulation . . . 40
10.3.3 Involved Parameters . . . 43
10.4 Method for Verification . . . 44
10.5 Implementation Overview . . . 44
10.5.1 Integration . . . 45
10.5.2 Weights Assigning . . . 45
10.5.3 Sensor Placement Correction Factor . . . 45
10.5.4 Strain pre-extrapolation Usage . . . 46
10.5.5 Hyperparameters Determination . . . 46
10.5.6 Input . . . 46
10.5.7 Layout . . . 47
10.6 Gangway Structure: Model Simplifications . . . 47
10.7 Sensor Network . . . 50
10.8 Performance Assessment . . . 50

11 Results **52**
11.1 iFEM Exploration 52
11.1.1 Effect of symmetric Boundary Condition (BC) 57
11.1.2 Effect of SEA 58
11.2 Design Simplified Configuration I - Isotropic Deck 58
11.3 Design Simplified Configuration II - Laminated Deck 61
11.4 Design Simplified Configuration III - Isotropic U-Shape 61
11.4.1 U-Shaped Geometry Literature Study Case 61
11.4.2 Actual Implementation 62
11.4.3 Influence of Local Coordinate System Definition on Smoothed Inverse Finite Element Methods (iFEM(s)) analysis 69
11.4.4 Effect of Hyperparameters on Deflection w_f, α Reconstruction 71

12 Conclusion **73**

13 Recommendations **75**

References **77**

III Appendices **84**

A Gangway Design **85**

B FEMAP Macros **86**

C Code **90**

Nomenclature

List of Acronyms

M-Boom	Main Boom
T-Boom	Telescopic Boom
FRP	Fiber Reinforced Polymer
CFRP	Carbon Fiber Reinforced Polymer
SHM	Structural Health Monitoring
LR	Lloyd's Register
DNV	Det Norske Veritas
AE	Acoustic Emission
FBG	Fiber Bragg grating
FO	Fiber Optics
iFEM	Inverse Finite Element Methods
iFEM(s)	Smoothed Inverse Finite Element Methods
FEM	Finite Element Methods
ANN	Artificial Neural Network
PZT	Lead zirconate titanate
NDT	Non-destructive testing
SEA	Smoothing Element Analysis
DIC	Digital Image Correlation
KOT	Ko's Displacement Theory
DOF	Degree of Freedom
IQS4	Inverse Quadrilateral Shell 4 Points
TL	Test Load
DL	Dead Load
DLTL	Dead Load Test Load

MAPD	Mean Absolute Percentage Difference
PD	Percentage Difference
BC	Boundary Condition
T3	Out-of-plane displacement
RZT	Refined Zig-Zag Theory

List of Symbols

Greek Symbols

α penalty parameter in SEA

β penalty parameter for function curvature in SEA

γ shear strain

ε membrane strain

θ_x Rotation around x-axis

θ_y Rotation around y-axis

θ_z Rotation around z-axis

ν Poison's Ratio

$\mathbf{\Phi}$ iFEM error functional

$\mathbf{\Phi_{SEA}}$ SEA error functional

$\mathbf{\Phi}_\varepsilon$,$\mathbf{\Phi}_\alpha$,$\mathbf{\Phi}_\beta$ residual terms SEA

Ψ_x x-derivative of interpolated strain in SEA

Ψ_y y-derivative of interpolated strain in SEA

Latin Symbols

$\mathbf{B}$ Strain-displacement matrices

$\mathbf{e}$ membrane strains vector

E Elastic Modulus

$\mathbf{f^e}$ input element matrix iFEM

$\mathbf{F}$ input global matrix IFEM

$\mathbf{g}$ transverse shear strains vector

G Shear Modulus

h plate mid-height

$\mathbf{k}$ curvatures vector

$\mathbf{k^e}$ analytical element matrix iFEM

$\mathbf{K}$ global analytical element matrix iFEM

L_i, M_i, N_i Shape functions

n number of strain measurements

s, t Natural Coordinates

$\mathbf{T^e}$ element transformation matrix

u Translation displacement in x-axis

$\mathbf{u^e}$ DOF vector of an element

$\mathbf{U}$ Displacement Vector

v Translation displacement in y-axis

w Translation displacement in z-axis

w_e, w_k, w_g iFEM weight factors per strain component

w_f weighting factor iFEM

Chapter 1

Introduction

Ensuring personnel safety remains a primary objective in offshore operations. The transfer of personnel and cargo offshore represents a notably vulnerable process, prompting the implementation of various methods such as Crew Transfer Vessels landings, transfer baskets, swing ropes, and more in an effort to enhance safety [Hu and Yung, 2020]. Recently, gangway-based solutions, particularly when integrated with motion control systems, have emerged as a safer alternative on the market.

Ampelmann Operations is a company that specializes in gangway-based offshore access solutions. In line with its commitment to innovation, the company is embarking on the development of the industry's first composite offshore gangway. The main drivers for switching away from steel lie in improved sustainability which is hoped to be enabled by the higher specific properties and corrosion durability of composites. However, venturing into new technological territory brings its fair share of challenges and uncertainties. To ensure the utmost safety in offshore access operations with a new gangway, the implementation of a Structural Health Monitoring (SHM) system is proposed.

The current work begins with a preliminary investigation on the needs of such a SHM system and which technologies could be employed and draws up a research proposal for a level 1-SHM system using shape reconstruction through Inverse Finite Element Methods (iFEM). The research aims to determine how iFEM can be used for the gangways, what would be a sensible sensing network and what could be the accuracy of such a system. The conducted work thus represents a feasibility study from the technical and operational perspectives.

The report is divided into three parts. Part I presents the literature study on the topic. Aspects such as the offshore environment, operation and maintenance of offshore access systems are discussed. The findings of the literature study are used for defining the research proposal in chapter 9.

Part II covers the work done in the project. Firstly, the methodology is laid down. This covers the theoretical background of iFEM and Smoothing Element Analysis (SEA), details on the implementation and how the sensor networks are evaluated. Lastly, Part III includes the appendices and could be relevant for readers interested in the programming side of the project.

Part I

Literature Study

Chapter 2

Literature Study Starting Point

The current project had as starting point the following task: "Design a Structural Health Monitoring (SHM) system for a composite gangway". To tackle this assignment and narrow down the scope of the MSc thesis, a literature study was first conducted on the topics of interest.

The research during the literature study phase was defined by the preliminary research questions:

- What characterizes the offshore environment?
- What are the common practices regarding design, operations and maintenance in the offshore access industry?
- What loading conditions does a gangway typically experience?
- What are the causes of damage in a gangway?
- What detection techniques can be used for determining damage in composites?
- What sensors can be used in these techniques? How suitable are they for the operating environment?

Using these research questions as a guide for navigating the current literature and resources, it was aimed to reach the following goals:

1. Establishing suitable sensors for the application.
2. Establishing a suitable damage mechanism or parameter to track.
3. Establishing a detection method for tracking the parameter in question.

The literature review adheres to the following structure. Chapter 3 offers background information regarding offshore access systems. Chapter 4 focuses on composites, covering general definitions, applications and the main considerations for switching to a composite gangway. Chapter 5 aims to give an understanding of both typical and accidental gangway operations.

Standard maintenance strategies and the current practices along with an introduction to SHM will be discussed in chapter 6. The findings from chapters 3-6 are discussed for narrowing down the scope of the SHM system in chapter 7. Chapter 8 offers background information on Inverse Finite Element Methods (iFEM). Lastly, in Chapter 9 a research proposal is formulated based on the findings of the literature review.

Chapter 3

Background

The offshore sector covers a wide range of activities such as oil and gas exploration, wind energy, fishing and telecommunications. Irrespective of the exact application, offshore accessibility remains of utmost importance. Both passengers and cargo need to be transferred effectively and safely even in extreme weather conditions. This is increasingly done through gangway access systems. Such systems are commonly mounted on vessels, allowing transfer between the origin vessel and the target vessel, wind farm or platform. Ampelmann is one of the main manufacturers of gangway access systems.

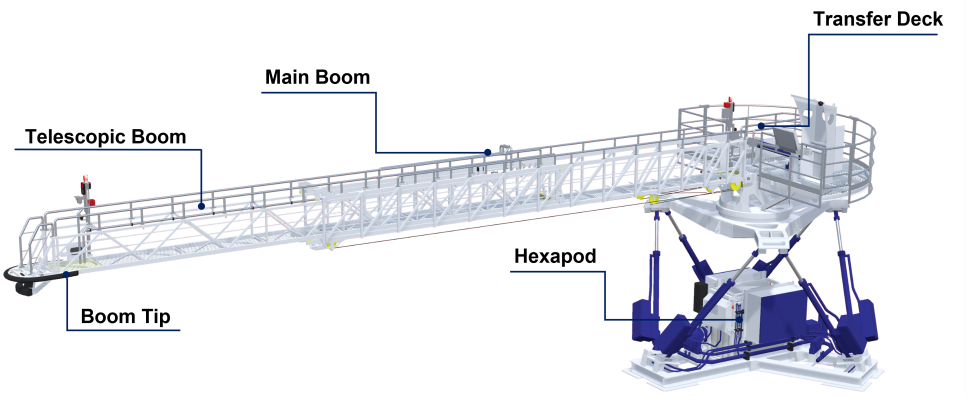


Figure 3.1: Ampelmann system diagram. Courtesy of Ampelmann.

Figure 3.1 shows a general diagram of an Ampelmann offshore access system. The bridge-like structure is known as the gangway and is made of two parts: the Main Boom (M-Boom) and the Telescopic Boom (T-Boom). The T-Boom slides in and out with respect to the M-Boom allowing for adjustable lengths of the gangway. The gangway itself can be mounted on different systems. The systems can be either motion-compensated systems (such as the A-type and E-type that are shown in the figure), or not (such as the F-type). These systems

play an important role in defining the loads acting on the gangway as each of them is designed for different operating conditions and cases.

There are three key terms describing the movement of a gangway:

- *Telescoping* represent the movement of the T-Boom along the M-Boom
- *Luffing* represents the vertical movement of the gangway.
- *Slewing* is the rotational movement along the base of the system and is provided by the slewing ring.

Chapter 4

Composites

The current chapter focuses on general information regarding composite materials and their application in the offshore sector. Section 4.1 offers general definitions regarding composite materials. Section 4.2 describes the main damage mechanisms of composite structures. Lastly, section 4.3 briefly highlights current applications of composites in the offshore sector and offers an overview of the considerations regarding a composite gangway.

4.1 General Definitions

Composites can be defined as materials that are made out of two or more constituent materials. Fiber Reinforced Polymer (FRP) are a popular type of composites used in advanced engineering applications (aerospace, automotive, marine etc.) due to their high specific properties. They are made out of a polymer matrix and continuous fibers as shown in Figure 4.1.

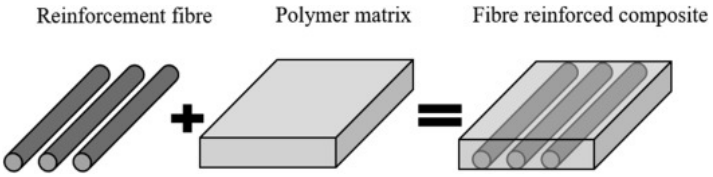


Figure 4.1: General structure of FRP. Courtesy of [Hesseler et al., 2021]

Laminated composites are a specific type of composite materials that consist of multiple layers. These layers are stacked together to form a laminate structure. Each layer is typically referred to as a lamina or a ply. A layer can be a continuous, unidirectional FRP ply for example.

In a laminated composite, the layers are stacked in a specific orientation to achieve desired properties and performance. The orientation and sequence of the layers can be engineered to optimize characteristics such as strength, stiffness, impact resistance, and other mechanical properties.

For the current gangway application, FRP's composites will be considered.

4.2 Damage Mechanisms of Composites

This section aims to give an overview of composites' damage mechanisms. [Talreja, 2016] identifies the damage mechanisms within composites. FRP can be subjected to interlaminar damage:

- Fiber Breakage. This can occur due to excessive tensile or compressive stresses.
- Matrix Cracking. Can be caused by tensile, compressive or shear stresses.
- Interface failure. This occurs at the interface of the composite constituents, leading to their separation.

Laminated composites can also have intralaminar failure which is also known as delamination, representing a fracture between two adjacent plies.

Nevertheless, isolating these damage mechanisms in real operation scenarios is quite difficult. Composites have complex and interrelated damage mechanisms [Kefal et al., 2021b]. For example, impact events can lead to a combination of some or all of these types of damages [Agrawal et al., 2014].

4.3 Composites Application in Offshore

One of the most well-established applications of composites in the offshore sector is wind turbine blades. They are typically made out of glass fibers reinforced epoxies. Measuring on average 52 [m], but reaching up to 107 [m] [1], composite wind turbine blades allow for considerable weight saving. Composite materials are also regularly used for offshore pipe risers [2] and offshore platform handrail system [3].

Even though the use of composites has become more widespread in the offshore sector, access gangways remained a steel-only part. The motivation for introducing composite gangways can be resumed as follows:

- Energy savings. Composites allow for high tailorability of the design. This usually leads to considerably lower weights of the designs. Moreover, due to the snowball effect, the weight of other components such as actuators for the motion-compensated systems will be reduced, leading to energy savings.
- Durability. As opposed to steel, composites are resistant to corrosion. In the offshore environment, corrosion is extremely aggressive, leading quite often to complete part replacement as repair does not suffice.

However, their implementation also comes with a series of challenges:

- Less effective visual inspection. As will be discussed in section 6.2, visual inspection is a highly used tool in the current maintenance strategy. Composite materials can present large internal damages, but only small indents at a surface-level [Shah et al., 2019].

[1] https://tinyurl.com/29j6z6vc
[2] https://tinyurl.com/pva63e97
[3] https://tinyurl.com/bdhpeezu

- Environmental effects. Offshore conditions such as high humidity or UV exposure can impact negatively composites' mechanical properties. This will be discussed in section 5.1.
- Limited knowledge in the material characterization for the gangway application and lack of current certification guidelines. This leads to a more complex certification process and a more conservative design which can limit the weight savings.

Chapter 5

Operations

The current chapter will focus on the operational aspects of access gangways. Section 5.1 gives an overview of the main environmental factors and their effect on composite structures. Section 5.2 aims to offer a general understanding of the loads that are defined for a gangway, while section 5.3 describes some of the incidents to which the gangway can be subjected.

5.1 Operational Environment

As the name indicates, offshore activities take place away from the terrestrial shore, typically in the open sea which leads to harsh environmental conditions, affecting both human operators and the integrity of the structures. The offshore environment can be characterized by the following factors:

- **Wind**. The wind creates side forces on the walls of offshore structures. [Hu and Yung, 2020] highlights the trend of moving wind farms farther from shore due to preferable conditions. For an open profile access gangway, this strong wind can lead to excessive torsion of the structure. To counteract this, openings are introduced on the side walls. This is done by decreasing the surface area and thus, the wind loading. This also allows for improving visibility and reducing mass.
- **Humidity**. While composites are not subject to corrosion issues such as traditional steel structures, humidity can still harm their performance due to the absorption of moisture into the matrix. [Randhawa and Patel, 2021] conducted a study on the effect of humidity on polymers' mechanical properties. The identified general trends were a decrease in Young's Modulus and strength, but an improvement in impact strength and elongation.
- **UV Exposure** The degradation effect of UV is typically limited to the top microns of the surface. However, it can lead to the variability of the mechanical properties, thus creating concentration factors [Aldajah et al., 2009]. [Shi et al., 2022] aimed to quantify the variation of mechanical properties of epoxy-based Carbon Fiber Reinforced

Polymer (CFRP) under accelerated UV exposure during 80 days. The results showed a decline of 23% in the longitudinal compressive strength, but also embrittlement of the matrix.

5.2 Operational Loads

This section aims to give a general understanding of the operational loads that an access gangway is expected to encounter. Certification codes for such systems will be used as a reference. The new composite gangway will be used in both people transfer Figure 5.1 and cargo transfer Figure 5.2.

Figure 5.1: Ampelmann system used for people transfer [1]

Figure 5.2: Ampelmann system used for cargo transfer [2]

For personnel transfer, some of the operational loads and conditions that are prescribed can be summarized as follows:

1. Principal Load (Art. 4.1.2 [DNV, 2017])
   (a) Self-weight of the structure and all installed equipment
   (b) Live load (maximum number of persons including luggage allowed on the gangway at the same time)
2. Vertical loads due to operational motions (Art. 4.1.3 [DNV, 2017])
   (a) Inertia forces due to acceleration or deceleration of horizontal motions. These forces are typically associated with the starting or interruptions of luffing motions.
3. Horizontal loads due to operational motions (Art. 4.1.4 [DNV, 2017])
   (a) Inertia forces due to acceleration or deceleration of horizontal motions. These forces are typically associated with the starting or interruptions of slewing motions.
   (b) Centrifugal forces created by slewing motions.
4. Loads due to climatic effects (Art. 4.1.5 [DNV, 2017])
   (a) Ice and snow loads if relevant.
   (b) Wind load. For operational cases, a value of minimum 20 [m/s] needs to be accounted for. However, the minimum value can be increased to 51.5 [m/s] in certain conditions. These values are described for 10 [m] above sea level and need to be adjusted to the gangway's height.

[2] https://tinyurl.com/2pjm786u
[2] https://tinyurl.com/bdhey24y

(c) Vortex-induced oscillations. [Fu, 2018] describes vortex shedding as the phenomenon where alternating vortices are released from one side to the other of a structure exposed to wind, creating fluctuating forces perpendicular to the wind direction due to alternating low-pressure zones on the downwind side. These forces can induce oscillations in the structure.
(d) Sea pressure loads (green sea loads). In case of extreme ship motions during storms, water can flow onto the gangway, leading to green sea loads [Buchner, 2002].

5. Loads due to motion of the vessel on which the gangway is mounted (Art. 4.1.6. [DNV, 2017]) Usually expressed in terms of vertical/transverse/longitudinal accelerations.
6. Contact loads (Art. 3.21 [LR, 2021a]). When the tip comes into contact with the target unit.
7. Mothership static inclinations (Art. 3.12 [LR, 2021a]). They are defined using the heel and trim angles of the ship.
8. Temperature effects (Art. 3.18 [LR, 2021a]). Loads can result from thermal expansion/contraction.

In addition to the personnel transfer loads, cargo lifting has other design considerations. The horizontal distances between the payload and the boom tip (offlead and sidelead) are of importance (Art. 2.2. [LR, 2021b]). They define the loading path on the tip and can create a load swing.

Both Det Norske Veritas (DNV) and Lloyd's Register (LR) define distinct load combinations. For each of these conditions, different indications are given for applying the loads (magnitudes, locations, inclinations etc.). Using the current legislation, a series of load cases for designing the gangway can be constructed.

5.3 Gangway Operational Incidents

Gangway incidents encompass a wide range of events, from equipment failures to human errors. This section aims to briefly present the main incidents affecting the structural integrity of the gangway.

1. Tool drops. Transferees are usually carrying with them different pieces of equipment and tools. Some of these tools can get dropped. Although the grating provides some protection from drops, some incidents still lead to dents in the gangways.
2. Drift off. The gangway is mounted on the target and fully extended. The target moves farther away from the origin (drift-off), leading to an over-extension of the gangway and a possibility of it getting stuck in that position, leading to increased stresses at the level of the telescoping rail.
3. Drift on. The gangway is mounted on the target and fully retracted. The target moves towards the origin (drift-on). In severe scenarios, this can cause the Telescopic Boom (T-Boom) to push excessively onto the Main Boom (M-Boom), possibly leading to buckling.
4. During parking of the tip, impacts between the target and the tip are quite common. However, when slewing for parking, also the gangway railing can be hit by the target platform leading to torsion of the gangway.

5. Collision between the gangway and a large body (another vessel, a wind turbine blade etc.)
6. Extreme weather conditions such as hailing can create structural damage to the gangway.

Chapter 6

Maintenance

Maintenance is of paramount importance for offshore structures due to the harsh and corrosive marine environment they are exposed to. Regular maintenance ensures longevity and integrity, minimizing the likelihood of costly repairs, and safeguarding the transferees from possible accidents. Section 6.1 offers an overview of general maintenance strategies, while section 6.2 covers the current maintenance strategies within Ampelmann. Lastly, section 6.3 offers an introduction to Structural Health Monitoring (SHM).

6.1 Maintenance Strategies

Factors such as requirements, part failure modes, available resources and costs associated with the downtime of the system will influence the maintenance strategy. [Ren et al., 2021] distinguishes between maintenance strategies as follows:

1. **Corrective Maintenance.** Also known as failure-based strategy, requires intervention when a part breaks down, being reactive and unscheduled. It offers the advantage of allowing a part to go through its whole operational life while reducing activity-based waste. Such a strategy is not optimal for a primary structure such as the gangway as its failure leads to an operational halt of the complete system.
2. **Preventive Maintenance.** Preventive maintenance implies scheduling maintenance activities at set predetermined time intervals. While it allows for planning into the operational schedule, it leads to increased waste in the form of labor and activities.
3. **Condition-based Maintenance.** Condition-based maintenance requires the mounting of sensors on the target structure to monitor its structural integrity. [Farrar and Worden, 2012] formulate axiom IVa of SHM which states: "Sensors cannot measure damage". Thus, these readings need to be processed in different ways to establish whether or not there is damage to the structure. This is generally done through damage feature extraction [Sohn et al., 2003]. These damage features are compared against an established threshold which, when exceeded, determines the need for maintenance activities.

4. **Predictive Maintenance.** Predictive maintenance is the next step of condition-based maintenance. Making use of the detected damage features and data analysis, it aims to predict system degradation and failures before they occur. Thus, the optimal timing for maintenance activities can be determined. This prevents unplanned downtime, minimizes costs, and maximizes operational efficiency.

6.2 Current Practices

Currently, for regular operation, Ampelmann uses a preventive maintenance strategy based on time intervals:

1. Daily, as done by operators using the Daily Progress Report (DPR). The DPR contains a checklist of different systems and areas to check. For structural components, the checks are highly based on visual inspection.
2. Weekly, as done by operators. Similar procedures to the daily checks, but different areas are investigated(less prone to damage or harder to access).
3. Monthly, as done by operators. Similar procedures to the daily and weekly checks, but different areas are investigated.
4. Yearly, done by technicians in agreement with legislation (App. B.3 [DNV, 2017]). This check focuses on thorough visual inspection, proper lubrication and Non-destructive testing (NDT) where considered necessary. The repair plan for any damaged primary structure must be agreed on with the certification body.
5. 5-yearly, done by engineers in agreement with legislation (App. B.4 [DNV, 2017]). In addition to the yearly check, this check can require repeating the load testing and examination done for the initial certification.

In case of operational incidents, the decisions regarding the maintenance approach are, usually but not only, taken between the reliability, asset and operations engineers. If needed, these decisions are also taken in consultation with certification bodies. Thorough inspection, potentially employing NDT, is typically used at the location of the incident and the locations specified by the *Critical Areas in Structure* document. This document is developed within Ampelmann during the design phase and identifies prime areas for inspection. These areas are typically high-stress points, welds or connection points.

6.3 Structural Health Monitoring

It is important to note that the current practices discussed in section 6.2 have been developed for steel gangways. Due to the limited experience of working with composites, the environmental effects discussed in section 5.1 and the challenges in section 4.3, it cannot be decided if the current preventive maintenance strategy will be sufficient for ensuring the safe and efficient operation of the composite gangway.

Opting for an additional condition-based maintenance plan can offer more confidence in the operation of the new design. For this purpose, an SHM system can be implemented. The importance and benefits of SHM systems for maritime application were officially recognized

already in 1994 by the International Maritime Organisation. Certification bodies for maritime and offshore applications such as Det Norske Veritas (DNV) (2011) and Lloyd's Register (LR) (2011) followed ([Kefal, 2019]), further contouring the necessity of SHM implementation in gangway systems.

According to [Güemes et al., 2020],a SHM system can be divided into three main components:

1. A sensing network.
2. Data acquisition system.
3. Algorithms for data processing.

Taking into account all three components, the current report aims to review relevant literature on SHM systems for composite offshore gangways.

SHM can be distinguished between active and passive methods. Active methods require excitation of the structure through energy self-generated by the SHM system ([Nelson and MacIver, 2006]). They allow for the repeatability of the measurement, and also for its variation through the probe's controllable variables such as intensity, direction or timing. Such methods are ultrasonics where ultrasonic waves need to be propagated externally from the structure or experimental modal analysis which requires excitation through external vibration such as an impact hammer.

Passive sensing methods are based on intrinsic energy sources. Strain measurements and Acoustic Emission (AE) are typical passive methods. Passive sensing is preferred to active sensing in situations when measurement during operation is required ([Saeedifar and Zarouchas, 2020]).

SHM technology can also be classified on multiple levels of [Farrar and Worden, 2010]:

Level 1 **Detection**. *Is there damage present in the structure?*
Level 2 **Localization**. *Where is the damage located?*
Level 3 **Type**. *What kind of damage occurred?*
Level 4 **Extent**. *What is the severity of the presented damage?*
Level 5 **Prognosis**. *What is the remaining useful life of the structure?*

Chapter 7

Discussion on gangway SHM goals

The current chapter aims to narrow down the scope of the literature study based on the previously discussed information. Section 7.1 discusses the sensing options, while section 7.2 defines the operational scope. Section 7.3 identifies which behavior shall be tracked. Lastly, section 7.4 discusses the possible methods for tracking the selected behavior.

7.1 Sensors

There is a wide variety of available sensors. For Structural Health Monitoring (SHM) applications, two sensor options emerge often in literature: Lead zirconate titanate (PZT) and Fiber Optics (FO) ([Tinghu and Jones, 2004], [Hafizi et al., 2015], [Kudela et al., 2008]). However, when looking at implementations in the offshore or marine industries, FO emerges in literature as a highly preferred choice. A comprehensive review of different uses of FO in marine applications can be found in [Min et al., 2021]. This preference is explained by the main advantages of FO, as highlighted by [Floris et al., 2021]'s review on FO shape sensing:

1. Compactness, small size and lightweight.
2. Embedding capability.
3. Resistance to harsh environments, including humidity, severe temperature, chemicals and radiation.
4. Electrically passive operation. This is a considerable advantage for operations in offshore environments. Humidity affects the conductivity of the sensors, leads to internal corrosion and decreases their average life. Moreover, carbon fibers are electric conductors and in case of current leakage in the SHM sensors, safety hazards can occur.
5. Immunity to Electromagnetic Interference (EMI).
6. Multiplexing capability. Large structures such as the gangway usually require an increased amount of sensors, which can lead to issues in terms of cable management. Thus, sensors with multiplexing capabilities such as FO are preferred.
7. High sensitivity and accuracy.

PZT sensors stand at a disadvantage for marine and offshore as they are electrically based. This explains the overall trend of moving towards optic-based sensors for measurement techniques using traditionally PZT sensors such as acoustic emission ([Vidakovic et al., 2016]) or ultrasonics ([Soman et al., 2021]).

The goal of the current research is developing an SHM system, not a sensing technology. This leads to a strong preference for previously proven sensors that can be used as building blocks in the project. FO are considered mature technologies for SHM applications [Inaudi and Glisic, 2005], [Rocha et al., 2021]. Thus, the current research scope will focus on FO-based SHM.

7.2 Operational Focus

As discussed in section 5.3, gangways can undergo a wide range of incidents, each resulting in different damage patterns. To be able to identify the damage in all of these situations, the SHM would require a highly dense network of sensors.

Nevertheless, even with such a dense network reliable detection cannot always be guaranteed. Multiple studies including [Ussorio et al., 2006], [Hafizi et al., 2015] [Vidakovic et al., 2016] highlight that even for lab conditions and network sensors which are optimized for a certain location and type of damage, the detection rate is not without failure. Overlapped with additional unknowns, the accuracy of such a system would only degrade. If there is little reliability in the SHM system, its justification for implementation becomes rather limited.

Moreover, the SHM system would not influence considerably the current maintenance approach in case of incidents. In case of an incident, the system would still require to be stopped from operation and assessed as described in section 6.2. Any SHM data would probably only be used as a confirmation at the end of the assessment. Thus, it is expected that it will not result in a big reduction in system downtime, limiting its economic justification at this time.

Using a system for normal operation would be of use from multiple perspectives. Firstly, it can highlight an issue before it is detected in regular maintenance or it leads to failure, increasing both safety and reducing the chance of unplanned downtime. Secondly, because the loadings and critical areas are better defined in normal operation, the SHM system can be optimized to a higher degree, which should increase the reliability of the provided results and decisions.

Limiting the scope of the research to a system tailored for the normal operation would allow for more confidence in the provided results and a higher chance of economic savings.

7.3 Tracked Behaviour

[Farrar and Worden, 2010] identifies multiple four key questions for conducting the underlying operational evaluation for the SHM system. A critical one in the current gangway case is *"How is damage defined for the system being investigated and, for multiple damage possibilities, which cases are of the most concern?"*. It can be considered critical due to the limited

knowledge and experience of composites in gangway applications. What is considered damage in the current case? Without answering this question, further defining of alert thresholds for autonomous measurement evaluation becomes impossible.

Looking at a local level, a simple matrix crack could be considered damage in the system. However, giving an alarm for the mere presence of a crack could lead to high waste for investigation activities. Ideally, it would be desired to give an alarm when such a crack, or any type of local damage really, starts affecting either the operational performance or the safety of the gangway.

Parameters that are set for evaluating either of these factors (maximum allowable strengths/deflections/angles) are of global nature. Drawing conclusions about the impact of localized damage on these global parameters, considering the limited application experience of composites in gangways, can result in significant inaccuracies.

Directly monitoring global behavior can prove as a better approach for the current application. A composite gangway would still need to comply with at least the current legislation which can be used as a starting point for threshold setting. Art. 7.3.2.2 [DNV, 2017] sets the standard for allowable deflections. The deflections are defined in Figure 7.1 and their maximum values in Table 7.1.

Table 7.1: Bridge load test condition prescribed by [DNV, 2017].

Condition	Limit for max	Limit for δ_2
G <2*TL	L /100	L /150
G =2*TL		L /200
G >2*TL		L /300

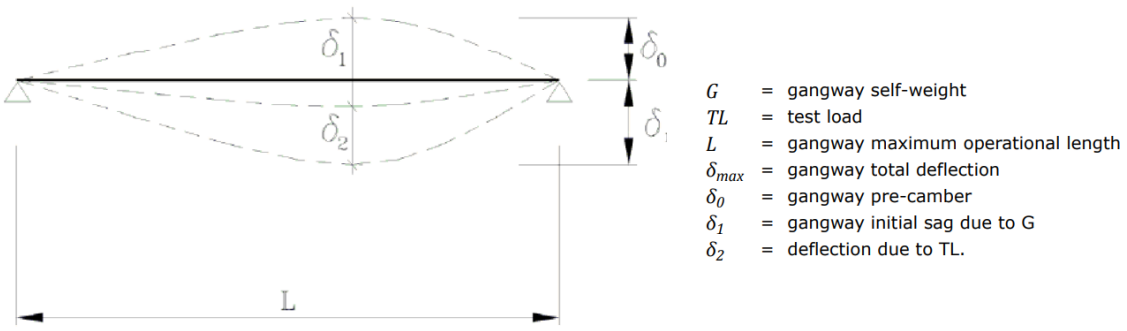


Figure 7.1: Bridge load test schematic. Courtesy of [DNV, 2017].

7.4 Methods for Tracking Deflection

FO shape-sensing has emerged as a popular technique for the 3D dynamical reconstruction of structures. First picked up in the 90's [Greenaway et al., 1999] as curvature sensing, it evolved over time to continuous strain measurement based on Rayleigh scattering.

Its advantages lie in its capability of mapping in the absence of visual contact [Floris et al., 2021]. Its applications are focused on curvature-critical applications such as medical equipment ([Larkin and Shafer, 2011], [Park et al., 2014]), bridges' deformation ([Yang et al., 2017]) and aircraft wing shape monitoring ([Sun et al., 2018]).

In their review work, [Esposito and Gherlone, 2020] distinguish four types of shape-sensing approaches using strain measurements:

1. Numerical integration of experimental strain
2. Linear continuous basis functions for displacement field approximations
3. Inverse Finite Element Methods (iFEM)
4. Artificial Neural Network-based methods

In the current study, the first three approaches will be discussed. The first three methods are model-based, while the fourth is data-driven. Data-driven methods are highly dependent on the dataset that is used for training. This fact was also confirmed for shape-sensing applications by [Mao, 2008]. The study compared the performance of a data-driven approach with a modal approach. The conclusions of the study confirmed that the modal approach always rendered more accurate results. The only exception was tested with concentrated forces. In these cases, the data-driven approach rendered better results because the system was trained with similar data.

There is the possibility of creating some training data numerically through Finite Element Methods (FEM) for different loading cases. However, for the current application, there is no in-operation experimental data available for training. This would make the system unprepared for handling the material degradation, inherent defects, possible damage or an unpredicted loading combination, reducing greatly the confidence in the obtained reconstruction. Thus, for this review, the focus will be put on model-based approaches.

[Tidriri et al., 2016] highlights that data-driven methods are advantageous for large-scale systems as they require few computations. Therefore, such an approach could be considered for a later gangway SHM implementation if adequate data is collected.

For now, the three other candidates will be discussed in subsections 7.4.1 to 7.4.3. subsection 7.4.4 concludes by choosing the deflection monitoring option that will be pursued in the study.

7.4.1 Numerical Integration

First introduced in [Ko et al., 2007], Ko's Displacement Theory (KOT) was developed for the reconstruction of an aircraft's wing shape. It is based on the Euler-Bernoulli beam theorem, making use of the numerical integration of experimental strains. Considering the classical beam differential equation:

$$\frac{d^2w}{dx^2} = \frac{M(x)}{EI} \tag{7.1}$$

Taking the case of a beam loaded in simple bending:

$$\sigma(x) = \frac{M(x)h}{EI} \tag{7.2}$$

Substituting $\epsilon(x) = \sigma(x)/E$, the basis equation of Ko's theory can be obtained:

$$\frac{d^2w}{dx^2} = \frac{\epsilon(x)}{h} \tag{7.3}$$

[Ko et al., 2007] developed formulations for different types of beams (uniform, tapered, cantilevered, simply supported etc.) and for bending and torsional loads. [Jutte et al., 2011] used the method for shape-sensing during a ground load test of a full-scale wing. The deflections are reconstructed accurately. However, the quality of twist reconstructions is considerably poorer due to an identified sensitivity to the error in bending. Generally, strain integration along a linear path in this approach leads to spatial resolution errors, thereby restricting its applicability primarily to beam structures [Freydin et al., 2019].

7.4.2 Linear Continuous Basis Functions

Modal methods are the most popular ones in this category ([Kefal et al., 2021b]). [Bang, 2012] applies Fiber Bragg grating (FBG)-based shape-sensing for the dynamic monitoring of displacement of a wind turbine. The displacements are determined using the displacement-strain transformation (DST) following Equation 7.4.

$$\{y\}_{N\times 1} = [T]_{N\times M}\{\varepsilon\}_{M\times 1} \tag{7.4}$$

where [T] is defined using according to Equation 7.5 using the strain mode shape matrix [Ψ] and the mode shape matrix [Φ].

$$[T]_{N\times M} = [\Phi]_{Nn} \cdot \left([\Psi]^T_{n\times M} \cdot [\Psi]_{M\times N}\right)^{-1} \cdot [\Psi]^T_{n\times M} \tag{7.5}$$

The displacement-strain matrix was constructed using the FEM results for the modes dominant in X-directional bending (axis pointing through the nacelle length). This choice was made as the motions of the structure are expected to be dominant in this direction as it aligns with the primary wind direction. Using the FBG reading, the tower top deflection was determined.

Although the authors present the current approach for SHM applications, it remains unclear how the approach can be used for damage/degradation identification. The DST is determined numerically from the idealized, damage-free FEM model. This makes the shape reconstruction dependent on both the material properties and the loading conditions, both being hard to obtain pieces of information during operation. Although [Rapp et al., 2009] conducted a successful experimental validation of the method for the simplified case of a cantilevered plate, [Bang, 2012]'s wind turbine application is also lacking a validation of the results.

[Pak, 2016] proposed a two-step method for shape reconstruction, combining both KOT and modal methods. The first step consists of the double-integration of strain over a straight line for obtaining deflections. Then, the slopes and deflections in the whole structure are determined using the System Equivalent Reduction and Expansion Process which required FEM-generated matrices. [Pak, 2016] validated the approach successfully on the test data of a cantilevered swept-plate wing model.

The application of modal methods for shape-sensing has been widely investigated in the field of aircraft aeroelasticity. Nevertheless, their potential for damage identification is currently restricted due to the reliance on FEM outputs for displacement reconstruction. This reliance decreases the accuracy of shape reconstruction which is the first step before damage identification.

7.4.3 Inverse Finite Element Methods

iFEM was first introduced by [Tessler, 2003] and [Tessler and Spangler, 2005]. The approach allows for the reconstruction of elastic deformations in plates and shell elements from experimental strain. It is presented as real-time reconstruction for SHM applications due to its robustness and reduced computation time compared with previous methods such as the modal transformation technique introduced by [Bogert et al., 2003].

Moreover, iFEM does not require any information on material properties or loading conditions for the reconstruction of both static and dynamic displacement responses ([Gherlone et al., 2014]). The iFEM was further developed, adding formulations for other types of elements such as curved shells ([Kefal, 2019]), beams ([Gherlone et al., 2014]) and solid elements ([Mooij et al., 2019]).

The iFEM was validated in multiple studies. [Abdollahzadeh et al., 2022] proved the shape reconstruction accuracy of iFEM of thin Carbon Fiber Reinforced Polymer (CFRP) plates under large deformation using FBG. [Gherlone et al., 2014] investigated the behavior of a circular cross-section Aluminium-6000 cantilever beam under both static and dynamic loading using strain gauges. Overall, there was good agreement between the iFEM-computed displacements and the displacements obtained using linear variable differential transformers. The errors did not exceed 10% for any of the investigated cases.

[Gherlone et al., 2014] also looked into the effect of strain gauge configurations. The results highlighted a change in accuracy for the different loading cases depending on the configuration. Typically a higher amount of sensors is preferred, however, their orientation also needs to be considered. For example, the tip twist rotation is better predicted by configuration C_2 (6 strain gauges out of which 1 inclined at $\beta = 45°$) than C_4 (8 strain gauges out of which 3 inclined at $\beta = 45°$). This was explained by the inherent loss in the accuracy of strain gauge measurements over curvatures. Thus, while iFEM offers a robust computation method, its results remain sensitive to sensor-generated errors. [Gherlone et al., 2014] concluded the accuracy of the present method could be increased when using FO-based sensing and an optimized distribution of measurement points.

[Kefal et al., 2021a] aimed to validate the use of iFEM on a larger specimen with a more advanced geometry than a beam or plate. A 1 [m] long composite sandwich wing undergoing tip deflections was monitored using both FBG and strain rosettes. Digital Image Correlation (DIC) is also used to monitor the leading edge of the wing (denoted as l_3). A high-fidelity FEM model was also built to replicate the test case. Figure 7.2 shows a comparison between the iFEM, FEM and DIC. What is especially interesting, is that, as noted by the authors, the DIC results are matched better by those of iFEM than those of FEM. The study increased confidence in the usefulness of iFEM for engineering structures of larger scales.

Although iFEM is more computationally costly than other shape-sensing techniques, it is judged as a fast algorithm, suitable for real-life monitoring ([Gherlone et al., 2011]).

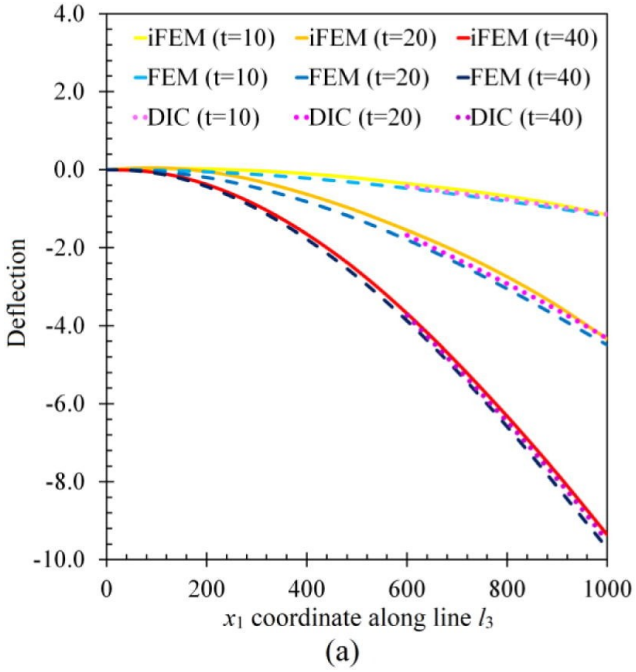


Figure 7.2: Comparison of deflection [mm] along the leading edge (l_3) between iFEM, FEM and DIC at different time stamps. Courtesy of [Kefal et al., 2021a]

7.4.4 Concluding Discussion

As discussed in the previous subsections, there are multiple methods for tracking deflection. It was already concluded that Artificial Neural Network (ANN) methods are placed out of scope due to the lack of relevant training data. Modal methods offer limited confidence in reconstruction due to their dependence on FEM outputs.

Although KOT is quite limited due to its simplicity, it is considered suitable to beam-like structures, category in which the gangway falls. Thus, the applicability of KOT in this application can still remain of interest for further studies. Nevertheless, it will not be pursued in the current study due to the limited timeframe. Moreover, KOT is currently restricted to only damage detection and the literature does not identify any potential for higher levels of SHM.

Out of all the presented methods, [Esposito and Gherlone, 2020] identifies that iFEM is the most accurate one for shape reconstruction. Its main disadvantage lies in the need for more sensing points compared to other methods. Nevertheless, this is not considered a showstopper as FO allows for scalability through its multiplexing and reduced cost per sensor [Guo et al., 2011]. Thus iFEM will be chosen as a method of interest for the study. Additional background information on iFEM will be presented in chapter 8.

Chapter 8

Inverse Finite Element Methods

The current chapter provides additional information on the Inverse Finite Element Methods (iFEM). The goal of this chapter lies in exploring the full capabilities of iFEM and its potential for higher levels of Structural Health Monitoring (SHM). Section 8.1 introduces the concepts of smoothing and its benefits, while section 8.2 highlights the use of iFEM for damage identification.

8.1 Data Smoothing

Smoothing Element Analysis (SEA) was first introduced by [Tessler, 1998] for obtaining continous strain and stress fields in Finite Element Methods (FEM) from discrete points. It uses a variational principle combining discrete least-squares and penalty constraint functions. The method has been applied also for recovering stresses and error computing in adaptive mesh refinement. In general, the advantage of smoothed FEM lies in more accurate results and higher convergence rates as noted by [Zeng and Liu, 2018]. SEA implementations also reported similar results, leading to superconvergent stress of significantly higher accuracy when compared with standard FEM ([Tessler, 1998]).

[Tessler and Spangler, 2005] states that SEA can also be coupled with iFEM for minimizing experimental error in each strain components. [Kefal et al., 2021b] offers a method for coupling SEA and iFEM and its advantages.

[Abdollahzadeh et al., 2022] opts for a polynomial interpolation for smoothing the experimental strain. The study gives a quantification of the contribution of smoothed iFEM. A visual comparison can be seen in Figure 8.1. The authors reported an error of 7% between iFEM (using discrete strains) and FEM, and an error of 0.9% between smoothed iFEM and FEM.

It can be concluded that smoothed iFEM methods render more accurate results compared to the standard iFEM implementation.

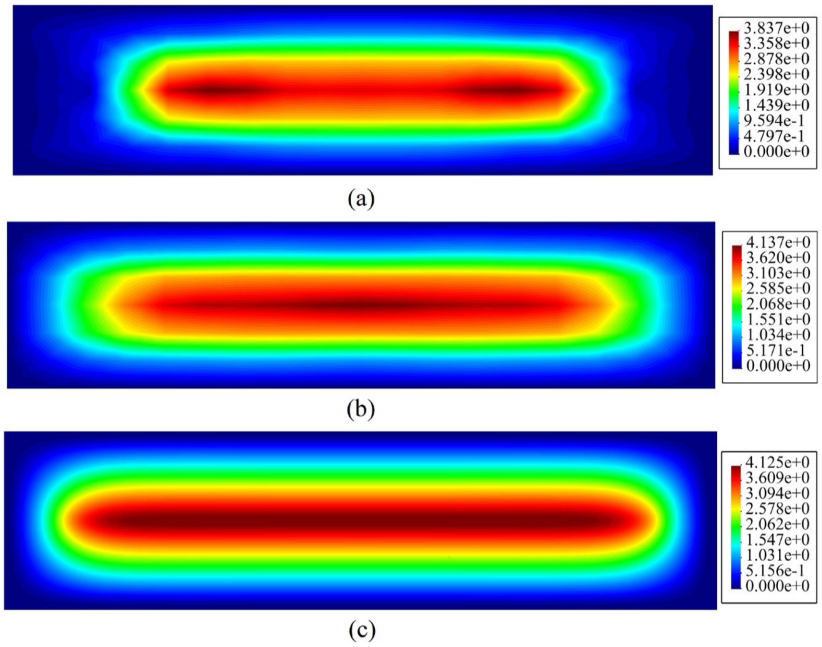


Figure 8.1: Comparison between displacement contours obtained through (a) iFEM using discrete strained (b) Smoothed iFEM (c) FEM

8.2 Damage Identification

iFEM started being explored for damage identification. The current techniques for doing so are based on the discrepancies within the FEM and iFEM strain fields. However, [Colombo et al., 2021] notes that at that time few applications of iFEM for damage identification are available and most of them are limited to metallic structures.

For example, [Roy et al., 2020] aims to identify cracks in a metal plate under bi-axial loading. The technique is based on identifying points where the ϵ_{eq} (reconstructed equivalent strain) is higher than $\epsilon_{eq,noise}$ (baseline equivalent strain including measurement uncertainties). The authors identify that $\epsilon_{eq,noise}$ is dependent on the measurement precision of the equipment, the placement and the density of the sensor network. $\epsilon_{eq,noise}$ defines the smallest damage that can be picked up. The method is verified numerically through ABAQUS. It was possible to localize the cracks with the precision of the grid cell. It is important to note that for this implementation, the loading case was known, allowing for accurate FEM modeling. However, knowing the load cases in true operational conditions is a challenge.

[Colombo et al., 2019] introduced a load adaptive method for damage identification through iFEM, allowing for damage localization independent of loading conditions. The authors define an anomaly index based on percentage difference. The authors separate their sensing grid into input sensors (at positions x_{in}) and test sensors(at positions x_t). The strains of the input sensors are then fed into the iFEM algorithm, allowing the reconstruction of the iFEM strain also at x_t locations.

Figure 8.2 highlights the workflow of the load-adaptive framework. For each unknown loading condition l, a test strain and an input strain will be measured. The input strain is inserted in iFEM and used for calculating an equivalent strain $\epsilon_{eq,iFEM}$ at every x_t position. The

test strains are used directly for computing the equivalent strain $\epsilon_{eq,t}$ at the x_t positions. The equivalent strains are then used for computing the anomaly index i at the x_t positions. If the anomaly index is 0, then the structure is considered to be "healthy". This condition only holds true under a strict set of assumptions, the authors recognizing that for operational applications a certain type of threshold based on the sensor layout, noise and uncertainty needs to be established.

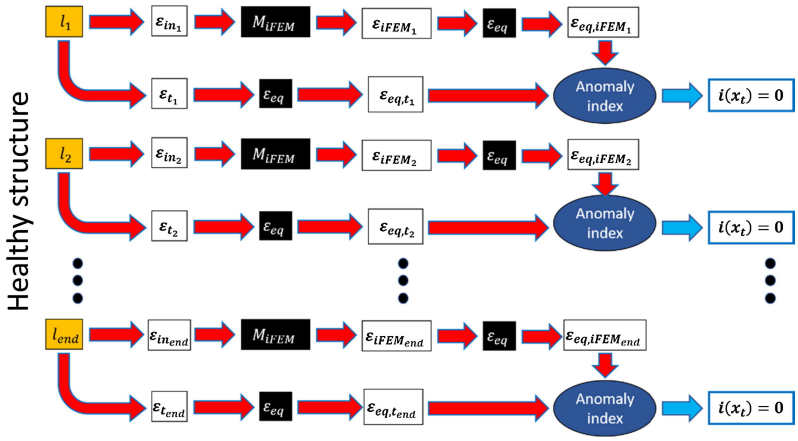


Figure 8.2: Schematic of load-adaptive algorithm for damage localisation. Courtesy of [Colombo et al., 2019]

[Colombo et al., 2019] verifies the load adaptive framework numerically in 4 loading conditions. Figure 8.3 shows visually the results for a plate containing two cracks. Subfigures (a),(c),(e) and (g) have the same position and size of the two cracks. Subfigures (b),(d),(f) and (h) share another configuration of the two cracks. Thus, each column has the same damage condition.

The study also tries different loading cases:

- Subfigures (a) and (b): Tension
- Subfigures (c) and (d): Bending
- Subfigures (e) and (f): Torque
- Subfigures (g) and (h): Combination of previous three loading cases

Thus, in Figure 8.3 each row of subfigures has the same loading condition.

The authors of the study consider that the damages were localized within reasonable accuracy. It was noticed that the anomaly index is dependent on the loading conditions, as can be seen by looking at the variation of the head map across the column in Figure 8.3.

Later on, the authors applied this framework also on a composite structure. [Colombo et al., 2021] recognizes it as the first application of damage identification through iFEM on composite structures. The method was validated for delamination within a Carbon Fiber Reinforced Polymer (CFRP) stiffened panel under impact damage and fatigue testing. Although the same load-adaptive framework was kept, the Mahalanobis distance was used for describing the anomaly index this time, as opposed to the previous percentage difference approach in [Colombo et al., 2019]. The Mahalanobis index was concluded to be independent of the

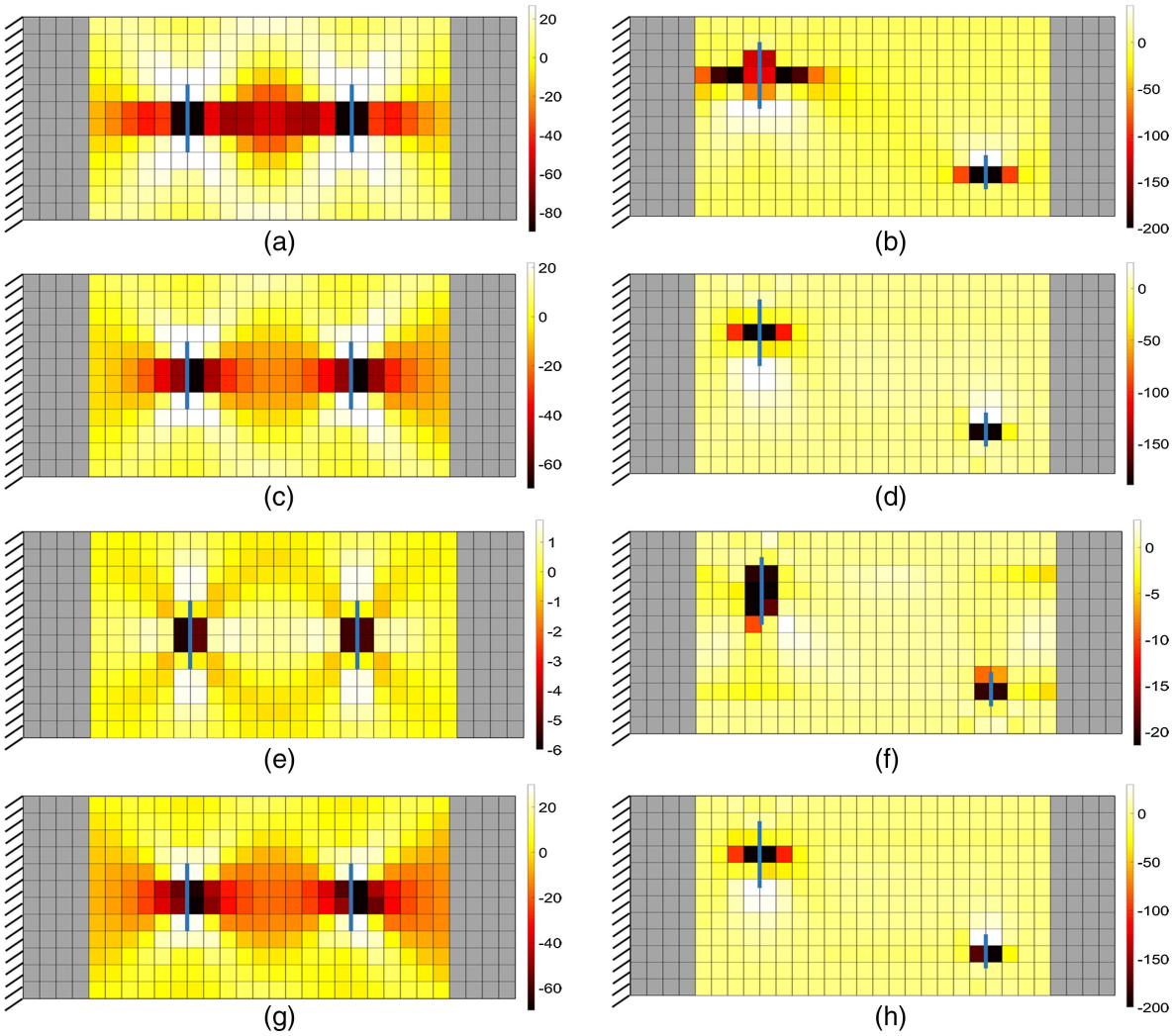


Figure 8.3: Anomaly index computed for plates with two cracks. Each row represents the same loading condition and each column represents the same damage condition. The heat map describes the anomaly index, while the blue lines used for marking the location of the cracks for reference. Courtesy of [Colombo et al., 2019]

applied load cases. In the same year, [Kefal and Tessler, 2021] proposes a different method for delamination identification through iFEM. Although a load-dependent method is used, its capability for thorough thickness damage detection remains of interest.

The research on damage identification through iFEM for composites is still quite limited and in its incipient phases. Nevertheless, the available studies agree in their conclusions regarding the potential of further development of iFEM for damage identification in SHM applications.

This possibility is of high interest as it could enable achieving in the future higher levels of SHM using iFEM based methods. This justifies further the usage of iFEM in this preliminary investigation compared to simpler methods such as Ko's Displacement Theory (KOT) which do not present the potential of upgrading in the future.

Chapter 9

Research Proposal

Based on the discussion from chapter 7 and the additional information about Inverse Finite Element Methods (iFEM) in chapter 8, the research questions of the subsequent project can be formulated.

The initial part of the research will focus on the implementation of iFEM. Aspects such as the numerical implementation, the type of elements that are relevant for the gangway and the iFEM representation of the gangway shall be tackled.

The next point of attention is investigating a suitable sensor network for shape reconstruction. Aspects such as the number of sensors, their location and directions of measurements shall be discussed.

Lastly, the performance of the proposed Structural Health Monitoring (SHM) system shall be assessed. A comparison between the accuracy of iFEM and Smoothed Inverse Finite Element Methods (iFEM(s)) is of interest. This would allow us to quantify the merits of smoothed-iFEM for the specific gangway application.

The research questions can be summarized as follows:

RQ1 **How can iFEM be implemented for SHM of a composite gangway?**

- RQ1.1 How can iFEM be implemented in a Python framework?
- RQ1.2 What type of inverse elements is suitable for the application?
- RQ1.3 How can the gangway structure be simplified for a first iFEM implementation?
  - RQ1.3.1 How can the geometry be simplified?
  - RQ1.3.2 What would be the dimensions of this shape?
  - RQ1.3.3 What are representative boundary conditions?
  - RQ1.3.4 What is a representative loading case? What is the magnitude of this loading?

RQ1.3.5 How do the deflections/deformations of the simplified load compare to the deformations of the reference model under the same load?

RQ1.3.6 What aspects of the real structure are ignored in this simplification?

RQ1.3.7 How would these aspects affect the deformation of the structure?

RQ1.3.8 Could these aspects be implemented in a later iFEM implementation? How?

RQ2 **What is a suitable sensing network architecture for the gangway?**

RQ2.1 Are all strain components necessary for deflection reconstruction? Which are dominant?

RQ2.2 Is it possible to use only uni-axial strain measurements?

RQ2.3 What kind of strain measurement configurations should be assessed?

RQ2.4 How should the accuracy/performance of a sensing network be assessed?

RQ3 **What is the performance of the proposed SHM system?**

RQ3.1 What is the accuracy of the iFEM reconstruction with respect to the Finite Element Methods (FEM) model?

RQ3.2 How does it change when opting for a smoothed iFEM implementation?

Part II

Thesis Work

Chapter 10

Methodology

This chapter outlines the steps taken for exploring and answering the research questions. Sections 10.1 to 10.3 build up the required theoretical background. section 10.4 highlights how verification is conducted, while section 10.5 covers the implementation of the Inverse Finite Element Methods (iFEM) framework in code. section 10.6 outlines how the gangway model was simplified for the current project. section 10.7 describes some of the considerations regarding the assessment of sensing networks and section 10.8 defines the indices for assessing the reconstruction.

10.1 iFEM

As discussed in chapter 8, iFEM offers the possibility of shape reconstruction of a structure without information regarding the material characterisation or the loading conditions. The completeness of the shape reconstruction is dependent on the formulation of the chosen inverse element type. These elements dictate the degrees of freedom for which the reconstruction can be done.

10.1.1 General Functional

The general approach of iFEM is first formulating a functional $\Phi_e(\mathbf{u}^e)$.Equation 10.1 gives a general form, accounting for membrane ($\mathbf{e}$), curvature ($\mathbf{k}$) and transverse shear ($\mathbf{g}$) strains.

This functional captures the difference between the analytical strain components $\mathbf{e}(\mathbf{u}^e)$, $\mathbf{e}(\mathbf{k}^e)$, $\mathbf{e}(\mathbf{g}^e)$ and their experimental counterparts $\mathbf{e}^\varepsilon$, $\mathbf{k}^\varepsilon$ and $\mathbf{g}^\varepsilon$. The squaring of the norm allows for reducing the effect created by a potential outlier on the complete reconstruction. This becomes very useful to experimental strains where noise or a faulty sensor can occur.

$$\Phi_e(\mathbf{u}^e) = w_e \|\mathbf{e}(\mathbf{u}^e) - \mathbf{e}^\varepsilon\|^2 + w_k \|\mathbf{k}(\mathbf{u}^e) - \mathbf{k}^\varepsilon\|^2 + w_g \|\mathbf{g}(\mathbf{u}^e) - \mathbf{g}^\varepsilon\|^2 \tag{10.1}$$

Such a functional needs to be expressed for each element, as indicated by the subscript e in Φ_e. The expression also accounts for the lack of strain measurement for an element through the weights w_e,w_k and w_g. These weights are positive, but always smaller or equal to 1. When the strain component of an element is recorded, then its value can be set to 1. However, when the strain component is lacking its value should be set to a lower value. It becomes apparent that their use becomes critical when sensor networks with a limited amount of strain sensing points are used. [Tessler and Spangler, 2005] assists in illustrating the role of these penalty parameters by describing them as a balancer between the correlation of measured and analytical strains.

The functional can be further expanded using the normalized Euclidian norms. This is shown for both instrumented elements in Equation 10.2. n denotes the number of discrete measurements within an inverse element and serves as a normalization parameter. Depending on the number of measurements assigned per element it could or not factor out.

$$\begin{aligned}
\|\mathbf{e}\left(\mathbf{u}^e\right) - \mathbf{e}^\varepsilon\|^2 &= \frac{1}{n} \iint_{A^e} \sum_{i=1}^{n} \left(\mathbf{e}\left(\mathbf{u}^e\right)_i - \mathbf{e}_i^\varepsilon\right)^2 dxdy \\
\|\mathbf{k}\left(\mathbf{u}^e\right) - \mathbf{k}^\varepsilon\|^2 &= \frac{(2h)^2}{n} \iint_{A^e} \sum_{i=1}^{n} \left(\mathbf{k}\left(\mathbf{u}^e\right)_i - \mathbf{k}_i^\varepsilon\right)^2 dxdy \\
\|\mathbf{g}\left(\mathbf{u}^e\right) - \mathbf{g}^\varepsilon\|^2 &= \frac{1}{n} \iint_{A^e} \sum_{i=1}^{n} \left(\mathbf{g}\left(\mathbf{u}^e\right)_i - \mathbf{g}_i^\varepsilon\right)^2 dxdy
\end{aligned} \tag{10.2}$$

For strainless elements, the expansion is given in Equation 10.3.

$$\begin{aligned}
\|\mathbf{e}\left(\mathbf{u}^e\right)\|^2 &= \iint_{A^e} \mathbf{e}\left(\mathbf{u}^e\right)^2 dxdy \text{ with } (w_e = \alpha) \\
\|\mathbf{k}\left(\mathbf{u}^e\right)\|^2 &= (2h)^2 \iint_{A^e} \mathbf{k}\left(\mathbf{u}^e\right)^2 dxdy \text{ with } (w_k = \alpha) \\
\|\mathbf{g}\left(\mathbf{u}^e\right)\|^2 &= \iint_{A^e} \mathbf{g}\left(\mathbf{u}^e\right)^2 dxdy \text{ with } (w_g = \alpha)
\end{aligned} \tag{10.3}$$

10.1.2 Variational Approach

The second step in the iFEM approach is using the least-square variation principle on the functional Φ_e for minimizing the error between the analytical and experimental solution.

$$\frac{\partial \Phi_e\left(\mathbf{u}^e\right)}{\partial \mathbf{u}^e} = 0 \tag{10.4}$$

The variational condition Equation 10.4 dictates that the functional should not vary with a change in any of the kinematic variables vector $\mathbf{u}^e$. This is one of the conditions for enabling a system with a Total Potential Energy in equilibrium, thus a compatible system for the iFEM problem.

10.2 Inverse Element Formulation

As mentioned in subsection 7.4.3, previous work was put into extending the library of inverse elements. Although it would be more simple to use beam elements for this application, the main issue with them is not allowing for a lot of flexibility in terms of configuring the sensor network as only the number of elements along the length of the gangway could be changed.

Three types of inverse shell element formulations are currently available in the literature. iMIN3 is a three-node plate element [Tessler and Spangler, 2004]. IQS4 is a 4-node quadrilateral shell element [Kefal et al., 2016]. IQS8 is an inverse curved shell element employing 8 nodes [Kefal, 2019].

Out of the three options for inverse plate elements, Inverse Quadrilateral Shell 4 Points (IQS4) will be investigated. Firstly, its accuracy exceeds that of IMIN3 [Abdollahzadeh et al., 2020] and the geometry of the current application is regular enough for being meshed with quadrilaterials. The geometry is also flat and the deformation pattern has a low omplexity, not requiring the use of the more computationally demanding 8-node element.

Moreover, IQS4 is the correspondent of the Nastran's CQUAD4 element that is used for modelling in the Finite Element Methods (FEM) of the current composite gangway design (will be explained further in section 10.6). This allows for better correspondence between the FEM and iFEM formulations. This allows the use of the same mesh in both the iFEM and FEM model and hopefully reduces the sources of error as the correspondence between the iFEM and FEM models is increased. Thus, the IQS4 formulation and its particularities will be discussed in this chapter.

10.2.1 Element Overview

The IQS4 element has 4 nodes, each defined by 6 Degree of Freedom (DOF) as shown in Figure 10.1. The formulation also includes a drilling degree of freedom for reducing the effect of shear-locking (artificial shear created in in-plane bending) and overall improvement membrane deformation reconstruction [Abdollahzadeh et al., 2020].

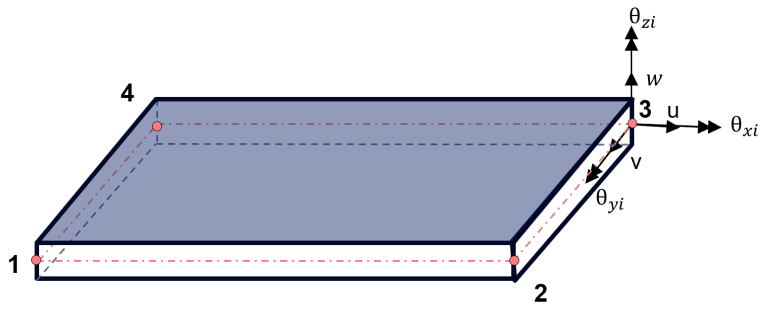


Figure 10.1: Visual representations of IQS4 element DOF.

The IQS4 is based on the Mindlin kinematic framework, which imposes the following assumptions:

MINDLIN-1 The displacement varies linearly across the thickness of the plate.

MINDLIN-2 Thickness remains unchanged throughout loading.

MINDLIN-3 Normals to the plate do not remain perpendicular to the mid-plane surface after deformation.

For each IQS4, the element nodal displacement vector is defined using Equation 10.5.

$$\mathbf{u}^e = \begin{bmatrix} \mathbf{u}_1^e & \mathbf{u}_2^e & \mathbf{u}_3^e & \mathbf{u}_4^e \end{bmatrix}^T \tag{10.5}$$

$\mathbf{u}_i^e$ vectors are defined using Equation 10.6 and the definitions of kinematic variables as illustrated in Figure 10.1.

$$\mathbf{u}^e = \begin{bmatrix} u_i & v_i & w_i & \theta_{xi} & \theta_{yi} & \theta_{zi} \end{bmatrix}^T \tag{10.6}$$

Following assumption MINDLIN-1, the displacement's components can be written as Equation 10.7. u and v are the displacements at the midplane.

$$\begin{aligned} u_x(x,y,z) &\equiv u_x = u + z\theta_y \\ u_y(x,y,z) &\equiv u_y = v - z\theta_x \\ u_z(x,y,z) &\equiv u_z = w \end{aligned} \tag{10.7}$$

The analytical strains can be generated through the derivation of the displacement components described in Equation 10.7 and are shown in Equation 10.8. The presence of the transverse shear strains γ_{xz} and γ_{yz} is permitted by the assumption MINDLIN-3. The ε_{zz} strain component is omitted in the formulation as it is set to 0 by assumption MINDLIN-2.

$$\begin{aligned} \varepsilon_{xx} &= \frac{\partial u_x}{\partial x} = \frac{\partial u}{\partial x} + z\frac{\partial \theta_y}{\partial x} \\ \varepsilon_{yy} &= \frac{\partial u_y}{\partial y} = \frac{\partial v}{\partial y} - z\frac{\partial \theta_x}{\partial y} \\ \gamma_{xy} &= \frac{\partial u_y}{\partial x} + \frac{\partial u_x}{\partial y} = \frac{\partial v}{\partial x} + \frac{\partial u}{\partial y} + z\left(\frac{\partial \theta_y}{\partial y} - \frac{\partial \theta_x}{\partial x}\right) \\ \gamma_{xz} &= \frac{\partial u_z}{\partial x} + \frac{\partial u_x}{\partial z} = \frac{\partial w}{\partial x} + \theta_y \\ \gamma_{yz} &= \frac{\partial u_z}{\partial y} + \frac{\partial u_y}{\partial z} = \frac{\partial w}{\partial y} - \theta_x \end{aligned} \tag{10.8}$$

The previous set of equations can be written in a compact form as shown in Equation 10.9. This is done through the use of strain-displacement matrices $\mathbf{B^{m,b,s}}$ and the division in membrane $\mathbf{e}(\mathbf{u}^e)$, bending $\mathbf{k}(\mathbf{u}^e)$ and transverse $\mathbf{g}(\mathbf{u}^e)$ strain components.

$$\begin{aligned} \begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \gamma_{xy} \end{Bmatrix} &\equiv \mathbf{e}(\mathbf{u}^e) + z\mathbf{k}(\mathbf{u}^e) = \mathbf{B}^m\mathbf{u}^e + z\mathbf{B}^b\mathbf{u}^e \\ \begin{Bmatrix} \gamma_{xz} \\ \gamma_{yz} \end{Bmatrix} &\equiv \mathbf{g}(\mathbf{u}^e) = \mathbf{B}^s\mathbf{u}^e \end{aligned} \tag{10.9}$$

The strain-displacement matrices will be further discussed in subsection 10.2.4.

10.2.2 Shape Functions

Before giving a formulation of the strain-displacement **B**, a discussion on the shape functions of IQS4 is required. Shape functions are used for describing the behaviour of the element. They allow for generating approximation functions for the kinematic variables. They essentially allow for interpolation throughout the element. [Cook, 1994] introduced the shape functions for a quadrilateral element which takes into account coupling with the drilling rotation. These shape functions are also used for the IQS4 formulation.

Shape functions require a switch to a natural coordinate system as illustrated in Figure 10.2. The natural coordinates s,t are defined over a [-1,1] interval.

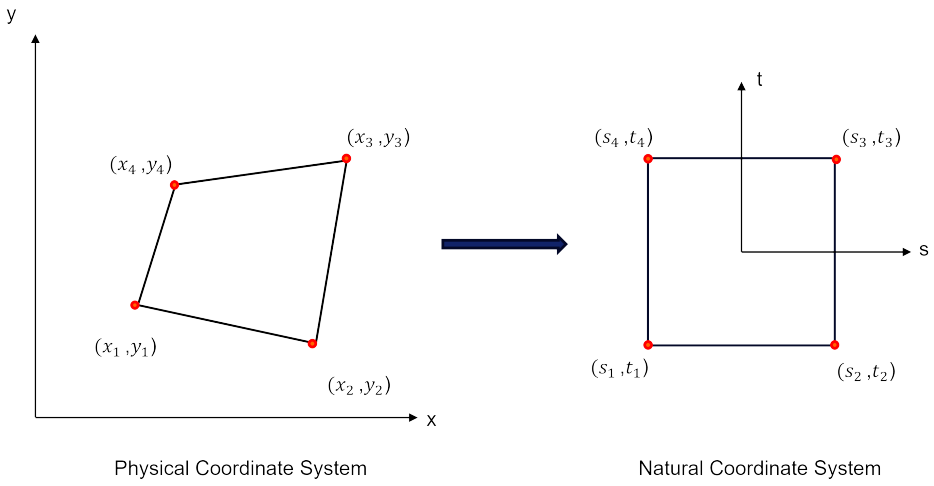


Figure 10.2: Natural Coordinate System

The isoparametric shape functions $N_{1,2,3,4}$ can be used as geometry mapping functions from the natural coordinate system to the local coordinate system.

$$\begin{aligned} x(s,t) \equiv x &= \sum_{i=1}^{4} N_i x_i \\ y(s,t) \equiv y &= \sum_{i=1}^{4} N_i y_i \end{aligned} \tag{10.10}$$

Starting from Equation 10.7, the in-plane translational displacements can be rewritten as:

$$\begin{aligned} u(x,y) \equiv u &= \sum_{i=1}^{4} N_i u_i + \sum_{i=1}^{4} L_i \theta_{zi} \\ v(x,y) \equiv v &= \sum_{i=1}^{4} N_i v_i + \sum_{i=1}^{4} M_i \theta_{zi} \end{aligned} \tag{10.11}$$

The interpolation of the out-of-plane displacement captures also the membrane-bending coupling caused by both nodal rotations θ_{xi} θ_{yi}.

$$w(x,y) \equiv w = \sum_{i=1}^{4} N_i w_i - \sum_{i=1}^{4} L_i \theta_{xi} - \sum_{i=1}^{4} M_i \theta_{yi} \tag{10.12}$$

The interpolation of the rotational kinematic variables is done solely through the isoparametric shape functions:

$$\begin{aligned}\theta_x(x,y) \equiv \theta_x &= \sum_{i=1}^{4} N_i \theta_{xi} \\ \theta_y(x,y) \equiv \theta_y &= \sum_{i=1}^{4} N_i \theta_{yi}\end{aligned} \tag{10.13}$$

The complete definition of the shape functions is provided as follows:

$$\begin{aligned}N_1 &= \frac{(1-s)(1-t)}{4} \\ N_2 &= \frac{(1+s)(1-t)}{4} \\ N_3 &= \frac{(1+s)(1+t)}{4} \\ N_4 &= \frac{(1-s)(1+t)}{4}\end{aligned} \tag{10.14}$$

$$\begin{aligned}N_5 &= \frac{\left(1-s^2\right)(1-t)}{16} \\ N_6 &= \frac{(1+s)\left(1-t^2\right)}{16} \\ N_7 &= \frac{\left(1-s^2\right)(1+t)}{16} \\ N_8 &= \frac{(1-s)\left(1-t^2\right)}{16}\end{aligned} \tag{10.15}$$

$$\begin{aligned}L_1 &= y_{14}N_8 - y_{21}N_5 \\ L_2 &= y_{21}N_5 - y_{32}N_6 \\ L_3 &= y_{32}N_6 - y_{43}N_7 \\ L_4 &= y_{43}N_7 - y_{14}N_8\end{aligned} \tag{10.16}$$

$$\begin{aligned}M_1 &= x_{41}N_8 - x_{12}N_5 \\ M_2 &= x_{12}N_5 - x_{23}N_6 \\ M_3 &= x_{23}N_6 - x_{34}N_7 \\ M_4 &= x_{34}N_7 - x_{41}N_8\end{aligned} \tag{10.17}$$

$$\left.\begin{aligned}x_{ij} &= x_i - x_j \\ y_{ij} &= y_i - y_j\end{aligned}\right\} \quad (i = 1,2,3,4; j = 1,2,3,4) \tag{10.18}$$

10.2.3 Strain Location

As can be seen in Equation 10.1, both the analytical and the experimental strain need to be divided in components. Depending on the loading condition, the strain placement with

respect to the top and bottom surface of the plane can be adjusted. For strictly in-plane loading, sensors on just one side of the plate element can be sufficient. Nevertheless, for cases where bending is also involved, sensors on only one side cannot capture also the bending strain components, leading to insufficient data for reconstruction.

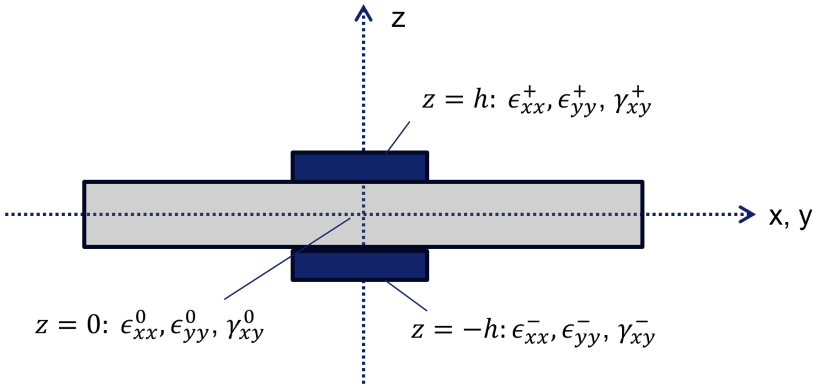


Figure 10.3: Strain sensor placement across plate.

Thus, a two-sided configuration as illustrated in Figure 10.3 is a preferred approach for cases where bending is involved such as the gangway. Based on these two measurements, the membrane strain and curvature can be reconstructed at the mid-plane level using Equation 10.19 and Equation 10.20.

$$\varepsilon_0 = \frac{\varepsilon^+ + \varepsilon^-}{2} \tag{10.19}$$

$$\kappa_0 = \frac{\kappa^+ - \kappa^-}{2h} \tag{10.20}$$

10.2.4 Matrix Formulation

Using the shape functions defined in subsection 10.2.2, the strain-displacement matrices $\mathbf{B}_i^b, \mathbf{B}_i^m, \mathbf{B}_i^s$ introduced in Equation 10.9 can be defined in their matrix notation.

$$\mathbf{B}_i^m = \begin{bmatrix} N_{i,x} & 0 & 0 & 0 & 0 & L_{i,x} \\ 0 & N_{i,y} & 0 & 0 & 0 & M_{i,x} \\ N_{i,y} & N_{i,x} & 0 & 0 & 0 & L_{i,y} + M_{i,y} \end{bmatrix} \tag{10.21}$$

$$\mathbf{B}_i^b = \begin{bmatrix} 0 & 0 & 0 & 0 & N_{i,x} & 0 \\ 0 & 0 & 0 & -N_{i,y} & 0 & 0 \\ 0 & 0 & 0 & -N_{i,x} & N_{i,y} & 0 \end{bmatrix} \tag{10.22}$$

$$\mathbf{B}_i^s = \begin{bmatrix} 0 & 0 & N_{i,x} & -L_{i,x} & -M_{i,x} + N_i & 0 \\ 0 & 0 & N_{i,y} & -L_{i,y} - N_i & -M_{i,y} & 0 \end{bmatrix} \tag{10.23}$$

It is also possible to rewrite the set of equations obtained from the variational principle in a matrix formulation as shown in Equation 10.24.

$$\begin{aligned} \frac{\partial \Phi_e\left(\mathbf{u}^e\right)}{\partial \mathbf{u}^e} &= \mathbf{k}^e\mathbf{u}^e - \mathbf{f}^e = 0 \\ \mathbf{k}^e\mathbf{u}^e &= \mathbf{f}^e \end{aligned} \tag{10.24}$$

Combining 10.2 ,10.9, 10.21, 10.22, 10.23 the following notations can be obtained.

$$\mathbf{k}^e = \iint_{A^e} \left(w_e \left(\mathbf{B}^m\right)^T \mathbf{B}^m + w_k (2h)^2 \left(\mathbf{B}^b\right)^T \mathbf{B}^b + w_g \left(\mathbf{B}^s\right)^T \mathbf{B}^s \right) dxdy \tag{10.25}$$

$$\mathbf{f}^e = \frac{1}{n} \iint_{A^e} \sum_{i=1}^{n} \left(w_e \left(\mathbf{B}^m\right)^T \mathbf{e}_i^\varepsilon + w_k (2h)^2 \left(\mathbf{B}^b\right)^T \mathbf{k}_i^\varepsilon + w_g \left(\mathbf{B}^s\right)^T \mathbf{g}_i^\varepsilon \right) dxdy \tag{10.26}$$

The **k,f** notations are used to highlight the similarity to direct FEM. However, the meaning of the matrices is different compared to FEM. **k** does not represent anymore a stiffness matrix as it does not actually contain any material properties. And **f** is not the standard load matrix. The literature is not consistent in nomenclature for either of these matrices in the context of iFEM. In this report, $\mathbf{k}^e$ will be referred to as the analytical element matrix and $\mathbf{f}^e$ as the input element matrix (it encapsulates the strain data).

10.2.5 Coordinate System Transformation

$\mathbf{k}^e$ and $\mathbf{f}^e$ are formulated in the local coordinate system of each element. In order to be assembled in the global matrix **K** and **F**, they need to be rotated to the global coordinate system and added to their corresponding positions as dictated by the DOF.

$$\mathbf{K} = \sum_{e=1}^{n_{el}} (\mathbf{T^e})^T \mathbf{k}^e \mathbf{T^e} \tag{10.27}$$

$$\mathbf{F} = \sum_{e=1}^{n_{el}} (\mathbf{T^e})^T \mathbf{f}^e \tag{10.28}$$

$\mathbf{T^e}$ is defined as a square matrix with 24 DOF:

$$\mathbf{T^e} = \begin{bmatrix} \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 0 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{T} \end{bmatrix} \tag{10.29}$$

Here, **T** is defined in Equation 10.30 as the vector allowing transformation from the local to the global coordinate system per [Oboe et al., 2021b].

$$\mathbf{T} = \begin{bmatrix} \mathbf{l}^T & \mathbf{p}^T & \mathbf{n}^T \end{bmatrix} \tag{10.30}$$

$\mathbf{n}$ and $\mathbf{p}$ describe the in-plane vectors, while $\mathbf{l}$ is the normal on the plane. The x_{ij} terms are represented by $x_i - x_j$.

$$\mathbf{n} = \frac{x_{31} \times x_{42}}{||x_{31} \times x_{42}||} \tag{10.31}$$

$$\mathbf{p} = \frac{x_{31} + x_{42}}{||x_{31} + x_{42}||} \tag{10.32}$$

$$\mathbf{l} = \mathbf{p} \times \mathbf{n} \tag{10.33}$$

[Oboe et al., 2021b] also gives a method for calculating the centroid of a general quadrilateral. The coordinates of the centroid are given in Equation 10.34.

$$\mathbf{C} = \frac{\sum_{\alpha=1}^{4} c_\alpha d_\alpha}{\sum_{\alpha=1}^{4} d_\alpha} \tag{10.34}$$

c_α describes the mid-point of every edge and d_α the length for each edge using a cyclic rotation on the coordinates of the edges $\mathbf{X}$ using α =1,2,3,4 and β=2,3,4,1.

$$c_\alpha = \frac{\mathbf{X}_\beta + \mathbf{X}_\alpha}{2} \tag{10.35}$$

$$d_\alpha = ||\mathbf{X}_\beta - \mathbf{X}_\alpha|| \tag{10.36}$$

The determination of the centroid is relevant for switching to the natural coordinate system which has its origin in the centroid itself. This is used when filling in the strain-displacement $\mathbf{B}$ matrices and the shape functions ($\mathbf{N}$, $\mathbf{L}$ and $\mathbf{M}$) when performing integrations as described later on subsection 10.5.1.

10.2.6 Boundary Conditions

Using our global matrices $\mathbf{K}$ and $\mathbf{F}$, the system Equation 10.37 could be solved for $\mathbf{U}$.

$$\mathbf{KU} = \mathbf{F} \tag{10.37}$$

Nevertheless, just like in direct FEM, iFEM should also include the kinematic boundary conditions and constraints. By removing the entries corresponding to them, the system can be reduced to its unknown form:

$$\begin{aligned} \mathbf{K_u U_u} &= \mathbf{F_u} \\ \mathbf{U_u} &= \mathbf{K_u}^{-1} \mathbf{F_u} \end{aligned} \tag{10.38}$$

From equations 10.25 and 10.26, it can be also noticed that only $\mathbf{F}$ depends on the strain measurements. $\mathbf{K}$ however is constant and only depends on the geometry, element formulation and sensor network, allowing for doing the inverting operation only once.

10.3 Smoothed Element Analysis

As discussed in section 8.1, the current project will also look into the performance of Smoothed Inverse Finite Element Methods (iFEM(s)) in this application. Smoothing Element Analysis (SEA) will be for strain pre-extrapolation due to its superior performance with respect to other methods such as polynomial interpolation. This superiority is enabled by its adaptivity to different complexity levels of the strain field [Oboe et al., 2021a] and its development based on the FEM framework.

10.3.1 Overview

[Kefal et al., 2021b], [Oboe et al., 2021a] are studies in which SEA is implemented using triangular elements. There is one study [Minigher et al., 2022] which proposes a quadrilateral SEA element. The authors report better predictability of k_{ψ_z} behavior when the drilling degree of freedom is included in the quadrilateral element compared to the triangular one.

The inclusion of the drilling degree of freedom in the current application is important due to the side walls which experience in-plane bending. Moreover, the presence of this DOF allows for proper transformation to the global coordinate system for 3D models [Minigher et al., 2022]. Moreover, the addition of the drilling degree of freedom also allows for the use of the same shape functions $\mathbf{N}, \mathbf{L}, \mathbf{M}$ as in IQS4.

10.3.2 Quadrilateral SEA Element Formulation

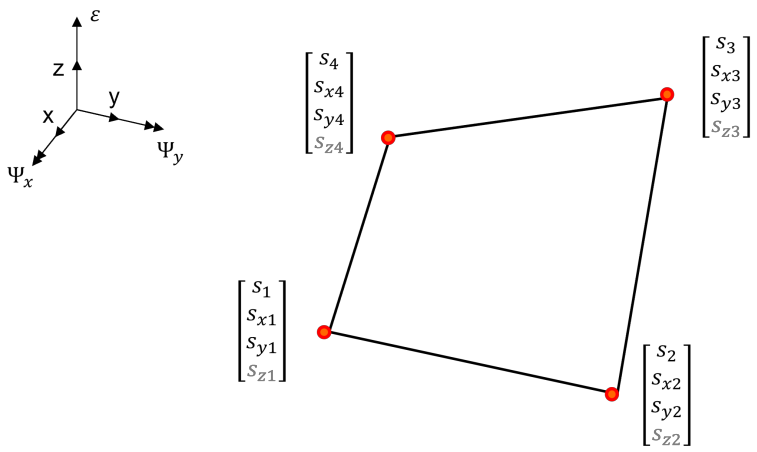


Figure 10.4: SEA 4-node element

Figure 10.4 gives an overview of the smoothed quadrilateral element. The degrees of freedom are defined as s, s_x, s_y, s_z. s_z is illustrated in gray as it is an artificial DOF that is added in later steps in the matrices of interest. Thus, in the following derivations the DOF vector is defined as $\mathbf{u^e_{SEA}} = [\mathbf{s s_x s_y}]$.

In some literature, such as [Oboe et al., 2021b] s_x, s_y are sometimes referred as θ_x, θ_y which can cause confusion. These variables simply represent the first derivatives of the extrapolated strain field and do not have any correlation with the rotational degrees of freedom.

This allows for interpolation of the ε field at any of the points. Moreover, it can be seen that strain is not divided in components in the SEA formulation. This is due to the fact that the pre-extrapolation is done for each strain component independently. Thus, the DOF of SEA can be interpolated using Equation 10.39.

$$\begin{aligned} \varepsilon &= \mathbf{N}\mathbf{s} - \mathbf{L}\mathbf{s}_{\mathbf{x}} - \mathbf{M}\mathbf{s}_{\mathbf{y}} \\ \psi_x &= \mathbf{N}\mathbf{s}_{\mathbf{x}} \\ \psi_y &= \mathbf{N}\mathbf{s}_{\mathbf{y}} \end{aligned} \tag{10.39}$$

Just like iFEM, SEA can be characterized by a general functional Equation 10.40. An additional SEA subscript will be added to avoid confusion. [Minigher et al., 2022] warns that $\Phi^{(e)}_{SEA}$ can be slightly different than other formulations such as [Oboe et al., 2021a]. The difference occurs due to the definitions within the quadrilateral framework concerning the rotation coupling, which can be noticed at the level of the transverse shear strains γ_{xz}, γ_{yz}.

When comparing with the iFEM functional in Equation 10.1, it can be seen that the discrete points $\mathbf{e}^{\varepsilon}$ has switched to $\varepsilon(\mathbf{x}_i)$ suggesting a continuous strain domain in SEA.

$$\begin{aligned} \Phi^{(e)}_{SEA} =& \frac{1}{N}\sum_{i=1}^{n^{(e)}} \left[\varepsilon_i^{\varepsilon} - \varepsilon(\mathbf{x}_i)\right]^2 \\ &+ \alpha \iint_{A^{(e)}} \left[\left(\frac{\partial \varepsilon}{\partial x} + \psi_y\right)^2 + \left(\frac{\partial \varepsilon}{\partial y} - \psi_x\right)^2\right] \mathrm{d}A^{(e)} + \\ &+ \beta A^{(e)} \iint_{A^{(e)}} \left[\left(\frac{\partial \psi_x}{\partial x}\right)^2 + \left(\frac{\partial \psi_y}{\partial y}\right)^2 + \frac{1}{2}\left(\frac{\partial \psi_x}{\partial y} + \frac{\partial \psi_y}{\partial x}\right)^2\right] \mathrm{d}A^{(e)} \end{aligned} \tag{10.40}$$

Equation 10.40 can be written in a compacted form such as Equation 10.41.

$$\Phi^{(e)}_{SEA} = \Phi_{\varepsilon} + \Phi_{\alpha} + \Phi_{\beta} \tag{10.41}$$

Next, each of the $\Phi_{\varepsilon}, \Phi_{\alpha}, \Phi_{\beta}$ components will be expanded. Just like in the regular iFEM formulation, the variational principle with respect to the $\mathbf{u}^{\mathbf{e}}_{\mathbf{SEA}}$ will be applied. After derivation, the results are rewritten in the classical FEM form $\mathbf{ku} = \mathbf{f}$. As can be seen in Equation 10.40, Φ_{ε} is the only component containing experimental strain $\varepsilon_i^{\varepsilon}$, thus the only component that will have a non-zero $\mathbf{f}$. Starting with Φ_{ε}:

$$\begin{aligned} \Phi_{\varepsilon} &= \frac{1}{N}\sum_i [\varepsilon_i^{\varepsilon} - \varepsilon(\mathbf{u}^{\mathbf{e}})]^2 \\ &= \frac{1}{N}\sum_i [\varepsilon_i^{\varepsilon} - [\mathbf{N} - \mathbf{L} - \mathbf{M}]\mathbf{u}^{\mathbf{e}}]^2 \\ &= \frac{1}{N}\sum_i [(\varepsilon_i^{\varepsilon})^2 + \mathbf{u}^{\mathbf{e}T}\tilde{\mathbf{N}}^T\tilde{\mathbf{N}}\mathbf{u}^{\mathbf{e}} - 2\varepsilon_i^{\varepsilon}\tilde{\mathbf{N}}\mathbf{u}^{\mathbf{e}}] \end{aligned} \tag{10.42}$$

where $\tilde{\mathbf{N}}$ is defined through Equation 10.43.

$$\tilde{\mathbf{N}} = \begin{bmatrix} \mathbf{N} & -\mathbf{L} & -\mathbf{M} \end{bmatrix} \tag{10.43}$$

$$\mathbf{k}_\varepsilon \mathbf{u}^\mathbf{e} = \mathbf{f}_\varepsilon \tag{10.44}$$

$$\mathbf{k}_\varepsilon = \frac{1}{N}\sum_i \tilde{\mathbf{N}}^T \tilde{\mathbf{N}} \tag{10.45}$$

$$\mathbf{f}_\varepsilon = \frac{1}{N}\sum_i \varepsilon_i^\xi \tilde{\mathbf{N}} \tag{10.46}$$

Φ_α can be expanded into Equation 10.47.

$$\begin{aligned}\Phi_\alpha &= \iint_{A^{(e)}} \left[\left(\frac{\partial \varepsilon}{\partial x} + \psi_y\right)^2 + \left(\frac{\partial \varepsilon}{\partial y} - \psi_x\right)^2\right] \mathrm{d}A^{(e)} \\ &= \iint_{A^{(e)}} \left[\left[\begin{array}{cc} \frac{\partial \mathbf{N}}{\partial x} & -\frac{\partial \mathbf{L}}{\partial x}\left(-\frac{\partial \mathbf{M}}{\partial x} + \mathbf{N}\right) \end{array}\right] \mathbf{u}^{(e)}\right]^2 + \left[\left[\begin{array}{cc} \frac{\partial \mathbf{N}}{\partial y} & \left(-\frac{\partial \mathbf{L}}{\partial y} - \mathbf{N}\right) \quad -\frac{\partial \mathbf{M}}{\partial y} \end{array}\right] \mathbf{u}^{(e)}\right]^2 \mathrm{d}A^{(e)}\end{aligned} \tag{10.47}$$

By applying the variational principle and rewriting in the form:

$$\mathbf{k}_\alpha \mathbf{u}^\mathbf{e} = \mathbf{0} \tag{10.48}$$

Equation 10.47 can be rewritten in the compacted form Equation 10.48.

$$\mathbf{k}_\alpha = \iint_{A^e} \left(\mathbf{B_1}^T \mathbf{B_1} + \mathbf{B_2}^T \mathbf{B_2}\right) \tag{10.49}$$

Matrices $\mathbf{B_1}$ and $\mathbf{B_2}$ are simply defined by equations Equation 10.50. (they do not have a relation to the strain-displacement $\mathbf{B}$ matrices)

$$\begin{aligned}\mathbf{B_1} &= \left[\begin{array}{cc} \frac{\partial \mathbf{N}}{\partial x} & -\frac{\partial \mathbf{L}}{\partial x}\left(-\frac{\partial \mathbf{M}}{\partial x} + \mathbf{N}\right) \end{array}\right] \\ \mathbf{B_2} &= \left[\begin{array}{cc} \frac{\partial \mathbf{N}}{\partial y} & \left(-\frac{\partial \mathbf{L}}{\partial y} - \mathbf{N}\right) \quad -\frac{\partial \mathbf{M}}{\partial y} \end{array}\right]\end{aligned} \tag{10.50}$$

Lastly, the Φ_β can be tackled.

$$\begin{aligned}\Phi_\beta &= \iint_{A^{(e)}} \left[\left(\frac{\partial \psi_x}{\partial y}\right)^2 + \left(\frac{\partial \psi_y}{\partial x}\right)^2 + \frac{1}{2}\left(\frac{\partial \psi_x}{\partial y} + \frac{\partial \psi_y}{\partial x}\right)^2\right] \mathrm{d}A^{(e)} \\ &= \iint_{A^{(e)}} \left[\left(\frac{\partial \mathbf{N}}{\partial y}\mathbf{s}_x\right)^2 + \left(\frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_y\right)^2 + \frac{1}{2}\left(\frac{\partial \mathbf{N}}{\partial y}\mathbf{s}_x + \frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_y\right)^2\right] \mathrm{d}A^{(e)} \\ &= \iint_{A^{(e)}} \left[\mathbf{s}_x^\top \left(\frac{\partial \mathbf{N}}{\partial y}\right)^\top \frac{\partial \mathbf{N}}{\partial y}\mathbf{s}_x + \mathbf{s}_y^T \left(\frac{\partial \mathbf{N}}{\partial x}\right)^T \frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_y + \frac{1}{2}\left(\left(\frac{\partial \mathbf{N}}{\partial y}\mathbf{s}_x\right)^2 + \left(\frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_y\right)^2 + 2\frac{\partial \mathbf{N}}{\partial y}\mathbf{s}_x\frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_y\right)\right] \mathrm{d}A^{(e)}\end{aligned} \tag{10.51}$$

Due to the lack of $\mathbf{u}^\mathbf{e}_\mathbf{SEA}$ as a whole, separate derivations for the $\mathbf{s_x}$ and $\mathbf{s_y}$ DOF that are present need to be done, as shown in equations 10.52 and 10.53.

$$\begin{aligned}\frac{\partial \Phi_\beta}{\partial \mathbf{s}_x} &= \iint_{A^{(e)}} \left[2\left(\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial y}\right)\mathbf{s}_x + \left(\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial y}\right)\mathbf{s}_x + \frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_y \right] \mathrm{d}A^{(e)} \\ &= \iint_{A^{(e)}} \begin{bmatrix} 0 & \left(\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial y} + \frac{1}{2}\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial y}\right) & \frac{1}{2}\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial x} \end{bmatrix} \mathbf{u}^{(e)} \mathrm{d}A^{(e)}\end{aligned} \tag{10.52}$$

$$\begin{aligned}\frac{\partial \Phi_\beta}{\partial \mathbf{s}_y} &= \iint_{A^{(e)}} \left[2\left(\frac{\partial \mathbf{N}^\top}{\partial x}\frac{\partial \mathbf{N}}{\partial x}\right)\mathbf{s}_y + \left(\frac{\partial \mathbf{N}^\top}{\partial x}\frac{\partial \mathbf{N}}{\partial x}\right)\mathbf{s}_y + \frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial x}\mathbf{s}_x \right] \mathrm{d}A^{(e)} \\ &= \iint_{A^{(e)}} \begin{bmatrix} 0 & \frac{1}{2}\frac{\partial \mathbf{N}^T}{\partial y}\frac{\partial \mathbf{N}}{\partial x} & \left(\frac{\partial \mathbf{N}^\top}{\partial x}\frac{\partial \mathbf{N}}{\partial x} + \frac{1}{2}\frac{\partial \mathbf{N}^\top}{\partial x}\frac{\partial \mathbf{N}}{\partial x}\right) \end{bmatrix} \mathbf{u}^{(e)} \mathrm{d}A^{(e)}\end{aligned} \tag{10.53}$$

By using the form:

$$\mathbf{k}_\beta \mathbf{u}^{(e)} = \mathbf{0} \tag{10.54}$$

$\mathbf{k}_\beta$ can be defined as:

$$\mathbf{k}_\beta = \iint_{A^{(e)}} \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \frac{3}{2}\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial y} & \frac{1}{2}\frac{\partial \mathbf{N}^T}{\partial y}\frac{\partial \mathbf{N}}{\partial x} \\ \mathbf{0} & \frac{1}{2}\frac{\partial \mathbf{N}^\top}{\partial y}\frac{\partial \mathbf{N}}{\partial x} & \frac{3}{2}\frac{\partial \mathbf{N}^\top}{\partial x}\frac{\partial \mathbf{N}}{\partial x} \end{bmatrix} \mathrm{d}A^{(e)} \tag{10.55}$$

Finally, after defining all the components they can all be put together using the

$$\mathbf{k}^{\mathbf{e}}_{\mathbf{SEA}} = \mathbf{k}_\varepsilon + \alpha \mathbf{k}_\alpha + \beta \mathbf{k}_\beta \tag{10.56}$$

$$\mathbf{f}^{\mathbf{e}}_{\mathbf{SEA}} = \mathbf{f}_\varepsilon \tag{10.57}$$

After constructing the element matrices $\mathbf{k}^{\mathbf{e}}_{\mathbf{SEA}}$ and $\mathbf{f}^{\mathbf{e}}_{\mathbf{SEA}}$, the artificial drilling degree of freedom s_z can be added. This is done through entries equal to the drilling stiffness k_{Ψ_β} in the $\mathbf{k}^{\mathbf{e}}_{\mathbf{SEA}}$ and 0 entries in the $\mathbf{f}^{\mathbf{e}}_{\mathbf{SEA}}$ matrix. Thus, from a 12x12 matrix, $\mathbf{k}^{\mathbf{e}}_{\mathbf{SEA}}$ turns into a 16x16 matrix and from 12x1, $\mathbf{f}^{\mathbf{e}}_{\mathbf{SEA}}$ gets a shape of 16x1.

Just like for regular iFEM, the local matrices need to be transformed to the global coordinate system through transformations 10.27 and 10.28. However, for computing the $\mathbf{U}_{\mathbf{SEA}}$, the inclusion of the boundary conditions is not mandatory. [Minigher et al., 2022] describes that this is due to the fact that the $\Phi^{(e)}_{SEA}$ simply represents an interpolation error minimization, rather than a Total Potential Energy minimization like in the one used in direct FEM.

10.3.3 Involved Parameters

This subsection will describe the parameters α,β,k_{Ψ_β}.

- α is a hyperparameter that defines the smoothness of the interpolation. A higher value of α allows for a smoother strain field while a smaller value of α permits for a better fit of the interpolated function through the control points. This effect is documented in [Oboe et al., 2021a]. Multiple sources indicate that the contribution of α [Oboe et al., 2021a], to the iFEM(s) is critical. Thus, α will be investigated.
- β controls the curvature of the interpolated function. Multiple studies such as [Oboe et al., 2021a], [Minigher et al., 2022] highlight that it has a negligible contribution to the quality of the results compared to α. In the current study, the effect of β was not investigated. Based on the literature, a factor of β of 10^{-4} was assumed.
- k_{Ψ_β} is the artificial drilling stiffness. It is a penalty parameter that allows for reducing singularity issues when the drilling degree of freedom is implemented in the element formulation. k_{Ψ_β}=10^{-5} was used as suggested in [Minigher et al., 2022], [M. Adam, 2013], . [Minigher et al., 2022] highlights that the variation of k_{Ψ_β} affects the quadrilateral SEA formulation only for very low values of α.

10.4 Method for Verification

In the current work, the verification of iFEM will be done through FEM simulations. This implies that before constructing any iFEM model, a reference FEM model will be built. The strains computed in the direct FEM analysis will be used as mock experimental data for the iFEM algorithm. Afterwards, a comparison between the reference FEM data and the reconstructed data through iFEM can be conducted.

The FEM software used in the current work is FEMAP 2022.1. Unless otherwise mentioned, the analysis uses simple or laminate 4-noded plate elements.

10.5 Implementation Overview

All the literature presented so far on iFEM makes use of in-house developed code implementations that are not made publicly available. Thus, for the purpose of this research, implementing the theoretical background of iFEM and SEA presented in sections section 10.1-section 10.3 was necessary.

It was chosen to use Python (v 3.11.4 is used) for development. This was done for multiple purposes. Firstly, Python is the main programming language at Ampelmann, allowing engineers to use it further and build on it. Secondly, it is an open-source language, allowing anyone to become a potential user.

The developed code uses elemental data patterns inspired by the pyfe3d package [Castro, 2023]. The currently developed iFEM package will be called and referred to as pyife3d.

Some aspects that are not immediately obvious from the previous discussions will be presented in subsections 10.5.1-10.5.4. A quick overview of how hyperparameters w and α are selected is given in subsection 10.5.5, while subsection 10.5.6 highlights the required input for the code. Lastly, subsection 10.5.7 gives a visualization of the implementation.

10.5.1 Integration

It can be seen throughout the methodology that most of the final expressions are expressed in integration form. These integrals are solved numerically through the Gaussian Quadrature as shown in Equation 10.58, where w_i represent weights that are determined based on the number of the used integration points. Throughout this study, unless otherwise mentioned, a 3-point integration is used. Nevertheless, the code implementation offers the possibility of using others as well.

$$f(x) \approx \sum_{i}^{n} w_i f(x_i) \tag{10.58}$$

This method is defined for the natural coordinate system [-1,1]. Thus, a change of variables from the local coordinate system to the natural coordinate system needs to be done. For the coordinates of the element corners, this can be done by relating the point as a distance from the centroid of the element (the new origin of the coordinate system). As for the partial derivates, this can be done with the help of the Jacobian as shown in Equation 10.59.

$$\partial x \partial y \rightarrow |J(\xi, \eta)| ds dt \tag{10.59}$$

Where $|J(\xi, \eta)|$ represents the Jacobian of the local to natural transformation.

$$|J(\xi, \eta)| = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} \end{bmatrix} \tag{10.60}$$

10.5.2 Weights Assigning

It was chosen to treat the w_e,w_k and w_g weights used in the iFEM functional as vectors with independent values for each strain component rather than constants. This allows for the investigation of strain networks with only axial measurements as opposed to being limited to only strain rosettes. The code checks for each element which strain measurements are measured. If the measurement is present, then the weight of the component stays 1. If it is not, then the weight becomes equal to a factor w_f. The same factor w_f is assigned to any missing strain component, no matter if it is membrane or curvature. By default, transverse shear will be set to a low value of $w_f = 10^{-8}$ as they cannot be directly measured without additional error-prone computations [Colombo et al., 2021]. Moreover, for thin shells it is possible to completely omit the transverse shear components from the iFEM formulation due to their small contribution [Kefal et al., 2018], [Abdollahzadeh et al., 2023]

10.5.3 Sensor Placement Correction Factor

A correction is added when integrating over the $\mathbf{f^e}$ in Equation 10.26. It is assumed that for each element the strain is recorded at its centroid. Thus, when the Gaussian integration point does not correspond to (0,0), an alignment factor is added. Its value was aimed to be kept low and in a similar range to typical values of the weight w_f. Thus, its value was set to a fixed value of 10^{-4}.

For $\mathbf{f}^{\mathbf{e}}_{\mathbf{SEA}}$, it can be seen that such a factor is not required by looking at Equation 10.46. In this case, only the actual strain measurements are taken into account, not requiring a correction factor.

10.5.4 Strain pre-extrapolation Usage

As also suggested by the notation, $\mathbf{U_{SEA}}$ is expressed for nodal values. Nevertheless, the interpolated strain needs to be expressed in an elemental value. This can be done by using the relation for ε from Equation 10.39. By filling in the shape functions with the natural coordinate of the centroid in the current case, a single strain value for the element can be obtained. This is verified by the matrix shapes:

$$\varepsilon = \underbrace{\mathbf{N}}_{1\times4}\underbrace{s}_{4\times1} - \underbrace{L}_{1\times4}\underbrace{s_x}_{4\times1} - \underbrace{M}_{1\times4}\underbrace{s_y}_{4\times1} \tag{10.61}$$

10.5.5 Hyperparameters Determination

The determination of the w_f and α parameters is done in a two-step coefficient selection. The upper limit for w_f is limited to 1 by its definition. The other boundary values are recovered from the literature. The iteration for w_f was conducted on the $[10^{-5}; 1]$ ([Kefal et al., 2016], [Tessler and Spangler, 2005], [Oboe et al., 2022]) interval, while for α on the $[10^{-6}; 10^{6}]$ ([Oboe et al., 2021a], [Minigher et al., 2022]).

Especially for α it is a long iteration due to the prolonged duration of running the SEA strain pre-extrapolation. Thus, a logarithmic scale was used for evaluating these intervals. Each coefficient was selected based on its performance which was assessed through a Mean Absolute Percentage Difference (MAPD) as discussed in section 10.8.

Figure 10.5: 2-step coefficient selection iFEM(s)

The following assumptions are made:

PARAM-1 It is assumed that the same optimum w can be used for both the iFEM and iFEM(s) analysises.

PARAM-2 The effect of β and k_{Ψ_β} on the reconstruction performance is considered minimal and is not currently investigated.

10.5.6 Input

In the current implementation, the iFEM code uses the same geometry and mesh discretization as in the direct FEM model.

1. Node coordinates. A file containing the (x,y,z) of all nodes.
2. Element nodes. A file containing the nodes of each element and their corresponding order. Their order is important for the establishment of the local coordinate system as described in section 10.2.
3. Strain Results. A file containing the ID of the element and the associated strains for top and bottom sides of the plate element. Can be only partially filled in.
4. Boundary Condition (BC). The BCs need to be hardcoded for the analyzed case in the **U** vector at the corresponding DOF's.
5. Thickness. The thickness of the plate elements needs to be hardcoded in the iFEM script that is run.
6. Material Direction. This input is optional and should only be given in the case of anisotropic materials. It is defined as a vector which shows the alignment directions of the fibers in the global coordinate system.
7. Reference U. This file is not specifically needed for running the code, but used for computing the performance of the reconstruction. More information on how this is assessed is offered in section 10.7.

The First three inputs can be directly be extracted from FEMAP with the use of VBA.NET Macros. Appendix B gives an example of such a macro and how the input files for the iFEM code need to be formatted.

10.5.7 Layout

The following assumptions/simplifications are made in the current code simplification.

CODE-1 All the plate elements have the same thickness.

CODE-2 The iFEM code uses the same mesh as the FEM model.

CODE-3 The strain is always measured at the centroid of the element.

CODE-4 The input strains are given in the local coordinate system of the elements.

CODE-5 Same α is used for the SEA of all strain components.

10.6 Gangway Structure: Model Simplifications

The structure simplification will be done in an iterative and incremental manner. This means that the starting point will be represented by the least complex simplification of the gangway structure. In each iteration, complexity will be added to the structure. This approach was chosen in order to allow inspection of individual effects and improvement of methods. The goal of the simplification was to replicate as much as possible the behavior of the designed composite gangway, despite the use of a simpler model.

The general approach for each design iteration is the following.

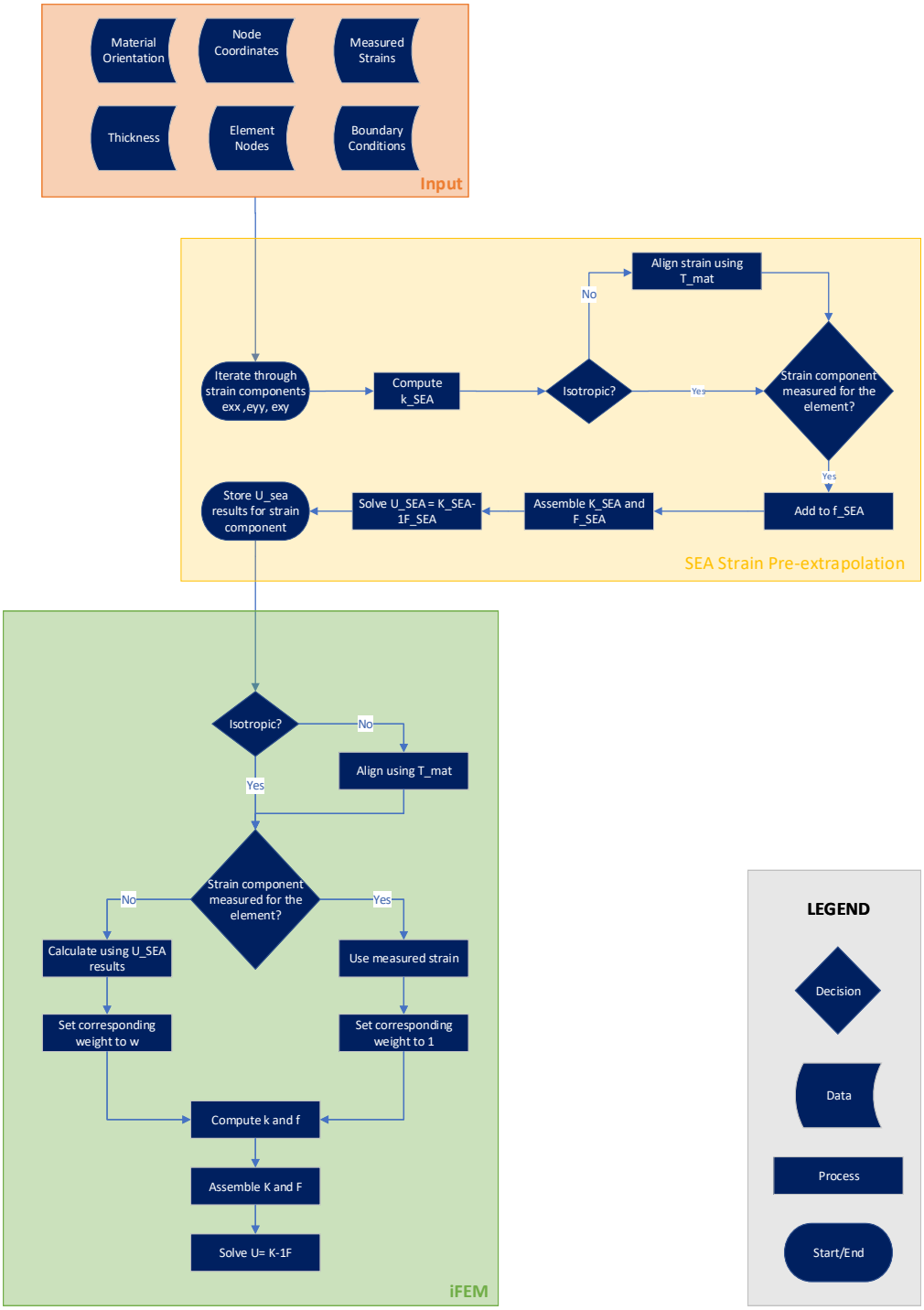


Figure 10.6: Block Diagram of IFEM SEA implementation

1. Aim to recreate as much as possible from the current dimensions of the gangway.
2. Use the thickness of the plate to match the Dead Load Test Load (DLTL) deflection of the designed gangway. The DLTL deflection is chosen for matching as it is the deflection

that is defined per legislation and can also be measured directly through iFEM. The Dead Load (DL) deflection is not regulated explicitly. The Test Load (TL) deflection is. However, it cannot be measured independently as the DL is always acting on the structure in real life. A preliminary thickness is determined through the use of analytical formulas. Afterwards, if necessary it is adapted through iteration based on the results obtained in FEM

3. The density of the material becomes an artificial parameter that is used to match the DL of the designed gangway through the total mass as given in Appendix A.

4. When the material is modeled as isotropic, the equivalent elastic properties of the laminate in Appendix A are used. This is done following relations Equation 10.62 from [Kassapoglou, 2013, p. 51] where the lower case letters a,b,d represent the entries of the matrix $(\mathbf{ABD})^{-1}$. When the material is modelled as anisotropic, only plies will be used for modelling. The foam part of the sandwich structure is disregarded currently as nor solid elements, nor other methods such as Refined Zig-Zag Theory (RZT) for handling are implemented. The low transverse modulus of the core can create contradictions in the first-order shear deformation theory that is assumed by the Mindlin plate [Birman and Genin, 2018].

5. In the case when the material is modeled as anisotropic, the plies are all stacked in the same direction to obtain the required thickness.

$$\begin{aligned} E_{1m} &= \frac{1}{ha_{11}} & E_{1b} &= \frac{12}{h^3 d_{11}} \\ E_{2m} &= \frac{1}{ha_{22}} & E_{2b} &= \frac{12}{h^3 d_{22}} \\ G_{12m} &= \frac{1}{ha_{66}} & G_{12b} &= \frac{12}{h^3 d_{66}} \\ v_{12m} &= -\frac{a_{12}}{a_{22}} & v_{12b} &= -\frac{d_{12}}{d_{22}} \\ v_{21m} &= -\frac{a_{12}}{a_{11}} & v_{21b} &= -\frac{d_{12}}{d_{11}} \end{aligned} \tag{10.62}$$

The assumptions regarding simplifying the gangway iFEM model can be identified as follows:

STR-SIM-1 The tip of the gangway is neglected in the current modelling.

STR-SIM-2 The cutouts on the side-walls are ignored in the geometry modelling. This is due to the complexity induced by the stress redistribution. The effect of this assumption would be especially important when modelling wind loading, as its effect is dependent on the surface area of the profile. Nevertheless, this type of loading is not considered in the current implementation of the Structural Health Monitoring (SHM) system.

STR-SIM-3 Connecting interfaces between the side walls and the gangway deck are ignored.

STR-SIM-4 The telescoping interface is neglected.

STR-SIM-5 Only the TL and DL are considered in a static loading case.

STR-SIM-6 The boundary conditions are modelled as a fixed end.

STR-SIM-7 DL includes only the structural mass of the gangway. Traditionally, the DL also includes the mass of all installed equipment. However, during this point of the composite gangway development, this is not determined and therefore excluded from the current study.

10.7 Sensor Network

The current study is interested in exploring the performance of different sensing networks, and hopefully identifying a suitable candidate for the SHM system. The general guidelines for conducting this task are:

1. Start from a complete set of measurements and reduce the number of sensing points gradually. This initial reconstruction with the set of complete measurements will represent the best accuracy that can be achieved.
2. Assess the performance of individual strain components. In certain load conditions and applications, certain strain components are more relevant than others. Assessing if only one component is sufficient for an acceptable reconstruction, then it is possible to keep the sensor network with only axial measurements. This can be preferred if distributed Fiber Optics (FO) is used for continuous measurements. Nevertheless, if this is not possible, placing Fiber Bragg grating (FBG) strain rosettes remains a viable possibility.
3. As in the current SHM system implementation only the DLTL case is considered, this implies that the gangway is mainly loaded in bending. Thus, the focus will be on line configurations that span along the whole length of the gangway.

The current approach introduces the following assumptions regarding the sensor network.

SENSING-1 The strain is measured at the centroid of the element.

SENSING-2 It is possible to place sensors at both the top and bottom of the plate at the locations of interest.

10.8 Performance Assessment

The performance of a sensor network reconstruction will be assessed using two measures. Firstly, the Percentage Difference (PD) between the maximum deflection in the FEM model and the reconstructed one. $PD_{max(T_3)}$ is the main factor of interest due to the legislation. Ampelmann agreed that a reconstruction for this first SHM implementation should offer an accuracy of 95% for the maximum deflection.

$$PD_{max(T_3)} = \frac{max(T_3)_{iFEM} - max(T_3)_{FEM}}{max(T_3)_{FEM}} \cdot 100 \tag{10.63}$$

Secondly, the MAPD between the FEM deflection field and the reconstructed one. This allows for assessing the performance of the whole reconstruction through a single value. This value

is considered of value to ensure that not only a critical value is replicated, but also the overall behavior of the structural deformation.

$$MAPD_{(T_3)} = \sum_{i=1}^{N_{nodes}} \frac{T_{3_{iFEM}} - T_{3_{FEM}}}{T_{3_{FEM}}} \cdot 100 \tag{10.64}$$

Chapter 11

Results

The current chapter presents and discusses the findings of the current study. section 11.1 highlights some findings obtained during initial verification procedures of the code. Sections 11.2 and 11.3 give an overview of the results of the deck configurations, while section 11.4 tackles the insight gathered when exploring the U-shape geometry.

11.1 Inverse Finite Element Methods (iFEM) Exploration

Before diving into the actual design configurations, an exploration of the iFEM implementation was done on some preliminary cases, including a cantilevered plate under a distributed load. This was chosen due to the representative loading to the gangway.

A cantilevered plate under a distributed load was studied. t=4mm E=72.4 GPa $\nu = 0.33$ [1]. The distributed load q=-1000Pa was chosen arbitrarily with the condition of keeping the deformation in the linear domain, which was considered achieved as a maximum Von Mises stress was 28.7 MPa, while the yield strength of an Aluminum alloy is typically in the range of 300 MPa.

One interesting phenomenon was observed when exploring sensing network options. The location of a sensing line (along the length) across the width of the plate can introduce an artificial torsion effect in the iFEM reconstruction. Three key locations are shown: the edge of the plate, the quarter of the plate and the middle of the plate. All these cases are instrumented with only one strain direction along the lnegth of the plate (defined as x-axis), as it was identified as sufficient for the reconstruction. A finer mesh was chosen in order to allow for space between these locations along the width.

Figures 11.2 to 11.10 show for each of these configurations how the sensors are placed, how the out-of-plane displacement reconstruction based on the iFEM analysis looks like and a map of errors. The map of errors will be used as a reference for the rest of this report. They are created by plotting the Percentage Difference (PD) error between the reconstructed variable

[1] https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma2024t4

and the reference variable from FEMAP. The values between the nodes are interpolated automatically by the function *tricontourf*[2] from Pyhton's matplotlib. Thus, a continuous contour map is obtained.

The twist effect was quantified through a twist angle as defined in Figure 11.1. The twist angle is defined at the mid-plane of the plate and computed at the free tip of the plate x=0.4 [m]. For the *EDGE-LINE-EXX* a 0.613°, *QUARTER-LINE-EXX* 0.446° and *MID-LINE-EXX* 0.063°. Thus, by using the same amount of sensors, simply moving the sensing location can diminish the twist angle by one order of magnitude. This also allows for an overall improvement of the reconstruction, reducing the Mean Absolute Percentage Difference (MAPD) from 5.86 % to 2.83%.

This phenomenon could be explained by the Finite Element Methods (FEM) formulation which imposes a unique value per node. Thus, when a strain is provided for one element, the adjacent elements are also affected as nodes are shared. If the sensing line is placed at the edge of the plate, then there is only one row of adjacent elements that gets influenced by the strain measurements as opposed to two sides (like in the case of the *QUARTER-LINE* and *MID-LINE* configurations).

The effect cascades to the rest of the structure and gets lesser with an increasing distance from the fed strain data. As such, the *MID-LINE* sensing network has a greater effect in propagating the values of the nodes, leading to a better reconstruction. Moreover, it also leads to a more uniform reconstruction which results in less twist.

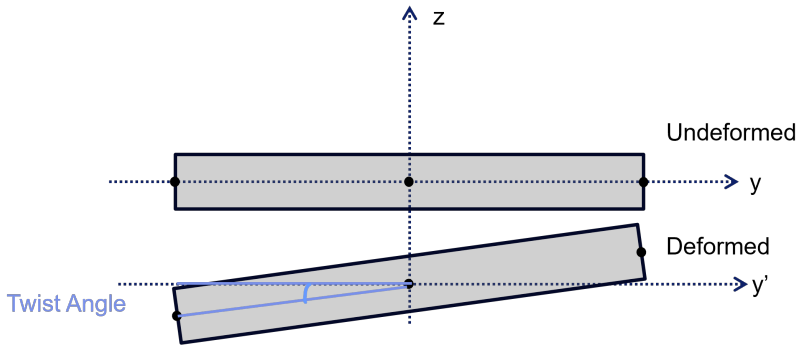


Figure 11.1: Twist angle visualization. The x-axis is aligned with the length of the plate, the y-axis with the width and z-axis with the thickness.

Nevertheless, sometimes there is not much freedom to change the locations of the sensing networks. Thus, it was investigated whether this effect can be minimized in other ways. Subsections 11.1.1 and 11.1.2 explore two alternatives for minimizing this artificial twist phenomenon on the cantilevered plate under distributed load study case.

[2] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.tricontourf.html

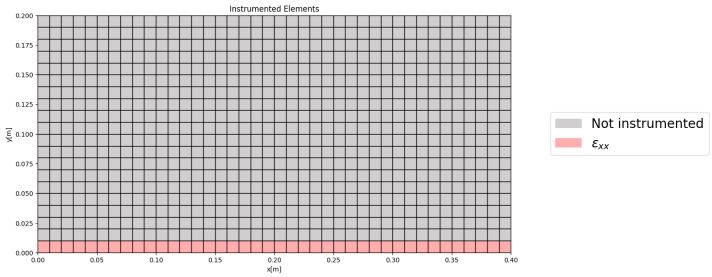


Figure 11.2: Cantilevered plate under distributed load discretized using a 800 element mesh. Visualization of instrumented elements for the *EDGE-LINE-EXX* strain sensing configuration.

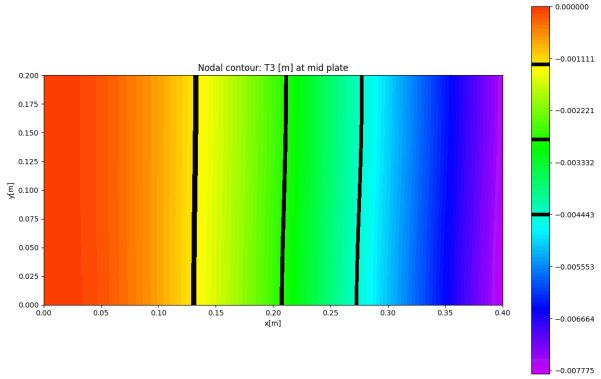


Figure 11.3: Cantilevered plate under distributed load discretized using a 800 element mesh. iFEM reconstruction of Out-of-plane displacement (T3) for the *EDGE-LINE-EXX* strain sensing configuration.

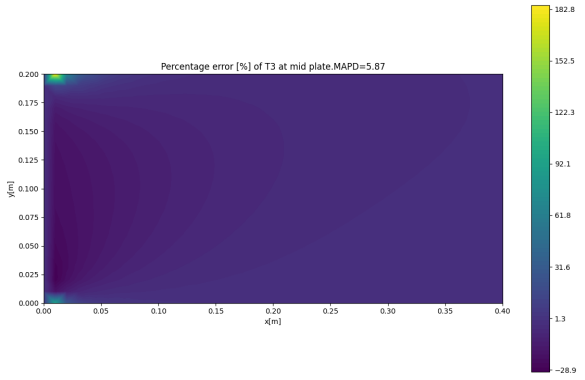


Figure 11.4: Cantilevered plate under distributed load discretized using a 800 element mesh. PD error map for T3 iFEM reconstruction wrt. FEM results for the *EDGE-LINE-EXX* strain sensing configuration.

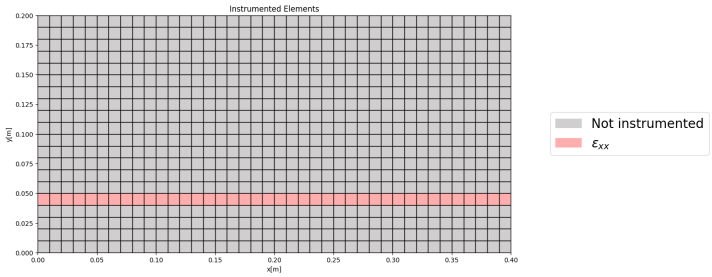


Figure 11.5: Cantilevered plate under distributed load discretized using a 800 element mesh. Visualization of instrumented elements for the *QUARTER-LINE-EXX* strain sensing configuration.

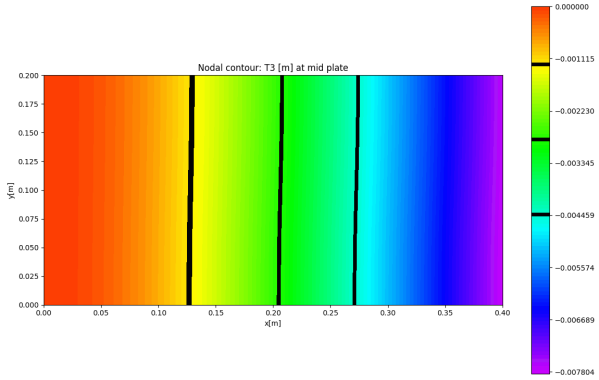


Figure 11.6: Cantilevered plate under distributed load discretized using a 800 element mesh. iFEM reconstruction of T3 for the *QUARTER-LINE-EXX* strain sensing configuration.

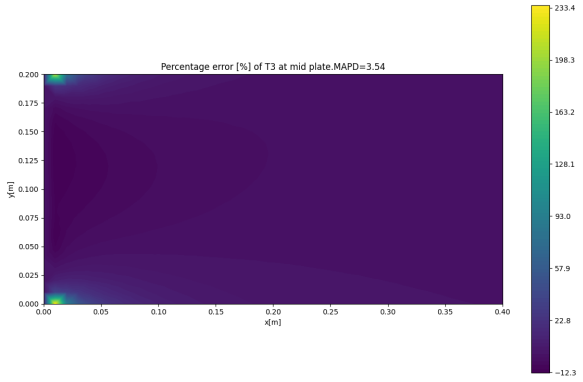


Figure 11.7: Cantilevered plate under distributed load discretized using a 800 element mesh. PD error map for T3 iFEM reconstruction wrt. FEM results for the *QUARTER-LINE-EXX* strain sensing configuration.

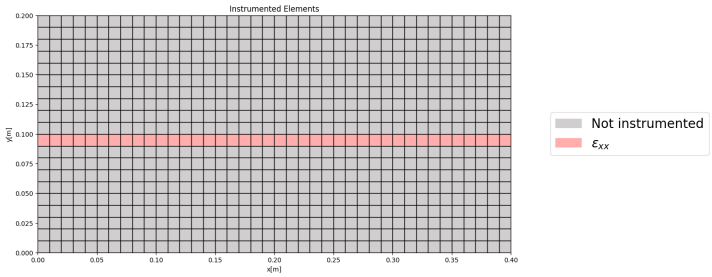


Figure 11.8: Cantilevered plate under distributed load discretized using a 800 element mesh. Visualization of instrumented elements for the *MID-LINE-EXX* strain sensing configuration.

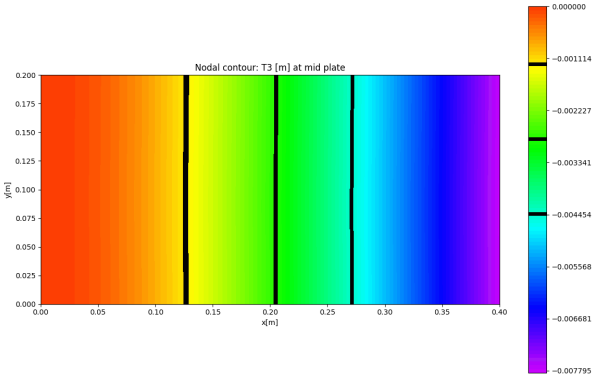


Figure 11.9: Cantilevered plate under distributed load discretized using a 800 element mesh. iFEM reconstruction of T3 for the *MID-LINE-EXX* strain sensing configuration.

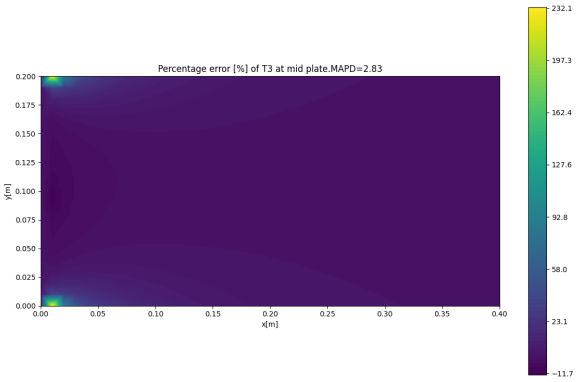


Figure 11.10: Cantilevered plate under distributed load discretized using a 800 element mesh. PD error map for T3 iFEM reconstruction wrt. FEM results for the *MID-LINE-EXX* strain sensing configuration.

11.1.1 Effect of symmetric Boundary Condition (BC)

Figures 11.4,11.7 and 11.10 all show the same behavior: a sudden jump in error to extremely high values at the corners of the cantilevered edge.

It is assumed that this effect is caused by the input strain gradient. By analyzing the FEMAP strain gradient in Figure 11.11, a discontinuity can also be noticed. This effect is caused by the Poisson effect. The width would tend to slightly shrink due to the bending, however, due to the fixed constraint that is not possible leading to the strain pattern. It can be seen that by moving the sensing line inwards, the gradient becomes smoother, leading to both smaller MAPD and smaller twist angle, as discussed earlier.

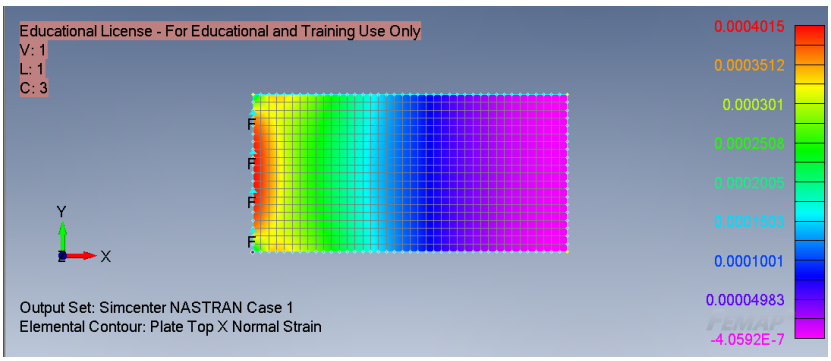


Figure 11.11: Cantilevered plate under distributed load discretized using a 800 element mesh. ε_{xx} strain field on the top plate surface obtained in FEM using fixed BC.

It is possible to remove this effect through the use of symmetric boundary conditions on the long edges of the plates. This modelling method essentially mocks an extended width. The FEMAP strain results using also the symmetric boundary conditions are given in Figure 11.12. It can be seen that the discontinuities and circular pattern is moved, replicating the bending stress behaviour expected for a cantilevered beam.

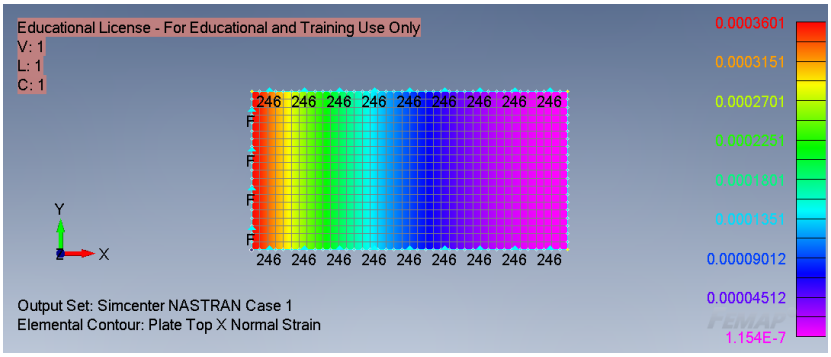


Figure 11.12: Cantilevered plate under distributed load discretized using a 800 element mesh. ε_{xx} strain field on the top plate surface obtained in FEM using fixed and symmetric BC.

This highlights the impact of BC modeling on the iFEM reconstruction. The updated PD error maps are shown in figures 11.13a to 11.13c. It is interesting to see that although now the strain field along the length stays similar for each sensing line location, the reconstruction

error has quite different patterns. It can be clearly seen in these figures that the error on the first line of nodes at the fixed end stays 0 due to the cantilevered constraining, while the second row generally inflicts the highest error. This could be caused by their proximity to the BC. For the *EDGE-LINE-EXX* a 0.582°, *QUARTER-LINE-EXX* 0.468° and *MID-LINE-EXX* 0.055° twist angles are obteined.

A robust reduction in twist cannot be concluded from this method as for the *QUARTER-LINE-EXX* the symmetric BC actually inflict a slight increase. The MAPD however is consistently reduced for all strain configurations. An overview of all the quantitative results is given in Table 11.1.

11.1.2 Effect of Smoothing Element Analysis (SEA)

While the solution for reducing artificial twist in subsection 11.1.1 presents an interesting dependency of iFEM to BC, it does not offer an alternative that could be implemented on a real-life structure that needs to be monitored, but a mere idealization.

By applying the SEA to this case, it was possible to reduce both the MAPD and the artificial twist. This was done for both the cantilevered and catilevered+symmetric BC cases. Thus SEA can provide a more robust method for diminishing the artificial twist effect in the reconstruction.

Table 11.1: Quantification of the effect of sensing line placement for the cantilevered plate under distributed load. The twist angle was computed at the free tip of the plate. Smoothed Inverse Finite Element Methods (iFEM(s)) columns reveals contribution of strain pre-extrapolation.

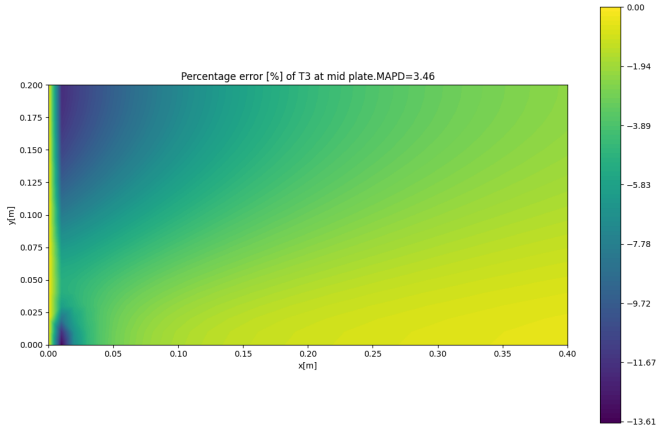
		iFEM		iFEM(s)	
Strain Configuration	BC	MAPD[%]	Twist[deg]	MAPD[%]	Twist[deg]
EDGE-LINE-EXX	Fixed Root	5.86	0.613	4.09	0.16
QUARTER-LINE-EXX	Fixed Root	3.57	0.446	2.9	0.145
MID-LINE-EXX	Fixed Root	2.83	0.063	2.81	0.022
EDGE-LINE-EXX	Fixed Root + Symmetry	3.46	0.582	0.77	0.248
QUARTER-LINE-EXX	Fixed Root + Symmetry	2.33	0.468	0.57	0.124
MID-LINE-EXX	Fixed Root + Symmetry	1.37	0.055	0.51	0.023

11.2 Design Simplified Configuration I - Isotropic Deck

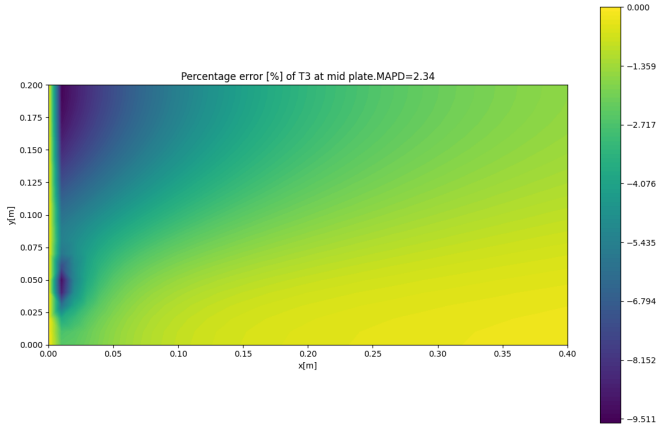
As described in section 10.6, the current gangway design will be simplified. The least complex case was identified to be the isotropic deck.

The deck was modelled as a cantilevered plate of 15.896[m] length. This length was chosen based on the [DNV, 2017] which requires the gangway to be extended to its "maximum operational length".The width of the plate was 0.858 [m], as an average between the Main Boom (M-Boom) and the Telescopic Boom (T-Boom).

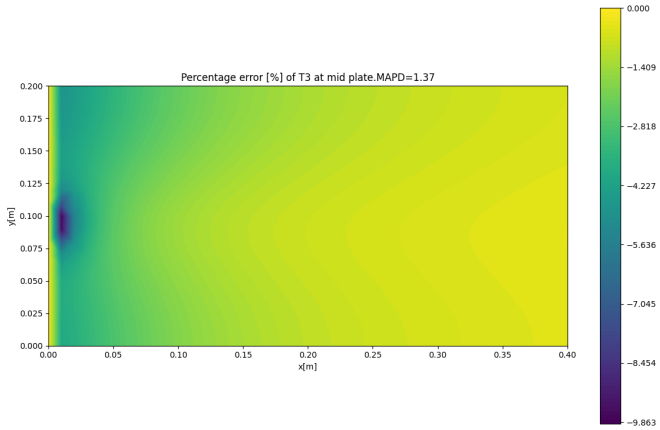
The equivalent material properties of the laminate that are used for this iteration are given in Table 11.2. The required thickness for matching the Dead Load Test Load (DLTL) maximum deflection of the gangway is calculated using the analytical formulas for a cantilevered beam



(a) PD error map for T3 iFEM reconstruction wrt. FEM results for the *EDGE-LINE-EXX* strain sensing configuration.



(b) PD error map for T3 iFEM reconstruction wrt. FEM results for the *QUARTER-LINE-EXX* strain sensing configuration.



(c) PD error map for T3 iFEM reconstruction wrt. FEM results for the *MID-LINE-EXX* strain sensing configuration.

Figure 11.13: Cantilevered plate under distributed load discretized using a 800 element mesh including symmetry BC.

Figure 11.14. As the current analysis is kept in the linear domain, the analytical cases can be overlapped for replicating DLTL loading. A thickness of 0.55 [m] was found and used in the modelling. For the length of 15.896 [m], this leads to a thickness-to-length ratio of well under 1/20 which is generally considered for beam idealisations, confirming the validity of using the formulas in Figure 11.14 for a quick thickness determination.

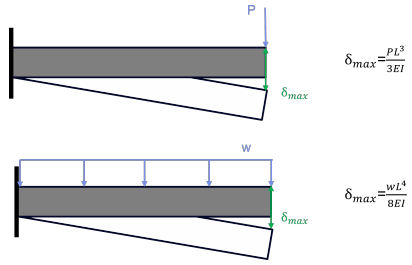


Figure 11.14: Analytical formulas for maximum deflection δ_{max} of a cantilevered beam.

Table 11.3 gives an overview of the results for the Isotropic Deck. Based on the findings provided in section 11.1, it was decided to focus on configurations using the mid-line sensing. As can be seen by the error computed using each strain component, only ε_{xx} is sufficient for correct reconstruction. Using a single uni-axial sensing line (*MID LINE EXX*) can be sufficient for accurate reconstruction. It is interesting to note that when also applying strain pre-extrapolation, the error for the maximum deflection of this configuration can match that of the complete strain configuration.

The effect of using a discontinuous sensing line was also explored through configurations *MID LINE EXX EVERY 6 ELEMENTS*. As reported in [Oboe et al., 2021a], a discontinuity in the sensing pattern can lead to a breakdown of iFEM. This is also reflected in the current results, where the errors for these strain sensing configurations increase in a highly unexpected manner. Applying SEA provides a method for considerably improving the reconstruction of such discontinuous sensing patterns for the isotropic deck.

Table 11.2: Equivalent elastic properties of the composite gangway deck floor laminate.

	Ex [GPa]	**Ey [GPa]**	**Gxy [GPa]**	$\nu_{xy}[-]$	$\nu_{yx}[-]$
Membrane	3.14	3.14	1.2	0.3	0.3
Bending	8.7	8.7	3.33	0.3	0.3

Table 11.3: Overview of T3 reconstruction performance for the Isotropic Deck under DLTL. Discretized with 396 elements.

Strain Configuration	**Sensing Elements x Strain Components**	**T3 MAPD[%]**		**max(T3) PD[%]**	
		iFEM	**iFEM(s)**	**iFEM**	**iFEM(s)**
Complete	396x3	0.59	-	-0.12	
ONLY EXX	396x1	0.88	-	-0.12	-
ONLY EXY	396x1	98.58	-	-100.00	-
ONLY EYY	396x1	98.76	-	-99.98	-
MID LINE EXX	66x1	0.91	0.80	-0.16	-0.12
MID LINE EXX EVERY 6 EL	11x1	96.27	6.06	-96.73	8.03

11.3 Design Simplified Configuration II - Laminated Deck

Advancing from the Isotropic Deck to the Laminated Deck did not create any hassle to the iFEM reconstruction. A thickness of 0.30728 [m] was determined for recreating the DLTL deflection of the composite gangway.The reconstruction behaviour was highly similar to that of the Isotropic Deck, thus only the final reconstruction results are shown in Table 11.4.

Table 11.4: Overview of T3 reconstruction performance for the Laminated Deck under DLTL. Discretized with 396 elements.

Strain Configuration	Sensing Elements x Strain Components	T3 MAPD[%]		max(T3) PD[%]	
		iFEM	iFEM(s)	iFEM	iFEM(s)
Complete	396x3	0.50	-	-0.35	
ONLY EXX	396x1	0.50	-	-0.35	-
ONLY EXY	396x1	98.51	-	-99.99	-
ONLY EYY	396x1	98.51	-	-99.99	-
MID LINE EXX	66x1	0.66	0.5	-0.39	-0.33
MID LINE EXX EVERY 6 EL	11x1	92.73	5.26	-92.74	-4.75

11.4 Design Simplified Configuration III - Isotropic U-Shape

The next step in advancing the geometry representation was including the side-walls. This was done through the use of a simple U-shape. subsection 11.4.1 highlights how this geometry was modeled and compares it against a case from the literature. subsection 11.4.2 covers the final results and findings of the U-shape applied for the gangway structure.

11.4.1 U-Shaped Geometry Literature Study Case

[Abdollahzadeh et al., 2023] investigated the performance of shape reconstruction of 3D beam-like structures using Inverse Quadrilateral Shell 4 Points (IQS4) elements. Multiple geometries including the U-shape were studied. In the paper, the U-beam was represented with a length of 1[m], a width of 0.02[m] and a height of 0.04[m]. The thickness was 5[mm], using a material with E=210 [GPa], $\nu = 0.3$ and $\rho = 3000[kg/m3]$.

The studied case was replicated to first of all confirm the correctness of the FEM modeling of such a structure. Reusing the same 90 elements mesh, the results of [Abdollahzadeh et al., 2023] are shown in Figure 11.15, and the replicated results are shown in Figure 11.16. In FEMAP, the one-piece structure was obtained using the "Nonmanifold Add" command for avoiding repetition of curves and nodes at the intersection of the surfaces. To recreate the flush cross-section, the thickness offset with respect to the nodes was adjusted for each wall.

It can be seen that the overall behaviour of the FEMAP simulation is agreeing with the results of [Abdollahzadeh et al., 2023], and the maximum total displacement results in a PD of 1.85% between the two models.

When it comes to comparing the performance of the iFEM reconstruction between the current implementaiton and that of [Abdollahzadeh et al., 2023], a considerable difference can be

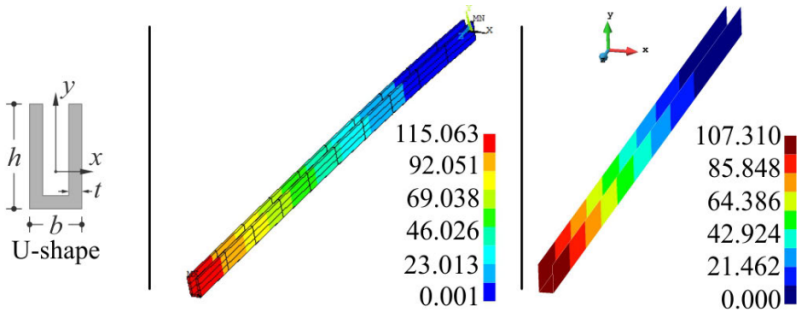


Figure 11.15: Results for FEM (left) and iFEM (rights) analysis for the U-shape study case. Total translation under its own weight in micrometer. Courtesy of [Abdollahzadeh et al., 2023].

noticed. The authors report a reconstruction PD error of 6.8%, while in the current study the error is reduced to about 0.17%. It was tried to reproduce to the best of the abilities the iFEM analysis using the same type of IQS4 elements, the same inverse mesh and the same $w_f = 10^{-4}$. It remains diffcult to further assess where this reconstruction performance error could be coming from, as no information is provided by the authors on the numerical implementation (as discussed in subsection 10.5.1 or subsection 10.5.3).

Nevertheless, this preliminary investigation satisfied the initial goal: gathering confidence in the chosen method of FEM modelling of beam-like structures using quadrilateral elements.

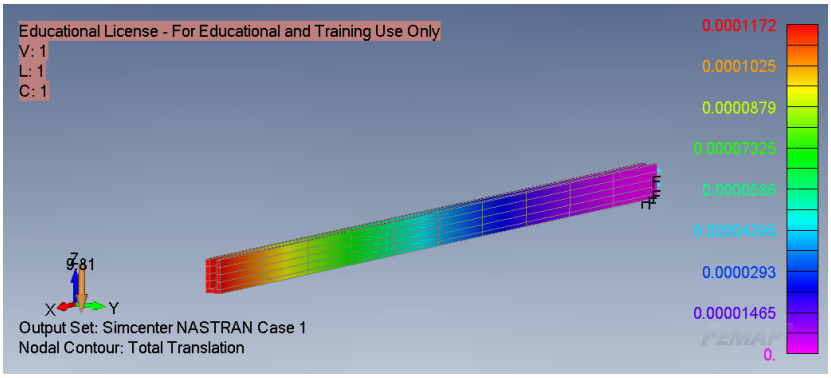


Figure 11.16: Reproduction of U-shape study case in [Abdollahzadeh et al., 2023] in FEMAP using strain rosettes on all elements. Total translation under its own weight in [m].

11.4.2 Actual Implementation

The height of the side walls was set to 1.39 [m], averaging the original measurements of the gangway design booms. The Test Load (TL) load was distributed equally over the two edge points to prevent stress concentrations. A thickness of 15.5 [mm] was computed for matching as much as possible the DLTL deflection of the reference FEMAP gangway model. The analytical beam model predicted a DLTL maximum displacement of 0.101 [m], while the FEMAP model led to a total displacement of 0.104 [m].

A mesh of 528 elements was used for this analysis. This was done in order to reduce the computational effort of running the required hyperparameters optimizations described in sub-

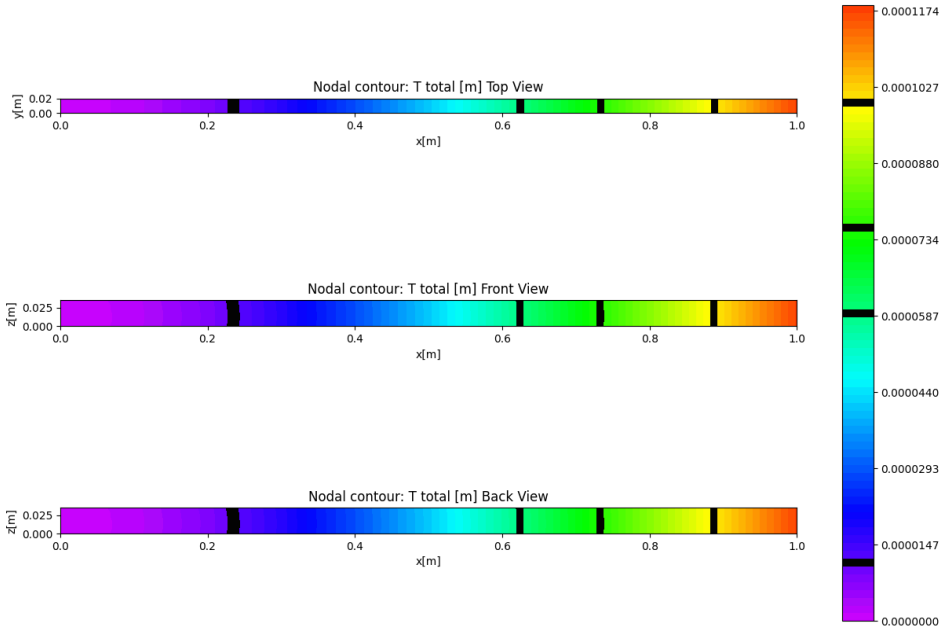


Figure 11.17: iFEM reconstruction of the total displacement for the U-shape study case in [Abdollahzadeh et al., 2023].

section 10.5.5 for the multiple investigated strain configurations. Secondly, the current mesh allows for separating the lines on all faces into *EDGE* and *MID* configurations with each their *SYMM* equivalent. For the side-walls, any possible *MID* line configurations are not taken into account as they would overlap with the cut-outs. Thus, it is important to see if the reconstruction for this simplified geometry can already be done without *MID* measurements on the side walls.

Complete Triaxial Reconstruction

Firstly, a complete reconstruction using tri-axial measurement was done to assess the most optimistic reconstruction scenario using the 528 mesh discretization. This led to a value an MAPD of 2.02% and -2.02% for the $PD(T_3)$.

Complete Uniaxial Reconstruction

As opposed to the previous simplified configurations section 11.2 and section 11.3, doing an iFEM reconstruction using only one strain component is not as trivial.

Due to the presence of the side walls, the elements will have different local coordinate systems based on the surface (bottom or side-walls) they are part of. Figure 11.18 shows the default local coordinate systems set by FEMAP for the U-shape geometry. We can see that if we were to select let's say the strain in X-global it would actually imply taking the local ε_{xx} measurements for the side-walls, but the local ε_{yy} measurements for the deck.

By default, the output of FEMAP (and other FEM software) comes in the local coordinate system. As the $\mathbf{f^e}, \mathbf{f^e_{SEA}}$ matrices are built using local strains taking advantage of this default

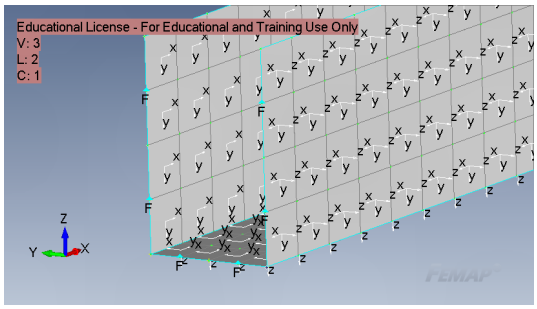


Figure 11.18: Default local coordinate systems for U-shape geometry.

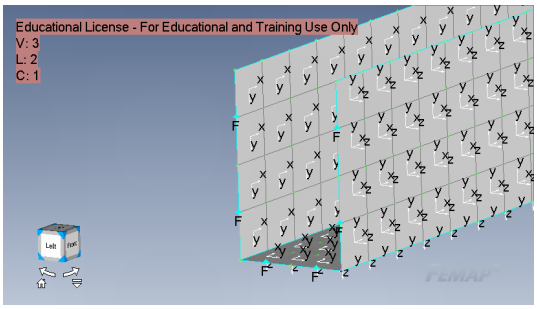


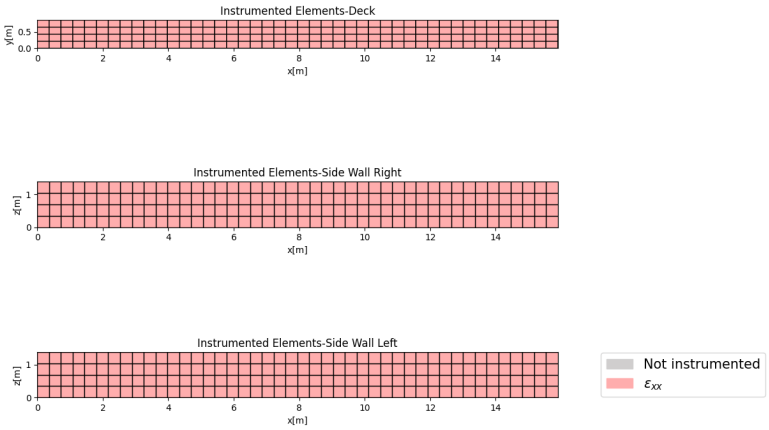
Figure 11.19: Local coordinate systems for U-shape geometry aligned along global x-axis.

option is advantageous. Nevertheless, for real-life measurements, this is not common practice. Typically, strain recording devices are placed using either a global direction or the alignment of a material (for anisotropic). Thus, an option for isolating the strain components consistently is to export the data from FEMAP in global coordinates, keep only the ε_{xx} and to transform it back to local coordinate systems within the iFEM implementation.

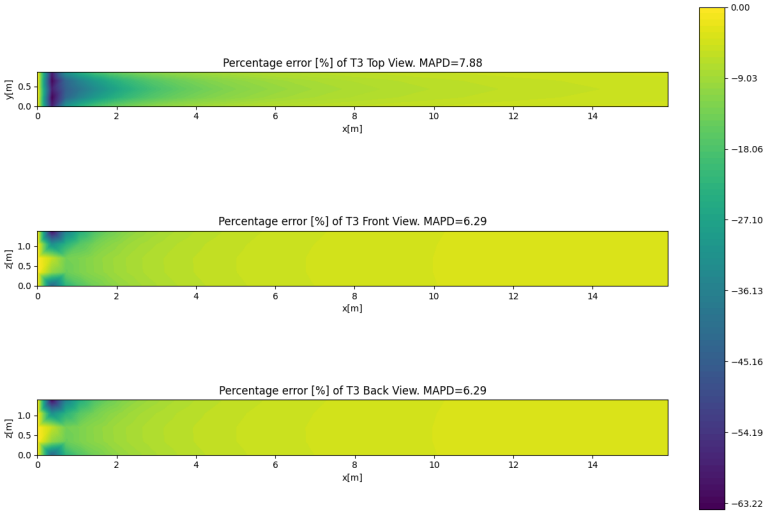
It is expected that the option of only using these strain transformations for the output will lead to issues for SEA analysis. This is due to the fact that the interpolation is done for the local strains. By using the default coordinate systems in Figure 11.18, all the strain components will have discontinuities at the level of the deck to side-wall interface. This is expected to pose a problem due to the assumption of C_1 continuity in SEA.

This issue can be reduced by aligning the local coordinate systems. Due to assumption CODE-1, an alignment of all axes is not possible as the ε_{zz} is ignored. It is possible to align along the global x-axis, as shown in Figure 11.19. This solution allows for both a consistent exploration uniaxial sensing configurations, and a better domain for applying SEA.

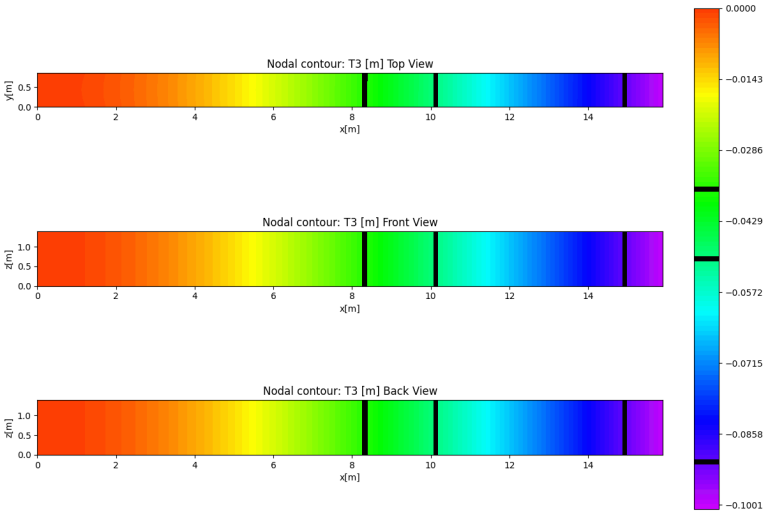
By reconstructing using uniaxial ε_{xx} measurements on all elements, the $MAPD(T_3)$ is increased to 6.85% and the $PD_{max(T_3)}$ to -3.79%. As it is still within the 95% accuracy, uniaxial configurations were still explored further. Figures 11.20a to 11.20c illustrate the results for the *ONLY EXX* configuration.



(a) Instrumented elements for the *ONLY EXX* strain configuration of Design Simplification III.



(b) PD error map for T3 iFEM reconstruction wrt. FEM results for the *ONLY EXX* strain sensing configuration.



(c) Reconstruction of T3[m].

Figure 11.20: Design Simplified Configuration III plate under DLTL discretized using a 528 element mesh.

Multi Surface Line Configurations Reconstruction

Figure 11.21 shows a schematic of how the investigated multi-surface line configurations are derived. This is done in an inverse incremental manner, reducing the number of sensing points in each sensor network configuration.

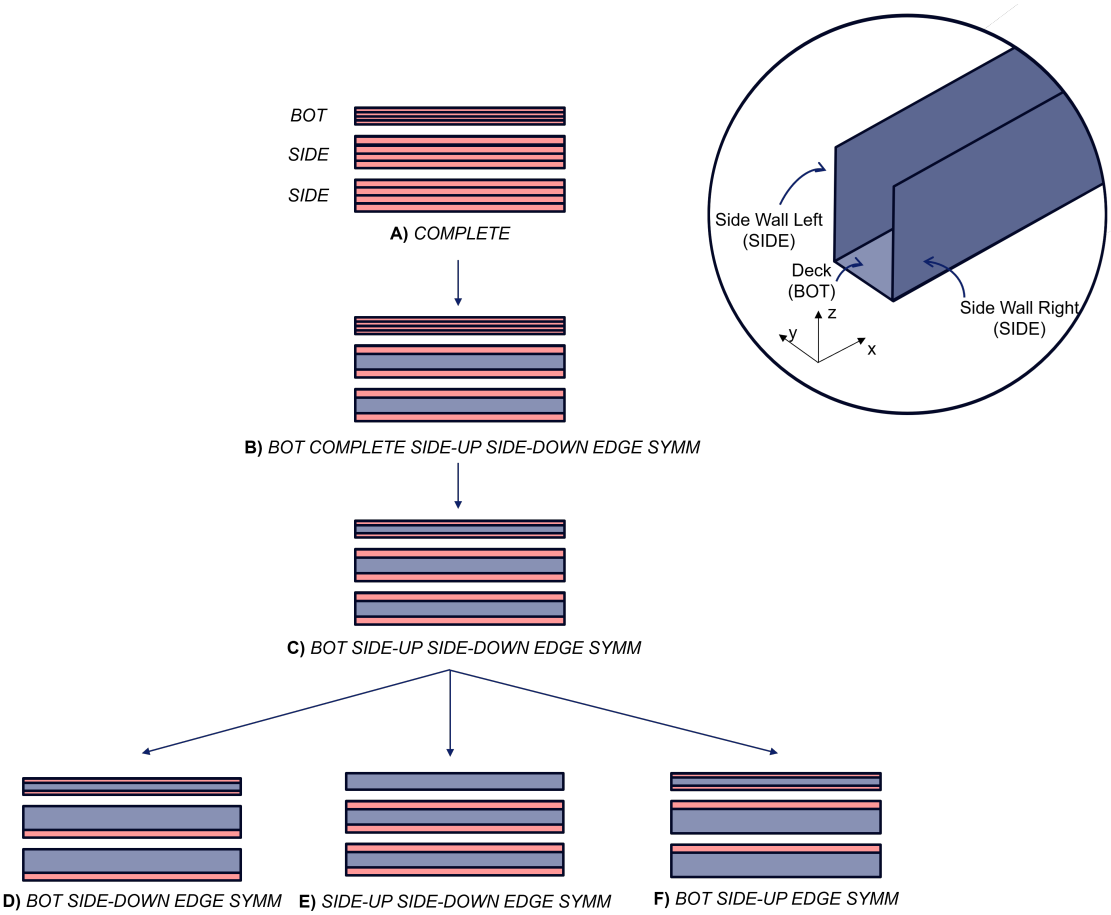


Figure 11.21: Strain configurations overview for the U-shape geometry. Red lines correspond to the strain sensing lines.

As mentioned previously, it is desired to obtain a sufficiently good reconstruction without placing sensors at the level of the side-wall cut-outs. Thus, the first step consists of removing the sensing lines in the middle of the side wall resulting in configuration **B**. Table 11.5 gives an overview of the key results of configurations **A,B,C** for both tri-axial and uni-axial strain measurements. A key observation for all these configurations is that switching from triaxial to uniaxial measurements has a higher impact on the overall T3 reconstruction and a lesser impact on the prediction of the maximum deflection.

As the results of configuration **C** are still within the required accuracy, the sensor network was even further reduced. Configurations **D, E, F** all use the same number of sensing elements: 176. Each configuration explores different combinations of edge-based locations. Table 11.6 shows the results for both uniaxial and axial **D, E, F** configurations.

Table 11.5: Error overview for sensing strain configurations **A, B, C**

Strain Configuration	Strain Elements	iFEM T3 MAPD[%]		iFEM max(T3) PD[%]	
		$\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}$	ε_{xx}	$\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}$	ε_{xx}
A)	528	2.02	6.85	-2.02	-3.79
B)	352	3.80	6.74	-2.38	-3.23
C)	264	5.12	7.25	-3.46	-3.81

Table 11.6 highlights the importance of the location of the sensors in iFEM and how a certain number of sensors cannot guarantee always a correct deflection reconstruction. Configurations **D** and **E** did not satisfy the limit required accuracy with using only the iFEM analysis. Thus, iFEM(s) was also applied to see if the reconstruction can be improved.

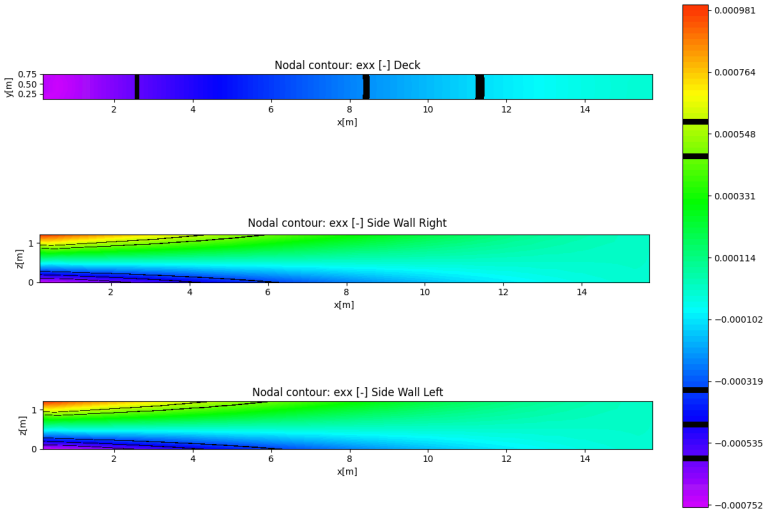
Table 11.6: Error overview for sensing strain configurations **D, E, F**

Strain Configuration	T3 MAPD[%]		max(T3) PD[%]	
	iFEM	iFEM(s)	iFEM	iFEM(s)
D) $\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}$	28.65	28.58	-29.87	-29.90
D) ε_{xx}	39.41	40.06	-36.13	-36.66
E) $\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}$	7.64	4.00	-5.85	-3.27
E) ε_{xx}	9.39	7.47	-6.05	-4.07
F) $\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{xy}$	6.09	3.14	-3.54	-1.84
F) ε_{xx}	7.12	7.02	-3.26	-3.15

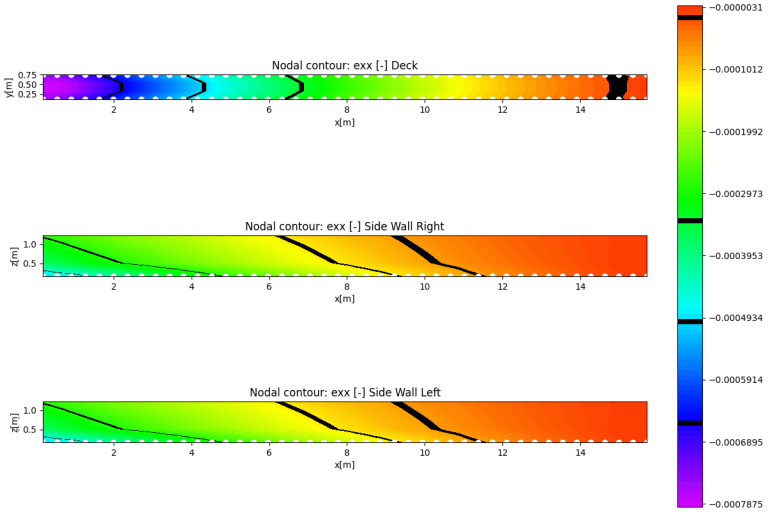
Overall, **D** offers the poorest reconstruction. This can be expected in a qualitative manner when looking at the strain distribution Figure 11.22a. There is a stress transition along the height of the side walls. When applying SEA the difficulty of reconstructing with only this data becomes clear. Figure 11.22b shows the reconstructed strain field through SEA for strain sensing configuration **D**. While the behaviour for the deck is relatively similar despite the presence of the radial pattern towards the root, the behaviour of the side-walls is completely inconsistent. Due to the lack of measurements at the top of the side-walls, the strain is interpolated as almost constant from the base to the top.

Figure 11.22b displays the interpolated strain gradient for strain sensing configuration **F**. It can be seen how the strain behavior of the side-walls can be much better replicated through SEA when critical strain values are recorded. This also illustrates how applying SEA improves the results for **E** and **F**, but leads to an even poorer reconstruction for **D** which simply does not cover the critical strain values. In Table 11.6 it can also be seen that the effect of SEA is lesser in uniaxial **F** than it is in uni-axial **E**. This could be explained by the fact that the error of uni-axial **F** is already approaching closely the complete uni-axial recomstruction in **A** (MAPD=6.85%).

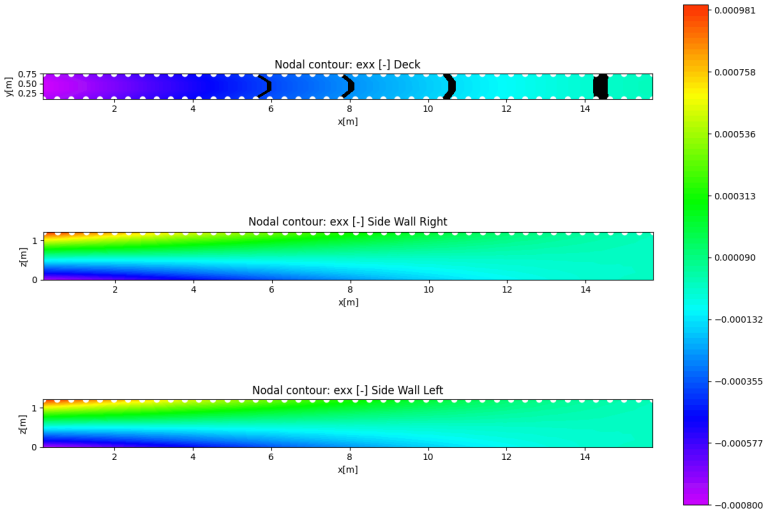
It was tried to further reduce the sensing networks by looking at asymmetric option of strain configuration **F**. Nevertheless, the reconstruction offered highly inaccurate overall results of over 23% MAPD(T_3). Thus, for the current implementation, four sensing lines (on each plate side) are required for correct reconstruction.



(a) Strain configuration Complete triaxial. FEMAP output.



(b) Strain configuration **D** triaxial. White dots are the FEMAP output, the rest of the points are determined through SEA.



(c) Strain configuration **F** triaxial. White dots are the FEMAP output, the rest of the points are determined through SEA.

Figure 11.22: Strain gradients for Design Simplified Configuration III.

11.4.3 Influence of Local Coordinate System Definition on iFEM(s) analysis

It was previously discussed how using local coordinates that are not aligned might impact negatively the contribution of SEA. The iFEM and iFEM(s) results were computed again for the default coordinate system created by FEMAP which does not have any aligned axis. Table 11.7gives a comparison overview.

Table 11.7: Errors comparison between default local coordinate systems and aligned local coordinate systems.

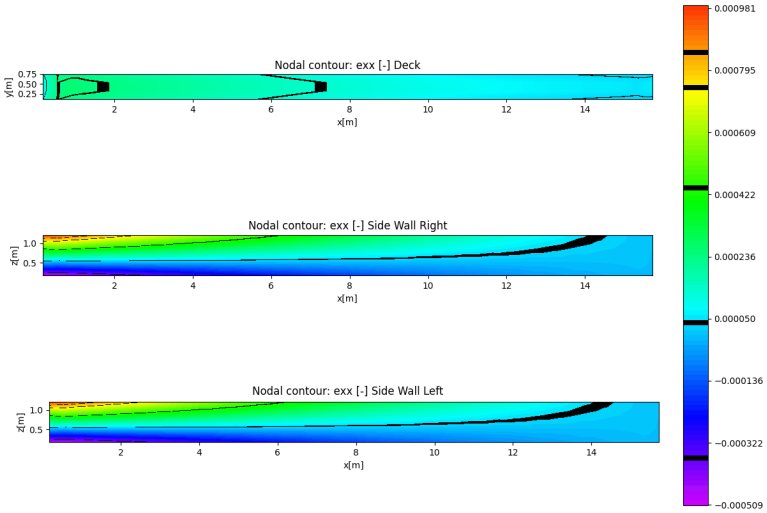
Strain Configuration	Coord. System	T3 MAPD[%]		max(T3) PD[%]	
		iFEM	iFEM(s)	iFEM	iFEM(s)
D	Aligned	28.65	28.58	-29.87	-29.90
	Default	28.65	32.13	-29.65	-28.82
E	Aligned	7.64	4.00	-5.85	-3.27
	Default	7.59	6.39	-5.83	-5.48
F	Aligned	6.09	3.14	-3.54	-1.84
	Default	5.89	7.81	-3.52	-3.08

It can be seen that the selected local coordinate systems do not have a big effect on the results of the iFEM analysis. Also, it is interesting to note that the iteration for w_f leads, for both the default and not-aligned coordinate systems, to the same MAPD(T_3)-minimizing w_f, further highlighting that the effect of local coordinate systems on iFEM results is small.

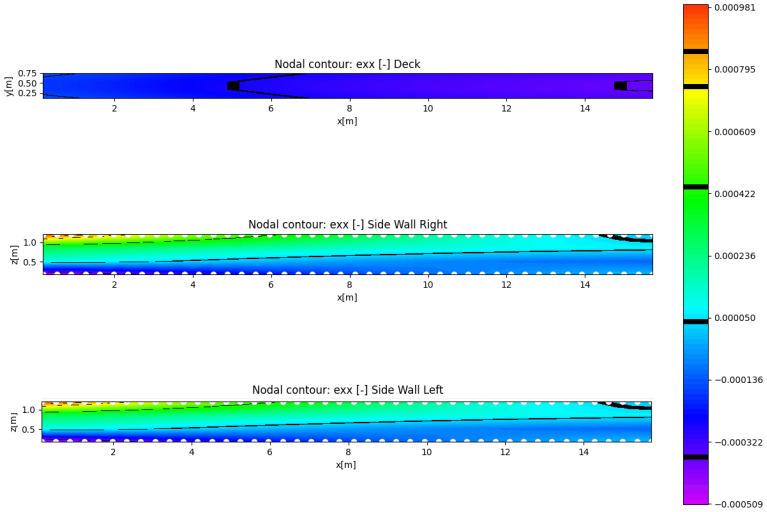
Nevertheless, the results start diverging for the iFEM(s) analysis. For **D** and **F**, the MAPD is actually increased by a few percentages. Figure 11.23c shows the cause of this. SEA is imposing the continuity condition at the interface of the side-walls and deck, which is not viable in the case of these local coordinate systems. In this way, both the strain patterns in the side-walls and the deck plate get destroyed. Moreover, only the highest strain value gets captured, but the minimum does not, also having an effect on the range of extrapolated values.

It can be seen that only for configuration **E** SEA leads to error reduction. Looking at Figure 11.23b, the success of the strain pre-extrapolation can be explained by the mere coincidence that the sensing points are grouped on the parallel surface (side-walls), rather than perpendicular ones which require the different local coordinate system. Thus, the strain gradient in the side-walls, which is continuous in the first place, is extrapolated correctly and only the strain pattern in the bottom deck gets affected. As the surface area of the deck is smaller, its contribution to MAPD is also smaller leading to an improved error despite its erroneous behavior reproduction.

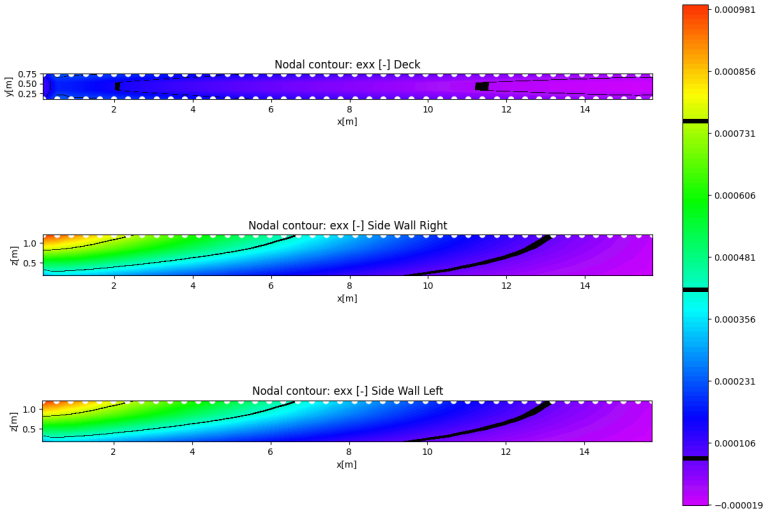
Thus, it can be seen that for not-aligned local coordinate systems, SEA is not a robust method for improving the performance of iFEM due to the discontinuities.



(a) Strain configuration Complete triaxial. FEMAP output.



(b) Strain configuration **E** triaxial. White dots are the FEMAP output, the rest of the points are determined through SEA.



(c) Strain configuration **F** triaxial. White dots are the FEMAP output, the rest of the points are determined through SEA.

Figure 11.23: Strain gradients for Design Simplified Configuration III. Default FEMAP local coordinate systems.

11.4.4 Effect of Hyperparameters on Deflection w_f, α Reconstruction

An interesting observation during the hyperparameter optimization was determining that the U-shape is more sensitive to changes in w_f than a simple rectangular beam.

Figures 11.24 and 11.25 illustrate the difference in behavior. Typically, the effect of w_f increases with the number of strainless elements. It can be seen that for a rectangular beam with an extremely reduced sensing network (2.8% sensing elements), the error barely varies 3%. However, for a U-shape geometry, even with a dense sensing network (66.66% sensing elements), the MAPD varies dramatically by up to 35%.

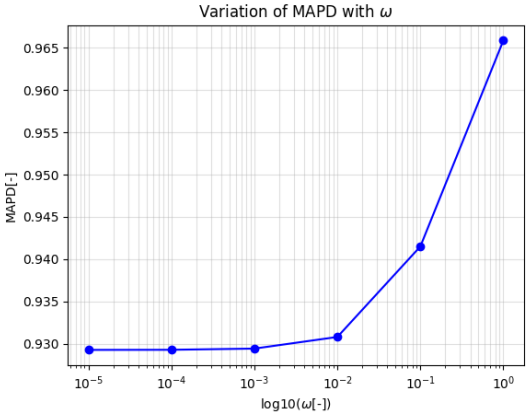


Figure 11.24: Variation of $MAPD(T_3)$ with w_f for Design Simplification I. Iteration run for *MID-LINE-EXX EVERY 6 ELEMENTS* using iFEM in which only 2.8% of the elements contain strain measurements.

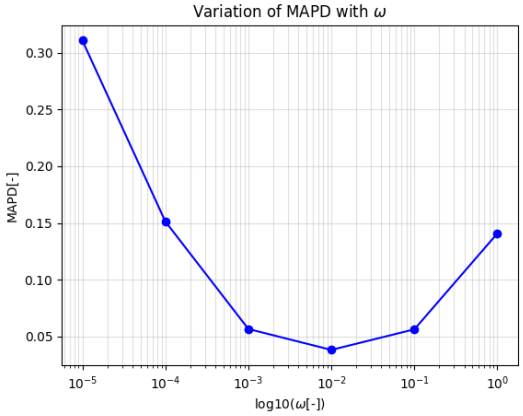


Figure 11.25: Variation of $MAPD(T_3)$ with w_f for Design Simplification III. Iteration run for strain sensing configuration **B** using iFEM in which only 66.66% of the elements contain strain measurements.

It was found that the impact of α on the reconstruction accuracy is lesser than that of w. Table 11.8 highlights this difference for the strain configurations. The standard deviation SD is computed for variation of MAPD(T_3) with respect to both w_f and α. A higher SD highlights a bigger impact of the parameter, highlighting increased sensitivity to it.

Table 11.8: Comparison of effect of w_f and α on the MAPD(T_3) for different strain configurations of Simplified Design Configuration III. Effect expressed in the standard deviation SD of MAPD(T_3) of different w_f α

Strain Configuration	SD(w_f)[%]	SD(α)[%]
C) $\varepsilon_{\mathbf{xx}}, \varepsilon_{\mathbf{yy}}, \varepsilon_{\mathbf{xy}}$	12.29	0.15
C) $\varepsilon_{\mathbf{xx}}$	9.37	0.08
D) $\varepsilon_{\mathbf{xx}}, \varepsilon_{\mathbf{yy}}, \varepsilon_{\mathbf{xy}}$	15.50	0.14
D) $\varepsilon_{\mathbf{xx}}$	10.44	1.53
E) $\varepsilon_{\mathbf{xx}}, \varepsilon_{\mathbf{yy}}, \varepsilon_{\mathbf{xy}}$	21.99	0.43
E) $\varepsilon_{\mathbf{xx}}$	14.75	0.36
F) $\varepsilon_{\mathbf{xx}}, \varepsilon_{\mathbf{yy}}, \varepsilon_{\mathbf{xy}}$	19.73	0.25
F) $\varepsilon_{\mathbf{xx}}$	12.16	0.02

This observation is highly interesting as generally, the literature highlights more often the importance of α, rather than that of w which is treated as a rather trivial parameter. One of the roles that w_f satisfies is creating continuity between the strain and strainless elements as noted by [Abdollahzadeh et al., 2020]. Thus, the increased sensitivity of the U-shape to w_f could be explained by the discontinuities present in the strain field patterns.

Chapter 12

Conclusion

The current study investigated the design of a Structural Health Monitoring (SHM) system for a still in-development, composite offshore access system gangway. The preliminary literature study focused predominantly on available in-situ monitoring techniques for composite structures. The obtained knowledge was framed from the perspective of gangway structures, concluding that a deflection monitoring system should be pursued based on the current legislation and knowledge of using composites in this application.

RQ1 How can Inverse Finite Element Methods (iFEM) be implemented for SHM of a composite gangway?

In the current analysis, an iFEM implementation was developed in-house in Python. Plate elements have been selected for the application due to the tailor-abilities they offer for simulating different sensing networks. Specifically, the Inverse Quadrilateral Shell 4 Points (IQS4) elements were implemented for the application in an effort to match as much as possible the reference FEMAP model which employs CQUAD4 NASTRAN elements.

In the current study, the most complex investigated structure was a constant cross-section, constant thickness isotropic U-shape beam. The simulated load respected the Det Norske Veritas (DNV) legislation. Thus, an Dead Load Test Load (DLTL) was used for the analysis with a Test Load (TL) of 300 kg and a dead load correspondent to the total mass of the gangway in the latest design iteration.

RQ2 What is a suitable sensing network architecture for the gangway?

Sensing networks were exploited hierarchically, starting from denser networks with both tri-axial and uni-axial measurements. Due to the bending load, sensing on both sides of the plate is required for separating the membrane and bending strain components.

In the investigated DLTL, uni-axial strain measurements of ε_{xx} were found to be sufficient for satisfactory deflection reconstruction which was established by Ampelmann at 95% accuracy for the maximum deflection.

Line configurations were investigated due to both the geometry at hand and the Fiber Optics (FO) mounting feasibility. It was shown that the extreme strain points need to be captured for correct reconstruction. This favourizes the placement of the sensing lines at the top of the side walls and in the deck plate.

RQ3 What is the performance of the proposed SHM system?

A 4-sensing line (with sensors on each side of the plate structure) configuration was deemed to be a possible alternative, allowing for the placement of FO outside the cut-outs region of the side-walls. For tri-axial strains, the reported iFEM mean absolute difference of the deflection was 6.09% while the percentage difference of the maximum deflection was -3.54% which were reduced through Smoothing Element Analysis (SEA) to 3.14% and -1.84%. For uni-axial measurements, the iFEM results were 7.12% and -3.26%. The SEA did not prove itself as effective as the errors only were reduced to 7.02% and -3.15%. This reduced effectiveness could be explained as the errors were already approaching those of a complete reconstruction with only uni-axial measurements.

Contribution

In conclusion, despite its current limitations, the study established the potential of iFEM based SHM system for offshore access gangways. It laid down a detailed guideline on both the theory and implementation of iFEM with IQS4 elements. The study also offers a first open-source iFEM implementation. It is one of the few studies quantifying iFEM deflection reconstruction performance for uni-axial strain measurements for plates under bending loads. Moreover, it is the first study on iFEM reconstruction using strainless elements on beam-like geometries with webs. The current work also highlights the pitfalls of SEA under such a geometry and proposes an easy modelling approach for improving the robustness of strain pre-extrapolation in such a case.

Chapter 13

Recommendations

Although the current study helped trace some initial ideas regarding Inverse Finite Element Methods (iFEM)-based Structural Health Monitoring (SHM) for gangways, the current investigations were not only subject to a series of limitations but also opened new questions that need to be tackled. The discussion on recommendations is divided into the following categories: iFEM model, strain pre-extrapolation, structure representation, hyperparameters tuning, gangway SHM and experimental validation.

iFEM model

The current implementation of the iFEM should be further developed. Firstly, only the Inverse Quadrilateral Shell 4 Points (IQS4) element types are currently set up. Other type of elements should be implemented. More information on how new element types can be added are provided in Appendix C. Refined Zig-Zag Theory (RZT) formulations should also be implemented, especially for the current gangway application. These would allow for overall better shape reconstruction of thick laminated and sandwich structures.

Additionally, it would be desired to also allow for meshes which use more than one type of element. This topic needs to be generally explored in iFEM as in the read literature there couldn't be found any reports on using multi-element meshes.

Another point of attention is implementing the use of an iFEM mesh that does not coincide with the Finite Element Methods (FEM) mesh. This would allow in more flexibility for sensitivity studies regarding the number of inverse elements. Moreover, it would become an asset for using iFEM with experimental strain data.

The iFEM code should also allow for more geometrical flexibility. Currently, the code only allows for constant thickness throughout the structure. This problem could be solved by also exporting the thickness of the element (where applicable) and storing it as an element attribute.

Strain Pre-extrapolation

It was shown in this study that the positive effect of Smoothing Element Analysis (SEA) can be rendered null for a U-shape geometry. An interesting topic to explore would be how can SEA be accommodated for such cases. An idea that was explored preliminarily during the study was separating the structure in sub-structures and conducting SEA individually, however, it was not possible to develop it to a tangible conclusion during the timeframe. Another research direction could be on whether different strain pre-extrapolation techniques other than SEA are better suited for such geometries.

Structure Representation

As mentioned throughout the report, quite a few simplifications were done on the iFEM gangway model. Now that this preliminary feasibility study has concluded favorably, using the recommendations regarding the implementation, more complexity should be added to the model. This would allow for a better assessment on the reliability of implementing such a system.

It was noticed that adding geometry complexity was more difficult in iFEM than adding material complexity. Thus, as a first step, probably simpler, sandwich composites should be modeled to the current U-shape. Secondly, the variable thickness should be enabled, allowing for the thicker side walls and thinner bottom deck. Furthermore, the cut-outs should be included. This step already will represent a challenge on its own in the field of iFEM. The work on iFEM reconstruction of tensile specimens with holes done in [Oboe et al., 2022] can represent a starting point for such an endeavor. However, scaling from simple holes to cut-outs resembling truss structures should not be treated as a trivial task.

Hyperparameters Tuning

In the current study, a 2 step optimization through iteration and error minimization was conducted for w_f and α. Currently, the relevant literature on iFEM mostly deals with uni-variate sensitivity analysis for the hyperparameters and rarely with bi-variate analysis [Minigher et al., 2022]. Thus, the relation and interaction of the 4 parameters w_f, α, β and k_{ψ_z} is not exactly known for SEA using neither quadrilateral nor triangular elements. Thus, a general recommendation for the further development of smoothed-iFEM is exploring a robust method for multi-variate optimization involving all the hyperparameters.

Gangway SHM

The current system proposal only covers level 1 SHM in a specific static loading condition. While this offers a tangible beginning for introducing SHM in gangway access systems, it leaves plenty of unexplored possibilities. Firstly, the acquired data could be post-processed in other ways. For example, by collecting the deflection data under the same loading over time,

a potential degradation pattern could be identified and possibly associated with a certain property or damage mechanism.

Secondly, a more advanced iFEM based SHM system could be evaluated. For example, the inclusion of dynamic cases would be of high value.

Experimental Validation

The current study focused solely on the feasibility of an iFEM implementation using numerically generated data. To confirm the potential of the current outcomes, experimental validation is required. At the current stage, a series of tests on U-shape beams under different bending loads and of different materials (both isotropic and anisotropic) are recommended for confirming the robustness of iFEM and Smoothed Inverse Finite Element Methods (iFEM(s)) of such a geometry. At more advanced stages, the proposed SHM could first be installed on one of the steel gangways to confirm its functionality in real-life operation conditions.

References

[Abdollahzadeh et al., 2022] Abdollahzadeh, M. A., Ali, H. Q., Yildiz, M., and Kefal, A. (2022). Experimental and numerical investigation on large deformation reconstruction of thin laminated composite structures using inverse finite element method. *Thin-Walled Structures*, 178:109485.

[Abdollahzadeh et al., 2023] Abdollahzadeh, M. A., Belur, M. Y., Basoglu, M. F., and Kefal, A. (2023). Shape Sensing of Beam-Like Structures Using the Robust iFEM-iQS4 Inverse Shell Element. *IEEE Transactions on Instrumentation and Measurement*, 72:7506209.

[Abdollahzadeh et al., 2020] Abdollahzadeh, M. A., Kefal, A., and Yildiz, M. (2020). A Comparative and Review Study on Shape and Stress Sensing of Flat/Curved Shell Geometries Using C0-Continuous Family of iFEM Elements. *Sensors*, 20(14):3808.

[Agrawal et al., 2014] Agrawal, S., Singh, K., and Sarkar, P. (2014). Impact damage on fibre-reinforced polymer matrix composite – A review.

[Aldajah et al., 2009] Aldajah, S., Al-omari, A., and Biddah, A. (2009). Accelerated weathering effects on the mechanical and surface properties of CFRP composites. *Materials & Design*, 30(3):833–837.

[Bang, 2012] Bang, H. (2012). Shape estimation and health monitoring of wind turbine tower using a FBG sensor array.

[Birman and Genin, 2018] Birman, V. and Genin, G. M. (2018). 1.15 Linear and Nonlinear Elastic Behavior of Multidirectional Laminates. In Beaumont, P. W. R. and Zweben, C. H., editors, *Comprehensive Composite Materials II*, pages 376–398. Elsevier, Oxford.

[Bogert et al., 2003] Bogert, P., Haugse, E., and Gehrki, R. (2003). Structural Shape Identification from Experimental Strains Using a Modal Transformation Technique. *44th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, page 1626.

[Buchner, 2002] Buchner, B. (2002). *Green water on ship-type offshore structures*. PhD thesis, [Doctoral thesis, Delft University of Technology].

[Castro, 2023] Castro, S. G. P. (2023). General-purpose finite element solver based on Python and Cython.

[Colombo et al., 2021] Colombo, L., Oboe, D., Sbarufatti, C., Cadini, F., Russo, S., and Giglio, M. (2021). Shape sensing and damage identification with iFEM on a composite structure subjected to impact damage and non-trivial boundary conditions. *Mechanical Systems and Signal Processing*, 148:107163.

[Colombo et al., 2019] Colombo, L., Sbarufatti, C., and Giglio, M. (2019). Definition of a load adaptive baseline by inverse finite element method for structural damage identification. *Mechanical Systems and Signal Processing*, 120:584–607.

[Cook, 1994] Cook, R. D. (1994). Four-node 'flat' shell element: Drilling degrees of freedom, membrane-bending coupling, warped geometry, and behavior. *Computers & Structures*, 50(4):549–555.

[DNV, 2017] DNV (2017). DNVGL-ST-0358 Certification of offshore gangways for personnel transfer.

[Esposito and Gherlone, 2020] Esposito, M. and Gherlone, M. (2020). Composite wing box deformed-shape reconstruction based on measured strains: Optimization and comparison of existing approaches. *Aerospace Science and Technology*, 99:105758.

[Farrar and Worden, 2010] Farrar, C. R. and Worden, K. (2010). An Introduction to Structural Health Monitoring. In *New Trends in Vibration Based Structural Health Monitoring*, pages 1–17. Springer, Vienna.

[Farrar and Worden, 2012] Farrar, C. R. and Worden, K. (2012). *Structural Health Monitoring: A Machine Learning Perspective*. John Wiley & Sons, Incorporated.

[Floris et al., 2021] Floris, I., Adam, J. M., Calderón, P. A., and Sales, S. (2021). Fiber Optic Shape Sensors: A comprehensive review. *Optics and Lasers in Engineering*, 139:106508.

[Freydin et al., 2019] Freydin, M., Rattner, M. K., Raveh, D. E., Kressel, I., Davidi, R., and Tur, M. (2019). Fiber-Optics-Based Aeroelastic Shape Sensing. *AIAA Journal*, 57(12):5094–5103.

[Fu, 2018] Fu, F. (2018). Chapter Two - Fundamentals of Tall Building Design. In Fu, F., editor, *Design and Analysis of Tall and Complex Structures*, pages 5–80. Butterworth-Heinemann.

[Gherlone et al., 2011] Gherlone, M., Cerracchio, P., Mattone, M., Di Sciuva, M., and Tessler, A. (2011). Dynamic Shape Reconstruction of Three-Dimensional Frame Structures Using the Inverse Finite Element Method. Technical report.

[Gherlone et al., 2014] Gherlone, M., Cerracchio, P., Mattone, M., Sciuva, M. D., and Tessler, A. (2014). An inverse finite element method for beam shape sensing: theoretical framework and experimental validation. *Smart Materials and Structures*, 23(4):045027.

[Greenaway et al., 1999] Greenaway, A. H., Burnett, J. G., Harvey, A. R., Blanchard, P. M., Lloyd, P. A., Mcbride, R., and Russell, P. S. J. (1999). Optical fibre bend sensor.

[Guo et al., 2011] Guo, H., Xiao, G., Mrad, N., and Yao, J. (2011). Fiber Optic Sensors for Structural Health Monitoring of Air Platforms. *Sensors*, 11(4):3687–3705.

[Güemes et al., 2020] Güemes, A., Fernandez-Lopez, A., Pozo, A. R., and Sierra-Pérez, J. (2020). Structural Health Monitoring for Advanced Composite Structures: A Review. *Journal of Composites Science*, 4(1):13.

[Hafizi et al., 2015] Hafizi, Z. M., Epaarachchi, J., and Lau, K. T. (2015). Impact location determination on thin laminated composite plates using an NIR-FBG sensor system. *Measurement*, 61:51–57.

[Hesseler et al., 2021] Hesseler, S., Stapleton, S. E., Appel, L., Schöfer, S., and Manin, B. (2021). 11 - Modeling of reinforcement fibers and textiles. In Akankwasa, N. T. and Veit, D., editors, *Advances in Modeling and Simulation in Textile Engineering*, The Textile Institute Book Series, pages 267–299. Woodhead Publishing.

[Hu and Yung, 2020] Hu, B. and Yung, C. (2020). Offshore Wind Access Report. Technical report, TNO.

[Inaudi and Glisic, 2005] Inaudi, D. and Glisic, B. (2005). Application of distributed Fiber Optic Sensory for SHM. *Proceedings of the ISHMII-2*, 1:163–169.

[Jutte et al., 2011] Jutte, C. V., Ko, W. L., Stephens, C. A., Bakalyar, J. A., and Richards, W. L. (2011). Deformed Shape Calculation of a Full-Scale Wing Using Fiber Optic Strain Data from a Ground Loads Test. Technical report, NASA.

[Kassapoglou, 2013] Kassapoglou, C. (2013). *Design and Analysis of Composite Structures: With Applications to Aerospace Structures*. John Wiley & Sons.

[Kefal, 2019] Kefal, A. (2019). An efficient curved inverse-shell element for shape sensing and structural health monitoring of cylindrical marine structures. *Ocean Engineering*, 188:106262.

[Kefal et al., 2018] Kefal, A., Mayang, J. B., Oterkus, E., and Yildiz, M. (2018). Three dimensional shape and stress monitoring of bulk carriers based on iFEM methodology. *Ocean Engineering*, 147:256–267.

[Kefal et al., 2016] Kefal, A., Oterkus, E., Tessler, A., and Spangler, J. L. (2016). A quadrilateral inverse-shell element with drilling degrees of freedom for shape sensing and structural health monitoring. *Engineering Science and Technology, an International Journal*, 19(3):1299–1313.

[Kefal et al., 2021a] Kefal, A., Tabrizi, I. E., Tansan, M., Kisa, E., and Yildiz, M. (2021a). An experimental implementation of inverse finite element method for real-time shape and strain sensing of composite and sandwich structures. *Composite Structures*, 258:113431.

[Kefal et al., 2021b] Kefal, A., Tabrizi, I. E., Yildiz, M., and Tessler, A. (2021b). A smoothed iFEM approach for efficient shape-sensing applications: Numerical and experimental validation on composite structures. *Mechanical Systems and Signal Processing*, 152:107486.

[Kefal and Tessler, 2021] Kefal, A. and Tessler, A. (2021). Delamination Damage Identification in Composite Shell Structures based on Inverse Finite Element Method and Refined Zigzag Theory. *Developments in the Analysis and Design of Marine Structures*, 7:354–363.

[Ko et al., 2007] Ko, W. L., Richards, W. L., and Tran, V. T. (2007). Displacement Theories for In-Flight Deformed Shape Predictions of Aerospace Structures. Technical report, NASA.

[Kudela et al., 2008] Kudela, P., Ostachowicz, W., and Żak, A. (2008). Damage detection in composite plates with embedded PZT transducers. *Mechanical Systems and Signal Processing*, 22(6):1327–1335.

[Larkin and Shafer, 2011] Larkin, D. Q. and Shafer, D. C. (2011). Robotic surgery system including position sensors using fiber bragg gratings.

[LR, 2021a] LR (2021a). *Code for Offshore Personnel Transfer Systems*.

[LR, 2021b] LR (2021b). *Cranes and Submersible Lifting Appliances*.

[M. Adam, 2013] M. Adam, F. (2013). Degenerated Four Nodes Shell Element with Drilling Degree of Freedom. *IOSR Journal of Engineering*, 03(08):10–20.

[Mao, 2008] Mao, Z. (2008). *Comparison of shape reconstruction strategies in a complex flexible structure*. PhD thesis, [Doctoral thesis, UC San Diego].

[Min et al., 2021] Min, R., Liu, Z., Pereira, L., Yang, C., Sui, Q., and Marques, C. (2021). Optical fiber sensing for marine environment and marine structural health monitoring: A review. *Optics & Laser Technology*, 140:107082.

[Minigher et al., 2022] Minigher, P., Gundlach, J., Castro, S. G. P., and Govers, Y. (2022). Shape Sensing with Sparse Strain Information for Aerospace Applications.

[Mooij et al., 2019] Mooij, C. d., Martinez, M., and Benedictus, R. (2019). iFEM benchmark problems for solid elements. *Smart Materials and Structures*, 28(6):065003.

[Nelson and MacIver, 2006] Nelson, M. E. and MacIver, M. A. (2006). Sensory acquisition in active sensing systems. *Journal of Comparative Physiology A*, 192(6):573–586.

[Oboe et al., 2021a] Oboe, D., Colombo, L., Sbarufatti, C., and Giglio, M. (2021a). Comparison of strain pre-extrapolation techniques for shape and strain sensing by iFEM of a composite plate subjected to compression buckling. *Composite Structures*, 262:113587.

[Oboe et al., 2021b] Oboe, D., Colombo, L., Sbarufatti, C., and Giglio, M. (2021b). Shape Sensing of a Complex Aeronautical Structure with Inverse Finite Element Method. *Sensors*, 21(4):1388.

[Oboe et al., 2022] Oboe, D., Sbarufatti, C., and Giglio, M. (2022). Physics-based strain pre-extrapolation technique for inverse Finite Element Method. *Mechanical Systems and Signal Processing*, 177:109167.

[Pak, 2016] Pak, C.-G. (2016). Wing Shape Sensing from Measured Strain. *AIAA Journal*, 54(3):1068–1077.

[Park et al., 2014] Park, Y.-L., Black, R. J., Moslehi, B., Cutkosky, M. R., Elayaperumal, S., Daniel, B., Yeung, A., and Sotoudeh, V. (2014). Steerable shape sensing biopsy needle and catheter.

[Randhawa and Patel, 2021] Randhawa, K. S. and Patel, A. (2021). The effect of environmental humidity/water absorption on tribo-mechanical performance of polymers and polymer composites – a review. *Industrial Lubrication and Tribology*, 73(9):1146–1158.

[Rapp et al., 2009] Rapp, S., Kang, L.-H., Han, J.-H., Mueller, U. C., and Baier, H. (2009). Displacement field estimation for a two-dimensional structure using fiber Bragg grating sensors. *Smart Materials and Structures*, 18(2):025006.

[Ren et al., 2021] Ren, Z., Verma, A. S., Li, Y., Teuwen, J. J., and Jiang, Z. (2021). Offshore wind turbine operations and maintenance: A state-of-the-art review. *Renewable and Sustainable Energy Reviews*, 144:110886.

[Rocha et al., 2021] Rocha, H., Lafont, U., and Nunes, J. P. (2021). Optimisation of Through-Thickness Embedding Location of Fibre Bragg Grating Sensor in CFRP for Impact Damage Detection. *Polymers*, 13(18):3078.

[Roy et al., 2020] Roy, R., Gherlone, M., and Surace, C. (2020). Damage Localisation in Thin Plates Using the Inverse Finite Element Method. In *Proceedings of the 13th International Conference on Damage Assessment of Structures*, Lecture Notes in Mechanical Engineering, pages 199–212, Singapore. Springer.

[Saeedifar and Zarouchas, 2020] Saeedifar, M. and Zarouchas, D. (2020). Damage characterization of laminated composites using acoustic emission: A review. *Composites Part B: Engineering*, 195:108039.

[Shah et al., 2019] Shah, S. Z. H., Karuppanan, S., Megat-Yusoff, P. S. M., and Sajid, Z. (2019). Impact resistance and damage tolerance of fiber reinforced composites: A review. *Composite Structures*, 217:100–121.

[Shi et al., 2022] Shi, Z., Zou, C., Zhou, F., and Zhao, J. (2022). Analysis of the Mechanical Properties and Damage Mechanism of Carbon Fiber/Epoxy Composites under UV Aging. *Materials*, 15(8):2919.

[Sohn et al., 2003] Sohn, H., Farrar, C. R., Hemez, F., and Czarnecki, J. (2003). A Review of Structural Health Monitoring Literature 1996 – 2001. *Los Alamos National Laboratory, USA*, 1:16.

[Soman et al., 2021] Soman, R., Wee, J., and Peters, K. (2021). Optical Fiber Sensors for Ultrasonic Structural Health Monitoring: A Review. *Sensors*, 21(21):7345.

[Sun et al., 2018] Sun, G., Wu, Y., Li, H., and Zhu, L. (2018). 3D shape sensing of flexible morphing wing using fiber Bragg grating sensing method. *Optik*, 156:83–92.

[Talreja, 2016] Talreja, R. (2016). Physical modelling of failure in composites. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2071):20150280.

[Tessler, 1998] Tessler, A. (1998). An improved variational method for finite element stress recovery and a posteriori error estimation. *Computer Methods in Applied Mechanics and Engineering*, 155(1-2):15–30.

[Tessler, 2003] Tessler, A. (2003). *A Variational Principle for Reconstruction of Elastic Deformations in Shear Deformable Plates and Shells.* National Aeronautics and Space Administration, Langley Research Center.

[Tessler and Spangler, 2004] Tessler, A. and Spangler, J. L. (2004). Inverse FEM for Full-Field Reconstruction of Elastic Deformations in Shear Deformable Plates and Shells. NTRS Author Affiliations: NASA Langley Research Center, Lockheed Martin Aeronautics Co. NTRS Meeting Information: 2nd European Workshop on Structural Health Monitoring; 2004-07-07 to 2004-07-09; undefined NTRS Document ID: 20040086696 NTRS Research Center: Langley Research Center (LaRC).

[Tessler and Spangler, 2005] Tessler, A. and Spangler, J. L. (2005). A least-squares variational method for full-field reconstruction of elastic deformations in shear-deformable plates and shells. *Computer Methods in Applied Mechanics and Engineering*, 194(2):327–339.

[Tidriri et al., 2016] Tidriri, K., Chatti, N., Verron, S., and Tiplica, T. (2016). Bridging data-driven and model-based approaches for process fault diagnosis and health monitoring: A review of researches and future challenges. *Annual Reviews in Control*, 42:63–81.

[Tinghu and Jones, 2004] Tinghu, Y. and Jones, B. (2004). Acoustic emission testing on large structures. *Measurement and Control*, 37(6):168–173.

[Ussorio et al., 2006] Ussorio, M., Wang, H., Ogin, S. L., Thorne, A. M., Reed, G. T., Tjin, S. C., and Suresh, R. (2006). Modifications to FBG sensor spectra due to matrix cracking in a GFRP composite. *Construction and Building Materials*, 20(1):111–118.

[Vidakovic et al., 2016] Vidakovic, M., McCague, C., Armakolas, I., Sun, T., Carlton, J. S., and Grattan, K. T. V. (2016). Fibre Bragg Grating-Based Cascaded Acoustic Sensors for Potential Marine Structural Condition Monitoring. *Journal of Lightwave Technology*, 34(19):4473–4478.

[Yang et al., 2017] Yang, Y., Liu, F., Fu, J., and Yang, D. (2017). Application of bridge continuous shape measurement system based on optical fiber sensing technology in bridge post-seismic detection. In *25th Optical Fiber Sensors Conference*, pages 1–4.

[Zeng and Liu, 2018] Zeng, W. and Liu, G. R. (2018). Smoothed Finite Element Methods (S-FEM): An Overview and Recent Developments. *Archives of Computational Methods in Engineering*, 25(2):397–435.

Part III

Appendices

Appendix A

Gangway Design

This is a confidential Appendix containing company information. This Appendix is only released to the graduation committee.

Appendix B

FEMAP Macros

```
'Macro for saving the displacements, strains, element nodes and node cooridnates from FEMAP to
    Excel

Sub Main
    'Define Number of Plies
    Dim NoPlies As Integer
    NoPlies = 1

    'Select OUTPUT ID to use
    Dim OutputID As Integer
    OutputID = 1
    Dim excelFilePathRoot As String
    excelFilePathRoot = "Your_Path"
    'excel Paths
    excelFilePathNodeCoord = excelFilePathRoot & "Node_coordinates.xlsx"
    excelFilePathElemNodes = excelFilePathRoot & "Element_Nodes.xlsx"
    excelFilePathReferenceU3 = excelFilePathRoot & "Reference_U3.xlsx"
    excelFilePathStrainResults = excelFilePathRoot & "StrainResults.xlsx"

    ' Create Excel objects and a workbooks
    Set objExcelNodeCoord = CreateObject("Excel.Application")
    Set objWorkbookNodeCoord = objExcelNodeCoord.Workbooks.Add

    Set objExcelElemNodes = CreateObject("Excel.Application")
    Set objWorkbookElemNodes = objExcelElemNodes.Workbooks.Add

    Set objExcelReferenceU3 = CreateObject("Excel.Application")
    Set objWorkbookReferenceU3 = objExcelReferenceU3.Workbooks.Add

    Set objExcelStrainResults = CreateObject("Excel.Application")
    Set objWorkbookStrainResults = objExcelStrainResults.Workbooks.Add

    ' Set up Excels
    objExcelNodeCoord.Visible = False
    objExcelNodeCoord.DisplayAlerts = False
    Set objWorksheetNodeCoord = objWorkbookNodeCoord.Sheets(1)

    objExcelElemNodes.Visible = False
    objExcelElemNodes.DisplayAlerts = False
    Set objWorksheetElemNodes = objWorkbookElemNodes.Sheets(1)
```

```
40
41      objExcelReferenceU3.Visible = False
42      objExcelReferenceU3.DisplayAlerts = False
43      Set objWorksheetReferenceU3 = objWorkbookReferenceU3.Sheets(1)
44
45      objExcelStrainResults.Visible = False
46      objExcelStrainResults.DisplayAlerts = False
47      Set objWorksheetStrainResults = objWorkbookStrainResults.Sheets(1)
48
49      '-------------------------------------------------------------------------------------------------------
50      'Iterate Per Node
51      Dim femap As femap.model
52      Set femap = GetObject(, "femap.model")
53      Dim nd As Object
54      Set nd = femap.feNode
55
56      'Results
57      Dim MyResultsQuery As femap.ResultsIDQuery
58      Set MyResultsQuery = femap.feResultsIDQuery
59      Dim MyNodalResults As femap.Results
60      Set MyNodalResults = femap.feResults
61
62      'Find ID's for Outputs of interest
63      Dim U3ID As Long
64      U3ID = MyResultsQuery.Nodal(VNV_TRANSLATION, VNT_Z)
65      femap.feAppMessage(FCM_NORMAL, "U3 ID:" + Str$(U3ID))
66
67      'Add columns
68      Dim nColumnsAdded As Long
69      Dim nColumnIndices As Variant
70      Dim dValU3 As Double
71      MyNodalResults.AddColumnV2(OutputID, U3ID, False, nColumnsAdded, nColumnIndices)
72      MyNodalResults.Populate
73
74      ' Write headers to Excel Node Coodinates
75      objWorksheetNodeCoord.Cells(1, 1).Value = "Node ID"
76      objWorksheetNodeCoord.Cells(1, 2).Value = "X Coordinate"
77      objWorksheetNodeCoord.Cells(1, 3).Value = "Y Coordinate"
78      objWorksheetNodeCoord.Cells(1, 4).Value = "Z Coordinate"
79
80      ' Write headers to Excel ReferenceU3
81      objWorksheetReferenceU3.Cells(1, 1).Value = "Node ID"
82      objWorksheetReferenceU3.Cells(1, 2).Value = "U3[m]"
83
84      Row = 1
85      While nd.Next
86          Row = Row + 1
87          objWorksheetNodeCoord.Cells(Row, 1).Value = nd.ID
88          objWorksheetNodeCoord.Cells(Row, 2).Value = nd.x
89          objWorksheetNodeCoord.Cells(Row, 3).Value = nd.y
90          objWorksheetNodeCoord.Cells(Row, 4).Value = nd.z
91          'femap.feAppMessage(FCM_NORMAL, "Value:" + Str$(nColumnIndices(0)))
92
93          objWorksheetReferenceU3.Cells(Row,1).Value = nd.ID
94          MyNodalResults.GetValue(nd.ID,nColumnIndices(0),dValU3)
95          objWorksheetReferenceU3.Cells(Row,2).Value = dValU3
96      Wend
97
98      '-------------------------------------------------------------------------------------------------------
99      'Iterate Per Element
100
101     ' Write headers to Element Nodes Excel
102     objWorksheetElemNodes.Cells(1, 1).Value = "ID"
```

```
    objWorksheetElemNodes.Cells(1, 2).Value = "C1"
    objWorksheetElemNodes.Cells(1, 3).Value = "C2"
    objWorksheetElemNodes.Cells(1, 4).Value = "C3"
    objWorksheetElemNodes.Cells(1, 5).Value = "C4"

    'Write headers to Element Strains Excel
    objWorksheetStrainResults.Cells(1, 1).Value ="ID"
    objWorksheetStrainResults.Cells(1, 2).Value ="Top Strain X"
    objWorksheetStrainResults.Cells(1, 3).Value ="Top Strain Y"
    objWorksheetStrainResults.Cells(1, 4).Value ="Top Strain XY"
    objWorksheetStrainResults.Cells(1, 5).Value ="Bot Strain X"
    objWorksheetStrainResults.Cells(1, 6).Value ="Bot Strain Y"
    objWorksheetStrainResults.Cells(1, 7).Value ="Bot Strain XY"

    Dim elems As Object
    Set elems = femap.feElem

    'Find ID's for Outputs of interest
    Dim topStrainXID As Long
    Dim topStrainYID As Long
    Dim topStrainXYID As Long

    Dim botStrainXID As Long
    Dim botStrainYID As Long
    Dim botStrainXYID As Long

    'Results
    Dim MyElementResults As femap.Results
    Set MyElementResults = femap.feResults

    topStrainXID = MyResultsQuery.Laminate(VPV_STRAIN, VPT_X, NoPlies, VPL_CENTROID)
    topStrainYID = MyResultsQuery.Laminate(VPV_STRAIN, VPT_Y, NoPlies, VPL_CENTROID)
    topStrainXYID = MyResultsQuery.Laminate(VPV_STRAIN, VPT_XY, NoPlies, VPL_CENTROID)

    botStrainXID = MyResultsQuery.Laminate(VPV_STRAIN, VPT_X, 1, VPL_CENTROID)
    botStrainYID = MyResultsQuery.Laminate(VPV_STRAIN, VPT_Y,1, VPL_CENTROID)
    botStrainXYID = MyResultsQuery.Laminate(VPV_STRAIN, VPT_XY, 1, VPL_CENTROID)

    'Add columns
    Dim eColumnsAdded As Long
    Dim eColumnIndicesTSX As Variant
    Dim eColumnIndicesTSY As Variant
    Dim eColumnIndicesTSXY As Variant
    Dim eColumnIndicesBSX As Variant
    Dim eColumnIndicesBSY As Variant
    Dim eColumnIndicesBSXY As Variant
    Dim dValTopStrainX As Double
    Dim dValTopStrainY As Double
    Dim dValTopStrainXY As Double
    Dim dValBotStrainX As Double
    Dim dValBotStrainY As Double
    Dim dValBotStrainXY As Double

    MyElementResults.AddColumnV2(OutputID, topStrainXID, False, eColumnsAdded, eColumnIndicesTSX)
    MyElementResults.AddColumnV2(OutputID, topStrainYID , False, eColumnsAdded, eColumnIndicesTSY)
    MyElementResults.AddColumnV2(OutputID, topStrainXYID, False, eColumnsAdded, eColumnIndicesTSXY)
    MyElementResults.AddColumnV2(OutputID, botStrainXID, False, eColumnsAdded, eColumnIndicesBSX)
    MyElementResults.AddColumnV2(OutputID, botStrainYID, False, eColumnsAdded, eColumnIndicesBSY)
    MyElementResults.AddColumnV2(OutputID, botStrainXYID, False, eColumnsAdded, eColumnIndicesBSXY)
    MyElementResults.Populate
    MyElementResults.SendToDataTable
    'femap.feAppMessage(FCM_NORMAL, "eColumnIndices:" +eColumnIndices)
    Row = 1
```

```
166     While elems.Next
167         Row = Row + 1
168         'elem_nodes = elems.node
169         objWorksheetElemNodes.Cells(Row, 1).Value = elems.ID
170         objWorksheetElemNodes.Cells(Row, 2).Value = elems.node(0)
171         objWorksheetElemNodes.Cells(Row, 3).Value = elems.node(1)
172         objWorksheetElemNodes.Cells(Row, 4).Value = elems.node(2)
173         objWorksheetElemNodes.Cells(Row, 5).Value = elems.node(3)
174 
175         MyElementResults.GetValue(elems.ID,eColumnIndicesTSX(0),dValTopStrainX)
176         MyElementResults.GetValue(elems.ID,eColumnIndicesTSY(0),dValTopStrainY)
177         MyElementResults.GetValue(elems.ID,eColumnIndicesTSXY(0),dValTopStrainXY)
178         MyElementResults.GetValue(elems.ID,eColumnIndicesBSX(0),dValBotStrainX)
179         MyElementResults.GetValue(elems.ID,eColumnIndicesBSY(0),dValBotStrainY)
180         MyElementResults.GetValue(elems.ID,eColumnIndicesBSXY(0),dValBotStrainXY)
181 
182         objWorksheetStrainResults.Cells(Row, 1).Value = elems.ID
183         objWorksheetStrainResults.Cells(Row, 2).Value = dValTopStrainX
184         objWorksheetStrainResults.Cells(Row, 3).Value = dValTopStrainY
185         objWorksheetStrainResults.Cells(Row, 4).Value = dValTopStrainXY
186         objWorksheetStrainResults.Cells(Row, 5).Value = dValBotStrainX
187         objWorksheetStrainResults.Cells(Row, 6).Value = dValBotStrainY
188         objWorksheetStrainResults.Cells(Row, 7).Value = dValBotStrainXY
189     Wend
190 
191     'Save Excels
192     objWorkbookNodeCoord.SaveAs excelFilePathNodeCoord
193     objWorkbookElemNodes.SaveAs excelFilePathElemNodes
194     objWorkbookReferenceU3.SaveAs excelFilePathReferenceU3
195     objWorkbookStrainResults.SaveAs excelFilePathStrainResults
196 
197     ' Clean up and close Excel
198     objWorkbookNodeCoord.Close
199     objExcelNodeCoord.Quit
200     Set objWorksheetNodeCoord= Nothing
201     Set objWorkbookNodeCoord = Nothing
202     Set objExcelNodeCoord = Nothing
203 
204     objWorkbookElemNodes.Close
205     objExcelElemNodes.Quit
206     Set objWorksheetElemNodes = Nothing
207     Set objWorkbookElemNodes = Nothing
208     Set objExcelElemNodes = Nothing
209 
210     objWorkbookReferenceU3.Close
211     objExcelReferenceU3.Quit
212     Set objWorksheetReferenceU3 = Nothing
213     Set objWorkbookReferenceU3= Nothing
214     Set objExcelReferenceU3 = Nothing
215 
216     objWorkbookStrainResults.Close
217     objExcelStrainResults.Quit
218     Set objWorksheetStrainResults = Nothing
219     Set objWorkbookStrainResults= Nothing
220     Set objExceStrainResults = Nothing
221 
222 End Sub
```

Listing B.1: Macro for exporting FEMAP model data as iFEM input for anisotropic plate elements

Appendix C

Code

The procedure for adding a new element type is the following:

- Using *IQS4_derivation.py*, the shape functions of the new element can be included for deriving using sympy the B matrices.
- The outcomes need to be hardcoded as functions in a file such as *iqs4_equations.py*.
- A new element class can be created using *iqs4.pyx*.
- The functional definition can be adapted by adapting or adding functions to *helpers.py*.

```
import numpy as np
from scipy.linalg import block_diag
from pyife3d.iqs4_equations import matrices_SEA, NLM_matrices
import pandas as pd
from functools import partial
import os

def Gaussian_option(Gauss_type):
    """
    Function for generating the data for the Gauss quadrature points

    Args:
        Gauss_type (_str_): The user can select the type of Gaussian quadrature. It is Based on
            the Gauss-Legendre quadrature. Current options available are "1-point", "2-point",
            "3-point", "4-point", "5-point".

    Returns:
        Gauss_points_weights (_list_): List of lists (can be converted to numpy array) where the
            first column is the evaluation point x and the second column is the associated weight
            for the sum
    """
    #NOTE Gauss quadratures from
    #https://keisan.casio.com/exec/system/1329114617

    if Gauss_type == "1-point":
        # 1-point Gauss-Legendre
```

```
        Gauss_points_weights = [
        [0, 2.],
        ]

    elif Gauss_type == "2-point":
    # 2-point Gauss-Legendre
        Gauss_points_weights = [
        [-0.5773502691896257645092, 1],
        [0.5773502691896257645092, 1],
        ]

    elif Gauss_type == "3-point":
        # 3-point Gauss-Legendre
        Gauss_points_weights = [
        [-0.7745966692414833770359, 0.5555555555555555555556],
        [0, 0.8888888888888888888889],
        [0.7745966692414833770359, 0.5555555555555555555556],
        ]

    elif Gauss_type == "4-point":
        # 4-point Gauss-Legendre
        Gauss_points_weights = [
        [-0.8611363115940525752 24, 0.3478548451374538573731],
        [-0.3399810435848562648027, 0.6521451548625461426269],
        [0.3399810435848562648027, 0.6521451548625461426269],
        [0.8611363115940525752 24, 0.3478548451374538573731],
                ]

    elif Gauss_type == "5-point":
        # 5-point Gauss-Legendre
        Gauss_points_weights = [
        [-0.9061798459386639927976, 0.2369268850561890875143],
        [-0.5384693101056830910363, 0.4786286704993664680413],
        [0, 0.5688888888888888888889],
        [0.5384693101056830910363, 0.4786286704993664680413],
        [0.9061798459386639927976, 0.2369268850561890875143],
        ]

    return Gauss_points_weights

def read_iFEM_files(path_node_coord, path_element_nodes, path_strain_data):
    """
    Function for reading the files required for solving the iFEM model. They can be in either
        excel or csv format using comma delimiter.

    Args:
        path_node_coord (str): Path to the Node_Coordinates file.
        path_element_nodes (str): Path to the Element_Nodes file.
        path_strain_data (str): Path to the StrainResults file.

    Returns:
        node_coord (array): Array of size (N_nodes,4) containing the node coordinates of the mesh
            in the format: ID | X | Y | Z.
        element_nodes (array): Array of size (N_elements,5) containing the nodes of each element
            in the format: element ID | node 1 ID | node 2 ID | node 3 ID | node 4 ID
        strain_data (array): Array of size (N_sensing,7) element ID | exx top | eyy top | gxy top
            | exx bot | eyy bot | gxz bot
    """
    #node_coord format: ID | X | Y | Z
    if path_node_coord[-3:]=="csv":
        node_coord = pd.read_csv(path_node_coord,delimiter=',')
    elif path_node_coord[-4:]=="xlsx":
        node_coord = pd.read_excel(path_node_coord)
```

```
    node_coord = node_coord.to_numpy()

    #element_nodes format: element ID | node 1 ID | node 2 ID | node 3 ID | node 4 ID
    if path_element_nodes[-3:]=="csv":
        element_nodes = pd.read_csv(path_element_nodes,delimiter=',')
    elif path_element_nodes[-4:]=="xlsx":
        element_nodes = pd.read_excel(path_element_nodes)
    element_nodes = element_nodes.to_numpy()

    #format: element ID | exx top | eyy top | gxy top | exx bot | eyy bot | gxz bot
    if path_element_nodes[-3:]=="csv":
        strain_data = pd.read_csv(path_strain_data, delimiter=',')
    elif path_element_nodes[-4:]=="xlsx":
        strain_data = pd.read_excel(path_strain_data)

    strain_data = strain_data.sort_values(by='ID', ascending=True)
    strain_data = strain_data.to_numpy()

    return node_coord, element_nodes, strain_data

def format_strain_data(N_elements,strain_data):
    """
    Function for formatting the fed strain data into top and bottom measurements. If a measurement
        is not registered, a 0 is filled in for that component.

    Args:
        N_elements (int): Number of elements in the model
        strain_data (array): Array of size (N_sensing_points,7) storing the strain corresponding
            to each element in the following format element ID | exx top | eyy top | gxy top |
            exx bot | eyy bot | gxz bot

    Returns:
        strain_gauge_top (array): Array of size (N_elements,4) storing the strain corresponding to
            each element in the following format element ID | exx top | eyy top | gxy top
        strain_gauge_bot (array): Array of size (N_elements,4) storing the strain corresponding to
            each element in the following format element ID | exx bot | eyy bot | gxz bot
    """
    #format: Element ID | exx | eyy | exy with N in ascending order
    strain_gauge_top = np.zeros((N_elements,4))
    strain_gauge_bot = np.zeros((N_elements,4))

    #Checks if an element is mentioned in the strain elements
    check_elements =  np.zeros(N_elements, dtype=bool)
    total_elements = np.arange(1,N_elements+1)
    check_elements = np.isin(total_elements,strain_data[:,0])

    strain_gauge_top[:,0] = total_elements
    strain_gauge_bot[:,0] = total_elements

    strain_gauge_top[check_elements,1] = strain_data[:,1]
    strain_gauge_top[check_elements,2] = strain_data[:,2]
    strain_gauge_top[check_elements,3] = strain_data[:,3]

    strain_gauge_bot[check_elements,1] = strain_data[:,4]
    strain_gauge_bot[check_elements,2] = strain_data[:,5]
    strain_gauge_bot[check_elements,3] = strain_data[:,6]

    #Handling nan values
    strain_gauge_bot[np.isnan(strain_gauge_bot)] = 0
    strain_gauge_top[np.isnan(strain_gauge_top)] = 0

    return strain_gauge_top,strain_gauge_bot

```

```
140 def assemble_strain_elements(strain_gauge_top):
141     """
142     Function used for saving which element ID's have strain measurements. This info is later used
            for checking of w parameter or SEA implementation.
143 
144     Args:
145         strain_gauge_top (array): Array of size (Nelements,4) containing the strain data of the
                top surface. The format is: Element ID | exx | eyy | exy with N in ascending order. 0
                values are inserted where measurements are missing
146     Returns:
147         strain_elements (dict): Dictionary containing arrays of the elements where strain is
                recorded for each strain component. Eg. for strain exx we know whic elemebnts record
                strain. The keys are "exx", "eyy" and "exy".
148     """
149     strain_elements = {}
150 
151     check_exx = np.where(strain_gauge_top[:,1]!=0)[0]
152     check_eyy = np.where(strain_gauge_top[:,2]!=0)[0]
153     check_exy = np.where(strain_gauge_top[:,3]!=0)[0]
154 
155     strain_elements["exx"] = strain_gauge_top[check_exx,0]
156     strain_elements["eyy"] = strain_gauge_top[check_eyy,0]
157     strain_elements["exy"] = strain_gauge_top[check_exy,0]
158 
159     return strain_elements
160 
161 def quadrature_alignment_factor(w,xi,eta):
162     """
163     This function is used for an additional correction factor alfa. The srain gauges are assummed
            to be placed at the the centroid of the element. But when we do the gauss quadrature, we
            calculate at multiple points. Some of these points do not coincide with the centroid. In
            the case where there is a measurement point (so weight is 1), its contirbution needs to
            be lowered with alfa as the integration point does not actually coincide with the
            measurement point.
164     Args:
165         w (float): Dictionary of weight per compoennts
166         xi (float) Natural coordinate
167         eta (float): Natural coordinate
168 
169     Returns:
170         alfa (float): Correction factor alfa that needs to be applied.
171     """
172     alfa = np.ones((8)) #for all of our strain.curvature components to keep consistency, although
            gxz and gyz we will never obtain experimentally
173 
174     if xi!=0 and eta!=0:
175         alfa[w>=1] = 1e-4
176 
177     return alfa
178 
179 def
        exp_strain_builder(quad,strain_elements,w,location,T_mat,strain_gauge_top,strain_gauge_bot,i,SEA_U_dict_top,
        SEA_U_dict_bot, N, L, M, h):
180     """
181     Function for building the experimental strain vectors. These vectors are built depending on
            whether or not the element has strains measured or not. Also, the w vector is updated
            depending on that.
182 
183     Args:
184         quad (object):IQS4 SEA object
185         strain_elements (dict): Dictionary containing arrays of the elements where strain is
                recorded for each strain component. Eg. for strain exx we know whic elemebnts record
                strain. The keys are "exx", "eyy" and "exy".
```

```
186         w (array): Array of size 8 representing the Weights for least-squares variational
                principle. Format: exx | eyy | gxy | kxx | kyy | kxy | gxz | gyz
187         location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
188         T_mat (array): Numpy array of size (3,3) representing the matrix for aligning a coordinate
                system with the material direction.
189         strain_gauge_top (array): Array of size (N_elements,4) storing the strain corresponding to
                each element in the following format element ID | exx top | eyy top | gxy top
190         strain_gauge_bot (array): Array of size (N_elements,4) storing the strain corresponding to
                each element in the following format element ID | exx bot | eyy bot | gxz bot
191         i (int): Iteration number for going through element indices.
192         SEA_U_dict (dict): Dictionary containing the U_SEA for the strain component "exx", "eyy",
                "exy
193         N (array): Numpy array of size (4,1) containing the filled in values of the shape
                functions for the element and xi=0 and eta=0.
194         L (array): Numpy array of size (4,1) containing the filled in values of the shape
                functions for the element and xi=0 and eta=0.
195         M (array): Numpy array of size (4,1) containing the filled in values of the shape
                functions for the element and xi=0 and eta=0.
196         h (float): Half thickness of the plate.
197
198     Returns:
199         exp_e (array): (3,1) numpy array containing the experimental strains exx | eyy | gxy
200         exp_k (array): (3,1) numpy array containing the experimental kurvatures kxx | kyy | kxy
201         w (array): Updated array of size 8 representing the Weights for least-squares variational
                principle. Format: exx | eyy | gxy | kxx | kyy | kxy | gxz | gyz
202     """
203
204     if quad.eid in strain_elements["exx"] or quad.eid in strain_elements["eyy"] or quad.eid in
            strain_elements["exy"]:
205
206         strain_gauge_top_mat = np.matmul(T_mat,strain_gauge_top[i,1:]) #start from index 1 cause 0
                is element ID
207         strain_gauge_bot_mat = np.matmul(T_mat,strain_gauge_bot[i,1:])
208
209         if location == "top":
210             exp_e = strain_gauge_top_mat
211             exp_k = np.zeros((3,1))
212         elif location == "mid":
213             exp_e = 1/2*(strain_gauge_top_mat+strain_gauge_bot_mat)
214             exp_k = 1/(2*h)*(strain_gauge_top_mat-strain_gauge_bot_mat)
215         else:
216             exp_e = strain_gauge_bot_mat
217             exp_k = np.zeros((3,1))
218
219         #Change indices appropiately
220         if quad.eid in strain_elements["exx"] and location=="mid":
221             w[0] = 1
222             w[3] = 1
223         elif quad.eid in strain_elements["exx"]:
224             w[0] = 1
225
226         if quad.eid in strain_elements["eyy"] and location=="mid":
227             w[1] = 1
228             w[4] = 1
229         elif quad.eid in strain_elements["eyy"]:
230             w[1] = 1
231
232         if quad.eid in strain_elements["exy"] and location=="mid":
233             w[2] = 1
234             w[5] = 1
235         elif quad.eid in strain_elements["exy"]:
236             w[2] = 1
237
```

```
238
239     else:
240         #We interpolate with SEA
241         DOF=4 #the DOF that are used in the U_SEA files
242         ind_nodes = np.array([quad.n1,quad.n2,quad.n3,quad.n4])-1
243         #-------------------Exx---------------------------
244         U_SEA = SEA_U_dict_top["exx"]
245         exx_top = np.matmul(N,U_SEA[ind_nodes*DOF]) \
246                 - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
247                 - np.matmul(M,U_SEA[ind_nodes*DOF+2])
248
249         U_SEA = SEA_U_dict_bot["exx"]
250         exx_bot = np.matmul(N,U_SEA[ind_nodes*DOF]) \
251                 - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
252                 - np.matmul(M,U_SEA[ind_nodes*DOF+2])
253
254         #-------------------Eyy---------------------------
255         U_SEA = SEA_U_dict_top["eyy"]
256         eyy_top = np.matmul(N,U_SEA[ind_nodes*DOF]) \
257                 - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
258                 - np.matmul(M,U_SEA[ind_nodes*DOF+2])
259
260         U_SEA = SEA_U_dict_bot["eyy"]
261         eyy_bot = np.matmul(N,U_SEA[ind_nodes*DOF]) \
262                 - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
263                 - np.matmul(M,U_SEA[ind_nodes*DOF+2])
264
265         #-------------------Exy---------------------------------
266         U_SEA = SEA_U_dict_top["exy"]
267         exy_top = np.matmul(N,U_SEA[ind_nodes*DOF]) \
268                 - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
269                 - np.matmul(M,U_SEA[ind_nodes*DOF+2])
270
271         U_SEA = SEA_U_dict_bot["exy"]
272         exy_bot = np.matmul(N,U_SEA[ind_nodes*DOF]) \
273                 - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
274                 - np.matmul(M,U_SEA[ind_nodes*DOF+2])
275
276         e_top = np.array([float(exx_top),float(eyy_top),float(exy_top)])
277         e_bot = np.array([float(exx_bot),float(eyy_bot),float(exy_bot)])
278
279         #Align with material
280         #Here the strains are not vectors just floats
281         strain_gauge_top_mat = np.matmul(T_mat,e_top)
282         strain_gauge_bot_mat = np.matmul(T_mat,e_bot)
283
284         if location == "top": #We do not change the coefficients now. It is not an actual
                measurement
285             exp_e = strain_gauge_top_mat
286             exp_k = np.zeros((3,1))
287         elif location == "mid":
288             exp_e = 1/2*(strain_gauge_top_mat+strain_gauge_bot_mat)
289             exp_k = 1/(2*h)*(strain_gauge_top_mat-strain_gauge_bot_mat)
290         else:
291             exp_e = strain_gauge_bot_mat
292             exp_k = np.zeros((3,1))
293
294     quad.probe.epsilontopSEA = strain_gauge_top_mat
295     quad.probe.epsilonbotSEA = strain_gauge_bot_mat
296
297     return exp_e, exp_k, w, quad
298
299 def compute_local_matrices(B_funct_partial, w, Gauss_points_weights,h,quad,exp_e,exp_k):
```

```
    """
    Function for computing the local matrices ke and fe for iFEM.

    Args:
        B_funct_partial (function): Partial function in which the natural coordinates of the
            points of the element have been filled in. Only the natural coordinates of the
            integration point (xi,eta) need to be filled in.
        w (array): Array of size 8 representing the Weights for least-squares variational
            principle. Format: exx | eyy | gxy | kxx | kyy | kxy | gxz | gyz
        Gauss_points_weights (list): List of lists (can be converted to numpy array) where the
            first column is the evaluation point x and the second column is the associated weight
            for the sum
        h (float): Half thickness of the plate.
        quad (_type_): _description_
        exp_e (array): (3,1) numpy array containing the experimental strains exx | eyy | gxy
        exp_k (array): (3,1) numpy array containing the experimental kurvatures kxx | kyy | kxy

    Returns:
        quad (object) : Updated quad object after computing the local matrices fe and ke
    """
    #Bb, Bs, Bm are already calculated for our selected gauss natural coordinates so they need to
        be calculated at the natural coordinates

    ke = np.zeros((24,24)) # the 3 is from the shape funct size4*6 DOF
    fe = np.zeros((24,1))

    #Calculate the strains at the mid-plane
    #Compute ke and fe at the same time so you do not need to rebuild the matrices
    for xi_val, wi in Gauss_points_weights:
        for eta_val, wj in Gauss_points_weights:
            wij = wi*wj #Gauss Quadrature factor
            #Calculate Bb, Bs, Bm for the current xi and eta
            alfa = quadrature_alignment_factor(w=w,xi=xi_val,eta=eta_val)
            # alfa = np.ones((8,1))
            detJ, Bm, Bb, Bs = B_funct_partial(xi=xi_val, eta=eta_val)
            area = 4 * detJ
            #Calculating the ke infinitesimal function at the prescribed xi and eta
            #For ke

            ke_curr_Bmexx =  w[0] * alfa[0] * np.outer(np.transpose(Bm[0,:]),Bm[0,:])
            ke_curr_Bmeyy =  w[1] * alfa[1] * np.outer(np.transpose(Bm[1,:]),Bm[1,:])
            ke_curr_Bmgxy =  w[2] * alfa[2] * np.outer(np.transpose(Bm[2,:]),Bm[2,:])
            ke_curr_Bm = ke_curr_Bmexx + ke_curr_Bmeyy + ke_curr_Bmgxy

            ke_curr_Bbexx = w[3] * pow(2*h,2) * alfa[3] * np.outer(np.transpose(Bb[0,:]),Bb[0,:])
            ke_curr_Bbeyy = w[4] * pow(2*h,2) * alfa[4] * np.outer(np.transpose(Bb[1,:]),Bb[1,:])
            ke_curr_Bbgxy = w[5] * pow(2*h,2) * alfa[5] * np.outer(np.transpose(Bb[2,:]),Bb[2,:])
            ke_curr_Bb = ke_curr_Bbexx + ke_curr_Bbeyy + ke_curr_Bbgxy

            ke_curr_Bsgxz = pow(10,-5) * alfa[6] * np.outer(np.transpose(Bs[0,:]),Bs[0,:]) #w[6]
            ke_curr_Bsgyz = pow(10,-5) * alfa[7] * np.outer(np.transpose(Bs[1,:]),Bs[1,:]) #w[7]
            ke_curr_Bs = ke_curr_Bsgxz + ke_curr_Bsgyz

            ke_curr = ke_curr_Bm + ke_curr_Bb + ke_curr_Bs
            ke = ke + wij*ke_curr/area*detJ #adding it to the integral taking into account the
                gauss

            #Calculating the fe infinitesimal function at the prescribed xi and eta
            fe_curr_Bmexx = w[0] * alfa[0] * np.transpose(Bm[0,:]) * exp_e[0]
            fe_curr_Bmeyy = w[1] * alfa[1] * np.transpose(Bm[1,:]) * exp_e[1]
            fe_curr_Bmgxy = w[2] * alfa[2] * np.transpose(Bm[2,:]) * exp_e[2]
            fe_curr_Bm =  fe_curr_Bmexx +  fe_curr_Bmeyy +  fe_curr_Bmgxy

```

```
356             fe_curr_Bbexx = w[3] * alfa[3] * pow(2*h,2) * np.transpose(Bb[0,:]) * exp_k[0]
357             fe_curr_Bbeyy = w[4] * alfa[4] * pow(2*h,2) * np.transpose(Bb[1,:]) * exp_k[1]
358             fe_curr_Bbgxy = w[5] * alfa[5] * pow(2*h,2) * np.transpose(Bb[2,:]) * exp_k[2]
359             fe_curr_Bb =  fe_curr_Bbexx +  fe_curr_Bbeyy +  fe_curr_Bbgxy
360
361             fe_curr = np.reshape((fe_curr_Bm + fe_curr_Bb),(24,1)) #the reshape is required else
                    (24,) shape afects the fe shape
362
363             fe = fe + wij*fe_curr/area*detJ
364
365     quad.ke = ke
366     quad.fe = fe
367
368     return quad
369
370 def compute_local_matrices_extrapolation(alfaSEA, betaSEA, drllingfact, B_funct_partial,
     Gauss_points_weights,quad,strain_gauge,strain_elements):
371     """
372     Function for calculating the local matrices ke and fe for the strain extrapolation part.
373
374     Args:
375         alfaSEA (float): Alfa factor of SEA strain extrapolation.
376         betaSEA (float): Beta factor of SEA strain extrapolation.
377         drllingfact (float): Drilling degree of freedom assumed factor.
378         B_funct_partial (function): Partial function in which the natural coordinates of the
                points of the element have been filled in. Only the natural coordinates of the
                integration point (xi,eta) need to be filled in.
379         Gauss_points_weights (list): List of lists (can be converted to numpy array) where the
                first column is the evaluation point x and the second column is the associated weight
                for the sum
380         quad (object):IQS4 SEA object
381         strain_gauge (array): Array of size (N_elements,2) containing strain values of the
                selected component with format of ELEMENT ID | strain measurements
382         strain_elements (list): List containing the indices of the strain elements for which the
                strain component is smeasured.
383
384     Returns:
385          quad (object) : Updated quad object after computing the local matrices fe and ke
386     """
387     DOF= 4
388     #Bb, Bs, Bm are already calculated for our selected gauss integration points
389     ke = np.zeros((12,12)) # 4 nodes * 3 DOF (we do not consider the drilling DOF here. we add it
            later)
390     fe = np.zeros((12,1))
391
392     n_sens = len(strain_elements) #number of sensing points
393
394     #Calculate the strains at the mid-plane
395     #Compute ke and fe at the same time so you do not need to rebuild the matrices
396     for xi_val, wi in Gauss_points_weights:
397         for eta_val, wj in Gauss_points_weights:
398             wij = wi*wj #Gauss Quadrature factor
399             #Calculate kalfa, kbeta and kepsilon for the current xi and eta
400             detJ, N_tilde, Kalfa, Kbeta, Kalfa_B1, Kalfa_B2  = B_funct_partial(xi=xi_val,
                    eta=eta_val)
401
402             #Calculating the ke infinitesimal function at the prescribed xi and eta
403             #For ke
404             ke_curr = wij*(alfaSEA* Kalfa * detJ + betaSEA * Kbeta * detJ)
405             ke = ke + ke_curr
406
407     if quad.eid in strain_elements: #for these components we do not need to go through the gauss
            integration
```

```
408             _,N_tilde,_, _, _, _ = B_funct_partial(xi=0, eta=0) #assume now that strain is placed in
                    the centroid of the element
409
410             Keps = 1/n_sens * np.matmul(np.transpose(N_tilde),N_tilde)
411             ke = ke + Keps
412             exp_e = strain_gauge[strain_gauge[:,0]==quad.eid,1]
413             fe = fe + 1/n_sens * exp_e * np.transpose(N_tilde)
414
415         ke_dril = ke
416         fe_dril = fe
417
418         for i in range(0,4):
419             ke_dril = np.insert(ke_dril,i*DOF+3,0,axis=0)
420             ke_dril = np.insert(ke_dril,i*DOF+3,0,axis=1)
421             ke_dril[i*DOF+3,i*DOF+3] = drllingfact
422             fe_dril = np.insert(fe_dril,i*DOF+3,0,axis=0)
423
424         quad.ke = ke_dril
425         quad.fe = fe_dril
426
427         return quad
428
429 def form_global_matrices(quads,N_nodes,DOF):
430     """
431     Function for assembling the global K matrix (K matrix of inverse FEM not the stiffness matrix)
432
433     Args:
434         quads (list): List containing quad elements.
435         N_nodes (int): Number of nodes in the iFEM model.
436         DOF (int): Number of degrees of freedom.
437
438     Returns:
439         K (array): Numpy array of size (N_nodes*DOF,N_nodes*DOF) representing the global K matrix
                of the iFEM model.
440         F (array): Numpy array of size (N_nodes*DOF,N_nodes*DOF) representing the global K matrix
                of the iFEM model.
441     """
442     K = np.zeros((N_nodes*DOF,N_nodes*DOF))
443     F = np.zeros((N_nodes*DOF,1))
444
445     for quad in quads: #now that we have all of our local matrices assembled we can create the
            global ones
446         T = quad.Te
447
448         if DOF == 6: #iFEM
449             Te = block_diag(T,T,T,T,T,T,T,T)
450         if DOF == 4: #iFEM SEA
451             #The last degree of freedom is artificial so we do not need rotation for it
452             Tr = np.eye(4)
453             Tr[1:4,1:4] = T
454             Te = block_diag(Tr,Tr,Tr,Tr)
455
456         K_curr = np.matmul(np.transpose(Te),np.matmul(quad.ke,Te))
457         F_curr = np.matmul(np.transpose(Te),quad.fe)
458
459         element_indices=[]
460         for node in [quad.n1,quad.n2,quad.n3,quad.n4]:
461             idx = node-1
462             if DOF ==6:
463                 element_indices.extend([idx*DOF,idx*DOF+1,idx*DOF+2,idx*DOF+3,idx*DOF+4,idx*DOF+5])
464             elif DOF ==4:
465                 element_indices.extend([idx*DOF,idx*DOF+1,idx*DOF+2,idx*DOF+3])
466
```

```
467         K[np.ix_(element_indices, element_indices)] = K[np.ix_(element_indices, element_indices)]
                + K_curr
468         F[element_indices] = F[element_indices] + F_curr
469 
470     return K, F
471 
472 def form_global_matrices_partSEA(quads,N_nodes,DOF):
473     """
474     Function for assembling the global K matrix (K matrix of inverse FEM not the stiffness matrix)
475 
476     Args:
477         quads (list): List containing quad elements.
478         N_nodes (int): Number of nodes in the iFEM model.
479         DOF (int): Number of degrees of freedom.
480 
481     Returns:
482         K (array): Numpy array of size (N_nodes*DOF,N_nodes*DOF) representing the global K matrix
                of the iFEM model.
483         F (array): Numpy array of size (N_nodes*DOF,N_nodes*DOF) representing the global K matrix
                of the iFEM model.
484     """
485     K = np.zeros((N_nodes*DOF,N_nodes*DOF))
486     F = np.zeros((N_nodes*DOF,1))
487 
488     indices_sub = []
489 
490     for quad in quads: #now that we have all of our local matrices assembled we can create the
            global ones
491         T = quad.Te
492 
493         if DOF == 6: #iFEM
494             Te = block_diag(T,T,T,T,T,T,T,T)
495         if DOF == 4: #iFEM SEA
496             #The last degree of freedom is artificial so we do not need rotation for it
497             Tr = np.eye(4)
498             Tr[1:4,1:4] = T
499             Te = block_diag(Tr,Tr,Tr,Tr)
500 
501         K_curr = np.matmul(np.transpose(Te),np.matmul(quad.ke,Te))
502         F_curr = np.matmul(np.transpose(Te),quad.fe)
503 
504         element_indices=[]
505         for node in [quad.n1,quad.n2,quad.n3,quad.n4]:
506             idx = node-1
507             if DOF ==6:
508                 element_indices.extend([idx*DOF,idx*DOF+1,idx*DOF+2,idx*DOF+3,idx*DOF+4,idx*DOF+5])
509             elif DOF ==4:
510                 element_indices.extend([idx*DOF,idx*DOF+1,idx*DOF+2,idx*DOF+3])
511 
512         K[np.ix_(element_indices, element_indices)] = K[np.ix_(element_indices, element_indices)]
                + K_curr
513         F[element_indices] = F[element_indices] + F_curr
514 
515         indices_sub.extend(element_indices)
516     indices_sub = np.unique(indices_sub)
517 
518     return K[np.ix_(indices_sub, indices_sub)], F[indices_sub], indices_sub
519 
520 def K_conditioning_number(calculate_bool, K, save_path, name_var, location):
521 
522     """
523     Calculates the conditioning number of the K matrix.
524 
```

```
    Args:
        calculate_bool (bool): Calculate or not. If true, save the value in the error txt file of
            the case. If not, just mentioned not calculated. This operation is very time
            consuming so most of the time is an option that should be skipped.
        K (array): Numpy array of size (N_nodes*DOF,N_nodes*DOF) representing the global K matrix
            of the iFEM model.
        save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
        name_var (str): Name of the variable for which error is computed.
        location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
    """
    if not os.path.exists(save_path):
            os.makedirs(save_path)
    f= open(save_path+f"\\error_{name_var}_{location}.png","a+")

    if calculate_bool:
        f.write(f"\n Inverse conditioning number for K={np.linalg.cond(K)}")
        f.write(f"\n Inverse conditioning number for inv K={np.linalg.cond(np.linalg.inv(K))}")
    else:
        f.write("\n Inverse conditioning number for K=Not Computed")

def alpha_iteration_rho_eta(quads, U_SEA, strain_gauge, Gauss_points_weights,strain_elements):
    """
    Function for calculating rho and eta for the alpha iteration throught e L2 curve.

    Args:
        quads (list): List containing quad elements.
        U_SEA (array): Array of size (N_nodes*4,1) containing the [s,sx,sy,sz] results of the SEA
            analysis in the format [s1,sx1,sy1,sz1,s2,sx2,sy2,sz2..]
        strain_gauge (array): Array of size (N_elements,2) containing strain values of the
            selected component with format of ELEMENT ID | strain measurements
        Gauss_points_weights (list): List of lists (can be converted to numpy array) where the
            first column is the evaluation point x and the second column is the associated weight
            for the sum
        strain_elements (list): List containing the indices of the strain elements for which the
            strain component is smeasured.

    Returns:
        floats:  phi_eps, phi_alpha
    """

    phi_eps = 0
    phi_alpha = 0
    DOF = 4 #this is how the U_SEA data is formatted

    for quad in quads:
        x_loc = quad.probe.x_nat
        matrices_SEA_partial = partial(matrices_SEA,x1=x_loc[0,0],
            y1= x_loc[1,0],
            x2= x_loc[3,0],
            y2= x_loc[4,0],
            x3= x_loc[6,0],
            y3= x_loc[7,0],
            x4= x_loc[9,0],
            y4= x_loc[10,0])

        # Initializing arrays of experimental strain and curavture of an element at its midplane
        n_sens = len(strain_elements)

        ind_nodes = np.array([quad.n1,quad.n2,quad.n3,quad.n4])-1
        u = np.zeros((12,1))
        u_loc = np.zeros((16,1))

        T = quad.Te
```

```
580         Tr = np.eye(4)
581         Tr[1:4,1:4] = T
582         Te = block_diag(Tr,Tr,Tr,Tr)
583 
584         u_loc[0::4] = U_SEA[ind_nodes*DOF]
585         u_loc[1::4] = U_SEA[ind_nodes*DOF+1]
586         u_loc[2::4] = U_SEA[ind_nodes*DOF+2]
587         u_loc[3::4] = U_SEA[ind_nodes*DOF+3]
588 
589         u_loc = np.matmul(Te,u_loc)
590         u[0::3] = u_loc[0::4]
591         u[1::3] = u_loc[1::4]
592         u[2::3] = u_loc[2::4]
593 
594 
595         for xi_val, wi in Gauss_points_weights:
596             for eta_val, wj in Gauss_points_weights:
597                 wij = wi*wj
598 
599                 detJ, N_tilde, Kalfa, Kbeta, Kalfa_B1, Kalfa_B2 =matrices_SEA_partial(xi=xi_val,
                        eta=eta_val)
600 
601                 phi_alfa1 =
                        np.matmul(np.transpose(u),np.matmul(np.transpose(Kalfa_B1),np.matmul(Kalfa_B1,u)))
602                 phi_alfa2 =
                        np.matmul(np.transpose(u),np.matmul(np.transpose(Kalfa_B2),np.matmul(Kalfa_B2,u)))
603 
604                 phi_alpha += wij*detJ*(phi_alfa1+phi_alfa2)
605 
606         if quad.eid in strain_elements:
607 
608             exp_e = strain_gauge[strain_gauge[:,0]==quad.eid,1]
609 
610             _,N_tilde,_, _, _, _ = matrices_SEA_partial(xi=0, eta=0)
611 
612             Keps = 1/n_sens * np.matmul(np.transpose(N_tilde),N_tilde)
613             f_eps = 1/n_sens * exp_e * np.transpose(N_tilde)
614 
615             phi_eps_curr =  1/n_sens * np.matmul(np.transpose(exp_e),exp_e)+
                    np.matmul(np.transpose(u),(np.matmul(Keps,u))) -
                    2*np.matmul(np.transpose(f_eps),u)
616 
617             phi_eps += phi_eps_curr
618 
619     return phi_eps, phi_alpha
620 
621 def errors_path(calculated_var,N_nodes,reference_path):
622     """
623     Function for plotting the percentage error at each node in the FEM model. 2d view.
624     When the reference value is 0, to avoid Nan the error is eplaced by 0. Might not be a
            representative error handling in all cases
625 
626     Args:
627         calculated_var (array): Array of size (N_nodes,1) with the values of a calculated array
628         N_nodes (int): Number of nodes in the iFEM mesh
629         reference_path (str): Path to the file where the reference measurements (FEM outputs) are
                stored.
630     """
631 
632     MPD = 0  #mean percentage difference
633     MAPD = 0 #mean absolute percentage difference
634     RMSD = 0 #root mean square difference
635 
```

```
636     if reference_path[-3:] == "csv":
637         referenece_var = pd.read_csv(reference_path,delimiter=',')
638     elif reference_path[-4:] == "xlsx":
639         referenece_var = pd.read_excel(reference_path)
640     referenece_var = referenece_var.to_numpy()
641 
642     error = np.zeros(np.shape(referenece_var))
643     error[:,0] = referenece_var[:,0]
644 
645     error[:,1] = (calculated_var[:,0]-referenece_var[:,1]) #simplle difference
646     RMSD = np.sqrt(np.sum(error[:,1]*error[:,1])/N_nodes)
647 
648     a = error[:,1]*100
649     b = referenece_var[:,1]
650     error[:,1] = np.divide(a, b, out=np.zeros_like(a), where=b!=0) #substitute nan by 0. PD
            obtained
651     MPD = np.sum(error[:,1])/N_nodes
652     MAPD = np.sum(np.absolute(error[:,1]))/N_nodes
653 
654     return RMSD, MPD, MAPD
655 
656 def errors(calculated_var,reference_var):
657     """
658     Function for plotting the percentage error at each node in the FEM model. 2d view.
659     When the reference value is 0, to avoid Nan the error is eplaced by 0. Might not be a
            representative error handling in all cases
660 
661     Args:
662         calculated_var (array): Array of size (N_nodes,1) with the values of a calculated array
663         reference_var(array): Path to the file where the reference measurements (FEM outputs) are
                stored.
664     """
665 
666     MPD = 0  #mean percentage difference
667     MAPD = 0 #mean absolute percentage difference
668     RMSD = 0 #root mean square difference
669     n = max(np.shape(reference_var))
670 
671     error = np.zeros(np.shape(reference_var))
672 
673     error = (calculated_var-reference_var) #simplle difference
674     RMSD = np.sqrt(np.sum(error*error)/n)
675 
676     a = error*100
677     b = reference_var
678     error = np.divide(a, b, out=np.zeros_like(a), where=b!=0) #substitute nan by 0. PD obtained
679     MPD = np.sum(error)/n
680     MAPD = np.sum(np.absolute(error))/n
681 
682     return RMSD, MPD, MAPD
683 
684 def SEA_interpolation_error(quads, strain_elements, U_SEA, strain_gauge):
685     """
686     Function for calculating the iterpolation error of SEA.
687 
688     Args:
689         quads (list): List containing quad elements.
690         strain_elements (dict): Dictionary containing arrays of the elements where strain is
                recorded for each strain component. Eg. for strain exx we know whic elemebnts record
                strain. The keys are "exx", "eyy" and "exy".
691         U_SEA (array): Array of size (N_nodes*4,1) containing the [s,sx,sy,sz] results of the SEA
                analysis in the format [s1,sx1,sy1,sz1,s2,sx2,sy2,sz2..]
692 
```

```
    """
    SEA_strain_lst = []
    exp_e_lst = []

    for quad in quads:
        if quad.eid in strain_elements:
            #Calculate N,L,M
            quad.update_nat_coord()
            x_loc = quad.probe.x_nat
            N, L, M =  NLM_matrices(xi=0, eta=0, #we calculate at centroid the strains
                                    x1=x_loc[0,0],
                                    y1= x_loc[1,0],
                                    x2= x_loc[3,0],
                                    y2= x_loc[4,0],
                                    x3= x_loc[6,0],
                                    y3= x_loc[7,0],
                                    x4= x_loc[9,0],
                                    y4= x_loc[10,0])

            #Calculate the SES strain
            DOF=4 #the DOF that are used in the U_SEA files
            ind_nodes = np.array([quad.n1,quad.n2,quad.n3,quad.n4])-1
            SEA_strain = np.matmul(N,U_SEA[ind_nodes*DOF]) \
                        - np.matmul(L,U_SEA[ind_nodes*DOF+1]) \
                        - np.matmul(M,U_SEA[ind_nodes*DOF+2])

            exp_e = strain_gauge[strain_gauge[:,0]==quad.eid,1]

            #Save values
            SEA_strain_lst.append(float(SEA_strain))
            exp_e_lst.append(float(exp_e))

    RMSD, MPD, MAPD = errors(np.array(SEA_strain_lst),np.array(exp_e_lst))

    return RMSD, MPD, MAPD
```

Listing C.1: *helpers.py*

```
from pyife3d import IQS4, IQS4Probe
import numpy as np
from pyife3d.helpers import compute_local_matrices
from sympy import var
from functools import partial
from pyife3d.iqs4_equations import B_matrices
import pandas as pd

def iFEM(N_elements,element_nodes,node_coord, strain_gauge_top, strain_gauge_bot, strain_elements,
    h, Gauss_points_weights,w_fact, isotropic, mat_direction, location):
    """
    Function for creating and updating all the elements in the mesh per iFEM algorithm.

    Args:
        N_elements (int): Number of elements in the model
        element_nodes (array): Array of size (N_elements,5) containing the nodes of each element
            in the format: element ID | node 1 ID | node 2 ID | node 3 ID | node 4 ID
        node_coord (array): Array of size (N_nodes,4) containing the node coordinates of the mesh
            in the format: ID | X | Y | Z.
        strain_gauge_top (array): Array of size (N_elements,4) storing the strain corresponding to
            each element in the following format element ID | exx top | eyy top | gxy top
        strain_gauge_bot (array): Array of size (N_elements,4) storing the strain corresponding to
            each element in the following format element ID | exx bot | eyy bot | gxz bot
        h (float): Half thickness of the plate.
        Gauss_points_weights (list): List of lists (can be converted to numpy array) where the
```

```
            first column is the evaluation point x and the second column is the associated weight
            for the sum
        w_fact (float): The weight factor to apply in IFEM when the strain measurement is missing.
        isotropic (bool): Describes whether the used material is sotropic or not.
        mat_direction (array): Numpy array of size (3,1) describing the material coordinate system.
       location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"

    Returns:
        quads (list): List of quad elements.
        probes (list): List of corresponding probe elements.
    """

    probes = []
    quads = []


    for i in range(0,N_elements):
        #Initialize the element
        probe = IQS4Probe()
        quad = IQS4(probe)

        #Idenitfy the nodes
        n1, n2, n3, n4 = element_nodes[i,1:]

        #Save the nodes for the element
        quad.eid = i+1 #ID of the element
        quad.n1 = n1
        quad.n2 = n2
        quad.n3 = n3
        quad.n4 = n4

        #Coordinates of the points
        r1 = np.reshape(np.array(node_coord[n1-1,1:]),(3,1))
        r2 = np.reshape(np.array(node_coord[n2-1,1:]),(3,1))
        r3 = np.reshape(np.array(node_coord[n3-1,1:]),(3,1))
        r4 = np.reshape(np.array(node_coord[n4-1,1:]),(3,1))

        #Save the coordinates
        probe.xe[0:3] = r1
        probe.xe[3:6] = r2
        probe.xe[6:9] = r3
        probe.xe[9:12] = r4

        #Obtain natural cooidnates
        quad.update_nat_coord()
        x_loc = quad.probe.x_nat

        #Create a B matrix for each elemenet by filling in all parameters except xi and eta
            (natural coordinates)
        B_matrices_partial = partial(B_matrices,x1=x_loc[0,0],
            y1= x_loc[1,0],
            x2= x_loc[3,0],
            y2= x_loc[4,0],
            x3= x_loc[6,0],
            y3= x_loc[7,0],
            x4= x_loc[9,0],
            y4= x_loc[10,0])

        # Initializing arrays of experimental strain and curavture of an element at its midplane
        exp_e = np.zeros((3,1))
        exp_k = np.zeros((3,1))

        #Weights for least-squares variational principle
```

```
        # exx | eyy | gxy | kxx | kyy | kxy | gxz | gyz
        w = w_fact*np.ones((8))

        #Start changing weights and exp_e and k only if any of the components is registered
        if quad.eid in strain_elements["exx"] or quad.eid in strain_elements["eyy"] or quad.eid in
             strain_elements["exy"]:

        #Accounting for aniisotropic material:
            if not isotropic:
                mat_direction_loc = np.matmul(quad.Te,mat_direction) #we align our local coord
                     system with the material direction
                theta = np.arccos(np.dot(mat_direction_loc[0:2].flatten(), np.array([1, 0])) /
                     (np.linalg.norm([1, 0]) * np.linalg.norm(mat_direction_loc[0:2]))) #in plane
                     angle between global x axis and material direction

                theta = abs(theta)

                if mat_direction_loc[1] >=0:
                    theta = -theta #positive rotation
                else:
                    theta = theta #negative rotation

                T_mat = np.array([
                    [np.cos(theta)**2, np.sin(theta)**2, np.sin(theta) * np.cos(theta)],
                    [np.sin(theta)**2, np.cos(theta)**2, -np.sin(theta) * np.cos(theta)],
                    [-2 * np.sin(theta) * np.cos(theta), 2 * np.sin(theta) * np.cos(theta),
                         np.cos(theta)**2 - np.sin(theta)**2]
                ])

            else:
                T_mat = np.eye(3)

            #Aligning with material
            strain_gauge_top_mat = np.matmul(T_mat,strain_gauge_top[i,1:]) #start from index 1
                 cause 0 is element ID
            strain_gauge_bot_mat = np.matmul(T_mat,strain_gauge_bot[i,1:])


            if location == "top":
                exp_e = strain_gauge_top_mat
            elif location == "mid":
                exp_e = 1/2*(strain_gauge_top_mat+strain_gauge_bot_mat)
                exp_k = 1/(2*h)*(strain_gauge_top_mat-strain_gauge_bot_mat)
            else:
                exp_e = strain_gauge_bot_mat

            #Save for plotting
            quad.probe.epsilontop = strain_gauge_top_mat
            quad.probe.epsilonbot = strain_gauge_bot_mat

            #Change indices appropiately
            if quad.eid in strain_elements["exx"] and location=="mid":
                w[0] = 1
                w[3] = 1
            elif quad.eid in strain_elements["exx"]:
                w[0] = 1

            if quad.eid in strain_elements["eyy"] and location=="mid":
                w[1] = 1
                w[4] = 1
            elif quad.eid in strain_elements["eyy"]:
                w[1] = 1

```

```
            if quad.eid in strain_elements["exy"] and location=="mid":
                w[2] = 1
                w[5] = 1
            elif quad.eid in strain_elements["exy"]:
                w[2] = 1

        quad = compute_local_matrices(B_funct_partial=B_matrices_partial, w=w,
            Gauss_points_weights=Gauss_points_weights,h=h,quad=quad,exp_e=exp_e,exp_k=exp_k)

        quads.append(quad)
        probes.append(probe)

    return quads, probes
```

Listing C.2: *iFEM_main.py*

```

from pyife3d.iqs4SEA import IQS4SEA, IQS4ProbeSEA
from pyife3d import IQS4, IQS4Probe
import numpy as np
from pyife3d.iqs4_equations import matrices_SEA, B_matrices, NLM_matrices
from pyife3d.helpers import compute_local_matrices, compute_local_matrices_extrapolation,
    exp_strain_builder
from functools import partial

def strain_extrapolation(alfaSEA, betaSEA, drllingfact,
    N_elements,element_nodes,node_coord,strain_gauge, strain_elements, Gauss_points_weights):
    """
    Function for creating and updating all the SEA elements ofr the strain extrapolation.

    Args:
        alfaSEA (float): Alfa factor of SEA strain extrapolation.
        betaSEA (float): Beta factor of SEA strain extrapolation.
        drllingfact (float): Drilling degree of freedom assumed factor.
        N_elements (int): Number of elements in the model
        element_nodes (array): Array of size (N_elements,5) containing the nodes of each element
            in the format: element ID | node 1 ID | node 2 ID | node 3 ID | node 4 ID
        node_coord (array): Array of size (N_nodes,4) containing the node coordinates of the mesh
            in the format: ID | X | Y | Z.
        strain_gauge (array): Array of size (N_elements,2) containing strain values of the
            selected component with format of ELEMENT ID | strain measurements
        strain_elements (list): List containing the indices of the strain elements for which the
            strain component is smeasured.
        Gauss_points_weights (list): List of lists (can be converted to numpy array) where the
            first column is the evaluation point x and the second column is the associated weight
            for the sum
    Returns:
        quads (list): List of quad elements.
        probes (list): List of corresponding probe elements.
    """
    probes = []
    quads = []

    for i in range(0,N_elements):
        probe = IQS4ProbeSEA()
        quad = IQS4SEA(probe)

        #Idenitfy the nodes
        n1, n2, n3, n4 = element_nodes[i,1:]

        #Save the nodes for the element
        # quad.eid = i+1 #ID of the element
        quad.eid = element_nodes[i,0]
```

```
40          quad.n1 = n1
41          quad.n2 = n2
42          quad.n3 = n3
43          quad.n4 = n4
44
45          #Coordinates of the points
46          r1 = np.reshape(np.array(node_coord[n1-1,1:]),(3,1))
47          r2 = np.reshape(np.array(node_coord[n2-1,1:]),(3,1))
48          r3 = np.reshape(np.array(node_coord[n3-1,1:]),(3,1))
49          r4 = np.reshape(np.array(node_coord[n4-1,1:]),(3,1))
50
51          #Save the coordinates
52          probe.xe[0:3] = r1
53          probe.xe[3:6] = r2
54          probe.xe[6:9] = r3
55          probe.xe[9:12] = r4
56
57          #Obtain natural cooidnates
58          quad.update_nat_coord()
59          x_loc = quad.probe.x_nat
60
61          matrices_SEA_partial = partial(matrices_SEA,x1=x_loc[0,0],
62              y1= x_loc[1,0],
63              x2= x_loc[3,0],
64              y2= x_loc[4,0],
65              x3= x_loc[6,0],
66              y3= x_loc[7,0],
67              x4= x_loc[9,0],
68              y4= x_loc[10,0])
69
70          quad = compute_local_matrices_extrapolation(alfaSEA=alfaSEA, betaSEA=betaSEA,
                drllingfact=drllingfact,  B_funct_partial=matrices_SEA_partial,
                Gauss_points_weights=Gauss_points_weights,quad=quad,strain_gauge=strain_gauge,strain_elements=strain_el
71
72          quads.append(quad)
73          probes.append(probe)
74
75      return quads, probes
76
77  def iFEM_SEA(N_elements,element_nodes,node_coord, strain_gauge_top, strain_gauge_bot,
        strain_elements, h, Gauss_points_weights,w_fact, isotropic, mat_direction, SEA_U_dict_top,
        SEA_U_dict_bot, location):
78      """
79      Function for creating and updating all the elements in the mesh per iFEM SEA algorithm. To be
            run after the strain pre-extrapoaltion was conducted for all the relevant components.
80
81      Args:
82          N_elements (int): Number of elements in the model
83          element_nodes (array): Array of size (N_elements,5) containing the nodes of each element
                in the format: element ID | node 1 ID | node 2 ID | node 3 ID | node 4 ID
84          node_coord (array): Array of size (N_nodes,4) containing the node coordinates of the mesh
                in the format: ID | X | Y | Z.
85          strain_gauge_top (array): Array of size (N_elements,4) storing the strain corresponding to
                each element in the following format element ID | exx top | eyy top | gxy top
86          strain_gauge_bot (array): Array of size (N_elements,4) storing the strain corresponding to
                each element in the following format element ID | exx bot | eyy bot | gxz bot
87          strain_elements (dict): Dictionary containing arrays of the elements where strain is
                recorded for each strain component. Eg. for strain exx we know whic elemebnts record
                strain. The keys are "exx", "eyy" and "exy"
88          h (float): Half thickness of the plate.
89          Gauss_points_weights (list): List of lists (can be converted to numpy array) where the
                first column is the evaluation point x and the second column is the associated weight
                for the sum
```

```
        w_fact (float): The weight factor to apply in IFEM when the strain measurement is missing.
        isotropic (bool): Describes whether the used material is sotropic or not.
        mat_direction (array): Numpy array of size (3,1) describing the material coordinate system.
        SEA_U_dict_top (dict): Dictionary containing the U_SEA for the strain component "exx",
             "eyy", "exy
         SEA_U_dict_bot (dict): Dictionary containing the U_SEA for the strain component "exx",
              "eyy", "exy
        location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"

    Returns:
        quads (list): List of quad elements.
        probes (list): List of corresponding probe elements.
    """
    probes = []
    quads = []

    for i in range(0,N_elements):
        #Initialize the element
        probe = IQS4Probe()
        quad = IQS4(probe)

        #Idenitfy the nodes
        n1, n2, n3, n4 = element_nodes[i,1:]

        #Save the nodes for the element
        quad.eid = i+1 #ID of the element
        quad.n1 = n1
        quad.n2 = n2
        quad.n3 = n3
        quad.n4 = n4

        #Coordinates of the points
        r1 = np.reshape(np.array(node_coord[n1-1,1:]),(3,1))
        r2 = np.reshape(np.array(node_coord[n2-1,1:]),(3,1))
        r3 = np.reshape(np.array(node_coord[n3-1,1:]),(3,1))
        r4 = np.reshape(np.array(node_coord[n4-1,1:]),(3,1))

        #Save the coordinates
        probe.xe[0:3] = r1
        probe.xe[3:6] = r2
        probe.xe[6:9] = r3
        probe.xe[9:12] = r4

        #Obtain natural coordinates
        quad.update_nat_coord()
        x_loc = quad.probe.x_nat

        #Insert the natural coordinates in the calculation of the strain-displacement matrices
        B_matrices_partial = partial(B_matrices,x1=x_loc[0,0],
            y1= x_loc[1,0],
            x2= x_loc[3,0],
            y2= x_loc[4,0],
            x3= x_loc[6,0],
            y3= x_loc[7,0],
            x4= x_loc[9,0],
            y4= x_loc[10,0])

        # Initializing arrays of experimental strain and curavture of an element at its midplane
        exp_e = np.zeros((3,1))
        exp_k = np.zeros((3,1))

        #Weights for least-squares variational principle
        # exx | eyy | gxy | kxx | kyy | kxy | gxz | gyz
```

```
        w = w_fact*np.ones((8))

        #Accounting for aniisotropic material:
        if not isotropic:
            mat_direction_loc = np.matmul(quad.Te,mat_direction) #we align or local coord system with the material direction
            theta = np.arccos(np.dot(mat_direction_loc[0:2].flatten(), np.array([1, 0])) / (np.linalg.norm([1, 0]) * np.linalg.norm(mat_direction_loc[0:2])))
            theta = abs(theta)

            if mat_direction_loc[1] >=0:
                theta = -theta #positive rotation
            else:
                theta = theta #negative rotation

            T_mat = np.array([
                [np.cos(theta)**2, np.sin(theta)**2, np.sin(theta) * np.cos(theta)],
                [np.sin(theta)**2, np.cos(theta)**2, -np.sin(theta) * np.cos(theta)],
                [-2 * np.sin(theta) * np.cos(theta), 2 * np.sin(theta) * np.cos(theta), np.cos(theta)**2 - np.sin(theta)**2]
            ])

        else:
            T_mat = np.eye(3)

        #We also need to calculate the NLM matrices
        N, L, M =  NLM_matrices(xi=0, eta=0, #we calculate at centroid the strains
                                x1=x_loc[0,0],
                                y1= x_loc[1,0],
                                x2= x_loc[3,0],
                                y2= x_loc[4,0],
                                x3= x_loc[6,0],
                                y3= x_loc[7,0],
                                x4= x_loc[9,0],
                                y4= x_loc[10,0])

        #Get the experimental strains by going thorugh each component
        exp_e, exp_k, w, quad =
            exp_strain_builder(quad,strain_elements,w,location,T_mat,strain_gauge_top,strain_gauge_bot,i,SEA_U_dict_
            SEA_U_dict_bot, N, L, M, h)

        quad = compute_local_matrices(B_funct_partial=B_matrices_partial, w=w,
            Gauss_points_weights=Gauss_points_weights,h=h,quad=quad,exp_e=exp_e,exp_k=exp_k)

        quads.append(quad)
        probes.append(probe)

    return quads, probes
```

Listing C.3: *iFEM_main.py*

```

import numpy as np
import sympy
from sympy import simplify, integrate, Matrix, diff
from sympy.vector import CoordSys3D, cross
import dill
import os

r"""

    4 ____ 3
     /   /
```

```
    /   /   positive normal in CCW
   /___/
   1    2

"""

DOF = 6
num_nodes = 4

save=0

sympy.var('h', positive=True, real=True)
sympy.var('x1, y1, x2, y2, x3, y3, x4, y4', real=True, positive=True)
sympy.var('rho, xi, eta, A, alphat')
sympy.var('A11, A12, A16, A22, A26, A66')
sympy.var('B11, B12, B16, B22, B26, B66')
sympy.var('D11, D12, D16, D22, D26, D66')
sympy.var('E44, E45, E55')

ONE = sympy.Integer(1)

R = CoordSys3D('R')
r1 = x1*R.i + y1*R.j
r2 = x2*R.i + y2*R.j
r3 = x3*R.i + y3*R.j
r4 = x4*R.i + y4*R.j

rbottom = r1 + (r2 - r1)*(xi + 1)/2
rtop = r4 + (r3 - r4)*(xi + 1)/2
r = rbottom + (rtop - rbottom)*(eta + 1)/2
xfunc = r.components[R.i]
yfunc = r.components[R.j]

# Jacobian theory
# http://kis.tu.kielce.pl/mo/COLORADO_FEM/colorado/IFEM.Ch17.pdf
# https://quickfem.com/theory/finite-element-analysis/

#
J = Matrix([[xfunc.diff(xi),  yfunc.diff(xi)],
            [xfunc.diff(eta), yfunc.diff(eta)]])

#Derivatives of Jacobian determinant wrt each coordinate of the quad element node
detJ = J.det().simplify()

#Invert that Jacobian
j = J.inv()

#Get the Jacobian terms for easier referral
j11 = j[0, 0].simplify()
j12 = j[0, 1].simplify()
j21 = j[1, 0].simplify()
j22 = j[1, 1].simplify()

#N shape functions
N1 = (eta * xi - eta - xi + 1) / 4
N2 = -(eta * xi + eta - xi - 1) / 4
N3 = (eta * xi + eta + xi + 1) / 4
N4 = -(eta * xi - eta + xi - 1) / 4

N5 = (1 - pow(xi, 2)) * (1 - eta) / 16
N6 = (1 + xi) * (1 - pow(eta, 2)) / 16
N7 = (1 - pow(xi, 2)) * (1 + eta) / 16
N8 = (1 - xi) * (1 - pow(eta, 2)) / 16
```

```
76
77  # xij and yij explained
78  x12 = x1 - x2
79  x23 = x2 - x3
80  x34 = x3 - x4
81  x41 = x4 - x1
82
83  y14 = y1 - y4
84  y21 = y2 - y1
85  y32 = y3 - y2
86  y43 = y4 - y3
87
88  # L shape functions
89  L1 = y14 * N8 - y21 * N5
90  L2 = y21 * N5 - y32 * N6
91  L3 = y32 * N6 - y43 * N7
92  L4 = y43 * N7 - y14 * N8
93
94
95  #M shape functions
96  M1 = x41*N8 - x12*N5
97  M2 = x12*N5 - x23*N6
98  M3 = x23*N6 - x34*N7
99  M4 = x34*N7 - x41*N8
100
101 #Derivatives wrt to xi
102 N1xi = N1.diff(xi)
103 N2xi = N2.diff(xi)
104 N3xi = N3.diff(xi)
105 N4xi = N4.diff(xi)
106 N5xi = N5.diff(xi)
107 N6xi = N6.diff(xi)
108 N7xi = N7.diff(xi)
109 N8xi = N8.diff(xi)
110
111 L1xi = L1.diff(xi)
112 L2xi = L2.diff(xi)
113 L3xi = L3.diff(xi)
114 L4xi = L4.diff(xi)
115
116 M1xi = M1.diff(xi)
117 M2xi = M2.diff(xi)
118 M3xi = M3.diff(xi)
119 M4xi = M4.diff(xi)
120
121 #Derivatives wrt to etas
122 N1eta = N1.diff(eta)
123 N2eta = N2.diff(eta)
124 N3eta = N3.diff(eta)
125 N4eta = N4.diff(eta)
126 N5eta = N5.diff(eta)
127 N6eta = N6.diff(eta)
128 N7eta = N7.diff(eta)
129 N8eta = N8.diff(eta)
130
131 L1eta = L1.diff(eta)
132 L2eta = L2.diff(eta)
133 L3eta = L3.diff(eta)
134 L4eta = L4.diff(eta)
135
136 M1eta = M1.diff(eta)
137 M2eta = M2.diff(eta)
138 M3eta = M3.diff(eta)
```

```
M4eta = M4.diff(eta)

#N derivatives wrt to x
N1x = j11*N1xi + j12*N1eta
N2x = j11*N2xi + j12*N2eta
N3x = j11*N3xi + j12*N3eta
N4x = j11*N4xi + j12*N4eta
N5x = j11*N5xi + j12*N5eta
N6x = j11*N6xi + j12*N6eta
N7x = j11*N7xi + j12*N7eta
N8x = j11*N8xi + j12*N8eta

#L derivatives wrt x
L1x = j11*L1xi + j12*L1eta
L2x = j11*L2xi + j12*L2eta
L3x = j11*L3xi + j12*L3eta
L4x = j11*L4xi + j12*L4eta

#M derivatives wrt x
M1x = j11*M1xi + j12*M1eta
M2x = j11*M2xi + j12*M2eta
M3x = j11*M3xi + j12*M3eta
M4x = j11*M4xi + j12*M4eta

#N derivatives wrt y
N1y = j21*N1xi + j22*N1eta
N2y = j21*N2xi + j22*N2eta
N3y = j21*N3xi + j22*N3eta
N4y = j21*N4xi + j22*N4eta
N5y = j21*N5xi + j22*N5eta
N6y = j21*N6xi + j22*N6eta
N7y = j21*N7xi + j22*N7eta
N8y = j21*N8xi + j22*N8eta

#L derivatives wrt y
L1y = j21*L1xi + j22*L1eta
L2y = j21*L2xi + j22*L2eta
L3y = j21*L3xi + j22*L3eta
L4y = j21*L4xi + j22*L4eta

#M derivatives wrt y
M1y = j21*M1xi + j22*M1eta
M2y = j21*M2xi + j22*M2eta
M3y = j21*M3xi + j22*M3eta
M4y = j21*M4xi + j22*M4eta

detJfunc = detJ

#exx = u,x = (dxi/dx)*u,xi + (deta/dx)*u,eta = j11 u,xi + j12 u,eta
Bmexx = Matrix([[N1x, 0, 0, 0, 0, L1x, N2x, 0, 0, 0, 0, L2x, N3x, 0, 0, 0, 0, L3x, N4x, 0, 0, 0,
    0, L4x]])
Bbexx = Matrix([[0, 0, 0, 0, N1x, 0, 0, 0, 0, 0, N2x, 0, 0, 0, 0, 0, N3x, 0, 0, 0, 0, 0, N4x, 0]])

#eyy = v,y = (dxi/dy)*v,xi + (deta/dy)*v,eta = j21 v,xi + j22 v,eta
Bmeyy = Matrix([[0, N1y, 0, 0, 0, M1y, 0, N2y, 0, 0, 0, M2y, 0, N3y, 0, 0, 0, M3y, 0, N4y, 0, 0,
    0, M4y]])
Bbeyy = Matrix([[0, 0, 0, -N1y, 0, 0, 0, 0, 0,-N2y, 0, 0, 0, 0, 0, -N3y, 0, 0, 0, 0, 0, -N4y, 0,
    0]])

#gxy = u,y + v,x = (dxi/dy)*u,xi + (deta/dy)*u,eta + (dxi/dx)*v,xi + (deta/dy)*v,eta
Bmgxy = Matrix([[N1y, N1x, 0, 0, 0, L1y+M1x, N2y, N2x, 0, 0, 0, L2y+M2x, N3y, N3x, 0, 0, 0,
    L3y+M3x, N4y, N4x, 0, 0, 0, L4y+M4x]])
Bbgxy = Matrix([[0, 0, 0, -N1x, N1y, 0, 0, 0, 0,-N2x, N2y, 0, 0, 0, 0, -N3x, N3y, 0, 0, 0, 0,
```

```
    -N4x, N4y, 0]])

Bm = Matrix([Bmexx, Bmeyy, Bmgxy])
Bb = Matrix([Bbexx, Bbeyy, Bbgxy])

#gxz
Bsgxz =  Matrix([[0, 0, N1x, -L1x, -M1x+N1, 0, 0, 0, N2x, -L2x, -M2x+N2, 0, 0, 0, N3x, -L3x,
    -M3x+N3, 0, 0, 0, N4x, -L4x, -M4x+N4, 0]])

#gxz
Bsgyz =  Matrix([[0, 0, N1y, -L1y-N1, -M1y, 0, 0, 0, N2y, -L2y-N2, -M2y, 0, 0, 0, N3y, -L3y-N3,
    -M3y, 0, 0, 0, N4y, -L4y-N4, -M4y, 0]])

    Bs = Matrix([Bsgxz, Bsgyz])

if save == 1:
    #Pickling all of our sympy variables
    absolute_path = os.path.dirname(os.path.dirname(__file__)) #needs to be applied two times to
        get to git directory
    pickle_folder = absolute_path + "\sympy_pickled_iqs4"

    #List of variables for which we do not need to create the pickled files
    var_exception = ["__builtins__",
    "__annotations__",
    "__cached__",
    "__doc__",
    "__file__",
    "__loader__",
    "__name__",
    "__package__",
    "__spec__",
    "absolute_path",
    "alphat",
    "CoordSys3D",
    "cross",
    "integrate",
    "np",
    "pickle_folder",
    "os",
    "simplify",
    " sympy"]

    for name in dir():
        if name not in var_exception:
            try:
                with open(pickle_folder+f"\sp_{name}.pkl",'wb') as file:
                    dill.dump(obj=eval(name),file=file)
            except:
                print("Pickling not working for variable ", name)

    print("SAVED!")

print("J",J)
print("-----------------------------------------------------")
print("j",j)
print("-----------------------------------------------------")
print("detJ",detJ)
print("-----------------------------------------------------")
print("N1x",N1x)
print("-----------------------------------------------------")
print("N2x",N2x)
print("-----------------------------------------------------")
print("N3x",N3x)
```

```
257 print("-------------------------------------------------")
258 print("N4x",N4x)
259 print("-------------------------------------------------")
260 print("N1y",N1y)
261 print("-------------------------------------------------")
262 print("N2y",N2y)
263 print("-------------------------------------------------")
264 print("N3y",N3y)
265 print("-------------------------------------------------")
266 print("N4y",N4y)
267 print("-------------------------------------------------")
268 print("-------------------------------------------------")
269 print("L1",L1)
270 print("-------------------------------------------------")
271 print("L2",L2)
272 print("-------------------------------------------------")
273 print("L3",L3)
274 print("-------------------------------------------------")
275 print("L4",L4)
276 print("-------------------------------------------------")
277 print("L1x",L1x)
278 print("-------------------------------------------------")
279 print("L2x",L2x)
280 print("-------------------------------------------------")
281 print("L3x",L3x)
282 print("-------------------------------------------------")
283 print("L4x",L4x)
284 print("-------------------------------------------------")
285 print("L1y",L1y)
286 print("-------------------------------------------------")
287 print("L2y",L2y)
288 print("-------------------------------------------------")
289 print("L3y",L3y)
290 print("-------------------------------------------------")
291 print("L4y",L4y)
292 print("-------------------------------------------------")
293 print("M1",M1)
294 print("-------------------------------------------------")
295 print("M2",M2)
296 print("-------------------------------------------------")
297 print("M3",M3)
298 print("-------------------------------------------------")
299 print("M4",M4)
300 print("-------------------------------------------------")
301 print("M1x",M1x)
302 print("-------------------------------------------------")
303 print("M2x",M2x)
304 print("-------------------------------------------------")
305 print("M3x",M3x)
306 print("-------------------------------------------------")
307 print("M4x",M4x)
308 print("-------------------------------------------------")
309 print("M1y",M1y)
310 print("-------------------------------------------------")
311 print("M2y",M2y)
312 print("-------------------------------------------------")
313 print("M3y",M3y)
314 print("-------------------------------------------------")
315 print("M4y",M4y)
316 print("-------------------------------------------------")
```

Listing C.4: $IQS4\_derivation.py$ adapted from [Castro, 2023]

```
"""
File containing the iQS4 equations hardcoded rather than reading from sympy. Dne for improving
    code speed
"""
import numpy as np

def funct_J(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return np.array([[-x1/2 + x2/2 + (eta + 1)*(x1/2 - x2/2 + x3/2 - x4/2)/2, -y1/2 + y2/2 + (eta
        + 1)*(y1/2 - y2/2 + y3/2 - y4/2)/2], [-x1/2 + x4/2 - (-x1 + x2)*(xi + 1)/4 + (x3 -
        x4)*(xi + 1)/4, -y1/2 + y4/2 - (xi + 1)*(-y1 + y2)/4 + (xi + 1)*(y3 - y4)/4]])

def funct_j(xi, eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return np.array ([[(-2*xi*y1 + 2*xi*y2 - 2*xi*y3 + 2*xi*y4 + 2*y1 + 2*y2 - 2*y3 -
        2*y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2
        + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 -
        x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3), (2*eta*y1 -
        2*eta*y2 + 2*eta*y3 - 2*eta*y4 - 2*y1 + 2*y2 + 2*y3 - 2*y4)/(eta*x1*y2 - eta*x1*y3 -
        eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 -
        x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 +
        x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)], [(2*x1*xi - 2*x1 - 2*x2*xi - 2*x2
        + 2*x3*xi + 2*x3 - 2*x4*xi + 2*x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3), (-2*eta*x1 + 2*eta*x2 - 2*eta*x3 + 2*eta*x4 + 2*x1 - 2*x2 -
        2*x3 + 2*x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 -
        eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1
        - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)]])

def funct_detJ(xi, eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return -eta*x1*y2/8 + eta*x1*y3/8 + eta*x2*y1/8 - eta*x2*y4/8 - eta*x3*y1/8 + eta*x3*y4/8 +
        eta*x4*y2/8 - eta*x4*y3/8 - x1*xi*y3/8 + x1*xi*y4/8 + x1*y2/8 - x1*y4/8 + x2*xi*y3/8 -
        x2*xi*y4/8 - x2*y1/8 + x2*y3/8 + x3*xi*y1/8 - x3*xi*y2/8 - x3*y2/8 + x3*y4/8 - x4*xi*y1/8
        + x4*xi*y2/8 + x4*y1/8 - x4*y3/8

def funct_N1(xi, eta):
    return (eta * xi - eta - xi + 1) / 4

def funct_N2(xi, eta):
    return -(eta * xi + eta - xi - 1) / 4

def funct_N3(xi, eta):
    return  (eta * xi + eta + xi + 1) / 4

def funct_N4(xi, eta):
    return -(eta * xi - eta + xi - 1) / 4

def funct_L1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return -(1 - eta)*(1 - xi**2)*(-y1 + y2)/16 + (1 - eta**2)*(1 - xi)*(y1 - y4)/16

def funct_L2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return (1 - eta)*(1 - xi**2)*(-y1 + y2)/16 - (1 - eta**2)*(xi + 1)*(-y2 + y3)/16

def funct_L3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return (1 - eta**2)*(xi + 1)*(-y2 + y3)/16 - (1 - xi**2)*(eta + 1)*(-y3 + y4)/16

def funct_L4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return -(1 - eta**2)*(1 - xi)*(y1 - y4)/16 + (1 - xi**2)*(eta + 1)*(-y3 + y4)/16

def funct_M1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return -(1 - eta)*(1 - xi**2)*(x1 - x2)/16 + (1 - eta**2)*(1 - xi)*(-x1 + x4)/16

def funct_M2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
```

```
43      return (1 - eta)*(1 - xi**2)*(x1 - x2)/16 - (1 - eta**2)*(x2 - x3)*(xi + 1)/16
44
45  def funct_M3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
46      return (1 - eta**2)*(x2 - x3)*(xi + 1)/16 - (1 - xi**2)*(eta + 1)*(x3 - x4)/16
47
48  def funct_M4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
49      return-(1 - eta**2)*(1 - xi)*(-x1 + x4)/16 + (1 - xi**2)*(eta + 1)*(x3 - x4)/16
50
51  def funct_N1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
52      return 2*(eta/4 - 1/4)*(-xi*y1 + xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 -
            eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
            x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
            x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(xi/4 - 1/4)*(eta*y1
            - eta*y2 + eta*y3 - eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 +
            eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 +
            x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 +
            x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)
53
54  def funct_N2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
55      return 2*(1/4 - eta/4)*(-xi*y1 + xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 -
            eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
            x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
            x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(-xi/4 - 1/4)*(eta*y1
            - eta*y2 + eta*y3 - eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 +
            eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 +
            x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 +
            x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)
56
57  def funct_N3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
58      return 2*(eta/4 + 1/4)*(-xi*y1 + xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 -
            eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
            x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
            x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(xi/4 + 1/4)*(eta*y1
            - eta*y2 + eta*y3 - eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 +
            eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 +
            x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 +
            x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)
59
60  def funct_N4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
61      return 2*(1/4 - xi/4)*(eta*y1 - eta*y2 + eta*y3 - eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 -
            eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
            x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
            x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(-eta/4 -
            1/4)*(-xi*y1 + xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 -
            eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 -
            x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 +
            x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)
62
63  def funct_N1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
64      return 2*(eta/4 - 1/4)*(x1*xi - x1 - x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 -
            eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
            x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
            x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(xi/4 - 1/4)*(-eta*x1
            + eta*x2 - eta*x3 + eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 +
            eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 +
            x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 +
            x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)
65
66  def funct_N2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
67      return 2*(1/4 - eta/4)*(x1*xi - x1 - x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 -
            eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
            x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
            x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(-xi/4 -
```

```
        1/4)*(-eta*x1 + eta*x2 - eta*x3 + eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 -
        eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 -
        x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 +
        x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)

def funct_N3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta/4 + 1/4)*(x1*xi - x1 - x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 -
        eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
        x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
        x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(xi/4 + 1/4)*(-eta*x1
        + eta*x2 - eta*x3 + eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 +
        eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 +
        x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 +
        x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)

def funct_N4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(1/4 - xi/4)*(-eta*x1 + eta*x2 - eta*x3 + eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 -
        eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 +
        x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 +
        x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3) + 2*(-eta/4 - 1/4)*(x1*xi
        - x1 - x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 +
        eta*x2*y4 + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 +
        x1*y4 - x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 +
        x4*xi*y1 - x4*xi*y2 - x4*y1 + x4*y3)

def funct_L1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(1 - xi)*(y1 - y4)/8 + (1 - xi**2)*(-y1 + y2)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(1 - eta)*(-y1 + y2)/8 - (1 - eta**2)*(y1 - y4)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(xi + 1)*(-y2 + y3)/8 - (1 - xi**2)*(-y1 + y2)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(-xi*(1 - eta)*(-y1 + y2)/8 - (1 - eta**2)*(-y2 + y3)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(xi + 1)*(-y2 + y3)/8 - (1 - xi**2)*(-y3 + y4)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(eta + 1)*(-y3 + y4)/8 + (1 - eta**2)*(-y2 + y3)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(1 - xi)*(y1 - y4)/8 + (1 - xi**2)*(-y3 + y4)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
```

```
        x4*y1 + x4*y3) + 2*(-xi*(eta + 1)*(-y3 + y4)/8 + (1 - eta**2)*(y1 - y4)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(1 - xi)*(y1 - y4)/8 + (1 - xi**2)*(-y1 + y2)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(1 - eta)*(-y1 + y2)/8 - (1 - eta**2)*(y1 - y4)/16)*(x1*xi - x1 -
        x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(xi + 1)*(-y2 + y3)/8 - (1 - xi**2)*(-y1 + y2)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(-xi*(1 - eta)*(-y1 + y2)/8 - (1 - eta**2)*(-y2 + y3)/16)*(x1*xi - x1
        - x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(xi + 1)*(-y2 + y3)/8 - (1 - xi**2)*(-y3 + y4)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(eta + 1)*(-y3 + y4)/8 + (1 - eta**2)*(-y2 + y3)/16)*(x1*xi - x1 -
        x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_L4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(1 - xi)*(y1 - y4)/8 + (1 - xi**2)*(-y3 + y4)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(-xi*(eta + 1)*(-y3 + y4)/8 + (1 - eta**2)*(y1 - y4)/16)*(x1*xi - x1 -
        x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(1 - xi)*(-x1 + x4)/8 + (1 - xi**2)*(x1 - x2)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(1 - eta)*(x1 - x2)/8 - (1 - eta**2)*(-x1 + x4)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(x2 - x3)*(xi + 1)/8 - (1 - xi**2)*(x1 - x2)/16)*(eta*y1 - eta*y2 + eta*y3 -
```

```
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(-xi*(1 - eta)*(x1 - x2)/8 - (1 - eta**2)*(x2 - x3)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(x2 - x3)*(xi + 1)/8 - (1 - xi**2)*(x3 - x4)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(eta + 1)*(x3 - x4)/8 + (1 - eta**2)*(x2 - x3)/16)*(-xi*y1 + xi*y2
        - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(1 - xi)*(-x1 + x4)/8 + (1 - xi**2)*(x3 - x4)/16)*(eta*y1 - eta*y2 + eta*y3 -
        eta*y4 - y1 + y2 + y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(-xi*(eta + 1)*(x3 - x4)/8 + (1 - eta**2)*(-x1 + x4)/16)*(-xi*y1 +
        xi*y2 - xi*y3 + xi*y4 + y1 + y2 - y3 - y4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4
        + eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(1 - xi)*(-x1 + x4)/8 + (1 - xi**2)*(x1 - x2)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(1 - eta)*(x1 - x2)/8 - (1 - eta**2)*(-x1 + x4)/16)*(x1*xi - x1 -
        x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(eta*(x2 - x3)*(xi + 1)/8 - (1 - xi**2)*(x1 - x2)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(-xi*(1 - eta)*(x1 - x2)/8 - (1 - eta**2)*(x2 - x3)/16)*(x1*xi - x1 -
        x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)

def funct_M3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
    return 2*(-eta*(x2 - x3)*(xi + 1)/8 - (1 - xi**2)*(x3 - x4)/16)*(-eta*x1 + eta*x2 - eta*x3 +
        eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
        eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
        x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
        x4*y1 + x4*y3) + 2*(xi*(eta + 1)*(x3 - x4)/8 + (1 - eta**2)*(x2 - x3)/16)*(x1*xi - x1 -
        x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
        eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
        x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
        x4*xi*y2 - x4*y1 + x4*y3)
```

```
119
120 def funct_M4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
121     return 2*(eta*(1 - xi)*(-x1 + x4)/8 + (1 - xi**2)*(x3 - x4)/16)*(-eta*x1 + eta*x2 - eta*x3 +
            eta*x4 + x1 - x2 - x3 + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 + eta*x3*y1 -
            eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 - x2*xi*y3 +
            x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 - x4*xi*y2 -
            x4*y1 + x4*y3) + 2*(-xi*(eta + 1)*(x3 - x4)/8 + (1 - eta**2)*(-x1 + x4)/16)*(x1*xi - x1 -
            x2*xi - x2 + x3*xi + x3 - x4*xi + x4)/(eta*x1*y2 - eta*x1*y3 - eta*x2*y1 + eta*x2*y4 +
            eta*x3*y1 - eta*x3*y4 - eta*x4*y2 + eta*x4*y3 + x1*xi*y3 - x1*xi*y4 - x1*y2 + x1*y4 -
            x2*xi*y3 + x2*xi*y4 + x2*y1 - x2*y3 - x3*xi*y1 + x3*xi*y2 + x3*y2 - x3*y4 + x4*xi*y1 -
            x4*xi*y2 - x4*y1 + x4*y3)
122
123 def B_matrices(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
124     detJ = funct_detJ(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
125
126     N1 = funct_N1(xi,eta)
127     N2 = funct_N2(xi,eta)
128     N3 = funct_N3(xi,eta)
129     N4 = funct_N4(xi,eta)
130     #-----------------------------------------------------------
131     N1x = funct_N1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
132     N2x = funct_N2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
133     N3x = funct_N3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
134     N4x = funct_N4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
135
136     N1y = funct_N1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
137     N2y = funct_N2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
138     N3y = funct_N3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
139     N4y = funct_N4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
140     #-----------------------------------------------------------
141     L1x = funct_L1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
142     L2x = funct_L2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
143     L3x = funct_L3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
144     L4x = funct_L4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
145
146     L1y = funct_L1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
147     L2y = funct_L2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
148     L3y = funct_L3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
149     L4y = funct_L4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
150     #-----------------------------------------------------------
151     M1x = funct_M1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
152     M2x = funct_M2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
153     M3x = funct_M3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
154     M4x = funct_M4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
155
156     M1y = funct_M1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
157     M2y = funct_M2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
158     M3y = funct_M3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
159     M4y = funct_M4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
160     #-----------------------------------------------------------
161
162     #Bm assembly
163     Bmexx = np.array([[N1x, 0, 0, 0, 0, L1x, N2x, 0, 0, 0, 0, L2x, N3x, 0, 0, 0, 0, L3x, N4x, 0,
            0, 0, 0, L4x]])
164     Bmeyy = np.array([[0, N1y, 0, 0, 0, M1y, 0, N2y, 0, 0, 0, M2y, 0, N3y, 0, 0, 0, M3y, 0, N4y,
            0, 0, 0, M4y]]) #TODO: comapring with kefal aper swithced from Mix
165     Bmgxy = np.array([[N1y, N1x, 0, 0, 0, L1y+M1x, N2y, N2x, 0, 0, 0, L2y+M2x, N3y, N3x, 0, 0, 0,
            L3y+M3x, N4y, N4x, 0, 0, 0, L4y+M4x]])
166
167     Bm = np.vstack((Bmexx,Bmeyy,Bmgxy))
168     #-----------------------------------------------------------
169
170     #Bb assembly
```

```
171     Bbexx = np.array([[0, 0, 0, 0, N1x, 0, 0, 0, 0, 0, N2x, 0, 0, 0, 0, 0, N3x, 0, 0, 0, 0, 0,
            N4x, 0]])
172     Bbeyy = np.array([[0, 0, 0, -N1y, 0, 0, 0, 0, 0,-N2y, 0, 0, 0, 0, 0, -N3y, 0, 0, 0, 0, 0,
            -N4y, 0, 0]])
173     Bbgxy = np.array([[0, 0, 0, -N1x, N1y, 0, 0, 0, 0,-N2x, N2y, 0, 0, 0, 0, -N3x, N3y, 0, 0, 0,
            0, -N4x, N4y, 0]])
174
175     Bb = np.vstack((Bbexx,Bbeyy,Bbgxy))
176     #-------------------------------------------------------------
177
178     #Bs assembly
179     Bsgxz =  np.array([[0, 0, N1x, -L1x, -M1x+N1, 0, 0, 0, N2x, -L2x, -M2x+N2, 0, 0, 0, N3x, -L3x,
            -M3x+N3, 0, 0, 0, N4x, -L4x, -M4x+N4, 0]])
180
181     #gxz
182     Bsgyz =  np.array([[0, 0, N1y, -L1y-N1, -M1y, 0, 0, 0, N2y, -L2y-N2, -M2y, 0, 0, 0, N3y,
            -L3y-N3, -M3y, 0, 0, 0, N4y, -L4y-N4, -M4y, 0]])
183
184     Bs = np.vstack((Bsgxz, Bsgyz))
185
186
187     return detJ, Bm, Bb, Bs
188
189 def PAOLO_matrices_SEA(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
190     """
191     Function for generating the matrices required for the SEA implementation of th einverse
            element IQS4.
192
193     Args:
194         xi (_type_): _description_
195         eta (_type_): _description_
196         x1 (_type_): _description_
197         x2 (_type_): _description_
198         x3 (_type_): _description_
199         x4 (_type_): _description_
200         y1 (_type_): _description_
201         y2 (_type_): _description_
202         y3 (_type_): _description_
203         y4 (_type_): _description_
204
205     Returns:
206         _float_: detJ determinannt of the Jacobian
207         _array_: N_tilde, Kalfa, Kbeta Arrays of sizes (12,12). 4 nodes and 3 DOF's of the inverse
                element.
208     """
209     detJ = funct_detJ(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
210
211     N1 = funct_N1(xi,eta)
212     N2 = funct_N2(xi,eta)
213     N3 = funct_N3(xi,eta)
214     N4 = funct_N4(xi,eta)
215     #-------------------------------------------------------------
216     L1 = funct_L1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
217     L2 = funct_L2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
218     L3 = funct_L3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
219     L4 = funct_L4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
220     #-------------------------------------------------------------
221     M1 = funct_M1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
222     M2 = funct_M2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
223     M3 = funct_M3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
224     M4 = funct_M4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
225     #-------------------------------------------------------------
226     N1x = funct_N1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
```

```
227     N2x = funct_N2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
228     N3x = funct_N3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
229     N4x = funct_N4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
230 
231     N1y = funct_N1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
232     N2y = funct_N2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
233     N3y = funct_N3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
234     N4y = funct_N4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
235     #-------------------------------------------------------
236     L1x = funct_L1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
237     L2x = funct_L2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
238     L3x = funct_L3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
239     L4x = funct_L4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
240 
241     L1y = funct_L1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
242     L2y = funct_L2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
243     L3y = funct_L3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
244     L4y = funct_L4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
245 
246     #-------------------------------------------------------
247     M1x = funct_M1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
248     M2x = funct_M2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
249     M3x = funct_M3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
250     M4x = funct_M4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
251 
252     M1y = funct_M1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
253     M2y = funct_M2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
254     M3y = funct_M3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
255     M4y = funct_M4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
256 
257     #-------------------------------------------------------
258 
259     #N tilde assembly
260     N = np.array([[N1, N2, N3, N4]])
261     L = np.array([[L1, L2, L3, L4]])
262     M = np.array([[M1, M2, M3, M4]])
263 
264     N_tilde = np.hstack((N,-L,-M))
265     #-------------------------------------------------------------
266 
267     #Kalfa components
268     Nx = np.array([[N1x, N2x, N3x, N4x]])
269     Lx = np.array([[L1x, L2x, L3x, L4x]])
270     Mx = np.array([[M1x, M2x, M3x, M4x]])
271 
272     Ny = np.array([[N1y, N2y, N3y, N4y]])
273     Ly = np.array([[L1y, L2y, L3y, L4y]])
274     My = np.array([[M1y, M2y, M3y, M4y]])
275 
276     #TODO: this is in Paolo's coordinate system ffs
277     Kalfa_B1 = np.hstack((Nx,-Lx,-Mx+N))
278     Kalfa_B2 = np.hstack((Ny,-Ly-N,-My))
279     Kalfa = np.matmul(np.transpose(Kalfa_B1),Kalfa_B1) + np.matmul(np.transpose(Kalfa_B2),Kalfa_B2)
280 
281     #Kbeta components
282     Ny = np.array([[N1y, N2y, N3y, N4y]])
283 
284     Kbeta = np.zeros((12,12))
285     Kbeta[4:8,4:8] = 3/2*np.outer(np.transpose(Ny),Ny)
286     Kbeta[4:8,8:] = 1/2*np.outer(np.transpose(Ny),Nx)
287     Kbeta[8:,4:8] = 1/2*np.outer(np.transpose(Ny),Nx)
288     Kbeta[8:,8:] = 3/2*np.outer(np.transpose(Nx),Nx)
289 
```

```
290     return detJ, N_tilde, Kalfa, Kbeta
291 
292 def matrices_SEA(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
293     """
294     Function for generating the matrices required for the SEA implementation of th einverse
            element IQS4.
295 
296     Args:
297         xi (_type_): _description_
298         eta (_type_): _description_
299         x1 (_type_): _description_
300         x2 (_type_): _description_
301         x3 (_type_): _description_
302         x4 (_type_): _description_
303         y1 (_type_): _description_
304         y2 (_type_): _description_
305         y3 (_type_): _description_
306         y4 (_type_): _description_
307 
308     Returns:
309         _float_: detJ determinannt of the Jacobian
310         _array_: N_tilde (12x1), Kalfa(12x12), Kbeta(12x12), Kalfa_B1(12x1) , Kalfa_B2(12x1) for 4
                nodes and 3DOF
311     """
312     detJ = funct_detJ(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
313 
314     N1 = funct_N1(xi,eta)
315     N2 = funct_N2(xi,eta)
316     N3 = funct_N3(xi,eta)
317     N4 = funct_N4(xi,eta)
318     #-----------------------------------------------------------
319     L1 = funct_L1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
320     L2 = funct_L2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
321     L3 = funct_L3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
322     L4 = funct_L4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
323     #-----------------------------------------------------------
324     M1 = funct_M1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
325     M2 = funct_M2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
326     M3 = funct_M3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
327     M4 = funct_M4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
328     #-----------------------------------------------------------
329     N1x = funct_N1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
330     N2x = funct_N2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
331     N3x = funct_N3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
332     N4x = funct_N4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
333 
334     N1y = funct_N1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
335     N2y = funct_N2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
336     N3y = funct_N3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
337     N4y = funct_N4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
338     #-----------------------------------------------------------
339     L1x = funct_L1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
340     L2x = funct_L2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
341     L3x = funct_L3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
342     L4x = funct_L4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
343 
344     L1y = funct_L1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
345     L2y = funct_L2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
346     L3y = funct_L3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
347     L4y = funct_L4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
348 
349     #-----------------------------------------------------------
350     M1x = funct_M1x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
```

```
    M2x = funct_M2x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M3x = funct_M3x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M4x = funct_M4x(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)

    M1y = funct_M1y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M2y = funct_M2y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M3y = funct_M3y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M4y = funct_M4y(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)

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

    #N tilde assembly
    N_tilde1 = np.array([[N1, -L1, -M1]])
    N_tilde2 = np.array([[N2, -L2, -M2]])
    N_tilde3 = np.array([[N3, -L3, -M3]])
    N_tilde4 = np.array([[N4, -L4, -M4]])

    N_tilde = np.hstack((N_tilde1,N_tilde2,N_tilde3,N_tilde4))
    #-------------------------------------------------------------

    #Kalfa components

    Kalfa_B1_1 = np.array([[N1x, -L1x, -M1x+N1]])
    Kalfa_B1_2 = np.array([[N2x, -L2x, -M2x+N2]])
    Kalfa_B1_3 = np.array([[N3x, -L3x, -M3x+N3]])
    Kalfa_B1_4 = np.array([[N4x, -L4x, -M4x+N4]])
    Kalfa_B1 = np.hstack((Kalfa_B1_1,Kalfa_B1_2,Kalfa_B1_3,Kalfa_B1_4))

    Kalfa_B2_1 = np.array([[N1y, -L1y-N1, -M1y]])
    Kalfa_B2_2 = np.array([[N2y, -L2y-N2, -M2y]])
    Kalfa_B2_3 = np.array([[N3y, -L3y-N3, -M3y]])
    Kalfa_B2_4 = np.array([[N4y, -L4y-N4, -M4y]])
    Kalfa_B2 = np.hstack((Kalfa_B2_1,Kalfa_B2_2,Kalfa_B2_3,Kalfa_B2_4))

    Kalfa = np.matmul(np.transpose(Kalfa_B1),Kalfa_B1) + np.matmul(np.transpose(Kalfa_B2),Kalfa_B2)

    #Kbeta components
    Nx = np.array([[N1x, N2x, N3x, N4x]])
    Ny = np.array([[N1y, N2y, N3y, N4y]])

    Kbeta = np.zeros((8,8))
    Kbeta[0:4,0:4] = 3/2*np.outer(np.transpose(Ny),Ny)
    Kbeta[0:4,4:] = 1/2*np.outer(np.transpose(Ny),Nx)
    Kbeta[4:,0:4] = 1/2*np.outer(np.transpose(Ny),Nx)
    Kbeta[4:,4:] = 3/2*np.outer(np.transpose(Nx),Nx)

    Kbeta1 = np.zeros((8,8))
    ind_dict = {"0":0,"1":4,"2":1,"3":5,"4":2,"5":6,"6":3,"7":7}
    for row in range(0,8): #TODO:Very ugly way of doing it.try to fix it
        for col in range(0,8):
            old_row=ind_dict[str(row)]
            old_col=ind_dict[str(col)]
            Kbeta1[row,col] = Kbeta[old_row,old_col]

    Kbeta = Kbeta1
    DOF=3 #before adding artifical drilling
    for i in range(0,4):
        Kbeta = np.insert(Kbeta,i*DOF,0,axis=0)
        Kbeta = np.insert(Kbeta,i*DOF,0,axis=1)

    return detJ, N_tilde, Kalfa, Kbeta, Kalfa_B1, Kalfa_B2

def NLM_matrices(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4):
```

```
    N1 = funct_N1(xi,eta)
    N2 = funct_N2(xi,eta)
    N3 = funct_N3(xi,eta)
    N4 = funct_N4(xi,eta)
    #-----------------------------------------------------------
    L1 = funct_L1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    L2 = funct_L2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    L3 = funct_L3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    L4 = funct_L4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    #-----------------------------------------------------------
    M1 = funct_M1(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M2 = funct_M2(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M3 = funct_M3(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)
    M4 = funct_M4(xi,eta,x1,x2,x3,x4,y1,y2,y3,y4)

    N = np.array([[N1, N2, N3, N4]])
    L = np.array([[L1, L2, L3, L4]])
    M = np.array([[M1, M2, M3, M4]])

    return N, L, M
```

Listing C.5: *iqs4_equations.py*

```
#cython: boundscheck=False
#cython: wraparound=False
#cython: cdivision=True
#cython: nonecheck=False
#cython: overflowcheck=False
#cython: embedsignature=True
#cython: infer_types=False
r"""
IQS4 - Inverse quadrilateral element with reduced integration (:mod:`pyife3d.iqs4`)
================================================================================

.. currentmodule:: pyife3d.iqs4

"""
from libc.math cimport fabs

import numpy as np

from .shellprop cimport ShellProp

cdef int DOF = 6
cdef int NUM_NODES = 4

cdef class IQS4Probe:
    r"""
    Probe used for local coordinates, local displacements, local stresses etc

    Attributes
    ----------
    xe, : array-like
        Array of size ``NUM_NODES*3=12`` containing the nodal coordinates
        global coordinate system, in the following order `{x_e}_1,
        {y_e}_1, {z_e}_1, `{x_e}_2, {y_e}_2, {z_e}_2`, `{x_e}_3, {y_e}_3,
        {z_e}_3`, `{x_e}_4, {y_e}_4, {z_e}_4`.
    ue, : array-like
        Array of size ``NUM_NODES*DOF`` containing the element displacements
        in the following order `{s}_1, {s_x}_1, {s_y}_1, {s_z}_1,
        {s}_2, {s_x}_2, {s_y}_2, {s_z}_2,
        {s}_3, {s_x}_3, {s_y}_3, {s_z}_3,
        {s}_4, {s_x}_4, {s_y}_4, {s_z}_4`.
```

```
    epsilon, : array-like
        Array of the size 8 containing the element strains.
        The strains are defined based on the developer file of quad4r and are defined in the
            following order
        `e_{xx} e_{yy} g_{xy} k_{xx} k_{yy} k_{zz} g_{yz} g_{xz}`
    epsilontopSEA/epsilonbotSEA, : array-like
        Array of the size 3 containing the element strains for the top and botoom surfaces
            respectively.
        The strains are defined based on the developer file of quad4r and are defined in the
            following order
        `e_{xx} e_{yy} g_{xy}`
    centroid, : array-like
        Array contains the coordinates of the midpoint of the element in the following order
        `x_{c} y_{c} z_{c}`
    x_nat, : array-like
        Array of size ``NUM_NODES*3=12`` containing the nodal coordinates
        natural coordinate system, in the following order `{x_e}_1,
        {y_e}_1, {z_e}_1, `{x_e}_2, {y_e}_2, {z_e}_2`, `{x_e}_3, {y_e}_3,
        {z_e}_3`, `{x_e}_4, {y_e}_4, {z_e}_4`.

    """
    cdef public xe
    cdef public ue
    cdef public epsilon
    cdef public epsilontop, epsilontopSEA
    cdef public epsilonbot, epsilonbotSEA
    cdef public centroid
    cdef public stresses
    cdef public x_nat


    def __cinit__(IQS4Probe self):
        self.xe = np.zeros((NUM_NODES*3,1), dtype=np.float64)
        self.ue = np.zeros((NUM_NODES*DOF,1), dtype=np.float64)
        self.epsilon = np.zeros((8,1), dtype=np.float64)
        self.epsilontop = np.zeros((3,1), dtype=np.float64)
        self.epsilontopSEA = np.zeros((3,1), dtype=np.float64)
        self.epsilonbot = np.zeros((3,1), dtype=np.float64)
        self.epsilonbotSEA = np.zeros((3,1), dtype=np.float64)
        self.centroid = np.zeros((3,1), dtype=np.float64)
        self.stresses = np.zeros((3,1), dtype=np.float64)
        self.x_nat = np.zeros((NUM_NODES*3,1), dtype=np.float64)

cdef class IQS4:
    r"""
    Nodal connectivity for the plate element similar to Nastran's IQS4::

         ^ y
         |
        4 ________ 3
         |        |
         |        |   --> x
         |        |
         |________|
        1         2

    The element coordinate system is determined identically what is explained
    in Nastran's quick reference guide for the CQUAD4 element, as illustrated
    below.

    .. image:: ../figures/nastran_cquad4.svg

    Attributes
```

```
    ----------
    eid, : int
        Element identification number.
    area, : double
        Element area.
    alphat, : double
        Element drilling penalty factor for the plate drilling stiffness,
        defined according to Eq. 2.20 in the reference below. The default value
        of ``alphat = 1.`` comes from the same reference::

            Adam, A.E. Mohamed, A.E. Hassaballa, Degenerated Four Nodes Shell
            Element with Drilling Degree of Freedom, IOSR J. Eng. 3 (2013)
            10-20. www.iosrjen.org (accessed April 20, 2020).

        For those familiar with NASTRAN, ``alphat`` can be calculated based on
        NASTRAN's ``K6ROT`` parameters as ``alphat = 1.e-6*K6ROT``. The default
        value according to AUTODESK NASTRAN's quick reference guide is ``K6ROT
        = 100.`` for static analysis and ``K6ROT=1.e4`` for modal solutions.
        MSC NASTRAN's quick reference guide states that ``K6ROT > 100.`` should
        not be used, but this is controversion, already being controversial to
        what AUTODESK NASTRAN's manual says.
    r11, r12, r13, r21, r22, r23, r31, r32, r33 : double
        Rotation matrix to the global coordinate system.
    m11, m12, m21, m22 : double
        Rotation matrix only for the constitutive relations. Used when a
        material direction is used instead of the element local coordinates.
    c1, c2, c3, c4 : int
        Position of each node in the global stiffness matrix.
    n1, n2, n3, n4 : int
        Node identification number.
    init_k_KC0, init_k_KG, init_k_M : int
        Position in the arrays storing the sparse data for the structural
        matrices.
    probe, : :class:`.Quad4RProbe` object
        Pointer to the probe.

    """
    cdef public int eid
    cdef public int n1, n2, n3, n4
    cdef public int c1, c2, c3, c4
    cdef public int init_k_KC0, init_k_KG, init_k_M
    cdef public double area
    cdef public double r11, r12, r13, r21, r22, r23, r31, r32, r33
    cdef public double m11, m12, m21, m22
    cdef public IQS4Probe probe
    cdef public Tet, Te
    cdef public ke, fe

    def __cinit__(IQS4 self, IQS4Probe p):
        self.probe = p
        self.eid = -1
        self.n1 = -1
        self.n2 = -1
        self.n3 = -1
        self.n4 = -1
        self.c1 = -1
        self.c2 = -1
        self.c3 = -1
        self.c4 = -1
        self.area = 0
        self.r11 = self.r12 = self.r13 = 0.
        self.r21 = self.r22 = self.r23 = 0.
        self.r31 = self.r32 = self.r33 = 0.
```

```
164         self.m11 = 1.
165         self.m12 = 0.
166         self.m21 = 0.
167         self.m22 = 1.
168         self.Tet = np.zeros((3,3))
169         self.Te = np.zeros((3,3))
170         self.ke = np.zeros((DOF*NUM_NODES,DOF*NUM_NODES))
171         self.fe = np.zeros((DOF*NUM_NODES,1))
172 
173     cpdef void update_probe_ue(IQS4 self, double [::1] u):
174         r"""Update the local displacement vector of the probe of the element
175 
176         .. note:: The ``probe`` attribute object :class:`.Quad4RProbe` is
177                   updated, not the element object.
178 
179         Parameters
180         ----------
181         u : array-like
182             Array with global displacements, for a total of `M` nodes in
183             the model, this array will be arranged as: `u_1, v_1, w_1, {r_x}_1,
184             {r_y}_1, {r_z}_1, u_2, v_2, w_2, {r_x}_2, {r_y}_2, {r_z}_2, ...,
185             u_M, v_M, w_M, {r_x}_M, {r_y}_M, {r_z}_M`.
186 
187         """
188         cdef int i, j
189         cdef int c[4]
190         cdef double s1[3]
191         cdef double s2[3]
192         cdef double s3[3]
193 
194 
195         # positions in the global stiffness matrix
196         c[0] = self.c1
197         c[1] = self.c2
198         c[2] = self.c3
199         c[3] = self.c4
200 
201         # global to local transformation of displacements
202         s1[0] = self.r11
203         s1[1] = self.r21
204         s1[2] = self.r31
205         s2[0] = self.r12
206         s2[1] = self.r22
207         s2[2] = self.r32
208         s3[0] = self.r13
209         s3[1] = self.r23
210         s3[2] = self.r33
211 
212         for j in range(NUM_NODES):
213             for i in range(DOF):
214                 self.probe.ue[j*DOF + i] = 0
215 
216         for j in range(NUM_NODES):
217             for i in range(DOF//2):
218                 # transforming translations
219                 self.probe.ue[j*DOF + 0] += s1[i]*u[c[j] + 0 + i]
220                 self.probe.ue[j*DOF + 1] += s2[i]*u[c[j] + 0 + i]
221                 self.probe.ue[j*DOF + 2] += s3[i]*u[c[j] + 0 + i]
222                 # transforming rotations
223                 self.probe.ue[j*DOF + 3] += s1[i]*u[c[j] + 3 + i]
224                 self.probe.ue[j*DOF + 4] += s2[i]*u[c[j] + 3 + i]
225                 self.probe.ue[j*DOF + 5] += s3[i]*u[c[j] + 3 + i]
226 
```

```
227     cpdef void update_probe_xe(IQS4 self, double [::1] x):
228         r"""Update the 3D coordinates of the probe of the element
229 
230         .. note:: The ``probe`` attribute object :class:`.Quad4RProbe` is
231                   updated, not the element object.
232 
233         Parameters
234         ----------
235         x : array-like
236             Array with global nodal coordinates, for a total of `M` nodes in
237             the model, this array will be arranged as: `x_1, y_1, z_1, x_2,
238             y_2, z_2, ..., x_M, y_M, z_M`.
239 
240         """
241         cdef int i, j
242         cdef int c[4]
243         cdef double s1[3]
244         cdef double s2[3]
245         cdef double s3[3]
246 
247         # positions in the global stiffness matrix
248         c[0] = self.c1
249         c[1] = self.c2
250         c[2] = self.c3
251         c[3] = self.c4
252 
253         # global to local transformation of displacements
254         s1[0] = self.r11
255         s1[1] = self.r21
256         s1[2] = self.r31
257         s2[0] = self.r12
258         s2[1] = self.r22
259         s2[2] = self.r32
260         s3[0] = self.r13
261         s3[1] = self.r23
262         s3[2] = self.r33
263 
264         for j in range(NUM_NODES):
265             for i in range(DOF//2):
266                 self.probe.xe[j*DOF//2 + i] = 0
267 
268         for j in range(NUM_NODES):
269             for i in range(DOF//2):
270                 self.probe.xe[j*DOF//2 + 0] += s1[i]*x[c[j]//2 + i]
271                 self.probe.xe[j*DOF//2 + 1] += s2[i]*x[c[j]//2 + i]
272                 self.probe.xe[j*DOF//2 + 2] += s3[i]*x[c[j]//2 + i]
273 
274         self.update_area()
275         self.update_centroid()
276         self.update_T_matrix()
277         self.update_nat_coord()
278 
279     cpdef void update_area(IQS4 self):
280         r"""
281         Update element area
282 
283         """
284         cdef double x1, x2, x3, x4, y1, y2, y3, y4
285         # NOTE ignoring z in local coordinates
286         x1 = self.probe.xe[0]
287         y1 = self.probe.xe[1]
288         # z1 = self.probe.xe[2]
289         x2 = self.probe.xe[3]
```

```
290         y2 = self.probe.xe[4]
291         # z2 = self.probe.xe[5]
292         x3 = self.probe.xe[6]
293         y3 = self.probe.xe[7]
294         # z3 = self.probe.xe[8]
295         x4 = self.probe.xe[9]
296         y4 = self.probe.xe[10]
297         # z4 = self.probe.xe[11]
298         self.area = 1/2.*fabs((x1*y2 + x2*y3 + x3*y4 + x4*y1) - (x2*y1 + x3*y2 + x4*y3 + x1*y4))
299 
300     cpdef void update_centroid(IQS4 self):
301         self.probe.centroid[0] = np.sum(self.probe.xe[0::3])/4
302         self.probe.centroid[1] = np.sum(self.probe.xe[1::3])/4
303         self.probe.centroid[2] = np.sum(self.probe.xe[2::3])/4
304 
305     cpdef void update_T_matrix(IQS4 self):
306         r"""Update the rotation matrix of the element
307 
308         Attributes ``r11,r12,r13,r21,r22,r23,r31,r32,r33`` are updated,
309         corresponding to the rotation matrix from local to global coordinates.
310 
311         The element coordinate system is determined, identifying the `ijk`
312         components of each axis: `{x_e}_i, {x_e}_j, {x_e}_k`; `{y_e}_i,
313         {y_e}_j, {y_e}_k`; `{z_e}_i, {z_e}_j, {z_e}_k`.
314 
315         The rotation matrix terms are calculated after solving 9 equations.
316 
317         Parameters
318         ----------
319         x : array-like
320             Array with global nodal coordinates, for a total of `M` nodes in
321             the model, this array will be arranged as: `x_1, y_1, z_1, x_2,
322             y_2, z_2, ..., x_M, y_M, z_M`.
323 
324         """
325         cdef x
326         cdef double xi, xj, xk, yi, yj, yk, zi, zj, zk
327         cdef double x1i, x1j, x1k, x2i, x2j, x2k, x3i, x3j, x3k, x4i, x4j, x4k
328         cdef double v13i, v13j, v13k, v42i, v42j, v42k
329         cdef double tmp, xmatnorm, ymati, ymatj, ymatk
330         cdef double tol
331 
332         x = self.probe.xe #instead of feeding x separately we just open it up form our quad element
333         #The notation with x is very misleading. essentially for each vertix of the quad we
                establish the i,j,k which is just the x,y,z coordinate. it is more of a naming
                convention to show that we are going to vector operations
334 
335         x1i = x[0]
336         x1j = x[1]
337         x1k = x[2]
338         x2i = x[3]
339         x2j = x[4]
340         x2k = x[5]
341         x3i = x[6]
342         x3j = x[7]
343         x3k = x[8]
344         x4i = x[9]
345         x4j = x[10]
346         x4k = x[11]
347 
348         #establishing the vectors of the two diagonals
349         v13i = x3i - x1i
350         v13j = x3j - x1j
```

```
351         v13k = x3k - x1k
352         v42i = x2i - x4i
353         v42j = x2j - x4j
354         v42k = x2k - x4k
355 
356         #Getting the normal vector coordinates
357         zi = v42j*v13k - v42k*v13j
358         zj = -v42i*v13k + v42k*v13i
359         zk = v42i*v13j - v42j*v13i
360 
361         #And normalizing it
362         tmp = (zi**2 + zj**2 + zk**2)**0.5
363         zi /= tmp
364         zj /= tmp
365         zk /= tmp
366         # NOTE defining tolerance to be 1/1e10 of normal vector norm
367         tol = tmp/1e10
368 
369         xi = (v13i + v42i)/2.
370         xj = (v13j + v42j)/2.
371         xk = (v13k + v42k)/2.
372         tmp = (xi**2 + xj**2 + xk**2)**0.5
373         xi /= tmp
374         xj /= tmp
375         xk /= tmp
376 
377         # y = z X x
378         yi = zj*xk - zk*xj
379         yj = zk*xi - zi*xk
380         yk = zi*xj - zj*xi
381         tmp = (yi**2 + yj**2 + yk**2)**0.5
382         yi /= tmp
383         yj /= tmp
384         yk /= tmp
385 
386         #rotation matrix attributes
387         self.r11 = xi
388         self.r21 = xj
389         self.r31 = xk
390         self.r12 = yi
391         self.r22 = yj
392         self.r32 = yk
393         self.r13 = zi
394         self.r23 = zj
395         self.r33 = zk
396 
397         self.Tet = np.array([[self.r11, self.r12, self.r13],
398                              [self.r21, self.r22, self.r23],
399                              [self.r31,self.r32, self.r33]])
400 
401         self.Te = np.transpose(self.Tet)
402 
403     cpdef void update_nat_coord(IQS4 self):
404         self.update_area()
405         self.update_centroid()
406         self.update_T_matrix()
407 
408         x_nat =
              np.matmul(self.Te,(np.transpose(np.reshape(self.probe.xe,(4,3)))-self.probe.centroid*np.ones((3,4))))
409         self.probe.x_nat = np.reshape(x_nat,(12,1),order="F")
```

Listing C.6: $iqs4.pyx$ adapted from [Castro, 2023]

```
#cython: boundscheck=False
#cython: wraparound=False
#cython: cdivision=True
#cython: nonecheck=False
#cython: overflowcheck=False
#cython: embedsignature=True
#cython: infer_types=False
r"""
IQS4 - Inverse quadrilateral element with reduced integration (:mod:`pyife3d.iqs4`)
===================================================================================

.. currentmodule:: pyife3d.iqs4

"""
from libc.math cimport fabs

import numpy as np

from .shellprop cimport ShellProp

cdef int DOF = 4 #we include the driling also
cdef int NUM_NODES = 4

cdef class IQS4ProbeSEA:
    r"""
    Probe used for local coordinates, local displacements, local stresses etc

    Attributes
    ----------
    xe, : array-like
        Array of size ``NUM_NODES*3=12`` containing the nodal coordinates
        global coordinate system, in the following order `{x_e}_1,
        {y_e}_1, {z_e}_1, `{x_e}_2, {y_e}_2, {z_e}_2`, `{x_e}_3, {y_e}_3,
        {z_e}_3`, `{x_e}_4, {y_e}_4, {z_e}_4`.
    ue, : array-like
        Array of size ``NUM_NODES*DOF`` containing the element displacements
        in the following order `{s}_1, {s_x}_1, {s_y}_1, {s_z}_1,
        {s}_2, {s_x}_2, {s_y}_2, {s_z}_2,
        {s}_3, {s_x}_3, {s_y}_3, {s_z}_3,
        {s}_4, {s_x}_4, {s_y}_4, {s_z}_4`.
    epsilon, : array-like
        Array of the size 8 containing the element strains.
        The strains are defined based on the developer file of quad4r and are defined in the
            following order
        `e_{xx} e_{yy} g_{xy} k_{xx} k_{yy} k_{zz} g_{yz} g_{xz}`
    epsilontopSEA/epsilonbotSEA, : array-like
        Array of the size 3 containing the element strains for the top and botoom surfaces
            respectively.
        The strains are defined based on the developer file of quad4r and are defined in the
            following order
        `e_{xx} e_{yy} g_{xy}`
    centroid, : array-like
        Array contains the coordinates of the midpoint of the element in the following order
        `x_{c} y_{c} z_{c}`
    x_nat, : array-like
        Array of size ``NUM_NODES*3=12`` containing the nodal coordinates
        natural coordinate system, in the following order `{x_e}_1,
        {y_e}_1, {z_e}_1, `{x_e}_2, {y_e}_2, {z_e}_2`, `{x_e}_3, {y_e}_3,
        {z_e}_3`, `{x_e}_4, {y_e}_4, {z_e}_4`.

    """
    cdef public xe
```

```
60      cdef public ue
61      cdef public epsilon
62      cdef public epsilontopSEA
63      cdef public epsilonbotSEA
64      cdef public centroid
65      cdef public stresses
66      cdef public x_nat
67
68
69      def __cinit__(IQS4ProbeSEA self):
70          self.xe = np.zeros((NUM_NODES*3,1), dtype=np.float64)
71          self.ue = np.zeros((NUM_NODES*DOF,1), dtype=np.float64)
72          self.epsilon = np.zeros((8,1), dtype=np.float64)
73          self.epsilontopSEA = np.zeros((3,1), dtype=np.float64)
74          self.epsilonbotSEA = np.zeros((3,1), dtype=np.float64)
75          self.centroid = np.zeros((3,1), dtype=np.float64)
76          self.stresses = np.zeros((3,1), dtype=np.float64)
77          self.x_nat = np.zeros((NUM_NODES*3,1), dtype=np.float64)
78
79  cdef class IQS4SEA:
80      #TODO: change documentation references
81      r"""
82      Nodal connectivity for the plate element similar to Nastran's IQS4::
83
84            ^ y
85            |
86          4 ________ 3
87            |        |
88            |        |   --> x
89            |        |
90            |________|
91          1          2
92
93      The element coordinate system is determined identically what is explained
94      in Nastran's quick reference guide for the CQUAD4 element, as illustrated
95      below.
96
97      .. image:: ../figures/nastran_cquad4.svg
98
99      Attributes
100     ----------
101     eid, : int
102         Element identification number.
103     area, : double
104         Element area.
105     alphat, : double
106         Element drilling penalty factor for the plate drilling stiffness,
107         defined according to Eq. 2.20 in the reference below. The default value
108         of ``alphat = 1.`` comes from the same reference::
109
110             Adam, A.E. Mohamed, A.E. Hassaballa, Degenerated Four Nodes Shell
111             Element with Drilling Degree of Freedom, IOSR J. Eng. 3 (2013)
112             10-20. www.iosrjen.org (accessed April 20, 2020).
113
114         For those familiar with NASTRAN, ``alphat`` can be calculated based on
115         NASTRAN's ``K6ROT`` parameters as ``alphat = 1.e-6*K6ROT``. The default
116         value according to AUTODESK NASTRAN's quick reference guide is ``K6ROT
117         = 100.`` for static analysis and ``K6ROT=1.e4`` for modal solutions.
118         MSC NASTRAN's quick reference guide states that ``K6ROT > 100.`` should
119         not be used, but this is controversion, already being controversial to
120         what AUTODESK NASTRAN's manual says.
121     r11, r12, r13, r21, r22, r23, r31, r32, r33 : double
122         Rotation matrix to the global coordinate system.
```

```
123     m11, m12, m21, m22 : double
124         Rotation matrix only for the constitutive relations. Used when a
125         material direction is used instead of the element local coordinates.
126     c1, c2, c3, c4 : int
127         Position of each node in the global stiffness matrix.
128     n1, n2, n3, n4 : int
129         Node identification number.
130     init_k_KC0, init_k_KG, init_k_M : int
131         Position in the arrays storing the sparse data for the structural
132         matrices.
133     probe, : :class:`.Quad4RProbe` object
134         Pointer to the probe.
135     #TODO: complete the docstring
136 
137     """
138     cdef public int eid
139     cdef public int n1, n2, n3, n4
140     cdef public int c1, c2, c3, c4
141     cdef public int init_k_KC0, init_k_KG, init_k_M
142     cdef public double area
143     # cdef public double alphat # drilling penalty factor for stiffness matrix, see Eq. 2.20 in
            F.M. Adam, A.E. Mohamed, A.E. Hassaballa, Degenerated Four Nodes Shell Element with
            Drilling Degree of Freedom, IOSR J. Eng. 3 (2013) 10-20. www.iosrjen.org (accessed April
            20, 2020).
144     cdef public double r11, r12, r13, r21, r22, r23, r31, r32, r33
145     cdef public double m11, m12, m21, m22
146     cdef public IQS4ProbeSEA probe
147     cdef public Tet, Te
148     cdef public ke, fe
149 
150     def __cinit__(IQS4SEA self, IQS4ProbeSEA p):
151         self.probe = p
152         self.eid = -1
153         self.n1 = -1
154         self.n2 = -1
155         self.n3 = -1
156         self.n4 = -1
157         self.c1 = -1
158         self.c2 = -1
159         self.c3 = -1
160         self.c4 = -1
161         self.area = 0
162         # self.alphat = 1. # based on recommended value of reference F.M. Adam, A.E. Mohamed, A.E.
                Hassaballa
163         self.r11 = self.r12 = self.r13 = 0.
164         self.r21 = self.r22 = self.r23 = 0.
165         self.r31 = self.r32 = self.r33 = 0.
166         self.m11 = 1.
167         self.m12 = 0.
168         self.m21 = 0.
169         self.m22 = 1.
170         self.Tet = np.zeros((3,3))
171         self.Te = np.zeros((3,3))
172         self.ke = np.zeros((DOF*NUM_NODES,DOF*NUM_NODES))
173         self.fe = np.zeros((DOF*NUM_NODES,1))
174 
175     cpdef void update_centroid(IQS4SEA self):
176         #Method from Shape Sensing of a Complex Aeronautical Structure with Inverse Finite Element
                Method
177         cdef list alpha, beta
178         cdef double c_nom
179         cdef c,d, c_den
180 
```

```
181         alpha = [1, 2, 3, 4]
182         beta = [2, 3, 4, 1]
183         c = np.zeros((4,3))
184         d = np.zeros((4,1))
185 
186         for i in range(0,4):
187 
188             d[i,0] =
                    np.linalg.norm(self.probe.xe[(alpha[i]-1)*3:alpha[i]*3]-self.probe.xe[(beta[i]-1)*3:beta[i]*3])
189             c[i,:] = (self.probe.xe[(alpha[i]-1)*3:alpha[i]*3,0] +
                    self.probe.xe[(beta[i]-1)*3:beta[i]*3,0])/2
190 
191         c_den = np.zeros((3,1))
192         c_nom = 0
193         for i in range(0,4):
194             c_den = c_den + np.reshape(c[i,:]*d[i,0],np.shape(c_den))
195             c_nom += d[i,0]
196 
197         self.probe.centroid[0] = (c_den/c_nom)[0,0]
198         self.probe.centroid[1] = (c_den/c_nom)[1,0]
199         self.probe.centroid[2] = (c_den/c_nom)[2,0]
200 
201     cpdef void update_T_matrix(IQS4SEA self):
202         r"""Update the rotation matrix of the element
203 
204         Attributes ``r11,r12,r13,r21,r22,r23,r31,r32,r33`` are updated,
205         corresponding to the rotation matrix from local to global coordinates.
206 
207         The element coordinate system is determined, identifying the `ijk`
208         components of each axis: `{x_e}_i, {x_e}_j, {x_e}_k`; `{y_e}_i,
209         {y_e}_j, {y_e}_k`; `{z_e}_i, {z_e}_j, {z_e}_k`.
210 
211         The rotation matrix terms are calculated after solving 9 equations.
212 
213         Parameters
214         ----------
215         x : array-like
216             Array with global nodal coordinates, for a total of `M` nodes in
217             the model, this array will be arranged as: `x_1, y_1, z_1, x_2,
218             y_2, z_2, ..., x_M, y_M, z_M`.
219 
220         """
221         cdef x
222         cdef double xi, xj, xk, yi, yj, yk, zi, zj, zk
223         cdef double x1i, x1j, x1k, x2i, x2j, x2k, x3i, x3j, x3k, x4i, x4j, x4k
224         cdef double v13i, v13j, v13k, v42i, v42j, v42k
225         cdef double tmp, xmatnorm, ymati, ymatj, ymatk
226         cdef double tol
227 
228         x = self.probe.xe #instead of feeding x separately we just open it up form our quad element
229         #The notation with x is very misleading. essentially for each vertix of the quad we
                establish the i,j,k which is just the x,y,z coordinate. it is more of a naming
                convention to show that we are going to vector operations
230 
231         x1i = x[0]
232         x1j = x[1]
233         x1k = x[2]
234         x2i = x[3]
235         x2j = x[4]
236         x2k = x[5]
237         x3i = x[6]
238         x3j = x[7]
239         x3k = x[8]
```

```
240         x4i = x[9]
241         x4j = x[10]
242         x4k = x[11]
243 
244         #establishing the vectors of the two diagonals
245         v13i = x3i - x1i
246         v13j = x3j - x1j
247         v13k = x3k - x1k
248         v42i = x2i - x4i
249         v42j = x2j - x4j
250         v42k = x2k - x4k
251 
252         #Getting the normal vector coordinates
253         zi = v42j*v13k - v42k*v13j
254         zj = -v42i*v13k + v42k*v13i
255         zk = v42i*v13j - v42j*v13i
256 
257         #And normalizing it
258         tmp = (zi**2 + zj**2 + zk**2)**0.5
259         zi /= tmp
260         zj /= tmp
261         zk /= tmp
262         # NOTE defining tolerance to be 1/1e10 of normal vector norm
263         tol = tmp/1e10
264 
265         xi = (v13i + v42i)/2.
266         xj = (v13j + v42j)/2.
267         xk = (v13k + v42k)/2.
268         tmp = (xi**2 + xj**2 + xk**2)**0.5
269         xi /= tmp
270         xj /= tmp
271         xk /= tmp
272 
273         # y = z X x
274         yi = zj*xk - zk*xj
275         yj = zk*xi - zi*xk
276         yk = zi*xj - zj*xi
277         tmp = (yi**2 + yj**2 + yk**2)**0.5
278         yi /= tmp
279         yj /= tmp
280         yk /= tmp
281 
282         #rotation matrix attributes
283         self.r11 = xi
284         self.r21 = xj
285         self.r31 = xk
286         self.r12 = yi
287         self.r22 = yj
288         self.r32 = yk
289         self.r13 = zi
290         self.r23 = zj
291         self.r33 = zk
292 
293         #Assembled rotation matrix
294         self.Tet = np.array([[self.r11, self.r12, self.r13],
295                              [self.r21, self.r22, self.r23],
296                              [self.r31,self.r32, self.r33]])
297 
298         self.Te = np.transpose(self.Tet)
299 
300     cpdef void update_area(IQS4SEA self):
301         cdef double x1, x2, x3, x4, y1, y2, y3, y4
302         x1 = self.probe.xe[0]
```

```
303         y1 = self.probe.xe[1]
304         x2 = self.probe.xe[3]
305         y2 = self.probe.xe[4]
306         x3 = self.probe.xe[6]
307         y3 = self.probe.xe[7]
308         x4 = self.probe.xe[9]
309         y4 = self.probe.xe[10]
310         self.area = 1/2*((x1*y2+x2*y3+x3*y4+x4*y1)-(x2*y1+x3*y2+x4*y3+x1*y4))
311 
312     cpdef void update_nat_coord(IQS4SEA self):
313         self.update_area()
314         self.update_centroid()
315         self.update_T_matrix()
316 
317         x_nat =
                np.matmul(self.Te,(np.transpose(np.reshape(self.probe.xe,(4,3)))-self.probe.centroid*np.ones((3,4))))
318         self.probe.x_nat = np.reshape(x_nat,(12,1),order="F")
```

Listing C.7: $iqs4SEA.pyx$ adapted from [Castro, 2023]

```
1  import numpy as np
2  import pyvista as pv
3  import matplotlib.pyplot as plt
4  from mpl_toolkits.mplot3d import Axes3D
5  from mpl_toolkits.mplot3d.art3d import Poly3DCollection
6  from matplotlib.colors import LinearSegmentedColormap
7  import pandas as pd
8  import matplotlib.patches as mpatches
9  import os
10 from matplotlib.patches import Polygon
11 
12 def undeformed_3D(quads, show_opt, save_opt, save_path):
13     """
14     Function for plotting the undeformed 3D structure.
15 
16     Args:
17         quads (list): List of quad objects
18         show_opt (bool): Determines if the figure is shown or not
19         save_opt (bool): Determines if the figure is saved or not
20         save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
21     """
22     plt.clf()
23 
24     #https://stackoverflow.com/questions/4622057/plotting-3d-polygons
25     fig = plt.figure()
26     ax = Axes3D(fig)
27     fig.add_axes(ax)
28 
29     xmin, xmax, ymin , ymax, zmin, zmax = (0, 0, 0, 0, 0, 0)
30 
31     for quad in quads:
32         x = quad.probe.xe[0::3].reshape(4).tolist()
33         y = quad.probe.xe[1::3].reshape(4).tolist()
34         z = quad.probe.xe[2::3].reshape(4).tolist()
35 
36         verts = [list(zip(x,y,z))]
37         ax.add_collection3d(Poly3DCollection(verts))
38 
39         if xmin >= min(x):
40             xmin = min(x)
41         if xmax <= max(x):
42             xmax = max(x)
43 
```

```
44          if ymin >= min(y):
45              ymin = min(y)
46          if ymax <= max(y):
47              ymax = max(y)
48
49          if zmin >= min(z):
50              zmin = min(z)
51          if zmax <= max(z):
52              zmax = max(z)
53
54      # ax.set_xlim([0.9*xmin, 1.1*xmax])
55      # ax.set_ylim([0.9*ymin, 1.1*ymax])
56      # ax.set_zlim([0.9*zmin, 1.1*zmax])
57
58      #Nicer for scaling
59      ax.set_xlim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
60      ax.set_ylim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
61      ax.set_zlim([0.9*zmin, 1.1*zmax])
62
63      if save_opt:
64          if not os.path.exists(save_path):
65              os.makedirs(save_path)
66          fig.savefig(save_path+f"\\undeformed.png")
67          plt.close(fig)
68
69      if show_opt:
70          plt.show()
71
72  def undeformed_3D_loc_coord(quads, show_opt, save_opt, save_path):
73      """
74      Function for plotting the undeformed 3D structure.
75
76      Args:
77          quads (list): List of quad objects
78          show_opt (bool): Determines if the figure is shown or not
79          save_opt (bool): Determines if the figure is saved or not
80          save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
81      """
82      plt.clf()
83
84      #https://stackoverflow.com/questions/4622057/plotting-3d-polygons
85      fig = plt.figure()
86      ax = Axes3D(fig)
87      fig.add_axes(ax)
88
89      xmin, xmax, ymin , ymax, zmin, zmax = (0, 0, 0, 0, 0, 0)
90
91      for quad in quads:
92          x = quad.probe.xe[0::3].reshape(4).tolist()
93          y = quad.probe.xe[1::3].reshape(4).tolist()
94          z = quad.probe.xe[2::3].reshape(4).tolist()
95
96          verts = [list(zip(x,y,z))]
97
98          collection = Poly3DCollection(verts)
99          collection.set_facecolor("#F8FFD2")
100
101         centroid = quad.probe.centroid
102         factor = 10
103
104         ax.plot3D([centroid[0],centroid[0]+quad.r11/factor],[centroid[1],centroid[1]+quad.r21/factor],[centroid[2],c
                #x
105         ax.plot3D([centroid[0],centroid[0]+quad.r12/factor],[centroid[1],centroid[1]+quad.r22/factor],[centroid[2],c
```

```
            #y
106         ax.plot3D([centroid[0],centroid[0]+quad.r13/factor],[centroid[1],centroid[1]+quad.r23/factor],[centroid[2],c
107
108         if xmin >= min(x):
109             xmin = min(x)
110         if xmax <= max(x):
111             xmax = max(x)
112
113         if ymin >= min(y):
114             ymin = min(y)
115         if ymax <= max(y):
116             ymax = max(y)
117
118         if zmin >= min(z):
119             zmin = min(z)
120         if zmax <= max(z):
121             zmax = max(z)
122
123     #Nicer for scaling in some cases
124     # ax.set_xlim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
125     # ax.set_ylim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
126     # ax.set_zlim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
127
128     ax.set_xlim(xmin, xmax)
129     ax.set_ylim(-2, 8)
130     ax.set_zlim(-4, 4)
131     ax.view_init(elev=32, azim=-132, roll=0)
132
133     if save_opt:
134         if not os.path.exists(save_path):
135             os.makedirs(save_path)
136         fig.savefig(save_path+f"\\undeformed.png")
137         plt.close(fig)
138
139     if show_opt:
140         plt.show()
141
142 def undeformed_3D_instrumented(quads, strain_elements, show_opt, save_opt, save_path):
143     """
144     Function for plotting the undeformed 3D structure in which the instrumented elements are
            highlighted.
145
146     Args:
147         quads (list): List of quad objects
148         strain_elements (dict): Dictionary containing arrays of the elements where strain is
                recorded for each strain component. Eg. for strain exx we know which elements record
                strain. The keys are "exx", "eyy" and "exy".
149         show_opt (bool): Determines if the figure is shown or not
150         save_opt (bool): Determines if the figure is saved or not
151         save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
152     """
153     plt.clf()
154
155     #https://stackoverflow.com/questions/4622057/plotting-3d-polygons
156     fig = plt.figure(figsize=(14, 8))
157     ax = Axes3D(fig)
158     fig.add_axes(ax)
159
160     color_map = {"inactive": "#d0cece",
161                  "exx": "#ffadad",
162                  "eyy": "#ffd6a5",
163                  "exy": "#fdffb6",
164                  "exxeyy": "#caffbf",
```

```
                "exxexy": "#9bf6ff",
                "eyyexy": "#a0c4ff",
                "exxeyyexy": "#bdb2ff"}

    xmin, xmax, ymin , ymax, zmin, zmax = (0, 0, 0, 0, 0, 0)
    for quad in quads:
        x = quad.probe.xe[0::3].reshape(4).tolist()
        y = quad.probe.xe[1::3].reshape(4).tolist()
        z = quad.probe.xe[2::3].reshape(4).tolist()

        verts = [list(zip(x,y,z))]
        element = Poly3DCollection(verts)

        #We start from the most restrictive conditions to the least restrictive
        #Depending on which strain components are insturmented, the element is highlighted in a
            specifc color
        if (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["eyy"]) and
            (quad.eid in strain_elements["exy"]):
            el_color = color_map["exxeyyexy"]
        elif (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["eyy"]):
            el_color = color_map["exxeyy"]
        elif (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["exy"]):
            el_color = color_map["exxexy"]
        elif (quad.eid in strain_elements["eyy"]) and (quad.eid in strain_elements["exy"]):
            el_color = color_map["eyyexy"]
        elif quad.eid in strain_elements["exx"]:
            el_color = color_map["exx"]
        elif quad.eid in strain_elements["eyy"]:
            el_color = color_map["eyy"]
        elif quad.eid in strain_elements["exy"]:
            el_color = color_map["exy"]
        else:
            el_color = color_map["inactive"]

        #Setting the appropriate color for the elements
        element.set_color(el_color)
        element.set_edgecolor('#000000')
        ax.add_collection3d(element)

        #Legend https://stackoverflow.com/questions/39500265/how-to-manually-create-a-legend
        inactive_patch = mpatches.Patch(color=color_map["inactive"]  , label='Not instrumented')
        exx_patch = mpatches.Patch(color=color_map["exx"], label='$\\epsilon_{xx}$')
        eyy_patch = mpatches.Patch(color=color_map["eyy"], label='$\\epsilon_{yy}$')
        exy_patch = mpatches.Patch(color=color_map["exy"], label='$\\gamma_{xy}$')
        exxeyy_patch = mpatches.Patch(color=color_map["exxeyy"], label='$\\epsilon_{xx},
            \\epsilon_{yy}$')
        exxexy_patch = mpatches.Patch(color=color_map["exxexy"], label='$\\epsilon_{xx},
            \\gamma_{xy}$')
        eyyexy_patch = mpatches.Patch(color=color_map["eyyexy"], label='$\\epsilon_{yy},
            \\gamma_{xy}$')
        exxeyyexy_patch = mpatches.Patch(color=color_map["exxeyyexy"], label='$\\epsilon_{xx},
            \\epsilon_{yy}, \\gamma_{xy}$')
        plt.legend(handles=[inactive_patch, exx_patch, eyy_patch, exy_patch, exxeyy_patch,
            exxexy_patch, eyyexy_patch,exxeyyexy_patch],loc='center left', bbox_to_anchor=(1.07,
            0.5))

        #Scaling purposes
        if xmin >= min(x):
            xmin = min(x)
        if xmax <= max(x):
            xmax = max(x)

        if ymin >= min(y):
```

```
220             ymin = min(y)
221         if ymax <= max(y):
222             ymax = max(y)
223 
224         if zmin >= min(z):
225             zmin = min(z)
226         if zmax <= max(z):
227             zmax = max(z)
228 
229     #Nicer for scaling
230     ax.set_xlim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
231     ax.set_ylim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
232     ax.set_zlim([0.9*zmin, 1.1*zmax])
233 
234     if save_opt:
235         if not os.path.exists(save_path):
236             os.makedirs(save_path)
237 
238         fig.savefig(save_path+f"\\instrumented.png")
239         #Add element nodes
240         if len(quads)<=100: #do this only if it is visible
241             for quad in quads:
242                     centroid_ar = np.asarray(quad.probe.centroid)
243                     ax.text(x=centroid_ar[0,0], y=centroid_ar[1,0],
                                z=centroid_ar[2,0],s=f"{quad.eid}",zorder=2*len(quads))
244                     fig.savefig(save_path+f"\\instrumented_elements.png")
245         else:
246             pass
247 
248     if show_opt:
249         plt.show()
250 
251 def deformed_3D(quads,U,DOF,show_opt, save_opt, save_path,location):
252     """
253     Function for plotting the deformed 3D structure.
254 
255     Args:
256         quads (list): List of quad objects
257         U (array): Array of size N_DOF,1 represesnting the containing the node displacements
258         in the following order `{u_e}_1, {v_e}_1, {w_e}_1, {{r_x}_e}_1, {{r_y}_e}_1, {{r_z}_e}_1`
                ...
259         DOF (int): number of DOF's per node
260         show_opt (bool): Determines if the figure is shown or not
261         save_opt (bool): Determines if the figure is saved or not
262         save_path (str): Determines where the figure is saved.
263         Required if save_opt is set to true.
264         location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
265     """
266     plt.clf()
267 
268     #https://stackoverflow.com/questions/4622057/plotting-3d-polygons
269     fig = plt.figure()
270     ax = Axes3D(fig)
271     fig.add_axes(ax)
272 
273     xmin, xmax, ymin , ymax, zmin, zmax = (0, 0, 0, 0, 0, 0)
274 
275     for quad in quads:
276         x = quad.probe.xe[0::3].reshape(4)
277         y = quad.probe.xe[1::3].reshape(4)
278         z = quad.probe.xe[2::3].reshape(4)
279 
280         u1 = np.array([U[(quad.n1-1)*DOF],
```

```
                        U[(quad.n2-1)*DOF],
                        U[(quad.n3-1)*DOF],
                        U[(quad.n4-1)*DOF]]).reshape(4)

        u2 = np.array([U[(quad.n1-1)*DOF+1],
                        U[(quad.n2-1)*DOF+1],
                        U[(quad.n3-1)*DOF+1],
                        U[(quad.n4-1)*DOF+1]]).reshape(4)

        u3 = np.array([U[(quad.n1-1)*DOF+2],
                        U[(quad.n2-1)*DOF+2],
                        U[(quad.n3-1)*DOF+2],
                        U[(quad.n4-1)*DOF+2]]).reshape(4)

        x_plt = x+u1.tolist()
        y_plt = y+u2.tolist()
        z_plt = z+u3.tolist()

        verts = [list(zip(x_plt,y_plt,z_plt))]
        ax.add_collection3d(Poly3DCollection(verts))

        if xmin >= min(x_plt):
            xmin = min(x_plt)
        if xmax <= max(x_plt):
            xmax = max(x_plt)

        if ymin >= min(y_plt):
            ymin = min(y_plt)
        if ymax <= max(y_plt):
            ymax = max(y_plt)

        if zmin >= min(z_plt):
            zmin = min(z_plt)
        if zmax <= max(z_plt):
            zmax = max(z_plt)

    # ax.set_zlim([0.9*zmin, 1.1*zmax])
    z_lim = max(abs(zmin),abs(zmax))

    #Nicer for scaling
    ax.set_xlim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
    ax.set_ylim([0.9*min(xmin,ymin), 1.1*max(xmax,ymax)])
    ax.set_zlim([-z_lim, z_lim])

    if save_opt:
        if not os.path.exists(save_path):
            os.makedirs(save_path)
        fig.savefig(save_path+f"\\deformed_{location}.png")
        plt.close(fig)

    if show_opt:
        plt.show()

def nodal_contour2D(node_coord, U, DOF, plot_var, show_opt, save_opt, save_path,location):
    """
    Function for plotting the 2D contour of one variable along the plate. It can plot the contour
        of T1, T2, T3 along the plate.

    Args:
        node_coord (array): Array of size (N_nodes,4) storing the coordinate of the nodes in the
            following format: ID | X | Y | Z
        U (array): Array of size N_DOF,1 represesnting the containing the node displacements
        in the following order `{u_e}_1, {v_e}_1, {w_e}_1, {{r_x}_e}_1, {{r_y}_e}_1, {{r_z}_e}_1`
```

```
            ...
        DOF (int): number of used DOF's
        plot_var (str): choose between T1, T2 and T3
        show_opt (bool): Determines if the figure is shown or not
        save_opt (bool): Determines if the figure is saved or not
        save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
        location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
    """
    plt.clf()
    x = node_coord[:,1]
    y = node_coord[:,2]

    if plot_var=="T1":
        z=np.reshape(U[0::DOF],np.shape(x))
    elif plot_var=="T2":
        z=np.reshape(U[1::DOF],np.shape(x))
    elif plot_var=="T3":
        z=np.reshape(U[2::DOF],np.shape(x))

    colors_from_img = np.load("pyife3d\supporting_files\colors_from_img.npy")
    N_entries,_ = np.shape(colors_from_img)
    my_cmap = LinearSegmentedColormap.from_list('my_cmap', colors_from_img, N=N_entries)

    fig = plt.figure(figsize=(12, 8))
    plt.tricontourf(x, y, z, levels=100,cmap=my_cmap)
    plt.xlabel("x[m]")
    plt.ylabel("y[m]")
    plt.title(f"Nodal contour: {plot_var} [m] at {location} plate")
    v = np.linspace(np.min(z), np.max(z), 9, endpoint=True)
    plt.colorbar(ticks=v)
    fig.tight_layout()
    plt.gca().set_aspect("equal")

    if save_opt:
        if not os.path.exists(save_path):
            os.makedirs(save_path)
        fig.savefig(save_path+f"\{plot_var}_{location}.png")
        plt.close(fig)

    if show_opt:
        plt.show()

def nodal_contour2D_Ushape(node_coord, U, DOF, plot_var, show_opt, save_opt, save_path,location):
    """
    Function for plotting the 2D contour of one variable along the plate. It can plot the contour
        of T1, T2, T3 along the plate.

    Args:
        node_coord (array): Array of size (N_nodes,4) storing the coordinate of the nodes in the
            following format: ID | X | Y | Z
        U (array): Array of size N_DOF,1 represesnting the containing the node displacements
        in the following order `{u_e}_1, {v_e}_1, {w_e}_1, {{r_x}_e}_1, {{r_y}_e}_1, {{r_z}_e}_1`
            ...
        DOF (int): number of used DOF's
        plot_var (str): choose between T1, T2, T3 and T total
        show_opt (bool): Determines if the figure is shown or not
        save_opt (bool): Determines if the figure is saved or not
        save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
        location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
    """
    plt.clf()

    #Creating subplots
```

```
401     colors_from_img = np.load("pyife3d\supporting_files\colors_from_img.npy")
402     N_entries,_ = np.shape(colors_from_img)
403     my_cmap = LinearSegmentedColormap.from_list('my_cmap', colors_from_img, N=N_entries)
404 
405     fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 8),layout="constrained")
406     #Select the correct displacement component
407     if plot_var=="T1":
408         U_cor= U[0::DOF]
409     elif plot_var=="T2":
410         U_cor= U[1::DOF]
411     elif plot_var=="T3":
412         U_cor= U[2::DOF]
413     elif plot_var=="T total":
414         U_cor = np.sqrt(np.power(U[0::DOF],2)+np.power(U[1::DOF],2)+np.power(U[2::DOF],2))
415 
416     #Bottom Plate
417     check = np.isclose(node_coord[:,3], 0.) #z is 0
418     ax1 = node_coord[check,1] #x coord
419     ax2 = node_coord[check,2] #y coord
420     z = np.reshape(U_cor[check],np.shape(ax1))
421 
422     bot_csf = axes[0].tricontourf(ax1, ax2, z,
            levels=100,cmap=my_cmap,vmin=np.min(U_cor),vmax=np.max(U_cor))
423     axes[0].set_title(f"Nodal contour: {plot_var} [m] Top View")
424     axes[0].set_xlabel("x[m]")
425     axes[0].set_ylabel("y[m]")
426 
427     total_z = z
428 
429     #Front Plate
430     check = np.isclose(node_coord[:,2], 0.) #y is 0
431     ax1 = node_coord[check,1] #x coord
432     ax2 = node_coord[check,3] #z coord
433     z = np.reshape(U_cor[check],np.shape(ax1))
434 
435     front_csf = axes[1].tricontourf(ax1, ax2, z,
            levels=100,cmap=my_cmap,vmin=np.min(U_cor),vmax=np.max(U_cor))
436     axes[1].set_title(f"Nodal contour: {plot_var} [m] Front View")
437     axes[1].set_xlabel("x[m]")
438     axes[1].set_ylabel("z[m]")
439 
440     total_z = np.append(total_z,z)
441 
442     #Back Plate
443 
444     check = np.isclose(node_coord[:,2], np.max(node_coord[:,2])) #y is max
445     ax1 = node_coord[check,1] #x coord
446     ax2 = node_coord[check,3] #z coord
447     z = np.reshape(U_cor[check],np.shape(ax1))
448 
449     back_csf = axes[2].tricontourf(ax1, ax2, z,
            levels=100,cmap=my_cmap,vmin=np.min(U_cor),vmax=np.max(U_cor))
450     axes[2].set_title(f"Nodal contour: {plot_var} [m] Back View")
451     axes[2].set_xlabel("x[m]")
452     axes[2].set_ylabel("z[m]")
453 
454     total_z = np.append(total_z,z)
455     z = total_z
456     v = np.linspace(np.min(z), np.max(z), 9, endpoint=True)
457 
458     complete_disp = plt.tricontourf(node_coord[:,1], node_coord[:,3],U_cor[:,0],
            levels=100,cmap=my_cmap,vmin=np.min(U_cor[:]),vmax=np.max(U_cor[:]))
459 
```

```
460     plt.colorbar(complete_disp,ax=axes,ticks=v)
461 
462     axes[0].set_aspect("equal")
463     axes[1].set_aspect("equal")
464     axes[2].set_aspect("equal")
465 
466     if save_opt:
467         if not os.path.exists(save_path):
468             os.makedirs(save_path)
469         fig.savefig(save_path+f"\{plot_var}_{location}_usection.png")
470         plt.close(fig)
471 
472     if show_opt:
473         plt.show()
474 
475 def perc_error(node_coord,calculated_var,N_nodes,strain_elements,reference_path,name_var,
     show_opt, save_opt, save_path,location):
476     """
477     Function for plotting the percentage error at each node in the FEM model. 2d view.
478     When the reference value is 0, to avoid Nan the error is eplaced by 0. Might not be a
          representative error handling in all cases
479 
480     Args:
481         node_coord (array): Array of size (N_nodes,4) storing the coordinate of the nodes in the
              following format: ID | X | Y | Z
482         calculated_var (array): Array of size (N_nodes,1) with the values of a calculated array
483         N_nodes (int): Number of nodes in the iFEM mesh
484         strain_elements (int): Dictionary containing arrays of the elements where strain is
              recorded for each strain component. Eg. for strain exx we know whic elemebnts record
              strain. The keys are "exx", "eyy" and "exy".
485         reference_path (str): Path to the file where the reference measurements (FEM outputs) are
              stored.
486         name_var (str): Name of the variable for which error is computed. Used for saving the value
487         show_opt (bool): Determines if the figure is shown or not
488         save_opt (bool): Determines if the figure is saved or not
489         save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
490         location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
491     """
492     plt.clf()
493 
494     MPD = 0  #mean percentage difference
495     MAPD = 0 #mean absolute percentage difference
496     RMSD = 0 #root mean  square difference
497 
498     if reference_path[-3:] == "csv":
499         referenece_var = pd.read_csv(reference_path,delimiter=',')
500     elif reference_path[-4:] == "xlsx":
501         referenece_var = pd.read_excel(reference_path)
502     referenece_var = referenece_var.to_numpy()
503 
504     error = np.zeros(np.shape(referenece_var))
505     error[:,0] = referenece_var[:,0]
506 
507     error[:,1] = (calculated_var[:,0]-referenece_var[:,1]) #simplle difference
508     RMSD = np.sqrt(np.sum(error[:,1]*error[:,1])/N_nodes)
509 
510     a = error[:,1]*100
511     b = referenece_var[:,1]
512     error[:,1] = np.divide(a, b, out=np.zeros_like(a), where=b!=0) #substitute nan by 0. PD
          obtained
513     MPD = np.sum(error[:,1])/N_nodes
514     MAPD = np.sum(np.absolute(error[:,1]))/N_nodes
515 
```

```
516     x = node_coord[:,1]
517     y = node_coord[:,2]
518     z = error[:,1]
519 
520     PD_max_def =
            (np.max(np.absolute(calculated_var[:,0]))-np.max(np.absolute(b)))/np.max(np.absolute(b))*100
521 
522     fig = plt.figure(figsize=(12, 8))
523     plt.tricontourf(x, y, z, levels=100)
524     plt.xlabel("x[m]")
525     plt.ylabel("y[m]")
526     plt.title(f"Percentage error [%] of {name_var} at {location} plate.MAPD={round(MAPD,2)}")
527 
528     #If branch to ensure that the color bar is consistent and always reaches to 0
529     if np.min(z) >=0 and np.max(z) >=0:
530         v = np.linspace(0, np.max(z), 9, endpoint=True)
531     elif np.min(z) <=0 and np.max(z) <=0:
532         v = np.linspace(np.min(z),0, 9, endpoint=True)
533     else:
534         v = np.linspace(np.min(z), np.max(z), 9, endpoint=True)
535     plt.colorbar(ticks=v)
536     fig.tight_layout()
537 
538     plt.gca().set_aspect("equal") #fixing up the ratio of the plate
539 
540     if save_opt:
541         if not os.path.exists(save_path):
542             os.makedirs(save_path)
543         #Saving graph
544         fig.savefig(save_path+f"\\error_{name_var}_{location}.png")
545         plt.close(fig)
546 
547         #Saving text file with extra info
548         (N_sens_xx,) = np.shape(strain_elements["exx"])
549         (N_sens_yy,) = np.shape(strain_elements["eyy"])
550         (N_sens_xy,) = np.shape(strain_elements["exy"])
551         f= open(save_path+f"\\error_{name_var}_{location}.txt","w+")
552         f.write(f"{N_sens_xx} sensing points in x.\n {N_sens_yy} sensing points in y.\n
                {N_sens_xy} sensing points in xy.\n MAPD={MAPD} \n MPD={MPD}\n RMSD={RMSD} \n PD
                maximum deflection {PD_max_def}")
553         f.close()
554     if show_opt:
555         plt.show()
556 
557 def perc_error_Ushape(node_coord,calculated_var,N_nodes,strain_elements,reference_path,name_var,
        show_opt, save_opt, save_path,location):
558     """
559     Function for plotting the percentage error at each node in the FEM model. 2d view.
560     When the reference value is 0, to avoid Nan the error is eplaced by 0. Might not be a
            representative error handling in all cases
561 
562     Args:
563         node_coord (array): Array of size (N_nodes,4) storing the coordinate of the nodes in the
                following format: ID | X | Y | Z
564         calculated_var (array): Array of size (N_nodes,1) with the values of a calculated array
565         N_nodes (int): Number of nodes in the iFEM mesh
566         strain_elements (int): Dictionary containing arrays of the elements where strain is
                recorded for each strain component. Eg. for strain exx we know whic elemebnts record
                strain. The keys are "exx", "eyy" and "exy".
567         reference_path (str): Path to the file where the reference measurements (FEM outputs) are
                stored.
568         name_var (str): Name of the variable for which error is computed. Used for saving the value
569         show_opt (bool): Determines if the figure is shown or not
```

```
570         save_opt (bool): Determines if the figure is saved or not
571         save_path (str): Determines where the figure is saved. Required if save_opt is set to true.
572         location (str): Location for running the iFEM algorithm. Can be "top","mid" or "bot"
573     """
574     plt.clf()
575 
576     #Calculation side
577     MPD = 0  #mean percentage difference
578     MAPD = 0 #mean absolute percentage difference
579     RMSD = 0 #root mean  square difference
580 
581     if reference_path[-3:] == "csv":
582         referenece_var = pd.read_csv(reference_path,delimiter=',')
583     elif reference_path[-4:] == "xlsx":
584         referenece_var = pd.read_excel(reference_path)
585     referenece_var = referenece_var.to_numpy()
586 
587     error = np.zeros(np.shape(referenece_var))
588     error[:,0] = referenece_var[:,0]
589 
590     error[:,1] = (calculated_var[:,0]-referenece_var[:,1]) #simple difference
591     RMSD = np.sqrt(np.sum(error[:,1]*error[:,1])/N_nodes)
592 
593     a = error[:,1]*100
594     b = referenece_var[:,1]
595     error[:,1] = np.divide(a, b, out=np.zeros_like(a), where=b!=0) #substitute nan by 0. PD
        obtained
596     MPD = np.sum(error[:,1])/N_nodes
597     MAPD = np.sum(np.absolute(error[:,1]))/N_nodes
598 
599     PD_max_def =
        (np.max(np.absolute(calculated_var[:,0]))-np.max(np.absolute(b)))/np.max(np.absolute(b))*100
600 
601     #Plotting side
602     fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 8),layout="constrained")
603 
604     #Bottom
605     check = np.isclose(node_coord[:,3], 0.) #z is 0
606     ax1 = node_coord[check,1] #x coord
607     ax2 = node_coord[check,2] #y coord
608     z = error[check,1]
609     bot_csf = axes[0].tricontourf(ax1, ax2, z, levels=100)
610     loc_n, = np.shape(error[check,1])
611     MAPDloc = np.sum(np.absolute(error[check,1]))/loc_n
612 
613     axes[0].set_title(f"Percentage error [%] of {name_var} Top View. MAPD={round(MAPDloc,2)}")
614     axes[0].set_xlabel("x[m]")
615     axes[0].set_ylabel("y[m]")
616 
617     total_z = z #for the colorbar limits
618 
619     #Front
620     check = np.isclose(node_coord[:,2], 0.) #y is 0
621     ax1 = node_coord[check,1] #x coord
622     ax2 = node_coord[check,3] #z coord
623     z = error[check,1]
624 
625     front_csf = axes[1].tricontourf(ax1, ax2, z, levels=100)
626     loc_n, = np.shape(error[check,1])
627     MAPDloc = np.sum(np.absolute(error[check,1]))/loc_n
628     axes[1].set_title(f"Percentage error [%] of {name_var} Front View. MAPD={round(MAPDloc,2)}")
629     axes[1].set_xlabel("x[m]")
630     axes[1].set_ylabel("z[m]")
```

```
631
632     total_z = np.append(total_z,z)
633
634     #Back
635     check = np.isclose(node_coord[:,2], np.max(node_coord[:,2])) #y is 0
636     ax1 = node_coord[check,1] #x coord
637     ax2 = node_coord[check,3] #z coord
638     z = error[check,1]
639
640     back_csf = axes[2].tricontourf(ax1, ax2, z, levels=100)
641     loc_n, = np.shape(error[check,1])
642     MAPDloc = np.sum(np.absolute(error[check,1]))/loc_n
643     axes[2].set_title(f"Percentage error [%] of {name_var} Back View. MAPD={round(MAPDloc,2)}")
644     axes[2].set_xlabel("x[m]")
645     axes[2].set_ylabel("z[m]")
646
647     total_z = np.append(total_z,z)
648     z = total_z
649     #If branch to ensure that the color bar is consistent and always reaches to 0
650     if np.min(z) >=0 and np.max(z) >=0:
651         v = np.linspace(0, np.max(z), 9, endpoint=True)
652     elif np.min(z) <=0 and np.max(z) <=0:
653         v = np.linspace(np.min(z),0, 9, endpoint=True)
654     else:
655         v = np.linspace(np.min(z), np.max(z), 9, endpoint=True)
656
657     plt.colorbar(back_csf,ax=axes,ticks=v)
658     # plt.gca().set_aspect("equal") #fixing up the ratio of the plate
659
660     axes[0].set_aspect("equal")
661     axes[1].set_aspect("equal")
662     axes[2].set_aspect("equal")
663
664     if save_opt:
665         if not os.path.exists(save_path):
666             os.makedirs(save_path)
667         #Saving graph
668         fig.savefig(save_path+f"\\error_{name_var}_Ushape.png")
669         plt.close(fig)
670
671         #Saving text file with extra info
672         (N_sens_xx,) = np.shape(strain_elements["exx"])
673         (N_sens_yy,) = np.shape(strain_elements["eyy"])
674         (N_sens_xy,) = np.shape(strain_elements["exy"])
675         f= open(save_path+f"\\error_{name_var}_Ushape.txt","w+")
676         f.write(f"{N_sens_xx} sensing points in x.\n {N_sens_yy} sensing points in y.\n
                {N_sens_xy} sensing points in xy.\n MAPD={MAPD} \n MPD={MPD}\n RMSD={RMSD} \n PD
                maximum deflection {PD_max_def}")
677         f.close()
678     if show_opt:
679         plt.show()
680
681 def iteration_RMSD_MPD(RMSD_lst, MPD_lst, MAPD_lst, parameterSEA_lst,save_path,location,parameter):
682     """
683     Plots and finds the best parameter for an iFEM reconstruction.
684
685     Args:
686         RMSD_lst (array): List of RMSF error in [-]
687         MPD_lst (array): List of MPD error in [%]
688         MAPD_lst (array): List of MAPD error in [%]
689         parameterSEA (array): List of parameter that was used for iteration
690         save_path (str):
691         location (str): Location along the plate: mid, top, bot
```

```
692         parameter (str): Name of the parameter: alpha/ beta/ w
693     """
694 
695     RMSD_ar = np.array(RMSD_lst)
696     MPD_ar = np.array(MPD_lst)/100
697     MAPD_ar = np.array(MAPD_lst)/100
698     parameterSEA_ar = np.array(parameterSEA_lst)
699 
700     best_MPD = np.min(np.abs(MPD_ar))
701     best_MAPD = np.min(np.abs(MAPD_ar))
702     best_RMSD = np.min(np.abs(RMSD_ar))
703 
704     best_MPD_parameter =  parameterSEA_ar[np.abs(MPD_ar)==best_MPD]
705     best_MAPD_parameter =  parameterSEA_ar[np.abs(MAPD_ar)==best_MAPD]
706     best_RMSD_parameter =  parameterSEA_ar[np.abs(RMSD_ar)==best_RMSD]
707 
708     if not os.path.exists(save_path+f"\\{parameter} iteration RMSD-MPD errors"):
709             os.makedirs(save_path+f"\\{parameter} iteration RMSD-MPD errors")
710 
711     #RMSD loglog
712     fig = plt.figure()
713     ax = plt.gca()
714     ax.plot(parameterSEA_ar,RMSD_ar, '-o', c='blue')
715     ax.set_xscale('log')
716     ax.set_yscale('log')
717     ax.set_xlabel(fr"log10($\{parameter}$[-])")
718     ax.set_ylabel("log10(RMSD[-])")
719     ax.set_title(fr"Variation of RMSD with $\{parameter}$")
720     plt.grid(True, which="both", alpha=0.4)
721     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD errors\\RMSDloglog_{location}.png")
722     plt.close(fig)
723 
724     #RMSD log
725     fig = plt.figure()
726     ax = plt.gca()
727     ax.plot(parameterSEA_ar,RMSD_ar, '-o', c='blue')
728     ax.set_xscale('log')
729     ax.set_xlabel(fr"log10($\{parameter}$[-])")
730     ax.set_ylabel("RMSD[-]")
731     ax.set_title(fr"Variation of RMSD with $\{parameter}$")
732     plt.grid(True, which="both", alpha=0.4)
733     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD errors\\RMSDlog_{location}.png")
734     plt.close(fig)
735 
736     #MPD loglog
737     fig = plt.figure()
738     ax = plt.gca()
739     ax.plot(parameterSEA_ar,MPD_ar, '-o', c='blue')
740     ax.set_xscale('log')
741     ax.set_yscale('symlog')
742     ax.set_xlabel(fr"log10($\{parameter}$[-])")
743     ax.set_ylabel("log10(MPD[-])")
744     ax.set_title(fr"Variation of MPD with $\{parameter}$")
745     plt.grid(True, which="both", alpha=0.4)
746     fig.savefig(save_path+f"\\{parameter} iteration RMSD-MPD errors\\MPDloglog_{location}.png")
747     plt.close(fig)
748 
749     #MPD log
750     fig = plt.figure()
751     ax = plt.gca()
752     ax.plot(parameterSEA_ar,MPD_ar, '-o', c='blue')
753     ax.set_xscale('log')
754     ax.set_xlabel(fr"log10($\{parameter}$[-])")
```

```
755     ax.set_ylabel("log10(MPD[-])")
756     ax.set_title(fr"Variation of MPD with $\{parameter}$")
757     plt.grid(True, which="both", alpha=0.4)
758     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD errors\\MPDlog_{location}.png")
759     plt.close(fig)
760 
761     #MAPD loglog
762     fig = plt.figure()
763     ax = plt.gca()
764     ax.plot(parameterSEA_ar,MAPD_ar, '-o', c='blue')
765     ax.set_xscale('log')
766     ax.set_yscale('symlog')
767     ax.set_xlabel(fr"log10($\{parameter}$[-])")
768     ax.set_ylabel("log10(MAPD[-])")
769     ax.set_title(fr"Variation of MAPD with $\{parameter}$")
770     plt.grid(True, which="both", alpha=0.4)
771     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD errors\\MAPDloglog_{location}.png")
772     plt.close(fig)
773 
774     #MAPD log
775     fig = plt.figure()
776     ax = plt.gca()
777     ax.plot(parameterSEA_ar,MAPD_ar, '-o', c='blue')
778     ax.set_xscale('log')
779     ax.set_xlabel(fr"log10($\{parameter}$[-])")
780     ax.set_ylabel("MAPD[-]")
781     ax.set_title(fr"Variation of MAPD with $\{parameter}$")
782     plt.grid(True, which="both", alpha=0.4)
783     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD errors\\MAPDlog_{location}.png")
784     plt.close(fig)
785 
786     #RMSD-MPD-MAPD loglog
787     fig = plt.figure()
788     ax = plt.gca()
789     ax.scatter(parameterSEA_ar,MPD_ar, c='blue', edgecolors='none',label="MPD")
790     ax.scatter(parameterSEA_ar,RMSD_ar, c='green', edgecolors='none',label="RMSD")
791     ax.scatter(parameterSEA_ar,MAPD_ar, c='red', edgecolors='none',label="MAPD")
792     ax.set_xscale('log')
793     ax.set_yscale('log')
794     ax.set_xlabel(fr"log10($\{parameter}$[-])")
795     ax.set_ylabel("log10(MAPD[-])/log10(MPD[-])/log10(RMSD[-])")
796     ax.set_title(fr"Variation of MPD/MAPD/RMSD with $\{parameter}$")
797     plt.grid(True, which="both", alpha=0.4)
798     fig.savefig(save_path+f"\\{parameter} iteration RMSD-MPD errors\\RMSDMPDloglog_{location}.png")
799     plt.close(fig)
800 
801     #RMSD-MPD logsymlog
802     fig = plt.figure()
803     ax = plt.gca()
804     ax.scatter(parameterSEA_ar,MPD_ar, c='blue', edgecolors='none',label="MPD")
805     ax.scatter(parameterSEA_ar,RMSD_ar, c='green', edgecolors='none',label="RMSD")
806     ax.scatter(parameterSEA_ar,MAPD_ar, c='red', edgecolors='none',label="MAPD")
807     ax.set_xscale('log')
808     ax.set_yscale('symlog')
809     ax.set_xlabel(fr"log10($\{parameter}$[-])")
810     ax.set_ylabel("log10(MAPD[-])/log10(MPD[-])/log10(RMSD[-])")
811     ax.set_title(fr"Variation of MPD/MAPD/RMSD with $\{parameter}$")
812     plt.legend()
813     plt.grid(True, which="both", alpha=0.4)
814     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD
            errors\\RMSDMPDlogsymlog_{location}.png")
815     plt.close(fig)
816 
```

```
817     #RMSD-MPD logsymlog zoomed
818     fig = plt.figure()
819     ax = plt.gca()
820     ax.scatter(parameterSEA_ar,MPD_ar, c='blue', edgecolors='none',label="MPD")
821     ax.scatter(parameterSEA_ar,RMSD_ar, c='green', edgecolors='none',label="RMSD")
822     ax.scatter(parameterSEA_ar,MAPD_ar, c='red', edgecolors='none',label="MAPD")
823     ax.set_xscale('log')
824     ax.set_yscale('symlog')
825     ax.set_xlabel(fr"log10($\{parameter}$[-])")
826     ax.set_ylabel("log10(MAPD[-])/log10(MPD[-])/log10(RMSD[-])")
827     ax.set_title(fr"Variation of MPD/MAPD/RMSD with $\{parameter}$")
828     ax.set_ylim(bottom=-1,top=1)
829     plt.legend()
830     plt.grid(True, which="both", alpha=0.4)
831     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD
            errors\\RMSDMPDlogsymlogzoomed_{location}.png")
832     plt.close(fig)
833
834     #RMSD-MPD log
835     fig = plt.figure()
836     ax = plt.gca()
837     ax.scatter(parameterSEA_ar,MPD_ar, c='blue', edgecolors='none',label="MPD")
838     ax.scatter(parameterSEA_ar,RMSD_ar, c='green', edgecolors='none',label="RMSD")
839     ax.scatter(parameterSEA_ar,MAPD_ar, c='red', edgecolors='none',label="MAPD")
840     ax.set_xscale('log')
841     ax.set_xlabel(fr"log10($\{parameter}$[-])")
842     ax.set_ylabel("MPD[-]/RMSD[-]")
843     ax.set_title(fr"Variation of MPD/MAPD/RMSD with $\{parameter}$")
844     plt.grid(True, which="both", alpha=0.4)
845     plt.legend()
846     fig.savefig(save_path+fr"\\{parameter} iteration RMSD-MPD errors\\RMSDMPDlog_{location}.png")
847     plt.close(fig)
848
849     f= open(save_path+fr"\\{parameter} iteration RMSD-MPD
            errors\\{parameter}_iteration_output.txt","w+")
850     f.write(f"RMSD error [-] {RMSD_ar}.\n MPD error [-] {MPD_ar}.\n MSPD error [-] {MAPD_ar}.\n
            best {parameter} MPD { best_MPD_parameter } at MPD {best_MPD}\n best {parameter} MAPD {
            best_MAPD_parameter } at MAPD {best_MAPD} \n best {parameter} RMSD {best_RMSD_parameter}
            at RMSD{best_RMSD}")
851     f.close()
852
853 def alpha_iteration_RMSD_MPD_component(RMSD_lst, MPD_lst, MAPD_lst,
        alfaSEA_lst,save_path,location, component):
854     """_summary_
855
856     Args:
857         RMSD_lst (_type_): _description_
858         MPD_lst (_type_): _description_
859         alfaSEA_lst (_type_): _description_
860         save_path (_type_): _description_
861         location (_type_): _description_
862         component (_type_): _description_
863     """
864     RMSD_ar = np.array(RMSD_lst)
865     MPD_ar = np.array(MPD_lst)
866     MAPD_ar = np.array(MAPD_lst)
867     alfaSEA_ar = np.array(alfaSEA_lst)
868
869     if not os.path.exists(save_path+f"\\Alpha Iteration Components\\{component}"):
870             os.makedirs(save_path+f"\\Alpha Iteration Components\\{component}")
871
872     #RMSD Plot log
873     fig = plt.figure()
```

```
874     ax = plt.gca()
875     ax.plot(alfaSEA_ar,RMSD_ar, '-o', c='blue')
876     ax.set_xlabel(r"$\alpha$[-]")
877     ax.set_xscale('log')
878     ax.set_ylabel("RMSD[-]")
879     ax.set_title(fr"Variation of RMSD with $\alpha$ for {component}")
880     plt.grid(True, which="both", alpha=0.4, axis="both")
881     fig.savefig(save_path+f"\\Alpha Iteration
            Components\\{component}\\RMSDvsalpha_{component}_{location}.png")
882     plt.close(fig)
883 
884     #MPD Plot log
885     fig = plt.figure()
886     ax = plt.gca()
887     ax.plot(alfaSEA_ar,MPD_ar, '-o', c='blue')
888     ax.set_xlabel(r"$\alpha$[-]")
889     ax.set_xscale('log')
890     ax.set_ylabel("MPD[%]")
891     ax.set_title(fr"Variation of MPD with $\alpha$ for {component}")
892     plt.grid(True, which="both", alpha=0.4)
893     fig.savefig(save_path+f"\\Alpha Iteration
            Components\\{component}\\MPDvsalpha_{component}_{location}.png")
894     plt.close(fig)
895 
896     #MADP Plotlog
897     fig = plt.figure()
898     ax = plt.gca()
899     ax.plot(alfaSEA_ar,MAPD_ar, '-o', c='blue')
900     ax.set_xlabel(r"$\alpha$[-]")
901     ax.set_xscale('log')
902     ax.set_ylabel("MAPD[%]")
903     ax.set_title(fr"Variation of MAPD with $\alpha$ for {component}")
904     plt.grid(True, which="both", alpha=0.4)
905     fig.savefig(save_path+f"\\Alpha Iteration
            Components\\{component}\\MAPDvsalpha_{component}_{location}.png")
906     plt.close(fig)
907 
908     best_MPD = np.min(np.abs(MPD_ar))
909     best_MAPD = np.min(np.abs(MAPD_ar))
910     best_RMSD = np.min(np.abs(RMSD_ar))
911 
912     best_MPD_alpha = alfaSEA_ar[np.abs(MPD_ar)==best_MPD]
913     best_MAPD_alpha = alfaSEA_ar[np.abs(MAPD_ar)==best_MAPD]
914     best_RMSD_alpha = alfaSEA_ar[np.abs(RMSD_ar)==best_RMSD]
915 
916     f= open(save_path+f"\\Alpha Iteration
            Components\\{component}\\alpha_iteration_{component}_output.txt","w+")
917     f.write(f"RMSD error [-] {RMSD_ar}.\n MPD error [-] {MPD_ar}. \n MAPD error [-] {MAPD_ar}. \n
            best alfa MPD { best_MPD_alpha } at MPD {best_MPD} \n best alfa MAPD { best_MAPD_alpha }
            at MAPD {best_MAPD}\n best alfa RMSD { best_RMSD_alpha } at RMSD{best_RMSD}")
918     f.close()
919 
920 def strain_Ushape(quads,plot_var,location,save_opt,show_opt,save_path,option, strain_elements):
921     """_summary_
922 
923     Args:
924         quads (_type_): _description_
925         plot_var (_type_): _description_
926         location (_type_): "top" "bot"
927         option(str): "SEA","FEM"
928     """
929     plt.clf()
930 
```

```
    #Creating subplots
    colors_from_img = np.load("pyife3d\supporting_files\colors_from_img.npy")
    N_entries,_ = np.shape(colors_from_img)
    my_cmap = LinearSegmentedColormap.from_list('my_cmap', colors_from_img, N=N_entries)

    fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 8), layout="constrained")
    #Select the correct displacement component

    if plot_var=="exx":
        ind = 0
    elif plot_var=="eyy":
        ind = 1
    elif plot_var=="exy":
        ind = 2

    strain_els = np.array(strain_elements[plot_var])
    x_coord, y_coord, z_coord, contour_var = [], [], [], []

    for element in quads:
        x_coord.append(float(element.probe.centroid[0]))
        y_coord.append(float(element.probe.centroid[1]))
        z_coord.append(float(element.probe.centroid[2]))

        if location=="top" and option=="SEA":
            contour_var.append(float(element.probe.epsilontopSEA[ind]))
        elif location=="top" and option=="FEM":
            contour_var.append(float(element.probe.epsilontop[ind]))
        elif location=="bot" and option=="SEA":
            contour_var.append(float(element.probe.epsilonbotSEA[ind]))
        elif location=="bot" and option=="FEM":
            contour_var.append(float(element.probe.epsilonbot[ind]))

    quad_ids = np.arange(1,len(quads)+1)
    xvals = np.vstack((quad_ids,np.array(x_coord)))
    yvals = np.vstack((quad_ids,np.array(y_coord)))
    zvals = np.vstack((quad_ids,np.array(z_coord)))
    strains = np.vstack((quad_ids,np.array(contour_var)))

    instr_check = np.in1d(xvals[0,:], strain_els) #check which elements are instrumented

    #Bottom Plate
    check = np.isclose(zvals[1,:], np.min(zvals[1,:])) #z is 0
    ax1 = xvals[1,check] #x coord
    ax2 = yvals[1,check] #y coord
    z = np.reshape(strains[1,check],np.shape(ax1))

    bot_csf = axes[0].tricontourf(ax1, ax2, z,
        levels=100,cmap=my_cmap,vmin=np.min(contour_var),vmax=np.max(contour_var),zorder=1)
    axes[0].set_title(f"Nodal contour: {plot_var} [-] Deck")
    axes[0].set_xlabel("x[m]")
    axes[0].set_ylabel("y[m]")

    if option=="SEA":
        final_check = np.logical_and(instr_check , check) #intersection between instrumentation
            and location
        axes[0].scatter(x=xvals[1,final_check], y=yvals[1,final_check],c="#FFFFFF",zorder=2)

    total_z = z

    #Front Plate
    check = np.isclose(yvals[1,:], np.min(yvals[1,:])) #y is min
    ax1 = xvals[1,check] #x coord
    ax2 = zvals[1,check] #z coord
```

```
992     z = np.reshape(strains[1,check],np.shape(ax1))
993 
994     front_csf = axes[1].tricontourf(ax1, ax2, z,
            levels=100,cmap=my_cmap,vmin=np.min(contour_var),vmax=np.max(contour_var),zorder=1)
995     axes[1].set_title(f"Nodal contour: {plot_var} [-] Side Wall Right")
996     axes[1].set_xlabel("x[m]")
997     axes[1].set_ylabel("z[m]")
998 
999     if option=="SEA":
1000        final_check = np.logical_and(instr_check , check) #intersection between instrumentation
               and location
1001        axes[1].scatter(x=xvals[1,final_check], y=zvals[1,final_check],c="#FFFFFF",zorder=2)
1002        axes[2].scatter(x=xvals[1,final_check], y=zvals[1,final_check],c="#FFFFFF",zorder=2)
1003 
1004    total_z = np.append(total_z,z)
1005 
1006    #Back Plate
1007    check = np.isclose(yvals[1,:], np.max(yvals[1,:])) #y is max
1008    ax1 = xvals[1,check] #x coord
1009    ax2 = zvals[1,check] #z coord
1010    z = np.reshape(strains[1,check],np.shape(ax1))
1011 
1012    back_csf = axes[2].tricontourf(ax1, ax2, z,
            levels=100,cmap=my_cmap,vmin=np.min(contour_var),vmax=np.max(contour_var),zorder=1)
1013    axes[2].set_title(f"Nodal contour: {plot_var} [-] Side Wall Left")
1014    axes[2].set_xlabel("x[m]")
1015    axes[2].set_ylabel("z[m]")
1016 
1017    total_z = np.append(total_z,z)
1018 
1019    complete_strain = plt.tricontourf(xvals[1,:], total_z, contour_var,
            levels=100,cmap=my_cmap,vmin=np.min(contour_var),vmax=np.max(contour_var))
1020 
1021    v = np.linspace(np.min(strains[1,:]), np.max(strains[1,:]), 9, endpoint=True)
1022 
1023    plt.colorbar(complete_strain,ax=axes,ticks=v)
1024 
1025    axes[0].set_aspect("equal")
1026    axes[1].set_aspect("equal")
1027    axes[2].set_aspect("equal")
1028 
1029    if save_opt:
1030        if not os.path.exists(save_path+f"\\{option}_strains"):
1031            os.makedirs(save_path+f"\\{option}_strains")
1032        fig.savefig(save_path+f"\\{option}_strains"+f"\{plot_var}_{location}_usection.png")
1033        plt.close(fig)
1034 
1035    if show_opt:
1036        plt.show()
1037 
1038 def undeformed_2d_instrumented_Ushape(quads, node_coord, strain_elements, show_opt, save_opt,
       save_path):
1039    plt.clf()
1040 
1041    color_map = {"inactive": "#d0cece",
1042                 "exx": "#ffadad",
1043                 "eyy": "#ffd6a5",
1044                 "exy": "#fdffb6",
1045                 "exxeyy": "#caffbf",
1046                 "exxexy": "#9bf6ff",
1047                 "eyyexy": "#a0c4ff",
1048                 "exxeyyexy": "#bdb2ff"}
1049 
```

```
    fig, axes = plt.subplots(nrows=3, ncols=1, figsize=(12, 8),layout="constrained")

    for quad in quads:

        #We start from the most restrictive conditions to the least restrictive
        #Depending on which strain components are insturmented, the element is highlighted in a
            specifc color
        if (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["eyy"]) and
            (quad.eid in strain_elements["exy"]):
            el_color = color_map["exxeyyexy"]
        elif (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["eyy"]):
            el_color = color_map["exxeyy"]
        elif (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["exy"]):
            el_color = color_map["exxexy"]
        elif (quad.eid in strain_elements["eyy"]) and (quad.eid in strain_elements["exy"]):
            el_color = color_map["eyyexy"]
        elif quad.eid in strain_elements["exx"]:
            el_color = color_map["exx"]
        elif quad.eid in strain_elements["eyy"]:
            el_color = color_map["eyy"]
        elif quad.eid in strain_elements["exy"]:
            el_color = color_map["exy"]
        else:
            el_color = color_map["inactive"]

        x = quad.probe.xe[0::3].reshape(4)
        y = quad.probe.xe[1::3].reshape(4)
        z = quad.probe.xe[2::3].reshape(4)

        if np.all(z==0):
            ax=0
            zipped = list(zip(x, y))
        elif np.all(y==0):
            ax=1
            zipped = list(zip(x, z))
        elif np.all(y==np.max(node_coord[:,2])):
            ax=2
            zipped = list(zip(x, z))
        zipped.append(zipped[0]) #close the polygon

        axes[ax].add_patch(Polygon(zipped,
                    edgecolor="black",
                    facecolor=el_color))

    axes[0].set_aspect("equal")
    axes[1].set_aspect("equal")
    axes[2].set_aspect("equal")

    axes[0].set_xlabel("x[m]")
    axes[0].set_ylabel("y[m]")

    axes[1].set_xlabel("x[m]")
    axes[1].set_ylabel("z[m]")

    axes[2].set_xlabel("x[m]")
    axes[2].set_ylabel("z[m]")

    axes[0].set_xlim([0,np.max(node_coord[:,1])])
    axes[0].set_ylim([0,np.max(node_coord[:,2])])

    axes[1].set_xlim([0,np.max(node_coord[:,1])])
    axes[1].set_ylim([0,np.max(node_coord[:,3])])
```

```
1111        axes[2].set_xlim([0,np.max(node_coord[:,1])])
1112        axes[2].set_ylim([0,np.max(node_coord[:,3])])
1113
1114        axes[0].set_title("Instrumented Elements-Deck")
1115        axes[1].set_title("Instrumented Elements-Side Wall Right")
1116        axes[2].set_title("Instrumented Elements-Side Wall Left")
1117
1118        inactive_patch = mpatches.Patch(color=color_map["inactive"]  , label='Not instrumented')
1119        exx_patch = mpatches.Patch(color=color_map["exx"], label='$\\epsilon_{xx}$')
1120        eyy_patch = mpatches.Patch(color=color_map["eyy"], label='$\\epsilon_{yy}$')
1121        exy_patch = mpatches.Patch(color=color_map["exy"], label='$\\gamma_{xy}$')
1122        exxeyy_patch = mpatches.Patch(color=color_map["exxeyy"], label='$\\epsilon_{xx},
                \\epsilon_{yy}$')
1123        exxexy_patch = mpatches.Patch(color=color_map["exxexy"], label='$\\epsilon_{xx},
                \\gamma_{xy}$')
1124        eyyexy_patch = mpatches.Patch(color=color_map["eyyexy"], label='$\\epsilon_{yy},
                \\gamma_{xy}$')
1125        exxeyyexy_patch = mpatches.Patch(color=color_map["exxeyyexy"], label='$\\epsilon_{xx},
                \\epsilon_{yy}, \\gamma_{xy}$')
1126        # plt.legend(handles=[inactive_patch, exx_patch, eyy_patch, exy_patch, exxeyy_patch,
                exxexy_patch, eyyexy_patch,exxeyyexy_patch],loc='center left', bbox_to_anchor=(1.07,
                0.5), fontsize="20")
1127        plt.legend(handles=[inactive_patch, exx_patch],loc='center left', bbox_to_anchor=(1.07, 0.5),
                fontsize="15")
1128
1129        if save_opt:
1130            if not os.path.exists(save_path):
1131                os.makedirs(save_path)
1132            fig.savefig(save_path+f"\\2d_instrumented_usection.png")
1133            plt.close(fig)
1134
1135        if show_opt:
1136            plt.show()
1137
1138    def undeformed_2d_instrumented_plate(quads, node_coord, strain_elements, show_opt, save_opt,
            save_path)
1139
1140        color_map = {"inactive": "#d0cece",
1141                     "exx": "#ffadad",
1142                     "eyy": "#ffd6a5",
1143                     "exy": "#fdffb6",
1144                     "exxeyy": "#caffbf",
1145                     "exxexy": "#9bf6ff",
1146                     "eyyexy": "#a0c4ff",
1147                     "exxeyyexy": "#bdb2ff"}
1148
1149        fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(12, 8),layout="constrained")
1150
1151        for quad in quads:
1152
1153            #We start from the most restrictive conditions to the least restrictive
1154            #Depending on which strain components are insturmented, the element is highlighted in a
                    specifc color
1155            if (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["eyy"]) and
                    (quad.eid in strain_elements["exy"]):
1156                el_color = color_map["exxeyyexy"]
1157            elif (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["eyy"]):
1158                el_color = color_map["exxeyy"]
1159            elif (quad.eid in strain_elements["exx"]) and (quad.eid in strain_elements["exy"]):
1160                el_color = color_map["exxexy"]
1161            elif (quad.eid in strain_elements["eyy"]) and (quad.eid in strain_elements["exy"]):
1162                el_color = color_map["eyyexy"]
1163            elif quad.eid in strain_elements["exx"]:
```

```
            el_color = color_map["exx"]
        elif quad.eid in strain_elements["eyy"]:
            el_color = color_map["eyy"]
        elif quad.eid in strain_elements["exy"]:
            el_color = color_map["exy"]
        else:
            el_color = color_map["inactive"]

        x = quad.probe.xe[0::3].reshape(4)
        y = quad.probe.xe[1::3].reshape(4)
        z = quad.probe.xe[2::3].reshape(4)

        zipped = list(zip(x, y))
        zipped.append(zipped[0]) #close the polygon
        axes.add_patch(Polygon(zipped,
                    edgecolor="black",
                    facecolor=el_color))

    axes.set_aspect("equal")

    axes.set_xlabel("x[m]")
    axes.set_ylabel("y[m]")

    axes.set_xlim([0,np.max(node_coord[:,1])])
    axes.set_ylim([0,np.max(node_coord[:,2])])

    inactive_patch = mpatches.Patch(color=color_map["inactive"]  , label='Not instrumented')
    exx_patch = mpatches.Patch(color=color_map["exx"], label='$\\epsilon_{xx}$')
    eyy_patch = mpatches.Patch(color=color_map["eyy"], label='$\\epsilon_{yy}$')
    exy_patch = mpatches.Patch(color=color_map["exy"], label='$\\gamma_{xy}$')
    exxeyy_patch = mpatches.Patch(color=color_map["exxeyy"], label='$\\epsilon_{xx},
        \\epsilon_{yy}$')
    exxexy_patch = mpatches.Patch(color=color_map["exxexy"], label='$\\epsilon_{xx},
        \\gamma_{xy}$')
    eyyexy_patch = mpatches.Patch(color=color_map["eyyexy"], label='$\\epsilon_{yy},
        \\gamma_{xy}$')
    exxeyyexy_patch = mpatches.Patch(color=color_map["exxeyyexy"], label='$\\epsilon_{xx},
        \\epsilon_{yy}, \\gamma_{xy}$')
    # plt.legend(handles=[inactive_patch, exx_patch, eyy_patch, exy_patch, exxeyy_patch,
        exxexy_patch, eyyexy_patch,exxeyyexy_patch],loc='center left', bbox_to_anchor=(1.07, 0.5))
    plt.legend(handles=[inactive_patch, exx_patch],loc='center left', bbox_to_anchor=(1.07, 0.5),
        fontsize="20")
    plt.show()

    if save_opt:
        if not os.path.exists(save_path):
            os.makedirs(save_path)
        fig.savefig(save_path+f"\\2d_instrumented_plate.png")
        plt.close(fig)

    if show_opt:
        plt.show()
```

Listing C.8: *plotters.py*

```

"""
Solving iFEM for a plate.

Input files:
    - Node_coordinates.xlsx : mesh node coordinates
    - Element_Nodes.xlsx: correspondence of nodes to each element
    - StrainResults.xlsx: the input strain
```

```
    - w_fact_dict.json: dictionary containing optimum w for each strain configuration
    - alfa_fact_dict.json: dictionary containing optimum alfa for each strain configuration

Make sure to change:
    - t depending on plate thickness (unit of t determines unit of obtained displacements)
    - bk depending on desired BC's

Processing Options:
    - Gauss_points_weights: type of integration
    - SEA (Smoothed Element Analysis) strain pre-extrapolation
    - isotropic material
    - location of strain calculation: top/mid or bottom of the plate.
    - betaSEA. Beta coefficient for curvature control in SEA.
    - drllingfact. Artificial stiffness added to SEA.

Post-processing options:

The analysis case, subcase and strain configuration are used for creating folder architecture.
    They are used to retrieve input data and to save the output in a mirrored folder which is
    also created by the code. An analysis case uses the same geometry and mesh. An analysis
    subcase can vary in terms of loading or BC. The strain configuration refers to different
    sensing networks.
"""

#IMPORTS
#Python Libraries
import sys
sys.path.append('..')

import numpy as np
import scipy
import pandas as pd
import json

#Developed Libraries
from pyife3d.plotters import deformed_3D, undeformed_3D, undeformed_3D_instrumented,
    nodal_contour2D, perc_error
from pyife3d.helpers import Gaussian_option, form_global_matrices, format_strain_data,
    read_iFEM_files, assemble_strain_elements, K_conditioning_number
from pyife3d.iFEM_main import iFEM
from pyife3d.iFEM_main_SEA import strain_extrapolation, iFEM_SEA
#-----------------------------------------------------------------------------------------------------------------------
#OPTIONs
SEA_opt = False
Gauss_points_weights = np.asarray(Gaussian_option(Gauss_type="3-point"))
isotropic = True
location = "mid" #"top" "mid" "bot"

#INPUT DATA
t = _  #[m] plate thickness
h = t/2    #[m] half thickness

analysis_case = "Insert Name of Analysis Case"
analysis_subcase = "Insert Name of Analysis Subcase"
strain_configurations = [
# "Configuration 1",
# "Configuration 2",
"Configuration 3",
]

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

#Composite
```

```
mat_direction = np.array([[1],
                          [0],
                          [0]]) #For example, principal direction is aligned with global X axis

input_file_root = f"iFEM examples\{analysis_case}\Macro Input\{analysis_subcase}\\"

#For non-SEA
with open(input_file_root+"w_fact_dict.json", 'r') as json_file:
    w_fact_dict= json.load(json_file)

#for SEA
if SEA_opt:
    with open(input_file_root+"alfaSEA_dict.json", 'r') as json_file:
        alfaSEA_dict= json.load(json_file)
    alfa_dict = alfaSEA_dict

betaSEA = 1e-4
drllingfact = 1e-5

#Options for saving and siplaying the generated graphs
show_opt = False
save_opt = True
cond_number_bool = False

component_dict = {"exx":1,"eyy":2,"exy":3}

for strain_configuration in strain_configurations:
    w_fact = w_fact_dict[strain_configuration]
    if SEA_opt:
        alfaSEA = {"exx":alfa_dict[strain_configuration],
                "eyy":alfa_dict[strain_configuration],
                "exy":alfa_dict[strain_configuration]}

    #-----------------------------------------------------SEA-----------------------------------------------------

    #READ FILES
    #These files will stay the same for a strain configuration no matter which component is
        extrapolated
    input_file_root = f"iFEM examples\{analysis_case}\Macro Input\{analysis_subcase}\\"
    node_coord, element_nodes, strain_data = read_iFEM_files(
                        path_node_coord=input_file_root+'Node_coordinates.xlsx',
                        path_element_nodes=input_file_root+'Element_Nodes.xlsx',
                            path_strain_data=input_file_root+f"{strain_configuration}\StrainResults.xlsx")
    #-------------------------------------------------------------------------------------------------------------

    #FORMAT DATA
    #Extracting number of nodes and elements
    (N_elements, _) =np.shape(element_nodes)
    (N_nodes, _) =np.shape(node_coord)

    strain_gauge_top,strain_gauge_bot = format_strain_data(N_elements,strain_data)

    strain_elements = assemble_strain_elements(strain_gauge_top)
    #-------------------------------------------------------------------------------------------------------------

    #ELEMENT ITERATION
    if SEA_opt:

        SEA_U_dict_top = {} #dictionary to store our SEA results
        SEA_U_dict_bot = {}

        #Bending only shortcut
        # component_dict = {"exx":1}
```

```
127
128         #Top Plate interpolation
129         for component in component_dict:
130             #Reformat the strain measurements for SEA
131             strain_elementsSEA = strain_elements[component]
132             strain_gauge = np.zeros((len(strain_elementsSEA),2)) #format of this is ELEMENT ID |
                    strain measurements
133             strain_gauge[:,0] = strain_elementsSEA
134             strain_gauge[:,1] =
                    strain_gauge_top[(strain_gauge[:,0]-1).astype(int),component_dict[component]]
135
136             #Strain extrapolation
137             quads, probes = strain_extrapolation(alfaSEA[component], betaSEA, drllingfact,
                    N_elements,element_nodes,node_coord,strain_gauge, strain_elementsSEA,
                    Gauss_points_weights)
138             DOF=4
139
140             #Asssembling global matrices using generated local matrices
141             K, F = form_global_matrices(quads=quads,N_nodes=N_nodes,DOF=DOF)
142
143             U = scipy.linalg.solve(K,F)
144
145             SEA_U_dict_top[component] = U
146
147         #Bottom Plate interpolation
148         for component in component_dict:
149             #Reformat the strain measurements for SEA
150             strain_elementsSEA = strain_elements[component]
151             strain_gauge = np.zeros((len(strain_elementsSEA),2)) #format of this is ELEMENT ID |
                    strain measurements
152             strain_gauge[:,0] = strain_elementsSEA
153             strain_gauge[:,1] =
                    strain_gauge_bot[(strain_gauge[:,0]-1).astype(int),component_dict[component]]
154
155             #Strain extrapolation
156             quads, probes = strain_extrapolation(alfaSEA[component], betaSEA, drllingfact,
                    N_elements,element_nodes,node_coord,strain_gauge, strain_elementsSEA,
                    Gauss_points_weights)
157             DOF=4
158
159             #Asssembling global matrices using generated local matrices
160             K, F = form_global_matrices(quads=quads,N_nodes=N_nodes,DOF=DOF)
161
162             U = scipy.linalg.solve(K,F)
163
164             SEA_U_dict_bot[component] = U
165
166         # #Bending only shortcut
167         # SEA_U_dict_top["eyy"] = np.zeros(np.shape(SEA_U_dict_top["exx"]))
168         # SEA_U_dict_top["exy"] = np.zeros(np.shape(SEA_U_dict_top["exx"]))
169         # SEA_U_dict_bot["eyy"] = np.zeros(np.shape(SEA_U_dict_bot["exx"]))
170         # SEA_U_dict_bot["exy"] = np.zeros(np.shape(SEA_U_dict_bot["exx"]))
171
172         #Using the pre-extrapolated strains, run now iFEM
173         quads, probes = iFEM_SEA(N_elements,element_nodes,node_coord, strain_gauge_top,
                strain_gauge_bot, strain_elements, h, Gauss_points_weights,w_fact, isotropic,
                mat_direction, SEA_U_dict_top, SEA_U_dict_bot, location)
174         DOF=6
175
176     else:
177         #No pre-extrapolation is Run
178         quads, probes = iFEM(N_elements,element_nodes,node_coord, strain_gauge_top,
                strain_gauge_bot, strain_elements, h, Gauss_points_weights,w_fact, isotropic,
```

```
            mat_direction, location)
        DOF=6

    #Asssembling global matrices using generated local matrices
    K, F = form_global_matrices(quads=quads,N_nodes=N_nodes,DOF=DOF)

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

    #BOUNDARY CONDITIONS
    bk = np.zeros(N_nodes*DOF, dtype=bool) #constrained DOF's

    #Here you can hard code your boundary conditions.
    #The example contains a cantilevered BC at x=0
    check = np.isclose(node_coord[:,1], 0.)
    bk[0::DOF] = check
    bk[1::DOF] = check
    bk[2::DOF] = check
    bk[3::DOF] = check
    bk[4::DOF] = check
    bk[5::DOF] = check

    bu = ~bk #unknown DOF's
    #-----------------------------------------------------------------------------------------------------------------
    #SOLVING THE SYSTEM
    Ku = K[:,bu][bu,:]
    Fu = F[bu]
    Uu = scipy.linalg.solve(Ku,Fu)
    U = np.zeros((N_nodes*DOF,1))
    U[bu] = Uu

    #-----------------------------------------------------------------------------------------------------------------
    #PLOTTING
    if SEA_opt:
        saving_path = f"iFEM examples\{analysis_case}\Results\{analysis_subcase}\iFEM
            SEA\{strain_configuration}"
    else:
        saving_path = f"iFEM
            examples\{analysis_case}\Results\{analysis_subcase}\iFEM\{strain_configuration}"

    undeformed_3D(quads=quads,show_opt=show_opt, save_opt=save_opt, save_path=saving_path)

    deformed_3D(quads=quads,U=U,DOF=DOF,show_opt=show_opt, save_opt=save_opt,
        save_path=saving_path, location=location)

    undeformed_3D_instrumented(quads=quads, strain_elements=strain_elements, show_opt=show_opt,
        save_opt=save_opt, save_path=saving_path)

    interest_vars = ["T3"]

    for interest_var in interest_vars:
        nodal_contour2D(node_coord=node_coord, U=U, DOF=DOF, plot_var=interest_var,
            show_opt=show_opt, save_opt=save_opt, save_path=saving_path, location=location)

    #-----------------------------------------------------------------------------------------------------------------
    # COMPUTE ERROR
        if interest_var == "T3":
            perc_error(node_coord=node_coord,calculated_var=U[2::DOF], N_nodes=N_nodes,
                strain_elements=strain_elements,
                reference_path=input_file_root+"Reference_U3.xlsx", name_var=interest_var,
                show_opt=show_opt, save_opt=save_opt, save_path=saving_path, location=location)

df = pd.DataFrame(U)
df.to_excel(saving_path+f'\\ResultsU_{location}.xlsx', index=False)
```

Listing C.9: *template.py*