Flexible Dynamics and Attitude Control of a Square Solar Sail

by

Mirue Choi

A thesis submitted in conformity with the requirements for the degree of Doctor of Philosophy
Graduate Department of Aerospace Engineering
University of Toronto

© Copyright 2015 by Mirue Choi
Abstract

Flexible Dynamics and Attitude Control of a Square Solar Sail

Mirue Choi
Doctor of Philosophy
Graduate Department of Aerospace Engineering
University of Toronto
2015

This thesis presents a comprehensive analysis of attitude and structural dynamics of a square solar sail. In particular, this research examines the use of corner-attached reflective vanes to control the attitude of the spacecraft. An introduction to known solar sail designs is given, then the mathematics involved in calculating solar radiation pressure forces are presented. A detailed derivation and implementation of the unconstrained nonlinear flexible structural dynamics with Finite Element Method (FEM) models are explored, with several sample simulations of published large deflection experiments used as verification measures. To simulate the inability of a thin membrane to resist compression, the sail membrane elements are augmented with a method that approximates the wrinkling and the slacking dynamics, which is followed by a simulation of another well-known experiment as a verification measure.

Once the structural dynamics are established, the usage of the tip vanes is explored. Specifically, a control allocation problem formed by having two degrees of freedom for each tip vane is defined and an efficient solution to this problem is presented, allowing desired control torques to be converted to appropriate vane angles. A randomized testing mechanism is implemented to show the efficacy of this algorithm. The sail shadowing problem is explored as well, where a component of the spacecraft casts shadow upon the sail and prevents solar radiation pressure force from being produced. A method to calculate the region of shadow is presented, and two different shadowing examples are examined - due to the spacecraft bus, and due to the sail itself.

Combining all of the above, an attitude control simulation of the sail model is presented. A simple PD controller combined with the control allocation scheme is used to provide the control torque for the sail, with which the spacecraft must orient towards a number of pre-specified attitude targets. Several attitude orientations are simulated, then a number of modifications to the control are explored. The controllability of the rigid and the elastic modes of the sail at different stable states is demonstrated as well, showing that the sail is controllable for all its modes in its stable state.
Acknowledgements

I would like to acknowledge my mother and my grandmother, both of whom have been ceaseless in their support of me throughout the years of study. I would not be here without them.

I would also like to sincerely thank Dr. Christopher J. Damaren, whose teaching and guidance were the core reasons for this research being completed. His expertise and knowledge bridged many gaps this author experienced during the years of research.

Finally, I would like to acknowledge the current and the previous members of the Spacecraft Dynamics and Controls Lab - Dr. James Forbes, Dr. Stefan LeBel, Dr. Ludwik Sobiesiak, Julian Stellini, Mark Binette, Javier Romualdez, Armen Meras, and Lun Li. They have created a great, friendly research environment for the last four years.
## Contents

1 Introduction 1  
1.1 Background History of Solar Sailing 2  
1.2 Mission Applications for Solar Sails 5  
1.2.1 Geomagnetic Storm Warning Missions 5  
1.2.2 Polar Observation Missions 6  
1.2.3 Interstellar Missions 6  
1.3 Thesis Motivation and Scope 7  

2 Solar Sail Structural Design 9  
2.1 Simple Square Solar Sail 9  
2.2 Cord Mat Square Solar Sail 11  
2.2.1 Pre-Constructed Sag within Sail 12  
2.3 Other Notable Sail Structural Designs 13  
2.3.1 Heliogyro 13  
2.3.2 Disc Solar Sail 14  
2.3.3 Interplanetary Kite-craft Accelerated by Radiation Of the Sun (IKAROS) 15  
2.4 Square Sail Boom Designs 16  
2.4.1 L’Garde Inflatable Boom 16  
2.4.2 Continuous Longeron Coilable Boom 17  
2.5 Sail Membrane Materials 17  
2.5.1 CP1 18  
2.5.2 Mylar 18  
2.5.3 Kapton 19  

3 Simple and Realistic Solar Radiation Pressure Forces and Torques 20  
3.1 Notations and Frames 20  
3.2 Simple Solar Radiation Pressure Force 21  
3.3 Realistic Solar Radiation Pressure Force 22  
3.3.1 Classical Realistic Solar Radiation Pressure Force 22  
3.3.2 Vector-Form of the Realistic Solar Radiation Pressure Force 24  
3.3.3 The Linear Photonic Thrust Model 24  
3.4 Comparison of Generated Forces between Different Solar Radiation Pressure Forces 25  
3.5 Calculation of Solar Radiation Pressure Forces and Torques on Sail FEM Model 26
4 Unconstrained Nonlinear Flexible Structural Dynamics

4.1 Deriving the Equation of Equilibrium with Geometrical Nonlinearities

4.1.1 Deriving the Constrained Equations of Motion with Geometrical Nonlinearities

4.1.2 Adding Rigid Dynamics to the Equations of Motion

4.1.3 Kinematics of the Motion

4.1.4 Calculation of Damping Matrix for Nonlinear Equations of Motion

4.2 Equations of Motion with Geometrical Nonlinearities for Beams

4.2.1 Beam Element Shape Function for FEM Model

4.3 Equations of Motion with Geometrical Nonlinearities for Plates

4.3.1 Triangular Plate Element Shape Function for FEM Model

4.3.2 Three Dimensional Displacements for Beam Element and Triangular Plate Element

4.4 Implementation of the FEM Model and the Dynamics Simulation

4.4.1 Nodes, Elements, and Frame Orientations

4.4.2 Element Stiffness Matrix Generation

4.4.3 Element Mass Matrix and Inertial Force Generation

4.4.4 Global Matrix Generation

4.4.5 Dynamics Simulation Via Numerical Differentiation Formulas

4.4.6 Sparse Matrix System

4.5 Sample Simulation: The Rotating Beam Problem

4.5.1 Problem Description

4.5.2 Simulation Setup

4.5.3 Simulation Results and Comparison to Known Literature

4.6 Sample Simulation: Clamped Plate Deformation Due to Vertical Loads

4.6.1 Problem Description

4.6.2 Simulation Setup

4.6.3 Simulation Results

4.7 Sample Simulation: Cord-Mat Solar Sail with SRP Force Loading

4.7.1 Problem Definition

4.7.2 Simulation Setup

4.7.3 Simulation Results

5 Wrinkling and Slacking Dynamics of a Membrane

5.1 Background Information

5.2 Describing Wrinkling and Slacking Behaviour Via Modification of the Constitutive Relationship

5.3 Describing String Slacking Behaviour Via Modification of Constitutive Relationship

5.4 Membrane State Criterion and the Behaviour of Wrinkling Parameters in a Dynamic System

5.5 Membrane State Criteria on Sail Deformation Due to Solar Radiation Pressure Loads

5.5.1 Standard Miller-Hedgepeth Mixed-Criterion Update
5.5.2 Parameter Modification Via Gradual Changes to the Parameter 90
5.5.3 Parameter Modification Via Weighted Criterion 92
   Average Weighted Criterion 93
   Accumulative Weighted Criterion 94
   Time-Limited Accumulative Weighted Criterion 94
   Selective Accumulative Weighted Criterion 96
5.5.4 Simulation Results with Division 8-2 Cord-Mat Solar Sail 96
5.6 Sample Simulation: Corner Loads on a Square Membrane 97
   5.6.1 Problem Description 100
   5.6.2 Results from Known Literature 100
   5.6.3 Simulation Setup 103
   5.6.4 Simulation Results 104

6 Control Allocation of Tip Vanes with Two Degrees of Freedom 108
   6.1 Solar Radiation Pressure Torques Due to One-DOF Vanes 108
   6.2 Solar Radiation Pressure Torques Due to Two-DOF Vanes 110
   6.3 The Control Allocation Problem 112
      6.3.1 Attainable Moment Set of Vanes and its Estimation 113
      6.3.2 Formal Definition of the Control Allocation Problem 117
      6.3.3 Analytical Solution to the Single Vane Problem 119
   6.4 Code Implementation 121
   6.5 Numerical Example 123

7 Sail Shadowing Due to Spacecraft Components 126
   7.1 Projection of a Component Onto the Sail Surfaces 126
   7.2 Determining Shadowed Elements in Sail FEM Model 129
      7.2.1 Numerical Example: Triangular Element Shading Due to Bus 130
   7.3 Numerical Example: Sail Membrane Shading Due to Itself 131

8 Attitude Control of a Solar Sail 138
   8.1 Quaternion PD Control with Vane Control Allocation 138
      8.1.1 Code Implementation 139
      8.1.2 Problem Definition and Simulation Setup 141
      8.1.3 Simulation Results: $\theta_x = \pi/2, \theta_y = \theta_z = 0$ 143
      8.1.4 Simulation Results: $\theta_x = \pi/4, \theta_y = \theta_z = 0$ 145
      8.1.5 Simulation Results: $\theta_x = \pi/2, \theta_y = \theta_z = \pi/4$ 147
      8.1.6 Simulation Results: $\theta_x = \pi/4, \theta_y = \theta_z = \pi/4$ 149
   8.2 Handling Disturbance Torques Due to Sail Structure 152
      8.2.1 Statically-Shifted Centre of Mass 153
      8.2.2 Static Bias Torque Subtraction 154
   8.3 Controllability of the Vane Actuators 155
      8.3.1 Linearization of the Equations of Motion 155
      8.3.2 Testing the Controllability of the Sail 159
      8.3.3 Controllability Simulation Results 160
9 Conclusion

A Implementation of the FEM Model and the Dynamics Simulation

A.1 Describing and Operating Upon Polynomials - Polynom_Module

remove_overlap .................................................. 170
add, sub ............................................................... 170
mult, div, power .................................................. 171
differentiate ....................................................... 172
integrate ............................................................. 172
value ................................................................. 173
save_poly, load_poly ............................................... 173

A.2 Element Stiffness Matrix Generator Modules .............................................. 174

A.3 Global Matrix Generation Module ................................................................. 176

A.3.1 Matrix Indexing Generation for Elements Belonging to One SailStruct ................. 176
A.3.2 Handling Multiple Shapes of Multiple Types of Elements ................................. 176
A.3.3 Nodes with Displacement Restrictions ........................................................ 177
A.3.4 Implementation of Sparse Matrix System ...................................................... 178

matrix_to_slu, slu_to_matrix ........................................ 178
slu_simplify ............................................................ 179
slu_add, slu_sub .................................................... 179
slu_mult ............................................................... 179
slu_decomp, slu_solve ............................................. 180
gen_global_sparse ................................................. 180

Modifications to SailStruct and ODEFnc_Module ............................................. 183

Bibliography

Bibliography
Chapter 1

Introduction

For an unknowable amount of time, the concept of travelling to the stars has captured the imagination of the many, but it was only within the last century that technological advances have allowed mankind to escape the Earth’s atmosphere for the first time. It took decades of research and testing to develop the means to escape the Earth’s gravity - a near-explosive thrust provided by chemical combustion from a rocket propellant - and because of the time invested in developing rocket propulsion, it is often the more commonly available and chosen means of propulsion even for in-space operations. However, this reliance on rocket propulsion is not ideal for any lengthy operations in space, because of its inherent need for propellants in order to operate. The $\Delta v$ produced by a rocket propulsion is limited by the amount of available propellant. One could argue that this is resolved by simply carrying more propellant, but such a solution requires larger storage space for the propellant, which, combined with the extra propellant, means more weight for the spacecraft in general and hence requiring more thrust force to achieve the same acceleration as before, which has to come from using more propellant. This exponential scaling does not bode well for the rocket propulsion for any long-term operations - mankind has barely breached the borders of our own solar system, let alone reached another star. It would seem that another method of propulsion would be needed for such endeavours to be viable.

Fortunately, there is a source available as a means of propulsion in space, similar to the winds on the Earth - namely, the light photons from the Sun. Solar power has been a primary source of electric power for orbiting satellites, allowing them to operate out on-orbit continuously for years on end. One can harness the photons in a different manner and achieve Solar Sailing - the process of reflecting sunlight off of large reflective surfaces, the momentum inherent in the light photons and the resulting reaction force on the reflective surface propelling the spacecraft forward. This method of propulsion requires no on-board propellant since its source of momentum-changing substance is completely detached from the spacecraft. It is in theory capable of unrestricted continuous travel in space, making it highly ideal for missions with very long lifetimes, such as station-keeping or deep-space missions.

The momentum carried by each photon is small however. Since the photons per square meter of area is fixed by what is received from the Sun, instead the size of the reflective area is increased to capture more momentum. In addition, this momentum translates to notable acceleration of the spacecraft only if it is relatively significant when compared against the mass of the object being accelerated. The two facts stated above essentially shapes the design of a solar sailing spacecraft - a large, deployable gossamer-like reflective membrane, constrained by lightweight support structures to be flat enough in a
desired direction and reflecting the photons to produce thrust, with the spacecraft bus attached to this membrane structure in some manner.

The behaviour of this membrane and its supporting structures are of great interest to many researchers in the field of solar sailing, as its shape directly determines in which direction the sail receives and reflects the photons to, resulting in possibly unwanted or skewed thrust directions and torques that may affect the spacecraft and the mission adversely. Yet understanding the dynamics of a membrane is not an easy endeavour due to its tendency to behave non-elastically by wrinkling and slacking. In addition, complete ground testing of a solar sail for a realistic structural analysis is infeasible as forces imparted upon the sail by gravity greatly outweigh that of the photons. Hence, it falls upon computer simulations to identify as much of the behaviour and potential pitfalls of the sail design and the structures that compose it before it is launched into the orbit, where real environmental conditions could cause damage upon it.

The large dimensions of a solar sail pose yet another problem - that of its attitude actuation. A typical spacecraft has a number of attitude actuation options, such as reaction wheels, magnetic torquers, control moment gyros and thrusters. With the exception of thrusters, these actuators are designed with compactly shaped satellites in mind. As solar sails possess relatively much larger moments of inertia compared to typical satellites, these actuators are unable to produce the torques needed to effectively control the sail’s attitude. While thrusters can be positioned at the far ends of the reflective surface to produce effective torques, these would naturally need propellant to operate, making them an unattractive option by limiting the spacecraft’s lifetime by the amount of propellants carried by the spacecraft once again.

There are alternatives to typical actuation schemes thanks to the peculiarities of solar sailing. One method takes advantage of torques produced by the sail itself when its centre of mass is shifted by using a shifting or gimbaled mass, while another method simply attaches additional, reorientable reflective surfaces to the edges of the sail to act like a thruster. The second implementation, here called the tip vane, is particularly attractive as it can provide full three-axis control regardless of the sail’s own orientation to the sun unlike the mass-centre-shifting method. It is also deceptively complicated to use, because its under-constrained nature provides multiple solutions to a particular desired torque, and its position at the ends of the sail making it susceptible to structural dynamics of the sail changing its behaviour. These are all interesting issues to consider and provide solutions for.

There are many other issues to consider as well - sail deployment mechanics, sail materials, optical properties, power considerations and orbital dynamics are all major topics of interest when a solar sail is being designed. The sail’s structural dynamics and control are selected as the focus of this research, and information pertaining to other topics are limited to references and simplifications.

1.1 Background History of Solar Sailing

Delving into the history of solar sailing reveals much about its lack of on-orbit data for use in its development, despite the idea being around for nearly a century now. McInnes notes in his book that the fundamental idea of solar sailing has been around since the 1920s, when the Soviet father of aviation, Konstantin Tsiolkovsky, and his co-worker Fridrichk Tsander, wrote of ‘using tremendous mirrors of very thin sheets’ and ‘using the pressure of sunlight to attain cosmic velocities’ [35]. However, they did not pursue these ideas further. It was only in the 1950s that the concept was re-invented by Carl Wiley,
an aeronautical engineer who in May 1951 wrote about the concept and the design of a solar sail in *Astounding Science Fiction* under a pseudonym Russell Sanders.

In 1958, a first solar sailing paper was published in the journal *Jet Propulsion* by Richard Garwin, noting its inherent benefits in comparison to other methods of propulsion being considered and used, coining the term ‘solar sailing’ in the process. The optimistic prospect suggested by the paper led to more detailed studies of solar sailing during the late 1950s and the early 1960s, resulting in many preliminary studies in solar sail orbits and mission applications. During this time, the well known short story *The Wind from the Sun* was published in 1963 by Arthur C. Clarke. A story about a manned solar sail race in Earth orbit, it popularized the concept of solar sailing to many prospective engineers. Yet despite the level of interest this garnered, there was a distinct lack of focus in solar sail research during this period, as there was no specific mission present to drive detailed analyses and development.

With the development of the Space Shuttle and various deployable structures in the early 1970s, the prospect of launching and deploying solar sails became closer to reality. In 1973, NASA began funding low-level studies of solar sailing at the Battelle laboratories in Ohio. During the continuation of this study, Jerome Wright discovered a trajectory that allowed a solar sail to perform a rendezvous mission with comet Halley at its perihelion in the mid-1980s, with a flight time of merely four years, allowing a late 1981 or early 1982 launch. Prior to this discovery, the mission was thought to be near-impossible with the level of technology available, with the next best option of solar-electric ion propulsion requiring near twice the flight time and hence requiring an early 1977 launch. This discovery sparked an engineering assessment study of the technological readiness level study of solar sails by the Jet Propulsion Laboratory. By September 1976, a formal proposal was submitted to NASA management, and by November, an actual mission design of the comet Halley rendezvous via solar sailing began.

Initially, a $800 \text{ m} \times 800 \text{ m}$ three-axis stabilized square solar sail was considered. The design was abandoned in favour of a $7.5 \text{ km}$ twelve-bladed spin-stabilized heliogyro, citing concerns with sail deployment. The heliogyro design, developed by Richard MacNeal at the Astro Research Centre ten years prior to the project, possessed the advantage of being able to be deployed by simply spinning and letting the resulting centrifugal force unroll the blades. The comet Halley rendezvous mission provided an excellent opportunity to disseminate its structural dynamics and the resulting attitude control.

However, solar-electric propulsion had a much larger advocacy group within NASA and in industry, and after a year of vigorous competition, solar-electric propulsion won out. It was officially chosen over solar sailing for the comet Halley mission in September 1977, citing less risk. Shortly after, the mission itself was canceled altogether due to rising cost estimates of the solar-electric propulsion and the scientific community’s desire to perform a rendezvous mission as opposed to a lower-cost flyby mission.

Nonetheless, the comet Halley mission greatly vitalized the field. Detailed studies were performed on the behaviour and feasibility of the two sails considered for the mission. The orbit trajectory to the comet and the required delta V were determined. Reflective films meant for the sail membrane were produced and analyzed to determine their properties. But most important of all, the design studies sparked greater interest in solar sailing throughout various space communities. Spurred by NASA’s design studies, the World Space Foundation (WSF) was formed in California in 1979 by Robert Staehle and others who were part of the Jet Propulsion Laboratory’s solar sail work, attempting to raise private funds to continue their work on solar sailing. Shortly after in 1981, the Union pour la Promotion de la Propulsion Photonique (U3P) was formed in Toulouse, proposing a solar sailing race to the Moon. In 1982, the Solar Sail Union of Japan (SSUJ) was formed, joining the other two groups in promoting
and advancing solar sailing technology. In 1992, the US Columbus Quincentennial Jubilee commission
proposed a solar sail race to Mars contest, one of its participants being the Canadian Solar Sail Project
(CSSP) group, composed of members from the Canadian Space Society and UTIAS.

During the 1990s, a key technology useful for solar sails was demonstrated on-orbit. It was in 1993
when the Russian Space Regatta Consortium successfully deployed Znamya, a spinning 20 m reflector
with the end goal of illuminating northern Russian cities during dark winter months. It was deployed
from a Progress supply vehicle and observed from the MIR space station, demonstrating the viability of
spin-driven deployment for flexible membranes. NASA demonstrated their own deployable structure as
well through the Inflatable Antenna Experiment (IAE). Built by a company called L’Garde Inc., a 14 m
diameter inflatable antenna made out of mylar was launched from the Space Shuttle Endeavour in May
1996. The antenna was inflated, and then was de-orbited a few days later. These were key proof-of-
concept demonstrations for solar sailing, lending credence to the idea of deploying very large membrane
structures in space. With the deployment technology available, the German Aerospace Agency (DLR)
successfully conducted a ground deployment test of a 20 m × 20 m solar sail, paving a promising start
for the millennium.

The new millennium saw a rapid build-up of solar sailing technology. In 2002, Team Encounter
successfully tested the deployment of a sail segment built by L’Garde, the success eventually prompting
funding from NASA’s In-Space Propulsion Technologies (ISPT) program to L’Garde’s striped-net sail
with inflatable boom and Able Engineering’s CP1 sail with coilable boom. This research led to two 20
m × 20 m sails of different designs being constructed and successfully being deployed on the ground in
2005. Neither of these sails flew however. Apart from these two sails, there was also the ill-fated Cosmos
1 solar sail by the Planetary Society, which was launched in 2001, and then in 2005, both times ending
in failure due to launch failures. Another sail that shared the same fate as Cosmos was Nanosail-D, a
solar sail developed by NASA. Nanosail-D was to be carried to Low Earth Orbit (LEO) by the Falcon 1
rocket in 2008, but was lost when Falcon-1’s stage 1 thruster collided with stage 2, causing the vehicle
to explode.

It was on 21 May 2010 when a solar sail was finally launched into orbit successfully. JAXA’s Interplanetary Kite-craft Accelerated by Radiation Of the Sun (IKAROS), a spin-stabilized solar sail,
confirmed full deployment of its sail and acceleration due to solar radiation in July. Apart from its
achievement as the world’s first solar sail in space, it also demonstrated attitude control via a reflectance
varying device and managed to perform a flyby of Venus, providing credence to the concept of inter-
planetary travel via solar sails. Shortly after IKAROS’ success, Nanosail-D2 was launched in December
2010, deploying successfully and cruising on-orbit for 240 days before re-entering the atmosphere.

The success of IKAROS has spurred others on. In particular, NASA announced the Sunjammer
project - a solar sailing project with a goal to reach a sub L1 point, slated to be launched in the near
future. Sunjammer is a 38 m × 38 m sail being built by L’Garde, possessing similar design characteristics
as the one that was developed for the ISTP. The Planetary Society’s sail project, LightSail-1, a square
sail possessing similar characteristics as Nanosail-D, is still ongoing as well. The European Space Agency
(ESA) has proposed a sail project Gossamer, designed to use solar pressure and atmospheric drag to
accelerate the deorbiting process of small satellites. All in all, the immediate future of research on solar
sailing seems bright.
1.2 Mission Applications for Solar Sails

There are several notable mission applications that make solar sails particularly attractive due to their continuous propulsive nature. Here, some of the prominent solar sailing mission prospects are introduced for reference purposes.

1.2.1 Geomagnetic Storm Warning Missions

Ever since the occurrence of March 1989’s geomagnetic storm causing the collapse of Hydro-Quebec’s electricity grid and leaving millions without power, prediction of such occurrences has been of a great interest to many. Subsequent disruptions and damage done to the satellites in orbit throughout the next decade led to the launch of the Advanced Composition Explorer (ACE) spacecraft, a spacecraft following a halo orbit around the L1 point. The L1 point, illustrated in Figure 1.1 along with the other Lagrangian points, is noteworthy in that it maintains a position between the Sun and the Earth. This allows any satellites residing at it to sense disturbances from the sun before they reach the Earth, hence providing an early-warning system for solar events - namely, solar photon events. However, the L1 point for the Sun-Earth system is very close to the Earth due to the significant scale difference in the two bodies' gravitation field, and the warning time is limited by this. It is desirable to provide earlier warnings in order to give associated systems on Earth more time to respond to the warning, yet conventional spacecraft would need to expend great amounts of $\Delta V$ for station-keeping in any locations other than the Lagrangian points, which quickly drives up the cost of any lengthy missions.

Solar sails provide a solution to this problem by counteracting the gravitational pull from the sun with the continuous photonic thrust from the sail. Specifically, the thrust generated by the solar sail acts as an additional variable within the equation of the restricted three body problem, resulting in
equilibrium solutions - the Lagrangian points - that can vary with this thrust. The region of all possible equilibrium deviating away from the L1 point via some predefined thrust performance is often referred to as the sub-L1 equilibrium region in the literature. This region allows reasonably-sized solar sails to operate in a stationary state far closer to the sun than the current L1 missions, with no fuel-based mission lifetime limitation. Numerous studies have been performed to validate and provide detailed analysis of the concept [31, 35].

One often-cited mission concept for geomagnetic storm warning is the Geostorm mission (also known as Heliostorm). It is a mission concept involving a solar sailing spacecraft to reach and station-keep at a sub-L1 point located approximately 0.98 AU away from the sun, which is twice the distance away from the Earth compared to the ACE spacecraft [60]. As a direct result of its closer proximity to the Sun, the warning time for geomagnetic storms would be improved from ACE’s 10 minutes to 1 to 2 hours prior to the occurrence on Earth.

1.2.2 Polar Observation Missions

Keplerian orbits, and by extension the majority of spacecraft, are limited in the sense that the motion must be around the centre of mass of a celestial mass. Any other orbital maneuver would require immense amounts of propulsion to counteract the pull of the gravity and in general is deemed infeasible due to it. This is problematic for any scientific endeavour requiring extended observation of the pole of a planet or a star, as any Keplerian orbit would only pass over the pole for a limited length of time per orbit. One can artificially extend the length of the orbit in which the pole is visible by placing its apoapsis far away from the planet, but in this case the observation must be performed from a distance. It would be more ideal to be able to perform continuous observation from the same distance.

Solar sails can provide a solution to this dilemma by introducing non-Keplerian orbits [35]. Non-Keplerian orbits, as the name suggests, are orbits that are different from Keplerian orbits in varying ways. For the use in polar observation, a typical Keplerian orbit can be raised out of the ecliptic plane with the continuous thrust from the solar sail, until the sail’s orbit is a halo on top of the planet. Alternatively, an artificial Lagrangian point can be created with the continuous thrust, allowing the spacecraft to remain stationary. Regardless of the choice of station-keeping method, with the elevated location of the orbit, the spacecraft is in a position where continuous observation of the pole at a close range can be performed, which a spacecraft in a Keplerian orbit cannot.

The mission is applicable for both the Earth and the Sun. For the Earth, a polar-sitting satellite can provide both environmental observation and high latitude communication service to the Arctic for scientific, commercial and governmental purposes. For the Sun, its pole is unobservable from the Earth, and a pole sitter would be able to provide useful scientific information about the Sun at the pole.

1.2.3 Interstellar Missions

One of the greatest advantages of solar sailing is its acceleration from the solar radiation pressure. Taking advantage of this limitless propulsion, a solar sail can approach the source of its thrust, the Sun, as closely as it physically can, and then spiral outward as it continues to rapidly gain speed. Proper orientation of the sail allows it to eventually slingshot itself away from the sun towards its target, still gaining speed as it continues to travel away from the sun. Theory suggests that a sufficiently large sail can propel a spacecraft beyond the solar system to other important targets within a human lifespan [35].
Typical targets of such interstellar missions vary. One mission involves sending a radio telescope to the solar gravitational lens at 550 AU away from the sun for astrophysics applications. Another mission aims to simply propel outward until the sail escapes the Sun’s gravitational field. Ambitious missions call for travel to Alpha Centauri. Most of these missions however are not within the reach of the current state of solar sailing however, requiring extremely large sails weighing far less.

1.3 Thesis Motivation and Scope

As mentioned earlier, the main scope of this thesis is on the solar sail structural dynamics and its attitude control. The purpose of the work is not the engineering modelling of sail behaviour but a pioneering first step in that direction. The main motivation of the research stems from a desire to study how to control a solar sail given its highly flexible structural dynamics. There is a potential issue that the deformation experienced by the solar sail during its operation may produce disturbance torques that adversely affect the performance of the sail’s attitude control. Several major topics must be analyzed in order to assess this possibility. Understanding the sail’s structural behaviour under solar radiation pressure loading is crucial. Just as important is the mapping of the desired control torque to the attitude actuation input. Disturbances torques inherent in solar sailing as well as the behaviour of the control system due to these disturbances must be considered as well. To facilitate the study, the cord-mat sail structural design by Greschik [20] is adopted, as well as using tip vanes for attitude actuation. This choice of sail design is motivated by the design itself posing numerous interesting problems to resolve. General and specific solar sail structural designs, including the cord-mat sail design, are briefly introduced in Chapter 2 as a reference.

In Chapter 3, the mathematics behind calculating Solar Radiation Pressure (SRP) forces are introduced, along with the frames used in this research. A simple, realistic and linear SRP force equations are derived and compared with each other to gauge their differences. As well, with the sail being modelled as membrane elements, a method of calculating the SRP forces due to the sail is introduced for future references.

A different look into the sail structural dynamics, which the membrane is typically modelled as a linear elastic film, is provided by an implementation of a geometrically-nonlinear Finite Element Method (FEM) sail model using a nonlinear plate for the membrane, introduced in Chapter 4. Derivation of the nonlinear dynamics equation is performed from the first principles for both the rigid and the elastic bodies. The two element types used, the Hermitian beam and the natural coordinate triangular plate elements, are derived as well. Many details involved in creating global equations of motion for the entire sail are provided, shedding light onto the inner workings of the simulator implemented by the author. To give credence to the simulator, two published examples of nonlinear elastic problems are simulated, followed by a simulation of the sail model itself. The simulation results are analyzed and compared against the known experimental results.

The equations of motion are combined with a membrane wrinkling model to account for the incompressibility of a thin membrane, detailed in Chapter 5. The membrane wrinkling model is a modified Miller-Hedgepeth model, which acts upon the constitutive relationship based on the current state of the membrane to force all negative stresses to zero, thereby simulating the membrane’s inability to be compressed and adding another layer of fidelity to the simulation. Several criteria update methods are explored in order to allow the criteria to provide a valid result for the simulation. A well-known corner-
loaded square membrane problem is simulated, comparing its experimental results with the simulation’s results to validate the model. While preliminary, this chapter provides a unique look into how a wrinkling membrane behaviour can be accounted for without the use of a highly-detailed FEM model.

Chapter 6 introduces a vane actuation scheme designed by the author and considers the control allocation problem posed by the scheme. As noted above, tip vanes are chosen as the sail’s attitude actuator. A method of attitude actuation using SRP forces is desirable to maintain the propellant-less nature of the solar sail, yet such typical actuators such as reaction wheels and moment gyros are ineffective for solar sails. The vanes provide a method of attitude actuation that takes advantage of its wide nature. A total of four vanes are attached, one for each corner of the square sail, each possessing two angular degrees of freedom (DOF). With three Cartesian direction torque components to produce from these actuators, the eight-DOF control system is under-constrained and multiple combinations of vane angles may exist for one particular set of torques. This is posed as a control allocation problem, which a solution for is provided as a part of this study.

During the operation, a portion of the reflective sail may become shaded by other spacecraft components. Chapter 7 explores this issue by deriving a method for determining how an object casts shadow onto the sail surface. This method is used to demonstrate how shadowing occurs due to the bus and the sail itself under certain sail angles. Correct determination of the shadowed area adds another layer of fidelity to the simulation for the edge-on cases and the attitudes near it, as well as providing a method for identifying and analyzing shadowed areas for other possible solar sail designs.

Using a simple feedback controller to provide input to the control allocation system, the sail structural dynamics is simulated with an arbitrary point to point attitude maneuver, paying close attention to the disturbances created by the sail flexibility and the performance of the tip vanes. The topics covered from previous chapters are integrated into the dynamics simulation and presented in Chapter 8. The validity of the work from the previous topics are analyzed and affirmed with this all-encompassing simulation. As an aside, the controllability of the vane actuation scheme is also shown to affirm the system as a fully controllable one.
Chapter 2

Solar Sail Structural Design

With over half a century of preliminary designing for solar sails by various researchers, several different sail design concepts have been created. These different design concepts are introduced below, along with variations in different sail components.

2.1 Simple Square Solar Sail

A common design concept found in the literature is that of a square solar sail, where the membranes are typically stretched between four equal-length support booms [5, 8, 13, 17, 19–21, 27, 34, 35, 38, 40, 41, 48, 51–56, 61]. A simple solar sail used by many researchers is shown in Figure 2.1. It is composed of a sail deployment bus at the centre of the structure, with four equal-length booms projecting out from its sides. The diagram here depicts four triangular membranes, with vertices anchored at the sail booms and the bus by tensioned strings to keep the membranes stretched out. The actual satellite bus,
not depicted here, is typically either built around the sail deployment bus, or is attached to the sail by a boom at the centre.

![Figure 2.2: Simple Square Sail Loading](image)

The square solar sail has several design factors to consider. The most important factor is that of the support boom. The load created by the solar radiation pressure on a square solar sail is supported entirely by the four support booms. A simple illustration of how the solar sail loading affects a supporting boom is given in Figure 2.2. As seen in the figure, the SRP force generated by the light photons pushes the sail downward. Since the reflective sail membrane is anchored against the sail boom which supports the membrane, a reactionary force is generated at the anchor connected to the sail membrane, which pulls at the boom. In case of the simple square solar sail, the symmetry of the sail causes the in-plane bending forces to cancel out, but the boom still suffers from the compressive component which pushes the boom into the body centre, and the bending components which bends the boom downward in the direction of the sun vector. Both of these force components threaten to buckle the boom unless it is designed to handle the expected loads.

The sail is also designed to be deployed in orbit due to the immense size of the fully deployed sail, and to be light in mass as larger mass is detrimental to the sail’s propulsion performance. This trait conflicts with the trait of the support boom, and hence must be traded off during the design process of the booms. Two examples of boom designs can be found in [38] and [20], both of which will be discussed in greater detail later on in this chapter.

Another consideration is the suspension of the sail. Figure 2.1 shows one type of suspension where the sail is divided into four quadrants, and each triangular quadrant is attached to the booms and the bus by the corners. Figure 2.3 shows several other suspension choices cited in the literature [20, 38]. Figures 2.3 (a) through (c) are the more common design concepts seen in the older literature. The four-point and five-point suspension concepts shown in Figure 2.3 (a) and (c) are attractive due to their simplicity and deployability - a large square membrane attached at its corners (and the centre) to the supporting structure, and would essentially deploy along with the booms. Its main disadvantage comes from the single sail membrane used for the entire sail, which can suffer from rip and tear propagating over all parts of the sail from just one quadrant, as well as posing a stowage structure and sail manufacturing challenges. The separate quadrants concept shown in Figure 2.3 (c), i.e. the simple solar sail design illustrated in Figure 2.1 alleviates this issue somewhat by dividing the sail into four quadrants, still maintaining all the advantages of the first two configurations. However, all three of these concepts still suffer from concentrated loads at the ends of the boom, requiring greater structural stiffness to prevent
the booms from collapsing.

![Figure 2.3: Square Sail Suspension Methods [20, 38]](image)

The continuous connection concept shown in Figure 2.3 (d) attempts to alleviate the issue with the load by dividing the loading from the sail across multiple anchor points. The striped architecture concept shown in 2.3 (e) is a step further from the continuous connection concept by introducing striped-sail design, where a series of reflective sail strips are strewn across from one boom to another, forming a square reflective region similar to the other designs, but each sail strip not necessarily connected to each other. This offers greater control over the loads on the booms. These concepts however have been criticized for their complicated sail construction requirements.

### 2.2 Cord Mat Square Solar Sail

One of the design concepts that deviate from the above is the cord mat square solar sail, introduced by Greschik [20]. This design concept is notable in that it inherently does not stretch the sail membrane, i.e., there is no tension applied on the membrane structure to keep it taut. Instead, the sail membrane is essentially ‘laid onto’ (and attached to) a set of cords suspended between the support booms, these cords being placed in tension. Owing to the nature of this structural peculiarity, some billowing occurs with the sail, but the shape and the size of the billow would be controlled by the cords. Any loading onto the support boom is restricted to those from the cords, allowing for more predictable loading on the booms. Specifically, the loading paths for this structure is such that the SRP force on the sail creates a reaction force perpendicular to the support cords, which, thanks to the symmetric nature of the cords and the sail material in between, translate into forcing on the booms that is along the path of the cords. More details on the loading can be found in Greschik’s work [20]. However, the membrane slackness has an inherent downside of making the prediction of the sail membrane dynamics and the resulting SRP force a complicated matter.

The basic structure of a cord mat sail is illustrated in Figure 2.4. The cords are suspended at equal lengths along the support booms, with their initial lengths chosen such that a predetermined sag is built into the structure. The sail membrane in theory is simply laid on top of these cords with no pretensioning, but as a matter of structural integrity the sail membranes are attached to the cords that they are laid on. From a manufacturing standpoint it would be more appropriate to say that the cords are built into the sail membrane. Regardless, the key point is that since the sail membrane naturally sags instead of being stretched taut, the planar stresses experienced by the support booms are significantly reduced, allowing a great reduction in mass.
2.2.1 Pre-Constructed Sag within Sail

It is assumed that the suspension cords, under a slack, stable initial state with very small uniform lateral external loading, form a catenary sag in the direction of the loading. A quadrant of the sail in such initial state is shown in Figure 2.5. The key pre-constructed sags in the sail surface geometries are noted in this figure.

First and the foremost, the sail quadrant as a whole is assumed to be sagged such that the cords form a gradually-deepening set of catenaries. Specifically, the cord sag grows linearly from zero at the sail centre to \( \Delta \) at the centre of the outermost cord. Given the illustrated two-cord model, the inner cord catenary has a centre depth of \( \Delta/2 \), whereas the outer cord catenary is sagged at the centre by \( \Delta \). This is a natural bagginess built into the cords to provide uniform stress levels at the support booms.

Secondly, the outer sail edge has an inward curve, noted by \( \Gamma \) in the diagram. This, noted by Greschik [20], is due to a small amount of inward tension applied to the cord by the membrane. Specifically, when
out-of-plane loads are applied to the membrane, the load is transferred to the cords. The intermediate
 cords would be loaded from both sides, but the outermost cord is loaded only on one side only, causing
it, and hence the sail edge, to camber inward. One can model this camber to be spread out amongst the
outer cords, but for simplicity it is assumed that the inward camber exists only for the outermost cords.

Lastly, the sail membrane in between the cords is assumed to be slack, and hence a catenary is formed
here as well. It is worth noting that the depth of this film billow is measured from the cord, which is
also curved. Hence the resulting portion of sail membrane forms a curvature, its shape defined by Ω and
 Δ (and Γ as well in the case of the outermost portion of the membrane).

2.3 Other Notable Sail Structural Designs

While the subsequent work in this thesis focuses on the square solar sail, there are several other sail
design concepts worth noting for their historical significance or structural novelty. Here, three such sails
are discussed - the heliogyro, the disc solar sail, and the boomless solar sail design concepts. The scope
of this thesis precludes studying these in detail.

2.3.1 Heliogyro

An often-cited solar sailing concept is the Heliogyro, conceived initially by Richard MacNeal at the
Astro Research Corporation and studied by him and John Hedgepeth. It was one of the two spacecraft
design concepts considered for the comet Halley rendezvous mission in NASA during the 1970’s [35]. A
signiﬁcant amount of concept studies were done for the concept owing to this consideration and resulted
in the characterization of its structural dynamics and control mechanism.

![Figure 2.6: The Heliogyro](image)

The heliogyro is composed of several long, slender blades of reflective surface, positioned as shown in
Figure 2.6. The entire structure spins slowly, using the resulting centrifugal force to deploy and maintain flat, uniform surface for the blades. Each blade rotates along its length in a cyclic fashion, similar to helicopter blades, in order to precess its spin axis as well as to offset the centre of pressure from the centre of mass, and hence provide a measure of attitude control. Since the spin keeps the reflective film flat, the blades would not require any significant support booms as the square solar sails would and hence the overall mass for its reflective area is reduced.

While the concept seems feasible, the heliogyro designed for the comet Halley rendezvous mission used twelve blades, each 7.5 km long. Aside from the manufacturing and packing challenges this massive blade would pose, it is questionable as to how blades of that length can be rotated along its length to provide spin axis control for the sailcraft as a whole. Such rotation is plausible only with support material along the edges of the blades to add torsional stiffness to the entire blade, but such added mass, especially for such long blades, would be quite detrimental to the sail’s performance. It is worthwhile to note however that smaller-scaled heliogyros have been designed as well, and NASA’s Langley Research Centre is currently developing one such design. It is to be seen if the new design is viable for future flights.

2.3.2 Disc Solar Sail

An alternative to the heliogyro design concept is the disc solar sail as seen in Figure 2.7, where a circular disc of reflective surface is spun out of the deployment bus and once again held flat by the spin-induced tension, along with number of radial spars along the sail. The spars for example can be used as the deployment mechanism, initially being wound around a central hub and then released during deployment, the resulting elastic energy from the unwinding spars unfurling the circular sail. This passive deployment strategy is attractive in its simplicity, requiring little additional mechanical components to deploy the sail.

![Figure 2.7: Disc Solar Sail](image-url)
Apart from the passive deployment strategy, its reflective area is dimensionally similar to a square sail but has a smaller mass requirement due to its lack of stiff support booms to keep the sail flat. The cyclic blade rotation attitude control strategy used by the heliogyro is not feasible however, and any strategies that take advantage of the stiff booms on square sail cannot be used either, which essentially limits its attitude control actuation to either a gimballed mass or a sail surface with variable reflectivity.

2.3.3 Interplanetary Kite-craft Accelerated by Radiation Of the Sun (IKAROS)

It is interesting to note that while much of the research on solar sailing seem to have gone to either the square solar sail or the heliogyro, the one that became the first solar sailing spacecraft was that of a spin-stabilized square solar sail, very similar to the disc solar sail introduced above. IKAROS, JAXA’s experimental solar sailing spacecraft and shown in Figure 2.8, was launched on 21 of May 2010, successfully deployed on 10 June, and acceleration due to solar radiation pressure (as well as its attitude control capacity) was verified on 9 July.

![Figure 2.8: JAXA’s IKAROS Solar Sailing Spacecraft [16]](image)

The sail has several noteworthy design points. Most notably, the sailcraft has no support booms or spars - instead, it has a long tether in place of where support booms would be for a typical square sail, and at the end of the tether is a tip mass, which, in combination with the rotating spacecraft, keeps the sail flat enough to produce acceleration. This design also acts in a similar manner as the spars in the disc solar sail during deployment - the masses are released, and as they recede away from the spacecraft by centrifugal force, the sail unfurls along with them. The mass savings for the lack of spars or special deployment mechanisms are no doubt significant.

The control of the sailcraft is performed by Reflectivity Control Devices (RCD) attached near the edges of the sail. The concept behind these devices is simple enough - by varying the reflectivity of a portion of the sail, there is a shift in the sail’s centre of pressure, away from the spacecraft’s centre of mass. The result translates to a torque for the sail. IKAROS has long since met its primary goal of demonstrating trajectory modification via solar radiation pressure in a controlled manner, which seems to show that attitude control of reasonable precision is quite possible with these devices.
Chapter 2. Solar Sail Structural Design

2.4 Square Sail Boom Designs

It should be noted that the focus of this research is not on the structural designs of sail booms, but on the structural dynamics of the sail as a whole and the resulting interaction with a control system. The simulations performed later use a simple cylindrical beam with Young’s modulus calculated to be equivalent to the composite beam structure. Nonetheless, for the sake of completeness, two notably different square sail boom designs are introduced here as a reference.

2.4.1 L’Garde Inflatable Boom

L’Garde inflatable boom is a type of boom that is designed by L’Garde, initially proposed as a part of the solar sailing project for NASA’s ISPT program, then for the Team Encounter Solar Sail project [30], and modified and adopted for the Sunjammer mission that is currently ongoing as of 2014. The design of the boom is shown in Figure 2.9 (a).

L’Garde’s boom is initially fitted into a tightly folded and packed state within its deployment bus. The sail is also folded along with the boom packaging, attached at a regular interval to the boom. When the time comes for the sail to be deployed, pressurized gas is released into the booms, causing them to inflate at a controllable rate - the controllability of the inflation rate being important in order to allow the attached sails to unfurl correctly. This inflation process is photographed in Figure 2.9 (b) during one of L’Garde’s tests in a water trough. Once the sail is inflated, epoxy within the boom undergoes a sub glass transition temperature rigidization, then the inflation gas is released into the environment. The result is a transition from a tightly packed material to a lengthy, rigid set of booms with unfurled sails attached to them, with very little active mechanical implements involved.

The main disadvantage of this particular method of deployment is that the resulting inflated boom may not be as stiff as one may desire. L’Garde’s sail designs are often seen with spreader bar systems attached to them [13, 20, 30], which stiffen the booms against loads in a particular direction, but are not as effective when loads are not in the direction the spreader bars are designed for. The above-mentioned cord-mat sail design and the resulting slackness about the sail film also support this idea - such untensioned sail film would minimize the load on the booms. Apart from this disadvantage, the design also suffers from having to carry extra weight in the form of the pressurized gas storage and the
related hardware to inflate the boom (though these can be ejected), as well as potential complications when folding the boom and the attached sail into the deployment bus.

2.4.2 Continuous Longeron Coilable Boom

Another popular boom design is the Continuous Longeron Coilable Boom (CLCB), more commonly known as the coilable boom design [35]. Developed by ATK (previously known as ABLE Engineering Company) as a graphite-epoxy evolution of Astro Research’s original fiberglass longeron Astromast design, CLCB is essentially a truss structure with triangular cross-sections, as shown in Figure 2.10. CLCB design has heritage in that it was used in various space missions, two notables being the Cassini spacecraft and the Mars Pathfinder. The sail that used this boom design was competing directly against the boom used in L’Garde’s solar sail in the In-Space Propulsion Technologies program [37].

Initially packed tightly in a coil, the elastic energy within the truss causes the boom to extend outward like a spring being released, with lanyards attached at the tip of the boom and the other end of the wire on active motors to control the deployment rate. The sail would be attached at the tip as well, deploying alongside the coiled boom. The resulting deployed boom is stiff due to its truss-like structural design, and able to support loads produced by its payload - in this case, the sail films. Apart from the passive deployment, the design also has the potential for retraction of the boom, which may be beneficial for situations where the sail becomes detrimental to the mission objective.

The main issue with this system seems to come from its compatibility with solar sails in general. For cases where the sail is only attached to the tip of the booms, this type of boom is entirely suitable, although it does result in great susceptibility to boom buckling, leading to much larger booms. For a case where continuous connections of the sail to the booms is desired, the sail cannot be attached to the coiled structure without risking serious entanglement before and during deployment. Active motors and riggings required to control the deployment rate also becomes extra weight for the sail.

2.5 Sail Membrane Materials

As noted above, the focus of this research is on the interactions between the dynamics and the control system instead of the individual structural dynamics, and the membrane is no exception. No effort has been made to describe the variance in dynamics of metallized membrane materials used here, instead
focusing on the membranes being unable to resist compression. The membrane materials, similar to the booms, are generalized to a flat plate model - with special considerations when it is under compression. More information about the membrane materials can be found in [35].

A typical sail membrane composition can be described as follows: highly-reflective aluminum (or similarly light) coating on the front (sun-ward) of the sail, a select membrane material possessing tear-resistance and durability at the core, and depending on the choice of the membrane material and thermal considerations, highly emissive chromium (or similarly emissive) coating on the back (anti sun-ward) of the sail. Depending on the design, grounding straps and rip stops may be part of the design as well.

The different core membrane materials considered for use in solar sailing missions typically are selected for their capacity to be produced as light, wide and ultra-thin sheets, while being tear-resistant and durable in the space environment. The low mass to area ratio of the sail contributes positively to the maximum acceleration the sail can achieve, yet the sail must be able to withstand the strains and hazards imposed upon it during its lifetime. The different material types available in the market offer different properties to consider, and here three of the major solar sail membrane materials are introduced for reference purposes. Much of the information presented here can be found in [7], while others have been derived directly from material data sheets.

2.5.1 CP1

CP1 is a transparent polyimide material by NeXolve. It was used as the core membrane material by ATK for their prototype sail [29], a 2 \( \mu \)m thick membrane with aluminum on the front and bare on the back. The material itself possesses 1.53 g/m\(^2\) areal density, a slightly higher value than other materials available commercially. The material is particularly attractive in that it has significant space heritage, with its good UV resistance and non-conductivity. The sail devised by ATK using this material resulted in a relatively light sail subsystem areal density of 12 g/m\(^2\) for a 20 m spacecraft.

Its glass transition temperature is at 263°C, and the membrane’s back emissivity is 0.3. It is also inherently weak in terms of Ultimate Tensile Strength (UTS), noted on the data sheet as 87 MPa.

2.5.2 Mylar

Mylar, also known as Biaxially-oriented PolyEthylene Terephthalate (BoPET) is a polyester film invented in the early 1950’s and currently being produced by DuPont. L’Garde used Mylar as a part of their sail membrane for their prototype sail [29] in the ground demonstration contract for the ISTP, composed of a 2 \( \mu \)m thick membrane with aluminum on the front for reflectivity and blackened chromium on the back for thermal emissivity. The material possesses 1.39 g/m\(^2\) areal density, lighter than CP1 polyimide. In addition to its light mass, it is also attractive in terms of durability, possessing UTS of 200 MPa.

Mylar unfortunately possesses a relatively low operating temperature, rated to operate at the highest temperature of 150°C, but losing more than half its tensile strength by the time the material reaches 100°C. It is also considered not UV stable, requiring metallic coating on both sides to prevent degradation due to UV exposure. This low capability to operate in the sun marks the material less attractive for use in solar sailing.
2.5.3 Kapton

Kapton is a polyimide film manufactured by DuPont. This material was initially under consideration for L’Garde’s prototype sail for the ISTP, but was not chosen due to the unavailability of the desired film thickness. However since then, the general interest in solar sails have spurred DuPont to make available thinner variants of the material, prompting the Sunjammer mission, NASA’s current ongoing effort for a sub-L1 station-keeping solar sail, to adopt the 5 µm Kapton-based membrane [3]. It is assumed that this sail is also composed of aluminum layer on the front and blackened chromium layer on the back. Its areal density is 1.42 g/m² and its UTS is 231 MPa, both its lightness and durability comparable to Mylar.

Kapton is also more attractive than Mylar in terms of its thermal stability - at 200°C, a temperature well-past Mylar’s operating point, Kapton still possesses a UTS of 139 MPa. It is also considered UV stable, an additional plus for Kapton. The back emissivity is assumed to be 0.4 as before due to the blackened chromium coating. Its only downside is that the currently-available 5 µm membrane is comparably thicker than the other materials introduced here, resulting in greater sail areal density.
Chapter 3

Simple and Realistic Solar Radiation Pressure Forces and Torques

3.1 Notations and Frames

There are several frames and notations used to describe the sail, shown in Figure 3.1. For the sake of symmetry, the centre of the sail is assumed to be the origin of its body frame $\mathbf{F}_b$. The distance vector from an inertial frame $\mathbf{F}_i$ to $\mathbf{F}_b$ is $\mathbf{r}$. Each element that composes the sail has an element frame $\mathbf{F}_e$ associated with it, with the distance vector from the body centre to the frame described by $\mathbf{\rho}_e$. The body deforms when forces are applied, hence the point associated with the original location of the element frame changes. The frame at this changed orientation is the deformed frame $\mathbf{F}_d$, and the deformation is described by $\mathbf{u}_e$.

The components of vectors are transformed from one frame to another using the rotation matrix $\mathbf{C}$, with subscripts to denote the frames involved. For example, $\mathbf{C}_{id}$ would represent a rotation matrix that rotates a vector’s components in the deformed frame $\mathbf{F}_d$ to their values in the inertial frame $\mathbf{F}_i$.

For the sake of clarity, the vector components in the following sections are all frame-neutral, and vectors of any of the frames described above can be used, as long as all vectors are described in the same frame.

Figure 3.1: Frames and Displacement Notations for a Solar Sail
3.2 Simple Solar Radiation Pressure Force

On an ideal optical surface, the entirety of the Solar Radiation Pressure (SRP) is specularly reflected off a flat surface with no absorption. The combination of the momentum absorbed from the incoming photons and the reaction provided by the reflected photons results in a force vector that is always normal to the flat surface, as shown in Figure 3.2. The unit vector from the sun to a point on the surface is $\hat{s}$, and the direction of reflection from this point represented by the unit vector $\hat{r}$. The unit normal vector for the reflective surface is $\hat{n}$, the resulting SRP force $f_{\text{simple}}$ is in the opposite direction of the unit normal, and the angle from $\hat{n}$ to $\hat{s}$ is $\alpha$.

McInnes provides the derivation of this simple solar radiation pressure force in Chapter 2 of his book [35], reproduced here for reference. Assuming $A$ to be the area of the reflected surface and $P$ to be the pressure exerted on the surface by the SRP, the force exerted on the reflective surface due to the momentum from the incoming photons is:

$$f_s = PA(\hat{s}^T \hat{n})\hat{s}$$

(3.1)

where $\hat{s}$ and $\hat{n}$ are the components of the vectors $\vec{s}$ and $\vec{n}$ in the same frame, and the other terms to follow adhere to the same convention. The term $A(\hat{s}^T \hat{n})$ is the reflective surface projected in the direction $\hat{s}$. Similarly, the force exerted on the surface due to the reflected photons can be described by

$$f_r = -PA(\hat{s}^T \hat{n})\hat{r}$$

(3.2)

Using the vector identity $\hat{r} - \hat{s} = 2(\hat{s}^T \hat{n})\hat{n}$, the combined force exerted on the reflective surface is:

$$f_{\text{simple}} = -2PA(\hat{s}^T \hat{n})^2\hat{n}$$

(3.3)

Alternatively, using the definition of the dot product, i.e. $a^T b = |a||b|\cos \theta$, where $\theta$ is the angle between the vectors $a$ and $b$, and as the norm of a unit vector is 1, the above equation can be rewritten as

$$f_{\text{simple}} = -2PA \cos^2 \alpha \hat{n}$$

(3.4)
The pressure due to solar radiation, \( P \), is a value that is dependent upon the distance from the Sun. Specifically, it is represented by

\[
P = \frac{W_E}{c} \left( \frac{R_E}{r} \right)^2
\]  

(3.5)

where \( W_E \) is the measured power flux of photons from the Sun at distance \( R_E \), \( c \) is the speed of light, \( R_E \) is the average distance from the Sun to the Earth, and \( r \) is the distance from the Sun to the reflective surface. An accepted mean value for \( W_E \) is 1368 \( \text{J} \cdot \text{s}^{-1} \cdot \text{m}^{-2} \), hence at 1 AU away from the Sun, the pressure \( P \) is calculated to be \( 4.56 \times 10^{-6} \text{ N} \cdot \text{m}^{-2} \). However, to consider the conservative case of the sail being closer to the Sun and apply greater force onto the sail, a value of \( P = 1.8253 \times 10^{-5} \text{ N} \cdot \text{m}^{-2} \) is used. This corresponds to the case where the sail is located approximately 0.5 AU away from the Sun.

### 3.3 Realistic Solar Radiation Pressure Force

A realistic calculation of solar radiation pressure force takes into account the non-specularity of the material used to reflect the photons. Various authors have derived different models to calculate such realistic solar radiation pressure force, but the resulting forces can be divided into a normal component \( f_n \) and a transverse component \( f_t \), as shown in Figure 3.3. The following subsections describe a number of derivations for realistic solar radiation pressure force.

**Figure 3.3: Realistic Solar Radiation Pressure Force on a Flat Surface**

#### 3.3.1 Classical Realistic Solar Radiation Pressure Force

McInnes [35] provides a realistic solar radiation pressure force equation that combines forces due to incident photon momentum, reflection, and emission by re-radiation, i.e.

\[
f = f_s + f_r + f_e
\]  

(3.6)

Now, looking back at Figure 3.3, unit vectors \( \hat{s} \) and \( \hat{r} \) can be defined in terms of the normal and transverse vectors as follows:

\[
\hat{s} = -\cos \alpha \hat{n} + \sin \alpha \hat{t}
\]

(3.7)

\[
\hat{r} = \cos \alpha \hat{n} + \sin \alpha \hat{t}
\]
Chapter 3. Simple and Realistic Solar Radiation Pressure Forces and Torques

The force due to incident photon momentum was introduced in Equation (3.1) from the previous section. This equation, using the dot product identity that $\hat{s} \cdot \hat{n} = \cos \alpha$, can be rewritten in terms of the sun angle $\alpha$ and the vectors $\hat{n}$ and $\hat{t}$ as

$$f_s = PA(-\cos^2 \alpha \hat{n} + \cos \alpha \sin \alpha \hat{t})$$

(3.8)

where $P$ is the solar radiation pressure and $A$ is the area of the reflective surface as before. A fraction $\tilde{r}$ of the incident photons are reflected. Of these, a fraction $s$ are specularly reflected while the rest are diffusively reflected. The specularly-reflected components of the reaction force can be written, based on the simple reflective force presented previously in Equation (3.2) and substituting in Equation (3.7), as

$$f_{rs} = -\tilde{r}sPA(\cos^2 \alpha \hat{n} + \cos \alpha \sin \alpha \hat{t})$$

(3.9)

The diffusive component of the force is assumed to be uniformly scattered, resulting in a force in the direction opposite to the surface normal, written as

$$f_{rd} = -B_f \tilde{r}(1 - s)PA \cos \alpha \hat{n}$$

(3.10)

where $B_f$ is the non-Lambertian coefficient, describing the deviation from a surface that appears equally bright when viewed from any aspect angle. This expression effectively states that the sum of the diffusive force is a fraction of the reflective force that is not specularly reflected.

The absorbed photons are emitted back to the environ from both the front and the back of the sail, which causes reaction forces. The magnitude of force on a unit area due to emission can be calculated as $f = \epsilon \tilde{\sigma} T^4/c$ where $\epsilon$ is the surface emissivity, $\tilde{\sigma}$ is the Stefan-Boltzmann constant, $T$ is the area temperature and $c$ is the speed of light as before. Assuming that this force is also uniformly diffusive, points opposite to the surface normal and affected by the non-Lambertian coefficient, its equation can be written as

$$f_e = -\frac{\tilde{\sigma} T^4}{c}(\epsilon_f B_f - \epsilon_b B_b) \hat{n}$$

(3.11)

where the subscripts $f$ and $b$ denote coefficients specific to the front and the back of the surface. Furthermore, the surface temperature $T$ can be derived via thermal balance: Specifically, equate the thermal input $(1 - \tilde{r})W \cos \alpha$ to the thermal output $(\epsilon_f + \epsilon_b)\tilde{\sigma} T^4$, then substitute the radiation pressure $P = W/c$ in and rearrange to isolate $T$:

$$T = \left[\frac{(1 - \tilde{r})cP \cos \alpha}{\tilde{\sigma}(\epsilon_f + \epsilon_b)}\right]^\frac{1}{4}$$

(3.12)

Substituting the above definition for $T$ into Equation (3.11) and simplifying,

$$f_e = -(1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b} PA \cos \alpha \hat{n}$$

(3.13)

Finally, adding Equations (3.8), (3.9), (3.10) and (3.13) together, simplifying and separating the terms into normal and transverse components,

$$f_n = -PA \left\{ (1 + \tilde{r}s) \cos^2 \alpha + B_f(1 - s)\tilde{r} \cos \alpha + (1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b} \cos \alpha \right\} \hat{n}$$

$$f_t = PA(1 - \tilde{r}s) \cos \alpha \sin \alpha \hat{t}$$

(3.14)
3.3.2 Vector-Form of the Realistic Solar Radiation Pressure Force

The derivation provided by McInnes can be modified slightly to provide an expression composed of vectors. Specifically, consider Equations (3.1) and (3.2), and assume once again that only a fraction \( \tilde{r} \) of the incident photons is reflected and a fraction \( s \) of the reflected photons are specularly reflected. Using the vector identity \( \hat{r} = \hat{s} + 2(\hat{s}^T \hat{n})\hat{n} \), the two equations are summed and rearranged to be

\[
\mathbf{f}_s + \mathbf{f}_{rs} = PA((\hat{s}^T \hat{n})\hat{s} - \tilde{r}s(\hat{s}^T \hat{n})(\hat{s} + 2(\hat{s}^T \hat{n})\hat{n})) = PA(\hat{s}^T \hat{n})((1 - \tilde{r} s)\hat{s} - 2\tilde{r}s(\hat{s}^T \hat{n})\hat{n}) \tag{3.15}
\]

The forces due to diffusion and emission are both defined to have a force magnitude with the fraction of \( f_s \) and in the direction opposite to the surface normal. The vector expressions for these two forces can be derived easily by substituting \( \cos \alpha = \hat{s}^T \hat{n} \) into Equations (3.10) and (3.13), i.e.

\[
\mathbf{f}_{rd} = -B_f \tilde{r}(1 - s)PA(\hat{s}^T \hat{n})\hat{n} \\
\mathbf{f}_e = -(1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b} PA(\hat{s}^T \hat{n})\hat{n} \tag{3.16}
\]

Combining Equations (3.15) and (3.16) together, the following expression for vector-form realistic solar radiation pressure force is derived:

\[
\mathbf{f} = -PA(\hat{s}^T \hat{n}) \left\{ (2\tilde{r}s(\hat{s}^T \hat{n}) + B_f \tilde{r}(1 - s) + (1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b}) \hat{n} - (1 - \tilde{r} s)\hat{s} \right\} \tag{3.17}
\]

This form is useful when the direction of the sun is represented with the sun vector \( \hat{s} \). This is a surprisingly common situation as it is simpler to derive a rotation matrix from quaternion attitude representations, which is then used to rotate the inertial sun vector into the body frame before calculating the force.

3.3.3 The Linear Photonic Thrust Model

Greschik offers a simplification to the above expression by neglecting the diffusion and emission components of the forces [19]. Dubbed the Linear Photonic Thrust Model, this model can be derived by summing Equations (3.8) and (3.1) and separating the resulting terms into normal and transverse components,

\[
\mathbf{f}_n = -PA\eta_n \cos^2 \alpha \hat{n} \\
\mathbf{f}_t = PA\eta_t \cos \alpha \sin \alpha \hat{t} \tag{3.18}
\]

where \( \eta_n = (1 + \tilde{r} s) \) and \( \eta_t = (1 - \tilde{r} s) \). The similarity to Equation (3.14) is apparent. The vector form for this equation is already derived above in Equation (3.15). This model can be considered a compromise between the simple model and the realistic model in terms of thrust accuracy and computational simplicity, requiring only two of the six coefficients used for the realistic model.
3.4 Comparison of Generated Forces between Different Solar Radiation Pressure Forces

Plotted in Figure 3.4 are solar radiation pressure force magnitudes predicted by the above-introduced equations for a reflective surface the size of a unit area and at a distance of 1 AU from the Sun. Surface optical parameters used by McInnes for a square solar sail in Chapter 2 of his book [35] are used here as well: $\tilde{r} = 0.88$, $s = 0.94$, $\epsilon_f = 0.05$, $\epsilon_b = 0.55$, $B_f = 0.79$, and $B_e = 0.55$. The discrepancies between the simple force generated by Equation (3.4) and the other two forces are easily visible - the force magnitude in the surface normal direction is greater by up to 10% compared to the realistic normal force magnitude, and as well the simple model produces no transverse force. The transverse component of the realistic and linear models are equivalent, while the figure depicts the normal forces to be similar to each other.

Table 3.1 shows normal force magnitudes estimated to be produced by a unit-area of reflective surface at 1 AU away from the Sun when the Sun angle $\alpha = 0$, using the realistic and the linear models with varying optical parameters. As seen on the first line, the force estimated by the realistic model is 99% of the force estimated by the linear model for the initially-given optical parameters, in agreement with Figure 3.4.

<table>
<thead>
<tr>
<th>Description</th>
<th>Optical Parameters</th>
<th>SRP Normal Forces ($\mu$N/m$^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Square Sail</td>
<td>$\tilde{r}$ 0.88</td>
<td>$s$ 0.94  $\epsilon_f$ 0.05</td>
</tr>
<tr>
<td>Reduced Reflectivity</td>
<td>$r$ 0.70   $s$ 0.94</td>
<td>$\epsilon_f$ 0.05  $\epsilon_b$ 0.55</td>
</tr>
<tr>
<td>Reduced Specularity</td>
<td>$r$ 0.88 0.75</td>
<td>$s$ 0.94  $\epsilon_f$ 0.05</td>
</tr>
<tr>
<td>Reduced Refl. and Spec.</td>
<td>$r$ 0.70 0.75</td>
<td>$s$ 0.94  $\epsilon_f$ 0.05</td>
</tr>
<tr>
<td>Reduced Frontal Emissivity</td>
<td>$r$ 0.88 0.94</td>
<td>$s$ 0.94  $\epsilon_f$ 0.15</td>
</tr>
<tr>
<td>All of the above</td>
<td>$r$ 0.70 0.75</td>
<td>$s$ 0.94  $\epsilon_f$ 0.15</td>
</tr>
</tbody>
</table>

Table 3.1: Comparison of Realistic and Linear SRP Normal Forces with Varying Optical Parameters
The lines two through six are deviations from this initial set of parameters, provided by Greschik [20], which were conservatively derived based on extreme environmental degradations predicted by Dachwald, et al. [10]. Each set of degradations provides a different amount of deviation between the realistic and the linear case, with the Reduced Specularity case providing the greatest deviation factor of 1.07.

One can formulate the maximum difference between the realistic model and the linear model as an optimization problem as follows: first subtract the normal force equation for the linear model from the realistic model:

\[
\mathbf{f}_{n,ni} - \mathbf{f}_{n,l} = -PA \left\{ (1 + \tilde{r}s) \cos^2 \alpha + B_f(1 - s)\tilde{r}\cos \alpha + (1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b} \cos \alpha - (1 + \tilde{r}s) \cos^2 \alpha \right\} \hat{\mathbf{n}}
\]

\[
= -PA \left\{ B_f(1 - s)\tilde{r} + (1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b} \right\} \cos \alpha \hat{\mathbf{n}}
\]

Only the terms within the outer brackets depend on optical parameters, hence the optimization problem can be defined as the maximization of \( f \), where

\[
f = B_f(1 - s)\tilde{r} + (1 - \tilde{r}) \frac{\epsilon_f B_f - \epsilon_b B_b}{\epsilon_f + \epsilon_b}
\]

and the parameters are by definition bound between 0 and 1. While the equation is nonlinear, it is simple enough to be solved via commonly-available nonlinear optimization algorithms, such as Matlab’s \textit{fmincon} function. Several simple optimization problems can be solved for with this setup - for example, the worst case discrepancy scenario for the linear model compared to the realistic model can be determined. Using the maximum and the minimum parameters used in the above table as the bounds, maximizing \( f \) reveals that the reduced specularity case combined with the degraded frontal emissivity case produces the biggest discrepancy between the realistic model and the linear model. While a more thorough optimization analysis can be done for the optical parameters if one desires to take advantage of the linear model’s simplicity, no such need was found during the course of this research.

### 3.5 Calculation of Solar Radiation Pressure Forces and Torques on Sail FEM Model

The sail constantly reflects photons even as it billows from the force produced by the photons. As the sail shape changes, it is likely for the thrust generated by the sail to change as well, hence there is a need to calculate the SRP force produced by the sail at every step of the simulation. In terms of the finite element model, this involves quantifying the orientation of every sail membrane element and using it to calculate the force and the body torque produced by the sail as a whole.

Determination of the reflective sail’s orientation can be simplified by taking advantage of the shape of the membrane element. Specifically, assuming that each element surface is flat for the purpose of calculating the SRP force, the three vertices of a triangular element used in this research can describe a unique three dimensional plane, from which a constant normal vector can be derived to describe the orientation of the element as a whole. This trait is not true for any element with less than three vertices, for which there are an infinite number of normal vectors for any point on the element, or for any element with more than three vertices, for which any plane through all of its arbitrarily positioned vertices must be curvilinear.
Chapter 3. Simple and Realistic Solar Radiation Pressure Forces and Torques

The vector forms for solar radiation pressure force have already been introduced above in Equations (3.3), (3.17) and (3.15). It is assumed that the location of the sun in the inertial frame is known by the sun vector $\hat{s}_i$, and its body frame counterpart $\hat{s}_b = C_{bi}\hat{s}_i$, where $C_{bi}$ is the inertial to body frame rotation matrix and is known. The normal vector of a flat triangular element is simply a cross product of two vectors formed from the three vertices of the triangle. Specifically, letting $p_i$ be the location of the node $i$ of the triangular element in the body frame, that is,

$$p_i = \rho_i + u_i$$  \hspace{1cm} (3.21)

where $\rho_i$ is the original location of the node and $u_i$ is the deformation of the node, the normal vector in the body frame can be written as

$$n_b = (p_2 - p_1) \times (p_3 - p_1)$$  \hspace{1cm} (3.22)

$$\hat{n}_b = \frac{n_b}{\|n_b\|}$$

The transverse vector is essentially a normal vector projected onto the sail plane from the direction of the sun. Given a plane represented by a normal vector $\hat{n}_b$, the inertial coordinate of a point $p$ projected onto it in the direction of the sun vector $\hat{s}_b$ can be calculated by

$$p_{proj} = p + t\hat{s}_b, \quad t = -\frac{\hat{n}_b^T(p - p_p)}{\hat{n}_b^T \hat{s}_b}$$  \hspace{1cm} (3.23)

where $p_p$ is any known point on the plane. A projected vector essentially would be the difference between two such projected coordinates, i.e.

$$v = p_{proj,2} - p_{proj,1}$$  \hspace{1cm} (3.24)

This process can be greatly simplified for our purposes by realizing that since the sun vector $\hat{s}$ does not change in any significant manner by the position of the sail element on the body due to the distance from the Sun, the frame of reference can be translated such that the tail end of the normal vector is at the origin, rendering $p_{proj,1} = 0$. In addition, since the tail end of the normal vector is attached to the plane, it can also be used in place of $p_p$, rendering that term to 0 as well. Coordinates of the front-end of $\hat{n}_b$ are the components themselves, hence $p = \hat{n}_b$. Plugging in these values into Equations (3.23) and (3.24) then simplifying, a simple equation for the transverse vector is derived:

$$t_b = \hat{n}_b - \frac{1}{\hat{n}_b^T \hat{s}_b} \hat{s}_b$$ $$\hat{t}_b = \frac{t_b}{\|t_b\|}$$  \hspace{1cm} (3.25)

Once both the normal and the transverse vectors are known in the body frame, any of the Equations (3.3), (3.17) and (3.15) can be picked to derive the solar radiation pressure force generated by the membrane element in the body frame. The resulting force is substituted into the appropriate rigid and elastic body forcing components in the following chapter.
Chapter 4

Unconstrained Nonlinear Flexible Structural Dynamics

The goal of this chapter is to derive and develop a geometrically nonlinear Finite Element Model (FEM) simulation system to be used for simulating solar sail models. Derived from first principles, this is the backbone of the dynamics simulations used in this research, and a number of result comparisons to previous experiments are made in order to verify the validity of the implementation.

A typically-cited advantage for a nonlinear model is that it can offer faster convergence to the solution for a less-detailed model. However, here it is used to induce the coupled behaviour between the out-of-plane and the in-plane deflections of a membrane - how a membrane’s in-plane deflection behaves against out-of-plane forces, and vice versa. While using a linear elastic film element would cost less computational power to simulate better-detailed models, it would not capture this particular relationship. It is author’s hope that the combination of the nonlinear plate model in this chapter and the wrinkling model in Chapter 5 will provide a unique preliminary look into the coupled behaviour.

4.1 Deriving the Equation of Equilibrium with Geometrical Nonlinearities

We start from a constrained flexible body with undeformed volume $V$ as shown in Figure 4.1. Positions in the body are described by the vector components $\rho$ expressed in the body frame $\mathbf{F}_b$. A small mass element $dm$ originally located at $\rho$ is elastically displaced by some displacement field $u$, affected by some body force $B$. This body is also fixed on the surface at $S_1$, while the free parts of the surface $S_2$ are affected by surface traction $T$.

For convenience, let $\rho$, $u$, $B$ and $T$ be the component column matrices of the vectors $\rho$, $\mathbf{u}$, $\mathbf{B}$ and $\mathbf{T}$ all in the body frame, and let $\rho_i$, $u_i$, $B_i$ and $T_i$ be the respective components of the column matrices. We will also use $\rho_1 = x$, $\rho_2 = y$, and $\rho_3 = z$ for the spatial coordinates and $u_1 = u$, $u_2 = v$, and $u_3 = w$ for the displacements. Then the equation of total potential energy for the above-shown body, using Einstein’s summation notation, can be written as [45]:

$$\Pi = U - \iiint_V B_i u_i(\rho) \, dV - \iint_{S_1} T_i u_i(\rho) \, dS$$

(4.1)
where $U$ is the strain energy. Surface traction is assumed to be zero for simplicity, in which case the above equation can be rewritten as follows:

$$\Pi = U - \iiint_V f^T u \, dv$$

(4.2)

where $f$ is the column of body force distribution $B_i$ and $u$ is the column matrix of displacement fields $u_i$.

The strain energy equation is

$$U = \iiint_V \left[ \int_0^{\epsilon_{ij}} \tau_{ij} \, d\epsilon_{ij} \right] \, dV$$

(4.3)

where $\tau_{ij}$ and $\epsilon_{ij}$ are the stress and strain tensors respectively. Lagrangian stress and strain terms are assumed for this research. For a purely linear elastic structure, the relationship between stress and strain tensors can be described by the Hooke’s Law:

$$\tau_{ij} = C_{ijkl} \epsilon_{kl}$$

(4.4)

where $C_{ijkl}$ is the fourth-order tensor of elastic moduli. Using Equations (4.3) and (4.4), the strain energy expression can be written as

$$U = \frac{1}{2} \int_V \epsilon_{ij} C_{ijkl} \epsilon_{kl} \, dV.$$

(4.5)

The strain term can be expressed in the following matrix notation [42]

$$\epsilon_{ij} = L_{ij}^T d + \frac{1}{2} d^T H_{ij} d$$

(4.6)

where $L_{ij}$ is a column matrix depending only on the spatial coordinates, $H_{ij}$ is a constant symmetric matrix, and $d$ is a column matrix of displacement gradients contributing to the strains $\epsilon_{ij}$. To clarify, here is an example using a nonlinear strain equation for a beam:

$$\varepsilon = u_x - y v_{xx} - z w_{xx} + \frac{1}{2} v_x^2 + \frac{1}{2} w_x^2$$

(4.7)
where \( u_x, v_x, w_x, v_{xx}, \) and \( w_{xx} \) refer to first and second partial derivatives of the \( x, y \) and \( z \) displacement functions \( u, v \) and \( w \). In this case, define \( \mathbf{d} \) as follows:

\[
\mathbf{d} = \begin{bmatrix} u_x & v_x & w_x & v_{xx} & w_{xx} \end{bmatrix}^T
\]  

(4.8)

then comparing Equations (4.6) and (4.7), one can deduce that

\[
\mathbf{L} = \begin{bmatrix} 1 & 0 & 0 & -y & -z \end{bmatrix}^T
\]  

(4.9)

\[
\mathbf{H} = \text{diag} \left\{ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \end{bmatrix} \right\}
\]  

(4.10)

One can confirm the validity of above definitions for \( \mathbf{d}, \mathbf{L} \) and \( \mathbf{H} \) by substituting them into Equation (4.6), which simplifies to Equation (4.7) as desired.

Given the above definition for the strain equation, the strain energy can be rewritten as

\[
U = \frac{1}{2} \iiint_V \mathbf{C}_{ijkl} \mathbf{d}^T \left[ \mathbf{L}_{ij} \mathbf{L}_{kl}^T + \mathbf{L}_{ij} \mathbf{d}^T \mathbf{H}_{kl} + \frac{1}{4} \mathbf{H}_{ij} \mathbf{dd}^T \mathbf{H}_{kl} \right] \mathbf{d} \, dV
\]  

(4.11)

Before proceeding, define time-invariant matrices \( \mathbf{\Psi} \) and \( \mathbf{D} \) such that

\[
\mathbf{u} = \mathbf{\Psi} \mathbf{q}_e = \mathbf{\psi}_\alpha \mathbf{q}_\alpha
\]  

(4.12)

\[
\mathbf{d} = \mathbf{D} \mathbf{q}_e
\]  

(4.13)

where \( \mathbf{q}_e \) is the column matrix of generalized coordinates, \( \mathbf{q}_\alpha \) is the component of the generalized coordinate matrix, and \( \mathbf{\psi}_\alpha \) is the \( \alpha^{th} \) column of the matrix \( \mathbf{\Psi} \). Note that \( \mathbf{\Psi} \) depends on the spatial coordinates and is commonly called the shape function of the element. The relationship between \( \mathbf{\Psi} \) and \( \mathbf{D} \) is that of a derivative. Then Equation (4.11) can be further expanded as

\[
U = \frac{1}{2} \mathbf{q}_e^T \iiint_V \mathbf{C}_{ijkl} \mathbf{D}^T \left[ \mathbf{L}_{ij} \mathbf{L}_{kl}^T + \mathbf{L}_{ij} \mathbf{d}^T \mathbf{H}_{kl} + \frac{1}{4} \mathbf{H}_{ij} \mathbf{dd}^T \mathbf{H}_{kl} \right] \mathbf{D} \, dV \mathbf{q}_e
\]  

(4.14)

For a linear equation of motion for small strains and displacements, the second and third terms within the square brackets are ignored. In this case, the strain energy equation simplifies to

\[
U_L = \frac{1}{2} \mathbf{q}_e^T \iiint_V \mathbf{C}_{ijkl} \mathbf{D}^T \mathbf{L}_{ij} \mathbf{L}_{kl}^T \mathbf{D} \, dV \mathbf{q}_e
\]  

(4.15)

The principle of total potential energy states that the first variation of the total potential energy is zero - that is, the total potential energy is an extrema. Hence, taking the partial derivative of the linearized total potential energy \( \Pi_L \) with respect to the unknown generalized coordinates \( \mathbf{q}_e \),

\[
\frac{\partial \Pi_L}{\partial \mathbf{q}_e} = \iiint_V \mathbf{C}_{ijkl} \mathbf{D}^T \mathbf{L}_{ij} \mathbf{L}_{kl}^T \mathbf{D} \, dV \mathbf{q}_e - \iiint_V \mathbf{\Psi}^T \mathbf{f} \, dv = 0
\]  

(4.16)

or in a more familiar form,

\[
\mathbf{K}_L \mathbf{q}_e = \mathbf{F}
\]  

(4.17)

where \( \mathbf{K}_L = \iiint_V \mathbf{C}_{ijkl} \mathbf{D}^T \mathbf{L}_{ij} \mathbf{L}_{kl}^T \mathbf{D} \, dV \) is the stiffness matrix and \( \mathbf{F} = \iiint_V \mathbf{\Psi}^T \mathbf{f} \, dv \) contains the
generalized forces. Hence the equation of equilibrium for small strains and displacements is derived.

However for larger displacements - often stated in the literature as 'geometric nonlinearity', the second and third terms cannot be omitted. Extremizing the total potential energy with respect to $q_e$ becomes more involved in this case. In particular, unlike the linear case where the Jacobian of the equation is simply $K_L$, it is not immediately obvious what the Jacobian of this nonlinear case is, and this becomes important for simulation purposes later on in this thesis.

This problem can be resolved by choosing some specific structures for the terms in Equation (4.14), as demonstrated by Rajasekaran and Murray [42]. Specifically, assume that the strain energy equation can be described as follows:

$$U = q_e^T \iint_V D^T \left[ \frac{1}{2} \hat{K} + \frac{1}{6} \hat{N}_1(q_e) + \frac{1}{12} \hat{N}_2(q_e) \right] D dV q_e \tag{4.18}$$

where $\hat{K}$ is a linear expression of stiffness, $\hat{N}_1(q_e)$ is the first order nonlinear expression of stiffness, and $\hat{N}_2(q_e)$ is the second order nonlinear expression of stiffness. It should be noted that these matrices are not the typical stiffness matrices, but they can be transformed into one by pre- and post-multiplying them with $D^T$ and $D$ and performing the spatial integration.

The linear term expression is as follows:

$$\hat{K} = C_{ijkl} L_{ij} L_{kl}^T \tag{4.19}$$

which becomes the linear stiffness matrix mentioned above.

The first-order nonlinear term - dubbed as such due to its first order dependence on the generalized coordinates $q_e$ - has several possible expressions. However, after considering the symmetry of $C_{ijkl}$ and the repeatability of terms upon differentiation, the following expression is chosen:

$$\hat{N}_1(q_e) = C_{ijkl} (L_{ij} d^T H_{kl} + d^T L_{ij} H_{kl} + H_{ij} dL_{kl}^T) \tag{4.20}$$

For similar reasons, one second-order nonlinear term is chosen over others:

$$\hat{N}_2(q_e) = C_{ijkl} (H_{ij} dd^T H_{kl} + \frac{1}{2} d^T H_{kl} dH_{ij}) \tag{4.21}$$

The result of choosing these terms becomes obvious when taking their derivatives. In particular, one can show that:

$$\frac{\partial \Pi}{\partial q_e} = \left[ K_L + \frac{1}{2} N_1(q_e) + \frac{1}{3} N_2(q_e) \right] q_e - \iint_V \Psi^T f dV = 0 \tag{4.22}$$

where $K_L = \iint_V D^T \hat{K} D dV$, $N_1 = \iint_V D^T \hat{N}_1(q_e) D dV$, and $N_2 = \iint_V D^T \hat{N}_2(q_e) D dV$ are the stiffness matrices of varying orders. The above equilibrium equation is once again simplified to

$$K(q_e) q_e = F, \quad K(q_e) = K_L + \frac{1}{2} N_1(q_e) + \frac{1}{3} N_2(q_e) \tag{4.23}$$

except this time, $K$ has an explicit dependence on the generalized coordinates $q_e$, leading to complications when it comes to simulations, which will be discussed in later sections.

Finally, note that the Jacobian of the above is

$$J_K = K_L + N_1(q_e) + N_2(q_e) \tag{4.24}$$
potentially allowing computationally-cheap calculation of the Jacobian due to the choice of expressions \( \hat{K}, \hat{N}_1 \) and \( \hat{N}_2 \).

### 4.1.1 Deriving the Constrained Equations of Motion with Geometrical Nonlinearities

Starting from extended Hamilton’s principle, which states that, of all the paths of admissible configurations that the body can take as it goes from configuration 1 at time \( t_1 \) to configuration 2 at time \( t_2 \), the path that satisfies Newton’s law at each instant during the interval (and is thus the actual locus of configurations) is the path that extremizes the time integral of the Lagrangian during the interval, or in mathematical terms:

\[
\delta \int_{t_1}^{t_2} L \, dt = 0 \tag{4.25}
\]

where \( L = T - \Pi \) is the Lagrangian, \( T \) is the kinetic energy and \( \Pi \) is the total potential energy. The Lagrangian \( L \) that solves this problem is a solution of the Euler-Lagrange equation:

\[
\frac{d}{dt} \left( \frac{\partial L}{\partial \dot{q}_e} \right) - \frac{\partial L}{\partial q_e} = 0 \tag{4.26}
\]

Given the previous definition of \( u \) from Equation (4.12), the kinetic energy equation can be written as follows,

\[
T = \frac{1}{2} \iiint_V \dot{u}^T \ddot{u} \, dm = \frac{1}{2} \ddot{q}_e^T \iint_V \Psi^T \Psi \sigma \, dV \quad = \frac{1}{2} \ddot{q}_e^T M \dot{q}_e,
\]

and \( \Pi \) is known from the previous section as

\[
\Pi = \frac{1}{2} q_e^T \left[ K_L + \frac{1}{3} N_1 + \frac{1}{6} N_2 \right] q_e - \iiint_V f^T \Psi q_e \, dv
\]

Substituting in the above expressions to \( L \), and \( L \) to Equation (4.26) then solving,

\[
M \ddot{q}_e + K(q_e) q_e = F \tag{4.27}
\]

which is the familiar equations of motion for a flexible object. It should be noted once again that the stiffness matrix \( K \) depends on the generalized coordinates \( q_e \), leading to computational issues during simulation.

### 4.1.2 Adding Rigid Dynamics to the Equations of Motion

Figure 4.2 is an extension of Figure 4.1. The body frame is now floating freely, with its location described by \( \vec{r} \) in the inertial frame \( \vec{F}_i \). The body itself is assumed to be ‘fixed’ to the origin of the body frame.

Given the above figure, the absolute position of a deformed elastic body mass element \( dm \), \( \vec{R} \), can be described by

\[
\vec{R}(t) = \vec{r}(t) + \vec{p}(t) + \vec{u}(t) \tag{4.28}
\]
Given the above, the absolute velocity of \( dm \) can then be written as
\[
\dot{\mathbf{R}} = \mathbf{v} + (\dot{\mathbf{\omega}} + \mathbf{\omega} \times \mathbf{p}) + (\dot{\mathbf{u}} + \mathbf{\omega} \times \mathbf{u})
\] (4.29)

where \( \mathbf{v} \) and \( \mathbf{\omega} \) are the absolute velocity and angular velocity of the frame \( \mathbf{F}_b \) with respect to \( \mathbf{F}_i \). Here \( \dot{()} \) denotes the derivative as seen in the rotating body frame. Noting that \( \dot{\mathbf{p}} = 0 \), the absolute acceleration of \( dm \) can then be written as
\[
\ddot{\mathbf{R}} = (\ddot{\mathbf{v}} + \mathbf{\omega} \times \dot{\mathbf{v}}) + (\ddot{\mathbf{\omega}} \times \mathbf{p} + \mathbf{\omega} \times (\mathbf{\omega} \times \mathbf{p})) + (\ddot{\mathbf{u}} + 2\mathbf{\omega} \times \dot{\mathbf{u}} + \dot{\mathbf{\omega}} \times \mathbf{u} + \mathbf{\omega} \times (\mathbf{\omega} \times \mathbf{u}))
\] (4.30)

Expressing all vectors in terms of \( \mathbf{F}_b \), the above equation is rewritten in its component form as
\[
\ddot{\mathbf{R}} = (\ddot{\mathbf{v}} + \mathbf{\omega} \times \dot{\mathbf{v}}) + (\ddot{\mathbf{\omega}} \times \mathbf{p} + \mathbf{\omega} \times (\mathbf{\omega} \times \mathbf{p})) + (\ddot{\mathbf{u}} + 2\mathbf{\omega} \times \dot{\mathbf{u}} + \dot{\mathbf{\omega}} \times \mathbf{u} + \mathbf{\omega} \times (\mathbf{\omega} \times \mathbf{u}))
\] (4.31)

Now, by Newton’s second law of motion,
\[
\mathbf{f}_e = \ddot{\mathbf{R}} \, dm
\] (4.32)

where \( \mathbf{f}_e \) is a column of force upon the mass element. Using the following expression for \( \mathbf{u} \):
\[
\mathbf{u} = \psi_\alpha \mathbf{q}_\alpha
\] (4.33)

and expressing all vectors in terms of \( \mathbf{F}_b \), then taking an integral over the whole body and substituting in Equation (4.31) for \( \ddot{\mathbf{R}} \) gives
\[
\iiint_V \mathbf{f}_e \, dm = \iiint_V \left[ (\ddot{\mathbf{v}} + \mathbf{\omega} \times \dot{\mathbf{v}}) + (\ddot{\mathbf{\omega}} \times \mathbf{p} + \mathbf{\omega} \times (\mathbf{\omega} \times \mathbf{p})) + (\ddot{\mathbf{u}} + 2\mathbf{\omega} \times \dot{\mathbf{u}} + \dot{\mathbf{\omega}} \times \mathbf{u} + \mathbf{\omega} \times (\mathbf{\omega} \times \mathbf{u})) \right] \, dm
\] \Rightarrow \mathbf{F} = (m\ddot{\mathbf{v}} + m\mathbf{\omega} \times \mathbf{v}) + (-c\dot{\mathbf{\omega}} - \omega \times c \mathbf{\omega}) + (P_\alpha \ddot{\mathbf{q}}_\alpha + 2\omega \times \psi_\alpha \dot{\mathbf{q}}_\alpha + \dot{\mathbf{\omega}} \times \psi_\alpha \mathbf{q}_\alpha + \mathbf{\omega} \times \mathbf{\omega} \times \psi_\alpha \mathbf{q}_\alpha)
\] \Rightarrow \mathbf{F} = m\ddot{\mathbf{v}} - (c + \delta c) \dot{\mathbf{\omega}} + m\mathbf{\omega} \times \mathbf{v} - \omega \times (c + \delta c) \mathbf{\omega} + P_\alpha \ddot{\mathbf{q}}_\alpha + 2\omega \times P_\alpha \dot{\mathbf{q}}_\alpha
\] (4.34)
\[ F = \iiint_V f_e \, dm, \quad m = \iiint_V \rho \, dm, \quad c = \iiint_V \psi_\alpha \, dm, \quad P_\alpha = \iiint_V \psi_\alpha \, dm, \quad \delta c = P_\alpha q_\alpha \] (4.35)

It should be noted that the \( \delta \) notation denotes nonlinear terms from this point onward.

Similarly, cross multiplying both sides of Equation (4.32) with \( (\rho + u) \) gives

\[ (\rho + u) \times \vec{R} = (\rho + u) \times f_e \] (4.36)

Once again expressing all vectors in \( \mathbf{F}_b \), taking an integral over the whole body and substituting in Equation (4.31),

\[ \iiint_V (\rho + u)^x f_e \, dm = \iiint_V ((\rho + u)^x[(\dot{v} + \omega^x v) + (\omega^x p + \omega^x \omega^x p)] + (\ddot{u} + 2\omega^x \dot{u} + \dot{\omega}^x u + \omega^x \omega^x u)) \, dm \]

Hence,

\[ \iiint_V (\rho^x f_e + \psi_\alpha^x q_\alpha f_e) \, dm = \iiint_V (\rho^x \dot{v} + \psi_\alpha^x q_\alpha \dot{v} + \rho^x \omega^x v + \psi_\alpha^x q_\alpha \omega^x v) \, dm \]

\[ + \iiint_V (\rho^x \omega^x p + \psi_\alpha^x q_\alpha \omega^x p + \rho^x \omega^x \omega^x p + \psi_\alpha^x q_\alpha \omega^x \omega^x p) \, dm \]

\[ + \iiint_V (\rho^x \psi_\alpha^x q_\alpha + \psi_\alpha^x q_\alpha \psi_\beta q_\beta + 2\rho^x \omega^x \psi_\alpha^x q_\alpha + 2\psi_\alpha^x q_\alpha \omega^x \psi_\beta q_\beta) \, dm \]

\[ + \iiint_V (\rho^x \psi_\alpha^x q_\alpha + \psi_\alpha^x q_\alpha \psi_\beta q_\beta + 2\rho^x \omega^x \psi_\alpha^x q_\alpha + 2\psi_\alpha^x q_\alpha \omega^x \psi_\beta q_\beta) \, dm \]

which gives

\[ \mathbf{G} + \delta \mathbf{G} = \iiint_V (\rho^x \dot{v} + \psi_\alpha^x q_\alpha \dot{v} + \rho^x \omega^x v + \psi_\alpha^x q_\alpha \omega^x v) \, dm \]

\[ + \iiint_V (\rho^x \omega^x p + \psi_\alpha^x q_\alpha \omega^x p + \rho^x \omega^x \psi_\alpha^x q_\alpha + \psi_\alpha^x q_\alpha \psi_\beta q_\beta) \, dm \]

\[ + \iiint_V (\rho^x \psi_\alpha^x q_\alpha + \psi_\alpha^x q_\alpha \psi_\beta q_\beta + 2\rho^x \omega^x \psi_\alpha^x q_\alpha + 2\psi_\alpha^x q_\alpha \omega^x \psi_\beta q_\beta) \, dm \]

where

\[ \mathbf{G} = \iiint_V \rho^x f_e \, dm, \quad \delta \mathbf{G} = \iiint_V \psi_\alpha^x f_e \, dm \] (4.37)

Now the first set of integrals can be simplified with relative ease:

\[ \iiint_V (\rho^x \dot{v} + \psi_\alpha^x q_\alpha \dot{v} + \rho^x \omega^x v + \psi_\alpha^x q_\alpha \omega^x v) \, dm \]

\[ = \iiint_V ((\rho^x + \psi_\alpha^x q_\alpha) \dot{v} + (\rho^x + \psi_\alpha^x q_\alpha) \omega^x v) \, dm \]

\[ = (c + \delta c)^x \dot{v} + (c + \delta c)^x \omega^x v \] (4.38)

where the expressions \( c \) and \( \delta c \) are as defined previously from Equation (4.35). The second set of
integrals are also trivial to simplify:

\[
\int\int\int_V (\rho^\times \dot{\omega}^\times \rho + \psi_\alpha^\times q_\alpha \omega^\times \rho + \rho^\times \dot{\omega}^\times \psi_\alpha q_\alpha + \psi_\alpha^\times q_\alpha \omega^\times \psi_\beta q_\beta) \, dm
= \int\int\int_V (-\rho^\times \rho^\times - \psi_\alpha^\times \rho^\times q_\alpha - \rho^\times \psi_\alpha^\times q_\alpha - \psi_\alpha^\times \psi_\beta^\times q_\alpha q_\beta) \, dm
= \mathbf{J} + (-\tau_\alpha + \tau_\alpha^T) q_\alpha - \Upsilon_{\alpha\beta} q_\alpha q_\beta)
\]

(4.39)

where

\[
\mathbf{J} = \int\int\int_V -\rho^\times \rho^\times \, dm, \quad \tau_\alpha = \int\int\int_V \psi_\alpha^\times \rho^\times \, dm,
\]

(4.40)

\[
\Upsilon_{\alpha\beta} = \int\int\int_V \psi_\alpha^\times \psi_\beta^\times \, dm, \quad \delta \mathbf{J} = (-\tau_\alpha + \tau_\alpha^T) q_\alpha - \Upsilon_{\alpha\beta} q_\alpha q_\beta
\]

The \( \mathbf{J} \) term is the familiar second moment of mass, while the remaining expressions can be considered the nonlinear terms of \( \mathbf{J} \) introduced by the elastic body. The third set of integrals requires a knowledge about two identities of cross matrices, specifically,

\[
\omega^\times v^\times \omega = (\omega^T \omega) v - (\omega^T v) \omega
\]

(4.41)

and

\[
\rho \rho^T = \rho^\times \rho^\times + \rho^T \rho 1
\]

(4.42)

Applying these identities to the first expression of the third set of integrals gives

\[
\rho^\times \omega^\times \omega^\times \rho \\
= -\rho^\times \omega^\times \rho^\times \omega \\
= -\rho^\times ((\omega^T \omega) \rho - (\omega^T \rho) \omega) \\
= -\rho^\times (\omega^T \omega) \rho - \omega^\times \rho \rho^T \omega \\
= -\omega^\times (\rho^\times \rho^\times + \rho^T \rho) \omega \\
= -\omega^\times \rho^\times \rho^\times \omega - \omega^\times \omega^\times (\rho^T \rho) \\
= -\omega^\times \rho^\times \rho^\times \omega
\]

Using a similar procedure, the third set of integrals can be simplified as follows:

\[
\int\int\int_V (\rho^\times \omega^\times \omega^\times \rho + \psi_\alpha^\times q_\alpha \omega^\times \rho + \rho^\times \omega^\times \psi_\alpha q_\alpha + \psi_\alpha^\times q_\alpha \omega^\times \psi_\beta q_\beta) \, dm
= \int\int\int_V (-\omega^\times \rho^\times \omega - \omega^\times \psi_\alpha^\times q_\alpha \omega - \omega^\times \rho^\times \psi_\alpha q_\alpha \omega - \omega^\times \psi_\alpha^\times \psi_\beta^\times q_\alpha q_\beta \omega) \, dm
= \omega^\times (\mathbf{J} + (-\tau_\alpha + \tau_\alpha^T) q_\alpha - \Upsilon_{\alpha\beta} q_\alpha q_\beta)
= \omega^\times (\mathbf{J} + \delta \mathbf{J}) \omega
\]

(4.43)

One can immediately tell the similarity of the above-expression with that of the final expression in (4.39).
Finally, the last set of integrals can be simplified as follows:

\[ \begin{align*}
\int\int\int_V (\rho^x \psi_\alpha \dot{q}_\alpha + \psi_\alpha \alpha \omega^x \psi_\beta \dot{q}_\beta + 2\rho^x \omega^x \psi_\alpha \dot{q}_\alpha + 2\psi_\alpha q_\alpha \omega^x \psi_\beta \dot{q}_\beta) \, dm \\
= \int\int\int_V [\rho^x \psi_\alpha - \psi_\alpha q_\beta \omega^x] \dot{q}_\alpha - (2\rho^x \psi_\alpha - 2\psi_\beta \psi_\alpha q_\beta) \omega^x \dot{q}_\alpha \, dm \\
= (H_\alpha + \nu_{\alpha\beta} q_\beta) \dot{q}_\alpha - 2(\tau_\alpha + \chi_{\alpha\beta} q_\beta)^T \omega_\alpha \\
= (H_\alpha + \delta H_\alpha) \dot{q}_\alpha - 2 \dot{\tau}_\alpha \omega_\alpha
\end{align*} \]

where

\[ \begin{align*}
H_\alpha &= \int\int\int_V \rho^x \psi_\alpha \, dm, \quad \delta H_\alpha = \nu_{\alpha\beta} q_\beta, \quad \dot{\tau}_\alpha = \tau_\alpha + \chi_{\alpha\beta} q_\beta \\
\end{align*} \]

Putting together the results of Equations (4.38), (4.39), (4.43) and (4.44),

\[ \mathbf{G} + \delta \mathbf{G} = (c + \delta c)^x \dot{v} + (c + \delta c)^x \omega^x \dot{v} + (J + \delta J) \omega + \omega^x (J + \delta J) \omega + (H_\alpha + \delta H_\alpha) \dot{q}_\alpha - 2 \dot{\tau}_\alpha \omega_\alpha \]

The equilibrium of the generalized elastic forces can be described as follows:

\[ \int\int\int_V \psi_\alpha^T \mathbf{f}_c \, dm = \int\int\int_V \psi_\alpha^T \mathbf{R} \, dm + f_\alpha, \quad \alpha = 1, 2, 3, \ldots \]

where \( f_\alpha \) refers to the generalized elastic stiffness force associated with the shape function \( \psi_\alpha \), which shall be dealt with later. First, expand the above using expressions for \( \mathbf{R} \) from Equation (4.31):

\[ \begin{align*}
\int\int\int_V \psi_\alpha^T \mathbf{f}_c \, dm &= \int\int\int_V \psi_\alpha^T [\dot{\mathbf{v}} + \omega^x \mathbf{v} + (\dot{\omega}^x + \omega^x \omega^x) \mathbf{p}] + (\dot{\mathbf{u}} + 2\omega^x \dot{\mathbf{u}} + \omega^x \mathbf{u} + \omega^x \omega^x \mathbf{u}) \, dm + f_\alpha \\
&= \int\int\int_V (\psi_\alpha^T \dot{\mathbf{v}} + \psi_\alpha^T \omega^x \mathbf{v} - \psi_\alpha^T \dot{\mathbf{p}} + \omega^x \psi_\alpha^T \mathbf{p}^x \, dm \\
&+ \int\int\int_V (\psi_\alpha^T \omega_{\alpha\beta} \dot{q}_\beta - 2\psi_\alpha^T \omega_{\beta\alpha} q_\beta \omega_{\beta\alpha} + \omega^x \psi_\alpha^T \psi_\beta \, dm + f_\alpha \\
&= \mathbf{P}_\alpha^T \dot{\mathbf{v}} + \mathbf{P}_\alpha^T \omega^x \mathbf{v} + H_\alpha^T \omega + \omega^T \tau_\alpha \omega + \mathbf{M}_{\alpha\beta} \dot{q}_\beta - 2\nu_{\beta\alpha} \omega_{\beta\alpha} q_\beta + \mathbf{P}_\alpha^T \omega_{\alpha\beta} \dot{q}_\beta + \mathbf{P}_\alpha^T \omega^x \mathbf{v} + \omega^T \dot{\tau}_\alpha \omega - 2 \nu_{\beta\alpha} \omega_{\beta\alpha} q_\beta + f_\alpha
\end{align*} \]

where simplifying matrices have been defined previously in Equations (4.35), (4.40), and (4.45). Now define the following matrices:

\[ \mathbf{M}_{re} = \begin{bmatrix} \text{row } \{\mathbf{P}_\alpha\} \\ \text{row } \{H_\alpha + \delta H_\alpha\} \end{bmatrix}, \quad \mathbf{M}_{ee} = \text{matrix } \{M_{\alpha\beta}\} \]

with the following forcing components:

\[ \begin{align*}
\mathbf{f}_{T,e} &= \text{col } \left\{ \int\int\int_V \psi_\alpha^T \mathbf{f}_c \, dm \right\}, \quad \mathbf{f}_{I,e} = \text{col } \left\{ -\mathbf{P}_\alpha^T \omega^x \mathbf{v} - \omega^T \dot{\tau}_\alpha \omega \right\}, \\
\delta \mathbf{f}_{I,e} &= \text{col } \left\{ 2\nu_{\beta\alpha} \omega_{\beta\alpha} q_\beta \right\}, \quad \mathbf{F}_K = \text{col } \{f_\alpha\}
\end{align*} \]

and letting \( \boldsymbol{\xi} = [\mathbf{v}^T \omega^T]^T \), Equation (4.48) can be rewritten as

\[ \mathbf{M}_{re} \dot{\boldsymbol{\xi}} + \mathbf{M}_{ee} \dot{\mathbf{q}}_e + \mathbf{F}_K = \mathbf{f}_{T,e} + \mathbf{f}_{I,e} + \delta \mathbf{f}_{I,e} \]
where \( q_e \) is the column of generalized coordinates \( q_\alpha \) as before. The form of \( F_K \) can be obtained by employing Castigliano’s First Theorem to the strain energy given by Equation (4.18) and expanding:

\[
F_K = \frac{\partial U}{\partial q_e} = \left[ K_L + \frac{1}{2} N_1 + \frac{1}{3} N_2 \right] q_e
\]

where the form of the right-hand side can be recognized from Equation (4.23). Now, defining the following matrix:

\[
M_{rr} = \begin{bmatrix}
    m & -(c + \delta c)^X \\
    (c + \delta c)^X & J + \delta J
\end{bmatrix}
\]  

(4.53)

and the following forces:

\[
f_{T,r} = \begin{bmatrix} F \\ G \end{bmatrix}, \quad \delta f_{T,r} = \begin{bmatrix} 0 \\ \delta G \end{bmatrix},
\]

\[
f_{I,r} = \begin{bmatrix}
    -m\omega^X v + \omega^X (c + \delta c)^X \omega \\
    -(c + \delta c)^X \omega^X v - \omega^X (J + \delta J) \omega
\end{bmatrix}, \quad \delta f_{I,r} = \begin{bmatrix}
    -2\omega^X P_\alpha \dot{q}_\alpha \\
    2\dot{T}_\alpha \omega \dot{q}_\alpha
\end{bmatrix}
\]  

(4.54)

Equations (4.34) and (4.46) can be combined and rearranged in a form similar to Equation (4.51):

\[
M_{rr} \ddot{\xi} + M_{re} \ddot{q}_e = f_{T,r} + \delta f_{T,r} + f_{I,r} + \delta f_{I,r}
\]  

(4.55)

Finally, putting together Equations (4.51) and (4.55),

\[
\begin{bmatrix}
    M_{rr} & M_{re} \\
    M_{re}^T & M_{ee}
\end{bmatrix}
\begin{bmatrix}
    \ddot{\xi} \\
    \ddot{q}_e
\end{bmatrix}
+ \begin{bmatrix} 0 & 0 \\ 0 & K_{ee} \end{bmatrix}
\begin{bmatrix}
    q_e
\end{bmatrix} = \begin{bmatrix}
    f_{T,r} + \delta f_{T,r} \\
    f_{I,r} + \delta f_{I,r}
\end{bmatrix}
\]  

(4.56)

where \( \dot{q} = [\xi^T \quad \dot{q}_e^T]^T \). The motion equation has a similar form to that of the constrained one from Equation (4.27), but with state dependencies on both the mass matrix \( M \) and the forces \( f_T \) and \( f_I \). The state dependencies on \( K \) remain as well. Note that because \( \omega \) is not integrable, there is no appropriate term for the first six states of \( q \) and hence it is omitted here.

### 4.1.3 Kinematics of the Motion

In addition to the dynamic equation, there is a need for kinematic equations to provide not only their rates, but rigid body displacements over time as well. The Cartesian displacement \( r \) and Euler Parameters \( q_n \) are used to describe the kinematics of the system. The subscript \( n \) is used to avoid confusion associated with using \( q \) to denote Euler Parameters and generalized coordinates alike. The differential equations for these quantities can be defined as follows:

\[
\dot{q}_n = \begin{bmatrix}
    \dot{\epsilon}_n \\
    \dot{\eta}_n
\end{bmatrix} = \frac{1}{2} \left[ (\epsilon_n + \eta_n) 1_3 \right] \omega
\]

(4.57)

\[
\dot{r} = \left[ (1 - 2\epsilon_n^T \epsilon_n) 1_3 + 2\epsilon_n \epsilon_n^T - 2\eta_n \epsilon_n^X \right] v
\]

These equations are solved alongside the equations of motion during simulation to provide translational and rotational information about the structure.
4.1.4 Calculation of Damping Matrix for Nonlinear Equations of Motion

There are several ways to generate a damping matrix, but here we use two very simple linear damping methods for the purpose of stabilizing the simulation. The first damping method is a diagonal decoupled system damping, used for verifying the static results. For the first method, assume that the system equations are decoupled, i.e.

\[ M_{ii} \ddot{q}_i + D_{ii} \dot{q}_i + K_{ii}q_i = 0 \]  \hspace{1cm} (4.58)

for all states \( i \). Note that \( M_{ii} \) and \( K_{ii} \) refer to the diagonal components of the constant matrices \( M_{ee} \) and \( K_L \), i.e. consisting only of the elastic portion. The above-form can be rearranged into the following form:

\[ \ddot{q}_i + 2\zeta\omega_0 \dot{q}_i + \omega_0^2 q_i = 0 \]  \hspace{1cm} (4.59)

a second order differential system with well-known properties - in this case, \( \omega_0 \) is the natural frequency, and \( \zeta \) is the damping ratio. Comparing the two equations, one can easily deduce that the natural frequency \( \omega_0 = \sqrt{\frac{K_{ii}}{M_{ii}}} \), hence

\[ D_{ii} = 2\zeta\omega_0 M_{ii} = 2\zeta \sqrt{\frac{K_{ii}}{M_{ii}}} \]  \hspace{1cm} (4.60)

i.e. for a decoupled system of equations, its damping matrix can be written as a diagonal matrix of \( D_{ii} \).

However, the system being simulated is not decoupled, and it is unclear as to whether it’s even diagonally dominant. That being said, the purpose of the damping matrix in this simulation is to stabilize the system to a point where it can be simulated. Given the nonlinear nature of \( K \), it would be highly costly to try to damp specific natural frequencies of the system, since \( K \)’s eigenvalues and eigenvectors will need to be recalculated every time the state changes. Even the act of recalculating \( K \) every time the state the changes is costly.

For the above-stated reasons, the damping matrix used is a simple constant diagonal matrix of the above form. This damping allows the simulation to enter steady state in a stable converging manner, whereas without a damping matrix or with other matrices with dependencies on the state, the system either oscillates or simply does not converge to a solution.

The other damping method is the standard Rayleigh damping, i.e.

\[ D = \alpha M(0) + \beta K(0) \]  \hspace{1cm} (4.61)

where \( \alpha \) and \( \beta \) are the damping parameters. The sail simulations are unstable and often not convergent when the diagonal damping is used. For this reason the sail simulations are performed with the Rayleigh damping exclusively. This choice of damping comes with an added bonus of simulations performing relatively faster.

4.2 Equations of Motion with Geometrical Nonlinearities for Beams

While the general forms of the equations of motion have been derived, the stiffness matrices forming the equations of motion have not been defined. Here, these matrices are derived for a beam element, expanding on the derivation presented by Rajasekaran and Murray [42]. The displacement of a beam can be described by three displacement functions, \( u(x, q_e) \), \( v(x, q_e) \) and \( w(x, q_e) \) as shown in Figure 4.3.
Define these displacement functions as follows:

\[ u(x, q_e) = U(x)q_x = U_\alpha q_{x,\alpha}, \quad v(x, q_e) = V(x)q_y = V_\alpha q_{y,\alpha}, \quad w(x, q_e) = W(x)q_z = W_\alpha q_{z,\alpha} \]  

(4.62)

where \( U, V \) and \( W \) are the shape functions and \( U_\alpha, V_\alpha, \) and \( W_\alpha \) are their components. Likewise, \( q_x, q_y, \) and \( q_z \) are the generalized coordinates associated with each displacement function while \( q_{x,\alpha}, q_{y,\alpha}, \) and \( q_{z,\alpha} \) are their components. Letting

\[
\psi = \begin{bmatrix} U & 0 & 0 \\ 0 & V & 0 \\ 0 & 0 & W \end{bmatrix}, \quad q_e = \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix}
\]  

(4.63)

with \( \psi_\alpha \) and \( q_\alpha \) as the components of \( \psi \) and \( q_e \), the equations of motion as given in Equation (4.56) can be derived in a straightforward manner, with the exception of \( K_{ee} \), which requires a form of the strain term to be derived.

For a beam, one can show that the maximum magnitude of the shear stress \( \tau_{xz} \) is proportional to the normal stress \( \tau_{xx} \) by \( \frac{h}{L} \), where \( h \) and \( L \) are the thickness and the length of the beam. Similarly, one can also show that the normal stress \( \tau_{zz} \) is proportional to the normal stress \( \tau_{xx} \) by \( \left( \frac{h}{L} \right)^2 \). Similar logic renders the terms \( \tau_{yy} \) and \( \tau_{xy} \) insignificant, and by extension \( \tau_{yz} \). Hence the only significant stress term is \( \tau_{xx} \), which is described entirely by the strain term \( \epsilon_{xx} \).

The strain term for a nonlinear beam can then be written as follows:

\[
\epsilon_{xx} = u_x - yu_{xx} - zw_{xx} + \frac{1}{2}u_x^2 + \frac{1}{2}w_x^2
\]  

(4.64)

where the subscripts \( x \) and \( xx \) denote the first and the second derivatives of a function.

Back in Section 4.1, an alternate form for the strain term was introduced - Equation (4.6), which is reproduced below for the beam element case:

\[
\epsilon_{xx} = \mathbf{L}^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \mathbf{H} \mathbf{d}
\]  

(4.65)

where the terms \( \mathbf{d}, \mathbf{L} \) and \( \mathbf{H} \) for the nonlinear beam strain were given in Equations (4.8), (4.9), and
The stress-strain relationship for a beam is simple, namely

\[ \tau_{xx} = E \varepsilon_{xx} \]  

(4.66)

where \( E \) is Young’s Modulus, hence \( C_{xxxx} = E \). Finally, write out matrix \( D \) defined in Equation (4.13) as follows:

\[
\mathbf{d} = \mathbf{Dq} = \begin{bmatrix}
U_x & 0 & U_x & 0 \\
V_x & 0 & V_x & 0 \\
W_x & 0 & W_x & 0 \\
v_{xx} & 0 & v_{xx} & 0 \\
w_{xx} & 0 & w_{xx} & 0
\end{bmatrix}
\begin{bmatrix}
q_x \\
q_y \\
q_z \\
q_{xx}
\end{bmatrix}
\]

(4.67)

where \( U_{x,\alpha}, V_{x,\alpha}, W_{x,\alpha}, V_{xx,\alpha}, \) and \( W_{xx,\alpha} \) are the components of the respective shape function derivatives. Using these definitions, the stiffness matrix expressions can be derived. The linear term is expanded as follows:

\[
\mathbf{K}_L = \int_V \mathbf{D}^T \mathbf{C}_{ijkl} \mathbf{L}_{ij} L_{kl}^T \mathbf{D} \, dV
\]

\[ = E \int_V \mathbf{D}^T \mathbf{LL}^T \mathbf{D} \, dV
\]

\[ = \text{diag} \left\{ \left[ EA \int_0^l U_x^T U_x \, dx \quad EI \int_0^l V_{xx}^T V_{xx} \, dx \quad EI \int_0^l W_{xx}^T W_{xx} \, dx \right] \right\}
\]

(4.68)

\[ = \int_0^l \begin{bmatrix}
\text{matrix} \{ EAU_{x,\alpha}U_{x,\beta} \} & 0 & 0 \\
0 & \text{matrix} \{ EIV_{xx,\alpha}V_{xx,\beta} \} & 0 \\
0 & 0 & \text{matrix} \{ EIW_{xx,\alpha}W_{xx,\beta} \}
\end{bmatrix} \, dx
\]

where an assumption of symmetry about the x-axis has been made, resulting in constant beam cross sectional area \( A \) and area moment of inertia \( I \), as well as a constant \( E \). Note that this matrix is a constant - it can be pre-calculated prior to simulation.

The first-order nonlinear geometric stiffness matrix \( \mathbf{N}_1 \) is expanded as follows:

\[
\mathbf{N}_1 = \int_V \mathbf{D}^T \mathbf{C}_{ijkl} (\mathbf{L}_{ij} \mathbf{d}^T \mathbf{H}_{kl} + \mathbf{d}^T \mathbf{L}_{ij} \mathbf{H}_{kl} + \mathbf{H}_{ij} \mathbf{d} \mathbf{L}_{kl}^T) \mathbf{D} \, dV
\]

\[ = E \int_V \mathbf{D}^T \mathbf{d} \mathbf{L}^T \mathbf{HD} + \mathbf{D}^T \mathbf{d} \mathbf{L}^T \mathbf{HD} + \mathbf{D}^T \mathbf{H} \mathbf{d} \mathbf{L}^T \mathbf{D} \, dV
\]

\[ = EA \int_0^l \begin{bmatrix}
0 & U_x^T V_x V_x & U_x^T W_x w_x \\
V_x^T U_x V_x & 0 & V_x^T V_x u_x \\
W_x^T U_x w_x & 0 & W_x^T W_x u_x
\end{bmatrix} \, dx
\]

(4.69)

\[ = EA \int_0^l \begin{bmatrix}
\text{matrix}_{\alpha \beta} \{ U_{x,\alpha} V_{x,\beta} V_{x,\gamma} \} & \text{matrix}_{\alpha \beta} \{ U_{x,\alpha} W_{x,\beta} W_{x,\gamma} \} & 0 \\
\text{matrix}_{\alpha \beta} \{ V_{x,\alpha} V_{x,\beta} V_{x,\gamma} \} & \text{matrix}_{\alpha \beta} \{ V_{x,\alpha} W_{x,\beta} W_{x,\gamma} \} & 0 \\
\text{matrix}_{\alpha \beta} \{ W_{x,\alpha} W_{x,\beta} W_{x,\gamma} \} & 0 & \text{matrix}_{\alpha \beta} \{ W_{x,\alpha} W_{x,\beta} W_{x,\gamma} \}
\end{bmatrix} \, dx
\]

where the last set of expressions is written out so as to demonstrate the exact form of each component in the matrix. In particular, every matrix component’s first order dependence on \( \mathbf{q} \) means that the matrix pre-calculation can only be performed to a certain degree. In particular, the components of cubic order must be stored after the pre-calculation process and used during the simulation. The generation of
\( N_1 \) requires multiplication of these components to components of \( q_r \), which means that unlike a linear simulation which uses a constant stiffness matrix, there is a cubic order calculation process involved as well.

The second-order nonlinear geometric stiffness matrix \( N_2 \) is expanded as follows:

\[
N_2 = \iiint_V D^T C_{ijkl} (H_{ij} dd^T H_{kl} + \frac{1}{2} d^T H_{ij} d H_{kl}) dV
= E \iiint_V D^T Hdd^T H dV + \frac{1}{2} D^T d^T H d dV \tag{4.70}
= EA \int_0^l \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & \frac{3}{2} V^T_x V_x \nu_x^2 + \frac{1}{2} V^T_x V_x \nu_x^2 & V^T_x W_x \nu_x \nu_x & 0 \\
0 & W^T_x V_x \nu_x \nu_x & 3 \frac{1}{2} W^T_x W_x \nu_x^2 + \frac{1}{2} W^T_x W_x \nu_x^2 & 0
\end{bmatrix} dx
\]

where

\[
\frac{3}{2} V^T_x V_x \nu_x^2 + \frac{1}{2} V^T_x V_x \nu_x^2 = \text{matrix} \left\{ \frac{3}{2} V_{x,\alpha} V_{x,\beta} V_{x,\gamma} V_{x,\alpha} q_{y,\gamma} q_{y,\gamma} + \frac{1}{2} V_{x,\alpha} V_{x,\beta} W_{x,\gamma} W_{x,\alpha} q_{z,\gamma} q_{z,\gamma} \right\}
\]

\[
V^T_x W_x \nu_x \nu_x = \text{matrix} \left\{ V_{x,\alpha} W_{x,\beta} V_{x,\gamma} W_{x,\alpha} q_{y,\gamma} q_{y,\gamma} \right\}
\]

\[
W^T_x V_x \nu_x \nu_x = \text{matrix} \left\{ W_{x,\alpha} V_{x,\beta} V_{x,\gamma} W_{x,\alpha} q_{y,\gamma} q_{y,\gamma} \right\}
\]

\[
\frac{3}{2} W^T_x W_x \nu_x^2 + \frac{1}{2} W^T_x W_x \nu_x^2 = \text{matrix} \left\{ \frac{3}{2} W_{x,\alpha} W_{x,\beta} W_{x,\gamma} W_{x,\alpha} q_{y,\gamma} q_{y,\gamma} + \frac{1}{2} W_{x,\alpha} W_{x,\beta} V_{x,\gamma} V_{x,\alpha} q_{y,\gamma} q_{y,\gamma} \right\}
\tag{4.71}
\]

Once again, the last set of expressions demonstrate the exact forms of each component of the nonlinear matrix. In this case, the quadratic dependence on \( q_r \) means that the component storage and the calculation of the nonlinear matrix are both of quartic order. This is the crux of the simulation - this is where majority of the computation time for the nonlinear stiffness matrix is spent. The expressions for \( K_L \), \( \frac{1}{2} N_1 \) and \( \frac{1}{4} N_2 \) are added together to form the stiffness matrix \( K_{se} \). Hence all terms in the equations of motion are now known.

Note that the expressions for the above matrices simplify greatly when one assumes that \( U = V = W = \psi \), i.e. all displacement functions are described by the same shape function. In this case, the matrices are described as follows:

\[
K_L = \int_0^l \begin{bmatrix}
\text{matrix} \{ EA \psi_{x,\alpha} \psi_{x,\beta} \} & 0 & 0 & 0 \\
0 & \text{matrix} \{ EI \psi_{x,\alpha} \psi_{x,\beta} \} & 0 & 0 \\
0 & 0 & \text{matrix} \{ EI \psi_{x,\alpha} \psi_{x,\beta} \} & 0 \\
\text{matrix} \{ K_{e,\alpha} \psi_{x,\beta} \} & 0 & 0 & \text{matrix} \{ K_{e,\alpha} \psi_{x,\beta} \}
\end{bmatrix} dx \tag{4.72}
\]

\[
N_1 = EA \int_0^l \begin{bmatrix}
0 & \text{matrix}_{\alpha,\beta} \{ \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} q_{y,\gamma} \} & \text{matrix}_{\alpha,\beta} \{ \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} q_{y,\gamma} \} & 0 \\
\text{matrix}_{\alpha,\beta} \{ \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} q_{y,\gamma} \} & 0 & 0 & \text{matrix}_{\alpha,\beta} \{ \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} q_{y,\gamma} \} \\
\text{matrix}_{\alpha,\beta} \{ \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} q_{y,\gamma} \} & 0 & 0 & \text{matrix}_{\alpha,\beta} \{ \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} q_{y,\gamma} \}
\end{bmatrix} dx
\]

\[
= \begin{bmatrix}
0 & \text{matrix}_{\alpha,\beta} \{ N_{1,\alpha,\gamma} q_{y,\gamma} \} & \text{matrix}_{\alpha,\beta} \{ N_{1,\alpha,\gamma} q_{y,\gamma} \} & 0 \\
\text{matrix}_{\alpha,\beta} \{ N_{1,\alpha,\gamma} q_{y,\gamma} \} & 0 & 0 & \text{matrix}_{\alpha,\beta} \{ N_{1,\alpha,\gamma} q_{y,\gamma} \}
\end{bmatrix} \tag{4.73}
\]
assume the following displacement function:

Now, given all of the above, all that remains is to define $\psi$.

### 4.2.1 Beam Element Shape Function for FEM Model

Now substituting $\alpha$ by this inverse to isolate $\alpha$

where

The above matrix is a constant matrix. Take the inverse of this matrix and multiply both sides of the equation by this inverse to isolate $\alpha$:

Now substituting $\alpha$ into Equation (4.76),

$$u(\hat{x}) = \psi \alpha = X_b \alpha$$

and hence significantly reduces the number of matrix components to store and calculate. This is a reasonable assumption for beams according to experimental results by Sharf, who shows that using the same cubic shape function describing the transversal displacement of a spinning beam to describe the axial displacement as well results in faster convergence to the experimental values as the number of elements forming the beam is increased [47].

4.2.1 Beam Element Shape Function for FEM Model

Now, given all of the above, all that remains is to define $\psi$. For the FEM Model being used here, we assume the following displacement function:

$$u(\hat{x}) = \alpha_1 + \alpha_2 \hat{x} + \alpha_3 \hat{x}^2 + \alpha_4 \hat{x}^3$$

where $\alpha_i$ are parameters, $\alpha$ is the column matrix of $\alpha_i$, $\hat{x} = \frac{x}{l}$ with $l$ as the length of the beam element as shown in Figure 4.3, and $X_b$ is the row matrix of $\hat{x}$ terms. Now let

$$\begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} u(0) \\ \frac{\partial u(0)}{\partial x} \\ u(1) \\ \frac{\partial u(1)}{\partial x} \end{bmatrix}$$

and

$$\begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & \frac{2}{l} & 1 & 0 \\ 0 & \frac{3}{l} & \frac{3}{l} & 1 \end{bmatrix} = A \alpha$$

where $A$ is a constant matrix. Take the inverse of this matrix and multiply both sides of the equation by this inverse to isolate $\alpha$:

$$\alpha = A^{-1} q_\alpha = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & l & 0 & 0 \\ -3 & -2l & 3 & -l \\ 2 & l & -2 & l \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$$

Now substituting $\alpha$ into Equation (4.76),

$$\psi = X_b A^{-1} q_\alpha$$

where

$$\psi_4.2.1$$

$$N_2 = EA \int_0^l \begin{bmatrix} 0 & 0 & 0 \\ 0 & \frac{3}{2} \psi_x^T \psi_x v_x v_x + \frac{1}{2} \psi_x^T \psi_x w_x \\ 0 & \psi_x^T \psi_x w_x + \frac{3}{2} \psi_x^T \psi_x v_x v_x \end{bmatrix} dx$$

$$= \begin{bmatrix} 0 & 0 & 0 \\ 0 & \text{matrix}_{\alpha\beta} \{ \frac{3}{2} N_{2,\alpha\beta\gamma} q_{\gamma} q_{\alpha \gamma} + \frac{1}{2} N_{2,\alpha\beta\gamma} q_{\gamma} q_{\alpha \gamma} \} \\ 0 & \text{matrix}_{\alpha\beta} \{ \frac{3}{2} N_{2,\alpha\beta\gamma} q_{\gamma} q_{\alpha \gamma} + \frac{1}{2} N_{2,\alpha\beta\gamma} q_{\gamma} q_{\alpha \gamma} \} \end{bmatrix}$$

and

$$N_1,_{\alpha\beta\gamma} = EA \int_0^l \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} dx, \quad N_2,_{\alpha\beta\gamma} = EA \int_0^l \psi_{x,\alpha} \psi_{x,\beta} \psi_{x,\gamma} \psi_{x,\delta} dx$$

(4.75)
Hence we can conclude that the shape function \( \psi_b = X_b A^{-1} \). This is the shape function for a standard Hermitian beam element.

One could question why a cubic shape function was used for axial displacement as well, when it is more typical for the axial displacement to be described using a linear function. Apart from requiring some other shape function to describe \( \theta_x \), it has been observed by Sharf [47] that a cubic axial displacement shape function tends to converge on a solution much faster than when a linear shape function is used. Hence, a cubic axial shape function is used in the hopes of improving convergence and keeping \( \theta_x \) as a state.

### 4.3 Equations of Motion with Geometrical Nonlinearities for Plates

Similar to the beam case, the displacement of a plate can be described by three displacement functions, \( u(x, y, q_x), v(x, y, q_y), w(x, y, q_z) \) as shown in Figure 4.4, with each defined as follows:

\[
\begin{align*}
 u(x, y, q_x) &= U(x, y)q_x = U_\alpha q_x, \\
 v(x, y, q_y) &= V(x, y)q_y = V_\alpha q_y, \\
 w(x, y, q_z) &= W(x, y)q_z = W_\alpha q_z.
\end{align*}
\]

(4.80)

The only difference from the definitions given previously for the beam element is that the shape functions \( U, V \) and \( W \) are dependent upon \( y \) as well as \( x \). As a result, once again the equations of motion can be derived in a straight-forward manner, with the exception of \( K_{xx} \), which requires the strain terms to be defined.

![Figure 4.4: Right Triangular Plate Element and its Generalized Coordinates about Node 1](image)

As per the classical theory of plates [49], the non-zero strain fields are

\[
\begin{align*}
 \epsilon_{xx} &= u_x - z w_{xx} + \frac{1}{2} w_x^2, \\
 \epsilon_{yy} &= v_y - z w_{yy} + \frac{1}{2} w_y^2, \\
 \gamma_{xy} &= u_y + v_x - 2 z w_{xy} + w_x w_y.
\end{align*}
\]

(4.81)
Once again referring to Equation (4.6) for the alternate strain field forms, the following matrices can be derived:

\[
\begin{align*}
\epsilon_{xx} &= \mathbf{L}_{xx}^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xx} \mathbf{d} \\
\epsilon_{yy} &= \mathbf{L}_{yy}^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{yy} \mathbf{d} \\
\gamma_{xy} &= \mathbf{L}_{xy}^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xy} \mathbf{d} \\
\mathbf{L}_{xx} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & -z & 0 & 0 \end{bmatrix}^T \\
\mathbf{L}_{yy} &= \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & -z & 0 \end{bmatrix}^T \\
\mathbf{L}_{xy} &= \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & -2z \end{bmatrix}^T \\
\mathbf{H}_{xx} &= \text{diag}\left\{\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}\right\} \\
\mathbf{H}_{yy} &= \text{diag}\left\{\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}\right\} \\
\mathbf{H}_{xy} &= 0, \quad (i, j) \neq (5, 6), (6, 5), \\
\mathbf{H}_{xy}(5, 6) &= \mathbf{H}_{xy}(6, 5) = 1
\end{align*}
\]

where \( \lambda \) is Poisson’s ratio and in this case, we use the following generalized form:

\[
\mathbf{H}_{xy}(i, j) = 0, \quad (i, j) \neq (5, 6), (6, 5), \\
\mathbf{H}_{xy}(5, 6) = \mathbf{H}_{xy}(6, 5) = 1
\]

The constitutive relationship for a typical plate is as follows:

\[
\mathbf{\tau} = \mathbf{E}' \mathbf{\epsilon}
\]

\[
\begin{bmatrix}
\tau_{xx} \\
\tau_{yy} \\
\tau_{xy}
\end{bmatrix} = \frac{E}{1 - \lambda^2} \begin{bmatrix}
1 & \lambda & 0 \\
\lambda & 1 & 0 \\
0 & 0 & \frac{1-\lambda}{2}
\end{bmatrix} \begin{bmatrix}
\epsilon_{xx} \\
\epsilon_{yy} \\
\gamma_{xy}
\end{bmatrix}
\]

where \( \lambda \) is Poisson’s ratio and in this case, we use the following generalized form:

\[
\mathbf{\tau} = \mathbf{E} \mathbf{\epsilon}
\]

\[
\begin{bmatrix}
\tau_{xx} \\
\tau_{yy} \\
\tau_{xy}
\end{bmatrix} = \begin{bmatrix} a & b & c \\
b & d & e \\
c & e & f \end{bmatrix} \begin{bmatrix}
\epsilon_{xx} \\
\epsilon_{yy} \\
\gamma_{xy}
\end{bmatrix}
\]

where the constants \( a \) through \( f \) represent the unknown, but symmetric, constitutive relationship. With
the terms defined above, we now begin expanding out the stiffness matrix, starting with $K_L$:

$$K_L = \int \int \int_V \mathbf{D}^T C_{ijkl} L_{ij} L_{kl}^T \mathbf{D} \, dV$$

$$= h \int_A \left[ aK_a + bK_b + cK_c + dK_d + eK_e + fK_f \right] \, dx \, dy$$

where

$$K_a = \mathbf{D}^T L_{xx} L_{yy}^T \mathbf{D} = \begin{bmatrix} U_x^T U_x & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{h^2}{12} W_{xx}^T W_{xx} \end{bmatrix}$$

$$K_b = \mathbf{D}^T L_{xy} L_{yy}^T \mathbf{D} = \begin{bmatrix} 0 & U_y^T V_y & 0 \\ V_y^T U_y & 0 & 0 \\ 0 & 0 & \frac{h^2}{12} (W_{xy}^T W_{xx} + W_{yx}^T W_{xx}) \end{bmatrix}$$

$$K_c = \mathbf{D}^T L_{xx} L_{yy}^T \mathbf{D} = \begin{bmatrix} 0 & U_x^T V_x & 0 \\ V_x^T U_x & 0 & 0 \\ 0 & 0 & \frac{h^2}{6} (W_{xy}^T W_{xx} + W_{yx}^T W_{xx}) \end{bmatrix}$$

$$K_d = \mathbf{D}^T L_{yy} L_{yy}^T \mathbf{D} = \begin{bmatrix} 0 & U_y^T V_y & 0 \\ V_y^T U_y & 0 & 0 \\ 0 & 0 & \frac{h^2}{12} W_{yy}^T W_{yy} \end{bmatrix}$$

$$K_e = \mathbf{D}^T L_{xy} L_{xy}^T \mathbf{D} = \begin{bmatrix} 0 & U_y^T V_y & 0 \\ V_y^T U_y & 0 & 0 \\ 0 & 0 & \frac{h^2}{6} (W_{xy}^T W_{yy} + W_{yx}^T W_{yy}) \end{bmatrix}$$

$$K_f = \mathbf{D}^T L_{xy} L_{xy}^T \mathbf{D} = \begin{bmatrix} 0 & U_x^T V_x & 0 \\ V_x^T U_x & 0 & 0 \\ 0 & 0 & \frac{h^2}{6} W_{xx}^T W_{xx} \end{bmatrix}$$

where the component form has been omitted this time, but it is straightforward to derive one.

Similar to above, $N_1$ is expanded as follows:

$$N_1 = \int \int \int_V \mathbf{D}^T C_{ijkl} (L_{ij} d^T H_{kl} + d^T L_{ij} H_{kl} + H_{ij} dL_{kl}^T) \mathbf{D} \, dV$$

$$= h \int_A \left[ aN_{1a} + bN_{1b} + cN_{1c} + dN_{1d} + eN_{1e} + fN_{1f} \right] \, dx \, dy$$

where

$$N_{1a} = \mathbf{D}^T (L_{xx} d^T H_{xx} + d^T L_{xx} H_{xx} + H_{xx} dL_{xx}^T) \mathbf{D}$$

$$= \begin{bmatrix} 0 & 0 & U_x^T W_{xx} \\ 0 & 0 & 0 \\ W_x^T U_x w_x & 0 & W_x^T W_{xx} \end{bmatrix}$$

$$N_{1b} = \mathbf{D}^T (L_{xx} d^T H_{xy} + d^T L_{xx} H_{yy} + H_{xx} dL_{yy}^T) \mathbf{D}$$

$$= \begin{bmatrix} 0 & 0 & U_y^T W_{xy} \\ 0 & 0 & V_x^T W_{xx} \\ W_y^T U_y w_y & W_x^T V_x w_x & W_x^T W_{xy} \end{bmatrix}$$

$$N_{1c} = \int \int \int_V \mathbf{D}^T C_{ijkl} (L_{xx} d^T H_{xy} + d^T L_{xx} H_{yy} + H_{xx} dL_{xx}^T) \mathbf{D} \, dV$$

$$= \begin{bmatrix} 0 & 0 & U_y^T W_{xy} + U_x^T W_{xy} + U_x^T W_{xy} \\ 0 & 0 & V_y^T W_{xy} \\ W_y^T U_y w_y + W_x^T U_x w_x + W_x^T U_x w_x & W_y^T V_x w_x & W_y^T W_{xy} + W_x^T W_{xy} u_x + W_x^T W_{xy} \end{bmatrix}$$

(4.85)
It should be noted in passing that the matrix has no terms with pure in-plane axial displacements.

Finally, \( \mathbf{N}_2 \) only has terms with pure out-of-plane transversal displacements, as shown below in terms of block diagonals:

\[
\mathbf{N}_2 = \iiint_V \mathbf{D}^T \mathbf{C}_{ijkl}(\mathbf{H}_{ij} \mathbf{d}^T \mathbf{H}_{kl} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{kl} \mathbf{d} \mathbf{H}_{ij}) \mathbf{D} \, dV
= h \iiint_A \left[ a \mathbf{N}_{2a} + b \mathbf{N}_{2b} + c \mathbf{N}_{2c} + d \mathbf{N}_{2d} + e \mathbf{N}_{2e} + f \mathbf{N}_{2f} \right] \, dx 
\]

where

\[
\mathbf{N}_{2a} = \mathbf{D}^T(\mathbf{H}_{xx} \mathbf{d}^T \mathbf{H}_{xx} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xx} \mathbf{d} \mathbf{H}_{xx}) \mathbf{D}
= \text{diag} \left\{ \begin{bmatrix} 0 & 0 & \frac{1}{2} \mathbf{W}_x^T \mathbf{W}_x w_x^2 \end{bmatrix} \right\}
\]

\[
\mathbf{N}_{2b} = \mathbf{D}^T(\mathbf{H}_{xx} \mathbf{d}^T \mathbf{H}_{xy} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xy} \mathbf{d} \mathbf{H}_{xy}) \mathbf{D}
= \text{diag} \left\{ \begin{bmatrix} 0 & 0 & \frac{1}{2} \mathbf{W}_x^T \mathbf{W}_y w_x^2 + \frac{1}{2} \mathbf{W}_y^T \mathbf{W}_y w_x^2 + \mathbf{W}_y^T \mathbf{W}_x w_x w_y + \mathbf{W}_x^T \mathbf{W}_y w_x w_y \end{bmatrix} \right\}
\]

\[
\mathbf{N}_{2c} = \mathbf{D}^T(\mathbf{H}_{xx} \mathbf{d}^T \mathbf{H}_{xy} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xy} \mathbf{d} \mathbf{H}_{xy}) \mathbf{D}
= \text{diag} \left\{ \begin{bmatrix} 0 & 0 & \frac{3}{2} \mathbf{W}_y^T \mathbf{W}_x w_x^2 + \frac{3}{2} \mathbf{W}_x^T \mathbf{W}_y w_y^2 + 3 \mathbf{W}_x^T \mathbf{W}_x w_x w_y \end{bmatrix} \right\}
\]

\[
\mathbf{N}_{2d} = \mathbf{D}^T(\mathbf{H}_{yy} \mathbf{d}^T \mathbf{H}_{yy} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{yy} \mathbf{d} \mathbf{H}_{yy}) \mathbf{D}
= \text{diag} \left\{ \begin{bmatrix} 0 & 0 & \frac{3}{2} \mathbf{W}_y^T \mathbf{W}_y w_y^2 \end{bmatrix} \right\}
\]

\[
\mathbf{N}_{2e} = \mathbf{D}^T(\mathbf{H}_{yy} \mathbf{d}^T \mathbf{H}_{xy} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xy} \mathbf{d} \mathbf{H}_{xy}) \mathbf{D}
= \text{diag} \left\{ \begin{bmatrix} 0 & 0 & \frac{3}{2} \mathbf{W}_y^T \mathbf{W}_x w_y^2 + \frac{3}{2} \mathbf{W}_x^T \mathbf{W}_y w_y^2 + 3 \mathbf{W}_x^T \mathbf{W}_y w_x w_y \end{bmatrix} \right\}
\]

\[
\mathbf{N}_{2f} = \mathbf{D}^T(\mathbf{H}_{xy} \mathbf{d}^T \mathbf{H}_{xy} + \frac{1}{2} \mathbf{d}^T \mathbf{H}_{xy} \mathbf{d} \mathbf{H}_{xy}) \mathbf{D}
= \text{diag} \left\{ \begin{bmatrix} 0 & 0 & \mathbf{W}_x^T \mathbf{W}_x w_x^2 + \mathbf{W}_y^T \mathbf{W}_y w_y^2 + 2 \mathbf{W}_y^T \mathbf{W}_x w_x w_y + 2 \mathbf{W}_y^T \mathbf{W}_x w_x w_y \end{bmatrix} \right\}
\]
As before, combining all the matrices above results in the complete stiffness term $\mathbf{K}_{ee}$. The above forms are quite cumbersome and it is quite obvious that the matrices of the subscripts $c$ and $e$ would be unnecessary if the typical constitutive relationship for a typical plate was used. It shall be shown during the next chapter why both of these matrices become necessary when a membrane is wrinkled.

Now, once again let us try to simplify the matrices by assuming that $\mathbf{U} = \mathbf{V} = \mathbf{W} = \psi$. However, because the shape functions are now dependent on both $x$ and $y$, the simplification results in much more varied terms than the beam case. Here are the components of the stiffness matrix with the above assumptions:

\[
\mathbf{K}_a = \begin{bmatrix} K_{a,x,x} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & K_{a,xx,xx} \end{bmatrix}, \quad \mathbf{K}_b = \begin{bmatrix} 0 & K_{a,x,y} & 0 \\ K_{a,x,y} & 0 & 0 \\ 0 & 0 & K_{t,xx,xx} + K_{t,xx,xx}^T \end{bmatrix}
\]

\[
\mathbf{K}_c = \begin{bmatrix} K_{a,x,y} + K_{a,x,y} & K_{a,x,x} & 0 \\ K_{a,x,y} & 0 & 0 \\ 0 & 0 & 2(K_{a,xx,xy} + K_{a,xx,xy}^T) \end{bmatrix}, \quad \mathbf{K}_d = \begin{bmatrix} 0 & 0 & 0 \\ 0 & K_{a,y,y} & 0 \\ 0 & 0 & K_{t,yy,yy} \end{bmatrix}
\]

\[
\mathbf{K}_e = \begin{bmatrix} 0 & K_{a,y,y} & 0 \\ K_{a,y,y} & K_{a,y,y} + K_{a,y,y}^T & 0 \\ 0 & 0 & 2(K_{a,yy,xy} + K_{a,yy,xy}^T) \end{bmatrix}, \quad \mathbf{K}_f = \begin{bmatrix} K_{a,y,y} & K_{a,x,x} & 0 \\ K_{a,y,y} & K_{a,x,x} & 0 \\ 0 & 0 & 4K_{t,xy,xy} \end{bmatrix}
\]

\[
\mathbf{N}_{1a} = \begin{bmatrix} 0 & 0 & N_{1,xx} \\ 0 & 0 & 0 \\ N_{1,xx} & 0 & N_{1,xx} \end{bmatrix}, \quad \mathbf{N}_{1b} = \begin{bmatrix} 0 & 0 & N_{1,yy} \\ 0 & 0 & N_{1,yy} \\ N_{1,yy}^T & N_{1,yy} & N_{1,xx}(q_y) \end{bmatrix}
\]

\[
\mathbf{N}_{1c} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & N_{1,xx} \\ N_{1,xx}(q_y) & N_{1,xx}(q_y) + N_{1,xx}(q_y) & N_{1,xx}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) \end{bmatrix}
\]

\[
\mathbf{N}_{1d} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & N_{1,yy} \\ N_{1,yy}(q_y) & N_{1,yy}(q_y) \end{bmatrix}, \quad \mathbf{N}_{1e} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & N_{1,yy}^T(q_y) + N_{1,yy}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) \\ N_{1,yy}(q_y) & N_{1,xx}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) \end{bmatrix}
\]

\[
\mathbf{N}_{1f} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & N_{1,yy}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) \\ N_{1,xx}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) + N_{1,xx}(q_y) \end{bmatrix}
\]

\[
\mathbf{N}_{2a} = \text{diag} \left\{ 0, 0, \frac{3}{2}N_{2,xxxx}(q_x) \right\}, \quad \mathbf{N}_{2b} = \text{diag} \left\{ 0, 0, \frac{1}{2}N_{2,xxyy}(q_x) + \frac{1}{2}N_{2,yyxx}(q_x) + N_{2,xyxy}(q_x)^T + N_{2,xyxy}(q_x) \right\}
\]

\[
\mathbf{N}_{2c} = \text{diag} \left\{ 0, 0, \frac{3}{2}N_{2,xyxx}(q_x)^T + \frac{3}{2}N_{2,xyxx}(q_x) + 3N_{2,xyxy}(q_x) \right\}, \quad \mathbf{N}_{2d} = \text{diag} \left\{ 0, 0, \frac{3}{2}N_{2,yyyy}(q_y) \right\}
\]

\[
\mathbf{N}_{2e} = \text{diag} \left\{ 0, 0, \frac{3}{2}N_{2,yyyy}(q_x)^T + \frac{3}{2}N_{2,yyyy}(q_x) + 3N_{2,yyxy}(q_x) \right\}, \quad \mathbf{N}_{2f} = \text{diag} \left\{ 0, 0, N_{2,xyyx}(q_x) + N_{2,yyxx}(q_x) + 2N_{2,xyxy}(q_x)^T + 2N_{2,xyxy}(q_x) \right\}
\]
where
\[ K_{a,i,j} = \psi_i^T \psi_j \]
\[ K_{t,i,j} = \frac{h^2}{12} \psi_i^T \psi_j \]

\[ N_{1,ijk}(q_m) = \psi_i^T \psi_j \psi_k q_m \]
\[ N_{2,ijkl}(q_m) = \psi_i^T \psi_j \psi_k q_m \psi_l q_m \]

and the \( i, j, k, \) and \( l \) refer to the differentiation subscripts in this case. There are three separately-defined \( K_{a,i,j} \) and \( K_{t,i,j} \) matrices, six such \( N_{1,ijk}(q_m) \) matrices, and nine such \( N_{2,ijkl}(q_m) \) matrices. This number can be brought down for the nonlinear terms - specifically, by using the component form and swapping the indices around. For example,

\[ N_{1,ijk}(a,b,c) = N_{1,kij}(c,a,b) \]

is an entirely valid method to find the component \( (a,b,c) \) of matrix \( N_{1,ijk} \) from matrix \( N_{1,kij} \). In this case, only four \( N_{1,ijk}(q_m) \) matrices and five \( N_{2,ijkl}(q_m) \) matrices are required to be pre-calculated and stored for access during simulation. This is still a very large number of matrices that need to be stored and accessed, resulting in a much slower simulation time compared to a model that only uses beams.

### 4.3.1 Triangular Plate Element Shape Function for FEM Model

As before, the shape function \( \psi \) must be defined. It is assumed that the plate being used is a triangular one. The displacement function describing the triangular plate is:

\[ u(L_1, L_2, L_3) = \alpha_1 L_1 + \alpha_2 L_2 + \alpha_3 L_3 + \alpha_4 (L_2^2 L_1 + \frac{1}{2} L_1 L_2 L_3) + \alpha_5 (L_3^2 L_2 + \frac{1}{2} L_1 L_2 L_3) \]
\[ + \alpha_6 (L_1^2 L_3 + \frac{1}{2} L_1 L_2 L_3) + \alpha_7 (L_1^2 L_2 + \frac{1}{2} L_1 L_2 L_3) \]
\[ + \alpha_8 (L_2^2 L_3 + \frac{1}{2} L_1 L_2 L_3) + \alpha_9 (L_3^2 L_1 + \frac{1}{2} L_1 L_2 L_3) \]
\[ = X_i \alpha \]

where \( L_1, L_2, \) and \( L_3 \) are coordinates that describe the areas of the triangle as shown in Figure 4.5, often called Natural Coordinates. These can be converted to the standard Cartesian coordinate representations.
Chapter 4. Unconstrained Nonlinear Flexible Structural Dynamics

The plate shape function

\[ q_i \]

will be clarified in the upcoming section.

nor is the explicit knowledge of the full expression required for the simulator. This will be clarified in

and \( i, j, \) and \( k \) are permutation indices. Specifically, if \( i = 1 \), then \( j = 2 \) and \( k = 3 \), if \( i = 2 \), then \( j = 3 \) and \( k = 1 \), and if \( i = 3 \), then \( j = 1 \) and \( k = 2 \). Given the above definition for the displacement function, \( q_\alpha \) can be written as follows:

\[
\mathbf{q}_\alpha = \begin{bmatrix}
    q_1 \\
    q_2 \\
    q_3 \\
    q_4 \\
    q_5 \\
    q_6 \\
    q_7 \\
    q_8 \\
    q_9
\end{bmatrix} = \begin{bmatrix}
    u(1,0,0) \\
    \frac{\partial u(1,0,0)}{\partial y} \\
    \frac{\partial u(1,0,0)}{\partial x} \\
    u(0,1,0) \\
    \frac{\partial u(0,1,0)}{\partial y} \\
    \frac{\partial u(0,1,0)}{\partial x} \\
    u(0,0,1) \\
    \frac{\partial u(0,0,1)}{\partial y} \\
    \frac{\partial u(0,0,1)}{\partial x}
\end{bmatrix} = \frac{1}{2A} \begin{bmatrix}
    2A & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    a_1 & a_2 & a_3 & 0 & 0 & a_3 & a_2 & 0 & 0 \\
    -b_1 & -b_2 & -b_3 & 0 & 0 & -b_3 & -b_2 & 0 & 0 \\
    0 & 2A & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    -b_1 & -b_2 & -b_3 & -b_1 & 0 & 0 & 0 & -b_3 & 0 \\
    0 & 0 & 2A & 0 & 0 & 0 & 0 & 0 & 0 \\
    a_1 & a_2 & a_3 & 0 & 0 & a_3 & a_2 & 0 & 0 \\
    -b_1 & -b_2 & -b_3 & 0 & 0 & 0 & 0 & -b_1 & 0
\end{bmatrix} \begin{bmatrix}
    \alpha_1 \\
    \alpha_2 \\
    \alpha_3 \\
    \alpha_4 \\
    \alpha_5 \\
    \alpha_6 \\
    \alpha_7 \\
    \alpha_8 \\
    \alpha_9
\end{bmatrix} = \mathbf{A} \alpha
\]

(4.104)

Once again taking the inverse of this matrix and multiplying to both sides of the equation:

\[
\alpha = \mathbf{A}^{-1} \mathbf{q}_\alpha = \begin{bmatrix}
    1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
    -1 & 0 & 0 & 1 & b_3 & a_3 & 0 & 0 & 0 \\
    0 & 0 & 0 & -1 & 0 & 0 & 1 & b_1 & a_1 \\
    1 & b_2 & a_2 & 0 & 0 & 0 & -1 & 0 & 0 \\
    1 & -b_3 & -a_3 & -1 & 0 & 0 & 0 & 0 & 0 \\
    0 & 0 & 0 & 1 & -b_1 & -a_1 & -1 & 0 & 0 \\
    -1 & 0 & 0 & 0 & 0 & 0 & 1 & -b_2 & -a_2
\end{bmatrix} \begin{bmatrix}
    q_1 \\
    q_2 \\
    q_3 \\
    q_4 \\
    q_5 \\
    q_6 \\
    q_7 \\
    q_8 \\
    q_9
\end{bmatrix}
\]

(4.105)

and substituting \( \alpha \) into Equation (4.101),

\[
u(\dot{x}) = X_\iota \mathbf{A}^{-1} \mathbf{q}_\alpha = \psi_\iota \mathbf{q}_\alpha
\]

(4.106)

The plate shape function \( \psi_\iota = X_\iota \mathbf{A}^{-1} \) is a messy set of polynomials, and is not fully determined here, nor is the explicit knowledge of the full expression required for the simulator. This will be clarified in the upcoming section.
4.3.2 Three Dimensional Displacements for Beam Element and Triangular Plate Element

The beam stiffness matrix and its nodal degrees of freedom are physically unique in a sense that each generalized coordinate can be associated with a unique physically-defined displacement - axial displacement is associated with \( x \)-axis Cartesian and angular displacements pair, while the two transversal displacements are associated with \( y \)-axis Cartesian - \( z \)-axis angular displacements pair and \( z \)-axis Cartesian - \( y \)-axis angular displacements pair, resulting in six nodal degrees of freedom.

For a triangular plate however, a node is associated with nine nodal degrees of freedom - each Cartesian displacement is associated with two displacements defined by the partial derivatives about \( y \) and \( x \). For \( z \)-axis Cartesian displacements, these are defined as angular displacements about \( x \) and \( y \) respectively, but for \( x \)-axis and \( y \)-axis Cartesian displacements, it is unclear what physical displacements its partial derivatives are associated with. In addition, it is also unclear how the nodal degrees of freedom are merged together when a beam element and a triangular plate element share a node.

Instead of using such physical association, the mathematical terms, i.e., the derivatives of the Cartesian displacement as shown in Equation (4.104), are considered directly. In such a case, there are no relationships binding the different elastic displacement states. Hence, there would be no need for the merging of the different degrees of freedom. The downside of this approach is that you end up with up to 27 generalized coordinate states per element. However, this seems necessary to maintain the results of the simulation to be consistent with the known literature.

In addition to the above, there’s the concern of how different elements merge together - the stacked form presented in Section 4.4 contains displacements of different nodes in between each other. To be specific, each matrix block defined in the section represents the combined displacements of three separate nodes - and by stacking the matrices describing the displacements in different direction together in the given form, the combination of element matrices sharing the same node becomes tricky.

To mitigate this issue, after the element matrix is generated in the form described in Section 4.4, its rows and columns are rearranged so that the displacements associated with a particular node are placed together. Specifically, the generalized coordinates, originally in the following form,

\[
\begin{align*}
q_e^{(1..9)} &= [u_1 \frac{\partial u_1}{\partial y} \frac{\partial u_1}{\partial x} u_2 \frac{\partial u_2}{\partial y} \frac{\partial u_2}{\partial x} u_3 \frac{\partial u_3}{\partial y} \frac{\partial u_3}{\partial x}]^T, \\
q_e^{(10..18)} &= [v_1 \frac{\partial v_1}{\partial y} \frac{\partial v_1}{\partial x} v_2 \frac{\partial v_2}{\partial y} \frac{\partial v_2}{\partial x} v_3 \frac{\partial v_3}{\partial y} \frac{\partial v_3}{\partial x}]^T, \\
q_e^{(19..27)} &= [w_1 \frac{\partial w_1}{\partial y} \frac{\partial w_1}{\partial x} w_2 \frac{\partial w_2}{\partial y} \frac{\partial w_2}{\partial x} w_3 \frac{\partial w_3}{\partial y} \frac{\partial w_3}{\partial x}]^T.
\end{align*}
\]

are arranged to have the following form instead:

\[
\begin{align*}
q_e^{(1..9)} &= [u_1 v_1 w_1 \frac{\partial v_1}{\partial y} \frac{\partial v_1}{\partial x} \frac{\partial v_1}{\partial y} \frac{\partial v_1}{\partial x} \frac{\partial v_1}{\partial y} \frac{\partial v_1}{\partial x} \frac{\partial v_1}{\partial y} \frac{\partial v_1}{\partial x}]^T, \\
q_e^{(10..18)} &= [u_2 v_2 w_2 \frac{\partial v_2}{\partial y} \frac{\partial v_2}{\partial x} \frac{\partial v_2}{\partial y} \frac{\partial v_2}{\partial x} \frac{\partial v_2}{\partial y} \frac{\partial v_2}{\partial x} \frac{\partial v_2}{\partial y} \frac{\partial v_2}{\partial x}]^T, \\
q_e^{(19..27)} &= [u_3 v_3 w_3 \frac{\partial v_3}{\partial y} \frac{\partial v_3}{\partial x} \frac{\partial v_3}{\partial y} \frac{\partial v_3}{\partial x} \frac{\partial v_3}{\partial y} \frac{\partial v_3}{\partial x} \frac{\partial v_3}{\partial y} \frac{\partial v_3}{\partial x}]^T.
\end{align*}
\]

Once all elements are grouped in this fashion, the merging of the two element matrices sharing the same node become logical and straight-forward. For example, let \( K_1 \) and \( K_2 \) be the stiffness matrices of two triangular plates, both in same orientation, with a common node at \( N_2 \). Then the two matrices can be
merged together as follows:

\[
\mathbf{K} = \begin{bmatrix}
\mathbf{K}_1(1.9, 1.9) & \mathbf{K}_1(1.9, 10.18) & \mathbf{K}_1(1.9, 19.27) & 0 & 0 \\
\mathbf{K}_2(10.18, 1.9) & \mathbf{K}_2(10.18, 10.18) + \mathbf{K}_2(10.18, 10.18) & \mathbf{K}_2(10.18, 19.27) & 0 & 0 \\
\mathbf{K}_2(19.27, 1.9) & \mathbf{K}_2(19.27, 10.18) & \mathbf{K}_2(19.27, 19.27) & 0 & 0 \\
0 & \mathbf{K}_2(1.9, 10.18) & 0 & \mathbf{K}_3(1.9, 1.9) & \mathbf{K}_3(1.9, 19.27) \\
0 & \mathbf{K}_2(19.27, 10.18) & 0 & \mathbf{K}_4(19.27, 1.9) & \mathbf{K}_4(19.27, 19.27)
\end{bmatrix}
\]

(4.109)

\[
\mathbf{K}(1.27, 1.27) = \mathbf{K}(1.27, 1.27) + \mathbf{K}_1 \\
+ \mathbf{K}(28.36, 10.18, 37.45), [28.36, 10.18, 37.45]) = \mathbf{K}(28.36, 10.18, 37.45), [28.36, 10.18, 37.45]) + \mathbf{K}_2 \\
+ \mathbf{K}(i_1, i_2, i_3, [i_4, i_5, i_6]) + \mathbf{K}_3
\]

(4.110)

where \(i_{gi}\) is the global matrix index for node \(i\). This element matrix merging process representation is very useful for simplifying codes for global matrix generation later on.

### 4.4 Implementation of the FEM Model and the Dynamics Simulation

An FEM model simulation system was initially developed using the Matlab scripting language, but was quickly abandoned after implementing the plate portion of the system and observing extremely slow simulation times. Instead, the development shifted to the Fortran 2003 programming language to take advantage of the speed inherent in a compiled language. As the Fortran version of the simulator was built upon the experiences gained from building the Matlab simulator, the implementation descriptions of the Matlab simulator are omitted here. Much of the specific implementation details are in the Appendix. Here, the theory behind the system implementation is provided, in the process making some references to functions used in the implemented simulator. It should be noted, in particular, that the system contains a custom-implemented Symbolic Math system called Polynom, constructed to allow simple mathematical operations performed on a data structure containing information about polynomials. Several references are made to this class during the discussion of the system.

#### 4.4.1 Nodes, Elements, and Frame Orientations

Prior to any actual matrix generation, there is a need to identify the location and the orientation of each element. This is also equivalent to determining the location of the nodes for each element, and then using that information to determine the rotation matrices.

Nodes and elements for beams are generated using the beam_node_gen - a simple function written by the author that accepts as input \(P_1\) and \(P_2\), the inertial coordinate location of the two ends of the beam, and \(n\), the number of elements to be generated. The process for generating the node locations is simple: A node is generated at every \(\frac{1}{n} (P_2 - P_1) + P_1\) step, where \(i = [1..n+1]\).

Once the node locations are known, the rotation matrix can be calculated. In the case of a beam, this is performed as follows:

1. Let \(p = P_2 - P_1\),

2. Let \(l = \frac{1}{2} p^T p\),
3. Initialize a rotation matrix \( \mathbf{C} \) such that, if \( p(1) < 0 \), \( \mathbf{C} = -\mathbf{C}_{y,z} \) and \( \mathbf{p} \) is rotated by \( c \). Else, \( \mathbf{C} = \mathbf{I}_3 \). (This step is necessary to prevent an error state during the next step.)

4. Use Singular Value Decomposition (SVD) to find the rotation matrix between \( \mathbf{p} \) and \( \mathbf{p}_r = [l, 0, 0]^T \), the element beam.

This SVD rotation matrix generator algorithm is as follows:

1. Let \( \mathbf{H} = \mathbf{pp}^T \).

2. Decompose using SVD: \( \mathbf{H} = \mathbf{USV}^T \).

3. Let \( \mathbf{C} = \mathbf{UV}^T \).

4. If \( \text{det} \mathbf{C} < 0 \), then let \( \mathbf{C}(1..3, 3) = -\mathbf{C}(1..3, 3) \) (This step guarantees \( \text{det} \mathbf{C} > 0 \)).

In addition, a transformation matrix corresponding to the rotation represented by \( \mathbf{C} \) is needed in order to rotate the matrices in an element frame to the body frame. This transformation is performed by considering the following: let \( X, Y, Z \), and \( U, V, W \) be the global representations of the Cartesian coordinates and displacement functions for a node, with \( x, y, z \) and \( u, v, w \) as the local representations as before. These parameters are related to each other as follows:

\[
\begin{bmatrix}
  u \\
  v \\
  w
\end{bmatrix} = 
\begin{bmatrix}
  C_{11} & C_{12} & C_{13} \\
  C_{21} & C_{22} & C_{23} \\
  C_{31} & C_{32} & C_{33}
\end{bmatrix}
\begin{bmatrix}
  U \\
  V \\
  W
\end{bmatrix} = 
\begin{bmatrix}
  C_{11}U + C_{12}V + C_{13}W \\
  C_{21}U + C_{22}V + C_{23}W \\
  C_{31}U + C_{32}V + C_{33}W
\end{bmatrix} \quad (4.111)
\]

\[
\begin{bmatrix}
  X \\
  Y \\
  Z
\end{bmatrix} = 
\begin{bmatrix}
  C_{11} & C_{12} & C_{13} \\
  C_{21} & C_{22} & C_{23} \\
  C_{31} & C_{32} & C_{33}
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix} = 
\begin{bmatrix}
  C_{11}x + C_{21}y + C_{31}z \\
  C_{12}x + C_{22}y + C_{32}z \\
  C_{13}x + C_{23}y + C_{33}z
\end{bmatrix} \quad (4.112)
\]

where \( C_{ij} \) are the \( i,j \)th component of \( \mathbf{C} \), the global to local element rotation matrix as before.

Briefly mentioned in the previous section, a beam is assumed to have three Cartesian and three angular displacements associated with each node, where each angular displacement is simply an \( x \)-derivative of the appropriate displacement function. The goal here is to represent these local displacement functions and their partial derivatives (the local generalized coordinates), in terms of the global displacement functions and their partial derivatives (the global generalized coordinates), making use of above coordinate rotations, and extract the transformation matrix out of that relation.

First, begin by noting that the displacement functions \( u, v \) and \( w \) are already represented in terms of \( U, V \) and \( W \) above. Only \( \frac{\partial u}{\partial x}, \frac{\partial w}{\partial x} \) and \( \frac{\partial w}{\partial x} \) need to be rearranged. The local partial derivatives can be represented in terms of the global partial derivatives by making use of the chain rule. For example, \( \frac{\partial u}{\partial x} \) is expanded as follows:

\[
\frac{\partial u}{\partial x} = C_{11} \frac{\partial U}{\partial x} + C_{12} \frac{\partial V}{\partial x} + C_{13} \frac{\partial W}{\partial x} = C_{11} \frac{\partial u}{\partial x} + C_{12} \frac{\partial v}{\partial x} + C_{13} \frac{\partial w}{\partial x}
\]

where the global coordinate partial derivatives are derived from Equation (4.112). Note here that the last equation contains nine unique global displacement function derivatives, whereas for beams we only
have three element displacement function derivatives, as the displacement functions only depend on \( x \) and hence the partial derivatives about \( y \) or \( z \) are zeros. The increased number of generalized coordinates is not desirable for several reasons, but most importantly because it introduces the task of having to monitor the resulting stiffness matrix for singularity caused by the beam aligning itself to one of the global axes and resizing the matrix to remove such singularities.

A simple alternative to the above is to use the very first line of Equation (4.113), that is, use \( \frac{\partial U}{\partial x} \), \( \frac{\partial V}{\partial x} \) and \( \frac{\partial W}{\partial x} \) in place of the actual global displacement function derivatives. The number of local and the global generalized coordinates are both fixed to six per node, avoiding above-stated problems.

There are restrictions however. As this is a partial derivative about the local \( x \), other elements being fixed to this element (i.e., \( C^1 \) continuity, shared partial derivative generalized coordinates) must be positioned along a common local \( x \)-axis. A straight beam composed of multiple beam elements satisfies this condition, while this can be taken as an approximation for a mildly curved beam with very small displacements along local \( y \) or \( z \) directions. Any other elements would have to be pinned to the beam element instead (i.e., \( C^0 \) continuity, unshared partial derivative generalized coordinates). In case of the solar sail, sail booms, sail membranes and the supporting strings are all considered pinned to each other, while the elements that make up each component are considered fixed to each other.

Having the above in mind, the coordinate transformations can then be grouped and rearranged as follows:

\[
\begin{bmatrix}
  u \\
  \frac{\partial u}{\partial x} \\
  v \\
  \frac{\partial v}{\partial x} \\
  w \\
  \frac{\partial w}{\partial x}
\end{bmatrix}
= \begin{bmatrix}
  C_{11} & 0 & C_{12} & 0 & C_{13} & 0 \\
  0 & C_{11} & 0 & C_{12} & 0 & C_{13} \\
  C_{21} & 0 & C_{22} & 0 & C_{23} & 0 \\
  0 & C_{21} & 0 & C_{22} & 0 & C_{23} \\
  C_{31} & 0 & C_{32} & 0 & C_{33} & 0 \\
  0 & C_{31} & 0 & C_{32} & 0 & C_{33}
\end{bmatrix}
\begin{bmatrix}
  U \\
  \frac{\partial U}{\partial x} \\
  V \\
  \frac{\partial V}{\partial x} \\
  W \\
  \frac{\partial W}{\partial x}
\end{bmatrix}
\]

(4.114)

Note that this is a transformation for just one node, and the actual generalized coordinates for one element is composed of two nodes. The actual global to element coordinate transformation can be written as follows:

\[
\begin{bmatrix}
  u_1 \\
  \frac{\partial u_1}{\partial x} \\
  u_2 \\
  \frac{\partial u_2}{\partial x} \\
  v_1 \\
  \frac{\partial v_1}{\partial x} \\
  v_2 \\
  \frac{\partial v_2}{\partial x} \\
  w_1 \\
  \frac{\partial w_1}{\partial x} \\
  w_2 \\
  \frac{\partial w_2}{\partial x}
\end{bmatrix}
= \begin{bmatrix}
  C_{11} & 1_2 & C_{12} & 1_2 & C_{13} & 1_2 \\
  C_{21} & 1_2 & C_{22} & 1_2 & C_{23} & 1_2 \\
  C_{31} & 1_2 & C_{32} & 1_2 & C_{33} & 1_2
\end{bmatrix}
\begin{bmatrix}
  q_x \\
  q_{dx} \\
  q_y \\
  q_{dy} \\
  q_z \\
  q_{dz}
\end{bmatrix}
\]

(4.115)
where \( T_b \) is the beam element transformation matrix and \( q_{ge} \) is the generalized coordinates in the global frame.

Nodes and elements for triangular plates are generated via a function written by the author, \texttt{tri_node_gen}. The inputs to this function are: vertex coordinates \( P_1, P_2 \) and \( P_3 \), and \texttt{n_form} parameter. \texttt{n_form} is a two dimensional integer array defined by the author that represents the location of a set of nodes in a rough relative form. Specifically, an internal function named \texttt{gen_tri_n_form} is used to generate an array that looks as follows:

\[
\begin{bmatrix}
15 & 0 & 0 & 0 & 0 \\
13 & 14 & 0 & 0 & 0 \\
10 & 11 & 12 & 0 & 0 \\
6 & 7 & 8 & 9 & 0 \\
1 & 2 & 3 & 4 & 5 \\
\end{bmatrix}
\] (4.116)

The positive integers represent a node, while the zeros are there as a filler for the array and not referenced. The nodes 1, 5 and 15 represent a vertex on the larger triangular plate. Below is the array superimposed with the triangular plate elements:

\[
\begin{bmatrix}
15 & 0 & 0 & 0 & 0 \\
13 & 14 & 0 & 0 & 0 \\
10 & 11 & 12 & 0 & 0 \\
6 & 7 & 8 & 9 & 0 \\
1 & 2 & 3 & 4 & 5 \\
\end{bmatrix}
\]

This shape was chosen specifically to be used for a square solar sail, where its membranes are often divided into four quadrants as demonstrated by both the archetypal and the cord-mat sail. This shape would represent one of those four quadrants.

With the above-defined \texttt{n_form} and the locations of \( P_i \)'s, the edge-nodes' positions can be determined easily: Letting the number of elements on the horizontal or the vertical edge be \( n \), the horizontal, vertical, and diagonal edge nodes are located at:

\[
P_{\text{form}(n+1,i)} = \frac{i-1}{n} (P_2 - P_1) + P_1 \\
P_{\text{form}(i,1)} = P_3 - \frac{i-1}{n} (P_3 - P_1) \\
P_{\text{form}(i,i)} = P_3 - \frac{i-1}{n} (P_3 - P_2)
\] (4.117)

where \( i = [2..n] \). Note that the above formulas correspond to node numbers as indicated by \texttt{n_form}(n + 1, i), \texttt{n_form}(i, 1), \texttt{n_form}(i, i) respectively, hence the nodes will be numbered as given in \texttt{n_form} without requiring any special calculation to determine the correct node.

Once the edge nodes are determined, the remaining nodes can also be determined with ease. Based on the arrangement of nodes above, one can let the \( i^{th} \) node on the vertical edge be the starting node \( P_s \), and let the node on the diagonal edge and horizontally across from the starting node be the ending node \( P_e \). Also, let the number of elements between those two nodes be \( e_n \). Then the coordinates of the
nodes in between can be described by

$$P_{n, \text{form}(i, j)} = P_n + \frac{j-1}{e_n} (P_e - P_n) \tag{4.118}$$

where \( j = [2..e_n] \). Once again, the formula and its indexing can be associated directly as \( \text{form}(i, j) \).

As before, the rotation matrix can be calculated once the nodes are known. The calculation of the rotation matrix for a triangulate plate is more straightforward due to the higher number of nodes available to determine uniquely the rotation matrix. The steps are as follows:

1. Calculate \( c \), the centroid of the triangular plate
2. Let \( p(1..3, i) = P_i - c, i = [1, 2, 3] \)
3. Let \( p_r = \begin{bmatrix} -1 & 2 & -1 \\ -1 & -1 & 2 \\ 0 & 0 & 0 \end{bmatrix}, \) the element triangular plate.
4. Use Singular Value Decomposition (SVD) to find the rotation matrix between \( p \) and \( p_r \).

The SVD rotation matrix algorithm is unchanged.

The transformation matrix for the triangular plate element is created by using Equations (4.111) and (4.112), in a more generic manner than the beam element case. Using the local partial derivative of the global displacement functions as was done so for the beam element is not acceptable for triangular membrane elements, as the generation of a shape of any significance using only triangular membrane elements necessitates rotation about the \( z \)-axis, which implies that local \( x \)'s and \( y \)'s of different elements adjacent to each other are unlikely to align with each other. Here the fact that the membrane elements used here are flat or near-flat initially render \( Z \) partial derivative terms insignificant. Using the last line of Equation (4.113), and neglecting all \( \frac{\partial}{\partial y} \) terms, the generalized coordinates for a triangular element node can be written as:

$$\begin{bmatrix}
  u \\
  \frac{\partial u}{\partial y} \\
  \frac{\partial u}{\partial x}
\end{bmatrix} \approx \begin{bmatrix}
  C_{11} & 0 & 0 & C_{12} & 0 & 0 & C_{13} & 0 & 0 \\
  0 & C_{11}C_{22} & -C_{11}C_{21} & 0 & C_{12}C_{22} & -C_{12}C_{21} & 0 & C_{13}C_{22} & -C_{13}C_{21} \\
  0 & -C_{11}C_{12} & C_{11}C_{11} & 0 & -C_{12}C_{12} & C_{12}C_{11} & 0 & -C_{13}C_{12} & C_{13}C_{11} \\
  C_{21} & 0 & 0 & C_{22} & 0 & 0 & C_{23} & 0 & 0 \\
  0 & C_{21}C_{22} & -C_{21}C_{21} & 0 & C_{22}C_{22} & -C_{22}C_{21} & 0 & C_{23}C_{22} & -C_{23}C_{21} \\
  0 & -C_{21}C_{12} & C_{21}C_{11} & 0 & -C_{22}C_{12} & C_{22}C_{11} & 0 & -C_{23}C_{12} & C_{23}C_{11} \\
  C_{31} & 0 & 0 & C_{32} & 0 & 0 & C_{33} & 0 & 0 \\
  0 & C_{31}C_{22} & -C_{31}C_{21} & 0 & C_{32}C_{22} & -C_{32}C_{21} & 0 & C_{33}C_{22} & -C_{33}C_{21} \\
  0 & -C_{31}C_{12} & C_{31}C_{11} & 0 & -C_{32}C_{12} & C_{32}C_{11} & 0 & -C_{33}C_{12} & C_{33}C_{11}
\end{bmatrix} \begin{bmatrix}
  U \\
  \frac{\partial U}{\partial y} \\
  \frac{\partial U}{\partial x}
\end{bmatrix}$$

\( = \begin{bmatrix}
  T_{11} & T_{12} & T_{13} \\
  T_{21} & T_{22} & T_{23} \\
  T_{31} & T_{32} & T_{33}
\end{bmatrix} q_n \tag{4.119}
\]

where \( T_{ij} \) are the \( 3 \times 3 \) block components of the transformation matrix for a node. The full triangular
element transformation can then be written as follows:

\[
\mathbf{q}_e = \begin{bmatrix}
T_{11} & 0 & 0 & T_{12} & 0 & 0 & T_{13} & 0 & 0 \\
0 & T_{11} & 0 & 0 & T_{12} & 0 & 0 & T_{13} & 0 \\
0 & 0 & T_{11} & 0 & 0 & T_{12} & 0 & 0 & T_{13} \\
T_{21} & 0 & 0 & T_{22} & 0 & 0 & T_{23} & 0 & 0 \\
0 & T_{21} & 0 & 0 & T_{22} & 0 & 0 & T_{23} & 0 \\
0 & 0 & T_{21} & 0 & 0 & T_{22} & 0 & 0 & T_{23} \\
T_{31} & 0 & 0 & T_{32} & 0 & 0 & T_{33} & 0 & 0 \\
0 & T_{31} & 0 & 0 & T_{32} & 0 & 0 & T_{33} & 0 \\
0 & 0 & T_{31} & 0 & 0 & T_{32} & 0 & 0 & T_{33}
\end{bmatrix}
\]

where \( \mathbf{q}_e \) has the form introduced in Equation (4.107) and \( \mathbf{T}_m \) is the triangular membrane transformation matrix.

A displaced version of the above-described functions exist, named \texttt{beam_node_gen_displaced} and \texttt{tri_node_gen_displaced}, meant for providing a catenary curve for in-plane and/or out-of-plane direction, similar to illustrated in Figure 4.6. The curve can be described by the following equation:

\[
\Delta = \frac{\Delta}{L} x^2 - L \Delta
\]

where \( \Delta \) is a unitless parameter that describes the depth of the curve and \( L \) is the length of the beam as defined in Figure 4.6. However, the above equation describes the curve in a frame where \( x = 0 \), which indicates the center of the line, whereas the coordinates of the end nodes may not be in such a frame. For a beam, this can easily be fixed by taking the following steps:

1. Let \( p_i = \frac{i-1}{n}(P_2 - P_1) + P_1 - c \), where \( c \) is the centroid,

2. Let \( p_{e,i} = C p_i \), i.e. rotate \( p_i \) by \( C \) from above.

3. Calculate local node coordinates as \( p_{e}(2) = p_{e}(2) + \frac{\Delta}{L} (p_{e}(1))^2 - L \Delta \) and \( p_{e,i}(3) = p_{e,i}(3) - \frac{1}{L} (p_{e,i}(1))^2 - L \Gamma \), where \( \Delta \) and \( \Gamma \) are unitless depth of the curve in-plane and out-of-plane respectively.

4. Then the actual node coordinate \( n_i = C^T p_{e,i} + c \).

The steps are more involved with the triangular plate. As seen in Figure 4.7, there are local displacements going on about the edge of the plate, as well as overall plate displacement. The two edges
attached to the sail boom sags between the sail’s support cords in the \( z \)-axis direction. The overall sag of the plate is defined to be dependent on the length between the widening beams, increasing as the distance from the frame origin increases. The general approach is as follows: the triangular plate is rotated such that the edge to be sloped down towards is along the \( y \)-axis. In the case of a square sail, this is the long diagonal edge. The edge displacements are then calculated, and from the displaced edge nodes, the internal node displacements are calculated as well. The rotation matrix that achieves the first step is calculated as follows:

1. Let \( \mathbf{p}(1..3, i) = \mathbf{P}_i - \mathbf{c}, i = [1, 2, 3] \)

2. Let \( \mathbf{p}_r = \begin{bmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \).

3. Use Singular Value Decomposition (SVD) to find the rotation matrix \( \mathbf{C} \) between \( \mathbf{p} \) and \( \mathbf{p}_r \).

This is similar to the steps taken to calculate the element rotation matrix, the only difference being the matrix \( \mathbf{p}_r \). Once this rotation matrix is calculated, the following steps are added during calculation of each node’s coordinates:

1. Calculate \( \mathbf{P}_{n, \text{form}(n+1,i)} \), \( \mathbf{P}_{n, \text{form}(i,1)} \) and \( \mathbf{P}_{n, \text{form}(i,i)} \) as given by Equation (4.117),

2. Calculate \( \mathbf{p}_{e,j} = \mathbf{C} \mathbf{P}_j \), where \( j \) is the node number,

3. Calculate local node coordinates: for the diagonal edge, \( \mathbf{p}_{e,j}(3) = \mathbf{p}_{e,j}(3) + \frac{\Delta}{L}(\mathbf{p}_{e,j}(2))^2 - L\Delta \) and \( \mathbf{p}_{e,j}(2) = \mathbf{p}_{e,j}(2) + \frac{\Gamma}{L}(\mathbf{p}_{e,j}(2))^2 - L\Gamma \), where \( \Delta \) and \( \Gamma \) are unitless depth of the diagonal curve in- and out-of-plan, and for the other two edges, \( \mathbf{p}_{e,j}(3) = \mathbf{p}_{e,j}(3) + \frac{\Omega}{L_s}(\text{mod}(\mathbf{p}_{e,j}(1), 2L_s) - L_s)^2 - L_s\Omega \).

4. Calculate the internal nodes \( \mathbf{p}_{n, \text{form}(i,j)} \) as given by Equation (4.118),

5. Calculated the internal node displacements as follows: \( \mathbf{p}_{e,j}(3) = \mathbf{p}_{e,j}(3) + \frac{\Delta}{L}(\mathbf{p}_{e,j}(2))^2 - L\Delta \)

6. For all of the above, the actual node coordinates are calculated as \( \mathbf{n}_j = \mathbf{C}^T \mathbf{p}_{e,j} \).

\( L \) is the length of the non-diagonal side and \( L_s \) is the length between two cords.

\[ \begin{align*}
\Delta & = \text{unitless depth of the diagonal curve in- and out-of-plan} \\
\Gamma & = \text{unitless depth of the diagonal curve in- and out-of-plan} \\
\Omega & = \text{unitless depth of the diagonal curve in- and out-of-plan} \\
\end{align*} \]

\[ \begin{align*}
\mathbf{p}_{n, \text{form}(n+1,i)} & = \mathbf{P}_i - \mathbf{c}, i = [1, 2, 3] \\
\mathbf{p}_r & = \begin{bmatrix} 0 & 1 & 1 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} \\
\mathbf{C} & = \text{rotation matrix} \\
\mathbf{p}_{e,j} & = \mathbf{C} \mathbf{P}_j, j \text{ is the node number} \\
\mathbf{p}_{e,j}(3) & = \mathbf{p}_{e,j}(3) + \frac{\Delta}{L}(\mathbf{p}_{e,j}(2))^2 - L\Delta \\
\mathbf{p}_{e,j}(2) & = \mathbf{p}_{e,j}(2) + \frac{\Gamma}{L}(\mathbf{p}_{e,j}(2))^2 - L\Gamma \\
\mathbf{p}_{e,j}(3) & = \mathbf{p}_{e,j}(3) + \frac{\Omega}{L_s}(\text{mod}(\mathbf{p}_{e,j}(1), 2L_s) - L_s)^2 - L_s\Omega \\
\mathbf{n}_j & = \mathbf{C}^T \mathbf{p}_{e,j} \\
\end{align*} \]
All of the node coordinate information generated above is stored in an array of Node objects in SailStruct. Prior to any generation of stiffness or inertial matrix coefficients, the BeamElement or TriElement arrays in SailStruct is initialized as well, providing them dimensional info - the sizing of the element, as well as the location of the element’s nodes, by providing pointers to the Node object belonging to the element. In the case of a beam, this is simply a matter of counting up - i.e. provide pointers to nodes 1 and 2 for the first element, 2 and 3 for the second element, and so forth. The nodes and element numbering is demonstrated by Figure 4.8, where \( N_i \) represents a node labeling and \( E_i \) represents an element labeling.

To assign nodes to elements of a triangular plate, \( n_{form} \) array is made use of. Specifically, refer back to the following:

\[
\begin{bmatrix}
15 & 0 & 0 & 0 & 0 \\
13 & 14 & 0 & 0 & 0 \\
10 & 11 & 12 & 0 & 0 \\
6 & 7 & 8 & 9 & 0 \\
1 & 2 & 3 & 4 & 5
\end{bmatrix}
\]

and note that the elements and the corresponding nodes can be associated by taking a \( 2 \times 2 \) submatrix of the above and checking which of the array components are non-zero, then manually assigning the nodes based on the node number specified by \( n_{form} \). For example, if a submatrix \( n_{form}(2..3,1..2) \) is taken, one can see that it has no non-zero components, then the nodes 10, 11 and 13 can form one triangular plate element, while the nodes 14, 13 and 11 form another triangular plate element. Similar checks can be made to a submatrix \( n_{form}(3..4,3..4) \) - this time there is one non-zero component, which corresponds to one triangular plate composed of nodes 8, 9 and 12. These checks can be made from top to bottom, resulting in element number assignment as illustrated by Figure 4.9.
Once the **BeamElement** or **TriElement** arrays are initialized, the element stiffness matrices can be generated.

### 4.4.2 Element Stiffness Matrix Generation

The general equations for the stiffness matrix coefficients are already known from Sections 4.2 and 4.3, but the equations are all in terms of the generic shape function \( \psi \). In the case of the beam, it is feasible to calculate the shape function and the stiffness matrix terms - the worst case is the coefficient calculation for \( N_2 \), requiring combination of 9 expressions with repetitions, i.e. \( \binom{9+9-1}{9} = 24310 \) total expressions - a rather unmanageable number of expressions to calculate manually. However, the previous section on **Polynom** module was created precisely to avoid this problem.

Specifically, consider the shape function definitions from Equations (4.79). Since the matrix \( A^{-1} \) and the row matrix \( X_b \) are known in a polynomial form, these values are initialized as **Polynom** arrays, then its components are multiplied together with **mult** in a nested-loop that performs matrix multiplication of polynomials. It should be noted that because the **Polynom** module does not have a matrix inverse functionality implemented, matrix inversion for \( A^{-1} \) term must be performed elsewhere, and its results input directly into **Polynom** format. In this case, Matlab’s Symbolic Math package is used to perform symbolic matrix inversion on \( A \).

The calculation of the other **Polynom** arrays listed in **BeamStruct** and **TriStruct** is mostly straightforward, with an element-specific caveat. In particular, the generation of the stiffness matrices require line/area integrals and derivatives of the Cartesian variables \( x \) and \( y \) as shown in Equations (4.75) and (4.99), whereas \( \psi_b \) and \( \psi_t \) are functions of \( \hat{x} \) and \( L_1, L_2, L_3 \) respectively. Hence the derivatives and integrals with respect to \( x \) or \( y \) must be converted to the respective operations with respect to \( \hat{x} \), \( L_1 \), \( L_2 \) and \( L_3 \). For the beam, one can show fairly easily from the relationship \( x = l \hat{x} \) that

\[
\frac{\partial f}{\partial x} = \frac{1}{l} \frac{\partial f}{\partial \hat{x}} \quad \int_0^l f \, dx = l \int_0^1 f \, d\hat{x}
\]  

(4.122)

which can easily be implemented by multiplying the results of the generic differentiation or integration functions by \( \frac{1}{l} \) or \( l \) respectively. For the triangular membrane with natural coordinate shape function, we take the natural coordinates defined with respect to the Cartesian coordinates as given in Equation (4.102). The partial derivatives then can be written as follows:

\[
\begin{align*}
\frac{\partial f}{\partial x} &= \frac{\partial}{\partial L_1} \frac{\partial L_1}{\partial x} + \frac{\partial}{\partial L_2} \frac{\partial L_2}{\partial x} + \frac{\partial}{\partial L_3} \frac{\partial L_3}{\partial x} = \frac{1}{2A} \left[ b_1 \frac{\partial f}{\partial L_1} + b_2 \frac{\partial f}{\partial L_2} + b_3 \frac{\partial f}{\partial L_3} \right] \\
\frac{\partial f}{\partial y} &= \frac{\partial}{\partial L_1} \frac{\partial L_1}{\partial y} + \frac{\partial}{\partial L_2} \frac{\partial L_2}{\partial y} + \frac{\partial}{\partial L_3} \frac{\partial L_3}{\partial y} = \frac{1}{2A} \left[ a_1 \frac{\partial f}{\partial L_1} + a_2 \frac{\partial f}{\partial L_2} + a_3 \frac{\partial f}{\partial L_3} \right]
\end{align*}
\]  

(4.123)

The formula for definite integration of natural coordinates is given without proof as follows [45]:

\[
\int_A \int_A \left( L_1^\alpha L_2^\beta L_3^\gamma \right) dA = 2A \frac{\alpha! \beta! \gamma!}{(\alpha + \beta + \gamma + 2)!}
\]

(4.124)

It should be noted that the above formula is not applicable to triangles with curved sides. A special function called **tri_int_nat** is used to implement the above-given natural coordinate integration formula, as it is different from the generic integration formula provided by the **Polynom** module.

With the above integration formulas, a series of lengthy analytical integration processes takes place.
pre-simulation to generate large quantities of coefficients to be used during simulation, its details in the Appendix. The actual two dimensional stiffness matrix generation is performed using these pre-generated coefficients along with the values for $q_e$. Because the majority of the linear combinations have already been performed during the pre-calculation phase, the generation of the element matrix is a deceptively simple matter. Specifically, the following summations occur for $\text{BeamElement}$: $\frac{1}{2} N_1(i,j,k)q_k$ and $\frac{1}{2} N_2(i,j,k,l)q_kq_l$, resulting in two-dimensional matrices. These values are added together to $K_L$, the result of which is the nonlinear stiffness matrix. For $\text{TriElement}$, instead of just one matrix, there are six matrices for each degree of nonlinearities, but the general idea remains the same. The equations to calculate the stiffness matrix $K$ are as follows:

$$K = KL + N_1 + N_2$$

$$KL = aKa + bKb + cKc + dKd + eKe + fKf$$

$$N_1(i,j) = aN_1a(i,j,k)q_k + bN_1b(i,j,k)q_k + cN_1c(i,j,k)q_k + dN_1d(i,j,k)q_k + eN_1e(i,j,k)q_k + fN_1f(i,j,k)q_k$$

$$N_2(i,j) = aN_1a(i,j,k,l)q_kq_l + bN_1b(i,j,k,l)q_kq_l + cN_1c(i,j,k,l)q_kq_l + dN_1d(i,j,k,l)q_kq_l + eN_1e(i,j,k,l)q_kq_l + fN_1f(i,j,k,l)q_kq_l$$

One of the major sources of calculation time derives from the double summation performed on each element of the $N_2$ matrix, but not much can be done - the calculations must be performed during simulation nonetheless due to its strong dependence on $q_e$.

### 4.4.3 Element Mass Matrix and Inertial Force Generation

The generation of the mass matrices and the inertial forces is similar to stiffness matrix generation in that they also involve integrating the $\text{Polynom}$ arrays. The two are distinct in a sense that many of the coefficients involved in calculating the mass matrix and the inertial forces involve the variable $\rho$, which describes the distance from the body frame’s origin to a point on the element. Hence the volume integration becomes slightly more involved and specific with every element. Having said that, every term in this category is at most two dimensional, and the polynomial calculation of these terms can be performed fast enough to be considered insignificant to the simulation time.

Assuming the element stiffness matrix generation process described in the previous section are performed, $\text{Polynom} f$, representative of the shape function $\psi$, is already initialized. Generation of the terms in Section 4.1.2 requires calculation of the integral Equations (4.35), (4.40), and (4.45). The coordinate $\rho$ must be defined in terms of the variables being integrated in order to solve the integrals: for beams, $\hat{x}$, and for the triangular plates, $L_1$, $L_2$ and $L_3$. For the beam, the $\rho$ term is defined as follows:

$$\rho = n_1 + \begin{bmatrix} \frac{L}{2} \\ 0 \\ 0 \end{bmatrix}$$

where $n_1$ contains the coordinates of the first node of the beam element in the body frame coordinates, but rotated to the element frame so that the increment in $\hat{x}$ corresponds to traversing the length of the beam. For the triangular plate, $\rho$ term uses the inverse relationship of Equation (4.102). Specifically, the following steps are taken to generate $\rho$:
1. Calculate the element centroid $c$.

2. Calculate $n_i = C(N_i - c)$, where $N_i$ are the node coordinates in body frame and $C$ is the element rotation matrix.

3. Rotate the element centroid with $C$, i.e. $c_e = Cc$.

4. Calculate $\rho = \begin{bmatrix} n_1 & n_2 & n_3 \end{bmatrix} \begin{bmatrix} L_1 \\ L_2 \\ L_3 \end{bmatrix} + c_e$

It should be noted that these steps render the $\rho(3)$ term constant, as it should be for a plate element defined in the $x$-$y$ plane.

The above process is used to create $\text{rho}$, a Polynom representation of $\rho$, and is used just as $f$ is. Using the two Polynom arrays, the integrals (4.35), (4.40), and (4.45) are calculated. These calculated expressions are stored in floating point arrays defined in $\text{Element}$.

Similar to calculating the stiffness matrix in the previous section, the actual calculation of the mass matrix and the inertial forces requires knowledge of the current $q_e$. These calculations, as defined by Equations (4.49), (4.50), (4.53) and (4.54), are straightforward to perform. It should be noted that $M_{rr}$, $M_{re}$, $M_{ee}$, $f_{lr}$ and $f_{Ir}$ are kept separate - that is, the rigid dynamics terms are not merged with the elastic dynamics term during the element-level generation stage, and instead passed to the function caller in separate forms. This was done because during the global matrix generation process, the rigid terms for the most part add to each other, while the elastic parts will have specific indices within the matrix that they must add to.

### 4.4.4 Global Matrix Generation

Generation of global matrices for a beam divided into multiple elements is simple and often illustrated in introductory FEM textbooks. Generation of global matrices for triangular plates is trickier, as one node ends up being shared by up to 6 elements at a time, and the system needs to be able to account for this when building the global matrix. When the structure consists of multiple types of elements, it also needs to be able to account for how different elements are connected to each other. Some displacements are fixed, which also need to be accounted for during the matrix generation. Finally, there is a matter of storage.

First, consider how to combine element matrices of the same type together - i.e. the array of elements within a $\text{SaIlStruct}$ object. As indicated in the previous sections, the element matrix coefficients are determined during the pre-simulation phase, while the actual element matrices are calculated during the simulation due to their dependence on $q_e$. Once the element matrices are determined however, a problem remains: How can one determine which element matrices belong where in the global matrix?

#### Matrix Indexing Generation

The element matrices are added together at the node level - that is, the portion of the element matrices sharing the same node should share the same indices in the global matrix, with each node having a unique set of indices that correspond to the node’s degrees of freedom. For example, assume a $6 \times 6$ beam element matrix with three degrees of freedom per node - $x$, $y$ and $z$ deflections. Let’s also assume
that this is a beam between nodes 2 and 4, with global indexing of [4..6] and [10..12] assigned to each node. Then the element matrix $\mathbf{M}$ would be assigned to the global matrix with the following indices:

$$
\begin{bmatrix}
\vdots \\
4 & M_{11} & M_{12} & M_{13} & \cdots & M_{14} & M_{15} & M_{16} & \cdots \\
5 & M_{21} & M_{22} & M_{23} & \cdots & M_{24} & M_{25} & M_{26} & \cdots \\
6 & M_{31} & M_{32} & M_{33} & \cdots & M_{34} & M_{35} & M_{36} & \cdots \\
10 & M_{41} & M_{42} & M_{43} & \cdots & M_{44} & M_{45} & M_{46} & \cdots \\
11 & M_{51} & M_{52} & M_{53} & \cdots & M_{54} & M_{55} & M_{56} & \cdots \\
12 & M_{61} & M_{62} & M_{63} & \cdots & M_{64} & M_{65} & M_{66} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots
\end{bmatrix}
$$

Mimicking the above behaviour is a matter of having the function generating the element matrix be aware of these indexing values and assigning the components of the element matrix onto the appropriate indices on the global matrix using those values. In case of the designed simulator, a special indexing array named $\text{d\_index}$ is created for each element through a process described in the Appendix such that, if $\mathbf{M_e}$ is assumed to be the element matrix and $\mathbf{M}$ the global matrix, the following simple line is used to assign the element matrix to the global matrix:

$$
\mathbf{M}(\text{d\_index}, \text{d\_index}) = \mathbf{M}(\text{d\_index}, \text{d\_index}) + \mathbf{M_e}
$$

I.e. $\text{d\_index}$ contains the information about which rows and columns in the global matrix the contents of the element matrix belong.

### Nodes with Displacement Restrictions

Some structures have nodes that are restricted in displacement about certain directions. Most commonly, these nodes are fixed such that no movement is allowed, or pinned such that Cartesian deflections are not allowed but angular deflections may still occur. In FEM, these behaviours are created by removing the restricted displacements from the global matrix. An array named $\text{matrix\_index}$ is generated for this purpose - this array contains the index information about which columns and rows of the global matrix are not restricted, which can be used to quickly remove any restricted indices as needed.

### The Global Matrix Generation Process in Simulation

Once all of the above are generated prior to the simulation, the actual global matrix generation process can be summarized as follows:

1. Loop for every SailStruct and Element:
   
   (a) Use the element matrix generator function to generate an element matrix. If needed, provide local generalized coordinates by indexing it using the Element’s $\text{d\_index}$ array.
   
   (b) Add the output of the element matrix generator function to the unrestricted global matrix, using $\text{d\_index}$ to assign the element matrix to the proper location on the global matrix.

2. Assign the unrestricted global matrix to the true global matrix, using $\text{matrix\_index}$ to trim out restricted displacements.
It should be noted that the element matrix generator function takes into account the rearrangement and rotation described in Section 4.3.2 and several subsections above this one.

### 4.4.5 Dynamics Simulation Via Numerical Differentiation Formulas

The dynamics simulation is performed by using first-order Ordinary Differential Equation (ODE) solvers. Since Equation (4.56), the equations of motion, is a second order ODE, it is rearranged into the following form instead [12]:

$$\mathbf{M}_g \ddot{\mathbf{q}}_g = \mathbf{f}_{T,g} + \mathbf{f}_{I,g} - \mathbf{K}_g \mathbf{q}_g$$  \hspace{1cm} (4.127)

which can be expanded into a matrix form as

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \mathbf{M}_{ee} & \mathbf{M}_{re} \\ 0 & \mathbf{M}_{re}^T & \mathbf{M}_{rr} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{q}}_e \\ \dot{\mathbf{q}}_r \\ \dot{\mathbf{q}}_n \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{f}_{T,e} \\ \mathbf{f}_{T,r} \end{bmatrix} + \begin{bmatrix} 0 & -1 & 0 \\ \mathbf{f}_{I,e} & \mathbf{K}_{ee} & 0 \\ \mathbf{f}_{I,r} & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{q}_e \\ \mathbf{q}_r \\ \mathbf{q}_n \end{bmatrix}$$  \hspace{1cm} (4.128)

and is a first-order ODE with respect to the combined variable \( \mathbf{q}_g = [\dot{\mathbf{q}}_e \quad \dot{\mathbf{q}}_r \quad \dot{\mathbf{q}}_n]^T \). Similarly, the kinematics equation from Section 4.1.3 is also rearranged to resemble the above-form - specifically,

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\mathbf{q}}_n \end{bmatrix} = \begin{bmatrix} \left((1 - 2\epsilon^T \boldsymbol{e}) \mathbf{I}_3 + 2\epsilon \mathbf{v}^T - 2\eta \mathbf{e}^T \mathbf{v} \right) \\
\frac{1}{2} \left(\epsilon \mathbf{I}_3\right) \omega \\
-\epsilon^T \omega \end{bmatrix}$$  \hspace{1cm} (4.129)

where the identity matrix at the front of the left-hand side is there to mimic the mass matrix in Equation (4.128).

As mentioned previously, the solution to the equations of motion is provided by \texttt{ode15s}, an ODE solver from Matlab that has been reproduced in Fortran language. The function \texttt{ode15s} uses the Numerical Differentiation Formulas (NDS), a modified version of the Backwards Differentiation Formula (BDS) as described in Shampine and Reichelt [46]. The theory and implementation details of the NDS and the BDS are not within the scope of this research, and are omitted. However, it should be noted that \texttt{ode15s} requires the Jacobian of the right-hand side of the above equation - whether it be exact or approximate. In this case, the exact Jacobian can be provided. The Jacobian of the stiffness matrix \( \mathbf{J}_K \) has already been defined in Equation (4.24). As for the inertial force terms, their Jacobians are simply calculated from their definitions. The various derivatives of the inertial forces are as follows:

$$\frac{\partial \mathbf{f}_{I,r}}{\partial \mathbf{q}_\beta} = \begin{bmatrix} -\omega^\times \omega^\times \mathbf{P}_\beta \\ (\omega^\times \mathbf{v})^\times \mathbf{P}_\beta - \omega^\times \left(-(\tau_\beta + \tau_\beta^T) - \mathbf{Y}_{\beta \alpha} \mathbf{q}_\alpha \right) \right) \omega \end{bmatrix}, \quad \frac{\partial \mathbf{f}_{I,r}}{\partial \mathbf{v}} = \begin{bmatrix} -m \omega^\times \\ -(c + \delta c)^\times \omega^\times \end{bmatrix}$$  \hspace{1cm} (4.130)

$$\frac{\partial (\delta \mathbf{f}_{I,r})}{\partial \mathbf{q}_\beta} = \begin{bmatrix} 0 \\ 2 \mathbf{Y}_{\alpha \beta}^T \omega \dot{\mathbf{q}}_\alpha \end{bmatrix}, \quad \frac{\partial (\delta \mathbf{f}_{I,r})}{\partial \mathbf{q}_\beta} = \begin{bmatrix} -2\omega^\times \mathbf{P}_\beta \\ 2\tau_\beta^T \omega \end{bmatrix}, \quad \frac{\partial (\delta \mathbf{f}_{I,r})}{\partial \mathbf{v}} = \begin{bmatrix} 2(\mathbf{P}_\alpha \dot{\mathbf{q}}_\alpha)^\times \\ 2\tau_\alpha^T \dot{\mathbf{q}}_\alpha \end{bmatrix}$$  \hspace{1cm} (4.131)
The above equations are then arranged into a Jacobian form as follows:

\[
\frac{\partial f_{I,e}}{\partial \dot{q}_a} = \text{col} \left\{ -\omega^T \mathbf{y}_{\alpha \beta} \omega \right\}, \quad \frac{\partial f_{I,e}}{\partial v} = \text{col} \left\{ -P^T_{a} \omega^x \right\}, \quad \frac{\partial f_{I,e}}{\partial \omega} = \text{col} \left\{ P_a \mathbf{v}^x - \omega^T (\mathbf{r}_a + \mathbf{\tau}_a^T) \right\}
\]

\[
\frac{\partial (f_{I,e})}{\partial q} = \text{col} \left\{ 2\nu_{\beta a} \dot{\omega} \right\}, \quad \frac{\partial (f_{I,e})}{\partial \dot{q}_a} = \text{col} \left\{ 2\nu_{\beta a} \dot{q}_a \right\}
\]

\[
(4.132)
\]

The global Jacobian matrix generation is performed in a similar manner as the matrix generation of \( \mathbf{M} \) or \( \mathbf{K} \) in that the method mentioned in Section 4.4.4 still applies. The actual arranged form for the global Jacobian matrix of the right-hand side is as follows:

\[
\mathbf{J}_{fl} = \begin{bmatrix}
0 & -1 & 0 \\
\mathbf{J}_{fl,e,q_n} & \mathbf{J}_{fl,e,q_n} & \mathbf{J}_{fl,e} \\
\mathbf{J}_{fl,r,q_n} & \mathbf{J}_{fl,r,q_n} & \mathbf{J}_{fl,r}
\end{bmatrix}
\]

\[
(4.134)
\]

The Jacobians of the kinematics equation are written as follows:

\[
\mathbf{J}_{q_n,q_n} = \frac{1}{2} \begin{bmatrix}
-\omega^x & \omega \\
-\omega^T & 0
\end{bmatrix}, \quad \mathbf{J}_{q_n,r} = \frac{1}{2} \begin{bmatrix}
0 & \epsilon^x + \eta \mathbf{l}_3 \\
0 & -\epsilon^T
\end{bmatrix}
\]

\[
\mathbf{J}_{r,q_n} = \frac{1}{2} \begin{bmatrix}
( -2 \mathbf{v}^T + \epsilon \mathbf{v}^T + \mathbf{v}^T \mathbf{l}_3 - \eta \mathbf{v}^x ) \\
2 \mathbf{v}^T \mathbf{v}
\end{bmatrix}, \quad \mathbf{J}_{r,r} = \begin{bmatrix}
((1 - 2 \epsilon \mathbf{e}) \mathbf{l}_3 + 2 \epsilon \mathbf{e}^T - 2 \eta \mathbf{e}^x)^T \\
0
\end{bmatrix}
\]

\[
(4.135)
\]

where it should be noted that \( r \) and \( r \) are unrelated notations - \( r \) refers to the inertial displacement, while \( r \) refers to the rigid dynamics equations of motion, i.e. terms related to \( \mathbf{v} \) and \( \omega \).

Given the above, we can now define the precise inputs to the ODE solver. First, the states to be solved by the ODE solver are defined as \( \mathbf{q}_e \quad \dot{\mathbf{q}}_e \quad \mathbf{\xi} \quad \mathbf{\mathbf{r}} \quad \mathbf{q}_n \) in the given order. The solver is given the initial state, the time span at which the solution is derived for, and four functions of abstract type \texttt{ode\_f} and \texttt{ode\_df} - an external force generator, one that generates the mass matrix, another that generates the right-hand side, and last that generates the Jacobian of the right-hand side. These function are the contents of \texttt{ODEFnc\_Module}, named \texttt{f\_ext}, \texttt{model\_mass}, \texttt{model\_y} and \texttt{model\_J} respectively.

The function \texttt{model\_mass} generates the following matrix:

\[
\begin{bmatrix}
1 & 0 & 0 & 0 & 0 \\
0 & M_{ee} & M_{e} & 0 & 0 \\
0 & M_{r}^T & M_{rr} & 0 & 0 \\
0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1
\end{bmatrix}
\]

\[
(4.136)
\]

where the last two rows and columns are added to account for the kinematics. The function \texttt{model\_y}
Chapter 4. Unconstrained Nonlinear Flexible Structural Dynamics

\begin{equation}
\begin{bmatrix}
\dot{q}_{e} \\
\mathbf{f}_{T,e} + \mathbf{f}_{I,e} - \mathbf{K}_{ee}\mathbf{q}_{e} - \mathbf{D}\mathbf{q}_{e} \\
\mathbf{f}_{T,r} + \mathbf{f}_{I,r} \\
\frac{1}{2} \left[ (\epsilon + \eta) \mathbf{l}_{3} \right] \omega \\
(1 - 2\epsilon^{T}\mathbf{1}_{3} + 2\epsilon\epsilon^{T} - 2\eta\epsilon^{x})^{T} \mathbf{v}
\end{bmatrix}
\end{equation}

where \( \mathbf{f}_{T,r} \) and \( \mathbf{f}_{T,e} \), the external force terms, are generated by \( \mathbf{f}_{\text{ext}} \). The function \( \text{model}_J \) generates the following:

\begin{equation}
\begin{bmatrix}
0 & -1 & 0 & 0 & 0 \\
\mathbf{J}_{fI,e,\mathbf{q}_{e}} - \frac{\partial \mathbf{K}_{ee}}{\partial \mathbf{q}_{e}} & \mathbf{J}_{fI,e,\mathbf{q}_{e}} & \mathbf{J}_{fI,e,r} & 0 & 0 \\
\mathbf{J}_{fI,r,\mathbf{q}_{e}} & \mathbf{J}_{fI,r,\mathbf{q}_{e}} & \mathbf{J}_{fI,r,r} & 0 & 0 \\
0 & 0 & \mathbf{J}_{r,r} & 0 & \mathbf{J}_{r,\mathbf{q}_{e}} \\
0 & 0 & 0 & \mathbf{J}_{\mathbf{q}_{e},r} & \mathbf{J}_{\mathbf{q}_{e},\mathbf{q}_{e}}
\end{bmatrix}
\end{equation}

Notice that the global mass matrix and the Jacobian contain significant portions of zeros, but for a dense matrix, these zero values need to be represented. The solver uses these generated matrices to eventually output a converged solution to the states for a particular time step. In the case of the current code, it is set up such that only a time step composed of two values are fed into \( \text{ode15s} \) currently, and multiple calls to \( \text{ode15s} \) with updated time steps are made via a loop. This allows the solution to a single time-step to be exported prior to the completion of the entire simulation, allowing the results of the simulation to be viewed as the simulation progresses. This also allows miscellaneous adjustments to be made to the system as the simulation progresses - in particular, the wrinkling parameters and the control forces, as will be detailed later on.

It should also be noted that there are steps that require solving a linear equation of the form \( \mathbf{Ax} = \mathbf{b} \) for \( \mathbf{x} \), and in the case of the implemented simulation, this is done by taking the LU decomposition of \( \mathbf{A} \) and multiplying the inverted \( \mathbf{L} \) and \( \mathbf{U} \) to the right-hand side. This decomposition and equation solving process is one of the more time-consuming processes in the simulation, and is one of the prime reasons for shifting to the sparse matrix system described below. The actual decomposition process is performed by LAPACK's \text{DGESTRF} \ function, while the inversion process is performed by \text{DTRTRI} \ function. Details about these functions' usages can be found in LAPACK Users' Guide - Linear Equations [4].

Justification for the Choice of the Simulator

In Matlab, there are several viable choices for simulators that can be used to solve a first-order ODE presented above - in particular, \( \text{ode45} \), using a Runge-Kutta method, is a popular choice for solving such problems. In addition, there is also a question of how much providing \( \text{ode15s} \) the mass matrix and the Jacobian of the right-hand side directly improves the performance of the integrator. These questions have been answered when the simulator was still in Matlab script form.

Figure 4.10 is the simulation result of a spinning beam problem that will be presented in the next section to verify the validity of the Fortran simulator. Specifically, \( \text{ode45} \) and \( \text{ode15s} \) with three different setups - one without providing the mass matrix and Jacobian, one providing only the mass matrix, and the last providing both, have been used to simulate the system. In the case of not being provided the mass matrix, it has been inverted and multiplied to the results of \( \text{model}_y \) instead. In the case of not
being provided the Jacobian matrix ode15s uses its internal numerical Jacobian calculator instead. The results are reasonably similar to each other.

Figure 4.10: Transverse Displacement of the Spinning Beam

However, the total time taken to provide the simulation results, as shown in Table 4.1, is significantly different. In particular, the ode15s case that uses the provided Jacobian matrix generator is significantly faster than the cases that do not. Also, ode45 is significantly slower than all cases using ode15s. Hence, based on the above results, ode15s was chosen to be used for all future simulations. This is the reason behind ode15s being implemented in Fortran over other ODE solvers.

<table>
<thead>
<tr>
<th>Integrator</th>
<th>Mass</th>
<th>Jacobian</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>ode15s</td>
<td>No</td>
<td>No</td>
<td>61.13</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>No</td>
<td>54.39</td>
</tr>
<tr>
<td></td>
<td>Yes</td>
<td>Yes</td>
<td>28.76</td>
</tr>
<tr>
<td>ode45</td>
<td>Yes</td>
<td>Yes</td>
<td>3690.59</td>
</tr>
</tbody>
</table>

Table 4.1: Simulation Time (s) for Varying Number of Elements and Integration Settings
4.4.6 Sparse Matrix System

The above system was initially designed for dense matrices, where the matrix is represented in the usual mathematical form of two-dimensional arrays, with every element assigned a memory space, including all zero elements. This form and all its associated calculations are easy to grasp and logically sound. However, such a matrix becomes overwhelmingly large in terms of the required storage as its columns and rows grow. For the case of the FEM model’s global matrices, a simple structure consisting of 20 nodes already has 120 generalized coordinates associated with it, resulting in matrices with over 10000 components. Increases in model fidelity would require hundreds of nodes, which would result in impossibly-large global matrices with very lengthy processing time. The size of the global matrices, in particular, becomes so large that it becomes surprisingly easy to run into an ‘out of memory’ error during simulation when a certain number of elements are exceeded.

To mitigate this issue, typical mathematical engines use sparse matrices - matrices with structures that only hold non-zero elements. Fortran does not have a default sparse matrix implementation, hence one with basic functionalities was implemented - it should be noted that several external sparse matrix libraries have been looked into, including SPARSEKIT, Sparse BLAS, PSPASES, PETSc, and SuperLU. Of these, SuperLU is used to perform LU decomposition and linear equation solving, but a separate sparse matrix storage is implemented.

The specific details of the implemented system can be found in the Appendix. It should be said however that the system is designed and implemented to allow access to each and every element of the sparse matrix at \(O(1)\) speed by making use of the static nature of the global matrix structure despite its nonlinearity - a crucial feature needed to generate a state-dependent global sparse matrix during simulation without slowing down the simulation.

4.5 Sample Simulation: The Rotating Beam Problem

One of the more classical examples of a nonlinear structural dynamics problem is that of a rotating beam. The problem involves describing the behaviour of a beam under high angular velocities, and has been tackled by various authors in the past [11, 12, 22, 47]. Here, the problem described by Hanagud and Sarkar [22] is observed, with small changes to the application of angular velocity - namely, torque is applied to induce a specific angular velocity instead of specifying certain values of angular velocity. As simple as the problem may sound, it demonstrates acutely the inaccuracy of a linear model under conditions of high angular velocity.

4.5.1 Problem Description

The problem can be described as follows: Given a long thin beam that is pinned at one end and free at another, gradually apply angular rotation to the beam at the pinned end until it reaches 6 rad/s, and observe the displacement experienced by the other end of the beam. The equation that describes the increase in angular rotation, along with the simulation setup, is as given in Figure 4.11 - the beam is at rest at the beginning and undergoes a sinusoidal increase to the prescribed 6 rad/s at 15 seconds. The torque required to produce the desired angular velocities \(\omega_d, G\omega_d\), is approximated by
\[ G\omega_d = J\dot{\omega}_d, \]
\[ \dot{\omega}_d(1) = \dot{\omega}_d(2) = 0, \]
\[ \dot{\omega}_d(3) = \begin{cases} 
(6/15)(1 - \cos(2\pi t/15)), & 0 \leq t \leq 15, \\
0, & t \geq 15 
\end{cases} \]  

(4.139)

where the resulting angular velocity profile is graphed in Figure 4.11. It should be noted that \( J \) given here is the constant \( J \) produced at the start of the simulation with the states equal to zero - while it may be more accurate to portray this value as being dependent upon the states, the end result of such a setup is that the simulation becomes unstable. Hence, the approximate alternative given above is used. The structural parameters used to describe this beam are as follows: beam length \( L = 10 \) m, cross-sectional area \( A = 4.601 \times 10^{-4} \text{ m}^2 \), second moment of area of the cross-section \( I = 2.931 \times 10^{-7} \text{ m}^4 \), Young’s modulus \( E = 6.895 \times 10^{10} \) N/m\(^2\) and volume density \( \rho = 2.608 \times 10^3 \) kg/m\(^3\).

\[ \omega_z \approx \frac{1}{15} \left( t - \frac{5}{2\pi} \sin \frac{2\pi t}{15} \right) \text{ rad/s, } 0 \leq t \leq 15 \text{ s} \]
\[ \omega_z = 6 \text{ rad/s, } t \geq 15 \text{ s} \]

(a) Illustration of the Rotating Beam Problem  
(b) Angular Velocity Profile of the Rigid Body Rotation  

Figure 4.11: Description of the Rotating Beam Problem

The diagram, and the problem being solved, depict the beam being rotated by a torque along one of the global Cartesian axes. However, the same results should be gleaned when the beam is rotated about some other local axes with equal torque. This is deemed to be a reasonable method to test out how robust the three-dimensional beam setup is, and is also simulated.

4.5.2 Simulation Setup

The structure that the simulation is performed on is a beam composed of one beam object, first node located at the origin and the last node located initially at \((10, 0, 0)\). The number of elements used to simulate the system is varied from 1 to 6, as per Sharf’s observation that end-node displacement convergence can be noted when using 6 elements [47]. The first node’s displacement terms are marked as restricted and removed from the system - due to the way the global matrix is built, these displacements always turn out to be the first six of the global matrix, hence their removal is straightforward. In addition, the rigid body Cartesian displacements are also constrained - i.e. the velocity is assumed to be zero at all times. The angular velocity remains free to move at will, though it is affected by the force meant to induce angular rotation. Damping coefficients are set to zero - the simulation is stable without
a need for a damping matrix. The external forces are defined in a custom function that replaces \( f_{\text{ext}} \). This function sets the rigid body torque to be as given in Equation (4.139), except that for a rotated axis of rotation case. Each simulation lasts 20 seconds with 0.1 second time-step - 200 calls to \texttt{ode15s}, with each data point recorded at every time step as mentioned earlier.

Another set of simulations was performed by rotating the axis of rotation about the \( x \)-axis in steps of \( \frac{\pi}{10} \), from 0 to \( \pi \). A third set of simulations is performed by rotating the original location of the beam with a randomly-generated rotation matrix (using the \texttt{rand} function on the angles of the three principal rotation matrices and multiplying them together). The damping coefficient is not set to zero for the beams with rotated initial position due to simulation stabilities issues. All of the rotated simulations use 6 elements.

All of the above simulations are performed using sparse matrices - a dense matrix and sparse matrix implementation performance comparisons are given at the end of this section.

### 4.5.3 Simulation Results and Comparison to Known Literature

Figure 4.12 shows the displacements experienced by the tip of the beam during the transient phase. In particular, one can observe that as the number of elements increase, the plots quickly converge to the lines described by the 6-element case. The maximum absolute axial displacement is -0.01853 m, and the maximum absolute transverse displacement is -0.5698 m, both at 6.9 seconds. The shape of the plots match up very well to those presented by Damaren and Sharf [11, 47]. The maximum tip transverse displacement noted by Sharf, 0.57 m, matches well with the maximum value observed in this simulation.

![Figure 4.12: Simulation Results for Rotating Beam Problem, with 1, 3, 5 and 6 Elements](image)

As mentioned previously, the simulations with rotated axis of rotation or initial positions have stability issues without a damping matrix - the time taken to simulate the same beam increases by about 100 times its normal speed. To avoid this issue, a small amount of diagonal damping is added. This damping does change the displacement slightly, in a manner observed in Figure 4.13. One can see that the damping ratio of \( 1 \times 10^{-3} \) does not change the displacement behaviour in a visible manner, but the
higher damping ratio of $1 \times 10^{-2}$ does so. Both of these results are calculated in comparable time to the no damping case. Hence, the damping ratio of $1 \times 10^{-3}$ is used.

![Graphs showing beam tip local y-axis displacements and process time for each loop with varying damping ratios](image)

(a) Beam Tip y-axis Displacements

(b) Simulation Time Per Loop

Figure 4.13: Simulation Results for Rotating Beam Problem, with Varying Damping Ratios

![Graphs showing beam tip local y-axis displacements and process time for each loop with varying x-axis rotation](image)

(a) Beam Tip y-axis Displacements

(b) Simulation Time Per Loop

Figure 4.14: Simulation Results for Rotating Beam Problem, Rotated Axis of Rotation

Above in Figure 4.14 are 11 plots, overlaid together, of the cases where the axis of rotation are rotated about the $x$-axis from 0 to $\pi$, with steps of $\frac{\pi}{10}$. As it can be observed, all the rotated results closely match the base result, with some process time caveats. Specifically, the cases where the axis of rotation are rotated by 0, $\frac{\pi}{2}$ and $\pi$ are represented by the much shorter process time on Figure 4.14 (b), while the rest of the cases end up with much longer process time after around 8 seconds simulation time, marked with numerous spikes. This is presumed to be due to the ODE solver having to deal with more states when the rotation is not about one of the axes - which it is when the axis of rotation is rotated by 0, $\frac{\pi}{2}$.
or π. It should be noted that while there are no requirements or goals in regards to the processing time of the simulation, it is still of interest as it directly impacts the efficiency of the research, especially for the more complex simulations.

Finally, Figure 4.15 shows an overlay of 21 plots, all with randomly-generated initial beam orientations. As with the rotated axis of rotation plots, the results all closely match each other, showing that the choice of shape functions and the implemented simulation system are robust to changes in orientations.

![Beam Tip y-axis Displacements](image1)

![Initial Location of the Beam](image2)

**Figure 4.15: Simulation Results for Rotating Beam Problem, Rotated Initial Starting Point of the Beam**

**Performance Difference Between Dense Matrix and Sparse Matrix Implementation**

To demonstrate the performance difference between a dense matrix system and a sparse matrix system, the above problem was simulated using 100 elements - which results in 600 total states and global matrices with 36000 components. The simulation was performed with no orientation or axis of rotation changes, with zero damping. The results are given in Figure 4.16. The majority of the time difference is from the linear equation solution. In the case of the dense matrix, the “linear solver time” refers to time taken to invert the L and U matrices, and the actual matrix multiplication process to find the answer is not included. A notable time difference can be observed from the LU factorization process though not as significant as the difference observed in the linear solving step. There are no notable time differences for processing functions in ODEFnc_MODULE - an expected behaviour as the indexing method used by the dense matrix generators are mostly mimicked by the sparse matrix generators. Given that the plate simulations involve hundreds of elements with more states than the beams, the time savings from using the sparse matrix implementation can be expected to be immense.
Chapter 4. Unconstrained Nonlinear Flexible Structural Dynamics

4.6 Sample Simulation: Clamped Plate Deformation Due to Vertical Loads

Examples of dynamics simulation of nonlinear plates are less common in the literature. However, there are examples of static plate displacements. The clamped plate deformation is a simple and classical example, where a rectangular plate clamped on all sides is subjected to an out-of-plane load. The displacement at the middle of the plate is measured. The case presented and referred to for this simulation is from Timoshenko and Woinowsky-Krieger’s classical work on plates [49]. Specifically, their work compared the static displacements between a linear plate and a nonlinear plate, with varying x and y dimension ratios. Here, those results are reproduced using dynamic equations of motion and letting
the simulation progress until a steady state is reached, at which point the displacement of the middle node is measured and compared against the classical work. While this will not verify the validity of the transient behaviour of the plate, the fact that the system can arrive at the correct steady state values would lend credence to the validity of the triangular plate’s shape function and the simulator, hence indirectly validating the transient behaviour as well.

4.6.1 Problem Description

This is a very simple problem: Given a thin plate that is clamped at all its edges as shown in Figure 4.17, gradually apply a distributed transverse load on the plate until a designated value $F$ is reached, then continue simulation until the states stabilize. The equation that applies the gradual load is a sine function as defined by $F_d$ in Figure 4.17, formulated such that the period of gradual force increase can be specified by the start time $t_s$ and the final time $t_f$. The structural parameters describing the plate are as follows: plate thickness $h = 1.285 \times 10^{-3}$ m, Young’s modulus $E = 4 \times 10^9$ N/m$^2$, Poisson’s ratio $\nu = 0.3$, and volume density $\rho = 1392$ kg/m$^3$. The values used for Young’s modulus and volume density are based on statistical data of Kapton, a material used for solar sail membranes.

$$F_d = \frac{F}{t_f}(t - t_s - \frac{T}{2\pi} \sin(2\pi \frac{t - t_s}{T})), t_s \leq t \leq t_f$$

$$F_d = 0, t < t_s, F_d = F, t > t_f$$

Figure 4.17: Description of the Clamped Plate Problem

The plate $y$-axis length $L_y$ remains fixed at 20 m. The plate $x$-axis lengths are varied between 20 m, 30 m and 40 m, to compare the displacement results against the steady-state results provided by Timoshenko and Woinowsky-Krieger. In particular, Figure 208, page 422 in [49] shows a graphical relationship between normalized forcing $\frac{q b^4}{Dh}$ and the normalized center deflection $\frac{w_{max}}{h}$, where $q$ is the distributed force, $b$ is the $y$-axis length, $D = \frac{E h^3}{12(1-\nu^2)}$ and $w_{max}$ is the maximum deflection of the center. The lengths are $a = \frac{L_x}{2}$ and $b = \frac{L_y}{2}$ respectively. By setting $q = 1 \times 10^{-5}$ N/m$^2$, the normalized forcing is set to 100 and the $y$-axis values on the graph can be read off with rough accuracy: 1.15 for $\frac{b}{a} = 1$, 1.46 for $\frac{b}{a} = \frac{2}{3}$, and 1.48 for $\frac{b}{a} = \frac{1}{2}$. In addition, the linear case for $\frac{b}{a} = 1$ can be read off as 2.0. These values are used as the basis for the expected steady state values of the center of the plate.
4.6.2 Simulation Setup

The structure is described by combining four TriStruct objects together in a manner similar to a solar sail. This is an elastic simulation - meaning none of the rigid-body dynamics are kept. In addition, all of the edge-nodes of the plate are removed - this can be performed by checking each node’s $x$ or $y$ coordinate values for an extrema and removing the node if its coordinate location contains such a value. The number of elements is varied by the number of edge elements per side. For example, the element division illustrated in Figure 4.17 is taken as an element division of 4. Nonlinear simulations are performed for element divisions of 4, 8 and 12. Ideally a model with element division 16 would be used too, but unfortunately this model ran into out-of-memory issues.

The simulation is performed by gradually applying a distributed transversal force over the entire plate, starting at 0 s and ending at 500 s. The simulation is composed of 200 time steps, each 5 seconds long, ending at 1000 s. The slow, gradual application of forces allows the system to remain reasonably steady throughout the simulation. A diagonal damping is used, and its damping coefficient is set to $1 \times 10^{-2}$. The plate simulation tends to be much harder to stabilize than the beam simulation, and this larger damping also contributes to reaching a steady state value faster.

The uniform forces are applied by letting $f_e = [0, 0, F_d]$ then using $f_{Te}$ from Equations (4.50). The value of $f_e$ is specified within the gentri_fTe function. The ODE solver calls a user-provided $f_{ext}$ function, which calls the gen_global_fTe function to construct a global $f_{Te}$ array with repeated calls to the gentri_fTe function.

Two simulations using $L = L_x = 20$ m are performed, one a linear result and another a nonlinear result - to observe the difference between the two as well as matching the results to ones provided by Timoshenko and Woinowsky-Krieger. The other two simulations with $L_x = 30$ m and $L_x = 40$ m are performed in succession.

4.6.3 Simulation Results

The simulation results for a plate sized $L_x = L_y = 20$ m are given in Figure 4.18. The maximum plate deformation at the centre seen for linear and nonlinear simulations of the element division 4 model in Figure 4.18(a) are 2.237 and 1.286 respectively, off by a notable amount from the deformation height given by Timoshenko and Woinowsky-Krieger, but the shape of the graph roughly resembles the given plot. Refining the results with element division 8 and 12 models, which produce values of 1.205 and 1.174 respectively, show that there is a reasonable convergence to the results provided by Timoshenko and Woinowsky-Krieger - specifically, there is a notable convergence to the deformation height of 1.15, the value seen in their results. A visual result of the plate’s displacement due to the uniform force for element division 12 model is given in Figure 4.18(b). One can say that the overall shape of the deformed plate is reasonably realistic.

The next set of simulations, shown in Figure 4.19 (a), are those of the case where $L_x$ is increased in length to 30 m - referring to the case $\frac{L_x}{L_y} = \frac{3}{2}$. The deformation height produced by element division 4 are not completely damped out until at the end of the simulation - at which point producing a value of 1.719. Element division 8 simulation are much more steady, and results in a normalized displacement value of 1.563. The same can be said for the element division 12 simulation, and its final value is at 1.52. Once again, there is a notable convergence towards the value - 1.46 - given by Timoshenko and Woinowsky-Krieger.
The simulation case with $L_x = 40$ m is given in Figure 4.19 (b). Simulation using element division 4 is unsteady once again, though it eventually settles down to a value of 1.931. Element division 8 simulations are also unsteady, but the resulting value at normalized forcing of 100 is 1.642 - closer to the value provided by Timoshenko and Woinowsky-Krieger for the $\frac{L}{a} = \frac{1}{2}$ case. Finally, element division 12 simulation produces a much more steady result, ending with a value of 1.596. While not very close to the desired converged value of 1.48, the convergence towards it can be observed.

It should be noted that for $L_x = 30$ m and $L_x = 40$ m cases, there is some asymmetric element generation occurring - specifically, the elements generated within the top and the bottom triangular plate sections are differently-shaped compared to the elements generated along the left and the right
triangular plate sections, as shown in Figure 4.20.

In light of this, consider a model arranged differently, in particular, that of Figure 4.21(a), where the triangular plate sections have been abandoned in favour of one rectangular section with its triangular element arranged in equal sizes. The number of elements used to create this model, dubbed element division 4-10 model, is only slightly above the number of elements used for the division 4 model (division 4 model uses 64 elements, whereas this uses 80), yet the simulation results, as seen on 4.21(b), are much better than that of the division 12 model, hence the simulation and its chosen set of shape functions have a capacity to perform very well, with the right kind of element arrangement.

(a) Element Division 4-10 Model, \( L_x = 40 \text{ m}, L_y = 20 \text{ m} \)

(b) Simulation Comparison of Two Models, m

Figure 4.21: Comparison of Two Clamped Plate Models, \( L_x = 40 \text{ m}, L_y = 20 \text{ m} \)
4.7 Sample Simulation: Cord-Mat Solar Sail with SRP Force Loading

With the verifications for both the beam and the plate elements, the actual sail structure can be simulated. The cord-mat solar sail design, described in Section 2.2, is modeled by using the cylindrical beams as sail booms and supporting cords and the triangular plate element as the sail membrane. A simple SRP force is applied to the sail’s surface in a gradual fashion until the system stabilizes. The general shape of the billowing sail and the maximum out-of-plane displacement are the desired information from the simulation. Unfortunately, similar dynamics or statics simulation for a model of this nature could not be found in the literature, with the exception being Greschik’s boom bending out-of-plane displacement predictions in [20]. Hence, other than the boom bending, the verification of the results rely on the individual element type verifications from the previous sections.

As mentioned earlier, the membranes are more commonly modeled as a linear elastic film, which allows for more detailed FEM models with less computing power required. The main reason for modeling the membranes as a plate is that the nonlinear plate model exhibits coupling behavior between the out-of-plane and in-plane deflections. Unfortunately, the cost of using a nonlinear model in place of a linear model is that the simulation requires much more computing resources, which limits the author to less detailed FEM models. Nonetheless, it is sufficient to demonstrate a proof of concept for the wrinkling model in the next chapter, which is based upon the nonlinear model presented here.

4.7.1 Problem Definition

The problem is described as follows: a cord-mat square solar sail is modeled in a fashion shown in Figure 4.22. The parameters $\Delta$, $\Gamma$ and $\Omega$ in the figure are the normalized values of the cord sag, the inward camber, and the film billow respectively. The cord sag and the inward camber are normalized against the length of the boom $L$, while the film billow is normalized against the width between two cords, $L_s$. Symbol $l$ is the hypotenuse length of the triangular element. The sail is theoretically indirectly attached to the booms via these cords, but as seen in Figure 4.22 (b) the membrane elements are pinned to the end of the cords, which are also pinned to the booms. Hence the sail is essentially pinned to the booms where the cords are pinned to the booms. A simple SRP force is gradually applied on the sail from the inertial $z$-axis direction using the method described in Section 3.5, and after the force application is completed the simulation continues until a steady state is reached. The sail is allowed to rotate about its body axis to gauge how the sail’s deformation affects its attitude.

Greschik’s works have been referred to in order to derive approximate structural parameters for the sail components [19–21]. Dimensional parameter values are as follows: $L = 70.7$ m, $\Delta = \Omega = 1 \times 10^{-2}$ and $\Gamma = 0.0608$. The structural parameters used here also closely resemble Greschik’s parameters from [20] and are given in Table 4.2. The solar sail loading of the above configuration with four support cords per quadrant is $\sigma = 18.8$ g/m² (which increases when the bus mass is taken into account).

4.7.2 Simulation Setup

Each sail quadrant, a TriStruct object, is defined by the number of beam elements forming a boom. This number is equal to the number of triangular membrane elements adjacent to it. In addition, the quadrants are also defined by the number of these edge elements between the cords supporting the sail
membrane. For example, the sail model shown in Figure 4.22 has four beam elements for each boom and two edge elements between the cords. For literary convenience, the models are referred to as division \( n\text{-}m \), where \( n \) is the number of beam elements for each boom and \( m \) is the number of edge elements between the cords. The sample sail model in Figure 4.22 is a division 4-2 sail model in this convention. The model being used for the simulation is a division 8-2 sail model. As noted above, the sail is allowed to rotate about its centre, hence the rotational rigid dynamics are kept, while the translational rigid dynamics are discarded.

The simulation is performed by gradually applying the SRP force generated by the inertial frame sun vector \( \hat{s} = [0 \ 0 \ -1]^T \) in a same manner as the clamped plate example given in the previous section, i.e. the distributed force is described by the following equation:

\[
\mathbf{F}_d = \frac{\mathbf{f}_s}{t_e} \left( t - t_s - \frac{t_s}{2\pi} \sin \left( 2\pi \frac{t - t_s}{t_e} \right) \right), \quad t_s \leq t \leq t_e
\]  

(4.140)

where \( \mathbf{f}_s \) is the SRP force calculated via equations in Section 3.5, \( t_s = 0 \) s and \( t_e = 5000 \) s are the start and the duration of the force application, and \( t \) is the current time in the simulation. An additional 1000 seconds is allocated to the simulation for stabilization purposes. Each simulation time step is 20 seconds long. Rayleigh damping is used, and the damping coefficients are set to \( \alpha = 1 \) and \( \beta = 1 \times 10^{-5} \). The combination of the damping coefficient and the slow force application allows a reasonably steady sail simulation. To facilitate generality in the code, the force is generated by calling \texttt{gen\_srp\_force} function within the previously-used \texttt{gen\_tri\_fTe} function. No other modifications are made, as the code propagates the modified \( \mathbf{f}_{T_e} \) to form the global generalized force vector.

<table>
<thead>
<tr>
<th>Beams</th>
<th>Triangular Plate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sail Boom</td>
</tr>
<tr>
<td>( E ) (N/m²)</td>
<td>2.87 \times 10^{10}</td>
</tr>
<tr>
<td>( A ) (m²)</td>
<td>3.2224 \times 10^{-5}</td>
</tr>
<tr>
<td>( I ) (m⁴)</td>
<td>1.6104 \times 10^{-7}</td>
</tr>
<tr>
<td>( \rho ) (kg/m³)</td>
<td>1440</td>
</tr>
</tbody>
</table>

Table 4.2: Structural Parameters for the Cord-Mat Square Solar Sail
4.7.3 Simulation Results

The simulated FEM model is shown in Figure 4.23. The top view given in Figure 4.23 (a) was included to clarify the division 8-2 element arrangement for the FEM model. The isometric view shows the general pattern of the sail’s out-of-plane displacement. The booms bend downward due to the solar radiation pressure produced by the sail, while the sail billows out as well, with the out-of-plane displacement of the sail deeper than the boom tip bending as expected.

![Figure 4.23: Cord-Mat Square Solar Sail FEM Model under SRP Force Loading](image)

The boom displacements given in Figure 4.24 (a) and (b) behave as expected. In particular, note the value of -1.271 m for the out-of-plane deflection. Greschik calculated the boom tip deflection to be -1.336 m for a sail with a 77 m tapered boom. Considering the shorter length and the fact that the boom used for this simulation is not tapered, the difference between the two values is within a reasonable range. No values were provided for the prediction of the in-plane deflection, but the compression observed in Figure 4.23 (b) is the expected behaviour, as the billowing sail would apply a compressive force upon the boom and cause the observed behaviour.

The symmetric uniformity of the displacements about the quadrants is also a desired trait, and the boom tip deflections seem to suggest this to be true. Unfortunately, closer inspection of the sail shape suggests otherwise. In particular, observe the sail’s side view shown in Figure 4.25 (a). One of the most visible asymmetry between the quadrants can be seen near the middle of the sail, where one quadrant of the membrane bends up, while the other quadrant bends down. This asymmetry then propagates out to the edges of the sail, resulting in two different out-of-plane deflections of the sail where symmetric deflections are expected. This becomes particularly glaring when the centres of the sail quadrant edges are plotted out, as shown in Figure 4.25 (b). Notice how some deflection plots are further displaced than the others. This is clearly not how a membrane would behave, but this would be a viable behaviour from a plate under loading - one quadrant is forced to bend downward by the compression induced by the restoring force from the adjacent plate, which is observed to be flat near the centre. The behaviour is only amplified by the presence of the support cords being modelled as a beam with compressive stiffness. This asymmetry highlights the need for a separate mechanism to differentiate a membrane model over a plate model in order to properly simulate a geometrically-nonlinear sail.
Despite the asymmetry observed from the above, the sail’s attitude profile only shows small variations, as can be seen in Figure 4.26 (a). It is interesting to note that a tiny oscillation is observed for the latter half of the simulation. One can correlate this to the changes in the principal stresses and the minor ‘bumps’ in the z-axis displacement plots at \( t = 2900 \) seconds provided in Figure 4.26 (b), (c) and (d). The magnitude of this oscillation is small enough to be regarded as a numerical error in the integration.

Figure 4.27 shows the simulation time taken for each time step, and as can be seen this is a very costly simulation, taking upwards of 10 minutes to simulate just a single time step. The majority of this simulation time is due to the LU decomposition process within the simulation integrator used to find the inverse of the mass and the incremental matrix, suggesting that perhaps an alternate method to finding these matrices may be desirable.
Figure 4.26: Cord-Mat Square Solar Sail Deflection Asymmetry

Figure 4.27: Simulation Processing Time
Chapter 5

Wrinkling and Slacking Dynamics of a Membrane

One of the key components of a solar sail is its reflective surface - typically a very thin light-weight membrane with a special reflective coating. As the shape of this reflective membrane directly affects the forces produced by the sail, understanding the structural behaviour of the sail membrane is crucial in predicting the solar sail’s performance and behaviour.

However, while a membrane behaves in an elastic manner when in tension, compressive forces do not simply compress the material along the direction of the force - instead it folds upon itself, creating wrinkles upon the material surface or simply collapsing into a slack state. One can easily picture this relationship by considering a piece of thin paper - a paper resists any motion when pulled outward from its edges, but it will crumple upon itself when pushed inward from its edges with the same magnitude of force. The former state is of tension and the paper behaves elastically (albeit a very stiff material), while the latter state is of compression and the paper collapses on itself instead.

In this research, modelling this discontinuous behaviour is performed by applying changes directly to the constitutive relationship. This chapter introduces and implements methods to modify the constitutive relationship in order to better reflect the nonlinear behaviour of a membrane under various forcing conditions.

5.1 Background Information

There are numerous papers available on the topic of wrinkling membranes. The subjects covered by these papers are two-fold: How a wrinkled membrane behaves, and how to determine when a membrane is taut/wrinkled/slack. The modelling of wrinkled membrane behaviour is roughly divided into two classes of approaches.

The first approach involves modifying the deformation gradient tensor, introduced by Roddeman et al. [44]. The idea is as follows: An elastically-behaving membrane will have negative stress when compressive forces are applied to it. A true membrane however will wrinkle under such compressive force, and would have zero stress associated in that direction. To model this behaviour, a fictive membrane is virtually stretched in the direction of the wrinkle until the magnitude of the stress is zero. This virtual stretching, its final length corresponding to the actual length of the material, is represented by modifying
Chapter 5. Wrinkling and Slacking Dynamics of a Membrane

the deformation gradient, and is controlled by a special parameter that describes the wrinkle magnitude. This idea has been used and extended by several authors [24–26, 32, 33, 39, 43] since its introduction. However, as Akita et al. [2] points out, incorporating this approach into a finite element model is not as intuitive as the idea itself, especially in the context of Chapter 4 and the equations derived therein.

The second approach involves modifying the constitutive relationship matrix directly. Introduced by Miller and Hedgepeth, this approach is mathematically simple. Assuming that the stress is uniaxial under wrinkling state, the strain transformation equation is modified to remove the contribution by the minor principal stress and these equations are substituted into the stress-strain equation to derive the constitutive relationship matrix. Adler and Mikulas [1] implemented this on several commercial finite element method codes, while Miyazaki [36], Akita et al. [2], and Lee and Youn [28] each offered their flavour of derivations to the modified constitutive relationship matrix as well as extensions to the approach. Miyazaki’s work in particular is notable, as it shows that the two approaches described above are in fact identical, despite the different physical interpretations and formulations used to describe each model. This, combined with the fact that the Miller-Hedgepeth approach to the membrane wrinkling is much more intuitive to implement on a finite element method model than Roddeman’s approach, is why the wrinkling behaviour in this research uses the second approach as its basis.

There are three common methods to determine the current state of the membrane: the strain criterion, the stress criterion, and the mixed stress-strain criterion. The strain criterion is as follows: If the minor principal strain is greater than zero, then the membrane is taut. If the minor principal strain is less than zero but the major principal strain is greater than zero, then the membrane is wrinkled. Else the membrane is slack. The stress criterion is equivalent to the strain criterion, except the principal strains are replaced with the principal stresses. The mixed stress-strain criterion is as follows: If the minor principal stress is greater than zero, then the membrane is taut. If the minor principal stress is less than zero but the major principal strain is greater than zero, then the membrane is wrinkled. Otherwise the membrane is slack. Kang and Im show that the stress criterion and the strain criterion tend to be less accurate than the mixed criterion [25].

5.2 Describing Wrinkling and Slacking Behaviour Via Modification of the Constitutive Relationship

A membrane in tension as shown in Figure 5.1 behaves elastically and can be modelled as a plate. Its principal stress-strain constitutive relationship can be described by

\[
\begin{bmatrix}
\tau_1 \\
\tau_2
\end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix}
1 & \nu \\
\nu & 1
\end{bmatrix} \begin{bmatrix}
\epsilon_1 \\
\epsilon_2
\end{bmatrix}
\]

\[
= \Gamma_t \begin{bmatrix}
\epsilon_1 \\
\epsilon_2
\end{bmatrix}, \quad \Gamma_t = \frac{E}{1-\nu^2} \begin{bmatrix}
1 & \nu \\
\nu & 1
\end{bmatrix}
\]

where \(\tau_1\) and \(\tau_2\) are the major and the minor principal stresses, \(\epsilon_1\) and \(\epsilon_2\) are the major and the minor principal strains, and \(\Gamma_t\) is the principal constitutive relationship matrix for an elastic membrane, with \(E\) as Young’s modulus and \(\nu\) as Poisson’s ratio. The relationship between principal strains and local
frame strains can be described as follows:

\[
\begin{bmatrix}
\epsilon_{xx} & \gamma_{xy}/2 \\
\gamma_{xy}/2 & \epsilon_{yy}
\end{bmatrix} = \begin{bmatrix}
c_\alpha & -s_\alpha \\
s_\alpha & c_\alpha
\end{bmatrix} \begin{bmatrix}
\epsilon_1 & \gamma_{12}/2 \\
\gamma_{12}/2 & \epsilon_2
\end{bmatrix} \begin{bmatrix}
c_\alpha & s_\alpha \\
s_\alpha & -c_\alpha
\end{bmatrix}
\]

(5.2)

where \(\alpha\) is the principal angle. Similarly, the relationship between principal stresses and local frame stresses can be described as:

\[
\begin{bmatrix}
\tau_{xx} & \tau_{xy} \\
\tau_{xy} & \tau_{yy}
\end{bmatrix} = \begin{bmatrix}
c_\alpha & -s_\alpha \\
s_\alpha & c_\alpha
\end{bmatrix} \begin{bmatrix}
\tau_1 & 0 \\
0 & \tau_2
\end{bmatrix} \begin{bmatrix}
c_\alpha & s_\alpha \\
s_\alpha & -c_\alpha
\end{bmatrix}
\]

(5.3)

Inverting Equation (5.2) and substituting the resulting expressions for the principal strains into Equation (5.1), then substituting the expressions for the principal stresses given by Equation (5.1) into Equation (5.3), expanding and simplifying all the results while minding that \(\gamma_{12} = 0\), the following constitutive relationship can be derived:

\[
\tau = E_t \epsilon
\]

\[
\begin{bmatrix}
\tau_{xx} \\
\tau_{yy} \\
\tau_{xy}
\end{bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix}
1 & \nu & 0 \\
\nu & 1 & 0 \\
0 & 0 & \frac{1-\nu}{2}
\end{bmatrix} \begin{bmatrix}
\epsilon_{xx} \\
\epsilon_{yy} \\
\gamma_{xy}
\end{bmatrix}
\]

(5.4)

which is the well-known plane stress constitutive relationship equation. A derivation process for this constitutive relationship is provided later on in this section.

A membrane is considered ‘wrinkled’ when it is being stretched in one direction while being compressed in another, as illustrated in Figure 5.2. Pure stretching occurs along the direction of the major principal stress, while pure compression occurs along the direction of the minor principal strain. As noted before, since a membrane under compression does not behave elastically but instead folds upon itself, the resulting minor principal stress is assumed to be zero, giving the following modified form of Equation (5.1):

\[
\begin{bmatrix}
\tau_1 \\
\tau_2
\end{bmatrix} = \Gamma_w \begin{bmatrix}
\epsilon_1 \\
\epsilon_2
\end{bmatrix}, \quad \Gamma_w = \begin{bmatrix}
E & 0 \\
0 & 0
\end{bmatrix}
\]

(5.5)
Once again inverting Equation (5.2) and substituting the resulting expressions for the principal strains into Equation (5.5), then substituting the resulting expression into Equation (5.3), expanding and simplifying all the results as before, the following constitutive relationship is derived:

\[
\tau = E_w \epsilon \\
\begin{bmatrix}
\tau_{xx} \\
\tau_{yy} \\
\tau_{xy}
\end{bmatrix} = \frac{E}{4} \begin{bmatrix}
2(1 + P) & 0 & Q \\
0 & 2(1 - P) & Q \\
Q & Q & 1
\end{bmatrix} \begin{bmatrix}
\epsilon_{xx} \\
\epsilon_{yy} \\
\gamma_{xy}
\end{bmatrix}, \\
P = \frac{\epsilon_{xx} - \epsilon_{yy}}{\epsilon_1 - \epsilon_2}, \quad Q = \frac{\gamma_{xy}}{\epsilon_1 - \epsilon_2}
\]

(5.6)

This is the modified constitutive relationship derived by Miller and Hedgepeth. A derivation process for this relationship is also provided below. Finally, a membrane is considered ‘slack’ when it is being compressed in both principal directions. In a pure slack state, stresses remain zero throughout - that is, the constitutive relationship matrix is simply \( E_s = 0 \). With this, all three membrane states and the corresponding stress-train constitutive relationship matrices are known.

Now, all three cases can be described in combination by using the following principal stress-strain constitutive relationship as the starting point:

\[
\begin{bmatrix}
\tau_1 \\
\tau_2
\end{bmatrix} = \frac{E}{1 - ab\nu^2} \begin{bmatrix}
b & ab\nu \\
ab\nu & a
\end{bmatrix} \begin{bmatrix}
\epsilon_1 \\
\epsilon_2
\end{bmatrix}
\]

(5.7)

where the constants \( a \) and \( b \) are modified based on the state of the material. The derivation of the general case constitutive relationship from the above is as follows: substituting Equation (5.7) into Equation (5.3) and expanding, the following equation can be derived:

\[
\begin{bmatrix}
\tau_{xx} & \tau_{xy} \\
\tau_{xy} & \tau_{yy}
\end{bmatrix} = \frac{E}{1 - ab\nu^2} \begin{bmatrix}
A & B \\
B & C
\end{bmatrix}
\]

(5.8)

where

\[
A = c_1^2 (b\epsilon_1 + ab\nu\epsilon_2) + s_0^2 (ab\nu\epsilon_1 + a\epsilon_2)
\]

\[
B = s_0 c_1 [(b - ab\nu)\epsilon_1 + (ab\nu - a)\epsilon_2]
\]

\[
C = s_0^2 (b\epsilon_1 + ab\nu\epsilon_2) + c_1^2 (ab\nu\epsilon_1 + a\epsilon_2)
\]

(5.9)
Using the inverse of Equation (5.2), the principal strains $\epsilon_1$ and $\epsilon_2$ can be described by

$$
\epsilon_1 = \alpha^2 \epsilon_{xx} + s^2_\alpha \epsilon_{yy} + s_\alpha c_\alpha \gamma_{xy}, \quad \epsilon_2 = s_\alpha^2 \epsilon_{xx} + c^2_\alpha \epsilon_{yy} - s_\alpha c_\alpha \gamma_{xy}
$$

and the principal shear strain $\gamma_{12}$ can be described by

$$
\gamma_{12} = s_2 \alpha (\epsilon_{yy} - \epsilon_{xx}) + c_2 \alpha \gamma_{xy}
$$

However, by the definition of the principal strain, this value is zero. This relationship, along with the following trigonometric identities

$$
s_2 \alpha = 2 s_\alpha c_\alpha, \quad c_2 \alpha = c^2_\alpha - s^2_\alpha = 1 - 2s^2_\alpha = 2c^2_\alpha - 1
$$

are the keys to simplifying Equation (5.8).

Now substitute Equation (5.10) into (5.9) and expand. Regroup the terms of A as follows:

$$
A = a(s^4_\alpha \epsilon_{xx} + s^2_\alpha c^2_\alpha \epsilon_{yy} - s^3_\alpha c_\alpha \gamma_{xy}) + b(c^4_\alpha \epsilon_{xx} + s^2_\alpha c^2_\alpha \epsilon_{yy} + s_\alpha c^3_\alpha \gamma_{xy})
$$

$$
+ abv(2s^2_\alpha c^2_\alpha \epsilon_{xx} + c^4_\alpha \epsilon_{yy} + s^4_\alpha \epsilon_{yy} - s_\alpha c^3_\alpha \gamma_{xy} + s^3_\alpha c_\alpha \gamma_{xy})
$$

The first term from above can be rearranged as follows:

$$
s^4_\alpha \epsilon_{xx} + s^2_\alpha c^2_\alpha \epsilon_{yy} - s^3_\alpha c_\alpha \gamma_{xy} = s^2_\alpha (1 - c^2_\alpha) \epsilon_{xx} + s^2_\alpha c^2_\alpha \epsilon_{yy} - \gamma_{xy}(s^3_\alpha c_\alpha - \frac{s_\alpha c_\alpha}{2}) - \gamma_{xy} \frac{s_\alpha c_\alpha}{2}
$$

$$
= s^2_\alpha \epsilon_{xx} - \gamma_{xy} \frac{s_\alpha c_\alpha}{2} + (\epsilon_{yy} - \epsilon_{xx}) s^2_\alpha c^2_\alpha - \gamma_{xy} s_\alpha c_\alpha(s^2_\alpha - \frac{1}{2})
$$

$$
= \frac{1}{2} \epsilon_{xx} - \frac{Q}{4} \gamma_{xy} + \frac{s_\alpha c_\alpha}{2} [s_2 \alpha (\epsilon_{yy} - \epsilon_{xx}) + c_2 \alpha \gamma_{xy}]
$$

where the trigonometric relationships from Equation (5.12) are used to derive the last line. Of the three terms in the last line, the last term is equal to $\gamma_{12}$, which is known to be zero. Hence, after some rearranging,

$$
s^4_\alpha \epsilon_{xx} + s^2_\alpha c^2_\alpha \epsilon_{yy} - s^3_\alpha c_\alpha \gamma_{xy} = \frac{1}{4} [2(1 - P) \epsilon_{xx} - Q \gamma_{xy}]
$$

In a similar fashion, the second term in Equation (5.13) can be simplified to

$$
c^4_\alpha \epsilon_{xx} + s^2_\alpha c^2_\alpha \epsilon_{yy} + s_\alpha c^3_\alpha \gamma_{xy} = \frac{1}{4} [2(1 + P) \epsilon_{xx} + Q \gamma_{xy}]
$$

The third term in Equation (5.13) is simplified as follows:

$$
2s^2_\alpha c^2_\alpha \epsilon_{xx} + c^4_\alpha \epsilon_{yy} + s^2_\alpha \epsilon_{yy} - s_\alpha c^2_\alpha \gamma_{xy} + s^3_\alpha c_\alpha \gamma_{xy}
$$

$$
= 2s^2_\alpha c^2_\alpha \epsilon_{xx} + c^2_\alpha (1 - s^2_\alpha) \epsilon_{yy} + s^2_\alpha (1 - c^2_\alpha) \epsilon_{yy} - s_\alpha c_\alpha (c^2_\alpha \gamma_{xy} - s^2_\alpha \gamma_{xy})
$$

$$
= 2s^2_\alpha c^2_\alpha \epsilon_{xx} + (c^2_\alpha + s^2_\alpha) \epsilon_{yy} - 2s^2_\alpha c^2_\alpha \epsilon_{yy} - s_\alpha c_\alpha (c^2_\alpha - s^2_\alpha) \gamma_{xy}
$$

$$
= 2s^2_\alpha c^2_\alpha \epsilon_{xx} - 2s^2_\alpha c^2_\alpha \epsilon_{yy} + \epsilon_{yy} - s_\alpha c_\alpha c_2 \alpha \gamma_{xy}
$$

$$
= 2s^2_\alpha c^2_\alpha (\epsilon_{xx} - \epsilon_{yy}) - s_\alpha c_\alpha c_2 \alpha \gamma_{xy} + \epsilon_{yy}
$$

$$
= - s_\alpha c_\alpha (s_2 \alpha (\epsilon_{yy} - \epsilon_{xx}) + c_2 \alpha \gamma_{xy}) + \epsilon_{yy}
$$
The last term is once again $\gamma_{12}$, hence is equal to zero, reducing the equation for the third term to

$$2s_a^2c_a^2\epsilon_{xx} + c_a^4\epsilon_{yy} + s_a^4\epsilon_{yy} - s_a^3c_a^3\gamma_{xy} = \epsilon_{yy}$$ (5.18)

Substituting Equations (5.15), (5.16) and (5.18) into (5.13) and rearranging,

$$A = \frac{1}{4} [2(a(1-P) + b(1+P))\epsilon_{xx} + 4ab\epsilon_{yy} + Q(b-a)\gamma_{xy}]$$ (5.19)

In a similar fashion, $C$ can also be simplified to

$$C = \frac{1}{4} [2(b(1-P) + a(1+P))\epsilon_{yy} + 4ab\epsilon_{xx} + Q(b-a)\gamma_{xy}]$$ (5.20)

The expression for $B$ is expanded and rearranged as follows:

$$B = b(s_a^3c_a^3\epsilon_{xx} + s_a^3c_a\epsilon_{yy} + s_a^2c_a^2\gamma_{xy})$$

$$- a(s_a^3c_a^3\epsilon_{xx} + s_a^3c_a\epsilon_{yy} - s_a^2c_a^2\gamma_{xy})$$

$$+ ab(s_a^3c_a\epsilon_{xx} - s_a^3c_a^3\epsilon_{xx} + s_a^3c_a^3\epsilon_{yy} - s_a^3c_a\epsilon_{yy} - 2s_a^2c_a^2\gamma_{xy})$$

(5.21)

The first term can be rearranged as follows:

$$s_a^3c_a^3\epsilon_{xx} + s_a^3c_a\epsilon_{yy} + s_a^2c_a^2\gamma_{xy}$$

$$= (s_a^3c_a - \frac{s_a^2c_a}{2})\epsilon_{xx} + \frac{s_a^3c_a}{2}\epsilon_{xx} + (s_a^3c_a - \frac{s_a^2c_a}{2})\epsilon_{yy} + \frac{s_a^3c_a}{2}\epsilon_{yy} + (s_a^2c_a^2 - \frac{1}{4})\gamma_{xy} + \frac{\gamma_{xy}}{4}$$

$$= \frac{Q}{4}\epsilon_{xx} + \frac{Q}{4}\epsilon_{yy} + \frac{\gamma_{xy}}{4} + s_a^3c_a(c_a^2 - \frac{1}{2})\epsilon_{xx} + s_a^3c_a(s_a^2 - \frac{1}{2})\epsilon_{yy} + \frac{1}{4}(s_a^2c_a^2 - 1)\gamma_{xy}$$

(5.22)

$$= \frac{1}{4}[Q\epsilon_{xx} + Q\epsilon_{yy} + \gamma_{xy} + s_{2a}c_{2a}\epsilon_{xx} - s_{2a}c_{2a}\epsilon_{yy} - c_{2a}\gamma_{xy}]$$

$$= \frac{1}{4}[Q\epsilon_{xx} + Q\epsilon_{yy} + \gamma_{xy} - c_{2a}(s_{2a}(\epsilon_{yy} - \epsilon_{xx}) + c_{2a}\gamma_{xy})]$$

The last term is once again $\gamma_{12}$, hence the above is simplified to

$$s_a^3c_a^3\epsilon_{xx} + s_a^3c_a\epsilon_{yy} + s_a^2c_a^2\gamma_{xy} = \frac{1}{4}[Q\epsilon_{xx} + Q\epsilon_{yy} + \gamma_{xy}]$$ (5.23)

The second term in Equation (5.21) is simplified in a similar manner, resulting in

$$s_a^3c_a\epsilon_{xx} + s_a^2c_a\epsilon_{yy} - s_a^2c_a\gamma_{xy} = \frac{1}{4}(Q\epsilon_{xx} + Q\epsilon_{yy} - \gamma_{xy})$$ (5.24)

The third term in Equation (5.21) can be rearranged as follows:

$$s_a^3c_a\epsilon_{xx} - s_a^3c_a^3\epsilon_{xx} + s_a^3c_a\epsilon_{yy} - s_a^3c_a\epsilon_{yy} - 2s_a^2c_a^2\gamma_{xy}$$

$$= s_a^3c_a(s_a^2 - \frac{s_a^2}{2})\epsilon_{xx} + s_a^3c_a(c_a^2 - \frac{s_a^2}{2})\epsilon_{yy} + 2(\frac{1}{4} - s_a^2c_a^2)\gamma_{xy} - \frac{\gamma_{xy}}{2}$$

$$= -\frac{1}{2}s_{2a}c_{2a}\epsilon_{xx} + \frac{1}{2}s_{2a}c_{2a}\epsilon_{yy} + \frac{1}{2}(1 - s_{2a}^2)\gamma_{xy} - \frac{\gamma_{xy}}{2}$$

(5.25)

$$= \frac{1}{2}[-s_{2a}c_{2a}\epsilon_{xx} + s_{2a}c_{2a}\epsilon_{yy} + c_{2a}\gamma_{xy} - \gamma_{xy}]$$

$$= \frac{1}{2}[s_{2a}(s_{2a}(\epsilon_{yy} - \epsilon_{xx}) + c_{2a}\gamma_{xy}) - \gamma_{xy}]$$
The first term is equal to $\gamma_{12}$, hence is set to zero, which means the above is simplified to

$$s_\alpha^3 c_\alpha \epsilon_{xx} - s_\alpha^3 c_\alpha \epsilon_{yy} - s_\alpha^3 c_\alpha \epsilon_{yy} - 2s_\alpha^2 c_\alpha^2 \gamma_{xy} = -\frac{\gamma_{xy}}{2} 
(5.26)$$

Substituting in Equations (5.23), (5.24) and (5.26) into (5.21) and rearranging,

$$B = \frac{1}{4} [Q(b - a)\epsilon_{xx} + \epsilon_{yy} + (b + a - 2ab\nu)\gamma_{xy}] 
(5.27)$$

Finally, substituting in Equations (5.19), (5.20) and (5.27) into (5.8) then rearranging the stresses into a vector form, we have:

$$\begin{bmatrix} \tau_{xx} \\ \tau_{yy} \\ \tau_{xy} \end{bmatrix} = \frac{E}{4(1 - ab\nu^2)} \begin{bmatrix} 2[a(1 - P) + b(1 + P)] & 4ab\nu & Q(b - a) \\ 4ab\nu & 2[b(1 - P) + ab(1 + P)] & Q(b - a) \\ Q(b - a) & Q(b - a) & b + a - 2ab\nu \end{bmatrix} \begin{bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{bmatrix} 
(5.28)$$

When $a = b = 1$, the above equation becomes equivalent to Equation (5.4), i.e., the plane stress-strain constitutive relationship. When $a = 0$ and $b = 1$, the above equation simplifies to Equation (5.6), i.e., the wrinkled stress-strain constitutive relationship. When $a = b = 0$, the above simply becomes $0$, i.e. the slack stress-strain constitutive relationship. Hence all three material states can be represented with the appropriate values of $a$ and $b$. Essentially, the degree of wrinkledness and slackness can be classified by the values of $a$ and $b$, where $a$ controls the degree of wrinkledness and $b$ controls the degree of slackness of the material.

### 5.3 Describing String Slacking Behaviour Via Modification of Constitutive Relationship

It is worthwhile to consider the behaviour of a string under compressive load in a similar manner as a wrinkling/slacking membrane. Note that a string being compressed along its length will simply collapse as opposed to providing elastic resistance, just as a membrane would. The only difference is that unlike a membrane, a string is one-dimensional and such compressive load immediately results in a slacked state. A wrinkled state has no physical meaning for a string. Given its one-dimensional nature and the conventions used in the previous section, the constitutive relationship can be modified in a following manner for the slacked state:

$$E_s = bE 
(5.29)$$

where $b$ is equivalent to the slacking parameter for the membrane. Logical dissemination of the modified constitutive relationship is that when the string is under compressive loading, its stiffness is modified to represent the near-zero resistance the string provides against the compression.
5.4 Membrane State Criterion and the Behaviour of Wrinkling Parameters in a Dynamic System

While the constitutive relationships for different membrane states are given, there is also a need for a method to determine what state each membrane element is in. For the statics problem, the general approach for the Miller-Hedgepeth constitutive relationship is to use a mixed stress-strain criterion, which is described as follows:

\[
E = \begin{cases} 
E_t & \text{if } f_1 \geq 0 \\
E_w & \text{if } f_2 \geq 0 \text{ and } f_1 < 0 \\
E_s & \text{if } f_2 < 0 
\end{cases}
\]  

(5.30)

where \( f_1 = \epsilon_2 + \nu \epsilon_1 \) and \( f_2 = \epsilon_1 \). The principal stresses and strains are calculated from an initially taut membrane, then the above criterion is used to modify the constitutive relationship, which is then used to solve the statics problem once again to derive a new set of stresses and strains. The process repeats itself until either the constitutive relationship does not change (i.e. convergence is reached) or if a maximum iteration count is exceeded. There are two other commonly-referred criteria, the strain criterion and the stress criterion, as noted by Kang and Im in [25]. As the name suggests, all three criteria are identical except that one uses strains and another uses stresses exclusively to determine the membrane’s state. These two criteria are noted by Kang and Im to be unstable, and are generally avoided in the literature.

The above criterion can easily be modified in terms of the parameters \( a \) and \( b \) as follows:

\[
\begin{cases} 
 a = 1, \ b = 1 & \text{if } f_1 \geq 0 \\
 a = 1, \ b = \delta & \text{if } f_2 \geq 0 \text{ and } f_1 < 0 \\
 a = \delta, \ b = \delta & \text{if } f_2 < 0 
\end{cases}
\]  

(5.31)

where \( \delta \) is a small number near zero, arbitrarily chosen to be \( 1 \times 10^{-12} \) for the purpose of the simulation. It would seem then that implementing the modified principal stress-strain relationship given in Equation (5.28) to the dynamics simulation is simply a matter of modifying the constitutive relationship at every time step. Unfortunately, it is not such a simple matter. Several modifications have been made to the above-given basic relationship, but the results of the simulations have been unsatisfactory.

5.5 Membrane State Criteria on Sail Deformation Due to Solar Radiation Pressure Loads

To demonstrate the issues surrounding the wrinkling model for the dynamics simulation, the solar sailing model simulated from Section 4.7 is used, with the constitutive relationship modified in manners to be outlined below. The same dimensional and structural parameters are used, but to expedite the simulation process a division 6-2 model is used. As some of the methods tested slows down the simulation to a point where it is infeasible to complete it, the simulation is only allowed to run until the time elapsed for a single time step exceeds ten times the average simulation time.
5.5.1 Standard Miller-Hedgepeth Mixed-Criterion Update

The first method tested is the most basic setup where the parameters \( a \) and \( b \) are updated at each time step using Equation (5.31). The constitutive relationship matrix is modified accordingly at each time step. The expectation was that the gradual loading on the membrane would allow the parameters to settle into steady state values. However upon simulation, it was discovered that instead of the parameters gradually settling down, they continue to oscillate. Figure 5.3 shows the final state of the sail before the simulation was terminated early due to exceeding the above-mentioned time limit. The green-shaded elements are taut, yellow-shaded elements are wrinkled, and the red-shaded elements are slack. The red borders indicate that the wrinkling state of the particular element is different from the previous time step. The blue lines represent the beams. One can observe the sail’s symmetric response to the loading as desired, but obviously as the simulation has halted this method is unusable.

Figure 5.3: Wrinkling via Standard Miller-Hedgepeth Mixed-Criterion Update on the Cord-Mat Sail

The growth in oscillation can be seen from the principal strains and stresses. The minor principal stress and the number of elements with different wrinkling parameters from the previous time-step are plotted in Figure 5.4 (a) and (b) respectively. While the element’s wrinkling states seem to slowly stabilize, it never reaches the actual stable state. Instead, the stress oscillation eventually drives the entire simulation procedure to a halt, as each time step takes progressively longer to simulate until it becomes infeasible to continue the simulation. The point in which it becomes infeasible to continue the simulation is very apparent, as the integration time steps taken by the solver becomes several magnitudes smaller than that of the previous simulation time steps. In this case, it is at \( t = 2160 \) s.

5.5.2 Parameter Modification Via Gradual Changes to the Parameter

One approach to stabilizing the simulation, considering the oscillatory response of the wrinkling parameters \( a \) and \( b \), is to implement methods to gradually vary them as opposed to applying discontinuous changes. One way to achieve this is by defining the parameters as polynomial functions of the minor principal stress and the major principal strain. Specifically, let parameter \( a \) be represented as a quintic
Chapter 5. Wrinkling and Slacking Dynamics of a Membrane

Figure 5.4: Corner Membrane Element Oscillation Plots for the Mixed-Criterion Update

polynomial of the following form:

\[ a = a_0 + a_3(f_1 + k)^3 + a_4(f_1 + k)^4 + a_5(f_1 + k)^5 \]

\[ a_0 = \delta, \quad a_3 = 10(1 - \delta) \frac{k^3}{k^3}, \quad a_4 = 15(\delta - 1) \frac{k^4}{k^4}, \quad a_5 = 6(1 - \delta) \frac{k^5}{k^5} \]

(5.32)

where \( k \) is a tuning parameter that determines at what value of \( f_1 \) the plot reaches the minimum value \( \delta \).

A plot of the parameter \( a \) for \( f_1 \) between \( k = 1 \times 10^{-6} \) and zero is given in Figure 5.5. As can be seen, the polynomial creates a plot that smoothly approaches both zero and \( \delta \) as \( f_1 \) approaches zero and \( -k \) respectively. The parameter \( a \) is fixed to 1 for \( f_1 > 0 \) and to \( \delta \) for \( f_1 < -k \). The result of this setup is that as increasingly compressive stresses are observed, \( a \) becomes progressively smaller until it reaches the minimum prescribed value of \( \delta \) when \( f_1 = -k \), then is fixed at that value. Hence a gradual change from the taut state to the wrinkled state is facilitated. In a similar manner, parameter \( b \) is also determined using a similar quintic polynomial, the main difference being that \( f_2 \) is used in place of \( f_1 \).

To perform the simulation, an arbitrarily small value of \( k \) is chosen (\( k \) is set to be equal among membrane parameters \( a \) and \( b \), as well as the beam parameter \( b \)) and the simulation is run. It was noticed immediately that the simulation slows down to a crawl for the majority of the small values of \( k \) in a similar fashion to the standard Miller-Hedgepeth criterion setup. Hence, the values of \( k \) are gradually increased until a stable simulation can be performed. For the sail simulation setup given in Section 4.7, \( k \) had to be increased to \( 3 \times 10^{-6} \) before the simulation could be run from the start to finish.

Shown in Figure 5.6 is the result of the stable simulation. As can be seen from Figure 5.6 (a) and (d), the wrinkled portion of the sail is drastically decreased, and the sail experiences the same asymmetry observed for the taut sail from Section 4.7. Figure 5.6 (b) shows that the edge displacement asymmetry has been reduced noticeably but it is still present. The oscillations in the wrinkling parameters, as well as the principal stresses, have been damped out as seen in Figure 5.6 (c) and (d). It would seem then that by further decreasing \( k \) and inducing greater changes to the wrinkling parameters, one could
potentially expect better representation of the wrinkling dynamics. Unfortunately as mentioned above, the value of $k$ used for the simulation was the smallest that could be used before the simulation failed to complete.

Several variations to $k$ have been made in an effort to allow the simulation to run - such as using a fraction or an average of the principal stresses and strains, varying parameter values between different wrinkling parameters and elements, and so forth. Unfortunately, all of the outlined efforts have failed to produce satisfactory results.

A similar approach pursued with the same goal of inducing gradual changes to the parameters $a$ and $b$ is that of discrete stepping function. Emulating the behaviour of a discrete derivative, a small sum is added or subtracted to the parameters $a$ and $b$ at each simulation time step. Specifically $a_{k+1}$, representing parameter $a$ at time step $k + 1$, is calculated by

$$a_{k+1} = a_k + k_a f_1 t$$

where $k_a$ is a tuning parameter and $t$ is the time step size. A saturator is placed after the equation to constrain the values of $a$ between $δ$ and 1.

The result of this approach is omitted as the conclusions remain the same - the tuning parameter has to be increased to a certain level before a significant changes to the wrinkling parameters are observed, yet such increases lead to the integration steps in the solver decreasing to a point where it is infeasible to continue the simulation. Several attempts have been made to dynamically change $k_a$, but all attempts have been met with unsatisfactory results.

### 5.5.3 Parameter Modification Via Weighted Criterion

Observing from Figure 5.4 (a) that the principal stress oscillation switches between a small value and a relatively larger value, a potential solution to the behaviour was concocted - that of using a value other than a zero to perform the comparison in the mixed-criterion check. That is, redefine the parameter
mixed-criterion as

\[
\begin{align*}
    a &= 1, \quad b = 1 \quad \text{if } f_1 \geq k_a \\
    a &= 1, \quad b = \delta \quad \text{if } f_2 \geq k_b \text{ and } f_1 < -k_a \\
    a &= \delta, \quad b = \delta \quad \text{if } f_2 < -k_b
\end{align*}
\]

(5.34)

where \( k_a \) and \( k_b \) are the tuning parameters. Setting constant values for \( k_a \) and \( k_b \) is impractical, as the stresses and strains will only increase in magnitude as the simulation progresses due to the gradual force application performed.

**Average Weighted Criterion**

As a first attempt, \( k_a \) and \( k_b \) are set to be an absolute value average for the current and the previous instances of \( f_1 \) and \( f_2 \) respectively. This choice was based on the observation that with the stress values
switching between small and large numbers, only one of the two values would satisfy the inequalities in the criterion and allow the state to be settled down.

The first attempt resulted in failure, but the idea seems to have merit, as seen in Figure 5.7. Specifically, the wrinkling pattern does stabilize briefly between 1920 seconds and 2140 seconds, although that stability is quickly lost and eventually drives the simulation to a halt at 2520 seconds, slightly further from where the standard Miller-Hedgepeth criterion simulation has stopped.

Figure 5.7: Wrinkling via Average Weighted Criterion on the Cord-Mat Sail

**Accumulative Weighted Criterion**

The given cases suggest that the changes in the constitutive relationship at later time steps are the main cause of the simulation failure, and prevention of such changes may be beneficial for the simulation as a whole, despite such a setup negatively reflecting upon the simulation. Hence, the following alternate method is proposed: simply accumulate $f_1$ and $f_2$, then use a fraction $\delta$ of that value as the bound. This causes the constants $k_a$ and $k_b$ to eventually grow to a point where the wrinkling parameters will no longer change.

The results of the simulation with the accumulative weighted criterion method with $\delta = 0.1$ is given in Figure 5.8. The simulation completes as desired. Observing the wrinkling parameter plot reveals that the parameter convergence occurred fairly early on in the simulation - at 1060 seconds. The wrinkling pattern observed also seems to be fairly limited, suggesting that the wrinkling pattern convergence may be happening too early in the simulation to properly capture its dynamics. Apart from it, the sail behaves reasonably, with no strange jumps in the principal stresses plot observed. The final recorded sail edge out-of-plane displacement is -2.076 m.

**Time-Limited Accumulative Weighted Criterion**

To allow the model to better capture the wrinkling dynamics, a modification to the above-stated method is made by allowing the simulation to run with the standard Miller-Hedgepeth criterion enabled until
a pre-designated time step $t_1$ before $k_a$ and $k_b$ are increased. In addition, to prevent issues seen in the regular accumulated weighted criterion, the constitutive relationship is forced into a fixed state after a second pre-designated time step $t_2$ passes. The idea is that between $t_1$ and $t_2$, the increase to $k_a$ and $k_b$ would allow the sail to settle into a temporary stable state, which can be used to perform the remaining portions of the simulation without risking changes induced by the criterion.

Figure 5.8: Wrinkling via Accumulative Weighted Criterion on the Cord-Mat Sail

Figure 5.9: Wrinkling via Time-Limited Accumulative Weighted Criterion on the Cord-Mat Sail

Figure 5.9 shows the sample implementation of the above with $t_1 = 1500$ s, $t_2 = 2500$ s, and $\delta = 0.1$ as before. With this setup the sail is rendered stable while possessing wrinkling patterns. The convergence time for the wrinkling parameters is 1620 s, and the final sail edge out-of-plane deflection is measured to be -1.932 m. The wrinkling pattern and displacement seen in Figure 5.9 is symmetric and wide-spread
along the initially-slack region, and the principal stresses as observed in Figure 5.9 are stable after the initial transient phase. The simulation performs within a reasonable time, a desirable trait.

Selective Accumulative Weighted Criterion

In order to provide even more diversity in the wrinkling pattern, another modification is introduced to the accumulative weighted criterion - that of limiting the accumulation to elements with changes to the wrinkling state. This is implemented easily by checking if the currently-determined wrinkling parameter is different from the wrinkling parameter from the previous time step, and increasing $k_a$ or $k_b$ only if the corresponding parameters were changed. This was implemented as an add-on to the time-limited accumulative weighted criterion, hence the accumulation still does not occur until $t_1$ has passed, and it (as well as the changes to the constitutive relationship) ceases once $t_2$ passes.

Figure 5.10 shows the sail simulation with the selective accumulative weighted criterion enabled, with $t_1 = 1500$ s, $t_2 = 2500$ s and $\delta = 0.1$ as before. Close inspection of Figure 5.10 (a) shows that the wrinkled area has expanded further. Figure 5.10 (b) further emphasizes this when compared against Figure 5.9 (b), where the number of visible negative minor principal stress plots have been decreased. The parameters are observed to be oscillating even at 2500 seconds, but most of the parameters have already settled as early as 1600 seconds, with only a few odd parameter oscillations occurring from then to $t_2$. The sail edge deflection at the end of the simulation is recorded as -2.077 m, reasonably close to the previous two results.

![Figure 5.10: Wrinkling via Selective Accumulative Weighted Criterion on the Cord-Mat Sail](image)

5.5.4 Simulation Results with Division 8-2 Cord-Mat Solar Sail

To provide comparison results to Section 4.7, the 8-2 division sail is simulated with the selective accumulative weighted criterion modifying the constitutive relationship at each time step. The dimensional and structural properties are equivalent, as is the gradual SRP forcing applied onto the sail. Criterion-specific parameters are set as follows: $t_1 = 1500$ s, $t_2 = 2500$ s and $\delta = 0.2$. The two time constraints
are set with the same values as the 6-2 division sail simulations, while $\delta$ has been increased to induce faster convergence on the sail wrinkling pattern.

Figures 5.11 and 5.12 show various plots associated with the simulation. One can see from Figure 5.11 (a) and (b) that the division 8-2 model possesses a similar wrinkling pattern as the division 6-2 model seen in Figure 5.10. The in-plane and out-of-plane displacements of the boom and the sail edge are shown in Figure 5.11 (c) through (f). Comparing these figures against Figures 4.23, 4.24 and 4.25, several differences can be spotted. For one, the sail is now fully symmetric about the quadrants, and the wrinkling caused both the boom and the sail to bend and sag further - an increase in displacement between 10 to 20 percent can be observed from the plots.

Figure 5.12 provides a number of plots with information to convey. Figures 5.12 (a) and (b) show the major and the minor principal stresses the sail membranes undergo. It is quickly apparent that the changes in the parameter induces significant spikes in these values, which disappear as soon as the changes to the parameter are suppressed by the selective accumulative weighted criterion at 2500 seconds. Comparing these plots to the taut case given in Figure 4.26 (c) and (d) however, one can notice that the wrinkling pattern has led to significant reduction in compressive stresses. While ideally there would be no compressive stresses, the reduction observed is still a significant improvement over the original taut model. There seems to be a small growth in the Euler angle as seen in Figure 5.12 (c), though it is a magnitude smaller than the oscillation observed in Figure 4.26 (a). The wrinkling parameters are still oscillating by 2500 seconds, but one can see that majority of the parameters have settled by 2020 seconds.

Figure 5.13 shows the simulation processing time. It is interesting to note that when compared against Figure 4.27, this simulation performs faster despite the added complexity, albeit few spikes in the simulation are still observed. Considering that the simulator takes smaller integration time steps as the simulation stability is compromised, one can speculate that the wrinkled simulation is more stable than the taut simulation despite the changing constitutive relationship.

### 5.6 Sample Simulation: Corner Loads on a Square Membrane

One of the classical static examples is that of a corner-loaded square membrane, extensively studied by Wong and Pellegrino [57–59] among others. Wong and Pellegrino’s results are referenced here as they offer experimental, analytical and numerical results to the problem. The problem is simple: apply tensioning loads to the four corners of a square membrane, vary the loading on one set of corners and observe the nonlinear out-of-plane displacement that occurs. The goal of this simulation is to capture the wrinkling pattern and the out-of-plane displacement due to it. A membrane represented solely by linear elements would not be able to capture such interaction as there are no terms that associate in-plane displacement to the out-of-plane displacement, but the nonlinear model implemented here does allow this with its inertial forces and geometric nonlinearities in the stiffness matrix. This interaction, combined with the modification to the constitutive relationship for wrinkled membranes, would emulate the behaviour observed in the experiment performed by Wong and Pellegrino.
Figure 5.11: Division 8-2 Wrinkling Solar Sail
Figure 5.12: Division 8-2 Wrinkling Solar Sail Principal Stresses, Euler Angles and Wrinkling Parameters

Figure 5.13: Simulation Processing Time
5.6.1 Problem Description

Given a thin square membrane suspended in air, in-plane loads along the $x$ and the $y$ axes are applied at the corners of the square membrane, as shown in Figure 5.14. The corner load $f_1$ is fixed at 5 N, while $f_2$ is set as 5 N, 10 N, 15 N and 20 N to produce four different results of interest. It is assumed that the gravitational load is applied as a distributed load upon the membrane, replacing the typical assumed imperfections needed to induce out-of-plane displacements. The dimensional and structural parameters for the membrane are as given in Wong and Pellegrino - that is, $L = 0.5$ m, $h = 2.5 \times 10^{-5}$ mm, $E = 3.5 \times 10^9$ N/m$^2$, $\nu = 0.31$, and $\rho = 1500$ kg/m$^3$.

![Figure 5.14: Description of the Membrane Corner Loading Problem](image)

5.6.2 Results from Known Literature

Prior to analyzing the simulation results, the results produced by Wong and Pellegrino’s experiments are presented here. Figure 5.15 shows the $x$-axis cross-sectional wrinkling plots for equal corner loads, one for 5 N and another for 20 N. Apart from the progressively reduced slope size as the cross section approaches the centre, it is also worthwhile to note that all of the plots have a very large upward slope at the edges. In addition, the plot shows that varying the magnitude of the corner load has little effect on the shape of the plot. It should be noted that due to this lack of variance in two results, the $f_1 = f_2 = 20$ N case is not simulated.

Figure 5.16 shows the cross-sectional plot at latter points from Figure 5.15 as well as one near the centre, with varying corner loads. This plot is particularly notable for the significant out-of-plane deflection experienced by the $f_2 = 20$ N case, along with a less significant, but noticeable slope shown on the third plot for the $f_2 = 15$ N case. The edges are upturned here as well, with the exception of the last plot - the corner loads prevent out-of-plane displacements for the two ends of the plot.

Wong and Pellegrino also provided photographs of the membrane being experimented on, as shown in Figure 5.17. The photographs show the wrinkling at the corners propagating outward as the forces are increased, until the two wrinkles meet to form a large diagonal slope as seen in Figure 5.17 (c) and (d). It is also worth noting that as $f_2$ is increased, the propagation of the wrinkles about the other corners decrease, giving way to the dominant diagonal wrinkle.
Figure 5.15: Cross sectional wrinkle profiles produced by Wong and Pellegrino for a 0.025 mm thick, square Kapton membrane, for \( f_1 = f_2 = 5 \) N (dashed) and \( f_1 = f_2 = 20 \) N (solid), at following distances from the origin: (a) \( y = 318 \) mm, (b) \( y = 300 \) mm, (c) \( y = 283 \) mm, (d) \( y = 248 \) mm, (e) \( y = 212 \) mm, (f) \( y = 176 \) mm [58]
Figure 5.16: Cross sectional wrinkle profiles produced by Wong and Pellegrino for a 0.025 mm thick, square Kapton membrane for $f_1 = 5$ N and varying values of $f_2$ (denoted as T1 on the plot), at following distances from the origin: (a) $y = 248$ mm, (b) $y = 176$ mm, (c) $y = 0$ mm [58]
Chapter 5. Wrinkling and Slacking Dynamics of a Membrane

Figure 5.17: Wrinkle pattern experiment performed by Wong and Pellegrino for a 0.025 mm thick, square Kapton membrane for: (a) $f_2 = 5$ N, (b) $f_2 = 10$ N, (c) $f_2 = 15$ N, (d) $f_2 = 20$ N [58]

5.6.3 Simulation Setup

The same structural setup used to simulate the clamped plate is used here - four TriStruct objects forming an element square plate, as shown in Figure 5.14. Using the convention established in Section 4.6, a division 10 model is used. To provide better stability to the simulation, z-axis deflections of the four corners are restricted, while in-plane deflections of the body centre are restricted as well. The rigid body dynamics are removed from the simulation - only elastic deflections are modelled.

The simulation is performed by gradually applying 1% of the actual corner load being applied initially to 0.2 seconds, in order to induce a stable wrinkling pattern and provide small tension against the gravitational load. Then the out-of-plane gravitational load is applied from 0.2 seconds to 0.7 seconds, calculated with the standard gravitation acceleration value of $g = 9.81$ m/s$^2$. Once the gravitation load is applied, the remaining corner loads are applied between 0.7 seconds to 1.2 seconds, with $f_1$ fixed to 5 N and $f_2$ varied from 5 N to 20 N. The simulation is then allowed to settle, completing at 1.5 seconds. Each simulation time step is $2 \times 10^{-3}$ seconds long. Rayleigh damping with damping constant values of $\alpha = 1$ and $\beta = 1 \times 10^{-5}$ is used - the same damping used for the sail simulation from the previous section. The selective accumulative weighted criterion is used, with $t_1 = 0.1$ s, $t_2 = 0.2$ s and $\delta = 0.1$. Values of $t_1$ and $t_2$ are set to match the time length in which the initial small corner loads are being applied, while $\delta$ is simply the same value used for the division 6-2 sail simulation.
5.6.4 Simulation Results

Figures 5.18 through 5.19 show the results of the simulation. Comparing the isometric model plot given in Figure 5.18 against the photographs in Figure 5.17, some similar traits can be detected - such as that of the deepening valley along the diagonal and receding wrinkles about the other diagonal. Apart from these, it is interesting to note how the magnitude of the edge displacement seem to ‘shift’ towards the diagonal with greater loading. It is also worth noting the lack of actual out-of-plane displacements for most of the elements marked as wrinkled.

Figure 5.18: Isometric View of the Membranes under Varying Corner Loads

Figure 5.19, showing the cross sectional deflection, emphasizes this issue. While the major slope seen for the $f_2 = 15$ N and $f_2 = 20$ N cases in Figure 5.16 (c) is reproduced here, none of the smaller wrinkles, such as those seen in 5.16 (a) are reproduced. Apart from the limitation due to number of elements representing the cross sections, it is also plausible to consider this to be a limitation of the method used to represent the wrinkling, as the reduction in stiffness via constitutive relationship essentially implies
that compressive forces will for the most part simply compress the material better. While it was hoped that the presence of the geometric nonlinearities in the stiffness matrix would provide some amount of out-of-plane deflections due to the in-plane forces, the results seem to indicate that its effects are limited. However, it should be noted that while the magnitude of the large trough experienced by the $f_2 = 20$ N case seems to be lacking compared to the results from Figure 5.16 (c), closer inspection reveals that the difference between the maximum and the minimum values of the $z$-axis displacement is just under 4 mm for both figures, lending credence to the validity of the simulation.

The wrinkling pattern seen in Figure 5.20 is also quite promising. The pattern’s propagation as the corner load is increased can be seen quite clearly from the figure. The locally-limited wrinkle waves seen in Figure 5.20 (a) propagate outward as the load is increased (and the corners with relatively smaller load sees receding wrinkling area), resulting in the wrinkling pattern seen in Figure 5.20 (b). Figure 5.20 (c) shows the creation of the trough due to the joining of the two wrinkling patterns propagating outward, marked by the solid yellow line, which expands as the corner loading increases as seen in Figure 5.20 (d).
Figure 5.20 also shows a quiver plot of each element’s major principal stress vector. One can observe a symmetric stress magnitude and direction about the four quadrants for the equal loading case, which changes as more loading is applied to the $y$-axis corners, causing the major principal stress vectors to gradually align with the loaded corners. Along the central regions of the membrane, one can observe that the major principal stress has mostly aligned with the slope present there. As the direction of wrinkling is defined to be towards the major principal stress vector, the observed values and directions are quite satisfactory.

The minor principal stress vectors, as shown in Figure 5.21 are satisfactory as well. The direction of the minor principal stress vectors are directly tied to that of the major principal stress vector. Hence, there is no other special meaning associated with the pattern observed. The magnitude however, is significant in that one can observe the magnitude decreasing as the elements approach the wrinkled region, with the wrinkled elements possessing no minor principal stress. This is as desired, given that
wrinkled elements do not possess any significant compressive stiffness and the wrinkled state can only occur if the minor principal stress is less than zero at any point in the simulation. As soon as the element enters the wrinkled state the minor principal stress is rendered zero by the virtue of the modified constitutive relationship. Many of the taut elements near the wrinkled elements seem to be experiencing a threshold situation where the minor principal stress is just high enough to not trigger the criterion that would mark them to be wrinkled.
Chapter 6

Control Allocation of Tip Vanes with Two Degrees of Freedom

The sail’s attitude control is based on a quad of vanes attached to the tips of the supporting booms. These vanes reflect the incoming photons just as a sail would, and use the resulting thrust and its position with respect to the sail centre to produce attitude controlling torques. Since the vane’s orientation with respect to the sun changes the torque it produces, the vane typically possesses angular Degrees Of Freedom (DOF) to allow reorientation.

The main problem is that with multiple DOF for each vane and a total of four vanes attached to the sail, there can be an infinite number of orientations for a particular desired torque. Hence, when the controller requests a certain torque vector, there is no straightforward way to find the orientation required to produce the torque.

The goal of this chapter is to define this as an under-constrained control allocation problem to derive a unique solution that reorients the vanes to produce the desired torque value. The resulting algorithm has to be such that it causes minimal changes to the vane orientation, and also allows for reasonably fast calculation of the angles for any given desired torque, in case on-orbit control allocation must be performed. This chapter will cover these issues along with the control allocation problem itself. For simplicity, the problem is limited to a vane possessing an optically ideal, flat surface.

6.1 Solar Radiation Pressure Torques Due to One-DOF Vanes

Prior to considering the two-DOF vane problem, the one-DOF vane problem should be considered first. Several researchers have already considered the problem of torque generation via vanes with singular degree of freedom. McInnes considered a simple case of two tip vanes attached along the sail x-axis, where its DOF is the angular rotation about the x-axis as shown in Figure 6.1 [35]. Assuming an ideal optical vane surface, the control torque provided by the system given cone angle $\alpha$ is

\[
\begin{align*}
G_x &= 0 \\
G_y &= 2 P A d \cos^2(\alpha)(\cos^3 \phi_1 - \cos^3 \phi_2) \\
G_z &= 2 P A d \cos^2(\alpha)(\cos^2 \phi_1 \sin \phi_1 + \cos^2 \phi_2 \sin \phi_2)
\end{align*}
\]
where $d$ is the length of the boom and $\phi_i$ is the rotation angle of vane $i$. Note that the tip vanes are unable to produce any $x$-axis torques. This highlights the main problem with control systems with one-DOF vanes as their actuators - the limitation on torques that can be produced by them.

Not only would this vane setup be unable to produce $G_z$ at all, but as shown in Figure 6.2 the desired $G_y$ will dictate the maximum $G_z$ as well, and vice versa (though to a lesser degree). The only torque magnitude that can be provided consistently regardless of the other component is 0. In addition, the torque being produced is dictated by the sun angle $\alpha$, which decreases the forces produced as it increases until the sun vector is aligned with the $x$-axis, preventing any torques from being produced by the vanes in such a case.

From the equation presented above, it is unclear as to how desired torque components correspond to vane angles. Wie proposed a partial solution to this problem by first assuming small vane angles, then letting $\Delta_c = \phi_2 - \phi_1$ and $\Theta_c = \phi_2 + \phi_1$ to write the two vane angles as

$$\phi_2 = \frac{\Delta_c + \Theta_c}{2}, \quad \phi_1 = \frac{\Delta_c - \Theta_c}{2} \tag{6.2}$$

where

$$\Delta_c = \frac{G_z}{2PA_d \cos^2 \alpha}, \quad \Theta_c = \frac{G_y}{G_z} \tag{6.3}$$

While it does provide a unique solution, there are singularity problems with the above control logic:
namely, \( G_z = 0 \) and \( \alpha \neq \pi/2 \). The second singularity problem can be solved simply by noting that as \( \alpha \) approaches \( \pi/2 \), both \( G_y \) and \( G_z \) go to zero, which can be achieved by simply setting \( \phi_1 = \phi_2 = 0 \). The first singularity is purportedly solved by using additional flaps attached to the sail, though the details of the solution are not outlined.

### 6.2 Solar Radiation Pressure Torques Due to Two-DOF Vanes

Two-DOF vanes are less common in the literature, presumably due to the mechanical complexity of double-jointed actuation and the control allocation complexity of many degrees of freedom posed by multiple two-DOF vanes. While the mechanical aspects of two-DOF vanes are not covered by this research, a solution for the control allocation aspect is presented. To this end, consider the vane represented by \( \hat{n}_1 \) in Figure 6.3. As this vane is considered an optically-ideal flat reflective surface, the force produced by this vane can be written using Equation (3.3) from a previous chapter. A normalized form of the equation is given below:

\[
\mathbf{f}_{\text{simple}} = -\left(\hat{\mathbf{s}}^T \hat{\mathbf{n}}\right)^2 \hat{\mathbf{n}}
\]

where both the sun vector \( \hat{\mathbf{s}} \) and the surface normal vector \( \hat{\mathbf{n}} \) are represented in the same frame.

In a three-dimensional space, the sun vector is represented by not only the cone angle \( \alpha \), but also by the clock angle \( \beta \) as shown in Figure 6.4. The sun vector in the sail body frame can then be written as

\[
\hat{\mathbf{s}}_b = \begin{bmatrix} S_\alpha C_\beta \\ S_\alpha S_\beta \\ -C_\alpha \end{bmatrix}
\]

where \( S \) and \( C \) represent sine and cosine functions respectively. Vane 1 is rotated along the body \( x \)-axis by \( \theta \), followed by a rotation along the rotated \( y \)-axis by \( \phi \). The rotation matrix describing this set of rotations can be represented by the rotation matrix

\[
\mathbf{C}_{eb} = \begin{bmatrix} C_\phi & S_\phi S_\theta & -S_\phi C_\theta \\ 0 & C_\theta & S_\theta \\ S_\phi & -C_\phi S_\theta & C_\phi C_\theta \end{bmatrix}
\]
Now define the vane frame to be aligned to the body frame when $\phi = \theta = 0$. By this definition $\hat{n} = [0 \ 0 \ 1]$. Now also assume that the vane rotation does not shift the centre of pressure for the SRP force, and the sail is perfectly flat along its $x$-$y$ plane, with its centre of pressure at the body frame origin, also on the $x$-$y$ plane. Hence it can be represented as a stationary point force attached to the end of a rigid boom. Given these assumptions, the SRP force generated by the vane in the vane frame can be written in terms of the sun angles $\alpha$ and $\beta$, and the vane angles $\phi$ and $\theta$, as

$$ f_{v1} = -\left(\hat{s}_{ex}^T \hat{n}_v\right)^2 \hat{n}_v = -\left((C_v \hat{s}_b)^T \hat{n}_v\right)^2 \hat{n}_v $$

To keep the equations simple in writing, the dot product $\left(\hat{s}_{ex}^T \hat{n}_v\right)$ is kept in its unexpanded form, i.e. the left side of the relationship $\hat{s}_{ex}^T \hat{n}_v = S_\phi S_\alpha C_\beta - C_\phi S_\beta S_\alpha S_\beta - C_\phi C_\theta C_\alpha$ is used. The force in the body frame can then be written as

$$ f_{b1} = C_v^T f_v $$

$$ = \begin{bmatrix} -S_\phi \\ C_\phi S_\theta \\ -C_\phi C_\theta \end{bmatrix} \left(\hat{s}_{ex}^T \hat{n}_v\right)^2 $$

(6.8)

As the sail is assumed to be flat and rigid, the torque produced by this vane is simply a cross product between a vector from the origin to the vane’s centre of pressure and the force calculated above, i.e.

$$ G_{b1} = r_1^x f_{b1} $$

$$ = \begin{bmatrix} 0 \\ C_\phi C_\theta \\ C_\phi S_\theta \end{bmatrix} \left(\hat{s}_{ex}^T \hat{n}_v\right)^2 $$

(6.9)

where the distance $r_1$ between the body origin and the vane centre of pressure has been omitted by
normalization. By following similar processes, torques produced by other vanes can also be calculated, resulting in the following vector equations:

\[ \mathbf{G}_{b3} = \begin{bmatrix} 0 \\ -C_\phi C_\theta \\ -C_\phi S_\theta \end{bmatrix}(\mathbf{s}_{v3}^T \mathbf{n}_v)^2 \quad (6.10) \]

\[ \mathbf{G}_{b2} = \begin{bmatrix} -C_\phi C_\theta \\ 0 \\ C_\phi S_\theta \end{bmatrix}(\mathbf{s}_{v2}^T \mathbf{n}_v)^2 \quad (6.11) \]

\[ \mathbf{G}_{b4} = \begin{bmatrix} C_\phi C_\theta \\ 0 \\ -C_\phi S_\theta \end{bmatrix}(\mathbf{s}_{v4}^T \mathbf{n}_v)^2 \quad (6.12) \]

where, for reiteration,

\[ \mathbf{s}_{v2}^T \mathbf{n}_v = S_\phi S_\alpha C_\beta - C_\phi S_\theta S_\alpha S_\beta - C_\phi C_\theta C_\alpha \quad (6.13) \]

and

\[ \mathbf{s}_{v4}^T \mathbf{n}_v = C_\phi S_\theta S_\alpha C_\beta - S_\phi S_\alpha S_\beta - C_\phi C_\theta C_\alpha \quad (6.14) \]

The total control torque offered by the set of tip vanes is the combination of the four above-presented torques, and is a complex nonlinear equation of eight controllable variables. An analytical solution for the problem seems unlikely, and an attempt to solve this equation using a nonlinear solver was met with failure. An alternate method is needed to generate vane angles for a given control torque.

### 6.3 The Control Allocation Problem

The goal of the control allocation problem posed is to derive a particular set of angles that correspond to the desired torque. To this end, an optimization problem is formed as given below:

\[
\begin{align*}
\text{minimize} & \quad \| \Phi - \Phi_i \|^2 \\
\text{subject to} & \quad \mathbf{G}_{b1} + \mathbf{G}_{b2} + \mathbf{G}_{b3} + \mathbf{G}_{b4} = \mathbf{G}_d \\
& \quad -\pi < \Phi_n < \pi
\end{align*}
\]

where \( \Phi = \text{col}\{\Phi_n\} = [\phi_1 \ \theta_1 \ \phi_2 \ \theta_2 \ \phi_3 \ \theta_3 \ \phi_4 \ \theta_4]^T \), \( \Phi_i \) is the previous iteration of \( \Phi \), and \( \mathbf{G}_d \) is the desired body frame torque vector. The cost function states that minimal change from previous vane angle position is desired, while the constraint demands that the vanes produce the desired torque. As mentioned in the previous section, this problem is challenging to solve due to its non-affine, non-convex nonlinear structure.

Instead of trying to solve the above problem, a modified problem is posed:

\[
\begin{align*}
\text{minimize} & \quad \| \mathbf{G}_b - \mathbf{G}_i \|^2 \\
\text{subject to} & \quad \mathbf{G}_{b1} + \mathbf{G}_{b2} + \mathbf{G}_{b3} + \mathbf{G}_{b4} = \mathbf{G}_d, \\
& \quad (\text{Satisfy Attainable Moment Set})
\end{align*}
\]

(Satisfy Attainable Moment Set)
where $\mathbf{G}_b$ and $\mathbf{G}_i$ are the current and the previous vane torques in vector forms. While this is not the same problem as the initial problem posed above, it can be said to be similar and it possesses an advantage over the first problem, which is taken advantage of in order to derive the proposed solution.

Due to the conditions imposed upon the sail, each vane only produces torque in two Cartesian directions, as seen on Equations (6.9), (6.10), (6.11) and (6.12). Hence $\mathbf{G}_b$ is written as

$$\mathbf{G}_b = \begin{bmatrix} G_{b1y} & G_{b1z} & G_{b2x} & G_{b2z} & G_{b3y} & G_{b3z} & G_{b4x} & G_{b4z} \end{bmatrix}^T$$

where $G_{b1y}$ is the y component of $\mathbf{G}_{b1}$, and so forth. Letting these components be the controllable variables, the equality constraint is now expanded as

$$\begin{bmatrix} G_{b2x} + G_{b4x} \\ G_{b1y} + G_{b3y} \\ G_{b1z} + G_{b2z} + G_{b3z} + G_{b4z} \end{bmatrix} = \mathbf{G}_d$$

which is a linear function of $\mathbf{G}_b$. The issue with this newly-defined problem is determining the Attainable Moment Set (AMS), or essentially defining the achievable set of torques that can be provided by each vane, and constraining the components of $\mathbf{G}_b$ within this set.

### 6.3.1 Attainable Moment Set of Vanes and its Estimation

Visualization of the AMS can be generated by iterating through set of values for the vane angles $\phi$ and $\theta$ and calculating the vane torque using the equations from the previous section, then finding the set of points that form the convex hull. The result of this procedure using Equation (6.9), while assuming the vane to be reflective only on one side, is given in Figure 6.5 (a). While this is the AMS, it is unclear how it can be represented in an equation form so that the inequality constraint for the optimization problem defined in the previous section can be solved for. Visual inspection suggests that an ellipse would roughly fit such a shape. Fitzgibbon et al. presents a direct least square fitting method for ellipses, which would generate an equation of the ellipse that best fits a given set of points [15]. Specifically, let the general
ellipse equation for the AMS of vane 1 be written as

\[ aG_{yi}^2 + bG_{yi}G_{zi} + cG_{zi}^2 + dG_{yi} + eG_{zi} + f = D_i c = 0 \]  \hspace{1cm} (6.19)

where \( a \) through \( f \) are the ellipse constants and

\[ D_i = \begin{bmatrix} G_{yi}^2 & G_{yi}G_{zi} & G_{zi}^2 & G_{yi} & G_{zi} & 1 \end{bmatrix}, \quad c = \begin{bmatrix} a & b & c & d & e & f \end{bmatrix}^T \]  \hspace{1cm} (6.20)

Given a set of torque component pairs \( G_{yi} \) and \( G_{zi} \) that compose the convex hull of the AMS, the general ellipse coefficients that best fit the AMS in a least squares sense can be calculated by solving the eigenvalue problem of the following form:

\[ Sc = \lambda Cc \]  \hspace{1cm} (6.21)

where

\[ S = D^T D, \quad D = \text{col} \{ D_i \}, \quad C = \begin{bmatrix} 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]  \hspace{1cm} (6.22)

The resulting general ellipse is plotted in Figure 6.5 (b). While it is a reasonable approximation to the AMS, it violates the boundaries, which means the ellipse contains areas where the vane is unable to produce the specified torque. Hence the ellipse in its current form is unsuitable as an estimation to the AMS.

To correct this discrepancy, a constrained minimization is performed on the ellipse coefficients derived by the least squares formulation. Specifically, the goal is to find the maximum ellipse area that fits within the constraint posed by the convex hull set that describes the AMS. The area for a general ellipse can be calculated by the following equation:

\[ A = \frac{2\pi h}{4ac - b^2} \]  \hspace{1cm} (6.23)

where

\[ h = dG_{y0} + eG_{z0} + f \]  \hspace{1cm} (6.24)

and \( G_{y0} \) and \( G_{z0} \) are the \( x \) and \( y \) components of the ellipse centroid respectively, which is calculated as follows

\[ \begin{bmatrix} G_{y0} \\ G_{z0} \end{bmatrix} = - \begin{bmatrix} 2a & b \\ b & 2c \end{bmatrix}^{-1} \begin{bmatrix} d \\ e \end{bmatrix} \]  \hspace{1cm} (6.25)

The bound on the ellipse - the maximum torque set that can be produced by the vane - is represented by the following linear inequality constraint:

\[ Dc > 0 \]  \hspace{1cm} (6.26)

Hence by maximizing Equation (6.23) with respect to \( c \) while satisfying the constraint Equation (6.26), a generalized ellipse that is within the AMS boundary is acquired. While the area function is nonlinear,
the initial value of $c$ calculated by solving the eigenvalue problem presented above is sufficiently close to the desired solution that typical numerical algorithms are easily able to provide a solution. The generalized ellipse formed by one such solution is plotted in Figure 6.6 (a), and it can be seen that this new ellipse fits tightly within the actual AMS as desired. The area ratio between the actual AMS and the ellipse estimation vary between 0.95 to 1 as the sun angles are changed.

Now, the above-presented method may or may not be viable for on-orbit calculation as the sun angles vary with the sail orientation, but here it is assumed that it is not viable. Hence, a parameterization of the ellipse with respect to the sun angle is made using the Fourier series. Specifically, the goal is to represent the ellipse constants $c_i$ in terms of a bivariate Fourier series as follows:

$$
c_i(\alpha, \beta) = \chi_1 + \chi_2 S_\alpha + \chi_3 C_\alpha + \chi_4 S_\beta + \chi_5 C_\beta + \chi_6 S_\alpha S_\beta + \chi_7 S_\alpha C_\beta + \ldots = \gamma(\alpha, \beta) \chi_i \tag{6.27}
$$

where $c_i$ is one of the ellipse constants, $\gamma(\alpha, \beta)$ is the row matrix of sinusoids and $\chi_i$ is the column matrix of coefficients. The maximum order of sinusoids is chosen arbitrarily, hence the problem devolves into finding the set of coefficients for the Fourier series that best fit the set of ellipse constants for all sun angles.

This parameterization is performed as follows: First, $c$ is calculated for a set of sun angles $\alpha$ and $\beta$ evenly spaced between $-\pi$ and $\pi$. In parallel, $\gamma(\alpha, \beta)$ is calculated for all of the $\alpha$ and $\beta$ pairs. Then, each component of $c$ and $\gamma(\alpha, \beta)$ are stacked column-wise and the following equation is solved for the Fourier series coefficients $\chi_i$:

$$
\Gamma \chi_i = c_i
$$

$$
\Gamma = \text{col} \{ \gamma(\alpha, \beta) \}
$$

$$
c_i = \text{col} \{ c_i(\alpha, \beta) \} \tag{6.28}
$$

The above equation is solved for all six ellipse constants. Once the above steps are taken, generating the estimated AMS ellipse is a simple matter of calculating the ellipse constants using the Fourier series and
the current sun angles. Figure 6.6 (b) shows one such reconstructed ellipse using fifth order bivariate
Fourier series parameterization, overlaid on top of the original ellipse to demonstrate how well the
Fourier series approximates the estimated AMS. Note that the Fourier series terms with zero or near-
zero coefficients are removed to reduce storage and increase calculation speed, and after this removal
procedure, only 6 to 9 terms remain significant. The actual Fourier series terms and coefficients are
given in Table 6.1.

<table>
<thead>
<tr>
<th>Sinusoid</th>
<th>Coefficient</th>
<th>Sinusoid</th>
<th>Coefficient</th>
<th>Sinusoid</th>
<th>Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4.6252 × 10^{-1}</td>
<td>1</td>
<td>-6.4156 × 10^{-1}</td>
<td>1</td>
<td>5.1679 × 10^{-1}</td>
</tr>
<tr>
<td>C_{a}</td>
<td>-8.6810 × 10^{-2}</td>
<td>C_{b}</td>
<td>-2.1984 × 10^{-1}</td>
<td>C_{d}</td>
<td>7.6645 × 10^{-2}</td>
</tr>
<tr>
<td>C_{a}</td>
<td>-2.3740 × 10^{-2}</td>
<td>C_{a}</td>
<td>-4.3972 × 10^{-3}</td>
<td>C_{b}</td>
<td>3.1945 × 10^{-2}</td>
</tr>
<tr>
<td>C_{a}</td>
<td>1.7733 × 10^{-4}</td>
<td>C_{d}</td>
<td>-7.4852 × 10^{-4}</td>
<td>C_{d}</td>
<td>-2.8168 × 10^{-3}</td>
</tr>
<tr>
<td>C_{d}</td>
<td>-1.4713 × 10^{-3}</td>
<td>C_{a}</td>
<td>2.9834 × 10^{-3}</td>
<td>C_{d}</td>
<td>8.1003 × 10^{-4}</td>
</tr>
<tr>
<td>C_{d}</td>
<td>2.9405 × 10^{-3}</td>
<td>C_{a}</td>
<td>2.4681 × 10^{-3}</td>
<td>C_{d}</td>
<td>-3.1478 × 10^{-2}</td>
</tr>
<tr>
<td>C_{d}</td>
<td>2.1478 × 10^{-3}</td>
<td>C_{d}</td>
<td>-3.6528 × 10^{-4}</td>
<td>C_{d}</td>
<td>-1.9212 × 10^{-3}</td>
</tr>
<tr>
<td>C_{d}</td>
<td>9.8842 × 10^{-4}</td>
<td>C_{d}</td>
<td>-3.2178 × 10^{-4}</td>
<td>C_{d}</td>
<td>-1.9994 × 10^{-3}</td>
</tr>
<tr>
<td>C_{d}</td>
<td>-7.3975 × 10^{-4}</td>
<td>C_{d}</td>
<td>-3.9749 × 10^{-4}</td>
<td>C_{d}</td>
<td>-6.1104 × 10^{-4}</td>
</tr>
</tbody>
</table>

Table 6.1: Fourier Series Sinusoid Terms and Corresponding Coefficients for the Estimated AMS Ellipse

The given Fourier series is specific to vane 1 only, and normally a different Fourier series would be
calculated for each vane. However after some comparison, it was found that the estimated AMS for other
vanes can be derived directly from the Fourier series for vane 1. Specifically, the relationships between
each vane’s ellipse constant parameterizations are as follows:

\[
\begin{align*}
\mathbf{c}_1(\alpha, \beta) &= \begin{bmatrix} a(\alpha, \beta) \\ b(\alpha, \beta) \\ c(\alpha, \beta) \\ d(\alpha, \beta) \\ e(\alpha, \beta) \\ f(\alpha, \beta) \end{bmatrix}, & \mathbf{c}_2(\alpha, \beta) &= \begin{bmatrix} a(\alpha, \beta + \pi/2) \\ b(\alpha, \beta + \pi/2) \\ c(\alpha, \beta + \pi/2) \\ d(\alpha + \pi/2, \beta + \pi/2) \\ e(\alpha, \beta - \pi/2) \\ f(\alpha, \beta + \pi/2) \end{bmatrix}, \\
\mathbf{c}_3(\alpha, \beta) &= \begin{bmatrix} a(\alpha, \beta) \\ b(\alpha, \beta) \\ c(\alpha, \beta) \\ -d(\alpha, \beta) \\ -e(\alpha, \beta) \\ f(\alpha, \beta) \end{bmatrix}, & \mathbf{c}_4(\alpha, \beta) &= \begin{bmatrix} a(\alpha, \beta + \pi/2) \\ b(\alpha, \beta + \pi/2) \\ c(\alpha, \beta + \pi/2) \\ -d(\alpha + \pi/2, \beta + \pi/2) \\ -e(\alpha, \beta - \pi/2) \\ f(\alpha, \beta + \pi/2) \end{bmatrix} \\
\end{align*}
\]
where the functions $a$ through $f$ are the Fourier series parameterization of the ellipse constants. With the above, the estimated AMS for all vanes can be conveniently generated in a general ellipse equation form regardless of sun angles.

### 6.3.2 Formal Definition of the Control Allocation Problem

Now that an estimate to the actual AMS is available in an equation form, the optimization problem from Equation (6.16) is rewritten as follows:

$$\min_{\mathcal{G}_s} \| \mathcal{G}_b - \mathcal{G}_i \|^2$$

subject to

$$\begin{bmatrix} G_{b2x} + G_{b4x} \\ G_{b1y} + G_{b3y} \\ G_{b1z} + G_{b2z} + G_{b3z} + G_{b4z} \end{bmatrix} = \mathcal{G}_d,$$

$$a_i G_{b1y}^{2} + b_i G_{b4y} G_{b1z} + c_i G_{b2z}^{2} + d_i G_{b4y} + e_i G_{b1z} + f_i < 0, i = 1, 3,$$

$$a_i G_{b1z}^{2} + b_i G_{b4z} G_{b1z} + c_i G_{b2z}^{2} + d_i G_{b4z} + e_i G_{b1z} + f_i < 0, i = 2, 4.$$  

(6.30)

Before, the constraint required that the torque values satisfy the AMS. Now, the constraint requires the torque values to remain inside the general ellipse that defines the estimated AMS. The problem is now that of a quadratic cost function, three linear equality constraints and four nonlinear inequality constraints, but more importantly every equations involved in the problem is convex. These types of optimization problems, classified as a convex optimization problem, guarantee arrival at a global optimum for any algorithm that can arrive at a local optimum, and in general are easier to solve when using known numerical optimization algorithms [6]. For testing purpose, Matlab’s fmincon function using either active-set line search or interior point methods is easily able to arrive at a solution.

There is an additional part to this problem: namely, that of determining the feasibility of the desired solution $\mathcal{G}_d$. Durham lists several methods to ascertain feasibility if each control variable was independently constrained [14]. This is not the case here unfortunately. For convenience, it is assumed that if the desired solution is not achievable, a scaled solution of the desired torque while preserving directionality is desired. Letting this scalar scaling factor be $\lambda$, this problem is posed as follows:

$$\min_{\lambda, G_{zx}, G_{zy}} \| 1 - \lambda \|^2$$

subject to

$$G_{zx} + G_{zy} = \lambda G_{dz},$$

$$a_k (\lambda G_k)^2 + b_k \lambda G_k G_{zk} + c_k G_{zk}^2 + d_k \lambda G_k + e_k G_{zk} + f_k < 0, \quad k = x, y,$$

(6.31)

where $G_{zx} = G_{b2z} + G_{b4z}$ and $G_{zy} = G_{b1z} + G_{b3z}$, $a_k$ through $f_k$ are the ellipse constants that define the estimated AMS boundary created by the Minkowski (geometric) sum of the estimated AMS ellipses for the pair of vanes on the same axis, and the subscript $k$ refers to the Cartesian torque component generated by the vanes. The scaling factor $\lambda$ drives the cost function to zero when it is equal to 1 - this is the case where the desired torque is in fact feasible, hence no changes are made to it when multiplied by $\lambda$. As the goal is to constrain the desired torque to within the estimated AMS, any $\lambda > 1$ solutions would violate the ellipse constraint if $\lambda = 1$ solution violates the constraint, hence all cases where the desired torque is not feasible would drive the scaling factor below 1, as desired.
on the same axis in a Minkowski (geometric) sum - for example, given created, which is then estimated as a generalized ellipse, whose constants are parameterized as Fourier series. The sinusoidal terms and the coefficients for this Fourier series are given in Table 6.2.

<table>
<thead>
<tr>
<th>Sinusoid</th>
<th>Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C^2$</td>
<td>$-8.2014 \times 10^{-2}$</td>
</tr>
<tr>
<td>$C^2$</td>
<td>$-2.2972 \times 10^{-1}$</td>
</tr>
<tr>
<td>$C^2$</td>
<td>$-2.4043 \times 10^{-3}$</td>
</tr>
<tr>
<td>$C^4$</td>
<td>$1.2695 \times 10^{-3}$</td>
</tr>
<tr>
<td>$C^4$</td>
<td>$-8.8293 \times 10^{-4}$</td>
</tr>
<tr>
<td>$C^4$</td>
<td>$1.8803 \times 10^{-3}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$3.1941 \times 10^{-2}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$1.4170 \times 10^{-3}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$-1.0340 \times 10^{-2}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$-2.5323 \times 10^{-4}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$3.8920 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sinusoid</th>
<th>Coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1$</td>
<td>$-3.7380 \times 10^{-8}$</td>
</tr>
<tr>
<td>$1$</td>
<td>$7.8201 \times 10^{-10}$</td>
</tr>
<tr>
<td>$1$</td>
<td>$-3.4393 \times 10^{-1}$</td>
</tr>
<tr>
<td>$C^2$</td>
<td>$-1.5279 \times 10^{-2}$</td>
</tr>
<tr>
<td>$C^2$</td>
<td>$1.5275 \times 10^{-2}$</td>
</tr>
<tr>
<td>$C^2$</td>
<td>$3.1485 \times 10^{-4}$</td>
</tr>
<tr>
<td>$C^2$</td>
<td>$3.0923 \times 10^{-4}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$-1.5668 \times 10^{-2}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$-3.3428 \times 10^{-4}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$9.058 \times 10^{-4}$</td>
</tr>
<tr>
<td>$C^2 \alpha C^2 \beta$</td>
<td>$2.7415 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

Table 6.2: Fourier Series Sinusoid Terms and Corresponding Coefficients for the Combined AMS Ellipse

It should be noted that the ellipse constants used for the feasibility optimization problem are generated separately from the ellipse constants used for the actual control allocation problem. These ellipse constants are generated by combining the points that form the estimated AMS ellipse for the two vanes on the same axis in a Minkowski (geometric) sum - for example, given $n$ points for vane 1 and $m$ points for vane 3, the Minkowski sum of these sets of points result in $nm$ points, where each new point is a sum of a point from vane 1 set and another point from vane 3 set. These generated points are then put through the same procedure used to generate the estimated AMS for individual vanes - a convex hull is created, which is then estimated as a generalized ellipse, whose constants are parameterized as Fourier series. The sinusoidal terms and the coefficients for this Fourier series are given in Table 6.2.

The above-given numbers are valid only for the combined AMS for the x-axis vanes, but once again there is a convenient relationship that links the combined AMS for the x-axis vanes to the combined AMS for the y-axis vanes, given below:

$$
\mathbf{c}_y(\alpha, \beta) = \begin{bmatrix} a_y(\alpha, \beta) \\ b_y(\alpha, \beta) \\ c_y(\alpha, \beta) \\ d_y(\alpha, \beta) \\ e_y(\alpha, \beta) \\ f_y(\alpha, \beta) \end{bmatrix}, \quad \mathbf{c}_x(\alpha, \beta) = \begin{bmatrix} a_x(\alpha, \beta + \pi/2) \\ b_x(\alpha, \beta + \pi/2) \\ c_x(\alpha, \beta + \pi/2) \\ d_x(\alpha + \pi, \beta + \pi/2) \\ e_x(\alpha, \beta - \pi/2) \\ f_x(\alpha, \beta + \pi/2) \end{bmatrix} \quad (6.32)
$$

Figure 6.7 shows the individual vane AMS, the combined AMS, as well as a solution set of vane torques generated by the above-given process for $\mathbf{G}_d = [0.5, 0.3, 1.3]$. The scaling factor is 1, i.e. the desired torque is fully achievable, and summing up the vane torques given in the figure does indeed result in the desired torque.
6.3.3 Analytical Solution to the Single Vane Problem

The solution to the above-introduced control allocation problem is a set of individual, achievable torque components for each vane. The original problem however calls for angular parameters that control the orientation of these vanes, hence the vane angles $\Phi$ must still be derived from the vane torques $G_b$.

Fortunately, there is an analytical solution to deriving vane angles from desired torque components when considering just one 2-DOF vane, by making use of Weierstrass substitution to formulate the SRP torque Equations (6.9), (6.10), (6.11) and (6.12) into root-solving problems.

Weierstrass substitution, also known as universal trigonometric substitution, was originally developed to find integrals of trigonometric functions. Specifically, let

$$t = \tan \frac{x}{2} \quad (6.33)$$

Then the following substitution can be made:

$$\sin x = \frac{2t}{1 + t^2} \quad (6.34)$$

$$\cos x = \frac{1 - t^2}{1 + t^2}$$

These substitutions are valid for $-\pi < x < \pi$, which is the assumed operating range for the vanes. These substitutions are made to the SRP torque equations to rearrange them into a univariate polynomial with respect to $t$, which is then used to solve for the vane angle.

For example, take $G_{b1}$ from Equation (6.9). Vane angle $\theta$ can be found easily by dividing $G_{bz}$ by $G_{by}$ (subscript 1 has been omitted for clarify):

$$\frac{G_{bz}}{G_{by}} = \frac{C_0 S_0 (\hat{s}^T v \hat{n}_v)^2}{C_0 C_0 (\hat{s}^T v \hat{n}_v)^2} = \tan \theta \quad (6.35)$$
Now, the only unknown is the vane angle $\phi$, for which the Weierstrass substitution is used. Begin by expanding $(\mathbf{s}_n^T \mathbf{n}_v)^2$:

$$(\mathbf{s}_n^T \mathbf{n}_v)^2 = (S_\phi S_\alpha C_\beta - C_\phi S_\alpha S_\beta - C_\alpha C_\beta C_\alpha)^2$$

Expanding and rearranging the above equation, we get:

$$a = S_\alpha^2 C_\beta^2$$

$$b = -2(S_\phi S_\alpha^2 S_\beta^2 + C_\phi S_\alpha C_\beta)$$

$$c = S_\phi^2 S_\alpha^2 S_\beta^2 + 2S_\phi C_\phi S_\alpha C_\beta + C_\alpha^2 C_\beta$$

(6.36)

Where constants $a$, $b$, and $c$ were defined for conciseness. The above expression can be substituted into either the expression for $G_{by}$ or $G_{bz}$. For demonstration purposes, $G_{by}$ is used here. Assuming $t = \tan \frac{\phi}{2}$ and making the appropriate substitutions to $S_\phi$ and $C_\phi$,

$$G_{by} = C_\phi C_\beta (S_n^T \mathbf{n}_v)^2$$

Expanding and rearranging the above equation,

$$(G_{by} + cC_\beta) t^6 + (-2bC_\phi) t^5 + [3G_{by} - (3c - 4a)C_\phi] t^4$$

$$+ (4bC_\phi) t^3 + [3G_{by} - (4a - 3c)C_\phi] t^2 + (-2bC_\phi) t + (G_{by} - cC_\phi) = 0$$

(6.38)

As the above is a univariate algebraic equation, its roots can be solved for using common root solving techniques. The vane angle $\phi$ can then be derived from $t$ by using the definition of Weierstrass substitution, i.e. $t = \tan \frac{\phi}{2}$. The polynomial is of sixth order, hence up to six solutions exist for $t$. Neglecting any imaginary solutions, the remaining roots and the resulting vane angles are then substituted back into Equation (6.9) to produce the vane torques. The vane angles that produce the correct vane torques and are closest to the previous vane angles are chosen as the true solution.

The above steps are repeated in a similar fashion for all vanes. For vane 2, Equation (6.11) is used to derive the equations for the vane angles. The equation for deriving $\theta_2$ can be written as

$$\frac{-G_{bz}}{G_{bx}} = \tan \theta_2$$

(6.39)

while the algebraic polynomial for $\phi_2$ is written as

$$(G_{bz} - cC_{\theta_2}) t^6 + (2bC_{\theta_2}) t^5 + [3G_{bz} + (3c - 4a)C_{\theta_2}] t^4$$

$$+ (-4bC_{\theta_2}) t^3 + [3G_{bz} + (4a - 3c)C_{\theta_2}] t^2 + (2bC_{\theta_2}) t + (G_{bz} + cC_{\theta_2}) = 0$$

(6.40)

Where

$$a = S_\phi^2 S_\alpha^2 S_\beta^2$$

$$b = 2(C_{\theta_2} S_\alpha C_\beta - S_{\theta_2} S_\alpha S_\beta C_\beta)$$

$$c = S_{\theta_2}^2 S_\alpha^2 C_\beta^2 - 2S_{\theta_2} C_{\theta_2} S_\alpha C_\beta + C_{\theta_2}^2 C_\alpha^2$$

(6.41)
The roots of the polynomial can be solved for and \( \phi_2 \) derived from the roots via Weierstrass substitution as before.

For vane 3, the same algorithm used to calculate the angles for vane 1 can be used, by noting that the right-hand side of Equation (6.10) is simply the sign-inverted right-hand side of Equation (6.9). Vane angle \( \theta_3 \) is calculated the same, while vane angle \( \phi_3 \) is calculated by running through the same process for \( \phi_1 \) while using \(-G_{63y}\) in place of \(G_{61y}\) to reflect the difference in sign. A similar process applies for vane 4, except it uses the algorithm for vane 2 instead of vane 1.

6.4 Code Implementation

The original simulation code was implemented in Matlab, taking advantage of its readily-available optimization solvers and root-finding algorithms. Convex hull sets are generated by using \texttt{conv} function on a set of pre-calculated torque values for a single-side reflective vane (reflective only on the positive z side in the vane frame). These sets are then used to determine the initial ellipse constants by using the \texttt{eig} function to solve for the eigenvector \( \mathbf{c} \). The ellipse constants determined this way are then used as the initial values for the minimization problem described in the previous section to constrain the ellipse to within the AMS, using \texttt{fmincon}.

The above-described process is then repeated for an evenly-spaced set of sun angles, the sun angles and the corresponding ellipse constants saved to a file. This is a very lengthy brute force process, which could in theory benefit from the faster computing speed Fortran would offer. However, this is a one-time process and implementing the code on Fortran adds an additional complication of having to implement matrix inversion and eigenvalue calculation functions. Hence Matlab is used here.

Once the ellipse constants for all sun angles are available, the data from the file is used for the Fourier series parameterization process described in the previous section. The six resulting Fourier series are each saved in a \( n \) by 5 floating number matrix format, where \( n \) is the total number of significant terms in the series. The five parameters for each term are the coefficient and the number of \( S_\alpha \), \( C_\alpha \), \( S_\beta \) and \( C_\beta \) present in the term. The saved Fourier series can be used to quickly determine the estimated AMS ellipse constants for the control allocation problem.

In addition to the ellipse constants for each vane, the ellipse constants for the combined torque are also needed for the purpose of determining feasibility of the desired torque, as noted before. The Fourier series for the combined torque ellipse constants is determined by first generating the two ellipse equations for the pair of vanes using the Fourier series determined above, then generating evenly-spaced points along the boundaries. These points are then added together in every possible combination, generating a set of geometric sums. The convex hull set of these newly-generated points are determined by using \texttt{conv}. Once the convex hull set is determined, the process described above to derive the ellipse constants are once again employed here. This process is once again repeated for every pair of sun angles and saved in a file. Once the file is saved, the exact same algorithm used to determine the Fourier series for the vane ellipses are used to determine the combined ellipse constants, then saved to a file for quick access at later date.

All the steps described above are one-time processes. Once they are executed properly, only the files containing the Fourier series are accessed for the next steps.

With the Fourier series, the ellipse constants for the sun angles describing the sail’s current orientation are calculated. With an input desired torque and the ellipse constants for the combined AMS,
problem defined by Equation (6.31) is first solved using \texttt{fmincon}. The resulting scale factor $\lambda$ is then multiplied to the desired torque. This scaled torque is then used as the desired torque for the equality constraint in Equation (6.30), once again solved using \texttt{fmincon}. The convex nature of the cost function and the nonlinear constraints allow the solvers to arrive at a solution at a surprisingly low number of iterations, keeping the process computationally light. This factor would be important for any on-orbit implementation of the algorithm due to the low computational power available for on-board computers.

Once the minimization problem Equation (6.30) is solved for and the desired torque for each vane is known, the desired torques are used to first determine the vane angle $\theta$ using Equations (6.35) or (6.39). Along with the sun angles and $\theta$, Equation (6.38) or (6.40), is formed, and the root $t$ for each of the vanes are solved for using the \texttt{roots} function. The roots are checked for any imaginary numbers, which are discarded. The remaining roots are then used to derive $\phi$ using the inverse of Weierstrass substitution, which is used to calculate $\hat{s}_v^T \hat{n}_v$. As $\hat{n}_v = [0 \ 0 \ 1]^T$, the result of the calculation is simply the $z$ value of $\hat{s}_v$, and its sign indicates whether the sun is shining on the correct side of the vane or not - namely, if $\hat{s}_v^T \hat{n}_v < 0$, this indicates that the sun is shining from the top of the reflective side, and hence the value of $\phi$ is valid. Otherwise, the sun is not shining onto the reflective side of the vane, and hence such solutions are discarded. While one could in theory allow solutions of such types and adjust the above-given solution process to accommodate such cases, there are other considerations such as different optical properties and thermal consequences for using the back side. These issues were not considered for this research, hence such cases are simply avoided altogether.

Once values of $\phi$ that generate incorrect vane orientations are discarded, the actual torque produced by the remaining $\phi$ are calculated and then compared against the desired torque. Any angle that produces a torque deviating from the desired torque beyond an error threshold value is discarded. Finally, if more than one valid $\phi$ remains, the one that is closest to the previous $\phi$ is taken to be the desired solution. The above process is repeated for all vanes to produce a total of eight vane angles. The last process of the Matlab’s code implementation involves re-calculating the vane torques using the produced vane angles and adding them together to confirm that the total produced torques equal that of the (modified) desired torque.

In Fortran, many of the algorithms used in the Matlab are not as readily available. However, since the Fourier series for the estimated AMS are known, it can either be loaded from the file or hard-coded into the program. Here the latter option was used in order to avoid the hassle of decoding Matlab’s file saving format. The part of the simulation code that generates the vane angles is essentially equivalent to that of Matlab’s own. The minimization algorithm \texttt{fmincon} is replaced with the NPSOL 5.0 package [18], while \texttt{roots} is simply replicated in Fortran with some help from the LAPACK package [4].

The Fortran implementation goes beyond the Matlab implementation by having a controller provide the actual control torque input and an actuator function accepting the output vane angles to produce the torque, which may not produce the actual torque desired due to the sail flexibility. These will be discussed in greater detail in Chapter 8. Here, the Matlab implementation and its simulation results are discussed instead.
6.5 Numerical Example

The capacity of the solution implemented in Matlab is demonstrated by using a randomly-generated set of desired torques and using the above algorithm to calculate a set of vane angles that would correspond to it. Specifically, a torque value is generated by using the following equation:

\[ G_{d,k} = G_{a,k-1} + \text{rand}_3(0.1) \]  

(6.42)

where \( G_{d,k} \) is the current desired torque, \( G_{a,k-1} \) is the previously-achieved torque, and \( \text{rand}_3(0.1) \) generates a \( 3 \times 1 \) vector containing random component values between -0.1 and 0.1. The number 0.1 is chosen to facilitate gradual traversal of the range of achievable normalized torques. The previously-achieved torque is used in place of previously desired torque in order to prevent the desired torques from building up to unreasonably high values. A total of 200 desired torques and the corresponding vane angles are generated, where the number 200 is chosen based on the observation that there’s sufficient variance in the desired torques by then to demonstrate the performance of the algorithm. For the minimization process, \textit{fmincon} offers a number of optimization algorithms. After test simulations, it was determined that the Active-set line search method works best, and is used for this simulation.

Figure 6.8 is a set of plots describing the results of the simulation. Figure 6.8 (a) shows the desired torques, as well as the torques achieved from the vane angles generated using the control allocation process. The vane angles are given in Figure 6.8 (b), and the vane torques are given in Figure 6.8 (c) and (d), along with each vane’s estimated AMS. It can be seen that the vane torques are constrained strictly within the estimated AMS, and the cost function that minimizes the changes in individual vane torques also has an effect on the vane angles, which changes gradually instead of spiking from one value to another as the desired torque changes.

The actual error between the desired and the achieved torques are given in Figure 6.8 (e), and a few spikes of error can be seen. This error is actually caused by the desired torque exceeding the achievable torque, and is a behaviour that is desired - this shows that the feasibility analysis is correctly performing its job and restricting the given desired torque to within the feasibility bound.

The processing speed for the algorithm is measured by the number of iterations taken by the \textit{fmincon} function to arrive at a solution, which is shown in Figure 6.8 (f). It can be seen that for both the feasibility analysis and the control allocation, the number of iterations to arrive at a solution remains small. There is a notable spike in the solution around the middle, but this is a boundary case, as Figure 6.8 (e) and its location of the error spike shows. Actual computing time for the entire simulation on a modern personal computer was measured to be around 20 seconds, which included continuous updates to the plots given above.

There is an issue with the algorithm - in particular with the manner in which the vane angles \( \theta \) are derived. As seen in Figure 6.9, despite the lack of large leaps in desired torques, the resulting vane angles show that, in particular for \( \theta \), large leaps do occur. Unlike \( \phi \), there are no multiple solutions from which \( \theta \) can be chosen from. The significant leaps observed in Figure 6.9 occur most frequently while the sail is edge-on to the sun, while such leaps are almost nonexistent when the sail is facing the sun. It is fortunate then that most missions typically want to keep the sail facing the sun, as the edge-on attitude causes the sail to lose its thrusting capabilities.
Figure 6.8: Simulation Results with $\alpha = \pi/4$, $\beta = \pi/3$
It is worth noting that while this simulation demonstrates the capability of the control allocation scheme under ideal conditions, it does not make any attempt to account for issues such as optical non-ideality of the reflective surface or the flexibility of the booms the vanes are attached to. It will be shown in Chapter 8 that these issues do not infringe upon the control performance as much as one might expect, even with a roughly-calibrated attitude controller.

Figure 6.9: Simulation Results with $\alpha = \pi/2$, $\beta = 0$
Chapter 7

Sail Shadowing Due to Spacecraft Components

Since a solar sail relies on the Sun being shone on its reflective surface to harness the momentum from the photons, problems arise when the sail becomes shadowed. In particular, when a portion of the sail is shadowed by some part of itself due to its orientation with respect to the direction of the sun, it operates with reduced total thrust force and any asymmetry in the shading pattern leads to disturbance torques that adversely affect its attitude profile. For example, an overly large spacecraft bus located at the centre of the sail may cast shadow on one side of the sail as shown in Figure 7.1 and cause disturbance torques along the axis perpendicular to the direction of the shadow. While its effects may be insignificant, providing better understanding to the behaviour of the sail shadowing can better-determine the sail behaviour in edge-on cases. To that end, a method to determine the shadowed portion of the sail given an arbitrary spacecraft component and orientation is introduced.

![Figure 7.1: Square Sail with Shadowing due to Bus](image)

7.1 Projection of a Component Onto the Sail Surfaces

Determining the shadowed area is essentially a problem of projecting the shadowing component onto the reflective sail. Assuming that the component is opaque, the following three pieces of information are needed to perform this projection: the Cartesian location of the component’s outer shell, the sun vector, and the plane in which the sail resides. For simplicity, assume that the component is convex and that its
The outer shell can be described by a finite number of vertices, which are known in the body frame. The sun vector \( \hat{s} \) is also assumed to be known in the body frame. With the FEM model, the sail membrane is represented as a collection of triangular elements, and the nodes forming the vertices of these elements can be used to define a unique Cartesian plane onto which the outer shell can be projected. The goal is to find a set of points that describes the outer boundary of the projected shadow on that portion of the sail.

The steps to find the projection are as follows. The first step is to determine if the component is positioned above or below the sail from the sun’s point of view. If the component is placed below the sail, it will obviously not be able to cast any shadow onto the sail. This is determined by taking the dot product of the component vertex \( p_i \) and the sun vector \( \hat{s} \), and comparing the result against the dot product of the element vertex \( v_j \) and the sun vector \( \hat{s} \). If the value of the first dot product is larger than that of the second dot product, it indicates that \( p_i \) is above \( v_i \). This result is simply due to the geometric definition of the dot product as a scalar projection. It is assumed that all vertices of the component must be above the element vertices to cast shadow upon it (there may be pathological cases where this is not true, but this is neglected for simplicity), hence the check is performed for all combinations of \( p_i \) and \( v_i \). The next steps are performed for individual \( v_i \) only if it passes this check.

The next step is to define the projection plane. Assuming that the coordinates of the triangular vertices are originally known in the body frame and are denoted by \( v_i \), the projection plane can be defined entirely by a normal vector from its origin, which is determined by

\[
\mathbf{n}_b = (v_2 - v_1) \times (v_3 - v_1)
\]

\[
\hat{n}_b = \frac{\mathbf{n}}{\|\mathbf{n}\|}
\]

(7.1)

Alternatively, the plane can also be defined by the canonical plane equation \( Ax + By + Cz + D = 0 \), where \( A = \hat{n}_b(1), B = \hat{n}_b(2), C = \hat{n}_b(3) \) and \( D = -\hat{n}_b^T v_1 \).

It is safe to assume that at any point in a solar sail’s operation, the sun vectors observed from any point of the spacecraft are near-equal. Hence, any projection from the component onto the sail would be a parallel projection. Given any vertex \( p \), finding its projection \( p_p \) on the plane defined by \( Ax + By + Cz + D = 0 \) is equivalent to solving the following system of equations for \( t \):

\[
p_p = p + \hat{s}t
\]

(7.2)

\[
Ap(1) + Bp(2) + Cp(3) + D = 0
\]

(7.3)

Solving the system for \( t \) and replacing \( A, B, C \) and \( D \) with the definitions above,

\[
t = -\frac{Ap(1) + Bp(2) + Cp(3) + D}{A\hat{s}(1) + B\hat{s}(2) + C\hat{s}(3)} = -\frac{\hat{n}_b^T (p - v_1)}{\hat{n}_b^T \hat{s}}
\]

(7.4)

Once \( t \) is known, \( p_p \) is known as well, hence the projection of the shadowing component onto the element plane is known in the body frame.

The projected set of vertices do not define the boundaries of the shadow yet. There is not enough information on how these points form the edges of the shadow on the sail. The next step involves finding out how the vertices form the outer bounds of the projection. First, the projected vertices are rotated into the planar frame. The rotation matrix \( C_{eb} \), which rotates vectors in the body frame to the element
plane frame, can be derived from the normal vector \( \hat{n}_b \) by first deriving its axis-angle representation as follows:

\[
\theta = \cos^{-1}(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \hat{n}_b) = \cos^{-1}(\hat{n}_b(3))
\]

\[
\hat{a} = (\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T) \times \hat{n}_b
\] (7.5)

Then the rotation matrix \( C_{eb} \) can be written as

\[
C_{eb} = \cos \theta \mathbf{1}_3 + (1 - \cos \theta) \hat{a}\hat{a}^T - \sin \theta \hat{a} \times
\] (7.6)

The projected vertices are then translated and rotated into the sail element plane as follows:

\[
p_e = C_{eb}(p_p - v_1)
\] (7.7)

The resulting projected vertices are two-dimensional (z-axis components are zeros). This set of points can be used to determine the convex hull of the set, which by definition is a set of points that define the outer hull of a larger set of points that form a convex shape. This is performed by using the gift-wrapping algorithm, also known as the Jarvis march [9].

The gift-wrapping algorithm is described as shown in Figure 7.2. First, from a set of points on the two-dimensional plane, a leftmost point (a point with the smallest x-axis component) is determined. From this leftmost point, another point amongst the remaining set of points that form the leftmost line (a line that forms the smallest angle with a y-axis line placed at the leftmost point) is chosen. This point is moved to a separate set of points (the leftmost point remains within the original set), then the above-described process is repeated with the remaining points, except that instead of using the y-axis line, the line drawn from the previous point to the current point is used. The process continues until the leftmost point is found once again, at which point the algorithm ends. The newly formed set of points, in the order that they were found, is the convex hull.

Figure 7.2: Determination of the Convex Hull Via Gift-Wrapping Algorithm

Finding the leftmost point is straightforward since the points have been rotated to reside within the plane. Finding the leftmost line can be brute-forced by individually calculating the angle between the
Chapter 7. Sail Shadowing Due to Spacecraft Components

7.2 Determining Shadowed Elements in Sail FEM Model

The shadowed area is identified by the set of points from the convex hull algorithm, but this does not directly address the problem of determining whether the particular element is shaded or not. This issue can be resolved as follows: in the FEM model, any forces applied to the model are resolved as generalized forces on the nodes that define the element. Hence, we make a simple assumption that if the node is shadowed, it produces no force. The accuracy of this assumption depends on the FEM model fidelity and the uniformity of the objects shadowing the sail. The problem then devolves into that of finding whether the individual vertices of the triangular element are within the shadow, i.e. the area defined by the convex hull set.

Conveniently, the \texttt{turn} function used above in the convex hull algorithm can be used to determine whether these nodes are within the shadow or not. In particular, the points of the convex hull set are arranged, thanks to the gift-wrapping algorithm, such that any points within the convex hull are always to the right of the lines created from point \( i \) to \( i + 1 \) of the convex hull set. Conversely, if even one line sees the point to its left, this point is outside of the shaded region. The two cases are illustrated in Figure 7.3.

To this end, the triangular element vertices \( \mathbf{v}_i \), assumed to be known in the body frame, are rotated and translated to the element frame as well, i.e.

\[
\mathbf{v}_{e,i} = C_{cb} (\mathbf{v}_i - \mathbf{v}_1) \tag{7.9}
\]
Then these points are compared to the convex hull set using the \texttt{turn} function, halting the process and marking the node to be not shaded if \texttt{turn} returns non-negative, but otherwise checking against every pair of points. Once all three nodes of the triangular element are checked against the convex hull set, the portion of the SRP force produced by the element and allotted to the shaded node is set to zero, reflecting its lack of contribution to the sail thrust. In this manner the shaded area and its effect on the sail thrust for the FEM model are determined.

### 7.2.1 Numerical Example: Triangular Element Shading Due to Bus

A simple example is created to demonstrate the use of the above-presented algorithm, involving a rectangular bus of dimensions $2 \text{ m} \times 2 \text{ m} \times 2 \text{ m}$ centred at the body frame origin, a randomly generated triangular element and a randomly generated sun vector. These values, assumed to be given in the body frame, are used to determine whether the rectangular bus casts shadow upon the triangular element. To facilitate this example, the procedure described in the previous sections is implemented as a Matlab function, which is called with the given and randomly generated parameters. A sample case is illustrated in Figure 7.4. Figure 7.4 (a) shows the rectangular bus in blue with the sun vector drawn at each of its vertices, and the triangular element drawn in black with its normal vector drawn at the centroid. The red circles indicate the projected vertices on the element plane. The plane in which they are projected onto can be seen more clearly in Figure 7.4 (b), where the first figure has been rotated until the element is seen to be flat. The projected vertices align with the element’s edge, indicating that they are all on its plane.

![Figure 7.4: 3D Diagram of Sail Shadowing Numerical Example](image)

Figure 7.5 illustrates the end result of the algorithm. The black triangle is the element, the blue circles are the projected vertices, and the blue lines represent the convex hull of the projection as calculated by the gift-wrapping algorithm. The red circle indicates an element node within the convex hull (and hence shaded) and the green circle indicates an element node outside of the convex hull (hence not shaded). The simulation performs at a sub-second speed, but this is apparent considering that this is an $O(nh)$ process with at most eleven coordinates involved in it.
Chapter 7. Sail Shadowing Due to Spacecraft Components

7.3 Numerical Example: Sail Membrane Shading Due to Itself

Another interesting example to consider is the case of the sail billowing shape causing shading to other parts of the sail. The model used for this simulation is the steady state sail from the wrinkling simulation, shown in Figure 7.6. The general methodology remains unchanged - one element is designated the shading component, while the others are checked against the element to see if they are shadowed. There are some implementation details to consider however.

\[ t = 6000 \text{ s} \]

Figure 7.6: Cord-Mat Solar Sail with Wrinkling Dynamics
First, because the elements form a larger membrane, their vertices are naturally attached to each other. This proximity causes the turn function to indicate the vertex to be within the convex hull, when in fact the two vertices are overlapping each other. One cannot simply exclude these cases either, because the adjacent elements can validly shadow each other. However, the issue can be mitigated by using an arbitrary point within the element as the points being tested for being shadowed. Specifically, each vertex is assigned a point between itself and the element’s centroid, and then this newly-formed point is checked against the convex hull of the shadowing element. Hence, the points used by the shadowing element and the element being shadowed by it are guaranteed to not overlap, avoiding this issue altogether.

Secondly, the testing has to be performed for every element against all other elements, making this an $O(n^2)$ process, and a particularly slow one since $n$ in this case is the number of membrane elements. This large calculation time overhead can be reduced slightly by keeping track of how many points within the element are shadowed and moving onto checking another element as soon as all three nodes are shaded. Unfortunately, as the sail is expected to operate in a sun-ward direction most of the time, this reduction is not very effective.

Finally, the example is implemented as a Matlab code for testing purposes. The sail billowing shape data used for the example is a static, stabilized set from the wrinkled cord-mat sail Fortran simulation, generated by applying SRP force from the top of the sail (i.e., $\mathbf{s}_i = [0 \ 0 \ -1]$). Hence, it is devoid of any dynamic reaction to the reduced thrust from the sail shading. In reality, a sail in a tilt subjected here to generate the shadows is likely to possess much shallower billows. Nonetheless, the given billow shape emphasizes the shading procedure in a much more exaggerated fashion, and hence is used for demonstrative purposes.

The simulation is performed on a sail inertially rotated along the $x$-axis, followed by another rotation along the $z$-axis. Six $x$-axis rotations are simulated, from an angle just prior to the sail being shadowed, to the angle where the sail is edge-on to the sun. Two different $z$-axis rotations are used, one with the boom pointing towards the sun and another with one sail quadrant towards the sun. The results of the simulation are shown in Figures 7.7 through 7.10.

Figure 7.7 shows the sail in a $\theta_z = 0$ orientation. The green and the yellow elements represent the taut and the wrinkled membranes as before, while the red circles represent the shadowed point between the vertices and the centroid of the element, as explained above. As $\theta_x$ increases gradually to $\pi/2$, the sail shadowing gradually propagates from the left and the right tips of the sail and towards the middle boom, though Figure 7.7 (d) indicates that some parts of the sail around the middle boom begin to be shaded prior to the corner-propagated shading reaching it. By $\theta_x = \pi/2$ shown in Figure 7.7 (f), the entire bottom half of the sail is in the shadow.

The shaded elements are separated and shown in Figure 7.8, where the shaded region is once again shown by the red circle with its element outlined in blue, and the red-outlined element is the element creating the shadow. It can be seen that for the smaller values of $\theta_x$ (i.e. Figures 7.8 (a) through (d)), the shading is almost exclusively caused by the adjacent elements. The shadow is cast by other sail quadrants for Figures 7.8 (e) and (f), but it should be noted that for these cases the shadow would be cast on the back of the sail as opposed to the front. In the purely edge-on case of Figure 7.8 (f), the reflective front of the sail’s bottom-half would be out of the sun regardless of the sail’s upper-half shading it.
Figure 7.9 shows the sail in a $\theta_z = \pi/4$ orientation. Once again the shadowing begins from the corners of the sail and gradually propagates to the centre. Due to the orientation however, the two side quadrants also become gradually shaded, propagating from the top corners and shifting downward until most of the quadrants are shaded as well. It is worth noting that the top corner and a strip along the edge of the quadrants are unshaded in Figure 7.9 (f), but this is the sun shining upon the back of the sail as opposed to the front. The shading pattern can be better seen in Figure 7.10. As before, the sail shading in smaller $\theta_z$ orientations are caused by the adjacent elements, but the larger tilt results in elements from other quadrants causing the shading. Comparing Figures 7.10 (e) and (f) also provides a clearer view of which elements become unshaded as the sail rotates into the edge-on orientation.

Comparing the two rotations using the figures - in particular Figures 7.8 (e) and 7.10 (e), there seems to be a greater imbalance in the forces produced by Figure 7.8 (e) than by Figure 7.10 (e), because for the second case the shading of the two side quadrants begins from the top-half of the sail as opposed to being constrained to the bottom-half of the sail. This creates an interesting situation where the disturbance torque caused by the imbalanced SRP force is much greater for the first case than the second case, suggesting that such edge-on maneuvers are more advantageous to perform with the second orientation than the first. In addition, considering that the typical goal of orienting a solar sail edge-on to the sun is to reduce thrust produced by the sail, this second orientation is once again more desirable as greater area is shaded.

It should be noted that the angles in which these sail shading situations occur are close to edge-on. Forcing the sail attitude orientation such that it is never closer than 5$^\circ$ from being edge-on to the sun for the given set of billowing profile will allow it to avoid being shaded by itself, thereby avoiding this problem altogether. It should also be noted that since the self-shadowing problem only becomes notable when it is near edge-on to the sun and the edge-on sail produces very small amounts of thrust, this problem may be a non-issue. However, because the continuously thrusting nature of the sail causes accumulation of the sailcraft velocity, even small differences in thrust direction may become significant given time.
Figure 7.7: 3D Plot of Sail in Inertial Frame with Shadowing, $\theta_z = 0$
Figure 7.8: Sail Shadowed Region, $\theta_z = 0$
Chapter 7. Sail Shadowing Due to Spacecraft Components

Figure 7.9: 3D Plot of Sail in Inertial Frame with Shadowing, $\theta_z = 0.7854$
Figure 7.10: Sail Shadowed Region, $\theta_z = 0.7854$
Chapter 8

Attitude Control of a Solar Sail

In this chapter, the topics introduced prior to this chapter are combined together to perform an attitude maneuver simulation of the solar sail. The resulting structural dynamics and its effect on the sail’s attitude dynamics, along with the controllability of the given setup, are analyzed here.

8.1 Quaternion PD Control with Vane Control Allocation

For robustness and simplicity, we use the quaternion PD control to determine desired control torques for the system. Hence, let the desired control torque in the body frame $\mathbf{G}_d$ be given by

$$
\mathbf{G}_d = -K_d \omega(t) - k_p \epsilon_e(t)
$$

$$
\epsilon_e(t) = (\eta_d \mathbf{1}_3 - \epsilon_d^x) \epsilon_n(t) - \epsilon_d \eta_n(t)
$$

where $\omega$ is the angular velocity, $\epsilon_e$ is the vector component of the error Euler Parameters, $\epsilon_d$ and $\eta_d$ are the vector and the scalar components of the desired Euler Parameters, and $\epsilon_n$ and $\eta_n$ are the vector and the scalar components of the current Euler Parameters. The values of the current Euler Parameters are given from integrating the kinematic relationship from Section 4.1.3, namely:

$$
\dot{q}_n = \begin{bmatrix} \dot{\epsilon}_n \\ \dot{\eta}_n \end{bmatrix} = \frac{1}{2} \begin{bmatrix} (\epsilon_n + \eta_1 \mathbf{1}_3) \\ -\epsilon_n^T \end{bmatrix} \omega
$$

The control gains $K_d$ and $k_p$ are intentionally tuned such that torques several magnitudes higher than what the vanes are capable of producing are generated until the sail attitude is near the desired attitude. This is because the feasibility portion of the control allocation algorithm automatically scales the requested torque into maximum values that can be achieved by the vanes. The control law would see comparably minimal changes on the attitude and continue to request the same extremely high torque values, holding the vane in the orientation that produces the maximum torque in the desired direction. This setup allows relatively fast slewing maneuvers to be performed. Since the solar sail’s thrust direction and power are directly related to its attitude, the fast slewing capacity for the controller is a highly desired one.

Since the controller at its base is a PD control, the magnitude of the control torque is reduced below the saturation point as the current attitude approaches the desired one. Once the control torque requested by the PD control law is reduced below the saturation point for the vanes, the system as
a whole begins to behave as a standard quaternion PD control, which has been proven to be globally asymptotically stable [50]. While the global asymptotic stability of the control law with the saturation is not proven, one can argue that for cases with small error values such that the torque is not saturated, local asymptotic stability can be achieved.

### 8.1.1 Code Implementation

The attitude control simulation is a compilation of the various topics addressed in this thesis. The basis of the simulation is the nonlinear dynamics simulation in Section 4.7, with the wrinkling criterion from Chapter 5 enabled. The attitude control is emulated by deriving a desired control torque from the PD control law, feeding it through the control allocation algorithm to scale the control torque and derive the vane angles required to achieve the scaled control torque, then passing the vane angles along to the dynamics simulator to calculate the vane forces and torques. The state integration for the current time step is performed to derive the next set of attitude profile, which the PD control uses to update its desired control torque. The whole process is outlined in Figure 8.1.

\[
\begin{align*}
\omega: & \text{ Angular Velocity} \\
q_n: & \text{ Euler Parameters} \\
T_d: & \text{ Desired control torque} \\
\phi_i, \theta_i: & \text{ Vane angles} \\
\end{align*}
\]

**Figure 8.1: Simplified Diagram of Attitude Control Simulation**

The detailed description of each step is as follows: First, the PD control law described in the previous section is used in conjunction with the current attitude profile of the sail to derive a desired control torque. For the first simulation, initial values for \( \omega \) and \( q_n \) (along with the generalized coordinates) are user-specified, while the consecutive runs are performed with the results from the previous run of the ODE solver.

The desired control torque from the PD control law is then scaled by the area of the vane and the solar radiation pressure constant to take into account the force normalization performed when deriving the control allocation method, i.e. \( G_d = G_d/(2PA) \). This scaled desired torque, along with the sun vector in the body frame and the representative sun angles, are used as inputs to the control allocation process described in Chapter 6 to derive the eight vane angles. The vane angles are used instead of the torques derived by the process to properly take into account the discrepancy that occurs due to the flexibility of the booms the vanes are attached to. The body frame sun vector \( \hat{s}_b \) is calculated using the rotation matrix \( C_{bi} \) derived from the Euler Parameters, i.e.

\[
\begin{align*}
\dot{\hat{s}}_b &= C_{bi}\hat{s}_i \\
C_{bi} &= (1 - 2\epsilon_n^T\epsilon_n)I_3 + 2\epsilon_n\epsilon_n^T - 2\eta_n\epsilon_n^\times 
\end{align*}
\]  

(8.3)
where $\epsilon_n$ and $\eta_n$ are the Euler Parameter components from either the previous simulation or the specified initial value.

It should be noted that there is a duplication of information when providing both the sun vector and the sun angles, and this is mainly due to the fact that the current implementation uses the sun vector to calculate the forces, but the solution to the single vane problem is designed around the sun angles. The derivation of the sun angles from the sun vector is straightforward when considering Equation (6.5): $\alpha = \cos^{-1} \hat{s}_b(3)$ and $\beta = \arctan(\hat{s}_b(2), \hat{s}_b(1)), \alpha \neq 0$. If $\alpha = 0$, then the value of $\beta$ cannot change the orientation of the sun vector and hence is set to zero.

Once the vane angles are known, the next time step of the nonlinear dynamics simulation can be simulated using the ODE solver, as per Chapter 4. An additional process is added onto the functions being integrated - that of calculating the force due to the vanes at the ends of each boom. The SRP force due to the vane is calculated with Equation (3.3), using the vane’s normal vector and the sun vector in the body frame. The sun vector is calculated in the same manner as before, while the vane’s normal vector is calculated using the following assumptions: 1. Each vane is aligned with the line formed by the last two nodes of the boom the vane is attached to (this frame is referred to as the deformed frame henceforth), 2. Vane rotations are performed at the deformed frame, and 3. Vane rotations do not cause force centre translation, which is assumed to be at the tip of the boom. Given the above assumptions, the calculation of the vane normal can be devolved into finding two rotation matrices - one that rotates the vane frame to the deformed frame, and another that rotates the deformed frame to the body frame. The first of these, $C_{dv}$, is calculated using the same set of rotation matrices used to calculate the torques in the control allocation problem - specifically,

$$C_{dv} = \begin{cases} (C_{y,\phi_1} C_{x,\theta_1})^T & \text{for vane 1} \\ (C_{x,\phi_2} C_{y,\theta_2})^T & \text{for vane 2} \\ (C_{y,\phi_3} C_{x,\theta_3})^T & \text{for vane 3} \\ (C_{x,\phi_4} C_{y,\theta_4})^T & \text{for vane 4} \end{cases}$$

(8.4)

where $C_{i, \theta}$ is a principal rotation of $\theta$ about the $i$-axis. These rotations are consistent with the degrees of freedom shown in Figure 6.3 from Chapter 6. The rotation matrix from the deformed frame to the body frame is calculated by noting that the rotation between two vectors are easily found by taking the cross product of the two vectors to find the axis of rotation, with its angle of rotation found from the dot product of the two vectors. Specifically, $C_{bd}$ is calculated by

$$C_{bd} = (\cos(\theta_a)1_3 + (1 - \cos(\theta_a))\hat{a}\hat{a}^T - \sin(\theta_a)\hat{a}^X)$$

$$\hat{a} = \frac{a}{\|a\|}, \quad a = \hat{t} \times \hat{n}_r, \quad \theta_a = \cos^{-1}(\hat{t} \cdot \hat{n}_r)$$

$$\hat{t} = \frac{t}{\|t\|}, \quad t = p_2 - p_1$$

(8.5)

$$\hat{n}_r = \begin{cases} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{bmatrix}^T & \text{for vane 1} \\ \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^T & \text{for vane 2} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}^T & \text{for vane 3} \end{cases}$$

where $p_i$ are the deformed coordinates of the last two nodes of the boom in the body frame. Once the
body frame sun vector and the vane normal vectors are known, the solar radiation pressure force can be calculated.

The force due to the vane is assumed to be a point force on the edge of the boom - in terms of the dynamics simulation, the resulting generalized force vector $f_{T,v}$ is written as follows:

$$f_{T,v} = \begin{bmatrix} F_v^T & G_v^T & f_{e,v}^T \end{bmatrix}^T$$

$$F_v = 2PA(\hat{s}_b \cdot \hat{n}_{v,b})^2 \hat{n}_{v,b}$$

$$G_v = p_2^x F_v$$

$$f_{e,v} = \tilde{P}_v^T F_v$$

$$\tilde{P}_v = \begin{bmatrix} 0_{3\times6}I_3 & 0_{3\times3} \end{bmatrix}$$

where $p_2$ denotes the deformed coordinate of the boom edge. The element generalized force vector $f_{e,v}$ must be placed in the appropriate index of the global generalized force vector to affect the edge node as intended, while $F_v$ and $G_v$ are simply added to the existing rigid body generalized force.

The PD control law acts outside of the ODE solver, but the control force provided by the vanes are calculated alongside the SRP force due to the sail and is dynamically modified within the ODE solver. The flexibility of the points in which the vanes are attached causes changes in the control torque generated by the vanes even when the vane angles aren’t changing. This is a plausible on-orbit scenario, as the on-orbit controller may only be able to provide control vane angles at pre-specified intervals due to electronic hardware limitations or for stability reasons. The simulation captures this dynamic with its continuous update to the control torque based on a fixed vane angle within a fixed time-step. The other realistic addition to the controller setup would have been implementing command and actuation delays, but these have been omitted here for simplicity.

The remaining details of the dynamics simulation are equivalent to those introduced in Chapters 4 and 5. The ODE solver provides the generalized displacement and the attitude information of the system at the next time step, then the wrinkling criterion uses the updated principal stresses and strains to determine the sail membrane’s current state and updates the constitutive relationship matrix appropriately. This process repeats itself with the newly-found states until a designated time step is reached.

### 8.1.2 Problem Definition and Simulation Setup

The problem is as follows: Rotate a steady state cord-mat solar sail to a prescribed orientation defined by a $\hat{z}$-$\hat{y}$-$\hat{x}$ Euler rotation by using the above-described control method. The simulation is performed for a designated period of time, which is chosen to show the sail settling down on the desired attitude profile.

While the simulation setup closely resembles that of Section 4.7, the details are reiterated here for clarity. The sail being simulated has the design shown in Figure 8.2. The control vanes are placed as shown in Figure 8.2 (a), with each triangular quadrant of the sail having the form given in Figure 8.2 (b). The same dimensions from Section 4.7 are used: $L = 70.7$ m, $\Delta = \Omega = 1 \times 10^{-2}$ and $\Gamma = 0.0608$. The structural parameters are given in Table 8.1. The vanes are assumed to be a right isosceles triangle with a side length of 12.5 m. This seems excessively large, but it is a necessary sizing requirement to counteract the disturbance torques produced by the sail, as will be explained later on in this section.
A bus with a dimension of $10 \text{ m} \times 10 \text{ m} \times 10 \text{ m}$ is assumed to be located at the centre of the sail. The bus is intentionally sized large to better see the effects of sail shadowing due to it. As mentioned previously, sail shadowing due to itself has not been implemented for the simulation due to its excessive computational load.

![Figure 8.2: Cord-Mat Square Solar Sail with Vanes](image)

The sail is pre-loaded by SRP with the sun vector $\mathbf{s}_i = [0 \quad 0 \quad -1]^T$ and stabilized before the controller is enabled. This pre-loading process is omitted, since it is amply demonstrated by the simulation given in Section 5.5.4. The simulation begins at 6000 seconds - when the pre-loading simulation ends, with the vane control algorithm enabled. The simulation is performed for 7500 seconds, with 50 seconds in between each time step for a total of 150 time steps. Four attitude profiles are tested with varying controller gains. $\theta_y = \theta_z = 0$ with $\theta_x = \pi/4, \pi/2$, and $\theta_z = \pi/4, \theta_y = 0$ with $\theta_x = \pi/4, \pi/2$. Different control gains are used for each simulation, as listed in Table 8.2. These gains are chosen by trial and error process to minimize settling time and overshoot. Note that $P$ is the SRP constant from Chapter 3, i.e. the gains are scaled by the pressure due to the Sun.

<table>
<thead>
<tr>
<th>Beams</th>
<th>Triangular Plate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sail Boom</td>
</tr>
<tr>
<td>$E (N/m^2)$</td>
<td>$2.87 \times 10^{10}$</td>
</tr>
<tr>
<td>$A (m^2)$</td>
<td>$2.87 \times 10^{-5}$</td>
</tr>
<tr>
<td>$I (m^4)$</td>
<td>$1.6104 \times 10^{-7}$</td>
</tr>
<tr>
<td>$\rho (kg/m^3)$</td>
<td>1440</td>
</tr>
</tbody>
</table>

Table 8.1: Structural Parameters for the Cord-Mat Square Solar Sail

<table>
<thead>
<tr>
<th>Euler Angles</th>
<th>Control Gains</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\theta_x$</td>
<td>$\theta_y$</td>
</tr>
<tr>
<td>$\pi/4$</td>
<td>0</td>
</tr>
<tr>
<td>$\pi/2$</td>
<td>0</td>
</tr>
<tr>
<td>$3\pi/4$</td>
<td>0</td>
</tr>
<tr>
<td>$\pi$</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 8.2: Commanded Euler Angle Orientations and Control Gains Used
8.1.3 Simulation Results: $\theta_x = \pi/2, \theta_y = \theta_z = 0$

Figure 8.3 shows the resulting sail shape in the body frame and the Euler angle rotation of the sail’s attitude, when commanded to an edge-on configuration. The end shape of the sail is very close to that of the sail without any SRP force loads. This is an expected behaviour, as the SRP force is minimal against a reflective surface edge-on to the sun. The sail attitude converges to the desired attitude in a reasonably rapid fashion, reaching 99% settling time (time taken to reach 99% norm of the desired attitude) within 4000 seconds. It is worthwhile to note that there is an odd ‘wriggling’ of $\theta_x$ during the slewing process. It should also be noted that while the membrane element nodes along negative $y$-axis were marked to be shaded, the entire region quickly stopped producing any SRP force due to the sail elements facing away from the sun.

The above-mentioned odd behaviour can be explained by observing Figure 8.4. In particular, note how the sail experiences a significant disturbance torque resisting the rotation induced by the controller. Dubbed the ‘shuttlecock’ effect, this is a phenomenon where the slanted shape of each sail quadrant, combined with the nature of the reflective surface to produce more SRP force when face-on towards the sun, results in a restoring torque that tries to keep the spacecraft in a sunward direction. As observed in Figure 8.4 (a), this disturbance torque grows as the sail is tilted, until it reaches the peak torque just prior to $\theta_x = \pi/4$ and then starts to fall off gradually until it is near zero when the sail is edge-on. The peak disturbance torque is very large - the vanes are only capable of producing at most 0.28 N·m along the $x$ or $y$-axis, leaving around 0.05 N·m of excess torque to control the spacecraft after counteracting the disturbance in worst cases. This disturbance is likely to drive the sizing requirement for the vanes.

The predicted and the actual control torques produced by the vanes are given in Figure 8.4 (b). The difference occurs due to the bending experienced by the boom, which is not part of the mathematical model used to derive the vane angles, but is part of the simulation due to the structural dynamics simulated. However, it can be seen that the difference between the two is minimal. When the sail is face-on towards the sun the difference between the two torques are visible in the plot, reflecting how
the SRP force bends the booms significantly when the sail is face-on to the sun. As the sail is tilted to the edge-on orientation, the plot demonstrates how the difference between the predicted and the actual control torques are reduced. The difference eventually becomes insignificantly small as the sail becomes edge-on to the sun, the boom sag reaching its minimum and only slightly deviated from the straight boom case assumed by the control allocation algorithm.

The sail displacements are shown in Figure 8.5, with the $y$-axis displacement omitted as it is essentially symmetric to the $x$-axis displacement given here. The individual line plots represent the displacement of a node over the simulation time. As the SRP force is reduced with the tilted sun vector on the sail, the sail’s displacement due to those forces are also gradually reduced, until the displacements become insignificant when the sail is edge-on to the sun. As noted above, the changing displacement results in varying accuracy of the vane-produced control torque.
Vanes angles commanded by the control allocation algorithm are plotted in Figure 8.6. It can be seen that for the most part the changes to the angles are not too abrupt, with the exceptions being $\phi_2$ dropping sharply at around 7000 seconds and $\theta_2$ and $\theta_4$ wrapping around at $\pi$. The wraparound experienced by $\theta$ is an expected behaviour, and the abrupt change experienced by $\phi_2$ occurs at around the same time $\theta_2$ goes from near zero to $-\pi$, which is most likely the main cause. A detailed study into the behaviour of the single-vane analytical solution would be needed to mitigate such changes.

![Figure 8.6: Vane Angles During Control Sim. for $\theta_x = \pi/2, \theta_y = \theta_z = 0$](image)

8.1.4 Simulation Results: $\theta_x = \pi/4, \theta_y = \theta_z = 0$

Shown in Figure 8.7 is the sail and its attitude when $\theta_x = \pi/4$ is commanded. The shape of the sail is a reduced catenary billow from the original steady state simulation, but with a caveat. There is a notable difference in displacement between the left and the right sides of the sail. The sail membranes are more perpendicular to the sun vector on the right side than the left side due to the inherent slant in each quadrant, which results in greater force produced on the right side and hence the greater sail deflection on the right side. Apart from this oddity, the attitude convergence is satisfactory - the 99% settled state is gradually reached by 2950 seconds from the start of the simulation. The disturbance torque and the control torques are shown in Figure 8.8. The disturbance remains biased, acting against the control torque to restore the sail attitude back to the sun-facing direction. One can clearly see the effect of the bias on the control torque - the control torque starts high as before, briefly dipping back down below zero (to counteract the angular velocity induced by the control torque), but then rapidly going back up to act against the biased disturbance.

The significant dip observed in the control torque plot is concerning - such rapid changes in torque may not be achievable with the vane actuation. The cause is very likely because of the high gain used for the controller, yet such high gain is necessary in order to counteract the effect of the bias torque while still converging to 99% norm of the desired attitude. While this issue is not explored in detail, it is an issue that should be considered for the future.
Figure 8.7: Control Sim. for $\theta_x = \pi/4$, $\theta_y = \theta_z = 0$

Figure 8.8: Disturbance and Control Torques During Control Sim. for $\theta_x = \pi/4$, $\theta_y = \theta_z = 0$

Once again as the sail tilts away from the sun, its displacements are reduced as seen in Figure 8.9. However the SRP force is still relevant in the final orientation, hence some amount of sail displacement remains still. It is interesting to note that the reduction in displacement along the $x$-axis is more pronounced than that of the $z$-axis - maximum $x$-axis displacement is reduced to approximately 40% of the original value, whereas the maximum $z$-axis displacement is reduced to 60% of the original value.

From Figure 8.10, similar issues observed for $\theta_x = \pi/2$ can be observed for $\theta_x = \pi/4$ as well, where few significant jumps in vane angles are observed. The jumps are located at earlier intervals than before, just prior to and during the significant dip the control torque undergoes. While the problem observed here may disappear if the control torque issue is fixed, it may also be an issue associated with the analytical solution to the vane angles as mentioned in the previous section, an entirely separate issue that should be considered.
8.1.5 Simulation Results: $\theta_x = \pi/2$, $\theta_y = \theta_z = \pi/4$

Figure 8.11 is the result of commanding the sail to point towards the orientation where $\theta_x = \pi/2$ and $\theta_z = \pi/4$. Once again, the final shape of the sail is close to that of the unloaded sail - an expected behaviour as the the SRP force on the edge-on sail is minimal, mimicking the behaviour seen from Section 8.1.3. The 99% angle convergence is seen to occur at 9050 seconds - a faster convergence than the first case, but this may simply be a gain optimization issue. It is worthwhile to note that a small overshoot is observed with $\theta_z$, along with significant undershoot for $\theta_y$. The overshoot is presumably due to the nature of the controller being used - it assumes same gain magnitudes across all dimensions despite the fact that greater control torque is available along $\theta_z$ than the other two axes. The undershoot on the other hand is more likely due to the disturbance the sail undergoes as it slews to the desired attitude.
As seen in Figure 8.12, there is a notable disturbance torque in the $y$-axis, explaining the $\theta_y$ undershoot. As before, the $x$-axis disturbance torque is also present, growing until a maximum value is reached at $\theta_x = \pi/4$ and reducing until near zero at $\theta_x = \pi/2$. Given that $\theta_x$ and $\theta_z$ are both changed and that disturbance torques are experienced about the $y$-axis, control torques about all three axes are demanded by the controller, as shown in Figure 8.12 (b). A similar spike as seen from Figure 8.4 (b) is seen here as well around 8000 seconds. Once again, as the sail approaches the edge-on orientation, the difference between the predicted and the actual control torques is reduced due to progressively smaller boom deflections.

Figure 8.11: Control Sim. for $\theta_x = \pi/2$, $\theta_y = 0$, $\theta_z = \pi/4$

Figure 8.12: Disturbance and Control Torques During Control Sim. for $\theta_x = \pi/2$, $\theta_y = 0$, $\theta_z = \pi/4$

Moderate changes to the vane angles can be observed from Figure 8.13, concentrated within the span of 7000 to 9000 seconds - the point in which rapid changes to the control torques are observed. The
previously-noted wraparound issue is seen here for \( \theta \) as well. Once again, these are problems where further research is warranted.

\[
\begin{align*}
\text{(a) Vane Angle } \phi \\
\text{(b) Vane Angle } \theta
\end{align*}
\]

Figure 8.13: Vane Angles During Control Sim. for \( \theta_x = \pi/2, \theta_y = 0, \theta_z = \pi/4 \)

8.1.6 Simulation Results: \( \theta_x = \pi/4, \theta_y = \theta_z = \pi/4 \)

When the sail is commanded to \( \theta_x = \pi/4 \) and \( \theta_z = \pi/4 \), the sail begins to have issues. Specifically, tuning the proportional gain to allow the Euler angles to settle to 99% of the desired attitude within a reasonable time causes significant overshoot along \( \theta_z \) which is deemed undesirable. Increasing the derivative gain to mitigate the overshoot results in significant oscillation in the control torques, a highly undesirable behaviour. What’s shown in Figure 8.14 is a result of multiple attempts to tune the gain parameters such that 99% settling state is reached without significant overshoot and control torque oscillation. One can see a similarly-tilted sail displacement pattern observed from Section 8.1.4 - with the sun to the right of the figure, the right side of the sail in general is more displaced than the left side of the sail. The Euler angle plot shown in Figure 8.14 indicates that there is a 20% overshoot for \( \theta_z \), along with an unimpressive settling time of nearly 5000 seconds (albeit 90% settling state was reached earlier on). The \( \theta_y \) undershoot observed from Section 8.1.5 is also observed here, behaving in a similar manner as before. A dip in attitude for \( \theta_x \) is also observed, around the time \( \theta_z \) experiences an overshoot - the most likely cause of the dip.

Once again, the disturbance torques along \( x \) and \( y \) axes can be seen just as it was seen in the previous section, and as with the simulation from Section 8.1.4, these disturbance torques persist throughout the simulation. As a result, a de-biasing control torque needs to be supplied by the vanes to counteract these disturbances and keep the sail pointing in the desired direction - Figure 8.15 (b) shows how \( G_x \) and \( G_y \) are set to non-zero values even after the slew has completed. The plot also shows that the changes in these control torques aren’t as sudden as the previous three simulations, but this is likely attributed to the slower convergence experienced by this particular simulation compared to them. In addition, a small amount of control torque oscillation can be seen between 9000 and 10000 seconds, which does damp out but is a cause for concern.
It is worth noting that because of the overshoot/undershoot the sail attitude undergoes, the sail deflection also undergoes a sloping behaviour, as seen in Figure 8.16. The final reduction in sail displacement is about the same as that of Figure 8.9, but the transient behaviour to the settled state experiences brief bumps in the displacements around the time the $\theta_z$ overshoot occurs. Considering Figure 8.14 (b) and 8.15 (a), it seems clear that the bump is caused by $\theta_x$ being reduced between 8000 and 9000 seconds and resulting in relatively larger SRP forces.

The general trend for the vane angles as seen in Figure 8.17 is that moderate changes are observed throughout the first 4000 seconds, which then experience a brief oscillation between 9000 to 10000 seconds before settling down to a smoother curve for the remainder of the simulation. The first 4000 seconds is highly reminiscent of Figure 8.13, and the behaviour once again corresponds to the rapid
changes in the control torque. The oscillation is clearly due to the control torque oscillation from Figure 8.15 (b). It is worthwhile to note that the vane angles do not experience the \( \theta \) wraparound seen in other simulations.

Note that this attitude can also be achieved by a sequentially-commanded set of attitudes - first turn the sail about \( z \)-axis by \( \pi/4 \) and then about \( x \)-axis by \( \pi/4 \) - the resulting attitude is equivalent, but it allows two different gains to be used to control each rotation maneuver separately. One such simulation is shown in Figure 8.18. To achieve this, the simulation was initially set with \( k_p = 1 \times 10^6 \) P Nm/rad and \( K_d = 2.2 \times 10^2 k_p l_3 \) Nm/rad/s, then with those control gains the sail is commanded to the first attitude in the sequence - \( \theta_x = \theta_y = 0 \), and \( \theta_z = \pi/4 \). These control gains and the desired attitude are maintained until the 99% settling state is reached - in case of this particular simulation, this is at 8000 seconds and denoted by the dot-dash line in Figure 8.18 (a).
Once the first sequence attitude is achieved, a new set of control gains are specified: $k_p = 1.5 \times 10^6 \text{Nm/rad}$ and $K_d = 9 \times 10^6 k_p \text{Nm/rad/s}$, with the new desired attitude set as $\theta_x = \pi/4$, $\theta_y = 0$, and $\theta_z = \pi/4$. One can immediately observe its effects from Figure 8.18 (a), where $\theta_x$ begins its ascent to the desired value. One can also observe a slight overshoot, no doubt caused by the changing $\theta_x$ and the resulting disturbance, but because it’s already close to its desired value it is able to quickly settle back. At the same time, since $\theta_x$ is the only angle with large changes, the controller is able to provide appropriate torques to converge to its desired value within a short period of time. The convergence behaviour is similar to the second simulation - after the second sequence is commanded, the sail attitude reaches 90% settling state at around 1000 seconds, and takes another 2000 seconds to converge to the 99% settling state. The combined 99% settling time for the two sequences is 5150 seconds, which is only 200 seconds longer than the previous simulation, and avoids most of the issues associated with it.

Note that based on what’s seen in Figure 8.18 (b), one could argue that the controller behaviour of the previous simulation is preferable to this one, which contains several steep jumps from one set of torques to another. Such changes to the control torques result in rapid and sudden changes to the vane angles as shown in Figure 8.19, particularly at around 9000 seconds when the sail reaches 90% settling state. Once again, this is presumably due to the compensation action for the rapid slewing experienced by the sail, which is necessary to keep the sail pointing in the right direction against the biased sail SRP torques.

### 8.2 Handling Disturbance Torques Due to Sail Structure

Considering the significant biased disturbance torques the sail produces, it would be ideal to simply calculate this disturbance torque from the current sail shape and subtract it off the desired torque to counteract its effects. However, determining the sail deflection on-orbit is not feasible at all, and hence it is unrealistic to assume such data will be available as a part of the controller. Instead, two other approaches are briefly considered here.
8.2.1 Statically-Shifted Centre of Mass

The model used for the simulation does not assume any mass due to the satellite bus. As a result the centre of mass of the sail system is located about the negative $z$-axis, shifting along it for the most part as the sail slews around at different angles. The addition of the spacecraft bus at the centre would change the location of the centre of mass. As a preliminary effort, a point mass, representing the spacecraft bus, is added to the sail such that it counterbalances the sail’s mass and sets the centre of mass at the origin.

Placing a rigid point mass into the FEM model is simple, as the modification is constrained to the values of $c$ and $J$. Specifically, a point mass weighing $m$ located at $\rho$ has the following values for $c$ and $J$:

$$
\begin{align*}
    c_{pm} &= m\rho \\
    J_{pm} &= -m\rho^2 \rho^x
\end{align*}
$$

The effect of the point mass to the sail model is that these values are added to the global $c$ and $J$.

In case of the division-6 sail used to perform the controller simulations above, the centre of mass is found to be located at $[0 \ 0 \ -1.13]$ m, with the entire sail weighing at 178 kg. Hence, a point mass weighing 178 kg is placed at $[0 \ 0 \ 1.13]$ m, forcing the centre of mass to the origin. The result of the simulation is omitted here, as the behaviour of the system is essentially unchanged. None of the above-given simulation cases experienced any changes to its results from the addition of the point mass representing the spacecraft bus. This result is somewhat predictable considering that the ratio between the changed distance in the centre of mass and the size of the sail is small.

Any significant modifications that can be made to influence the result are infeasible. Increasing the spacecraft bus mass is undesirable as the mass directly influences the efficiency of the solar sail and hence smaller spacecraft bus masses are desirable, yet increasing the point mass to a tonne barely affects the simulation result. Shifting the centre of mass further below the sail may actually lessen the disturbance torque due to the shuttlecock effect, but at a cost of reducing stability at the resting position. Such
a configuration also indicates that majority of the spacecraft bus is located below the sail, reducing valuable bus surface area visible in the sun for power and attitude determination purposes. Placing the mass further out will also affect the performance, but such setup indicates that the spacecraft bus is connected to the sail via a boom, which introduces an entirely different structural dynamics problem to handle and is not within the scope of this research.

8.2.2 Static Bias Torque Subtraction

While the sail structural dynamics and the resulting disturbance torques are infeasible to calculate, such a calculation based on a fixed, static sail is entirely possible. In fact, one can easily create a Fourier series approximation or a table of values that would specify what the disturbance torques would be, with the sun vector or the sun angles as the variables.

In case of the control simulation present, a simple method of calculating the torques using the initial, stable deformed state at 6000 seconds is used. The initial state of the sail represents the maximum deflection the sail will undergo, hence the disturbance torques predicted by it is quite conservative. This predicted disturbance torque is subtracted from the control torque generated by the PD control law before being used by the control allocation algorithm to derive the vane angles.

One such example is shown in Figure 8.20 - this is the control simulation of the attitude $\theta_x = \theta_z = \pi/4$, $\theta_y = 0$, with a static bias torque subtracted off the desired control torque. Comparing Figure 8.20 (a) against Figure 8.14 (b), a much faster 99% settling time is observed. However, Figure 8.20 (b) shows the controller oscillation mentioned in the previous section. In addition, apart from the controller oscillation, the control torque generated by this setup is not very different from the control torque generated by the controller without the static bias - the general pattern is essentially the same. The addition of static bias seems to emulate the result of increasing the control gains, suggesting that this method, unfortunately, is not very effective in providing a better control system.

Figure 8.20: Vane Angles During Control Sim. for $\theta_x = \theta_z = \pi/4$, $\theta_y = 0$, with Static Biasing Torques
8.3 Controllability of the Vane Actuators

While the above simulations, along with the degrees of freedom the vane actuator system possesses, suggests that the spacecraft possesses full controllability of its rigid modes, no such statements can be made for the flexible modes, and verification of the rigid mode controllability would also be desirable. Unfortunately, such an endeavour is not an easy task due to the significant nonlinearities of the equations of motion preventing the use of any of the typical controllability analyses, which rely on the linearity of the system.

For simplicity, a linearized system is created and the controllability test performed on the linearized system. For clarity, the linearization is performed from first principles for the mass and the stiffness matrices, while due to the complexity of the equations involved the linearization for the vane torque terms are derived via the standard differentiation method.

8.3.1 Linearization of the Equations of Motion

Recall the nonlinear equations of motion from Chapter 4:

\[ M(q)\ddot{q} + K(q)q = f_T(q, \theta) + f_f(q, \dot{q}) \]  

(8.8)

where the actuation state has been added to \( f_T \) as the vane angles \( \theta = [\phi_1 \ \theta_1 \ \phi_2 \ \theta_2 \ \phi_3 \ \theta_3 \ \phi_4 \ \theta_4] \).

It is assumed that \( q = q_0 + \delta q \) and \( \dot{q} = \dot{\delta q} \), i.e. the sail is initially at rest and deflected to an initial position \( q_0 \), which is then perturbed by a small value \( \delta q \). While performing the linearization, any quadratic or higher terms of \( \delta q \) are deemed small and are neglected.

The stiffness term \( Kq \) is expanded as

\[
Kq = \begin{bmatrix}
0 & 0 & 0 \\
0 & 0 & 0 \\
0 & K_L + (1/2)N_1(q_e) + (1/3)N_2(q_e) & 0 \\
\end{bmatrix}
\]

(8.9)

where \( q_e \) is the generalized coordinates for the elastic body. Expanding \( N_1(q_e)q_e \) results in

\[
N_1(q_e)q_e = N_{1,ijk}q_e q_{e0,j} + N_{1,ijk}q_{e0,j}q_{e,j} + N_{1,ijk}q_{e,j}q_{e0,j} + \ldots
\]

\[
\approx N_1(q_{e0})q_{e0} + N_1(q_{e0})\delta q_e + N_{1,1}\delta q_e,
\]

(8.10)

where \( N_{1,ijk} \) represent the \( k^{th} \) term of the \( ij^{th} \) matrix element as explained in Chapter 4. Einstein’s summation notation is used here. The omitted terms are those quadratic to \( \delta q_e \). Similarly for \( N_2(q_e)q_e \),

\[
N_2(q_e)q_e = N_{2,ijkl}q_e q_{e0,k}q_{e0,l}q_{e0,j} + N_{2,ijkl}q_{e0,k}q_{e0,l}q_{e,j} + N_{2,ijkl}q_{e,j}q_{e0,k}q_{e0,l}q_{e0,j} + \ldots
\]

\[
\approx N_2(q_{e0})q_{e0} + N_2(q_{e0})\delta q_e + N_{2,1}\delta q_e + N_{2,2}\delta q_e,
\]

(8.11)

where \( N_{2,ijkl} \) represents the \( k^{th} \) term of the \( ij^{th} \) matrix element. Several quadratic and cubic \( \delta q_e \) terms have been omitted once again.

Note that since \( N_1 \) and \( N_2 \) are symmetric in the sense that \( N_{1,ijk} = N_{1,ikj} \) and \( N_{2,ijkl} = N_{2,ikjl} = \ldots \),
The mass term \( M(q)\dot{q} \) is expanded as:

\[
M(q)\dot{q} = \begin{bmatrix}
  mI_3 & -c^x - (\delta c(q))^x & P \\
  c^x + (\delta c(q))^x & J + \delta J(q) & H + \delta H(q) \\
  P^T & H^T + (\delta H(q))^T & M_{ee}
\end{bmatrix} \begin{bmatrix}
  \dot{\nu} \\
  \delta \omega \\
  \delta \dot{q}_e
\end{bmatrix}
\]

where because of the assumption that the sail is initially at rest, the derivative states are all initially zero and hence omitted here. The key terms to consider here include \((\delta c(q))^x \delta \omega, \delta J(q) \delta \omega\) and \(\delta H(q)\delta \dot{q}_e\) - these are terms that are not linear about the states. Expanding each of these terms out,

\[
(\delta c(q))^x \delta \omega = (P_i(q_{e0,i} + \delta q_{e,i}))^x \delta \omega
\]

\[
= (Pq_{e0})^x \delta \dot{\omega} + ...
\]

\[
\approx (\delta c(q_{e0}))^x \delta \dot{\omega}
\]

\[
\delta J(q)\delta \dot{\omega} = (-(\tau_i + \tau_i^T)q_{e0,i} + \delta q_{e,i}) - \chi_{ij}(q_{e0,i} + \delta q_{e,i})(q_{e0,j} + \delta q_{e,j}) \delta \dot{\omega}
\]

\[
= (-(\tau_i + \tau_i^T)q_{e0,i} + \delta q_{e,i}) \delta \dot{\omega} + ...
\]

\[
\approx (-(\tau_i + \tau_i^T)q_{e0,i} + \delta q_{e,i}) \delta \dot{\omega} + ...
\]

\[
\delta H(q)\delta \dot{q}_e = \nu_{ij}(q_{e0,j} + \delta q_{e,j}) \delta \dot{q}_{e,i}
\]

\[
\approx \nu_{ij} q_{e0,j} \delta \dot{q}_{e,i}
\]

\[
= \delta H(q_{e0}) \delta \dot{q}_e
\]

That is, all of the nonlinear terms when linearized are simply the expressions derived using \(q_{e0}\). Substituting the above expressions back into Equation 8.15, we have

\[
M(q)\dot{q} \approx \begin{bmatrix}
  mI_3 & -c^x - (\delta c(q_{e0}))^x & P \\
  c^x + (\delta c(q_{e0}))^x & J + \delta J(q_{e0}) & H + \delta H(q_{e0}) \\
  P^T & H^T + (\delta H(q_{e0}))^T & M_{ee}
\end{bmatrix} \begin{bmatrix}
  \dot{\nu} \\
  \delta \omega \\
  \delta \dot{q}_e
\end{bmatrix} = M(q_{e0})\delta \dot{q}
\]
Since $\dot{q}_0$ is assumed to be zero, the above expression can also be written as

$$M(q)\ddot{q} \approx M(q_0)\ddot{q}_0 + \left[ \frac{\partial M(q)}{\partial \dot{q}} \right]_{q_0} \delta \dot{q}$$

(8.18)
i.e. the small change approximation results in a linearization equation with the derivative term with respect to $q$ omitted.

The inertial forcing term $f_T(q, \dot{q})$, expanded in Equations (4.50) and (4.54), are all zeros when linearized, because every term in it involves two or more derivative states. The vane forcing term $f_T(q, \theta)$ is linearized by differentiation:

$$f_T(q, \theta) \approx f_T(q_0, \theta_0) + \left[ \frac{\partial f_T}{\partial \dot{q}} \right]_{q_0, \theta_0} \delta \dot{q} + \left[ \frac{\partial f_T}{\partial \theta} \right]_{q_0, \theta_0} \delta \theta$$

(8.19)
and then calculating the derivatives directly. The expression for $f_T$ is the combination of the sum of the four vane forces $f_{T,v}$ introduced in Section 8.1.1 and the forces generated by the sail itself, $f_{T,c}$.

Let the rigid body force produced by the vane be $F_{T,v}$. Its partial derivatives can then be written in terms of the known variables as follows:

$$\frac{\partial F_v}{\partial \dot{q}_e} = -2PA \left[2s_b^T \hat{n}_{v,b} \hat{n}_{v,b} s_b^T + (s_b^T \hat{n}_{v,b})^2 l_3 \right]$$

$$\frac{\partial \hat{n}_{v,b}}{\partial a} = (1 - \cos \theta_a)(\hat{\alpha}_T (C_{d_v} \hat{n}_v) l_3 + \hat{\alpha}(C_{d_v} \hat{n}_v)^T) + \sin \theta_a (C_{d_v} \hat{n}_v)^x$$

$$\frac{\partial \hat{\alpha}}{\partial a} = \frac{1}{a^T a} l_3 - \frac{1}{(a^T a)^{3/2}} a a^T$$

$$\frac{\partial \hat{t}}{\partial \dot{t}} = \frac{1}{\sqrt{t^T t}} l_3 - \frac{1}{(t^T t)^{3/2}} t t^T$$

$$\frac{\partial \hat{n}_{v,b}}{\partial \dot{q}_a} = (-\sin \theta_a l_3 + \cos \theta_a \hat{\alpha}^T - \cos \theta_a \hat{\alpha}^x) C_{d_v} \hat{n}_v, \quad \frac{\partial \theta_a}{\partial \dot{t}} = \frac{-T}{\sqrt{1 - T^T t} \hat{n}_r}$$

(8.21)

$$\frac{\partial \hat{n}_{v,vi}}{\partial \phi_i} = C_{bd} \left( \frac{\partial C_{y,vi}}{\partial \phi_i} C_{x,theta} \right)^T \hat{n}_v, \quad \frac{\partial \hat{n}_{v,vi}}{\partial \theta_i} = C_{bd} \left( \frac{\partial C_{y,phi} C_{x,theta}}{\partial \theta_i} \right)^T \hat{n}_v, \quad i = 1, 3$$

$$\frac{\partial \hat{n}_{v,vi}}{\partial \phi_i} = C_{bd} \left( \frac{\partial C_{x,phi}}{\partial \phi_i} C_{y,theta} \right)^T \hat{n}_v, \quad \frac{\partial \hat{n}_{v,vi}}{\partial \theta_i} = C_{bd} \left( \frac{\partial C_{x,theta} C_{y,theta}}{\partial \theta_i} \right)^T \hat{n}_v, \quad i = 2, 4$$

$$\frac{\partial C_{x,theta}}{\partial \theta} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & -\sin \theta & \cos \theta \\ 0 & -\cos \theta & -\sin \theta \end{bmatrix}, \quad \frac{\partial C_{y,theta}}{\partial \theta} = \begin{bmatrix} -\sin \theta & 0 & -\cos \theta \\ 0 & 0 & 0 \\ \cos \theta & 0 & -\sin \theta \end{bmatrix}$$

While lengthy, the above derivations are simply a series of chain rules upon the equations introduced in
Similarly, let the rigid-body torque and the generalized elastic forces for each element be written as

$$\mathbf{G}_e = \hat{c}^\times \mathbf{F} + \hat{\mathbf{P}}_i^T \mathbf{F} \eta_i$$

where

$$\hat{c} = \frac{c}{\rho h}, \quad \hat{\mathbf{P}} = \frac{\mathbf{P}}{\rho h}$$

Given the above notations, the derivative terms can be written as

$$\frac{\partial \mathbf{F}}{\partial \mathbf{q}_e} = \frac{\partial \mathbf{F}}{\partial \mathbf{v}} \frac{\partial \mathbf{v}}{\partial \mathbf{q}_e} + \sum_{i=1}^{3} \frac{\partial \mathbf{v}}{\partial \mathbf{u}_i} \frac{\partial \mathbf{u}_i}{\partial \mathbf{q}_e}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{n}_{e,b}} = \left[ 2 \mathbf{6}^T \mathbf{f}_{e,b} \mathbf{n}_{e,b}^T + (\mathbf{s}_b^T \mathbf{n}_{e,b}) \right]$$

$$\frac{\partial \mathbf{v}}{\partial \mathbf{n}_{e,b}} = \frac{1}{\sqrt{\mathbf{n}_{e,b}^T \mathbf{n}_{e,b}}} \mathbf{1}_3 - \frac{1}{(\mathbf{n}_{e,b}^T \mathbf{n}_{e,b})^{3/2}} \mathbf{n}_{e,b} \mathbf{n}_{e,b}^T$$

(8.28)
for the rigid body forces, and
\[
\frac{\partial G}{\partial q_e} = \frac{\partial G}{\partial F} \frac{\partial F}{\partial q_e} + \text{row} \frac{\partial G}{\partial q_i}, \quad \frac{\partial f_{T, e}}{\partial q_e} = \frac{\partial f_{T, e}}{\partial F} \frac{\partial F}{\partial q_e} \tag{8.29}
\]
for the rigid-body torques and the generalized elastic forces.

The actual equation of motion that is of interest is created by substituting Equations (8.13), (8.18) and (8.19) into Equation (8.8), then subtracting the resulting expression by the equations of motion at the initial value \(q_0\):
\[
M(q)\ddot{q} - M(q_0)\ddot{q}_0 + K(q)q - K(q_0)q_0 = f_T(q, \theta) - f_T(q_0, \theta_0) + f_i(q, \dot{q}) - f_i(q_0, \dot{q}_0)
\]
\[
M(q_0)\delta \ddot{q} + \left[ \frac{\partial K(q)}{\partial q} \right]_{q_0} \delta q = \left[ \frac{\partial f_T}{\partial q} \right]_{q_0, \theta_0} \delta q + \left[ \frac{\partial f_T}{\partial \theta} \right]_{q_0, \theta_0} \delta \theta \tag{8.30}
\]
The last equation is linear about \(\delta q\) and \(\delta \theta\). However, the \(\delta q\) term on the right hand side is undesirable. The controllability result to be used requires a linear form of \(Mq + Kq = Bu\) where \(M\) and \(K\) are both symmetric, yet because of the \(\left[ \frac{\partial f_T}{\partial q} \right]_{q_0, \theta_0}\) \(\delta q\) term, the symmetry possessed by \(\left[ \frac{\partial K(q)}{\partial q} \right]_{q_0}\) is lost. Hence, it is assumed that \(\left[ \frac{\partial f_T}{\partial q} \right]_{q_0, \theta_0}\) is sufficiently small to be ignored. With that assumption, the linearized equations of motion can be written as
\[
M(q_0)\delta \ddot{q} + \left[ \frac{\partial K(q)}{\partial q} \right]_{q_0} \delta q = \left[ \frac{\partial f_T}{\partial \theta} \right]_{q_0, \theta_0} \delta \theta \tag{8.31}
\]

### 8.3.2 Testing the Controllability of the Sail

Once the linearized equations of motion is present, the actual check for the system controllability is quite simple. Referring to a paper by Hughes and Skelton [23], assume that a system is defined by
\[
M\ddot{q} + Kq = Bu \tag{8.32}
\]
where
\[
M = \begin{bmatrix} J & H^T \\ H & M_{ee} \end{bmatrix}, \quad K = \begin{bmatrix} 0 & 0 \\ 0 & K_{ee} \end{bmatrix}
\]
\[
B = \begin{bmatrix} B_{rr} & B_{re} \\ 0 & B_{ee} \end{bmatrix}, \quad u = \begin{bmatrix} u_r \\ u_e \end{bmatrix}
\]
The matrix \(B\) defines the manner in which the control actuation input \(u\) affects the system as a whole. The components of \(u_r\), \(u_e\), and \(u_e\), are the control inputs to the rigid and the elastic bodies. The submatrices \(B_{rr}\), \(B_{re}\) and \(B_{ee}\) represent how \(u_r\) and \(u_e\) affect the rigid and the elastic bodies. Given such a system, Hughes and Skelton show that the controllability of the rigid modes and the elastic modes are defined by \(C_r\) and \(C_e\), where
\[
C_r = \left[ \frac{\det(B_{rr}B_{rr}^T + B_{re}B_{re}^T)}{\det J} \right]^{1/6}
\]
\[
C_e = \text{row} \left\| T_{e}^T B \right\|
\]


where $T_\alpha$ represents an elastic mode eigenvector when the eigenvalue problem

$$KT = \Lambda MT$$

(8.35)
is solved. Each mode of the system is controllable if

$$C_r > 0, \quad C_{e,i} > 0$$

(8.36)

where $C_{e,i}$ represent the $i^{th}$ element of the vector $C_e$. The magnitudes of $C_r$ and $C_{e,i}$ represent the degree of controllability for each mode.

Comparing Equations (8.31) to (8.32), it is clear that

$$\dot{q} = \begin{bmatrix} \delta\omega \\ \delta q_e \end{bmatrix}, \quad q = \begin{bmatrix} - \\ q_e \end{bmatrix}$$

$$M = \begin{bmatrix} J + \delta J(q_0) & (H + \delta H(q_0))^T \\ H + \delta H(q_0) & M_{ee} \end{bmatrix}, \quad K = \begin{bmatrix} 0 & 0 \\ 0 & \frac{\partial f_{Te}}{\partial \theta} \end{bmatrix}$$

(8.37)

Furthermore, noting that $\theta$ is an actuation input at the elastic bodies and no actuation inputs exist at the rigid body,

$$u_r = 0, \quad u_e = \delta\theta$$

$$B_{rr} = 0, \quad B_{re} = \left[ \frac{\partial G}{\partial \theta} \right]_{q_0,\theta_0}, \quad B_{ee} = \left[ \frac{\partial f_{Te}}{\partial \theta} \right]_{q_0,\theta_0}$$

(8.38)

With the lack of actuation at the rigid body, the controllability criteria can be rewritten as follows:

$$C_r = \left[ \frac{\det(B_{re}B_{re}^T)}{\det J} \right]^{1/6}$$

$$C_e = \text{row}_{\alpha} \left\| T_\alpha^T \begin{bmatrix} B_{re} \\ B_{ee} \end{bmatrix} \right\|$$

(8.39)

All the expressions needed to perform the above-given controllability check are readily available from the simulation, with the exception of the eigenvector matrix $T$.

### 8.3.3 Controllability Simulation Results

To test the controllability of the sail with the tip vanes, the values of $C_r$ and $C_e$ are calculated for the initial state and the final states of the control simulations by exporting the appropriate matrices during the simulation procedure. The exported matrices are imported using a Matlab script, which then proceeds to solve the eigenvalue problem Equation (8.35), after which the values of $C_r$ and $C_e$ are generated via the equations introduced in the previous section. The values calculated for $C_r$ are listed in Table 8.3, and the values for $C_e$ are plotted in Figure 8.21, from the lowest to the highest frequency. First, consider Table 8.3. The rough trend is that the sail becomes more controllable when the sail is tilted than the sunward or the edge-on directions. Considering that the $\theta_x = \pi/4$ cases require torques from the tip vanes to counteract the bias disturbance, it seems likely that the resulting vane angle configurations offer greater controllability to the sail within the configurations' local vicinities.
Table 8.3: Values of $C_r$ for Varying Attitude States

<table>
<thead>
<tr>
<th>$\theta_x$</th>
<th>$\theta_y$</th>
<th>$\theta_z$</th>
<th>$C_r$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>$1.3202 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.5708</td>
<td>0</td>
<td>0</td>
<td>$1.8380 \times 10^{-4}$</td>
</tr>
<tr>
<td>0.77793</td>
<td>0</td>
<td>0</td>
<td>$3.9576 \times 10^{-4}$</td>
</tr>
<tr>
<td>1.5708</td>
<td>0</td>
<td>0.7854</td>
<td>$2.1265 \times 10^{-4}$</td>
</tr>
<tr>
<td>0.77474</td>
<td>0</td>
<td>0.7854</td>
<td>$4.9474 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

The plots given in Figure 8.21 show that the first 100 lowest frequency modes are, to one degree or another, controllable, the values varying from $10^{-3}$ to $10^{-7}$ (It is worth noting that all of the modes are shown to be controllable according to the simulation results). Unfortunately, due to the nonlinear nature of the model being used, it is hard to interpret the observed patterns in a useful manner. Nonetheless, it does show that the elastic modes of the sail are clearly controllable. Hence, it can safely be concluded that both the rigid and the elastic modes of the sail are controllable via the four two DOF tip vanes.
Figure 8.21: Values of $C_e$ for the First 100 Lowest Frequencies for Varying Attitude
Chapter 9

Conclusion

The research involved analyzing the dynamics involved in controlling a solar sail using tip vanes. There were various uncommon hurdles to overcome, each issue intertwined with one another. Several simplifying assumptions were made for each topic to perform research on them within the allotted time for the thesis and facilitate combination with each other, culminating in the construction of a simulator encompassing all of the topics covered in this thesis.

Unconstrained nonlinear flexible dynamics was introduced in Chapter 4. Its simulation was implemented to capture the behaviour of an intentionally untensioned sail such as the cord-mat square solar sail used in this research. The equations of motion were derived from the first principles, then the global FEM model generator, along with its simulator based on the numerical differentiation method, was implemented in Fortran. This simulator was used to verify the validity of the two finite element types used in this research - the hermitian cubic beam element and the natural coordinate triangular plate element, by making use of well-known large-deflections experiments and comparing the results generated by the simulator against the experiment results. Once verified, the elements were used to construct a simplified model of the cord-mat square solar sail and the model was simulated with SRP forces applied on its surface. The results were analyzed, with some simple comparisons made against static results produced by Greschik, showing that the sail model used for this research emulates Greschik’s static results and providing credence to the validity of the model. Further work could be performed on this topic however.

Other types of element can be introduced to the sail model, either as a supplement to the individual component of the sail, or as an entirely different or a replacement component. The fidelity of the model can be varied, for example by varying the number of elements representing the corner regions of the sail. It is worthwhile to note that it may be much more practical to simply adopt one of the commercial FEM model simulators, despite the effort it may take to adopt other portions of this research for the chosen simulator, simply because of the significant simulation time the currently-implemented simulator incurs. Optimization of the simulation code is such a daunting task that it is more practical to port the remainder of the research instead. The major contribution of this section to the field of solar sailing is the application of the nonlinear dynamics model to an FEM model for a solar sail, and the subsequent verification of the simulation results to a known static model.

The wrinkling dynamics model was presented in Chapter 5 and implemented into the nonlinear dynamics model via the use of a modified Miller-Hedgepeth constitutive relationship derived by the author. The modified constitutive relationship sets the appropriate principal stresses to zero when
negative strains and stresses are incurred. This constitutive relationship is represented as a single matrix with two variables, where the values of two variables denote the current state of the membrane as being taut, wrinkled or slack. This constitutive relationship was implemented as a part of the dynamics simulation from above to emulate the behaviour of a membrane being unable to resist compression. Unfortunately, the simulation was unable to process the abrupt changes to the constitutive relationship very well. Various approaches were taken to provide solution to the issue at hand, and in the end a quasi-static solution of preventing changes to the wrinkling pattern as the simulation progressed had to be taken. Even with such a solution, promising verification results were produced when the implementation was used to simulate the well-known corner-loading experiment, the simulation results being comparable to the experimental results. Modeling the sail membrane wrinkling behaviour into a flexible dynamics model for a solar sail is a feat that has not been performed previously, and is expected to be a major contribution to the field from this Chapter.

A method of determining control angles for four two-DOF tip vanes was developed and presented. Specifically, when using a total of four tip vanes with two DOF, the large number of actuation points pose an under-constrained control allocation problem. A solution capable of generating a set of control actuation angles from the desired torque in Cartesian coordinates was developed and presented in Chapter 6. The solution takes advantage of the geometry of the tip vane placement to limit the dimension of the produced torques to two per vane, estimates the range of produce-able torques as an ellipse, then defines a convex optimization based on the ellipse boundary to derive a set of torques each vane must provide. These torques are then used to solve the single-vane problem to derive the set of vane angles from the given vane torques. The sum of the vane torques produced by the generated vane angles is verified to be either the desired torque or a scaled variant of the desired torque in case it is physically impossible to generate, the verification performed using hundreds of randomly-generated desired torques. More research could be performed on the effect of realistic reflective surfaces and the deflecting booms. While the latter has been shown via simulation to be not significant enough to adversely affect rough attitude pointing, finer pointing schemes may require explicit modelling of such behaviour to counteract it effectively. It goes without saying that structurally and optically realistic surfaces will have significant effects on such an endeavour as well. The major contribution from this chapter is the algorithm as a whole. With the information provided in this chapter, any interested party should be able implement a two-DOF vane control actuation scheme.

A method of determining whether a particular node of the FEM model is within the shadow of another arbitrary component or not was developed and presented in Chapter 7. This involved projecting the shadowing component onto a plane defined by a sail element, then using a convex hull algorithm to determine if the node is within the convex hull formed by the projected vertices of the shadowing component. A simple test was devised using a randomly-generated triangular element and a randomly positioned box, shone upon by the sun from a random direction. One sample case was presented showing how the algorithm is able to determine based on the given information whether the element nodes are shaded by the cube or not. The idea was taken further to demonstrate the shading effects of the sail due to itself during certain attitudes. While the shadowing algorithm was implemented in the main simulation for the shading effects due to a spacecraft bus, it was not implemented for simulating the shading due to the sail, because of the computational challenge such an endeavour entailed. An in-depth research into the dynamical sail shading and the resulting system behaviour may be warranted for pathological cases with the back of the sail to the sun. The algorithm for determining the shadowed
area serves as the major contribution from this chapter, as well as the observed shadowing behaviour of the sail.

All of the above were combined into a composite simulation, with a simple PD controller used to simulate various pointing attitudes from a sun-facing settled state, presented in Chapter 8. The sail was able to settle to 99\% of the commanded attitude despite the simplicity of the controller used and the various sail-specific disturbances present. A number of attempts were made to mitigate the significant bias torque created by the shuttlecock effect, but the conclusion was that the PD controller combined with the control allocation scheme functioned the best. Controllability of the sail as a whole was briefly explored as well, showing that both the sail’s rigid and the elastic modes were controllable at different attitudes, though with varying degree of controllability. More research can be performed to address the omitted $\delta q$ expression from the right hand side of the linearized equations of motion. More advanced controllers can be considered for use as well to mitigate some of the undesirable behaviours observed in the simulations. This chapter serves to reinforce the validity of the results above, as well as showing that simple PD controller is sufficient to provide decent attitude control for the solar sail.

The author believes that what has been explored in each case has merit, and deserves further investigation and research to advance the understanding of solar sails as a whole. The combined result of the current research provides an understanding of the sail behaviour with the tip vane actuation scheme, and further investigation has the potential to provide a high fidelity dynamics simulation setup for sail attitude control. The simulation could be a testbed for using actuation schemes not necessarily limited to tip vanes, but other structure-dependent schemes as well, the dynamic simulation allowing identification and revision of potential issues with the proposed schemes.
Appendix A

Implementation of the FEM Model and the Dynamics Simulation

A simplified UML diagram of the Fortran simulator is given in Figure A.1. This diagram describes the system’s data structures and their flow, as well as listing the functions that act upon those data structures, with the end goal of using the data to perform the simulation. The diagram omits many of the programming and matrix indexing details that are crucial for the simulation to work, but unnecessary to describe the functionalities of the system in general. It should be noted prior to further discussions that all floating point numbers are 8 bytes long as opposed to Fortran’s standard 4 bytes long real numbers.

The system is divided into several major components. The leaf component is the Node structure, a simple structure with a Cartesian coordinate and an integer index indicating its numbering. It also has a temporary array used to determine the associated matrix indices, but is omitted on the diagram.

The Element class contains an array of Node classes as its attributes, its purpose to describe one element of the FEM model. This structure has several floating point number matrices - namely, the linear mass matrix constants $M$, $P$, $H$, $c$, and $J$, and the nonlinear terms arising from large deformation assumption, $\tau$, $\nu$, and $\Upsilon$. While it stands to reason that the matrices that are dependent upon $\rho$, the distance from the body center, are calculated and stored individually for each element, the reason for the other element matrices being stored individually on each element is because due to the displacement imposed on the cord-mat sail, many of the elements end up in skewed forms, changing these matrices from element to element. Also due to this reason, each element has non-standard orientation, hence the rotation matrix for each element is stored within Element class.
Figure A.1: UML Diagram of the Simulation Code
The reason the stiffness components of the element matrices are not stored in the Element class is because the required storage for each element differs. The child classes BeamElement and TriElement are created to compensate for this difference, namely by providing differing numbers of matrix storage. The BeamElement, meant for beams, contains $K_L$, $N_1$, and $N_2$. The TriElement, for triangular plates, contain 6 stiffness matrices for each degree of nonlinearity, totalling to 18 matrices. As well, because of the structural differences between the elements, the initialization procedure is overridden within each child class, this process marked in the UML diagram by labeling the functions Init : BeamInit and Init : TriInit to indicate that calling the Init function results in either BeamInit or TriInit function to be called in its stead, depending on the child class used.

The SailStruct class contains an array of Element classes (and subsequently, an array of Node classes) that represents a large shape that can be described entirely by one particular type of an element - for example, a beam that is composed of several beam elements of the same type. These elements are common in that they share the same shape functions $\psi$. In the code, this is $f$, a variable array of Polynom class - a class designed to describe polynomials, which will be discussed in greater detail in the following subsection. This class contains generators for $M, f_i, \frac{\partial f_i}{\partial q}$ and $D$ at an element level, as well as abstract functions for $f_T, K$ and $\frac{\partial K}{\partial q}$. The initializer Init.SailStruct only contains very basic operations, meant to be overridden by its type-specific child classes.

The BeamStruct and TriStruct classes are the child classes of SailStruct, replacing the abstract functions and overriding the initializer, in addition to several functions that assist in the initialization process, which will be explained in the subsequent subsections. These classes keep an array of BeamElement or TriElement class objects, effectively acting as type-casted objects of the gen.e array from SailStruct and allowing its functions to access child-specific data structures. In addition, shape function derivatives and their multiples are stored here, and are listed in the figure. The symbol $f$ denotes the shape function as before, and the subscripts denote the variable the differentiation is performed with respect to. For example, attribute $f_{xxf_{xy}}$ refers to the Polynom array of $\frac{\partial^2 f}{\partial x^2} \frac{\partial^2 f}{\partial x \partial y}$.

The SailStruct class is part of SailStruct.Module, which contains functions that take an array of SailStruct objects, then generate global inertial or stiffness matrices for the given set. The precise steps will be discussed later, but needless to say, it makes use of the element matrix generators defined and implemented in SailStruct and its child classes.

The actual initiation of the above-described global matrix generation steps are done by Combined Function Module, specifically the prep_sail function and its siblings. The prep_sail function is coded specifically for the cord-mat sail system, while other structures are coded as separate functions in this module, omitted in the diagram. The output of these functions is an array of SailStruct objects, all initialized and ready to generate the matrices described in Sections 4.1, 4.2, and 4.3. The actual steps taken are described in the subsequent subsections.

Apart from generation of the global matrices, the system also contains the integrator that solves the equations of motion to derive the states. This solver is provided by ODE.Module, which contains ode15s function, which is a reproduction of Matlab’s ode15s function, but only implementing functionalities useful for this research. This function takes as input four functions: $F_{ext}$ (the external force function) and $f$ (the right-hand side of the equations of motion) with prototype $ode_f$, and mass (the mass matrix function) and jac (the right-hand side Jacobian) with prototype $ode_d$, where these functions are used to generate the equations of motion and their Jacobians as needed. LUDecomp, the LU Decomposition function for a matrix, is also shown on the diagram. The LU Decomposition is a major portion of the
total simulation time, and changes made to this function effects the overall simulation speed greatly, which will be demonstrated in the subsequent subsections.

The four input functions are implemented in ODEfnc Module. Previously there were multiple functions, with each function referring to a different level of rigid body constraints and equations of motion nonlinearities - for example, one could have constrained rigid dynamics where translational dynamics are ignored altogether, or the system is to be simulated only with linear matrices and no inertial forces. However, some work has been made to combine all these into generic input functions. The functions now use a number of ‘if’ checks to avoid generating and referring to any terms associated with constrained rigid body dynamics and undesired nonlinear matrices and inertial terms, when desired.

Finally, Combined Function Module, ODE Module and ODEfnc Module are used by Main module and the main function inside. The module itself defines various structural parameters to be used by Combined Function Module, and during its call to prep_sail, passes those parameters along. When prep_sail returns the SailStruct array, it defines the array containing set time where the results of the equations of motion should be outputted, exports structural and matrix indexing information, then begins looping through all set time the results are desired, calling ode15s at every time step, providing it the SailStruct array produced prior to the start of the loop. The system post-processes and appends the result of ode15s to an output file at the end of every loop. This process repeats until the loop completes, at which point the simulation is complete.

As Fortran lacks a native graphical interface, the output files produced are then parsed via Matlab with the help of the structural and matrix indexing information exported prior to the start of the simulation. The parsed information is then used to plot the graphs that are used to describe the results of the simulation in this thesis.

The following subsections describe specific implementation details of various modules and their functions essential in the simulation, in a scope-increasing order.

A.1 Describing and Operating Upon Polynomials - Polynom Module

Before approaching any structurally-related modules, Polynom Module and Polynom class must be discussed, the UML diagram of the module and its associated classes shown in Figure A.2. Polynom class is a structure designed to hold information about a polynomial of any order. It is composed of a variable length array of Term type, a data structure meant to hold one ‘term’ of a polynomial as the name suggests.
**Term** contains a variable length integer array named **order**, and a float named **coeff**. Each integer of the **order** array describe the order of the variable associated with that particular index, while **coeff** describes the coefficient multiplying the particular term of the polynomial. This polynomial storing method is more succinctly described in Figure A.3, demonstrating how an arbitrary polynomial is stored within its data structure.

![Figure A.3: A Sample Polynomial Described by Polynom Class](image)

The module itself contains several operations that can be performed on the **Polynom** class - **remove_overlap**, **add**, **sub**, **mult**, **div**, **power**, **value**, **differentiate**, **integrate**, **save_poly** and **load_poly**. These are basic operations necessary to calculate the inertial and stiffness terms, and the uses will be demonstrated in the following subsections.

**remove_overlap**

The function **remove_overlap** is a simple function that searches for **Term** with same **order**, and combines them together. The process is as follows: the system creates a variable length array of **Term**, then begins comparing **order** of this array against that of the **order** in the input **Polynom**. If the variable length **Term** array contains no **Term** with the same order as that from the input **Polynom**, it is added to the variable length **Term** array. Otherwise, **coeff** is added to the corresponding **Term**'s **coeff**. This process continues until all **Terms** from the original input **Polynom** are searched. This $O(\log n)$ process is described in Figure A.4.

![Figure A.4: Sample Execution of remove_overlap Function](image)

**add**, **sub**

The functions **add** and **sub** are interface functions to a core function named **addsub**, which takes in two **Polynom** inputs. It creates a new **Polynom** class with a **Term** array that has a combined length of the **Term** arrays in the two input **Polynom**, then transfers the contents of the input **Polynom**'s **Term** array to
the newly-created Polynom, then calling remove_overlap to deal with combinable terms. Due to use of remove_overlap, this can also be considered an $O(\log n)$ process.

\[
\begin{align*}
  f &= 3x_2x_3 \\
  3x_2x_3 & \quad \begin{bmatrix} 3 & 0 & 1 & 1 \end{bmatrix} \\
  f &= 5x_3 \\
  5x_3 & \quad \begin{bmatrix} 5 & 0 & 0 & 1 \end{bmatrix} \\
  f &= 3x_2x_3 - 5x_3 + 2 \\
  3x_2x_3 & \quad \begin{bmatrix} 3 & 0 & 1 & 1 \end{bmatrix}
  \quad -5x_3 & \quad \begin{bmatrix} -5 & 0 & 0 & 3 \end{bmatrix}
  \quad 2 & \quad \begin{bmatrix} 2 & 0 & 0 & 0 \end{bmatrix}
\end{align*}
\]

Figure A.5: Sample Execution of add and sub Functions

In the case of add, addsub can be called directly, while for sub, a new Polynom that mirrors the second input Polynom is created and is used as the second input in place of the original Polynom. For both cases, float can be the input instead of Polynom - in which case the corresponding float input is converted to a Polynom and used in place.

mult, div, power

The function mult takes two Polynom inputs. Every Term in the first input are multiplied to the every Term in the second input. The Term multiplication procedure is simple: The coefficients are multiplied together, while the order arrays are added together. The newly-generated Term is deposited into a new Polynom. Once the multiplication procedure is completed, remove_overlap is called to combine any Term with the same number of variables. Once again, float can replace Polynom as the input. In this case however, instead of creating a new Polynom for the float input, the float input is immediately multiplied to the other Polynom input’s coeff to generate the output Polynom. This is an $O(n^2)$ process.

Only div with inputs Polynom and float in that order are supported - this is the only type of division operator needed for this system. In this case, the mult function for Polynom-float inputs is used, with the second input as one over the original divisor input. This, and the above mult function behaviour, are illustrated in Figure A.6.

The function power, while not illustrated, simply behaves as a multiple call to mult. Specifically, given an input Polynom and the integer exponent, it makes exponent minus one calls to mult, with both inputs as power’s Polynom input, with an exception case of when the exponent is zero - in this case the function simply returns 1 in Polynom object. This is an $O(n^3)$ process and very inefficient, but the exponent value is not expected to be large (at most 4), hence no optimization attempt is made to implement more optimal algorithms.
differentiate

Differentiating a polynomial is a straightforward operation, and differentiate is no different. It takes a Polynom and an integer value indicating which variable is being differentiated - for example, a value of 2 would indicate this to be a derivative of $x_2$. As described by Figure A.7, for every Term, coeff is multiplied by the corresponding order value of the variable being differentiated, and the order value is subtracted by one, all of which happens on a newly-created Polynom object - unless the coeff is zero after processing, in which case it is simply omitted. After every Term is processed, remove_overlap is called to handle any newly-created overlaps. remove_overlap is the highest order process in this function, hence this is an $O(\log n)$ process.

integrate

The integrate function behaves in a very similar manner to differentiate, in a sense that the operations performed on differentiate are reversed for integrate. Specifically, the order value of the variable being integrated is increased by one, then the corresponding Term’s coeff value is divided by the increased order value, all of these actions occurring on a new Polynom object to be outputted. Once again remove_overlap is called at the end of the process, making this an $O(\log n)$ process. Figure
A.8 illustrates the above-described process.

\[
f = x_3^1 x_2^3 + 4x_1 x_2^3 x_3^1 x_2^2
\]

\[
\begin{array}{cccc}
1 & 3 & 2 & 0 \\
4 & 1 & 0 & 2
\end{array}
\]

Divide coeff by order of \( x_2 + 1 \)

Add 1 to order of \( x_2 \)

\[
f = \frac{1}{3} x_1^3 x_2^2 + 4x_1 x_2 x_3^2
\]

\[
\begin{array}{cccc}
\frac{1}{3} & 3 & 3 & 0 \\
4 & 1 & 1 & 2
\end{array}
\]

Figure A.8: Sample Execution of \texttt{integrate} Function on \( x_2 \)

**value**

The function \texttt{value} is a simple function used to calculate a real number produced by the polynomial for a fixed value for all variables, where it takes \texttt{Polynom} and a set of \texttt{float} values \( x \) that correspond to inputs to the variables in the polynomial described by \texttt{Polynom}. For each \texttt{Term}, the function takes \texttt{order} as the exponent to the corresponding \( x \) value, multiplies them together along with the \texttt{Term}'s \texttt{coeff}, and adds this value to a pre-initialized output value. This process is repeated for every \texttt{Term}, making this an \( O(n^2) \) process. However, the number of variables used in the system can be limited to 2, hence its actual performance is closer to \( O(n) \).

**save\_poly, load\_poly**

The full element stiffness matrix generation process takes a very lengthy amount of time to perform for every simulation. Hence, a pre-calculated portion of the \texttt{Polynom} arrays are saved into a text file, which is read to reload the pre-calculated \texttt{Polynom} arrays every time the simulation is run. To perform this task, \texttt{save\_poly} and \texttt{load\_poly} are implemented. The function \texttt{save\_poly} has a \texttt{unit} input - an \texttt{integer} value identifier for an open file - and the \texttt{Polynom} object to be exported. The file is assumed to be open in an append mode prior to the function call, and \texttt{save\_poly} simply writes the number of variables, number of terms, then for every \texttt{Term}, the \texttt{order} array followed by the \texttt{coeff} value. \texttt{load\_poly} does the precise opposite - from an open file, it reads the number of variables and terms, then begins reading the \texttt{order} array and the \texttt{coeff} value to a pre-provided \texttt{Polynom} object. Both functions are of \( O(n) \).
A.2 Element Stiffness Matrix Generator Modules

With the above-defined Polynom class and given information about element nodes and orientation, the element matrix generation can be discussed, starting with the calculation of the stiffness matrix coefficients. In terms of objects within the simulation code, the generation of the element is designed such that when the global element matrices are generated, the higher-level code does not make the distinction between different types of elements. Specifically, as illustrated in Figure A.9, while BeamStruct and TriStruct stores its element information to BeamElement and TriElement arrays - classes inheriting Element, the element stiffness matrix generator functions gen_beam_Ke and gen_tri_Ke override the virtual functions gen_Ke and gen_dKe of their parent class SailStruct. In addition, SailStruct holds an array of Element, which are in fact a type-casted BeamElement or TriElement array. The result of this relationship is that during the function call gen_Ke or gen_dKe, the code can refer to BeamElement or TriElement array and hence access attributes unique to those child classes, while the calls to mass matrix and inertial forces generator refer to the type-casted version, the Element array, which doesn’t confer any access to child-specific attributes but still allows access to attributes generic to the Element class. By virtue of the overridden functions, calling gen_Ke or gen_dKe will call the appropriate element-type-specific stiffness matrix generator, while the mass matrix and the inertial forces generators share the same set of operations and hence are kept within SailStruct. Hence, generating a global equation of motion and its terms only requires knowledge of SailStruct and its functions, without referring directly to either BeamStruct or TriStruct.

Figure A.9: UML Diagram of Classes Associated with Generating Element Matrices

Given the differentiation and integration formulas from Section 4.4.2, the calculation process for the remaining Polynom arrays in BeamElement and TriStruct is simple, but time-consuming. Specifically, single dimension derivative polynomials, i.e. fx, fy and so forth, are calculated and stored in a temporary array. Then the remaining terms come naturally in a form of polynomial multiplications - for example, fxxfyy is formed as a two dimensional array of all possible multiples of the components of fxx to that
of \( f_{yy} \). These multiplied forms are then integrated and stored in the respective Polynom arrays. This coefficient generation process is performed by \texttt{gen\_beam\_coeffs} and \texttt{gen\_tri\_coeffs} functions.

Because of the displaced structure the cord-mat sail takes on as its initial shape, some of the ‘constant’ parameters of the polynomials must actually be kept as a variable. For beams, the length of the beam \( l \) is kept as a variable in the Polynom object. For triangular plates, constants that are dependent on the position of the nodes, \( a_1, a_2, a_3, b_1, b_2, b_3 \) and \( A \), are all kept as variables.

The process is straightforward, but it has an issue of consuming a lengthy amount of time (approximately 5-6 hours on a modern computer), due to the 60000+ Polynom multiplications that must be performed. To mitigate this issue, a simple exporting and importing system for the Polynom arrays listed is implemented. Specifically, when \texttt{save\_beam\_coeffs} function is called, all of the Polynom arrays from \texttt{BeamElement} are exported via repeated use of the \texttt{save\_poly} function described in the previous section to a code-designated file. This file then can be read from by using the \texttt{load\_beam\_coeffs} function, which repeatedly calls the \texttt{load\_poly} and deposits the loaded Polynom to the Polynom arrays in the same order as they were exported. The same procedures apply when the \texttt{save\_tri\_coeffs} and \texttt{load\_tri\_coeffs} functions are called. This process takes around 1 to 2 minutes to perform, a tremendous improvement in speed when compared against the calculation process for the coefficients. Having said that, because Polynom is stored in a text format, the size of the storage file is quite large - approximately 28 kB for the beam coefficients, and 312 MB for the triangular plate coefficients. However, these files are well within the limits of modern computing storage, and this is deemed a worthy sacrifice for over 100 times the speed gain it confers.

Once the Polynom arrays are either calculated or loaded from file, the actual matrices are generated on an element-by-element basis using the formulas given in Sections 4.2 and 4.3. The generic formulas for each coefficient matrix are equivalent, but the dimension and orientation of each element is allowed to vary, changing the element stiffness matrix from one element to another. At this point, since the matrix being calculated is element-specific, the known element parameters are used as an input to the \texttt{value} function to generate a floating point value matrix, which linearly combine to generate the stiffness matrices in the element objects. The multi-dimensional floating point arrays \( KL, N_1, \) and \( N_2 \) in a \texttt{BeamElement} object are generated by first taking Polynom arrays \( f_{fx}, f_{fxfx}, f_{fxfxfx}, \) and \( f_{fxfxfxfx} \), calling \texttt{value} with these arrays with \( l \) as the input, then using the resulting floating point value to fill the stiffness matrices in \texttt{BeamElement} as directed by Equations (4.72), (4.73), (4.74) and (4.75). As \( N_1 \) and \( N_2 \) are dependent on \( q_e \) and \( q_e \) is a variable that is calculated in the simulation, \( N_1 \) is defined as a three-dimensional array with its third index corresponding to the components of \( q_e \) multiplied to it, while \( N_2 \) is defined as a four-dimensional array with its third and fourth indices corresponding to the components of \( q_e \) multiplied to it.

Similarly, the floating point arrays listed as attributes of a \texttt{TriElement} object are generated as linear combinations of the results of calling \texttt{value} on the Polynom arrays of the \texttt{TriStruct} with \( a, b, \) and \( A \) as the input. These linear combinations are given by Equations (4.96), (4.97), and (4.98), with the polynomial matrix definitions of Equation (4.99). Once again, \( N_1 \) and \( N_2 \) are dependent on \( q_e \), hence the components for \( N_1 \) are all three-dimensional while those for \( N_2 \) are four-dimensional. At the end of the element stiffness matrix generation process, all of the floating point stiffness coefficient arrays in every \texttt{BeamElement} or \texttt{TriElement} objects are filled.
Appendix A. Implementation of the FEM Model and the Dynamics Simulation

A.3 Global Matrix Generation Module

A.3.1 Matrix Indexing Generation for Elements Belonging to One SailStruct

In the case of the simulation code, generation of the d_index array described in Section 4.4.4 is achieved by first assigning unique matrix indices to an array of integers in Node objects called gndi. The array consists of up to 9 indices - 3 for Cartesian displacements and 6 for angular/derivative displacements. This array is not directly accessed by the global matrix generator however. Instead, an array called d_index in Element is initialized such that it is equal to column-length of the element matrix, then filled with the contents of gndi for the nodes belonging to that element. For the example given above, this is equivalent to letting $d_{index} = [4, 6, 10, 12]$. If $M_e$ is assumed to be the element matrix and $M$ the global matrix, the following simple line is used to assign the element matrix to the global matrix:

$$M(d_{index}, d_{index}) = M(d_{index}, d_{index}) + M_e$$

A similar approach is taken for column matrices such as $f_{lc}$. In addition, $d_{index}$ is also used to pick the set of generalized coordinates in $q_e$ that are used in the nonlinear element matrix calculation. Specifically, if $q$ is the global generalized coordinates and $q_e$ the local generalized coordinates of the element in global frame, then $q_e = q(d_{index})$. The vector $q_e$ is the global frame generalized coordinates used in the previous section.

A.3.2 Handling Multiple Shapes of Multiple Types of Elements

Each shape of the overall structure is represented by the SailStruct object, and each SailStruct object consists of one type of element. There are several steps that must be taken to combine all elements in the array of SailStruct objects together into one global matrix. The first - and the simplest - step is to verify and merge the nodes generated for each piece of the structure. Specifically, when the Node array for each structure is generated, it is uniquely written onto a master array of Node - that is, any node that occupies the same location as a node already generated is assumed to be an equivalent node, and the node from the structure-specific Node array is replaced with the equivalent one from the master array.

The second step is determining which degrees of freedom should overlap and how they should overlap. For example, one can have a plate that is merely pinned to a beam instead of fixed, i.e. only Cartesian deflections are restricted, while angular deflections are free from each other. In such a case, the node can end up with one set of Cartesian displacements but multiple angular displacements of the same type associated with it.

To handle this case, first, an array of custom type containing the following information called node_merge_info is created: Node number, SailStruct object index that the original node and the node being merged belongs to, and a logical array the size of largest unique set of displacements associated with the node. The component of this logical array is each associated with a displacement, set to true if the displacement is being merged and false otherwise. The generation of this list is strictly dependent upon the structure being simulated. However, the general approach is to check each SailStruct’s Node array and compare Node association with each other. When association is found, write down the Node index number and the two SailStruct objects that the association was found between. Follow a user-specified rule for filling the logical array - for example, when one SailStruct is a beam and the
other a triangular plate, user-rule could specify them to be pinned to each other, i.e. the logical array components associated with Cartesian deflections are set to true, while the others remain false.

Once generated, this array is used to determine how to arrange the global matrix indices. Similar steps as given in the previous section are taken. First, for each node, an array of indices gndi is generated. However this time, the array is sized such that its first index goes up to 6 as before, but it now has second index that goes up to the total number of SailStruct objects in the array. Then, a nested loop traverses through every Node in every SailStruct with the following checks and corresponding actions. For SailStruct i,

- If gndi has not been written to before, initialize and assign unique indices to gndi(1..6,i),

- If gndi has been written to before but gndi(:,i) is already filled, cycle loop,

- if gndi has been written to before and gndi(:,i) is not filled, then check node_merge_info for which SailStruct the node should be merged to. Call this SailStruct index j. Then for every displacement k, check the logical array in node_merge_info to determine whether the displacement is to be merged - in which case gndi(k,i) = gndi(k,j) - or not - in which case a unique index is assigned to gndi(k,i).

Once the above nested loop completes, gndi should be filled with all the relevant information to generate d_index. A second nested loop parses through every Element in every SailStruct such that for SailStruct i, gndi(:,i) for every Node of the Element is arranged in node order to generate d_index. Once again, d_index is used to assign the contents of an element matrix to a global matrix.

### A.3.3 Nodes with Displacement Restrictions

To generate matrix_index, first, a list of displacements to be removed are created by the user - an integer array containing the nodes to be removed, and a two dimensional logical array with first index corresponding to the integer array, and the second index corresponding to the number of displacements per node. The nodes and the displacements to be removed vary from one simulation to another, hence their generation method also varies.

The above-generated lists are used as follows: first, temp_index, a temporary array the size of the global state pre-node-removal, n, is created, then numbered from 1 to n. Now, for every Node to be restricted in displacement, gndi is accessed. If the logical array states that displacement i is to be removed, then the value of temp_index at index gndi(i,j) for every j that gndi(i,j) is non-zero for is set to zero. Once every nodes to be restricted are processed in the above manner, then matrix_index is initialized with its size equal to the number of non-zero values in temp_index. All non-zero values in temp_index are assigned to matrix_index. This array then corresponds to all displacements in the global matrix that are not restricted in movement.

To actually apply the displacement restriction, after the unrestricted global matrix is generated, it is reduced in size with the following line:

\[
M_{\text{global}} = M_{\text{raw}}(\text{matrix_index}, \text{matrix_index})
\]  
\[\text{(A.1)}\]

i.e. the restricted displacements are removed from the global matrix, as desired.
A.3.4 Implementation of Sparse Matrix System

The sparse matrix structure used is the typical Compressed Sparse Column (CSC) format. This format has the following structure: \texttt{nnz}, number of non-zero components in the matrix, \texttt{nrow} and \texttt{ncol}, number of rows and columns of the matrix, \texttt{nzval}, non-zero components of the matrix in one-dimensional array form, \texttt{rowind}, row indices of the non-zero components in \texttt{nzval}, and \texttt{colptr}, a list of \texttt{nzval} indices where each column starts. This format was chosen mainly because the above-mentioned SuperLU uses CSC-formatted sparse matrices to perform its decomposition and equation solving.

The UML diagram of the association between different components that define, provides functions to, and use the sparse matrix structure, is given in Figure A.10. Within the code, the sparse matrix structure described above is defined as a structure \texttt{SLU_NC} - a remnant name during development when it was initially conceived to be but a temporary storage for the actual sparse matrix structure provided by SuperLU library. The functions that operate upon this structure are grouped together in \texttt{SuperLU\_Handle}, which contains functions that are used to convert back and forth between dense and sparse matrices (\texttt{matrix\_to\_slu}, \texttt{slu\_to\_matrix}), a function for keeping the size of the sparse matrix minimal (\texttt{slu\_simplify}), binary operator functions that allow the sparse matrix to mathematically behave in the same manner as a dense matrix (\texttt{slu\_add}, \texttt{slu\_sub}, \texttt{slu\_mult}), and solver functions that make use of the SuperLU library to provide solutions to linear equations (\texttt{slu\_decomp}, \texttt{slu\_solve}). This module is used by \texttt{SailStruct\_Module} and \texttt{ODE\_Module}, which are modified to accommodate \texttt{SLU_NC} as their global matrix structure. In addition, \texttt{Sail\_Sparse\_Module}, which contains a sparse matrix generator routine for global matrices named \texttt{gen\_global\_sparse}, also uses \texttt{SuperLU\_Handle}. This function is called upon by \texttt{Main} prior to the simulation to initialize the global matrices used by \texttt{SailStruct\_Module} and \texttt{ODE\_Module}.

![Figure A.10: UML Diagram of Classes Associated with Sparse Matrices](image)

Prior to discussing how global matrices are generated, the implementation details for the functions in \texttt{SuperLU\_Handle} and \texttt{Sail\_Sparse\_Module} are discussed.

\texttt{matrix\_to\_slu, slu\_to\_matrix}

The conversion from one matrix type to another is reasonably straight-forward. When converting from a dense matrix to sparse matrix, first the number of non-zero entries as well as the dimensions of the dense matrix are recorded in \texttt{nnz, nrow} and \texttt{ncol} respectively, which are used to establish the sizes of \texttt{nzval (nnz, rowind (nnz) and colptr (ncol+1)). Then the dense matrix is traversed through row-first, filling up nzval and rowind for every non-zero entry in it. Whenever a change in column occurs, the index of nzval at which the change occurred is noted in colptr. At the end of the loop, the last entry of
colptr is updated to be nnz+1. This last step is important in avoiding special cases for other operations. This is an $O(n^2)$ process.

When converting from sparse matrix to dense matrix, the operation is coded very simply by the virtue of Fortran’s capacity to refer to array components using index range. Specifically, the dense matrix is generated by the following pseudocode:

$$A(\text{rowind}(\text{colptr}(i):\text{colptr}(i+1)-1), i) = \text{nzval}(\text{colptr}(i):\text{colptr}(i+1)-1)$$

where $i$ is the column number. That is, each column of the dense is filled by the contents of nzval at appropriate rowind by using two adjacent values of colptr as the index range. This is an $O(n)$ process.

slu_simplify

This function simplifies the sparse matrix structure by traversing it and removing any zero-entries in it. This is performed by creating a temporary storage of nzval_temp, rowind_temp, and colptr_temp. colptr_temp is initially assigned with the values of colptr. Then the original sparse matrix’s nzval is traversed, moving every non-zero values of nzval to nzval_temp and updating rowind_temp with nzval’s corresponding rowind value. If however a zero value in nzval is detected, the value of the components of colptr starting from the current column’s index to the end of colptr are subtracted by one. This subtraction process once again takes advantage of Fortran’s index ranging to reduce the operation to one line as opposed to a loop.

Once nzval is traversed, the resulting temporary arrays are used to generate a new sparse matrix process, which is returned as the output.

slu_add, slu_sub

These functions are one of the main reasons for implementing a custom sparse matrix handler. slu_add is a function for adding two sparse matrices together (and slu_sub a caller to slu_add with negative nzval). This function has two operating modes: First is a typical sparse matrix addition, where for each column, a dense column matrix is generated using a method similar to how slu_to_matrix function generates a dense matrix. This is performed for both operands of slu_add, which are then added together to form a new dense column matrix. This column matrix is then appended to a pre-generated sparse matrix structure, calculating the component of colptr for the current column as well as removing any zeroed components of nzval in the process. This operation is an $O(n^2)$ process.

The second mode of operation is triggered when both sparse matrices have the same values on nnz, rowind and colptr. Specifically, this is a situation where both sparse matrices have non-zero values at the same matrix indices. In this special case, solution to the addition is a simple matter of adding the two sparse matrices’ nzval arrays together. This is an $O(1)$ operation - extremely fast to execute compared to the first mode of operation. This second mode of operation also does not check for zeroed values, because it is desirable for the size of nzval to remain constant so that this behaviour can be taken advantage of. This will be discussed in more detail when explaining the behaviour of gen_global_sparse.

slu_mult

The function slu_mult has several prototypes - mostly to handle real or integer value to matrix multiplication, which is performed by simply multiplying the provided scalar number to nzval of the input
sparse matrix. A vector can be multiplied as well, in which case each component of the vector is multiplied to a dense column of the sparse matrix (which is easily generated by the index-ranging method described to convert sparse matrix to dense matrix), then added to the output dense column matrix. The scalar-matrix multiplication are all $O(1)$ processes, while matrix-vector multiplication is an $O(n)$ process.

In the case of the multiplication between two sparse matrices, this is essential to performing $n$ matrix-vector multiplications. Specifically, each output column $i$ is generated by multiplying the multiplier sparse matrix by a column $i$ of the multiplicand sparse matrix in dense matrix form. This output column is then processed in the same manner as columns in matrix_to_slu, non-zero components added to nzval and its row index noted in rowind, then colptr for the particular column specified based on the current number of components in nzval. Because of its dependence on matrix-vector multiplication, this becomes an $O(n)$ process.

**slu_decomp**, **slu_solve**

The functions **slu_decomp** and **slu_solve** are interface functions to SuperLU library’s LU decomposition and linear solver. The function **slu_decomp** accepts the contents of SLU\_NC in separate forms. The values of rowind and colptr are subtracted by one, because SuperLU is a library written in C and array indices for C starts from 0 unlike Fortran, which starts from 1. These values are passed into a C interface function, which generates its own sparse matrix storage then calls an internal LU decomposition function **dgstrf** to generate L and U in its native sparse-matrix form. Instead of converting these matrices back into SLU\_NC, the decomposition matrices are kept in its native form, then its pointer is passed back as an output. **slu_decomp** also returns this pointer to the caller. This was done on purpose as the only function that makes use of the results of the LU decomposition is **slu_solve**.

The function **slu_solve** is a linear solver that provides the solution $x$ to a linear problem $Ax = b$, where $A$ is a sparse matrix and $b$ is a dense column matrix. Its input is the pointer to the native LU decomposition sparse matrices to the matrix $A$, which is generated by calling **slu_decomp**, and the dense matrix $b$. Once again, these values are passed into a C interface function, which calls its internal solver **dgstrs** with the given LU decomposition and the dense matrix $b$ and provides a solution in dense column matrix form. This solution is passed back to the function caller.

The reason for this particular design of separating LU decomposition and linear solver is because the primary user of these functions, the **ODE_Module**, often updates $b$ without updating $A$ when the linear solver is called. Instead of generating the decomposition when the solver is called, the decomposition is generated when $A$ is updated, reducing the frequency at which the decomposition function is called. This leads to significant time saving when the simulation is performed.

**gen_global_sparse**

There is a particularly glaring omission in terms of the functionalities the above sparse-matrix implementation should have, and that is ‘component-insertion’, for inserting blocks of element matrices into the global sparse matrix. However, a generic insertion of a component or a block-component into a sparse matrix is a very time-consuming and costly process, because one cannot simply use indexing values to pinpoint the location at which a particular component should be added - instead, a portion of the rowind specified by colptr has to be searched to find the correct index, and then depending on whether a component already exists or not, an additional component may need to be added in between existing
components, leading to memory shifting or transfer processes. In the worst case scenario, adding one component to the global matrix is an $O(n^2)$ process - highly costly and preferably something to avoid.

It can in fact be avoided, by taking the following steps: 1. Pre-generate a global sparse matrix with all potentially-non-zero components already accounted for, 2. Every element keeps a series of indexing information matrix that it needs to access the appropriate component of nzval, and 3. Use the global sparse matrix in its pre-generated form without removing any non-zero entries. These three steps effectively makes the global sparse matrix a static structure that takes memory in between a pure sparse matrix and a dense matrix and can process some matrix operations faster than either a dense matrix or a pure sparse matrix, at a cost of sacrificing the flexibility to add new components to the matrix. `gen_global_sparse` is a function that performs the first two steps of the above, while step 3 is performed by coding sparse matrix operations to behave in certain manner - for example, the second mode of operation for slu_add.

`gen_global_sparse` is called after the array of SailStruct and its Element matrices, as well as various rigid body degrees of freedom indices, have been fully established. It first goes through every possible set of matrix blocks and adds the total number of components up using a series of if and loop statements, as given below:

- For every SailStruct object, account for $K$ by adding (number of elements)(number of element states)$^2$ to $idxsize$.
- For every SailStruct object, add (number of elements)(number of element states) to $esize$.
- Account for $\frac{\partial K}{\partial q}$ and the identity matrix portion of $M$ by adding the total number of generalized coordinates to $idxsize$ and multiplying $idxsize$ by two.
- If $v$ is a used state, then account for $P$, mass, and their portions of the Jacobian matrix by adding $esize*9 + 3$ to $idxsize$.
  - If $\omega$ is also a used state, then account for $c$ by adding 18 to $idxsize$.
- If $\omega$ is a used state, then account for $H$, $J$ and their portions of the Jacobian matrix by adding $esize*9 + 9$ to $idxsize$.
- If $q_n$ is a used state, then account for its Jacobian terms by adding 28 to $idxsize$.
  - If $r$ is a used state, then account for its Jacobian terms by adding 24 to $idxsize$.

The above steps generate the absolute maximum size of nzval of the global sparse matrix, defined by $idxsize$. Once this size is determined, the actual index assignment can begin. The indexing process uses a structure `indexer`, which is composed of the following integer values: SailStruct object index ($s$), Element object index (within SailStruct) ($e$), element matrix row and column indices ($r$ and $c$), global matrix row and column indices ($gr$ and $gc$), value location in nzval ($vc$), and ‘Type’ of index which determines the matrix storage used to store the nzval location ($t$).

For convenience, assume that $q$, $dq$, $v$, $w$, $r$ and $qtrn$ refer to matrix index list of the corresponding state. In addition to convenience, let the size of $q$ and $dq$ be equal to those without displacement restrictions mentioned previously, but the index only be assigned to ones without such restrictions and initialize the rest as -1. The result of this setup will become clear later in this section. Also, define following parameters: $ns = \text{size of SailStruct array}$, $ne = \text{size of Element array in one SailStruct}$.
Appendix A. Implementation of the FEM Model and the Dynamics Simulation

object, \( nd \) = number of generalized coordinates in an element, and \( tnc \) = total number of generalized coordinates. The contents of \( \text{idx} \), an array of \( \text{indexer} \) the size of \( \text{idxsize} \), is then generated for the following components:

- The identity matrix for \( M_g \), \((r, c) = (gr, gc) = (q(i),q(i)), t = 9, i = [1..tnc]\),
- The identity matrix for \( \frac{\partial K_g}{\partial q} \), \((r, c) = (i, i), (gr, gc) = (q(i),dq(i)), t = 10, for i = [1..tnc]\),
- \( K_e, s = i, e = j, (r, c) = (k, l), (gr, gc) = (dq(d\_\text{index}(k)),q(d\_\text{index}(l))), t = 1, for i = [1..ns], j = [1..ne], k = [1..nd], l = [1..nd] \),
- \( dK_e, s = i, e = j, (r, c) = (k, l), (gr, gc) = (dq(d\_\text{index}(k)),dq(d\_\text{index}(l))), t = 2, for i = [1..ns], j = [1..ne], k = [1..nd], l = [1..nd] \),
- If \( v \) is a used state,
  - \( P_e, s = i, e = j, (r, c) = (k, l), (gr, gc) = (v(k),q(d\_\text{index}(l))), t = 3, for i = [1..ns], j = [1..ne], k = [1..3], l = [1..nd] \),
  - \( \frac{\partial P_e}{\partial q} \), \( s = i, e = j, (r, c) = (k, l), (gr, gc) = (v(k),dq(d\_\text{index}(l))), t = 4 and s = i, e = j, (r, c) = (l, k), (gr, gc) = (dq(d\_\text{index}(l)),v(k)), t = 5, for i = [1..ns], j = [1..ne], k = [1..3], l = [1..nd] \),
  - \( mI_3 \), \( (r, c) = (i, j), (gr, gc) = (v(i),v(j)), t = 11, for i = [1..3], j = [1..3] \),
  - If \( \omega \) is also a used state, \( c^\times, (r, c) = (i, j+3), (gr, gc) = (v(i),w(j)), t = 11 and (r, c) = (j+3, i), (gr, gc) = (w(j),v(i)), t = 11, for i = [1..3], j = [1..3] \),
- If \( \omega \) is a used state,
  - \( H_e, s = i, e = j, (r, c) = (k+3, l), (gr, gc) = (w(k),q(d\_\text{index}(l))), t = 3, for i = [1..ns], j = [1..ne], k = [1..3], l = [1..nd] \),
  - \( \frac{\partial H_e}{\partial q} \), \( s = i, e = j, (r, c) = (k+3, l), (gr, gc) = (w(k),dq(d\_\text{index}(l))), t = 4 and s = i, e = j, (r, c) = (l, k+3), (gr, gc) = (dq(d\_\text{index}(l)),w(k)), t = 5, for i = [1..ns], j = [1..ne], k = [1..3], l = [1..nd] \),
  - \( J, (r, c) = (i+3, j+3), (gr, gc) = (w(i),w(j)), t = 11, for i = [1..3], j = [1..3] \),
- If \( q_n \) is a used state,
  - Jacobian of the right-hand side of \( q_n \), \( (r, c) = (i+9, j+3), (gr, gc) = (qtrn(i),w(j)), t = 11 for i = [1..4], j = [1..3] and (r, c) = (i+9, j+9), (gr, gc) = (qtrn(i),qtrn(j)), t = 11 for i = [1..4], j = [1..4] \),
  - if \( r \) is a used state, Jacobian of the right-hand side of \( \dot{r} \), \( (r, c) = (i+6, i+6), (gr, gc) = (r(i),r(i)), t = 11 for i = [1..3], (r, c) = (i+6, j+9), (gr, gc) = (r(i),qtrn(j)), t = 11 for i = [1..3], j = [1..4] and (r, c) = (i+6, j), (gr, gc) = (r(i),v(j)), t = 11 for i = [1..3], j = [1..3] \).
During the above process, a counter is kept to determine how many entries have been made.

After the above process is complete, \( \text{idx} \) is sorted through such that the components of \( \text{idx} \) are ordered from the smallest \( \text{gc} \), then \( \text{gr} \) values to the largest, hence in a manner that \( \text{nzval} \) of the sparse matrix would be ordered. In-place version of the Quicksort algorithm is used, modified to compare both \( \text{gr} \) and \( \text{gc} \). Immediately following the sorting, \( \text{vc} \) is updated for every component of \( \text{idx} \). This update process is such that a incremental counter is kept during the traversing process, and the value of this counter is assigned to \( \text{vc} \) at every step. However, this counter is incremented only if the values of \( \text{gr} \) and \( \text{gc} \) of previous component of \( \text{idx} \) does not equal that of the current component of \( \text{idx} \). Note that this situation can, and should, happen in a multi-element system because \( \text{gr} \) and \( \text{gc} \) of \( \text{idx} \) were generated by traversing through every element’s \( \text{d-index} \) values, and shared nodes between elements would result in same indexing values within \( \text{d-index} \). This overlap is intentional, and it denotes several elements sharing a particular displacement - hence global matrix index.

Once every \( \text{vc} \) is filled, the contents of \( \text{idx} \) is used to generate \( \text{rowind} \), and \( \text{colptr} \) for a new sparse matrix. Specifically, \( \text{idx} \) is traversed, filling \( \text{rowind} \) with \( \text{gr} \) (only when \( \text{gr} \) or \( \text{gc} \) changes, as noted above), while \( \text{colptr} \) is filled with the current rowind index when a change in \( \text{gc} \) is detected. \( \text{nzval} \) is simply initialized as zeros. It should be noted that because of the way \( \text{q} \) and \( \text{dq} \) were initialized, one can get -1 for index values. These cases are ignored for the generation of the sparse matrix.

A new set of indexing matrices are created as well - specifically, a group of 5 indexing matrices named \( \text{s-index} \) are created within every \text{Element}. Each \( \text{s-index} \) is initialized for different purposes: \( \text{s-index}(1) \) and \( \text{s-index}(2) \) are \( \text{nd} \times \text{nd} \) matrices for components in \( \text{qe} \times \text{dq} \) and \( \text{dq} \times \text{dq} \) index range - \( \text{s-index}(1) \) specifies indices for \( \text{K} \) and \( \text{dK} \), while \( \text{s-index}(2) \) is used for \( \text{De} \) and \( \text{M}_{re} \). \( \text{s-index}(3) \) and \( \text{s-index}(3) \) are \( 6 \times \text{nd} \) matrices for \([v,w] \times \text{qe} \) and \([v,w] \times \text{dq} \) index range - \( \text{s-index}(3) \) holds indices for portions of the Jacobian of \( \text{f}_e \), while \( \text{s-index}(3) \) contains indices for \( \text{M}_{re} \) and portions of the Jacobian of \( \text{f}_e \). Finally, \( \text{s-index}(5) \) is a \( \text{nd} \times 6 \) matrix for \( \text{qe} \times [v,w] \) index range, providing indices for \( \text{M}_{re}^T \) and portions of the Jacobian of \( \text{f}_e \). In addition to these element-specific matrices, three global matrices are created as well: column matrices \( \text{eqq-index} \) and \( \text{eqdq-index} \) of size \( \text{tnc} \) that holds indices of the identity matrix components, and \( \text{r-index} \), which contains indices of matrix components associated with rigid-body dynamics and kinematics, i.e. \( \text{M}_{re} \) and the Jacobians of the right-hand side of \( \text{q}_a \) and \( \text{r} \).

Once the indexing matrices have been initialized, \( \text{idx} \) are traversed for the final time, this time checking \( \text{t} \). Components of \( \text{idx} \) with \( \text{t} = 1 \) through 5 are assigned to \( \text{s-index} \) of \text{SailStruct} \( \text{s} \) and \text{Element} \( \text{e} \) - i.e. the row and column specified by \( \text{r} \) and \( \text{c} \) of the matrix \( \text{s-index}(t) \) are set to \( \text{vc} \). Components with \( \text{t} = 9 \) and 10 are assigned to \( \text{eqq-index} \) and \( \text{eqdq-index} \) respectively. Components with \( \text{t} = 11 \) are assigned to \( \text{r-index} \). At the end of this loop, all indexing matrices should be filled with relevant information to refer to the correct \( \text{nzval} \) location for each component of the element matrix.

**Modifications to SailStruct and ODEFnc Module**

Generation of element matrices are still performed in dense-matrix form and returned as such, allowing the element matrix generators to remain unchanged. Similarly, global generators for column matrices such as \( \text{f}_e \) are unchanged. Global generators for \( \text{K}, \frac{\partial \text{K}}{\partial \text{q}}, \text{D}, \) and \( \frac{\partial \text{D}}{\partial \text{q}} \) are changed. In particular, the global matrix generators are provided with a pre-generated sparse matrix structure, which must be filled by the functions. To correctly find the right index for \( \text{nzval} \), the indexing process that used before \( \text{d-index} \) are replaced with that of \( \text{s-index} \), which is referred to find the correct index in \( \text{nzval} \) which corresponds to a specific set of row and column index.
Because of the static nature of the sparse matrix structure designed above, one cannot add the restricted displacements and remove them later using `matrix_index` as done so before. However, since $q$ and $dq$ were initialized with -1’s for the restricted displacements, one can easily check and exclude parts of the element matrix that do not have a place in the global sparse matrix. Unfortunately, this also means that the single-lined addition of each element matrix into the global one using $d_{index}$ cannot be performed anymore, since each component of the indexing matrix has to be checked for -1. Instead, a loop is used to iterate through each components of $s_{index}$ and only adds the component of the element matrix to the global one if its index is not -1.

Different global generators make use of different $s_{index}$. Generators for $K$ and $\frac{\partial K}{\partial q}$ use $s_{index}(1)$. Matrix $D$ generator uses $s_{index}(2)$. Matrix $M$ generator uses $s_{index}(2)$ for $M_{ee}$, $s_{index}(4)$ for $M_{re}$, $s_{index}(5)$ for $M_{re}^t$, and $r_{index}$ for $M_{rr}$. The $\frac{\partial f}{\partial q}$ generator uses all of $s_{index}$ and $r_{index}$. These different index matrix uses are specified by the generator function.

`ODEFnc_Module` is also modified in a similar manner as above in terms of indexing. $r_{index}$ in particular is used frequently within its functions to ascertain locations of kinematic terms within the global matrix. In addition, these functions must provide the global matrix generators a static pre-generated sparse matrix mentioned above, but this matrix is actually generated using `gen_global_sparse` in `main` after the `SailStruct` array generation process is completed. This pre-generated sparse matrix is then passed onto a modified `ode15s`, which passes them into the functions of `ODEFnc_Module` as appropriate.

In addition to the modification to use the modified functions of `ODEFnc_Module` and pass along few pre-initialized sparse matrices, the actual matrix calculations within `ode15s` are also done with sparse matrix operations. Notably, the solution of equations of the form $Mx = b$ with $M$ as the mass matrix is performed by initially calling `slu_decomp` for the mass matrix when it is created and updated, then calling `slu_solve` whenever a linear equation must be solved. These functions replace LAPACK’s LU decomposition and inversion steps, and are the crux of improving the ODE solver’s performance with a larger set of states.


