Dissertation presented to the Instituto Tecnológico de Aeronáutica and University of Twente as part of the Graduate Double Degree Program requirements for the Master of Science in Aeronautical Engineering and Mechanical Engineering, in the Field of Aeronautical Design, Structures, and Aerospace Systems.

Thorben Fabian Koch
UT Graduation Number: 444

COMPARISON OF CONSTRAINED OPTIMISATION METHODS FOR AEROSTRUCTURAL DESIGN

Dissertation approved in its final version by signatories below:

Prof. Dr. N. R. Secco
Advisor

Dr. ir. A. van Garrel
Co-advisor

Prof. Dr. E. Villani
Pro-Rector of Graduate Courses

Campo Montenegro
São José dos Campos, SP - Brazil
2024

UNIVERSITY OF TWENTE.

Cataloging-in Publication Data
Documentation and Information Division

Koch, Thorben Fabian
Comparison of constrained optimisation methods for aerostructural design / Thorben Fabian Koch. São José dos Campos, 2024.
88f.

Dissertation of Master of Science – Course of Aeronautical Engineering. Area of Aeronautical Design, Structures, and Aerospace Systems – Instituto Tecnológico de Aeronáutica, 2024. Advisor: Prof. Dr. N. R. Secco. Co-advisor: Dr. ir. A. van Garrel.

1. Multidisciplinary optimisation. 2. Aerostrucutral Analysis. 3. Constrained Optimisation. I. Instituto Tecnológico de Aeronáutica. II. Title.

BIBLIOGRAPHIC REFERENCE

KOCH, Thorben Fabian. **Comparison of constrained optimisation methods for aerostructural design**. 2024. 88f. Dissertation of Master of Science – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Thorben Fabian Koch
PUBLICATION TITLE: Comparison of constrained optimisation methods for aerostructural design.
PUBLICATION KIND/YEAR: Dissertation / 2024

Thorben Fabian Koch
Praça Marechal Eduardo Gomes, 50
12228-900 – São José dos Campos–SP Brasil

COMPARISON OF CONSTRAINED OPTIMISATION METHODS FOR AEROSTRUCTURAL DESIGN

Thorben Fabian Koch

Thesis Committee Composition:

Prof. Dr.	R.T.L. Ferreira	Chairman	-	ITA
Prof. Dr. ir.	C.H. Venner	Chairman	-	UT
Prof. Dr.	N. R. Secco	Advisor	-	ITA
Dr. ir.	A. van Garrel	Co-advisor	-	UT
Dr.ir.	G.T. Havinga		-	UT

ITA - UT

This work would not have been the same without the excellent guidance of my supervisors. A special thanks to Prof. Dr. Ney Rafael Secco and Dr. ir. Arne van Garrel.

Acknowledgments

I would like to express my sincere gratitude to the individuals who have played a significant role in the successful completion of my Master's thesis as part of the double degree program between the University of Twente (UT) and the Instituto Tecnológico de Aeronáutica (ITA).

First and foremost, I extend my heartfelt thanks to my supervisor, Prof. Dr. Ney Rafael Secco from ITA. His unwavering support, insightful guidance, and constant motivation were instrumental in shaping this thesis. Prof. Dr. Ney Rafael Secco not only provided valuable feedback but also generously dedicated his time to assist me whenever I encountered challenges. His expertise and commitment have been invaluable throughout this journey.

I am equally grateful to my second supervisor, Dr. ir. Arne van Garrel from UT, for his valuable input and guidance. Dr. ir. van Garrel's dedication to my thesis and his willingness to provide assistance and feedback have been greatly appreciated. His expertise and insights have enriched the quality of this work.

I would also like to extend my appreciation to Prof. Dr. ir. Venner, the Chair of the Engineering Fluid Dynamics Group at UT. Prof. Dr. Venner's instrumental role in the double degree program has been a driving force behind my academic journey. His valuable feedback, advice, and support have been instrumental in shaping the direction of my research.

As a non-native English speaker, the text in this Master's thesis was reviewed and checked for grammatical accuracy using AI language tools (e.g. Overleaf AI assistance, Grammarly).

This thesis would not have been possible without these individuals' collective guidance and encouragement. I am truly grateful for their contributions to my academic growth and the successful completion of my Master's thesis.

"Nothing at all takes place in the universe
in which some rule of maximum or minimum does not appear."
— Leonhard Euler

Resumo

A demanda por aeronaves altamente eficientes e ecologicamente corretas é prioritária na busca pela aviação sustentável. Esse imperativo destaca as complexidades do acoplamento aeroestrutural, ressaltando a necessidade de Otimização de Projeto Multidisciplinar (Multidisciplinary Design Optimisation - MDO) e técnicas de otimização baseadas em gradiente. Um ponto central da otimização aeroestrutural é o complexo problema de gerenciar várias restrições de desigualdade, como tensões estruturais, que os métodos tradicionais de otimização muitas vezes têm dificuldade de tratar de forma eficiente. Métodos atuais, como o uso de funções *max*, a função Kreisselmeier-Steinhauser (KS), e abordagens de conformidade, enfrentam desafios no gerenciamento eficiente das restrições à medida que o número de funções de restrição aumenta. Esta dissertação explora o potencial do método do Lagrangiano Aumentado (Augmented Lagrangian method - ALM) para contornar essas limitações, oferecendo uma abordagem mais eficiente para gerenciar as restrições de desigualdade. Essa abordagem é aplicada em um modelo de análise aeroestrutural que emprega um modelo de elementos finitos para análise estrutural e uma adaptação da da linha sustentadora de Prandtl para a aerodinâmica. Central a essa investigação é a comparação da abordagem ALM com uma abordagem de otimização que não agrega restrições, e com uma abordagem agregadora que emprega uma função KS para consolidar essas restrições. Com foco no modelo de aeronave Helios Pathfinder Plus, o estudo destaca a eficácia do ALM, especialmente no contexto do cálculo de derivadas pelo método adjunto. Isso contrasta com as complexidades e as demandas computacionais da abordagem de otimização que não agrega restrições, especialmente ao lidar com modelos estruturais complexos. A abordagem agregadora, embora eficiente em termos de tempo, não atinge o mesmo nível de precisão e eficiência computacional que o ALM. Em otimizações de peso estrutural altamente discretizadas, o ALM supera significativamente as abordagens concorrentes, atingindo velocidades quase duas vezes mais rápidas que a abordagem agregadora e dez vezes mais rápidas que a abordagem não agregadora. Além disso, o ALM demonstrou de forma consistente um comportamento de convergência no contexto de otimizações aeroestruturais altamente complexas, onde as abordagens não agregadoras e as agregadoras tiveram dificuldades para alcançar a convergência. A pesquisa ressalta a importância do ajuste fino dos parâmetros do ALM para equilibrar a exploração do cenário de otimização e a busca de soluções viáveis. Configurações de parâmetros excessivamente conservadoras podem levar o algoritmo ALM a soluções abaixo do ideal, priorizando a viabilidade rápida em detrimento da exploração aprofundada.

Abstract

In the pursuit of sustainable aviation, the demand for highly efficient and environmentally friendly aircraft takes precedence. This imperative brings to light the complexities of aerostructural coupling, underscoring the necessity for Multidisciplinary Design Optimisation (MDO) and gradient-based optimisation techniques.

Central to aerostructural optimisation is the intricate problem of managing numerous inequality constraints, such as structural stresses, which traditional optimisation methods often struggle to handle efficiently. The current methods, such as using *max* functions, the Kreisselmeier-Steinhauser (KS) function, and compliance approaches, encounter challenges in efficiently managing the constraints as the number of constraint functions increases. This thesis explores the potential of the Augmented Lagrangian method (ALM) to transcend these limitations by offering a more effective approach to managing inequality constraints.

This thesis explores aerostructural optimisation through a comprehensive study that employs a finite element model for structural analysis and an adaptation of Prandtl's lifting line theory for aerodynamics. Central to this investigation is the comparison of the ALM approach with both a non-aggregating optimisation approach, which directly handles all inequality constraints, and an aggregating approach that employs a KS function to consolidate these constraints.

Focusing on a model of the Helios Pathfinder Plus aircraft, the study highlights the effectiveness of the ALM, particularly in the context of derivative calculation via the adjoint method. This contrasts with the complexities and computational demands associated with the non-aggregating optimisation approach, especially in handling complex structural models. The aggregating approach, while time-efficient, does not achieve the same level of accuracy and computational efficiency as the ALM.

In highly discretised structural weight optimisations, the ALM significantly outperforms competing approaches, achieving speeds nearly twice as fast as the aggregating approach and ten times faster than the non-aggregating approach. Moreover, in the context of highly complex aerostructural optimisations, where both the non-aggregating and aggregating approaches struggled to achieve convergence, the ALM consistently demonstrated robust and reliable convergence behavior. The research underscores the significance the fine-tuning of ALM parameters to balance exploration of the optimisation landscape and the pursuit of feasible solutions. Overly conservative parameter settings may lead the ALM algorithm to suboptimal solutions by prioritising rapid feasibility over thorough exploration.

List of Figures

List of Tables

List of Abbreviations and Acronyms

AD	Automatic differentiation
ALM	Augmented Lagrangian method
BFGS	Broyden–Fletcher–Goldfarb–Shanno
CAD	Computer-aided design
CDF	Computational fluid dynamics
CSM	Computational structural mechanics
DoF	Degree of Freedom
EU	European Union
ETS	Emissions Trading System
FEM	Finite element method
ITA	Instituto Tecnológico de Aeronáutica (eng. Aeronautic Institute of Technology)
KS	Kreisselmeier–Steinhauser
L-BFGS	Limited memory BFGS algorithm
L-BFGS-B	Limited memory BFGS algorithm including bound constraints
LLT	Lifting-line theory
NASA	National Aeronautics and Space Administration
MDO	Multidisciplinary optimisation
QASTRO	Quick AeroSTructural Optimisation
SLSQP	Sequential Least SQuares Programming
SLSQP KS	SLSQP using constraint aggregation with KS function
SQP	Sequential Quadratic Programming
UT	University of Twente

List of symbols

Symbol	Unit	Description
$\boldsymbol{A}$		Matrix in the linear least squares problem in the SLSQP optimisation method
$\mathcal{A}$		Augmented Lagrangian function
A	$[m^2]$	Cross-sectional area of the beam-truss element
$\boldsymbol{a}$		Column vector of scalar functions, consisting of the objective function f and the residual functions $\boldsymbol{r}$
A_i	$[m^2]$	Cross-sectional area of frame element i
A_i	$[m^2]$	Cross-sectional area of airfoil section i
α	$[rad]$	Angle of attack
$\boldsymbol{\alpha}_0$	$[rad]$	Vector of twist angles of each horseshoe of the wing
α_i	$[rad]$	Local angle of attack at wing section i
α_k		Scalar value determining the step size in the optimisation iteration
$\alpha_{\max}$		Maximum step size
$\alpha_{\min}$		Minimum step size
$\boldsymbol{A'}$		Modified matrix in the linear least squares problem after eliminating equality constraints in the SLSQP optimisation method
$\boldsymbol{B}$		Approximated Hessian matrix or actual Hessian matrix

Symbol	Unit	Description
$\boldsymbol{b}$		Arbitrary vector used in the calculation of the residual function $\boldsymbol{r_\Psi}$ for solving the linear system in automatic differentiation
$\boldsymbol{b}$		Vector in the linear least squares problem in the SLSQP optimisation method
b	$[m]$	Span of both wings
β		Coefficient in the curvature condition, equal to 0.9
$\boldsymbol{b'}$		Modified vector in the linear least squares problem after eliminating equality constraints in the SLSQP optimisation method
$\boldsymbol{C}$		Matrix of gradients of inequality constraints in the SLSQP optimisation method
C		Direction cosine
c		Coefficient in the Armijo condition, equal to 10^{-4}
c	$[m]$	Chord length
c_{0i}		Zero lift coefficient of section i
c_{mi}		Moment coefficient of section i
c_{Battery}	$[J/kg]$	Specific energy of the battery
C_D		Three-dimensional drag coefficient
c_d		Two-dimensional drag coefficient
$c_{d\,\min}$		Minimum drag coefficient
C_L		Three-dimensional lift coefficient
c_l		Two-dimensional lift coefficient
c_{l0}		Zero lift coefficient
$c_{l\alpha}$	$[1/rad]$	Lift slope
$c_{l\alpha i}$		Lift slope of section i

Symbol	Unit	Description
c_{li}		Local lift coefficient of wing section i
c_{m0}		Zero moment coefficient
c_{m0i}		Zero moment coefficient of section i
$c_{m\alpha}$	$[1/rad]$	Moment curve slope
$c_{m\alpha i}$		Moment curve slope of section i
$\boldsymbol{C'}$		Augmented matrix of inequality constraints in the simplified linear least squares problem in the SLSQP optimisation method
$c_{\text{Solar panel}}$	$[W/m^2]$	Produced power per square meter of the solar panels
D	$[N]$	Drag force
$\boldsymbol{D}$		Diagonal scaling matrix used in optimisation algorithms to scale the search direction
$\boldsymbol{d}$		Vector in the inequality constraints in the SLSQP optimisation method
$\boldsymbol{d}$	$[m]$	Displacement and rotation vector of all elements
Δ		Step bound. Maximum allowable step size in optimisation algorithm
δ_i	$[rad]$	Flap angle at wing section i
$d\boldsymbol{F}_i$	$[N]$	Lift force acting on the wing section i due to a vortex element
$d\boldsymbol{l}$	$[m]$	Length of the vortex bound
$\boldsymbol{d}_{\text{local}}$	$[m]$	Local displacement
$\boldsymbol{D}$		Diagonal matrix in the LDL-decomposition of matrix $\boldsymbol{B}$
dM_i	$[Nm]$	Acting moment of each lifting surface
$d\boldsymbol{M}_i$	$[Nm]$	Moment vector of each lifting surface in three-dimensional space

Symbol	Unit	Description
$\boldsymbol{d'}$		Augmented vector in the inequality constraints in the simplified linear least squares problem in the SLSQP optimisation method
E	$[N/m^2]$	Elastic modulus of the material
ϵ		Small threshold value set by the user to terminate an algorithm
$\boldsymbol{\eta}_k$		Difference of the gradients of the Lagrangian function between successive iterations
$\boldsymbol{F}$	$[N], [Nm]$	Vector containing the acting forces and moments
f		Objective function in the optimisation problem
$\boldsymbol{f}^k_{\text{local}}$	$[N], [Nm]$	Local force vector representing the forces and moments acting on element k in a structure
F_s	$[N]$	Combined shear force calculated as $\sqrt{F_y^2 + F_z^2}$
F_x	$[N]$	Axial force acting on a node in the x-direction
F_y	$[N]$	Shear force acting on a node in the y-direction
F_z	$[N]$	Shear force acting on a node in the z-direction
$\boldsymbol{G}$		Matrix of gradients of equality constraints in the SLSQP optimisation method
G	$[N/m^2]$	Shear modulus of the material
$\tilde{\boldsymbol{g}}$		Modified vector function of inequality constraints in the ALM algorithm
$\boldsymbol{g}$		Inequality constraints in the optimisation problem
$\boldsymbol{\Gamma}$	$[m^2/s]$	Vector containing all vortex strengths
Γ	$[m^2/s]$	Vortex strength or circulation strength of a vortex horseshoe
γ		Scalar factor for adjusting the penalty parameter ρ_k in the ALM algorithm chosen by the user

Symbol	Unit	Description
$\boldsymbol{\gamma}_k$		Determines the Lagrange multiplier update at iteration k of SQP method
$\bar{g}_{KS}$		KS aggregation function. Used in the optimisation approach with constraint aggregation
$\boldsymbol{h}$		Equality constraints in the optimisation problem
$\tilde{\boldsymbol{h}}$		Modified vector function of equality constraints in the ALM algorithm
$\boldsymbol{I}$		Identity matrix
i		Index variable
I_y	$[m^4]$	Second moment of area of the beam-truss element around the y-axis
I_z	$[m^4]$	Second moment of area of the beam-truss element around the z-axis
J	$[m^4]$	Second moment of area of the beam-truss element around the x-axis
j		Index variable
$\boldsymbol{J}_{\boldsymbol{a}_{\boldsymbol{x}}}$		Jacobian matrix of the vector $\boldsymbol{a}$ with respect to $\boldsymbol{x}$
$\boldsymbol{J}_{\boldsymbol{r}}$		Jacobian matrix with respect to the residual functions $\boldsymbol{r}$
$\boldsymbol{K}$	$[N/m]$	Global stiffness matrix obtained by assembling the stiffness matrices of all elements
k		Index variable, also used as iteration index
$\boldsymbol{k}_{\text{global}}$	$[N/m]$	Stiffness matrix of one element in the global coordinate system
$\boldsymbol{k}_{\text{local}}$	$[N/m]$	Stiffness matrix of one element
$\mathcal{L}$		Lagrangian function

Symbol	Unit	Description
L	$[N]$	Lift force
L	$[m]$	Length of the beam-truss element
$\boldsymbol{l}$		Lower bounds for the design variables $\boldsymbol{x}$ in the optimisation problem
$\boldsymbol{\lambda}$		Equality Lagrange multipliers
LF		Load factor, equal to three
$\boldsymbol{L}$		Lower triangular matrix in the LDL-decomposition of matrix $\boldsymbol{B}$
$\boldsymbol{l_q}$		Lower bound of the search direction $\boldsymbol{q}$ in the sub-optimisation problem
$\boldsymbol{m}(\boldsymbol{p})$		Structural stress margin vector
M_B	$[Nm]$	Combined moment calculated as $\sqrt{M_y^2 + M_z^2}$
m_{Battery}	$[kg]$	Battery mass
m_k		Quadratic model approximating the augmented Lagrangian function at $\bar{\boldsymbol{x}}_k$ in the ALM algorithm
$\boldsymbol{\mu}$		Inequality Lagrange multipliers
μ	$[kg/(m\,s)]$	Dynamic viscosity of air
m_{w0}	$[kg]$	Initial mass of the tubular wing spar
M_x	$[Nm]$	In-plane moment (torsion moment) acting on a node
M_y	$[Nm]$	Out-of-plane bending moment acting on a node in the y-direction
M_z	$[Nm]$	Out-of-plane bending moment acting on a node in the z-direction
n_{beams}		Number of beam-truss elements in the finite element model

Symbol	Unit	Description
n_{cond}		Number of considered flight conditions in an optimisation problem
n_{DoF}		Number of degrees of freedom
n_f		Number of objective functions in the optimisation problem
n_g		Number of inequality constraints in the optimisation problem
n_Γ		Number of vortex horseshoes, also called panels
n_h		Number of equality constraints in the optimisation problem
n_{mc}		Number of previous iterations used for Hessian matrix approximation in the L-BFGS-B algorithm
n_p		Number of state variables, equal to the number of residual functions n_r
n_r		Number of residual functions
ν		Passion's ratio of material of beam-truss elements
n_x		Number of design variables in the optimisation problem
Ω		Feasible set defined by the constraints of the optimisation problem
$\boldsymbol{p}$		State variables of the system, including the displacement vector $\boldsymbol{d}$ and the vector of vortex strengths $\boldsymbol{\Gamma}$
$P_{\text{ratio }0}$		Power ratio of the initial design
$\boldsymbol{\Psi}_{\mathcal{A}}$		Adjoint variables associated with the augmented Lagrangian function $\mathcal{A}$
$\boldsymbol{\Psi}_f$		Adjoint variables associated with the objective function f
$\boldsymbol{q}$		Search direction vector

Symbol	Unit	Description
r		Scalar parameter, chosen by the user, determining the condition for increasing the penalty parameter ρ_k in the ALM algorithm
r	$[m]$	Structural dimension representing the radius of the beam-truss element
$\boldsymbol{r}$	$[N]$	Vector containing all residual functions
$\boldsymbol{R}$		Direction cosines matrix
$\boldsymbol{r}_1$	$[m]$	Vector from the vortex tip to an arbitrary point in space
$\boldsymbol{r}_2$	$[m]$	Vector from another vortex tip to the same arbitrary point in space
Re		Reynolds number
ρ_0		Initial penalty factor in ALM algorithm chosen by the user
ρ_{Air}	$[kg/m^3]$	Air density
ρ_k		Penalty parameter in iteration k in the ALM algorithm
ρ_{KS}		Parameter in the KS function
ρ_{mat}	$[kg/m^3]$	Material density of beam-truss elements
$\boldsymbol{r}_{\boldsymbol{\Psi}}$		Residual function for determining adjoint variables
σ_{bot}	$[N/m^2]$	Stress at the bottom of the beam-truss element caused by out-of-plane bending moment M_B
σ_{top}	$[N/m^2]$	Stress at the top of the beam-truss element caused by out-of-plane bending moment M_B
σ_x	$[N/m^2]$	Uniform stress in beam-truss element caused by axial force F_x
σ_Y	$[N/m^2]$	Yield strength of the material
$\boldsymbol{s}_k$		Difference between successive design variable vectors in optimisation iterations

Symbol	Unit	Description
S_{ref}	$[m^2]$	Reference wing area used in the calculation of aerodynamic coefficients C_L and C_D
t		Step size parameter in the projection of the steepest descent direction and in determining the generalised Cauchy point in the L-BFGS-B algorithm
$\boldsymbol{t}$	$[m]$	Vector of the beam-truss element thicknesses
t	$[m], [mm]$	Structural dimension representing a dimension of one beam-truss element
$\boldsymbol{T}$		Transformation matrix between the local and global coordinate systems
$\boldsymbol{t}_0$	$[m]$	Vector of the initial beam-truss element thicknesses
τ_s	$[N/m^2]$	Shear stress in beam-truss caused by shear force F_s at a node
τ_T	$[N/m^2]$	Shear stress in beam-truss due to the torsion moment M_x
t^c		Optimal step size determined by the generalised Cauchy point in the L-BFGS-B algorithm
$t_{\text{Endurance 0}}$	$[s]$	Endurance of the initial design
θ_k		Scalar parameter in the definition of $\boldsymbol{y}_k$ in the BFGS formula
$\boldsymbol{u}$		Upper bounds for the design variables $\boldsymbol{x}$ in the optimisation problem
$\boldsymbol{u}_{ai}$		Unit vector aligned with the free stream of wing section i
$\boldsymbol{u}_{si}$		Unit vector perpendicular to the airfoil section i
$\boldsymbol{u}_{\infty}$		Unit vector in the direction of the free stream velocity
$\boldsymbol{u}_{ni}$		Unit vector normal to the wing section i, indicating the direction perpendicular to the wing's surface at that section. In this direction acts the lift force.

Symbol	Unit	Description
$\boldsymbol{u_q}$		Upper bound of the search direction $\boldsymbol{q}$ in the sub-optimisation problem
$\boldsymbol{V}$	$[m/s]$	Velocity vector
v		Shear factor used in the calculation of shear stress τ_s
$\bar{\boldsymbol{v}}$		Reversed seed vector used in the context of automatic differentiation
$\bar{v}_{\mathcal{A}}$		Seed value for the augmented Lagrangian function $\mathcal{A}$ in the context of automatic differentiation
$\bar{v}_f$		Seed value for the objective function f in the context of automatic differentiation
$\boldsymbol{V}_i$	$[m/s]$	Local velocity vector on the quarter-chord line of wing section i
$\boldsymbol{V}_\infty$	$[m/s]$	Free stream velocity
$V_{\text{ind}\,j}$		Induced velocity of horseshoe j on control point i
$\boldsymbol{v}_{j,i}$		Masure of the influence of horseshoe j on the bound segment of horseshoe i
V_k		Metric for adjusting the penalty parameter ρ_k in the ALM algorithm
$V_{k\,g}$		Metric for the inequality constraints in the ALM algorithm, representing the norm of the maximum value between $\boldsymbol{g}(\boldsymbol{x}_k)$ and $-\frac{\boldsymbol{\mu}_k}{\rho_k}$
$V_{k\,h}$		Metric for the equality constraints in the ALM algorithm, representing the norm of $\boldsymbol{h}(\boldsymbol{x}_k)$
$\bar{\boldsymbol{v}}_r$		Seed vectors for the residual functions $\boldsymbol{r}$ in the context of automatic differentiation
W_0	$[N]$	Initial weight of the aircraft
W	$[N]$	Weight of the aircraft
$w_{\mathcal{A},i}$		Weight of the augmented Lagrangian function for condition i in an optimisation problem

Symbol	Unit	Description
$\boldsymbol{x}$		Vector of design variables
$\bar{\boldsymbol{x}}_k$		Design variables at outer ALM iteration k in the L-BFGS-B algorithm
$\bar{\boldsymbol{x}}^*$		Solution of the ALM sub-problem, solved using the L-BFGS algorithm
$\boldsymbol{x}^*$		Solution of optimisation problem
$\boldsymbol{x}_{\text{Beam}}$	$[m]$	Location of the beam-truss element along the x-axis, i.e., along the chord length
$\bar{\boldsymbol{x}}^c$		Cauchy point in the L-BFGS-B algorithm, representing the first local minimiser of the one-dimensional function $m_k(\boldsymbol{x}(t))$ along the projection of the steepest descent direction
$\boldsymbol{X}_{c\,i}$	$[m]$	Midpoint of horseshoe i, which lies on the bound vortex of the horseshoe
$\boldsymbol{x}_{\text{complete}}$		Complete set of design variables
$\boldsymbol{x}'$		Canonical basis vector of the local coordinate system in the x'-axis
$\boldsymbol{x}_{\text{reduced}}$		Reduced set of design variables
$\boldsymbol{X}_{\text{ref}}$		Reference point for calculating the total moment of the aircraft
$\boldsymbol{y}_k$		Vector used in BFGS formula for Hessian approximation, similar to $\boldsymbol{s}_k$
$\boldsymbol{y}'$		Canonical basis vector of the local coordinate system in the y'-axis
$\boldsymbol{z}'$		Canonical basis vector of the local coordinate system in the z'-axis
$\boldsymbol{z}_{\text{user}}$		Given z'-axis in the global coordinate system by the user

Contents

1 Introduction

As one of the fastest-growing sources of greenhouse gas emissions, the aviation industry stands at a critical juncture in the global effort to combat climate change. The European Union (EU), recognising the significant environmental impact of aviation, has embarked on ambitious initiatives to reduce these emissions. With the revision of the EU Emissions Trading System (ETS) Directive and the European Green Deal, the EU aims for a net reduction of greenhouse gas emissions by at least 55% by 2030 and a 90% reduction in transport emissions by 2050, compared to 1990 levels (European Commission, 2023). These goals underscore the urgent need for innovative aviation design and operation solutions. Despite improvements in fuel efficiency, with a 24% reduction in fuel burned per passenger from 2005 to 2017, the sustained growth in air traffic continues to outpace environmental gains. In 2017, aviation accounted for 3.8% of the EU's total CO_2 emissions and 13.9% of transport emissions, making it the second largest source of transport greenhouse gases after road transport. The stark reality is further illustrated by the fact that a round-trip flight from Lisbon to New York generates emissions comparable to an average EU citizen's annual home heating (European Commission, 2023). This alarming scenario, coupled with the non-CO_2 climate impacts of aviation, such as the release of nitrogen oxides and soot particles at high altitudes (European Commission, 2023), brings into sharp focus the critical role of aerostructural optimisation. This thesis is situated within the urgent need for environmentally sustainable innovations in aviation, aligning with the EU's climate goals.

A prime example of the advancements driven by this environmental mandate is the groundbreaking design of the Boeing 787 Dreamliner. One of the Dreamliner's most significant innovations is its more flexible wing design. This flexibility results from substantial mass reduction achieved through new structural concepts, including using materials with increased strain allowance. These advancements have been incorporated into the latest generation of Boeing aircraft, such as the 787 and the 777-8/9 seriesc(WUNDERLICH *et al.*, 2022). The Dreamliner's wings demonstrate an impressive feat of engineering; under ultimate load conditions, the wingtip displacement can reach up to almost 8 metres, approximately 27% of the half wingspan (LOH, 2022), see figure 1.1. This remarkable flexibility is not just a structural advancement but also a strategic aerodynamic adaptation. The wings' static aeroelastic effects allow for passive load alleviation, further reducing wing mass and, as a result, improving the aircraft's overall fuel efficiency and reducing its environmental impact.

FIGURE 1.1 – Boeing 787 Dreamliner wing displacement under ultimate load conditions. Illustration from Dodt (2011) (edited).

The design of the Boeing 787 Dreamliner offers a clear illustration of what we call aerostructural optimisation. Aerostructural optimisation is an engineering approach that synergistically considers both aerodynamic and structural aspects of aircraft design to enhance performance and efficiency. Engineers can significantly improve the aircraft's efficiency and environmental footprint by optimising the structural integrity and aerodynamic properties simultaneously. This integrated approach to design is crucial in modern aviation, where each improvement can have substantial implications for both performance and sustainability.

In modern aeronautical engineering, Multidisciplinary Optimisation (MDO) plays a pivotal role. This concept extends beyond the conventional, compartmentalised approach to design, advocating for a holistic view that encompasses multiple engineering disciplines simultaneously. MDO is not just about optimising individual components in isolation; it is about understanding and exploiting the intricate interplay between different aspects of aircraft design to achieve the best overall outcome.

At the heart of MDO lies the principle that structural integrity and aerodynamic performance are fundamentally interconnected. The structural design of an aircraft not only determines its weight and resilience to various stressors but also influences its aerodynamic efficiency. For instance, the weight and strength of the structure directly impact fuel consumption and flight stability. On the other hand, aerodynamic considerations such as drag reduction and lift optimisation play a critical role in determining how the aircraft interacts with the air around it. In MDO, these aspects are not treated as separate challenges but as complementary facets of a single, unified design problem (MARTINS; NING, 2021).

This integrative approach utilises advanced computational models and algorithms to address these multidisciplinary factors simultaneously. By doing so, MDO facilitates a more nuanced and effective optimisation process. The optimisation of one aspect, such as the structural design for weight reduction, is done with a keen awareness of how it affects aerodynamic performance and vice versa. The result is a design that balances and harmonises the various demands of weight, strength, efficiency, and performance.

In essence, MDO represents a shift in engineering philosophy – from isolated optimisation within specific domains to a comprehensive, integrated approach. This shift is crucial in pursuing more efficient, sustainable, and high-performing aircraft designs. As we progress in this thesis, the focus will shift to applying these MDO principles to the Helios aircraft, exploring how this integrated approach can enhance its design for optimal environmental sustainability and efficiency.

1.1 Objective

In the realm of aircraft design, the quest for optimal performance has always been a delicate balance between aerodynamics and structural integrity. Aircraft design optimisation involves balancing aerodynamics and structural integrity. Ludwig Prandtl's work on elliptical lift distribution, which minimises drag through the lifting-line theory (LLT), has historically informed design principles (ANDERSON, 2017). However, Prandtl also acknowledged the efficiency of non-elliptical lift distributions when considering structural constraints, as discussed in his paper "Über Tragflügel kleinsten induzierten Widerstandes" (On Wings of Minimum Induced Drag) (PRANDTL, 1933), highlighting the complexity of aerostructural optimisation where solely focusing on aerodynamics might not yield the most efficient designs.

Modern aerostructural optimisation has transitioned from Prandtl's analytical methods to numerical, gradient-based techniques propelled by computational advances and sophisticated algorithms for derivative computation. This evolution allows for detailed optimisations that

integrate both aerodynamic and structural considerations, addressing challenges previously unfeasible due to computational limitations. This shift underscores the intricate understanding required in contemporary aerostructural optimisation.

In gradient-based optimisation methodologies, a primary obstacle is the extensive number of inequality constraints derived from the structural model (MARTINS *et al.*, 2005). These constraints represent a critical factor in maintaining structural integrity under various stress and load conditions. However, addressing them within the optimisation framework is not straightforward. The need to compute gradients for each constraint adds layers of computational complexity, making the process both time-consuming and resource-intensive. This often forms a bottleneck in optimisation efforts, as the computational demands can escalate rapidly with the increase in the number of constraints.

To address this, a common strategy in aerostructural optimisation has been the adoption of aggregation functions, notably the Kreisselmeier–Steinhauser (KS) function (POON; MARTINS, 2007). The KS function effectively consolidates numerous inequality constraints into a singular aggregate constraint. This approach simplifies the optimisation problem, making it more computationally manageable by reducing the number of derivative calculations required. However, this simplification comes at a cost. The use of the KS function inherently involves an approximation of all inequality constraints, which can impede the optimiser from converging to the best possible solution. While it eases the computational burden, this aggregation can lead to suboptimal or imprecise solutions in representing the true constraint landscape.

This limitation forms the crux of this thesis and motivates the exploration of a third optimisation approach: the augmented Lagrangian method (ALM). Unlike the KS function, ALM incorporates constraints directly into its formulation. It does this by augmenting the Lagrangian function with penalty terms related to constraint violations, thereby maintaining a closer adherence to the original problem structure. This approach promises a more accurate and potentially more effective means of handling the numerous constraints inherent in aerostructural optimisation. By comparing these three optimisation approaches—standard gradient-based, KS function-based, and ALM — this thesis aims to provide insights into their respective efficiencies and effectiveness, particularly in the context of the complex, constraint-rich landscape of aerostructural design.

In circling back to the outset of our discussion—the imperative for environmentally sustainable aviation—we find a perfect embodiment of this vision in the Helios Aircraft. Helios, a marvel of modern engineering, stands as a testament to the potential for genuinely sustainable flight. Powered solely by solar energy, it eschews the need for fossil fuels, presenting a groundbreaking step towards an eco-friendly aviation future (NASA, 2002). The Helios Aircraft not only symbolises the aspiration for greener skies but also serves as the practical foundation for applying the optimisation methods discussed in this thesis.

In this Master's thesis, the Helios Aircraft will be the focal point of our optimisation studies. By applying the three optimisation approaches to this solar-powered aircraft, we aim to explore how these methodologies can enhance an already environmentally conscious design. This endeavour is more than a technical exercise; it is a foray into the realm of possibility, demonstrating that optimisation techniques can significantly contribute to the development of aircraft that align with our environmental ambitions. The optimisation of the Helios Aircraft serves as a microcosm of the larger goal: to show that sustainable aviation is not just a concept but an achievable reality. Through this thesis, we aim to contribute to this vital transition, advancing the cause of environmentally responsible aviation and underscoring the crucial role of aerostructural optimisation in achieving this goal.

The box below summarises the objective of this thesis and gives a step-by-step description of how we want to achieve our objective.

Thesis' objective

This thesis aims to evaluate the efficacy of three distinct optimisation methodologies within the context of aerostructural optimisation, characterised by its dense constraint environment. These methodologies include a conventional gradient-based approach, an aggregation technique, and an optimisation strategy based on ALM. The goal is to conduct a comparative analysis of these approaches to identify the most effective strategy.

Step-by-Step description to achieve objective:

1. **Methodological overview**: Comprehensive description of each optimisation approach, detailing the theoretical foundations.
2. **ALM advantage illustration**: Illustration of the benefits of the ALM-based approach regarding the gradient evaluation efficiency.
3. **Testbed definition**: Definition of structural and aerostructural testbeds that will serve as the basis for comparing the optimisation approaches.
4. **Performance analysis and comparison**: Assess and juxtapose the performance of the optimisation strategies with respect to:
   (a) Optimisation duration
   (b) Optimisation outcomes
5. **Conclusion & recommendations**: Summary of conclusions and recommendations for further research regarding constraint-rich aerostructural optimisations.

1.2 Organisation of the thesis

The thesis is organised into chapters to facilitate structured topic exploration. The Theoretical Background chapter in the appendix A provides essential information for understanding the thesis. The reader is encouraged to start with this chapter, especially if unfamiliar with the relevant concepts. However, it is essential to note that readers can revisit this chapter later if they encounter unclear or unfamiliar topics during their reading, as it serves as a valuable reference for clarification. Chapter 2 offers a comprehensive literature review, delving into the current challenges in aerostructural optimisation. In Chapter 3, the methodology of the three optimisation approaches is explained, along with the definition of the optimisation problem for the Helios aircraft. This problem is divided into structural and aerostructural optimisation components. Chapter 4 presents the numerical implementation and methodology for each optimisation approach. It demonstrates the behaviour of these approaches in optimising the Helios aircraft. Chapter 5 discusses the results of these optimisations, highlighting the performance and trade-offs observed. Lastly, chapter 6 summarises the conclusions drawn from the findings, especially regarding the comparison of the three optimisation approaches. It also suggests avenues for future research, for the optimisation program used and for each optimisation approach.

2 Literature Review

This chapter embarks on a detailed literature review, crucial for situating the research in the context of MDO, an essential aspect of aerostructural optimisation. It seeks to elucidate the significance of MDO in realising optimised designs and examines methodologies for gradient computation with an emphasis on the adjoint method. Additionally, it explores constraint aggregation techniques, focusing on the KS function and the ALM. These components are vital in grasping the complexities of aerostructural optimisation. The review aims to seamlessly connect theoretical concepts with their practical applications, setting a robust foundation for the thesis.

The transition in aircraft design methodologies towards computational techniques marks a critical juncture in the field. As Martins *et al.* (2005) highlights, the shift from traditional methods, based on simplified theories and wind tunnel tests, to computational fluid dynamics (CFD) and computational structural mechanics (CSM) reflects a growing confidence in computational approaches. This evolution represents the industry's move towards more advanced, precise, and efficient design strategies, paving the way for the subsequent discussion on the necessity of MDO.

2.1 Multidisciplinary Design Optimisation

MDO is a pivotal concept in modern engineering systems, integrating multiple disciplines to depict a unified system. As Martins and Ning (2021) explain, MDO utilises coupled models and solvers for an integrated analysis of multidisciplinary systems, surpassing traditional single-discipline models. This holistic approach incorporates physical, economic, and human factors, more accurately reflecting real-world system interactions and performance.

The advantages of MDO are significant, including enhanced system performance, reduced design time, cost-effectiveness, and minimised uncertainty. Unlike traditional sequential component optimisation, MDO optimises various components simultaneously, yielding more effective system designs. As Sobieszczanski-Sobieski and Haftka (1997) note, engineering systems are typically modelled as assemblages of software modules representing different aspects. In MDO, these modules are cohesively optimised, allowing for optimal trade-offs and overcoming the limitations of sequential optimisation, which often results in suboptimal solutions and constraint violations.

A specific application of MDO, aerostructural optimisation, focuses on the interplay between structural and aerodynamic aspects. This integration is critical, as Martins *et al.* (2005) describe, with aerodynamic flow solutions and structural responses being interdependent.

Mono-disciplinary approaches in aircraft design have limitations, as Peter and Dwight (2010) illustrate. Optimising a wing solely for aerodynamics, ignoring structural deformations, often leads to suboptimal results. MDO addresses this by simultaneously considering aerodynamic and structural objectives, as Martins and Kennedy (2021) emphasise. Aerodynamic optimisation can yield impractical designs, such as excessively large wings, without integrating structural factors.

MDO inherently balances these requirements, ensuring realistic and feasible designs.

The superiority of MDO over sequential optimisation in aerostructural design is well-established. A study by Grossman *et al.* (1988) demonstrates MDO's effectiveness in optimising a glider's wing, coupling aerodynamic and structural models. This integrated approach produces superior outcomes with a 11% lower weight than sequential methodologies, a point reinforced by Sobieszczanski-Sobieski and Haftka (1997).

However, MDO's implementation is complex, particularly compared to single-discipline optimisation. As Martins *et al.* (2005) point out, developing high-fidelity MDO methods is challenging due to the inter-disciplinary coupling and heterogeneity of design problems. Additionally, as Sobieszczanski-Sobieski and Haftka (1997) note, the complexity and dimensionality of multidisciplinary systems often hinder the extension of single-discipline sensitivity analysis principles. To manage this complexity, MDO models are often simplified, coupling design space search with simpler approximations of objective functions and constraints rather than direct multidisciplinary analysis (GIUNTA *et al.*, 1994); (TOROPOV; MARKINE, 1996). This simplification is essential for practical and effective MDO application in aerostructural design optimisation.

2.2 Gradient evaluation

Gradient-based optimisation, a cornerstone in aerostructural design, hinges on the accurate and efficient computation of gradients. The initial forays into this domain, as explored by Hicks et al. in the mid-1970s (HICKS *et al.*, 1974),(VANDERPLAATS; HICKS, 1976), (HICKS; HENNE, 1977), involved the use of finite differences for gradient information in airfoil and wing design, primarily with lower fidelity calculations (NEWMAN *et al.*, 1998). However, while straightforward, this approach suffered from significant drawbacks, notably in terms of computational cost and accuracy issues, as the cost is proportional to the number of design variables and plagued by the step-size dilemma (MARTINS; KENNEDY, 2021).

Recognising the inefficiencies of finite differences, the field shifted towards more time-efficient methods for gradient computation. The need for rapid and accurate gradient evaluation became increasingly apparent, as highlighted by Poon and Martins (2007), especially when traditional methods proved time-consuming. This shift marked a significant advancement in optimisation techniques within aerostructural design.

The advent of direct and adjoint methods revolutionised gradient computation in large systems. Section A.1.7 explains in detail both methods. Originating from optimal control theory (e.g. (BRYSON; HO, 1975)) and structural design (MARTINS; KENNEDY, 2021), these methods were adapted for aerostructural applications, offering more efficient alternatives to finite difference methods. The pioneering work of Jameson (1988) introduced the use of the adjoint method in CFD, drawing from control theory, to efficiently compute gradients for aerodynamic optimisation, as highlighted by his formulation of the adjoint equation (JAMESON, 1988). The direct method, which is particularly effective when there are few design variables and many functions of interest, was first extended to multidisciplinary problems by Sobieszczanski-Sobieski (1990). On the other hand, the adjoint method, suitable for scenarios with many design variables and few functions of interest, has been widely applied in structural optimisation problems and aerostructural design (POON; MARTINS, 2007). The choice between these methods largely depends on the specific requirements of the optimisation problem, particularly the number of design variables and functions of interest (AKGÜN *et al.*, 2001).

The adjoint method, in particular, has gained prominence for its ability to compute sensitivities for an arbitrary number of design variables at a cost similar to that of a single function evaluation, making it highly efficient for large-scale optimisations (MARTINS *et al.*, 2005). Various

studies have demonstrated its application in aerostructural design, including the work of Mader *et al.* (2008), who used the adjoint method to determine derivatives in coupled aerostructural models.

In contrast, the direct method has proven to be highly efficient and easier to implement in scenarios with fewer design variables, as exemplified by Lund (1994) in the context of Finite element method (FEM) programs. This method's efficiency and ease of implementation make it a viable option for specific optimisation scenarios.

As the discussion of gradient computation methods concludes, it naturally leads to the subsequent section on aggregation functions. Notably, the adjoint method's efficiency is tempered by its increasing computational cost as the number of design functions grows, underscoring the need for function aggregation to maintain computational feasibility in large-scale optimisations. This transition is crucial, particularly in addressing the challenge of managing a large number of constraint functions arising from inequality stress constraints in aerostructural optimisations. The choice of gradient computation method, whether direct or adjoint, plays a pivotal role in addressing these challenges, as will be further explored in the following section on aggregation functions.

2.3 Aggregation functions

In the realm of aerostructural optimisation, particularly when employing FEM, a prominent challenge arises from the multitude of structural constraints. As Martins *et al.* (2005) elucidates, while the aerodynamic aspect of such optimisation problems typically involves a single objective function and a few constraints, the structural part is characterised by an extensive array of constraints. This disparity in the nature of constraints between aerodynamics and structures significantly complicates the optimisation process.

The direct implication of this multitude of constraints is the enormity of the optimisation problem. As described by Lambe *et al.* (2017), a practical approach to enforcing failure constraints in a continuum structure would ideally require their application throughout the material domain, leading to an impractically high number of constraints. This problem is compounded in high-fidelity structural models, where element-wise enforcement of failure constraints can lead to thousands or even millions of constraints, posing substantial challenges to optimisation efforts.

Traditional alternative solutions, such as focusing solely on maximum stress or compliance constraints, have proven to be insufficient. The work of Ghazlane *et al.* (2011), which uses the maximum stress from a FEM model on aerostructural optimisation, exemplifies this limitation. While such approaches may offer some utility, as Kennedy and Hicken (2015) point out, they fall short in large-scale design problems when gradient-based optimisation methods are employed. Using the maximum as stress as a constraint yields a discontinuous design space, which makes the gradient computation challenging. Additionally, compliance constraints, as highlighted by Arreckx *et al.* (2016), often result in designs with stress concentrations that could lead to failure, rendering them impractical from an engineering design perspective.

A more effective solution to this challenge is the KS function, a widely used method for constraint aggregation in gradient-based optimisation. The section A.1.5 in the Theoretical Background chapter explained the theory of the KS function. The KS function has found extensive application in diverse areas ranging from aerodynamic shape optimisation (POON; MARTINS, 2007), chemical process design (ROONEY; BIEGLER, 2002), aircraft design (STETTNER; SCHRAGE, 1992; AKGÜN *et al.*, 2001; MARTINS *et al.*, 2004), dynamic systems (COUCEIRO *et al.*, 2022), and topology optimisation (XIA; SHI, 2016), demonstrating its versatility and effectiveness in reducing the number of constraints from a potentially overwhelming number to a manageable few.

This reduction not only streamlines the optimisation process but also makes adjoint methods more feasible and efficient.

However, the implementation of the KS function has its challenges. A critical aspect of using this function is the choice of the parameter ρ_{KS}. As Lambe *et al.* (2017) have noted, the value of ρ_{KS} significantly impacts the quality of the solution and the duration of the optimisation process. A low ρ_{KS} value can lead to overly conservative solutions, while a high value can result in longer optimisation times and issues related to ill-conditioning.

Addressing this dilemma, Qin and Nguyen (1994) proposed an approach that can define the aggregation function's error beforehand. Furthermore, an adaptive approach to setting ρ_{KS} has been proposed by Poon and Martins (2007) as a different effective strategy. This approach, as discussed by Zhang *et al.* (2019), involves adjusting ρ_{KS} during the optimisation process, balancing the accuracy of the optimal solution against the potential for ill-conditioning. Such an approach has been applied successfully in various optimisation scenarios, including aeroservoelastic design and high-fidelity aerostructural optimisation.

Despite the benefits offered by the KS function in managing a large number of constraints, it is essential to note that this method does not recreate the actual feasible design space as precisely as the original constraints would. As Lambe *et al.* (2017) point out, the final design determined by the optimiser using the KS function will vary depending on the aggregation scheme employed. This aspect underscores the need for careful consideration in applying the KS function, particularly in complex aerostructural optimisations.

Concluding this discussion, it becomes apparent that while the KS function and other aggregation methods offer significant advantages in managing numerous constraints, they may not always provide the most optimal solution. This realisation paves the way for exploring alternative methods, such as the ALM, which may offer better solutions in scenarios where traditional inequality aggregation falls short. The work of Poon and Martins (2007) provides a compelling example, demonstrating that the solution corresponding to the maximum of constraints formulation was infeasible. The KS function formulation converged to a minimum, but the resulting structure was not fully stressed, leading to a final weight that was 5.8% higher than the reference result obtained without constraint aggregation. The work of Poon and Martins (2007) is crucial because, in this Master's Thesis, we will perform a similar comparison. We want to compare an optimisation approach without constraint aggregation, one with constraint aggregation and another where we use an ALM approach. The ALM method is further reviewed in the next section.

2.4 Augmented Lagrangian method

In aerostructural optimisation, constraint aggregation effectively manages numerous structural failure constraints but may not yield optimal results due to overestimations, such as in final structural mass (ARRECKX *et al.*, 2016). This approach limits optimisation to a subset of possible solutions, highlighting the need for alternatives like the ALM that encompass the full range of constraints for more accurate outcomes in complex designs.

ALM, initially termed as the "method of multipliers," was a significant development in the field of optimisation, pioneered by Hestenes (1969), Powell (1969), Haarhoff and Buys (1970). These foundational works laid the groundwork for transforming constrained optimisation problems into a series of more manageable unconstrained sub-problems. While the intricacies of these methods are beyond the scope of this review, comprehensive analyses and reviews can be found in the works of Rockafellar (1973a), Rockafellar (1973b), Fletcher (1974), Pierre and Lowe (1975), and Powell (1978a).

Initially focused on equality constraints, ALM faced a significant challenge when extended to inequality constraints, a common scenario in complex aerostructural optimisation problems. One of the earliest approaches to handling inequality constraints was the concept of the 'active set.' This method involved identifying constraints that were 'active' (i.e., constraints that directly impact the solution) at the optimal point. However, a significant disadvantage of this method was the initial unknown status of the active set, requiring iterative guessing and verification, which proved to be computationally impractical and inefficient.

Some researchers, like Armand and Omheni (2017b), proposed an alternative to replace inequality constraints with logarithmic barriers. While conceptually straightforward, this approach did not adequately address the complexities of inequality constraints in optimisation. Essentially, replacing inequality constraints with barrier functions echoed the limitations of constraint aggregation methods like the KS function, where the optimisation is constrained to a subset of the feasible region, potentially leading to suboptimal solutions.

Another proposed solution was the introduction of slack variables to transform inequality constraints into equality constraints (GILL *et al.*, 1986). This transformation involved adding a slack variable for each inequality constraint, effectively converting the inequality into an equality constraint by ensuring that the sum of the original constraint function and the slack variable equalled zero. While this method successfully converted inequality constraints into a more manageable form, it significantly increased the problem's dimensionality by adding as many slack variables as there were inequality constraints.

Rockafellar (1973a) presented an approach which does not suffer from the issue of increased dimensionality. His Powell-Hestenes-Rockafellar augmented Lagrangian function managed inequality constraints without explicitly solving for slack variables, thus avoiding increasing problem dimensionality. Fletcher (1975) later revisited the Powell-Hestenes-Rockafellar augmented Lagrangian function and proposed a mathematically identical augmented Lagrangian function but easier to program. Furthermore, Fletcher (1975) discussed practical strategies for guaranteeing convergence. Further discussions on the nuances of these methods, including local convergence rates, effects of approximate unconstrained minimisation, and adjustments of Lagrange multipliers, have been contributed to by Buys (1972), Polyak and Tretyakov (1974), Polak and Sangiovanni-Vincentelli (1979), and Bertsekas (1982). These works provide deeper insights into the evolution and application of multiplier methods in optimisation.

A crucial aspect of the ALM lies in the careful initial selection and adjustment of the penalty factor, ρ. As emphasised by Arora *et al.* (1991), the convergence rate of ALM heavily depends on this factor. Although a larger value of ρ accelerates convergence, it must be increased judiciously to avoid sub-problem ill-conditioning, highlighting the importance of its initial setting. Echoing this, Bertsekas (1999), supported by recommendations in the literature, including Bertsekas (1996) and Arora *et al.* (1991), advise starting with a moderate ρ and progressively increasing it. This strategy aims to balance convergence efficiency against the risk of ill-conditioning. Armand and Omheni (2017a) even presented an approach where the penalty factor can both increase and decrease during the optimisation. This intricate balance required for the penalty factor in ALM, contrasting with the single parameter choice in methods like the KS function, demonstrates the method's inherent complexity and the careful consideration needed for its effective application in (aero-)structural optimisations.

In the landscape of optimisation methods, while the ALM provides a robust approach for handling large-scale optimisation problems, it is often compared unfavourably with Sequential Quadratic Programming (SQP) methods, particularly regarding local convergence properties. As noted by Arreckx *et al.* (2016), one of the main disadvantages of ALM is that it typically does not exhibit the favourable local convergence properties characteristic of SQP methods. This has led to some preference for SQP over ALM in specific optimisation scenarios.

The efficiency of ALM in large-scale problems has been debated among researchers. While some have found ALM less efficient for large-scale optimisation problems (COUCEIRO *et al.*, 2022), others, such as Birgin and Martínez (2019), have highlighted a renewed interest in ALM due to its ability to solve such problems effectively. This dichotomy in viewpoints underscores the situational effectiveness of ALM.

Interestingly, Birgin and Martínez (2019) emphasises that the strength of ALM lies not just in its efficiency but in its ability to solve problems other algorithms fail to solve. The heterogeneous nature of constrained optimisation suggests that the choice between ALM and other methods should not be solely based on subtle efficiency criteria but also on their suitability for specific problem types.

In this context, the combination of ALM with the adjoint method presents a significant advantage, particularly in scenarios involving a large number of design variables and constraint functions. When paired with ALM, the adjoint method can outperform other optimisation methods in such scenarios due to its efficient handling of multiple constraints and variables when combined with the adjoint method. However, despite its potential, relatively little research has explored this combination, particularly in aerostructural optimisation. This thesis aims to fill this gap by investigating the performance of ALM in conjunction with the adjoint method, particularly in situations characterised by many design variables and constraints, where traditional methods may not be as effective.

As we delve deeper into the applications of the ALM, a spectrum of approaches and their limitations becomes apparent. An early example by Qin and Nguyen (1994) demonstrates the use of an aggregation function in conjunction with the ALM for optimisation. However, this approach mirrors the disadvantages encountered when using the KS function, as the ALM's potential in handling constraints within its formulation is not fully realised. Larsson and Rönnqvist (1995) presented an approach to structural optimisation in applying the ALM. While effectively managing numerous inequality constraints, Larsson and Rönnqvist (1995) incorporated the residual equations, determining the state variables, as equality constraints within the ALM. This approach, treating residual equations directly as constraints, showcases one of the varied strategies in optimisation. Subsequent advancements in the field have suggested the adjoint method as a more streamlined alternative for handling residual equations. By implicitly addressing these equations, the adjoint method simplifies the optimisation process, avoiding the complexities of including residual equations as part of the explicit constraint set.

The work of Larsson and Rönnqvist (1995) represents an early exploration within the realm of optimisation, particularly in handling inequality constraints using the ALM. However, it is noteworthy that the combination of ALM with the adjoint method, a potentially more efficient approach, especially in (aero-)structural optimisation, appears to be underutilised in the field. This observation underscores a significant research opportunity, pointing towards a gap in the current body of knowledge where the synergistic benefits of ALM and the adjoint method have not been fully explored or harnessed in these complex optimisation scenarios.

In contrast, Arreckx *et al.* (2016) adopted a more refined approach to addressing the challenges of aerostructural optimisation. Instead of treating the residual equation as a direct constraint, Arreckx *et al.* (2016) formulated the optimisation problem in a reduced space. This method, while closely aligned with the adjoint method, is not identical to it. The reduced-space formulation implicitly integrates the residual function into the optimisation process, a strategy that offers a more efficient alternative to directly handling the residual equation as a constraint, as highlighted in Biros and Ghattas (2005).

While effective in its own right, the method of Arreckx *et al.* (2016) does not extend to multidisciplinary aerostructural designs and does not employ the adjoint method. Utilising the adjoint method in this context would offer an even more significant simplification of the

process by requiring only the gradient information of a single function—the gradient of the augmented Lagrangian function. The paper's authors anticipate that the matrix-free augmented Lagrangian optimisation method would be particularly advantageous for coupled aerodynamic and structural optimisation. This outlook is a driving force behind this thesis, motivating the development of a matrix-free augmented Lagrangian algorithm within a program called QASTRO. QASTRO stands for Quick AeroSTRuctural Optimisation and is being developed at the Aeronautic Institute of Technology (ITA). QASTRO was already successfully utilised by Cruz (2020) and Almeida (2021) in the context of and box wing and propeller slipstream optimisations, demonstrating its effectiveness and underscoring the framework's adaptability and robustness in the context of aerostructural optimisations.

Integrating the adjoint method in the ALM algorithm is anticipated to be significantly beneficial, particularly in aerostructural optimisations, as it obviates the need for computing the Jacobian of the constraint functions and further simplifies the optimisation process.

The evolution of ALM's application reached a notable development in the work of Conn *et al.* (1997), whom the author believed to be among the first to combine ALM with the adjoint method. In the context of circuit optimisation, Conn *et al.* (1997) compared the time to get the sensitivities of the direct method, the adjoint method, and the ALM combined with the adjoint method. Conn *et al.* (1997) showed that the ALM combined with the adjoint method leads to short sensitivity computation times and highlights the efficiency of this combination. Importantly, Conn *et al.* (1997) found that when dealing with a large number of design variables and functions of interest, combining ALM with the adjoint method reduces computational time. However, the study of Conn *et al.* (1997) focused only on the time to compute sensitivities and did not compare optimisation results or apply the method to MDO. More recent applications by Maute and Allen (2004) and Senhora *et al.* (2020) in topology optimisation demonstrate an understanding of the advantages of combining ALM with the adjoint method. While not explicitly MDO, these studies show a reduction in the computational burden by needing to compute fewer adjoint variables. However, they do not fully explore the method's potential in more complex scenarios, such as aerostructural optimisation.

These examples, especially when viewed in the context of the belief of Arreckx *et al.* (2016) in the significant potential of ALM for complex aerostructural optimisations, highlight a substantial research gap. This thesis aims to address this gap by applying ALM combined with the adjoint method in MDO, particularly in aerostructural optimisation. Unlike previous studies, this thesis will compare optimisation results, not just computation times, to fully assess the effectiveness of this combined approach in a more complex and multidisciplinary setting.

3 Methodology

The Methodology chapter plays a crucial role in this Master's Thesis. This chapter provides a detailed explanation of how the research was carried out. Furthermore, the chapter explains the research design, methods, and techniques used for the (aero-) structural optimisation. This chapter aims to give the reader a detailed view of the research process and demonstrate the study's validity.

As mentioned in the Literature Review, the author enhanced a program called QASTRO with two additional optimisation approaches. Readers unfamiliar with QASTRO are encouraged to keep revisiting section B.1 in the Appendix while reading the following four sections. This section together with the following four gives a bigger picture of the implementation of QASTRO.

QASTRO is an optimisation program that couples a finite element model with an aerodynamic model based on LLT. First, the structural model, then the aerodynamic model, is explained in the following two sections. The third section explains how those two models are combined into an aerostructural model, to perform aerostructual analyses. The aerostructural analysis is computed in Fortran. The Fortran codes are wrapped in a Python code, which is the user interface of QASTRO. The aerostructural model can be used to perform aerostructural optimisations. All optimisations are performed on the Python interface, where we utilise the SciPy library (VIRTANEN *et al.*, 2020). The chapter continues by describing the methodology of the three optimisation approaches, which will be compared in the Result chapter. Ultimately, the chapter defines the structural and aerostructural optimisation problems.

3.1 Structural model

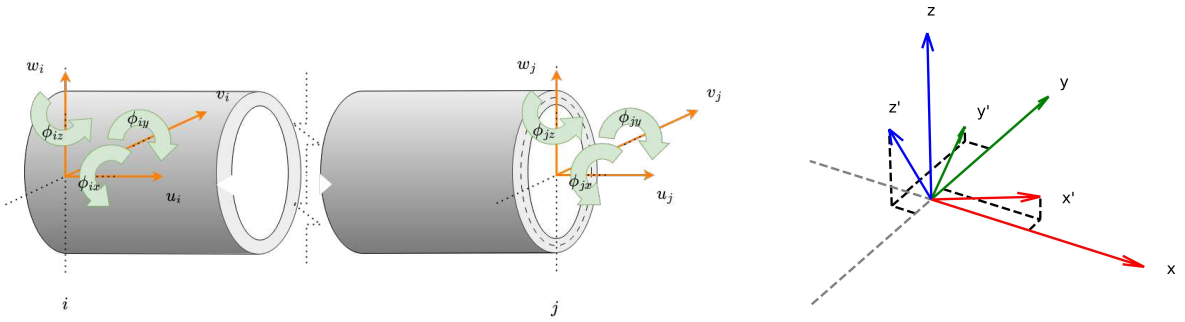
In QASTRO is a finite element model implemented to calculate the displacement of the aeroplane structure due to aerodynamic forces. QASTRO uses beam-truss elements with a tubular cross-section to model the aircraft structure. Every beam-truss element has six degrees of freedom (DoFs) at every node. Figure 3.1a shows the DoFs of the tubular beam-truss element in a sketch. The stiffness matrix of one element is $\boldsymbol{k}_{\text{local}}$ and is a function of the elastic modulus E, the shear modulus G, the cross-sectional area of the beam-truss element A and the length of the element L, the second moment of area of the beam-truss element around the x-,y-, and z-axis, denoted by J, I_y, and I_z, respectively (LOGAN, 2012).

The local displacement vector $\boldsymbol{d}_{\text{local}}$ of one element is

$$\boldsymbol{d}_{\text{local}}^T = \Big(\underbrace{u_i \quad v_i \quad w_i \quad \phi_{ix} \quad \phi_{iy} \quad \phi_{iz}}_{i} \quad \underbrace{u_j \quad v_j \quad w_j \quad \phi_{jx} \quad \phi_{jy} \quad \phi_{jz}}_{j}\Big), \tag{3.1}$$

where the first six elements are the displacements and rotations at node i, and the last six are the displacements and rotations at node j.

We need to establish a relationship between the local coordinate system of the element and the global coordinate system. Figure 3.1b illustrates the local coordinate system of an element



(a) DoF of finite element.

(b) Local coordinate system of beam-truss element (indicated by an apostrophe) and global coordinate system.

FIGURE 3.1 – DoF and illustration of coordinate systems of FEM model.

in the global coordinate system. The apostrophe symbol represents the local coordinates. The finite element is aligned with the $\boldsymbol{x}'$ axis of the local coordinate system, as shown in figure 3.2. For each element, the user must specify the z-axis of the element in the global coordinate system. This information allows us to compute the transformation matrix $\boldsymbol{T}$ between the local and global coordinate systems. The transformation matrix is defined as

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R}_{3\times3} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{R}_{3\times3} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R}_{3\times3} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{R}_{3\times3} \end{bmatrix}, \tag{3.2}$$

where $\boldsymbol{R}$ is called the direction cosines matrix. The rows of the matrix $\boldsymbol{R}$ are the direction cosine concerning the x'-axis, y'-axis and z'-axis, respectively. Thus, the rows of $\boldsymbol{R}$ are the canonical basis vectors $\boldsymbol{x}'$, $\boldsymbol{y}'$, and $\boldsymbol{z}'$ of the local coordinate system. The direction cosine is indicated by the letter C. The subscript of C indicates the two axes of the direction cosine. For example, $C_{zy'} = \cos\theta_{zy'}$ is the direction cosine of the global z-axis and the local y'-axis. With this notation we can write $\boldsymbol{R}$ as

$$\boldsymbol{R} = \begin{bmatrix} C_{xx'} & C_{yx'} & C_{zx'} \\ C_{xy'} & C_{yy'} & C_{zy'} \\ C_{xz'} & C_{yz'} & C_{zz'} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}' \\ \boldsymbol{y}' \\ \boldsymbol{z}' \end{bmatrix}. \tag{3.3}$$

Please note that $C_{yx'}$ and $C_{xy'}$ are not necessarily equal. The beam-truss element is along the local x'-axis orientated. Thus, the direction cosines of the x'-axis are

$$C_{xx'} = \cos\theta_{xx'} = \frac{x_j - x_i}{L}, \quad C_{yx'} = \cos\theta_{yx'} = \frac{y_j - y_i}{L}, \quad C_{zx'} = \cos\theta_{zx'} = \frac{z_j - z_i}{L}, \tag{3.4}$$

where x, y and z are the coordinates of nodes i and j of the beam-truss element in the global coordinate system, which the user needs to specify. We can determine the canonical basis vector $\boldsymbol{y}'$ in the global coordinate system by the cross product of $\boldsymbol{z}'$, and $\boldsymbol{x}'$ (LOGAN, 2012).

$$\boldsymbol{y}' = \boldsymbol{z}' \times \boldsymbol{x}' \tag{3.5}$$

$\boldsymbol{z}'$ will be provided by the user. Please note that this requires that the user gives $\boldsymbol{z}'$ that is orthogonal to $\boldsymbol{x}'$. If this is not the case, QASTRO calculates $\boldsymbol{z}'$ by

$$\boldsymbol{z}' = (\boldsymbol{z}_{\text{user}} - \text{proj}_{\boldsymbol{x}'}\, \boldsymbol{z}_{\text{user}})/\|\boldsymbol{z}_{\text{user}}\| \tag{3.6}$$

where $\boldsymbol{z}_{\text{user}}$ is the $\boldsymbol{z}'$ -axis in the global coordinate system given by the user. By this, QASTRO

ensures that $\boldsymbol{z}'$ is orthogonal to $\boldsymbol{x}'$. We can repeat this procedure for every element to calculate the canonical basis vectors in the global coordinate system for every element. The canonical basis vectors give us the matrix $\boldsymbol{R}$, which then gives us the transformation matrix $\boldsymbol{T}$ for every element. Once $\boldsymbol{T}$ is calculated, we can transform the local stiffness matrix $\boldsymbol{k}_{\text{local}}$ in the stiffness matrix of one element in the global coordinate system by

$$\boldsymbol{k}_{\text{global}} = \boldsymbol{T}^T \boldsymbol{k}_{\text{local}} \boldsymbol{T}, \tag{3.7}$$

where the superscript T indicates the transpose (LOGAN, 2012). Next, QASTRO assembles the global stiffness matrices of all elements according to their connectivity and applies the boundary conditions. Section B.2 illustrates how QASTRO applies the boundary conditions. After applying the boundary conditions, we are left to solve the linear system

$$\boldsymbol{K}\boldsymbol{d} = \boldsymbol{F}, \tag{3.8}$$

for $\boldsymbol{d}$ where $\boldsymbol{d}$ is the displacement vector of all elements, the vector $\boldsymbol{F}$ contains the acting forces and moments on the aeroplane structure, $\boldsymbol{K}$ is the stiffness matrix of the structure. We can solve for the displacement vector $\boldsymbol{d}$ by defining a residual function $\boldsymbol{r}$

$$\boldsymbol{r}(\boldsymbol{d}) = \boldsymbol{K}\boldsymbol{d} - \boldsymbol{F} \tag{3.9}$$

and find $\boldsymbol{d}$ such that $\boldsymbol{r}(\boldsymbol{d}) = \boldsymbol{0}$. We try to find $\boldsymbol{d}$ such that $\boldsymbol{r}(\boldsymbol{d}) = \boldsymbol{0}$ by defining the optimisation problem

$$\min_{\boldsymbol{d}} \quad \|\boldsymbol{r}(\boldsymbol{d})\|. \tag{3.10}$$

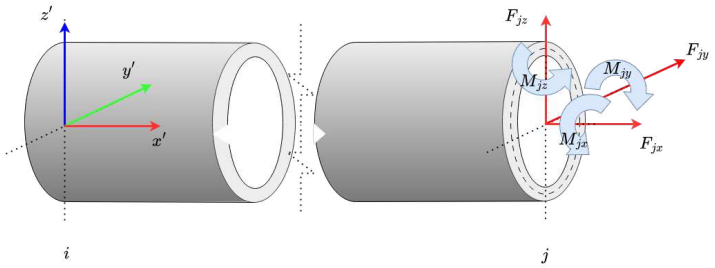


FIGURE 3.2 – Local coordinate system of beam-truss element on node i. Forces and moments acting on the element on node j in the local coordinate system.

Section 3.3 describes the solving procedure of this optimisation problem 3.10. Once the displacements and rotations are known, we can determine the stress in the structure. The first step is to calculate the local forces and moments in every beam-truss element by transforming the forces in the global coordinate system into the local coordinate system. The local force vector of element k is

$$\boldsymbol{f}^k_{\text{local}} = \Big(\underbrace{F_{ix} \quad F_{iy} \quad F_{iz} \quad M_{ix} \quad M_{iy} \quad M_{iz}}_{i} \quad \underbrace{F_{jx} \quad F_{jy} \quad F_{jz} \quad M_{jx} \quad M_{jy} \quad M_{jz}}_{j}\Big)^T, \tag{3.11}$$

and is determined by

$$\boldsymbol{f}^k_{\text{local}} = \boldsymbol{T}^k \boldsymbol{F}^k, \tag{3.12}$$

where $\boldsymbol{F}^k$ are the elements of the global force vector associated with the DoF of element k. Figure 3.2 illustrates the forces and moments on the beam-truss element. $\boldsymbol{T}^k$ is the transformation matrix of element k (COOK *et al.*, 2001). Figure 3.2 shows the forces and moments acting on node j on a beam-truss element.

On every node are acting three forces. F_x is an axial force and F_y and F_z are shear forces. Both shear forces can be combined in one shear force F_s by

$$F_s = \sqrt{F_y^2 + F_z^2}. \tag{3.13}$$

Next, on every node, act three moments. M_x is a in-plane moment (torsion moment) and M_y and M_z are out-of-plane bending moments. Both out-of-plane bending moments can be combined in one moment

$$M_B = \sqrt{M_y^2 + M_z^2}. \tag{3.14}$$

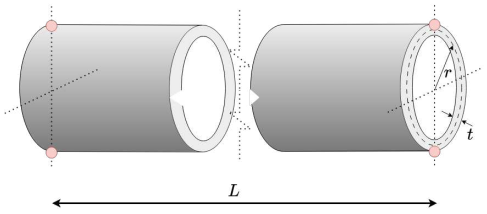


FIGURE 3.3 – Dimensions of the beam-truss element. The red dots indicate the locations where the stress of the beam-truss element will be calculated.

We will calculate the stress at four locations of every beam-truss element. The four locations are at the top and bottom of every node. Figure 3.3 visualises the four stress locations and indicates the three structural dimensions L, r and t of every beam-truss element. The axial force F_x causes a uniform stress σ_x, and the shear force F_s causes a shear stress τ_s. Both stresses are determined by

$$\sigma_x = \frac{F_x}{A}, \qquad \text{and} \qquad \tau_s = v\,\frac{F_s}{A},$$

where A is the cross-sectional area of the element and v is the shear factor. The shear stress is the largest at the midplane, but we will use the maximum shear factor $v = 2$ at the top and bottom to be conservative. The out-of-plane bending moment causes linear varying stress, which is at the top and bottom

$$\sigma_{\text{top}} = \frac{M_B\, r}{I}, \qquad \text{and} \qquad \sigma_{\text{bot}} = -\frac{M_B\, r}{I},$$

respectively. $I = I_y = I_z$ is the second-moment area around the axis of the local coordinate system and r is the radius of the beam-truss element, see figure 3.3. The shear stress due to the torsion moment M_x is

$$\tau_T = \frac{M_x}{2\,A_e\,t}, \tag{3.15}$$

where $A_e = 2\pi r_c^2$ is the enclosed area of the tubular beam-truss element with the radius $r_c = r + t/2$, where r is the radius and t is the wall thickness. Figure 3.3 indicates by a red dot the four different locations where we determine the stress in the beam-truss elements. The equivalent von Mises stress can determine the stress at the top and bottom of every node (WITTEL *et al.*, 2019).

$$\sigma_{vM,\,top} = \sqrt{(\sigma_x + \sigma_{top})^2 + 3(\tau_T + \tau_s)^2}, \qquad \sigma_{vM,\,bot} = \sqrt{(\sigma_x + \sigma_{bot})^2 + 3(\tau_T + \tau_s)^2}$$

The normalised four failure margins at node i and j of the beam-truss element k are

$$m_{k,1}(\boldsymbol{p}) = 1 - \frac{\sigma_{vM\,top\,i}(\boldsymbol{p})}{\sigma_Y} \qquad m_{k,2}(\boldsymbol{p}) = 1 - \frac{\sigma_{vM\,bot\,i}(\boldsymbol{p})}{\sigma_Y}$$

$$m_{k,3}(\boldsymbol{p}) = 1 - \frac{\sigma_{vM\,top\,j}(\boldsymbol{p})}{\sigma_Y} \qquad m_{k,4}(\boldsymbol{p}) = 1 - \frac{\sigma_{vM\,bot\,j}(\boldsymbol{p})}{\sigma_Y},$$

where σ_Y is the yield strength. All failure margins are stored in one margin vector $\boldsymbol{m}(\boldsymbol{p})$

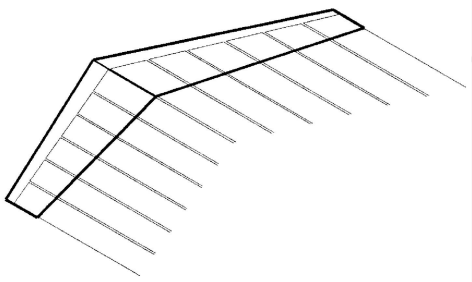
$$\boldsymbol{m}(\boldsymbol{p}) = \Big(m_{1,1}(\boldsymbol{p}),\quad m_{1,2}(\boldsymbol{p}),\quad m_{1,3}(\boldsymbol{p}),\quad m_{1,4}(\boldsymbol{p}),\quad m_{2,1}(\boldsymbol{p}),\ldots\Big). \tag{3.16}$$

The structure does not fail if $\boldsymbol{m}(\boldsymbol{p})$ has no negative elements.

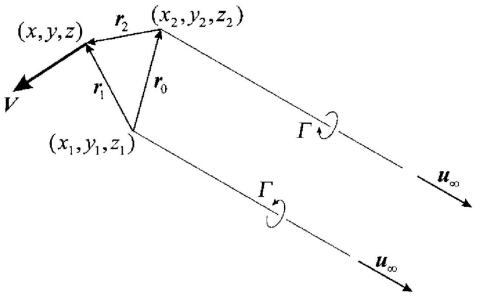
3.2 Aerodynamic model

The aerodynamic model in QASTRO is heavily inspired by a modern adaption of Prandtl's classic LLT of Phillips and Snyder (2000). Section A.2.2 explains the classical Prandtl's LLT. To understand the modern adaption, it is essential to be familiar with the conventional LLT. One of the most significant restrictions of Prandtl's LLT is that the formulation is only valid for a single lifting surface without a sweep or dihedral angle. The downwash in Prandtl's classical LLT is calculated based on the bound vortex and the trailing vortex. The bound vortex is along the lifting-line, and the trailing vortex is parallel to the free stream in the wake; see figure A.11. However, suppose we want to include multiple lifting surfaces and/ or a sweep or dihedral angle. In that case, we also need to encounter the induced velocity by the bound vortex segments on each other. In the classical LLT, we can neglect the interaction of the bound vortices on each other because, due to the parallel nature of each vortex segment, they do not influence each other.

Consequently, in the classical LLT, we can calculate the lift solely based on the induced velocity by the trailing vortex sheet and the free stream velocity. In the modern adaption, Phillips and Snyder (2000) allow multiple lifting surfaces, sweep angles and dihedral angles, which means that they include the interaction of the bound vortices on each other in their formulation. The following part describes the aerodynamic formulation of QASTRO with the same principle ideas as of Phillips and Snyder (2000).



(a) Finite wing with sweep and dihedral angle modelled with horseshoe vortices at the quarter chord.



(b) Horseshoe vortex to model finite wing in QASTRO.

FIGURE 3.4 – Explanation of modern adaption of Prandtl's LLT. Illustrations from Phillips and Snyder (2000).

Let us consider a wing with a sweep and dihedral angle and place multiple horseshoes on the wing as shown in figure 3.4a. The bound vortex of each horseshoe is along the wing quarter-chord line. Please note that the horseshoes are placed differently than in the classical LLT. In figure 3.4a, the horseshoes are placed next to each other instead of on top of each other as in Prandtl's theory; see figure A.13b. From figure 3.4a, it becomes clear that the bound vortex of each vortex creates a downwash on the other side of the wing. If all bound vortices are collinear, as is the case for the classical LLT, then the bound vortices have no interaction on each other.

Figure 3.4b shows one single horseshoe of figure 3.4a. All horseshoes are precisely next to each other such that the left trailing vortex of the right horseshoe lies on the right trailing vortex of the left horseshoe. Each horseshoe vortex consists of three straight vortex lines. The induced velocity of each vortex line is given by figure A.9 and equation A.204 or equation A.205. By combining equation A.205 for all three straight vortex elements of the horseshoe, we get the induced velocity

$$\boldsymbol{V} = \frac{\Gamma}{4\,\pi}\left[\frac{\boldsymbol{u}_\infty \times \boldsymbol{r}_2}{r_2(r_2 - \boldsymbol{u}_\infty \cdot \boldsymbol{r}_2)} + \frac{(r_1 + r_2)(\boldsymbol{r}_1 \times \boldsymbol{r}_2)}{r_1 r_2(r_1 r_2 + \boldsymbol{r}_1 \cdot \boldsymbol{r}_2)} - \frac{\boldsymbol{u}_\infty \times \boldsymbol{r}_1}{r_1(r_1 - \boldsymbol{u}_\infty \cdot \boldsymbol{r}_1)}\right] \tag{3.17}$$

of one horseshoe at an arbitrary point in space (PHILLIPS; SNYDER, 2000). $\boldsymbol{u}_\infty$ is a unit vector in the direction of the freestream, $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are two vectors in space from the vortex tips to an arbitrary point in space. $r_1 = \|\boldsymbol{r}_1\|$ and $r_2 = \|\boldsymbol{r}_2\|$. Please note that the vortex strength in equation 3.17 is still unknown. One possibility to determine the vortex strengths is by defining n_Γ zero flow conditions, where n_Γ is the number of horseshoe vortices. With the additional n_Γ zero flow equations, we can solve for the n_Γ unknown vortex strengths. Nevertheless, QASTRO uses a different approach to determine the vortex strengths $\boldsymbol{\Gamma}$. Let us divide the finite wing into n_Γ sections, where each section has the width of one horseshoe. Every horseshoe has a control point at the midpoint of the bound vortex. We assume that we can express the lift force of the wing section i by the lift force of a two-dimensional airfoil:

$$d\boldsymbol{F}_i = \rho_{\text{Air}}\Gamma_i \boldsymbol{V}_i \times d\boldsymbol{l}_i \tag{3.18}$$

$d\boldsymbol{l}_i$ is the length of the vortex bound. We need to know the local velocity $\boldsymbol{V}_i$ at our control point i which is on the quarter-chord line of the wing section i to compute the lift force of the section. The local velocity $\boldsymbol{V}_i$ along the bound segment of the horseshoe vortex i can be computed by equation 3.17. Each horseshoe j influences the local velocity on the wing section i, including the horseshoe i itself. We can express the local velocity $\boldsymbol{V}_i$ by

$$\boldsymbol{V}_i = \boldsymbol{V}_\infty + \sum_{j=1}^{n_\Gamma} \Gamma_j \boldsymbol{v}_{j,i}, \qquad \text{where} \qquad \boldsymbol{v}_{j,i} = \frac{V_{\text{ind}\,j}}{\Gamma_i}. \tag{3.19}$$

$\boldsymbol{V}_\infty$ is the freestream velocity and $V_{\text{ind}\,j}$ is the induced velocity of horseshoe j on control point i, computed by equation 3.17. $\boldsymbol{v}_{j,i}$ can be seen as a measure of the influence of the horseshoe j on the bound segment of horseshoe i. Please note that each bound segment is placed on the quarter-chord line. Given the local velocity at the quarter-chord of segment i, we get an expression for the spanwise differential lift force of the wing section i:

$$d\boldsymbol{F}_i = \rho_{\text{Air}}\Gamma_i(\boldsymbol{V}_\infty + \sum_{j=1}^{n_\Gamma} \Gamma_j \boldsymbol{v}_{j,i}) \times d\boldsymbol{l}_i, \tag{3.20}$$

where ρ_{Air} is the air density. A second possibility to express the lift force of section i, which is based on the 2D section properties, is

$$|dF_i| = \frac{1}{2}\rho_{\text{Air}} V_i^2 c_{li}(\alpha_i, \delta_i) dA_i \tag{3.21}$$

where dA_i is the wing area of section i, $V_i = \|\boldsymbol{V}_i\|$ and c_{li} is the local lift coefficient of section i which is a function of the local angle of attack α_i and the flap angle δ_i. The local angle of attack α_i is given by

$$\alpha_i = \tan^{-1}\left(\frac{\boldsymbol{V}_i \cdot \boldsymbol{u}_{ni}}{\boldsymbol{V}_i \cdot \boldsymbol{u}_{ai}}\right) \tag{3.22}$$

One can find the definition of $\boldsymbol{u}_{ni}$ and $\boldsymbol{u}_{ai}$ in figure 3.5. Equation 3.21 gives the magnitude of the lift force and $\boldsymbol{u}_{ni}$ the direction of the lift force. Hence, we can define the lift force vector

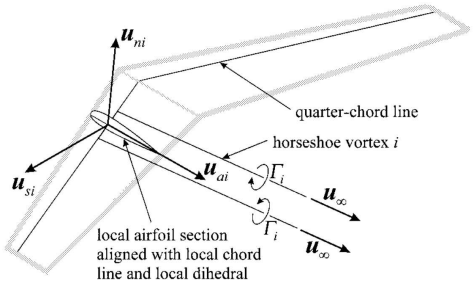


FIGURE 3.5 – Definition of normal vectors on local airfoil section (PHILLIPS; SNYDER, 2000).

as

$$d\boldsymbol{F}_i = |dF_i|\boldsymbol{u}_{ni} = \frac{1}{2}\,\rho_{\text{Air}}\,V_i^2\,c_{li}\,dA_i\,\boldsymbol{u}_{ni}, \tag{3.23}$$

where the lift coefficient is $c_{li} = c_{0i} + \alpha_i\,c_{l\alpha i}$ and $\boldsymbol{u}_{ni}$ is the normal vector aligned with the direction of the lift force. c_{0i} and $c_{l\alpha i}$ are the zero lift coefficient and the lift slope of section i, respectively. In the final step, we need to find the vortex strengths $\boldsymbol{\Gamma}$ such that the lift force vector of equation 3.20 and 3.23 are equal. For this, QASTRO defines the residual function

$$r_i = 2\,\|d\boldsymbol{F}_i - d\boldsymbol{F}_i\| \tag{3.24}$$

$$= \left\| 2\Gamma_i(\boldsymbol{V}_\infty + \sum_{j=1}^{n_\Gamma}\Gamma_j\boldsymbol{v}_{j,i}) \times d\boldsymbol{l}_i - V_i^2 c_{li}(\alpha_i,\delta_i)dA_i\boldsymbol{u}_{ni} \right\|, \tag{3.25}$$

which is solved by an iterative solver for the vortex strength Γ_i. Please note that all residual functions r_i are coupled. Once the vortex strengths are known, we can compute the forces and moments acting on the aircraft. We get the total force vector by adding all force vectors of each section up. Thus,

$$\boldsymbol{F} = \sum_{i=1}^{n_\Gamma} d\boldsymbol{F}_i. \tag{3.26}$$

The drag force D is defined to be parallel to the free stream unit vector $\boldsymbol{u}_\infty$. If we place the wing on the x-y plane, the free stream unit vector without a slipstream (often denoted by $\beta = 0$) is

$$\boldsymbol{u}_\infty = \begin{pmatrix}\cos(\alpha) & 0 & \sin(\alpha)\end{pmatrix}^T, \tag{3.27}$$

where α is the angle of attack. Thus, we can calculate the drag force D by the vector projection of the total force along the free stream direction.

$$D = \boldsymbol{F}\cdot\boldsymbol{u}_\infty = \cos(\alpha)F_x + \sin(\alpha)F_z. \tag{3.28}$$

The lift force L is defined to be perpendicular to the free stream. The lift force is

$$L = \boldsymbol{F}\cdot\boldsymbol{u}_n = -\sin(\alpha)F_x + \cos(\alpha)F_z. \tag{3.29}$$

Given the drag and lift forces, we can calculate the aerodynamic coefficients C_L and C_D.

$$C_L = \frac{L}{\frac{1}{2}\rho_{\text{Air}}S_{\text{ref}}V_\infty^2} \qquad\qquad C_D = \frac{D}{\frac{1}{2}\rho_{\text{Air}}S_{\text{ref}}V_\infty^2} \tag{3.30}$$

S_{ref} is the reference surface area. The acting moment of each lifting surface is

$$dM_i = \frac{1}{2}\rho_{\text{Air}}V_i^2 A_i\,c_i\,c_{mi}, \tag{3.31}$$

where the moment coefficient is $c_{mi} = c_{m0i} + \alpha_i\, c_{m\alpha i}$. c_{m0i} and $c_{m\alpha i}$ are the zero moment coefficient and the moment curve slope of section i, respectively. The moment dM_i of the lifting surface i is acting around the unit vector $\boldsymbol{u}_{si}$. With this unit vector, we can calculate the moment vector of each lifting surface in the three-dimensional space by $d\boldsymbol{M}_i = dM_i \boldsymbol{u}_{si}$. The total moment of the aircraft around the point $\boldsymbol{X}_{\text{ref}}$ is

$$\boldsymbol{M} = \sum_{i}^{n_\Gamma} d\boldsymbol{F}_i \times (\boldsymbol{X}_{ci} - \boldsymbol{X}_{\text{ref}}) + d\boldsymbol{M}_i \tag{3.32}$$

where $\boldsymbol{X}_{ci}$ is the midpoint of the horseshoe i. The midpoint lies on the bound vortex of the horseshoe.

3.3 Aerostructural model

The aerostructural model is a coupling of the structural and aerodynamic models described in sections 3.1 and 3.2. Figure 3.6 illustrates an aerostructural model of a wing. The wing is divided in multiple finite wings. The aerodynamics are described by horseshoe vortices, the structure is modelled by beam-truss elements. The aerodynamic and structural model interact with each other, meaning that the displacements and rotations $\boldsymbol{d}$ of the structural model affect the aerodynamic behaviour, and the aerodynamic forces and moments, in turn, affect the displacements of the structural model. The goal is to find the displacement/rotation vector $\boldsymbol{d}$ and circulation vector $\boldsymbol{\Gamma}$ such that the residual functions of both models are zero. The residual functions of the structural and aerodynamic models are given in equations 3.9 and 3.25, respectively. The workflow of the aerostructural analysis is depicted in figure B.1 and can help to understand this section.

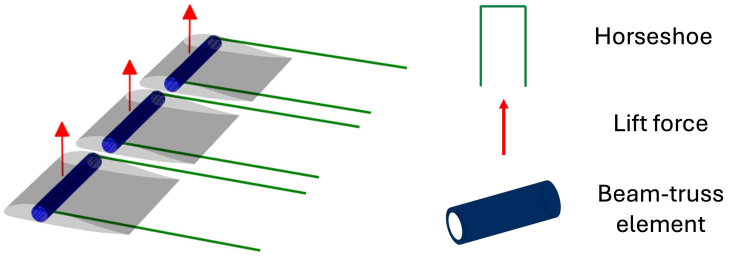


FIGURE 3.6 – Representation of aerostructural model in QASTRO.

To solve the aerostructural model, the user must first make an initial guess for the displacement/rotation vector $\boldsymbol{d}$ and the vortex circulation vector $\boldsymbol{\Gamma}$. The initial guess for the displacements is then transferred onto the aerostructural mesh. It is important to note that the structural and aerostructural mesh in QASTRO do not necessarily need to be identical as in figure 3.6. With the new displaced mesh, QASTRO determines the residual function of the aerodynamic model (eq. 3.25) using the current guess of the vortex circulations. Next, QASTRO calculates the aerodynamic loads of each section i using the current guess of the vortex circulations. However, the aerodynamic forces and moments are not accurate until the residual functions are zero.

The aerodynamic forces and moments calculated by the aerodynamic model serve as input for the structural model, which returns the residual function (eq. 3.9) using the current guess of the displacements. At this point, we have the values of the residual function of both models and can make a new guess for the displacements $\boldsymbol{d}$ and vortex strengths $\boldsymbol{\Gamma}$. The goal is to reduce all residual functions until they are zero or close to zero. Once the residual functions are zero, we have found the aerodynamic model's displacements and circulation strengths. The box below summarises how QASTRO solves the aerostructural model.

Aerostructural solver algorithm steps

1. Check for convergence: If $\boldsymbol{r} \leq \epsilon$, then stop, where
$$\boldsymbol{r} = \begin{pmatrix} \boldsymbol{r}_{struc}^T & \boldsymbol{r}_{aero}^T \end{pmatrix}^T$$
2. Update the mesh with the current displacements $\boldsymbol{d}_k$.
3. Calculate the residual function of the aerodynamic model $\boldsymbol{r}_{aero}(\boldsymbol{\Gamma}_k)$ (eq. 3.25).
4. Determine the aerodynamic forces and moments of each section i with the current circulations $\boldsymbol{\Gamma}_k$:
$$d\boldsymbol{F}_i = |dF_i|\boldsymbol{u}_{ni} = \frac{1}{2}\rho_{\text{Air}} V_i^2 c_{li}(\alpha_i, \delta_i) dA_i \boldsymbol{u}_{ni}$$
$$d\boldsymbol{M}_i = \frac{1}{2}\rho_{\text{Air}} V_i^2 A_i \, c_i \, c_{mi} \, \boldsymbol{u}_{si}$$
5. Calculate the residual function of the structural model $\boldsymbol{r}_{struc}$ (eq. 3.9) with the current displacements vector $\boldsymbol{d}_k$ and the aerodynamic forces and moments calculated in the previous step.
6. Make a new guess for the displacement vector $\boldsymbol{d}_{k+1}$ and circulation vector $\boldsymbol{\Gamma}_{k+1}$.

Making a good new guess for the state variables $\boldsymbol{d}_{k+1}$ and $\boldsymbol{\Gamma}_{k+1}$ can be challenging. The residual functions of the structural and aerostructural model, which are both computed in Fortran, are given to a non-linear Python solver from the SciPy library (VIRTANEN *et al.*, 2020). The non-linear Python solver makes a new guess for the state variables by defining the optimisation problem

$$\min_{\boldsymbol{p}} \quad \left\| \boldsymbol{r}(\boldsymbol{p}) \right\|, \tag{3.33}$$

where $\boldsymbol{r}$ are the residual functions of the structural and aerostructural model, and $\boldsymbol{p}^T = \begin{pmatrix} \boldsymbol{d}^T & \boldsymbol{\Gamma}^T \end{pmatrix}$ are our state variables. We use an SciPy algorithm[1] which uses a modification of the Powell hybrid method (MORE *et al.*, 1980) to solve the optimisation problem stated in equation 3.33. The algorithm determines the search direction $\boldsymbol{q}$ by solving the optimisation problem

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \left\| \boldsymbol{r}(\boldsymbol{p}_k) + \boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{p}_k)\boldsymbol{q} \right\| \\ \text{s.t.} \quad & \|\boldsymbol{D}\boldsymbol{q}\| \leq \Delta, \end{aligned} \tag{3.34}$$

by the Powell hybrid method (POWELL, 1970), where $\boldsymbol{p}_k$ are the state variables and $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{p}_k)$ is the Jacobian matrix at the current iteration k of the residual function $\boldsymbol{r}$. $\boldsymbol{D}$ is a diagonal scaling matrix and Δ is a step bound. Unfortunately, the user needs to provide the full Jacobian matrix instead of a function which returns the directional derivative $\boldsymbol{J}_{\boldsymbol{r}}(\boldsymbol{p}_k)\boldsymbol{q}$. The Jacobian-vector product could be computed with a single AD execution. However, it requires r_n AD calls to get the full Jacobian matrix, where r_n is the total number of residual functions. The directional derivative we could get with only a single forward-mode AD call. For the aerostructural problem, we compute the Jacobian matrix by the forward-mode AD; see section A.1.6.1. If we solve only

[1] `https://github.com/scipy/scipy/blob/main/scipy/optimize/_root.py` accessed at 27.08.23

the structural problem to perform structural optimisations, we do not provide the Jacobian matrix to the algorithm. In this case, the Jacobian matrix is approximated by the forward difference method at the first iteration. At the next iterations, the algorithm determines the Jacobian matrix $\boldsymbol{J_r}(\boldsymbol{p}_k)$ by the Broyden rank-1 updating method (BROYDEN, 1965). This is in contrast to the original Powell hybrid method. The Jacobian matrix becomes again determined by a finite difference step if the rank-1 updating method no longer provides satisfactory progress. If the optimisation problem stated in equation 3.34 does not provide a suitable search direction $\boldsymbol{q}$ (e.g. not a sufficient decrease), Δ is decreased or, if needed, $\boldsymbol{J_r}$ is computed again. The next iteration of the modified Powell hybrid algorithm is

$$\boldsymbol{p}_{k+1} = \boldsymbol{p}_k + \boldsymbol{q}. \tag{3.35}$$

The optimisation problem stated in equation 3.34 is solved again at the new iteration, and this procedure is repeated until $\boldsymbol{r}(\boldsymbol{p}^*) \approx \boldsymbol{0}$. For the exact convergence criteria and further details, the author refers to (MORE *et al.*, 1980).

Once the state variables $\boldsymbol{p}$ are known, we also know all variables which depend on the state variable, such as the failure margins or the lift force.

3.4 Optimisation methods

This Master's Thesis compares three distinct approaches to solving a constrained aerostructural optimisation problem. This section gives the reader a detailed view of how the optimisation methods work. All three approaches solve the general optimisation problem

$$\begin{aligned} \min_{\text{w.r.t. } \boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{subjected to (s.t.)} \quad & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{0},\ \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{p}) \leq \boldsymbol{0},\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ \text{while solving (w.s.)} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{3.36}$$

where $\boldsymbol{x}$ are the design variables, $\boldsymbol{p}$ are the state variables $\boldsymbol{d}$ and $\boldsymbol{\Gamma}$, f is the objective function, $\boldsymbol{h}$ are the equality constraints, $\boldsymbol{g}$ are the inequality constraints, $\boldsymbol{l}$ is the lower bound vector and $\boldsymbol{u}$ is the upper bound vector. The solution of the optimisation problem 3.36 is denoted by $\boldsymbol{x}^*$. We consider optimisation problems with many inequality constraints. The purpose of comparing the three optimisation approaches is to see how they handle those many inequality constraints. Please note that it is essential to be familiar with the Theoretical Background in the Appendix chapter to understand the optimisation approaches.

3.4.1 SLSQP approach

The SLSQP approach uses the Sequential Least SQuares Programming (SLSQP) method of Wilson (1963), Han (1977) and Powell (1978b), which is described in the paper of Schittkowski (1981a). The description of the SLSQP algorithm[2] is given in the paper of Dieter Kraft (KRAFT, 1988) and is implemented in the SciPy library (VIRTANEN *et al.*, 2020), which QASTRO utilises. The SLSQP method solves the original constrained optimisation problem of equation 3.36 without any aggregation functions. The SLSQP algorithm of Dieter Kraft is an enhancement of the SQP method explained in section A.1.4.6. In contrast to Schittkowski's and Dieter Kraft's original paper, this section shows the SLSQP method's derivation based on the SQP method. The disadvantage of the SQP method is that it requires the Hessian matrix of the Lagrangian func-

[2] `https://github.com/scipy/scipy/blob/v1.10.1/scipy/optimize/_slsqp_py.py#L68-L211` accessed at 27.08.23

tion $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T \boldsymbol{g}(\boldsymbol{x})$, where we denote the equality and inequality Lagrange multipliers with $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$, respectively. The Hessian of the Lagrangian function is computationally expensive to obtain. The SLSQP method uses the Quasi-Newton method of section A.1.4.2 to avoid calculating the Hessian matrix of the Lagrangian function and formulates the linear system at each iteration as a least square optimisation problem. Nevertheless, let us go through it step-by-step. The SQP method solves an optimisation problem of a quadratic approximation of the Lagrangian function subjected to the linearised constraints at each iteration. The sub-optimisation problems at each iteration k of the SQP method, including inequality constraints, is

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & d(\boldsymbol{q}) = \frac{1}{2} \boldsymbol{q}^T \left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \right] \boldsymbol{q} + \boldsymbol{q}^T \left[\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \right] \\ & = \frac{1}{2} \boldsymbol{q}^T \left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \right] \boldsymbol{q} + \boldsymbol{q}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) + \boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda} + \boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{\mu} \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = \boldsymbol{0}, \, \boldsymbol{g}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}_k)^T \boldsymbol{q} \leq \boldsymbol{0}, \end{aligned} \tag{3.37}$$

where $\boldsymbol{q}$ is the search direction. Note that we did not include the bound constraints. We will address later how we ensure that the algorithm gives a search direction within the bounds. Let us give the term $\boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda}$ a little bit more attention. The term is a scalar value, so the transpose of the expression is an equivalent scalar value. Thus,

$$\boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda} = \left\{ \boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda} \right\}^T \tag{3.38}$$

$$= \boldsymbol{\lambda}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q}. \tag{3.39}$$

From the constraint follows that $\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = -\boldsymbol{h}(\boldsymbol{x}_k)$. Hence,

$$\boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda} = -\boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}_k), \tag{3.40}$$

which shows that the term $\boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda}$ of the objective function $d(\boldsymbol{q})$ is independent of $\boldsymbol{q}$. The same holds for the term $\boldsymbol{q}^T \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}) \boldsymbol{\mu}$. The solution to the optimisation problem only depends on the active constraints; hence, we can write the inequality constraints as equality constraints at $\boldsymbol{q}^*$. Moreover, all inactive constraints at $\boldsymbol{q}^*$ do not influence the solution and can anyway be ignored. That means that we can ignore both terms and we can define an equivalent optimisation problem as follows

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & d(\boldsymbol{q}) = \frac{1}{2} \boldsymbol{q}^T \boldsymbol{B} \boldsymbol{q} + \boldsymbol{q}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = \boldsymbol{0}, \, \boldsymbol{g}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}_k)^T \boldsymbol{q} \leq \boldsymbol{0}, \end{aligned} \tag{3.41}$$

with $\boldsymbol{B} = \left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \right]$. However, as mentioned above, it is computationally expensive to compute the Hessian at every iteration. But we can approximate the Hessian matrix by the BFGS - formula proposed by Powell (1978b)

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}, \tag{3.42}$$

where $\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k = \alpha_k \boldsymbol{q}$. α_k is a scalar value and determines the step size. $\boldsymbol{y}_k$ is not the difference of the gradient of the Lagrangian function as one would expect, but Powell defined $\boldsymbol{y}_k = \theta_k \boldsymbol{\eta}_k + (1 - \theta_k) \boldsymbol{B}_k \boldsymbol{s}_k$, where $\boldsymbol{\eta}_k \nabla \mathcal{L}(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_k) - \nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)$ is the difference of the gradients of the Lagrangian function and θ_k is determined by

$$\theta_k = \begin{cases} 1 & \text{if } \boldsymbol{s}_k^T \eta_k \geq 0.2 \boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k, \\ \frac{0.8 \boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}{\boldsymbol{s}_k \boldsymbol{B}_k \boldsymbol{s}_k^T - \boldsymbol{s}_k^T \eta_k} & \text{otherwise.} \end{cases} \tag{3.43}$$

The choice to define $\boldsymbol{y}_k$ in this way ensures that $\boldsymbol{B}$ is positive definite if $\boldsymbol{B}$ is initialised with a positive definite matrix. The value 0.2 is chosen empirically. By using the LDL-decomposition $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$, where $\boldsymbol{L}$ is a lower triangular matrix and $\boldsymbol{D}$ is a diagonal matrix (SCHITTKOWSKI, 1981b), (KRAFT, 1988) we can replace the sub-optimisation problem stated in equation 3.41 by

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \left\| \boldsymbol{D}^{1/2}\boldsymbol{L}^T\boldsymbol{q} + \boldsymbol{D}^{-1/2}\boldsymbol{L}^{-1}\nabla f(\boldsymbol{x}_k) \right\| \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}_k)^T\boldsymbol{q} = \boldsymbol{0},\ \boldsymbol{g}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}}\boldsymbol{g}(\boldsymbol{x}_k)^T\boldsymbol{q} \leq \boldsymbol{0}. \end{aligned} \tag{3.44}$$

Thus, instead of solving the optimisation problem 3.41 at every iteration, we solve the linear least squares problem 3.44 at every iteration to get the search direction $\boldsymbol{q}$. We broke down a complex non-linear optimisation problem into several linear sub-optimisation problems. At this point, the advantage of rewriting the problem might become clear. The optimisation problem 3.44 is equivalent to solve the overdetermined system $\boldsymbol{A}\boldsymbol{q} = \boldsymbol{b}$ for $\boldsymbol{q}$, but with constraints. Solving overdetermined systems is already well understood, which helps to solve the optimisation problem 3.44. However, we still need to take care of our constraints. Unfortunately, Kraft (1988) did not document how the SLSQP algorithm handles bound constraints, but one can verify in the source code[3] that the SLSQP algorithm includes the bounds by defining a bound constraint for the search direction $\boldsymbol{q}$ in the sub-optimisation problem 3.44. The lower and upper bound of the search direction $\boldsymbol{q}$ are

$$\boldsymbol{l}_{\boldsymbol{q}} = \boldsymbol{l} - \boldsymbol{x}_k, \qquad \text{and} \qquad \boldsymbol{u}_{\boldsymbol{q}} = \boldsymbol{u} - \boldsymbol{x}_k, \tag{3.45}$$

respectively. Those bound constraints ensure that the sub-problem will give us a search direction such that $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{q}$ satisfies the bound $\boldsymbol{l} \leq \boldsymbol{x}_{k+1} \leq \boldsymbol{u}$. Please note that including bounds like that is equivalent to truncating the search direction to the bound, which can cause small steps in the algorithm. There are more sophisticated methods to include bound constraints, as we will see in section 3.4.3. Let us define the optimisation problem we need to solve at each sub-iteration, including the new bound constraints:

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \|\boldsymbol{A}\boldsymbol{q} - \boldsymbol{b}\| \\ \text{s.t.} \quad & \boldsymbol{G}\boldsymbol{q} - \boldsymbol{b} = \boldsymbol{0}, \quad \boldsymbol{C}\boldsymbol{q} - \boldsymbol{d} \leq \boldsymbol{0}, \quad \boldsymbol{l}_{\boldsymbol{q}} \leq \boldsymbol{q} \leq \boldsymbol{u}_{\boldsymbol{q}}, \end{aligned} \tag{3.46}$$

where in our case $\boldsymbol{A} = \boldsymbol{D}^{1/2}\boldsymbol{L}^T$, $\boldsymbol{b} = -\boldsymbol{D}^{-1/2}\boldsymbol{L}^{-1}\nabla f(\boldsymbol{x}_k)$, $\boldsymbol{G} = \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}_k)^T$, $\boldsymbol{b} = -\boldsymbol{h}(\boldsymbol{x}_k)$, $\boldsymbol{C} = \nabla_{\boldsymbol{x}}\boldsymbol{g}(\boldsymbol{x}_k)^T$ and $\boldsymbol{d} = -\boldsymbol{g}(\boldsymbol{x}_k)$. Without any constraints, it is easier to solve the optimisation problem stated in equation 3.46, so we will try to eliminate the constraints. The SLSQP algorithm rewrites the bound constraint $\boldsymbol{l}_{\boldsymbol{q}} \leq \boldsymbol{q} \leq \boldsymbol{u}_{\boldsymbol{q}}$ as two inequality constraints,

$$-\boldsymbol{I}\boldsymbol{q} + \boldsymbol{l}_{\boldsymbol{q}} \leq \boldsymbol{0}, \qquad \text{and} \qquad -\boldsymbol{I}\boldsymbol{q} - \boldsymbol{u}_{\boldsymbol{q}} \leq \boldsymbol{0} \tag{3.47}$$

where $\boldsymbol{I}$ is the identity matrix. Replacing the bound constraint with inequality constraints is generally not the best option because bound constraints are 'simpler' than inequality constraints. Furthermore, we exchange one bound constraint for two inequality constraints, which is inefficient. The equality constraints in 3.46 state that we need to satisfy the linear equations $\boldsymbol{G}\boldsymbol{q} = \boldsymbol{b}$. Lawson and Hanson (1995) show that you can eliminate linear equality constraints by substituting the linear equality constraints in the linear least squares problem. This requires to compute the orthogonal basis of the nullspace of $\boldsymbol{G}$. After replacing the bound constraints with inequality constraints and eliminating the equality constraints, we are left to solve

[3] `https://github.com/scipy/scipy/blob/v1.8.0/scipy/optimize/slsqp/slsqp_optmz.f#L792` accessed at 27.08.23

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \left\| \boldsymbol{A}' \boldsymbol{q} - \boldsymbol{b}' \right\| \\ \text{s.t.} \quad & \boldsymbol{C}' \boldsymbol{q} - \boldsymbol{d}' \leq \boldsymbol{0}, \end{aligned} \tag{3.48}$$

where $\boldsymbol{A}'$ and $\boldsymbol{b}'$ are the modified versions of $\boldsymbol{A}$ and $\boldsymbol{b}$ such that we automatically satisfy the equality constraints. $\boldsymbol{C}'$ and $\boldsymbol{d}'$ are the augmented matrix and vector of $\boldsymbol{C}$ and $\boldsymbol{d}$ by the inequality constraints. Problem 3.48 is a linear least squares problem with only inequality constraints. The SLSQP algorithm follows the approach of Lawson and Hanson (1995) to solve the problem for the search direction $\boldsymbol{q}$, satisfying all constraints. Further, the step size α_k is chosen by solving the constrained one-dimensional problem

$$\begin{aligned} \min_{\alpha_k} \quad & f(\boldsymbol{x}_k + \alpha_k \boldsymbol{q}_k) \\ \text{s.t.} \quad & \alpha_{\min} \leq \alpha_k \leq \alpha_{\max}, \end{aligned} \tag{3.49}$$

where $\alpha_{\min}$ is the minimum step size of the algorithm and $\alpha_{\max} = 1$ because with $\alpha_{\max} > 1$, it is not guaranteed anymore that we will satisfy the bound constraints of the original optimisation problem stated in equation 3.36. One could use any optimisation method which can handle bound constraints to calculate the step size α_k. This SLSQP algorithm combines the Golden Section and a successive quadratic interpolation to solve the one-dimensional optimisation problem 3.49. The box below describes the steps of the SLSQP algorithm. Note that the user needs to initialise $\boldsymbol{B}$, which gives the matrices $\boldsymbol{D}$ and $\boldsymbol{L}$ for the first sub-problem to solve.

Sequential least squares programming method algorithm steps

1. Check for convergence: If $\|\boldsymbol{q}\| \leq \epsilon$, then stop.
2. Solve sub-problem to get $\boldsymbol{q}$:

$$\begin{aligned} \min_{\boldsymbol{q}} \quad & \left\| \boldsymbol{D}^{1/2} \boldsymbol{L}^T \boldsymbol{q} + \boldsymbol{D}^{-1/2} \boldsymbol{L}^{-1} \nabla f(\boldsymbol{x}_k) \right\| \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = \boldsymbol{0},\ \boldsymbol{g}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x}_k)^T \boldsymbol{q} \leq \boldsymbol{0} \\ & \boldsymbol{l}_{\boldsymbol{q}} \leq \boldsymbol{q} \leq \boldsymbol{u}_{\boldsymbol{q}} \end{aligned}$$

3. Perform line search to get α_k
4. Set $\boldsymbol{s}_k = \alpha_k \boldsymbol{q}_k$ and make step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$
5. Calculate $\boldsymbol{y}_k = \theta_k \boldsymbol{\eta}_k + (1 - \theta_k) \boldsymbol{B}_k \boldsymbol{s}_k$
6. Update approximation of the Hessian

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}.$$

7. Perform LDL - decomposition $\boldsymbol{B} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$

3.4.2 SLSQP with aggregation function approach

As section 3.1 shows, every element of the finite element model gives four failure margins. Those failure margins are the inequality constraints of the optimisation problem given in equation 3.36. Consequently, the optimisation problem becomes cumbersome with many finite elements. Aggregation functions merge all inequality constraints in one single scalar function and simplify the optimisation problem, as explained in section A.1.5. The second optimisation approach uses

the SLSQP algorithm described in the previous section but merges all inequality constraints in the function

$$\bar{g}_{KS}(\rho_{KS}, \boldsymbol{g}) = \max_j g_j + \frac{1}{\rho_{KS}} \ln \left[\sum_{j=1}^{n_g} \exp \left(\rho_{KS} \left\{ g_j - \max_j g_j \right\} \right) \right], \tag{3.50}$$

which is an alternative KS function. The original KS function, shown in equation A.104, can easily result in overflow (MARTINS; NING, 2021). However, both KS functions are equivalent. ρ_{KS} determines how accuratly the inequality constraints are aggregated. For $\rho_{KS} \to \infty$, the KS function becomes the *max* function. n_g is the total number of inequality constraints. The optimisation problem of the aggregation function approach (SLSQP KS) reads

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{0},\ \bar{g}_{KS}(\rho_{KS}, \boldsymbol{g}) \leq 0,\ \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}. \end{aligned} \tag{3.51}$$

3.4.3 Augmented Lagrangian approach

The third approach to solve the optimisation problem given in equation 3.36 is by the ALM of Fletcher (1975) described in section A.1.4.5. The augmented Lagrangian $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ function of the ALM algorithm is

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right\|_2^2 + \left\| \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle \right\|_2^2 \right) \tag{3.52}$$

$$= f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \rho_k) \right\|_2^2 + \left\| \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k, \rho_k) \right\|_2^2 \right), \tag{3.53}$$

where $\tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \rho_k) = \left(\boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right)$ and $\tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k, \rho_k) = \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle$ where $\langle a \rangle = max(0, a)$. ρ_k is the penalty factor. $\boldsymbol{\lambda}_k$ and $\boldsymbol{\mu}_k$ are the Lagrange multipliers associated with the equality and inequality constraints at the current iteration k, respectively. The ALM algorithm can be divided into three steps as described in section A.1.4.5. The steps of the algorithm in this Master's Thesis are described in the box below, where $\boldsymbol{l}$ and $\boldsymbol{u}$ are vectors of the lower and upper bound of $\boldsymbol{x}$, respectively. In step 3, it is indicated that ρ_{k+1} must not essentially increase. In fact, if ρ_k becomes too large, the problem might become ill-conditioned. For this reason ρ_{k+1} becomes only increased by $\rho_{k+1} = \rho_k \gamma$ if $V_k \geq r\, V_{k-1}$, where

$$V_k = \max \left\{ \left\| \boldsymbol{h}(\boldsymbol{x}_k) \right\|, \left\| \max \left\{ \boldsymbol{g}(\boldsymbol{x}_k), -\frac{\boldsymbol{\mu}_k}{\rho_k} \right\} \right\| \right\}. \tag{3.54}$$

V_k can be seen as a measure of feasibility. The goal is to increase only the penalty factor if updating the Lagrange multipliers is insufficient to obtain feasible enough design states. This choice for increasing ρ_k can also be found in different papers, e.g. in Birgin and Martínez (2019). The term $\max \left\{ \boldsymbol{g}(\boldsymbol{x}_k), -\frac{\boldsymbol{\mu}_k}{\rho_k} \right\}$ is a vector, which is at position j the maximum value between g_j and $-\mu_j/\rho_k$, where g_j is the constraint value at iteration k of constraint j and μ_j is the corresponding Lagrange multiplier ($\mu_j \geq 0$). The condition $V_k \geq r\, V_{k-1}$ ensures we only increase ρ_k if iteration k is infeasible enough. $r = 0$ implies that we will increase ρ_k at every iteration, and, e.g. $r = 1$ implies that iteration k must be as least as infeasible as the previous iteration that we increase ρ_k. For $r > 1$, iteration k must be even more infeasible than iteration $k - 1$, that ρ_k gets increased. The user is free to use the initial penalty factor ρ_0.

Additionally, the user has the option to use a separate penalty factor for the equality and the inequality constraints. In this case, we define a separate V_k value for the equality and inequality constraints, which read

$$V_{k\,h} = \left\|\boldsymbol{h}(\boldsymbol{x}_k)\right\| \qquad \text{and} \qquad V_{k\,g} = \left\|\max\left\{\boldsymbol{g}(\boldsymbol{x}_k), -\frac{\boldsymbol{\mu}_k}{\rho_k}\right\}\right\|. \tag{3.55}$$

Please note that we get the same expression for V_k if we optimise only with equality or inequality constraints and not both constraint types simultaneously.

The ALM algorithm is written by the author. However, the sub-problems are solved by an optimiser of the SciPy library (VIRTANEN *et al.*, 2020). The remaining part of this section describes the solving procedure of the sub-problem.

Augmented Lagrangian approach algorithm steps

1. Check for optimality: If
$$|\mathcal{A}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) - \mathcal{A}(\boldsymbol{x}_{k-1}, \boldsymbol{\lambda}_{k-1}, \boldsymbol{\mu}_{k-1}, \rho_{k-1})| \leq \epsilon,$$
then stop.

2. Solve sub-problem:
$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \mathcal{A}(\boldsymbol{x}) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\|\tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k)\right\|_2^2 + \left\|\tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k)\right\|_2^2\right) \\ \text{s.t.} \quad & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \end{aligned}$$
which gives $\boldsymbol{x}_{k+1}$.

3. Update $\boldsymbol{\lambda}_k$, $\boldsymbol{\mu}_k$ and ρ_k
$$\begin{aligned} \boldsymbol{\lambda}_{k+1} &= \boldsymbol{\lambda}_k + \rho_k \boldsymbol{h}(\boldsymbol{x}_{k+1}) \\ \boldsymbol{\mu}_{k+1} &= \max\left\{0, \boldsymbol{\mu}_k + \rho_k \boldsymbol{g}(\boldsymbol{x}_{k+1})\right\} \\ \rho_{k+1} &\geq \rho_k \end{aligned}$$

Section A.1.4.5 shows that the ALM converts the constrained optimisation problem given in equation 3.36 into multiple unconstrained sub-optimisation problems. However, in this Master's Thesis, we must constrain the sub-problems with a lower and upper bound. $\boldsymbol{x}$ are the design variables such as the length L, the radius r and the thickness t of the finite element beam-truss elements, see figure 3.3. Those dimensions must remain greater than zero. The author chose to use the algorithm[4] of Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu (BYRD *et al.*, 1995), which is implemented in the SciPy library (VIRTANEN *et al.*, 2020). The algorithm is a limited storage Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm with bound constraints (L-BFGS-B) and is used to solve the sub-problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\|\tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k)\right\|_2^2 + \left\|\tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k)\right\|_2^2\right) \\ \text{s.t.} \quad & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}. \end{aligned} \tag{3.56}$$

of the ALM to get the next iteration step $\boldsymbol{x}_{k+1}$. Please note that the only free variable of the optimisation problem given in equation 3.56 is $\boldsymbol{x}$; $\boldsymbol{\lambda}_k$, $\boldsymbol{\mu}_k$ and ρ_k are fixed. We denote the iterations of the L-BFGS-B algorithm with a bar, e.g., $\bar{\boldsymbol{x}}_{k+1}$, to distinguish them from the iteration of the ALM algorithm. We give to the algorithm a function which can compute the augmented Lagrangian function $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ and its gradient $\nabla\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ at any point

[4] `https://github.com/scipy/scipy/tree/main/scipy/optimize/lbfgsb_src` accessed at 27.08.23

$\boldsymbol{x}_k$. Providing this information allows us to approximate the augmented Lagrangian function by a quadratic model at $\bar{\boldsymbol{x}}_k$,

$$m_k(\boldsymbol{x}) = \mathcal{A}(\bar{\boldsymbol{x}}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) + \nabla\mathcal{A}(\bar{\boldsymbol{x}}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)^T(\boldsymbol{x} - \bar{\boldsymbol{x}}_k) + \frac{1}{2}(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)^T\boldsymbol{B}_k(\boldsymbol{x} - \bar{\boldsymbol{x}}_k). \tag{3.57}$$

The approximated Hessian matrix $\boldsymbol{B}_k$ is not computed explicitly. Section A.1.4.3 gives an idea of how limited memory approximation of the Hessian matrix works. Instead of computing $\boldsymbol{B}_k$, we compute directly $(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)^T\boldsymbol{B}_k(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)$ by the gradients of n_{mc} previous iterations. The author chose to use $n_{mc} = 15$ because this gives, in general, a good estimation of the Hessian matrix (ZHU *et al.*, 1997). The computation of $(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)^T\boldsymbol{B}_k(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)$ is described in section B.3 in the Appendix.

Figure 3.7 shows a two-dimensional example of an arbitrary augmented Lagrangian function $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ and it's quadratic approximation $m_k(\boldsymbol{x})$ at $\bar{\boldsymbol{x}}_k$.

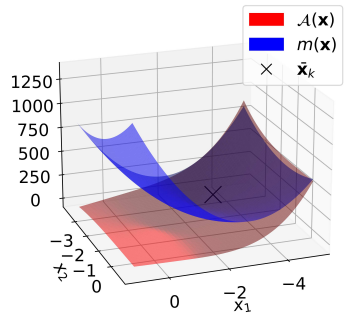


FIGURE 3.7 – Augmented Lagrangian function $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ and it's quadratic approximation $m_k(\boldsymbol{x})$ at $\bar{\boldsymbol{x}}_k$.

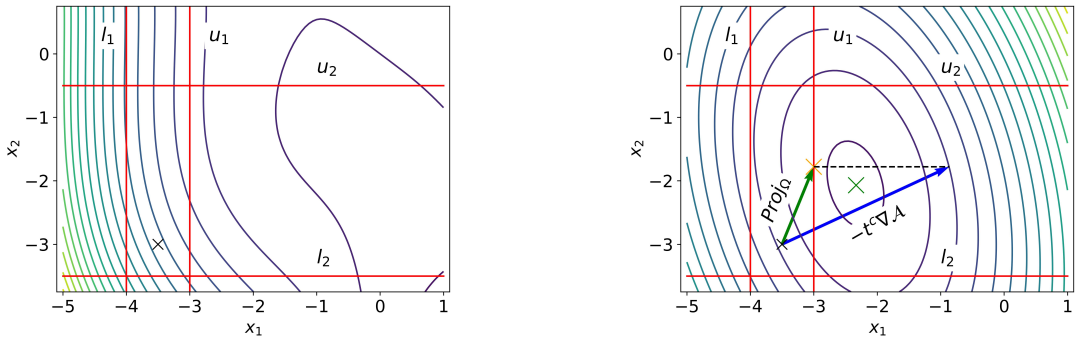
The algorithm minimises a sequence of quadratic approximations $m_k(\boldsymbol{x})$ subjected to the bounds. Bounds are the most simple constraints to optimisation problems. The algorithm determines the active bounds and fixes the design variables associated with the active bounds by the value of the Cauchy point. Next, let us discuss how the algorithm defines the active set and the Cauchy point. The active bounds are determined by the projection of the steepest descent direction with an optimal step size t^c onto the feasible region Ω. The steepest descent direction is the negative direction of the gradient, scaled by a step size t. Figure 3.8a and 3.8b show the contour lines of $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ and $m_k(\boldsymbol{x})$ of figure 3.7, respectively. The black cross indicates the current design iteration $\bar{\boldsymbol{x}}_k$. The blue arrow shows the steepest descent direction of $m_k(\bar{\boldsymbol{x}})$, and the green arrow is a projection onto the feasible region Ω. The projection onto Ω is obtained by

$$Proj_\Omega(x, l, u)_i = \begin{cases} l_i, & \text{if } x_i < l_i \\ x_i, & \text{if } x_i \in [l_i, u_i] \\ u_i, & \text{if } x_i > u_i \end{cases}. \tag{3.58}$$

The projection of the steepest descent direction is the continuous, piecewise differentiable function

$$\boldsymbol{x}(t) = Proj_\Omega(\{\boldsymbol{x}_k - t\nabla\mathcal{A}\}, \boldsymbol{l}, \boldsymbol{u}), \tag{3.59}$$

which is a function of the step size t. We get the optimal step size t^c by the generalised Cauchy point $\bar{\boldsymbol{x}}^c$, which is indicated by the orange cross in figure 3.8b. The generalised Cauchy point is the first local minimiser of the one-dimensional function $m_k(\boldsymbol{x}(t))$. $m_k(\boldsymbol{x}(t))$ is a continuous, quadratic piecewise differentiable function which is along the projection onto Ω of the steepest descent direction. Figure 3.9a shows the univarate function $m_k(\boldsymbol{x}(t))$ of our two-dimensional



(a) Contour lines of $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$

(b) Contour lines of $m_k(\boldsymbol{x})$

FIGURE 3.8 – Explanation of sub-optimisation method of ALM approach. The black cross is the current iteration $\bar{\boldsymbol{x}}_k$; the red lines show the bounds, the blue arrow shows the steepest descent direction with step size t_c, and the green arrow shows the projection of the blue arrow onto the feasible region Ω.

plots. After the first non-differential location at $t \approx 0.0025$, the steepest descent direction gets projected onto the upper bound of x_1; hence one bound is active. The second non-differential location implies that the second bound is active, which means that the steepest descent direction is projected onto the upper right corner of the feasible set. Any increase in the step size t cannot change the value of $m_k(\boldsymbol{x}(t))$ anymore. In this example, the first (and only) local minimiser is at $t \approx 0.013$, which is the optimal step size. Note that for this step size, one bound is active. The optimal step size determines the Cauchy point $\bar{\boldsymbol{x}}^c$. The active set is a set of all variables whose values are at the upper or lower bound at the Cauchy point. Next, we can define the optimisation problem of our quadratic approximation of $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ with our active set

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \left\{ m_k(\boldsymbol{x}) : x_i = x_i^c \quad \forall \in [l_i, u_i] \right\} \\ \text{s.t.} \quad & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \quad \forall \notin [l_i, u_i] \end{aligned} \tag{3.60}$$

with the solution $\bar{\boldsymbol{x}}^*$. Note that we fixed the active bounds with the value of the Cauchy point. In fact, we reduce the problem's dimensionality by the number of active bounds at the Cauchy point. The optimisation problem stated in 3.60 is easy to solve because it is a quadratic problem, and we can use the Cauchy point as our first initial guess, which is usually already close to the solution. Figure 3.9b shows the remaining optimisation problem of our example throughout this explanation. The upper bound of the x_1-dimension is active at the Cauchy point; hence, we are left with an optimisation problem in the x_2-dimension. In this simple example, the Cauchy point is even the solution to the optimisation problem of equation 3.60. The orange cross indicates the Cauchy point $\bar{\boldsymbol{x}}^c = \bar{\boldsymbol{x}}^*$.

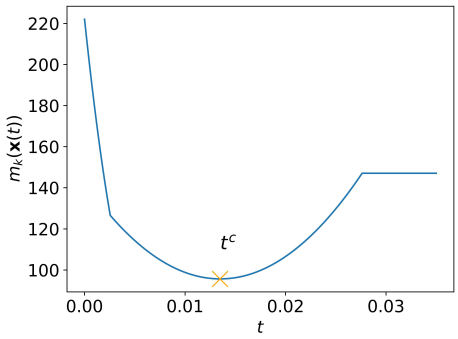
Once we get the solution $\bar{\boldsymbol{x}}^*$ to the optimisation problem in equation 3.60 we get the search direction $\bar{\boldsymbol{q}}_k = \bar{\boldsymbol{x}}^* - \bar{\boldsymbol{x}}_k$. Along this search direction, we can find a step size α_k which satisfies the sufficient decrease condition of Armijo

$$\mathcal{A}(\bar{\boldsymbol{x}}_{k+1}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) \leq \mathcal{A}(\bar{\boldsymbol{x}}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) + \alpha_k c \nabla \mathcal{A}(\bar{\boldsymbol{x}}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)^T \bar{\boldsymbol{q}}_k, \tag{3.61}$$

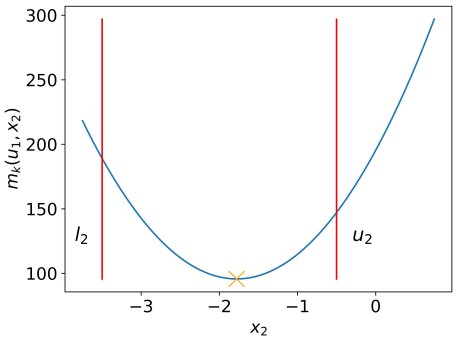
where $c = 10^{-4}$ and also satisfies the curvature condition

$$|\nabla \mathcal{A}(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)^T \bar{\boldsymbol{q}}| \leq \beta |\nabla \mathcal{A}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)^T \bar{\boldsymbol{q}}|, \tag{3.62}$$

where $\beta = 0.9$. Given the step size α_k, which satisfies both conditions, also known as the Wolf conditions, the next iteration of the sub-problem is $\bar{\boldsymbol{x}}_{k+1} = \alpha_k \bar{\boldsymbol{q}}_k$. At $\bar{\boldsymbol{x}}_{k+1}$ the algorithm computes $\mathcal{A}(\bar{\boldsymbol{x}}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}, \rho_{k+1})$, $\nabla \mathcal{A}(\bar{\boldsymbol{x}}_{k+1}, \boldsymbol{\lambda}_{k+1}, \boldsymbol{\mu}_{k+1}, \rho_{k+1})$ and $(\boldsymbol{x} - \bar{\boldsymbol{x}}_{k+1})^T \boldsymbol{B}_{k+1} (\boldsymbol{x} - \bar{\boldsymbol{x}}_{k+1})$ again and repeats the process until convergence. The convergence criteria of the sub-problem is



(a) m_k along the projection onto Ω of the steepest descent direction as a function of step size t.



(b) Optimisation problem of eq. 3.60 with active set determined by the projection onto Ω of steepest descent direction onto the feasible set.

FIGURE 3.9 – Explanation of sub-optimisation method of ALM approach.

$$|\mathcal{A}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) - \mathcal{A}(\boldsymbol{x}_{k-1}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)| \leq \epsilon = 10^{-6}. \tag{3.63}$$

3.5 Computing derivatives

The previous section described the optimisation methods to solve the optimisation problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{0},\ \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{p}) \leq \boldsymbol{0} \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}, \end{aligned} \tag{3.64}$$

where we ignore the bound constraints since they do not influence the derivative computation. However, there was no attention given that we need to minimise f while solving $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$. In our case, $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x})$ are the residual functions of the structural and aerostructural models given in equation 3.9 and 3.25, respectively. There are two methods to solve an optimisation problem while solving $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$; the direct and the adjoint method. Section A.1.7 describes both methods. We will use the adjoint method because structural and aerostructural optimisation problems usually have more design variables than functions of interest. The adjoint method computes the derivatives along a linear approximation of $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$ and gives those derivatives to the optimisers described in the previous section. By giving the optimiser derivatives along a linear approximation of $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$ we implicitly ensure that the optimiser encounters for the relationship between the design and state variables.

QASTRO calculates the partial derivatives for the adjoint by AD. Section A.1.6 of the Theoretical Background chapter describes how the AD works. There are two AD methods: the direct method and the adjoint method. QASTRO uses the adjoint method instead of the direct method because there are usually more design variables $\boldsymbol{x}$ than functions of interest. Please ensure you are familiar with the adjoint method described in section A.1.7. The partial derivatives for the adjoint method are obtained by Tapenade AD (HASCOËT; PASCUAL, 2013), a program that performs the AD method to a Fortran script.

This section plays a crucial part in this Master's Thesis because it shows the approach of combining the adjoint method with the ALM, which saves computing the adjoint variables for all constraint functions. More on this later. First, this section shows how the adjoint method is used for the SLSQP optimisation approach, described in section 3.4.1. Gradient-based optimisers, such as the SLSQP or BFGS, require gradients of the objective and, in the case of SLSQP,

gradients of the constraint functions. The total derivative of the objective function f along a linear approximation of $\boldsymbol{r}(\boldsymbol{p};\boldsymbol{x}) = \boldsymbol{0}$ with respect to the design variables $\boldsymbol{x}$ is

$$\frac{df}{d\boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{x}} - \boldsymbol{\Psi}_f^T \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}, \tag{3.65}$$

where someone can get the adjoint variables $\boldsymbol{\Psi}_f$ by solving the linear system

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \boldsymbol{\Psi}_f = \left[\frac{\partial f}{\partial \boldsymbol{p}}\right]^T. \tag{3.66}$$

We could get the four partial derivatives by AD—the computational cost of the backward-mode AD scales with the number of output functions. Hence, we can get $\partial f/\partial \boldsymbol{x}$ and $\partial f/\partial \boldsymbol{p}$ within one execution of the backward-mode AD script because f is a single scalar function. However, we must also determine the derivative of the residual functions $\boldsymbol{r}$. There are as many residual functions as state variables, thus $n_r = n_{DoF} + n_\Gamma$, where n_{DoF} are the number of DoF in the FEM model and n_Γ the number of vortex horseshoes in the LLT model. In this case, the backward-mode AD is not advantageous. However, neither the forward-mode AD is advantageous because the computational cost of the forward-mode AD scales with the function inputs $\boldsymbol{x}$ and $\boldsymbol{p}$. The number of design variables n_x and the number of state variables $n_p = n_r$ are both large integers.

Instead of calculating the full Jacobian matrices $\partial \boldsymbol{r}/\partial \boldsymbol{x}$ and $\partial \boldsymbol{r}/\partial \boldsymbol{p}$, we will exploit the way the adjoint method and the backward-mode AD are defined to solve the linear system 3.66. Tapenade AD provides a code that takes a vector $\boldsymbol{x}$ as input and outputs the results obtained through forward-mode or backward-mode AD. When we perform backward-mode AD, differentiating with respect to the input vector $\boldsymbol{x}$, we obtain the following result

$$\bar{\boldsymbol{x}}_f = \boldsymbol{J}_{\boldsymbol{a_x}}^T \cdot \bar{\boldsymbol{v}} \tag{3.67}$$

where $\boldsymbol{J}_{\boldsymbol{a_x}} = (\partial \boldsymbol{a}/\partial \boldsymbol{x})^T$ is the Jacobian matrix of the vector $\boldsymbol{a}$. $\boldsymbol{a}$ is a column vector of scalar functions. $\bar{\boldsymbol{v}}$ is the reversed seed vector. Let us define $\boldsymbol{a}$ as

$$\boldsymbol{a} = \begin{bmatrix} f & \boldsymbol{r}^T \end{bmatrix}^T, \tag{3.68}$$

where f is our objective function and $\boldsymbol{r}$ is a vector of all residual functions. Thus, the Jacobian matrix is

$$\boldsymbol{J}_{\boldsymbol{a_x}} = \left[\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right]^T = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_{n_x}} \\ \frac{\partial r_1}{\partial x_1} & \frac{\partial r_1}{\partial x_2} & \cdots & \frac{\partial r_1}{\partial x_{n_x}} \\ \frac{\partial r_2}{\partial x_1} & \frac{\partial r_2}{\partial x_2} & \cdots & \frac{\partial r_2}{\partial x_{n_x}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial r_{n_r}}{\partial x_1} & \frac{\partial r_{n_r}}{\partial x_2} & \cdots & \frac{\partial r_{n_r}}{\partial x_{n_x}} \end{bmatrix} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \end{matrix}}\right\} \frac{\partial f}{\partial \boldsymbol{x}} \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \\ a \end{matrix}}\right\} \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} \end{matrix} \tag{3.69}$$

If we carefully observe equation 3.67 and equation 3.69 again, we can see that we can express the result of the backward-mode AD by

$$\bar{\boldsymbol{x}}_f = \left[\frac{\partial f}{\partial \boldsymbol{x}}\right]^T \cdot \bar{v}_f + \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}\right]^T \cdot \bar{\boldsymbol{v}}_r, \tag{3.70}$$

where $\bar{v}_f$ is the seed value of the objective function and $\bar{\boldsymbol{v}}_r$ is the seed vector of the residual functions. We can get $\left[\partial f/\partial \boldsymbol{x}\right]^T$ by setting $\bar{v}_f = 1$ and $\bar{\boldsymbol{v}}_r = \boldsymbol{0}$. In the same way, we can express the result of the backward-mode AD by differentiating with respect to the state variables $\boldsymbol{p}$ by

$$\bar{\boldsymbol{p}}_f = \left[\frac{\partial f}{\partial \boldsymbol{p}}\right]^T \cdot \bar{v}_f + \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \cdot \bar{\boldsymbol{v}}_r. \tag{3.71}$$

We can get the right hand side of equation 3.66 by setting $\bar{v}_f = 1$ and $\bar{\boldsymbol{v}}_r = \boldsymbol{0}$ in equation 3.71. If we set $\bar{v}_f = 0$ and $\bar{\boldsymbol{v}}_r = \boldsymbol{b}$, we get the left hand side of equation 3.66 for an arbitrary vector $\boldsymbol{b}$ of equation 3.66. That means we can use a matrix-free linear system solver to solve for $\boldsymbol{\Psi}_f$ in equation 3.66. $\boldsymbol{\Psi}_f$ is determined by defining the residual function

$$\boldsymbol{r}_{\boldsymbol{\Psi}}(\boldsymbol{b}) = \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \boldsymbol{b} - \left[\frac{\partial f}{\partial \boldsymbol{p}}\right]^T, \tag{3.72}$$

where we need to find $\boldsymbol{b}$ such that $\boldsymbol{r}_{\boldsymbol{\Psi}}(\boldsymbol{b} = \boldsymbol{\Psi}_f) = \boldsymbol{0}$. Again, we use the non-linear Python solver from the SciPy library (VIRTANEN *et al.*, 2020), which is a modification of the Powell hybrid method to solve $\boldsymbol{r}_{\boldsymbol{\Psi}}(\boldsymbol{b} = \boldsymbol{\Psi}_f) = \boldsymbol{0}$. The method is briefly described in section 3.3 and can be found in (MORE *et al.*, 1980). Once the vector $\boldsymbol{b} = \boldsymbol{\Psi}_f$ is known, we can set $\bar{v}_f = 1$ and $\bar{\boldsymbol{v}}_r = -\boldsymbol{\Psi}_f$ in equation 3.70 and get

$$\bar{\boldsymbol{x}}_f = \left[\frac{\partial f}{\partial \boldsymbol{x}}\right]^T \cdot 1 + \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}\right]^T \cdot -\boldsymbol{\Psi}_f \tag{3.73}$$

$$= \left[\frac{df}{d\boldsymbol{x}}\right]^T, \tag{3.74}$$

according to equation 3.65. That means we got the total derivative for the optimisation procedure for our first function of interest (objective function). Next, we can move on and repeat the procedure with all constraint functions to get $\left[d\boldsymbol{h}/d\boldsymbol{x}\right]^T$ and $\left[d\boldsymbol{g}/d\boldsymbol{x}\right]^T$. In total, we need to execute this process $n_f + n_h + n_g = 1 + n_h + n_g$ times. n_f is the number of objective functions, n_h is the number of equality constraints and n_g is the number of inequality constraints.

The second optimisation approach is to merge all inequality constraints $\boldsymbol{g}$ in one aggregation function $\bar{g}_{KS}$ and is described in section 3.4.2. In this case, n_g reduces to one. Thus, in total, we need to solve $n_f + n_h + n_g = 1 + n_h + 1$ linear systems to get the adjoint variables.

The third optimisation approach uses the ALM described in section A.1.4.5 and section 3.4.3. We need to solve only one linear system for the ALM approach to get the adjoint variables. To solve the sub-problem, we need to compute the derivative of the augmented Lagrangian function along a linear approximation of $\boldsymbol{r}(\boldsymbol{p};\boldsymbol{x}) = \boldsymbol{0}$, which is $d\mathcal{A}/d\boldsymbol{x}$. Please note that it is not in any step required to know explicitly the derivative of the objective function $df/d\boldsymbol{x}$ or any of the constraint functions $d\boldsymbol{h}/d\boldsymbol{x}$ or $d\boldsymbol{g}/d\boldsymbol{x}$. This is a considerable advantage compared to the SLSQP method. The SLSQP method requires knowing $d\boldsymbol{h}/d\boldsymbol{x}$ and $d\boldsymbol{g}/d\boldsymbol{x}$ because it linearises the constraints at each sub-problem.

By the adjoint method, we can express this derivative of the augmented Lagrangian function as

$$\frac{d\mathcal{A}}{d\boldsymbol{x}} = \frac{\partial \mathcal{A}}{\partial \boldsymbol{x}} - \boldsymbol{\Psi}_{\mathcal{A}}^T \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}, \tag{3.75}$$

where we can get the adjoint variables $\boldsymbol{\Psi}_{\mathcal{A}}$ by solving the linear system

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \boldsymbol{\Psi}_{\mathcal{A}} = \left[\frac{\partial \mathcal{A}}{\partial \boldsymbol{p}}\right]^T. \tag{3.76}$$

The result of the backward-mode AD applied to the augmented Lagrangian function is

$$\bar{\boldsymbol{x}}_A = \left[\frac{\partial \mathcal{A}}{\partial \boldsymbol{x}}\right]^T \cdot \bar{v}_{\mathcal{A}} + \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}\right]^T \cdot \bar{\boldsymbol{v}}_r \tag{3.77}$$

$$\bar{\boldsymbol{p}}_{\mathcal{A}} = \left[\frac{\partial \mathcal{A}}{\partial \boldsymbol{p}}\right]^T \cdot \bar{v}_{\mathcal{A}} + \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \cdot \bar{\boldsymbol{v}}_r. \tag{3.78}$$

By setting $\bar{v}_{\mathcal{A}} = 1$ and $\bar{\boldsymbol{v}}_r = \boldsymbol{0}$ in equation 3.78 we get $\partial\mathcal{A}/\partial\boldsymbol{p}$, the right hand side of equation

3.76. Next, we set $\bar{v}_{\mathcal{A}} = 0$ and $\bar{\boldsymbol{v}}_r = \boldsymbol{b}$, where $\boldsymbol{b}$ is an arbitrary vector to solve equation 3.76 for $\boldsymbol{\Psi}_{\mathcal{A}}$. In the final step we can set $\bar{v}_{\mathcal{A}} = 1$ and $\bar{\boldsymbol{v}}_{\mathcal{A}} = -\boldsymbol{\Psi}_{\mathcal{A}}$ in equation 3.77 to get $\left[d\mathcal{A}/d\boldsymbol{x}\right]^T$ according to equation 3.75.

Table 3.1 summarises how many adjoint equations we need to solve for the three different optimisation approaches. For the SLSQP approach, we must provide the gradient of the objective function and all constraint functions. In this case, we need to solve $1+n_h+n_g$ adjoint equations to get the derivative of all functions: one adjoint equation for the objective function, n_h for the equality constraints and n_g for the inequality constraints. For the SLSQP KS optimisation approach, we aggregate the inequality constraint functions and reduce the number of inequality constraints to one. Thus, we only need to solve $1 + n_h + 1$ adjoint equations. We still need to solve one adjoint equation for the objective function and n_h for the equality constraints but only one for all inequality constraint functions because we aggregated all inequality constraints in a single scalar function. If we use the augmented Lagrangian approach, we only need to solve one adjoint equation at each iteration because we only need to know the derivative of $\mathcal{A}(\boldsymbol{x})$. The knowledge of the derivative of the objective function of the constraint functions is not required.

Optimisation approach	**SLSQP**	**SLSQP KS**	**ALM**
Requires to know derivatives of	$f(\boldsymbol{x})$, $\boldsymbol{h}(\boldsymbol{x})$, $\boldsymbol{g}(\boldsymbol{x})$	$f(\boldsymbol{x})$, $\boldsymbol{h}(\boldsymbol{x})$, $g_{KS}(\boldsymbol{x})$	$\mathcal{A}(\boldsymbol{x})$
Which function gives how many adjoint equations to solve?	$f(\boldsymbol{x}) \to 1$ $\boldsymbol{h}(\boldsymbol{x}) \to n_h$ $\boldsymbol{g}(\boldsymbol{x}) \to n_g$	$f(\boldsymbol{x}) \to 1$ $\boldsymbol{h}(\boldsymbol{x}) \to n_h$ $g_{KS}(\boldsymbol{x}) \to 1$	$\mathcal{A}(\boldsymbol{x}) \to 1$
Number of adjoint equations to solve at every algorithm iteration	$1 + n_h + n_g$	$1 + n_h + 1$	1

TABLE 3.1 – Table shows which optimisation approach requires the derivative of which function. A different number of adjoint equations must be solved to get the derivatives for each optimisation approach. For the SLSQP approach, we need to solve the most adjoint equations. We need to solve just a single adjoint equation for the ALM optimisation approach.

In the subsequent sections of this dissertation, we will delve into the exploration of both structural and aerostructural optimisations. The focus for structural optimisations will be on the variation of beam-truss thicknesses, which serve as the design variables. At the same time, the mass of the structures is employed as the objective function. As shown in section B.4 of the Appendix, this specific scenario allows for a more straightforward calculation of derivatives. This is due to the straightforward determination of partial derivatives. The incorporation of this method significantly diminishes the computational expenses. Nevertheless, the primary aim of this Master's thesis is to conduct a comparative analysis of the three distinct optimisation methodologies (SLSQP, SLSQP KS and ALM). In most instances, this simplified approach to get the derivatives is not applicable. Therefore, while acknowledging the existence of a more rudimentary method for derivative calculation in structural optimisation, this study deliberately refrains from employing it, to maintain a comprehensive assessment of the optimisation techniques under consideration.

3.6 Optimisation problems

We will test the performance of the three optimisation approaches on an atmospheric satellite aircraft called Helios aircraft. The Helios aircraft has multiple configurations; in this Master's Thesis, we will optimise the Pathfinder-Plus configuration, shown in figure 3.10a. The Helios aircraft is a flying wing developed under the National Aeronautics and Space Administration (NASA) Environmental Research Aircraft and Sensor Technology project. The project aimed to demonstrate a sustainable aircraft and create an aeroplane that can fly non-stop. The non-

stop flying goal is not possible with the Pathfinder-Plus configuration. Solar panels on the aircraft's wings power the eight brushless direct-current electric motors. The aircraft is equipped with backup batteries, which provide enough power to power the motors for several hours, depending on the flight conditions, but not long enough to fly the whole night without sunlight. The atmospheric satellite aircraft is remotely controlled on Earth and can be used for various Earth and atmospheric science missions, including storm tracking studies, atmospheric sampling, spectral imaging for agriculture, natural resources monitoring and pipeline monitoring. However, the aircraft can also serve as relay platforms for telecommunications systems (NASA, 2002).

The Helios aircraft is made of composite materials such as carbon fiber, graphite epoxy, Kevlar, styrofoam, and a thin, transparent plastic skin (CURRY, 2002). In QASTRO, it is not possible to model those materials. The finite element formulation in QASTRO can only model isotropic materials. The main tubular wing spar is made of carbon fibre. However, we will model the tubular wing spar by n_{beams} aligned aluminium beam-truss elements and assume that the tubular wing spar represents the whole structure. Table 3.2 lists the material properties of the aluminium alloy used to model the tubular wing spar. The circular beam-truss elements have an initial thickness of $\boldsymbol{t} = 5\,mm$ and a slightly lower diameter than the thickness of the airfoil. The airfoil's maximum thickness is $0.137\,c = 0.3341\,m$ and ,the beam-truss elements' diameter is $2r = 0.2926\,m$.

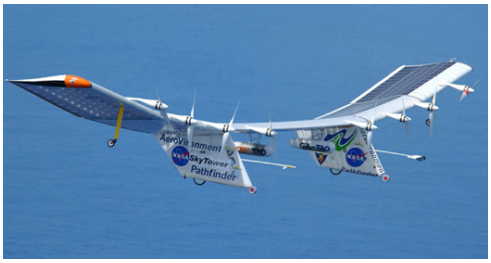
Parameter	**Symbol**	**Value**	**Unit**
Elasticity modulus	E	$73.1 \cdot 10^9$	$[N/m^2]$
Shear modulus	G	$27.48 \cdot 10^9$	$[N/m^2]$
Poisson's ratio	ν	0.33	$[-]$
Yield strength	σ_Y	$324 \cdot 10^6$	$[N/m^2]$
Material density	ρ_{mat}	2780	$[kg/m^3]$

TABLE 3.2 – Assumed Helios Pathfinder-Plus aircraft material specifications. We assume an aluminium alloy called 2024-T4; 2024-T351. The material properties are from MatWeb (2023).

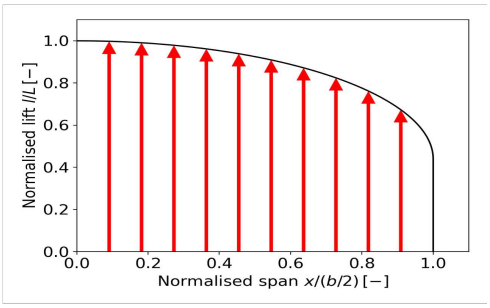
For the Helios aircraft, we will assume a fixed mass of $100\,kg$ for the solar panels and the engines. Furthermore, we assume a battery mass of $25\,kg$ and an additional $30\,kg$ for technical equipment. Table 3.3 summarises the assumed masses.

	Includes	**Value**	**Unit**
Fixed mass	Solar panels, engines	100	$[kg]$
Battery mass	Battery	25	$[kg]$
Technical equipment	e.g. Telecommunication system	30	$[kg]$

TABLE 3.3 – Helios Pathfinder-Plus aircraft fixed masses estimation.



(a) Pathfinder-Plus[5]



(b) Lift distribution

FIGURE 3.10 – Helios aircraft Pathfinder-Plus. Lift distribution of rectangular wing according to Schrenk (1940). Red arrows indicate distributed lift force.

Table 3.4 lists the Pathinder-Plus wing dimensions. Note that the aircraft wings are simply rectangular. The flight altitude at cruise conditions is about $18.30\,km$ $(60,000\,ft)$. At this altitude, the flight speed is about $28.65\,m/s$. We will assume an air density of $\rho_{\text{Air}} = 0.1225\,kg/m^3$ and a dynamic viscosity of $\mu = 1.432 \cdot 10^{-5}\,kg/(m\,s)$, which are the air properties at around the flight altitude according to the United States standard atmosphere (DRELA, 2014).

Parameter	Symbol	Value	Unit
Span width	b	36.3	$[m]$
Chord length	c	2.434	$[m]$

TABLE 3.4 – Pathfinder-Plus aircraft wing dimensions (NASA, 2002).

Parameter	Symbol	Value	Unit
Free stream	V_∞	28.65	$[m/s]$
Air density	ρ_{Air}	0.1225	$[kg/m^3]$
Viscosity	μ	$1.432 \cdot 10^{-5}$	$[kg/(m\,s)]$

TABLE 3.5 – Pathfinder-Plus aircraft flight parameters.

We will perform a structural and an aerostructural optimisation on the Helios aircraft. For the structural optimisation, we do not couple the structural model with the aerodynamic model of the aircraft. The structural and the aerostructural optimisation definitions are described in section 3.6.1 and 3.6.2, respectively.

3.6.1 Structural optimisation problems

First, we will perform a structural optimisation on the Helios aircraft without coupling the aerodynamic loads with the structural displacements. The aim is to minimise the weight of the aircraft by solving

$$\begin{aligned} \min_{\boldsymbol{x}=\boldsymbol{t}} \quad & f(\boldsymbol{x}=\boldsymbol{t}) = \frac{1}{m_{w0}} \sum_{i}^{n_{\text{beams}}} \rho_{\text{mat}\,i}\, L_i\, A_i = \frac{1}{m_{w0}} \sum_{i}^{n_{\text{beams}}} 2\pi \rho_{\text{mat}\,i}\, L_i\, r_i\, t_i \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{p}=\boldsymbol{d}) = -\boldsymbol{m}(\boldsymbol{p}=\boldsymbol{d}) \leq \boldsymbol{0},\, \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}=\boldsymbol{d}; \boldsymbol{x}=\boldsymbol{t}) = \boldsymbol{K}\,\boldsymbol{d} - LF \cdot \boldsymbol{F} = \boldsymbol{0}, \end{aligned} \tag{3.79}$$

where the semicolon detonates that the design variables $\boldsymbol{x}$ are fixed when the residual functions $\boldsymbol{r}$ are solved for the state variables $\boldsymbol{p}$. m_{w0} is for normalisation purposes, ρ_{mat} is the material density, L_i and A_i is the length and the cross-sectional area of the beam-truss element i. r_i and t_i is the radius and the wall thickness of the beam-truss element, see figure 3.3. The failure margins $\boldsymbol{m}(\boldsymbol{p})$ are given in equation 3.16. LF is the load factor which is equal to three, $\boldsymbol{K}$ is the stiffness matrix, $\boldsymbol{d}$ is the displacement/ rotation vector and $\boldsymbol{F}$ is the force/ moment vector.

We model only half of the aircraft because the aircraft is symmetric. The design variables are the beam-truss element thicknesses $\boldsymbol{x} = \boldsymbol{t}$ with the bound $0.5 \leq \boldsymbol{t}_0 \leq 5\ mm$ for each beam-truss element. Table 3.4 and 3.5 list the wing dimensions and the flight conditions, respectively. Given those specifications we can determine the lift distribution according to Schrenk's distribution (SCHRENK, 1940), which gives us the force/ moment vector $\boldsymbol{F}$. Morse (1944) gives step-by-step instructions to determine Schrenk's lift distribution, which is used as a baseline for determining the Helios Pathfinder-Plus lift distribution. Figure 3.10b shows Schrenk's lift approximation for the Helios aircraft wing. After determining Schrenk's normalised lift approximation, we need to determine the lift force of the aircraft to get the dimensional lift distribution. The lift force with an assumed cruise lift coefficient $C_L = 0.5$ of one wing is

$$L = \frac{1}{2}\, C_L\, \rho_{\text{Air}} V_\infty^2\, c\, b/2 = 1112.5\,[N], \tag{3.80}$$

which means that both wing's lift is $2225.0\,N$. In other terms, the generated lift can carry $226.81\,kg$. It is challenging to find the exact weight specifications of the Pathfinder-Plus, but

[5] `https://www.avinc.com/innovative-solutions/hale-uas` accessed at 27.08.23

the Pathfinder configuration, which has a slightly smaller wingspan compared to the Pathfinder-Plus configuration, has an empty weight of $226.80\,kg$ (FLITTIE; CURTIN, 1998). Consequently, the total lift is in a realistic range.

Given the lift distribution of the Helios aircraft wing, we can determine the equivalent force and moment at each node. The equivalent forces and moments are determined such that the displacement of the beam-truss elements is equivalent to the displacement due to the distributed load. Please note that the distributed load is assumed to vary linearly along a beam-truss element. Logan (2012) shows how to get the equivalent forces and moments for a linearly varying load on a beam-truss element. Furthermore, we consider a load factor of three to ensure that the Helios aircraft does not fail for any speeds higher than the cruise speed $V_\infty = 28.65\,m/s$. That means that the Helios aircraft model is loaded with equivalent forces and moments, three times higher than due to the lift distribution at cruise speed. Figure 4.2 shows how the Helios aircraft is modelled. The black lines are the horizontally aligned beam-truss elements. The blue area represents the wing, and the red arrows are the lift forces.

3.6.2 Aerostructural optimisation problems

For the aerostructural optimisation problem, we consider again the Helios aircraft shown in figure 3.10a. The difference to the structural optimisation of the Helios aircraft described in section 3.6.1 is that in the aerostructural optimisation we couple the structural model with the aerodynamic model to estimate the lift distribution. Figure 3.12 shows the initial design of the Helios aircraft model. The Helios aircraft utilises an airfoil called LA2573A. The properties of this airfoil are used for each wing section and are listed in table 3.6. The parameters are obtained by Xfoil (DRELA, 1989) with the Reynolds number

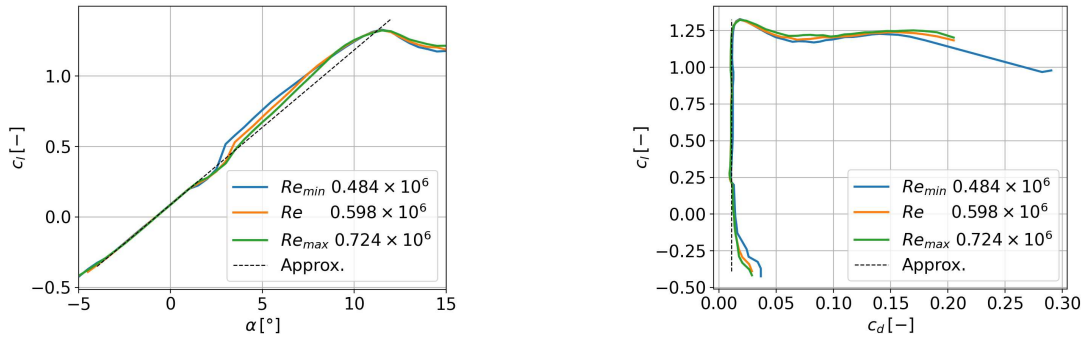
$$Re = \frac{\rho_{\text{Air}} V_\infty c}{\mu} = \frac{0.1225 \cdot 28.65 \cdot 2.4384}{1.432 \cdot 10^{-5}} = 0.598 \cdot 10^6\,[-]. \tag{3.81}$$

For the aerostructural optimisation, we allow to increase and decrease the chord length and the flight speed by 10 %. That means the Reynolds number changes and, consequently, the airfoil properties. The maximum and minimum Reynolds number we can obtain are $Re_{max} = 0.724 \cdot 10^6$ and $Re_{min} = 0.484 \cdot 10^6$, respectively. Figure 3.11a shows the lift curve for all three Reynolds numbers. One can see that the curves are sufficiently close to each other such that we can assume that the airfoil parameters are equal for all Reynolds numbers. The dashed line indicates the approximated lift coefficient c_l with the airfoil parameters of table 3.6. Figure 3.11b shows the drag polar of the LA2573A airfoil. QASTRO can approximate the drag polar of an airfoil by a quadratic approximation. However, for the LA2573A airfoil, it seems to be a good estimation to take a constant drag coefficient c_d for all lift coefficients. The dashed line in figure 3.11b indicates the approximated constant minimum drag coefficient for the Helios aircraft model.

Parameter	Symbol	Value	Unit
Zero lift coefficient	c_{l0}	5.15	$[^\circ]$
Lift slope	$c_{l\alpha}$	$2\,\pi$	$[1/rad]$
Zero moment coefficient	c_{m0}	0.01	$[-]$
Moment curve slope	$c_{m\alpha}$	0	$[1/rad]$
Minimum drag coefficient	$c_{d\,\min}$	0.011	$[-]$

TABLE 3.6 – Helios Pathfinder-Plus aircraft airfoil (LA2573A) specifications.

Table 3.7 lists the initial design variables and their bounds for the aerostructural optimisations. The sum of the upper bound from the twist and angle of attack is 12°. This is the maximum angle of attack before stall occurs; see the lift curve in figure 3.11a. The bounds of the angle of attack are chosen so that the pilot has a range of 10° to control the aircraft. Figure



(a) Lift curve

(b) Drag polar

FIGURE 3.11 – Lift curve and drag polar of airfoil LA2573A for different Reynolds numbers.

3.12 shows the Helios aircraft model in QASTRO. The black lines are the horizontally aligned beam-truss elements.

Parameter	Symbol	Initial value	Upper bound	Lower bound	Unit
Twist	$\boldsymbol{\alpha}_0$	3	5	-2	$[°]$
Beam-truss thickness	$\boldsymbol{t}$	5	5	0.5	$[mm]$
Chord length	$\boldsymbol{c}$	2.4384	$1.1\,c$	$0.9\,c$	$[m]$
Beam-truss location	$\boldsymbol{x}_{\text{Beam}}$	50% chord line	trailing edge	leading edge	$[m]$
Free stream	V_∞	28.65	$1.1\,V_\infty$	$0.9\,V_\infty$	$[m/s]$
Angle of attack	α	2 or 5	7	-3	$[°]$

TABLE 3.7 – Initial design variables and its bounds. Bold symbols indicate that the parameter is discretised along the wing span. The initial angle of attack is 5° for the high load condition, otherwise 2°.

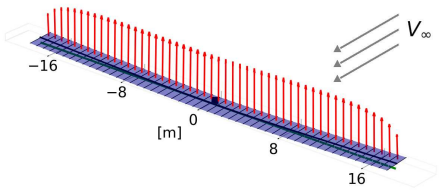


FIGURE 3.12 – Initial Helios aircraft design. Red arrows indicate lift forces. Green dashed line shows undeformed tubular wing spar. The initial design is infeasible.

The initial mass of the aircraft, including the tubular wing spar of the Helios aircraft, is $541.52\,kg$, but the configuration is infeasible because the lift constraint(s) are not satisfied. In order to satisfy the lift constraint(s), the aircraft's weight reduces to at least $250\,kg$, which is in the weight range of the Helios Pathfinder-Plus. We can further decrease the weight of the aircraft by optimising the aircraft. Again, we want to design the Helios pathfinder-Plus aircraft with a load factor of three. The optimisation problem for the aerostructural optimisation reads

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{p}) = \boldsymbol{0},\, \boldsymbol{g}(\boldsymbol{p}) \leq \boldsymbol{0},\, \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u}, \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}. \end{aligned} \tag{3.82}$$

We will consider two different optimisations with each different constraints. The first optimisation extrapolates the failure margins for $LF = 3$, while the second case computes two separate

lift distributions: one for the cruise condition and the second for the high load condition. For the first optimisation problem, the constraint and residual functions are

Level flight constraint: $\boldsymbol{h}(\boldsymbol{x}) = L(\boldsymbol{p})/W - 1,$ Failure margin constraint: $\boldsymbol{g}(\boldsymbol{x}) = -(LF \cdot \boldsymbol{m}(\boldsymbol{p}) - LF + 1),$ Residual functions: $\boldsymbol{r}(\boldsymbol{p};\boldsymbol{x}) = \begin{Bmatrix} \boldsymbol{r}_{struc}(\boldsymbol{p};\boldsymbol{x}) \\ \boldsymbol{r}_{aeros}(\boldsymbol{p};\boldsymbol{x}) \end{Bmatrix},$

where $\boldsymbol{m}(\boldsymbol{p})$ are the failure margins and $LF = 3$ is the load factor. L is the total lift force and W is the total weight of the aircraft. $\boldsymbol{r}_{struc}$ and $\boldsymbol{r}_{aero}$ are the residual functions of the structural and aerodynamic model given in equation 3.9 and 3.25, respectively. In aerostructual optimisation, the state variables are $\boldsymbol{p}^T = \begin{bmatrix} \boldsymbol{d}^T & \boldsymbol{\Gamma}^T \end{bmatrix}$ where $\boldsymbol{d}$ are the displacements of the structure and $\boldsymbol{\Gamma}$ are the circulation strengths of the horseshoes. We will consider two sets of design variables - one reduced set $\boldsymbol{x}_{\text{reduced}}$ and one complete set $\boldsymbol{x}_{\text{complete}}$. The reduced set includes the beam-truss element thicknesses $\boldsymbol{t}$; the wing twist $\boldsymbol{\alpha}_0$ and the angle of attack α.

$$\boldsymbol{x}_{\text{reduced}} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{\alpha}_0 & \alpha \end{bmatrix}^T, \tag{3.83}$$

where the design variables are row vectors. The complete set of design variables includes the beam-truss element thicknesses $\boldsymbol{t}$, the wing twist $\boldsymbol{\alpha}_0$, the angle of attack α, the chord lengths $\boldsymbol{c}$, the free stream velocity V_∞ and the beam-truss element location in the x - direction $\boldsymbol{x}_{\text{Beam}}$.

$$\boldsymbol{x}_{\text{complete}} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{\alpha}_0 & \alpha & \boldsymbol{c} & V_\infty & \boldsymbol{x}_{\text{Beam}} \end{bmatrix}^T, \tag{3.84}$$

where the design variables are row vectors.

Next, let us define the constraint for the second optimisation problem. Usually, we do not want to optimise the aircraft in this high-load condition. We want to optimise the cruise condition. In the second optimisation problem, we optimise the aircraft's cruise condition. For this, we will create two separate Helios aircraft models. We will call the cruise condition *condition* 1 and the flight condition with a load factor of three we will call *condition* 2. For the second flight condition, we require that the aircraft be able to generate a lift force that is equal to the weight multiplied by the load factor LF. By this, we ensure that the aircraft will not fail even if the lift force is LF times higher than in a cruise condition. Furthermore, we ensure the aircraft can generate enough lift to reach the desired manoeuvring condition. It does not make sense to oversize the wing structure if the aircraft can not create the lift forces, which can cause the wing structure to fail. The constraint functions and residual function of the second optimisation problem where we optimise the cruise conditions read

Level flight constraints: $\boldsymbol{h}(\boldsymbol{x}) = \begin{Bmatrix} L_1(\boldsymbol{p})/W - 1 \\ L_2(\boldsymbol{p})/(LF \cdot W) - 1 \end{Bmatrix},$ Failure margin constraints: $\boldsymbol{g}(\boldsymbol{x}) = \begin{Bmatrix} -\boldsymbol{m}_1(\boldsymbol{p}) \\ -\boldsymbol{m}_2(\boldsymbol{p}) \end{Bmatrix},$ Residual functions: $\boldsymbol{r}(\boldsymbol{p};\boldsymbol{x}) = \begin{Bmatrix} \boldsymbol{r}_{struc\,1}(\boldsymbol{p};\boldsymbol{x}) \\ \boldsymbol{r}_{aeros\,1}(\boldsymbol{p};\boldsymbol{x}) \\ \boldsymbol{r}_{struc\,2}(\boldsymbol{p};\boldsymbol{x}) \\ \boldsymbol{r}_{aeros\,2}(\boldsymbol{p};\boldsymbol{x}) \end{Bmatrix},$

where the subscripts 1 and 2 indicate that the functions are from the first or second flight condition, respectively. The complete set of design variables for the optimisation with two flight conditions is

$$\boldsymbol{x}_{\text{complete 1, 2}} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{\alpha}_0 & \alpha_1 & \alpha_2 & \boldsymbol{c} & V_{\infty 1} & V_{\infty 2} & \boldsymbol{x}_{\text{Beam}} \end{bmatrix}^T, \tag{3.85}$$

where the design variables are row vectors and the subscripts 1 and 2 indicate the design variable of the first or second flight condition, respectively. Please note that for the SLSQP KS approach, we aggregate the inequality constraints in one KS function for each condition, giving us two KS functions in total. With two flight conditions, it is impossible to reach a feasible

solution by optimising with respect to the reduced set of design variables. Thus, we will only consider the complete set of design variables for the optimisation with two conditions.

Given the optimisation problems for a single flight condition and two flight conditions, let us define the objective function for the optimisation problems. Possible objective functions for the Helios aircraft are the weight, the endurance and the power ratio. The power ratio is the ratio between the required power to operate the aircraft and the generated energy by the solar panels. The objective function to minimise the weight W of the aircraft reads

$$f(\boldsymbol{x})_{\text{Weight}} = \frac{1}{W_0} W(\boldsymbol{x}), \tag{3.86}$$

where $f(\boldsymbol{x}_0)_{\text{Weight}} = W_0$ is the initial weight of the aircraft. The objective function to maximise the endurance is

$$f(\boldsymbol{x})_{\text{Endurance}} = -\frac{1}{t_{\text{Endurance 0}}} \frac{m_{\text{Battery}}\, c_{\text{Battery}}}{g\, D(\boldsymbol{x})\, V_\infty}, \tag{3.87}$$

where $f(\boldsymbol{x}_0)_{\text{Endurance}} = t_{\text{Endurance 0}}$ is the endurance of the initial design, $g = 9.81\, m/s^2$ is the gravity, $m_{\text{Battery}} = 25\, kg$ is the battery mass and c_{Battery} is the specific energy of the battery. We assume a specific energy of the battery of $c_{\text{Battery}} = 250\, Wh/kg$. Lastly, the objective function to decrease the power ratio is

$$f(\boldsymbol{x})_{\text{Power ratio}} = \frac{1}{P_{\text{ratio 0}}} \frac{D(\boldsymbol{x})\ V_\infty}{c_{\text{Solar panel}}\, S(\boldsymbol{x})}, \tag{3.88}$$

where $f(\boldsymbol{x}_0)_{\text{Power ratio}} = P_{\text{ratio 0}}$ is the initial power ratio, $c_{\text{Solar panel}} = 205.83 W/m^2$ is the produced power per square metre of the solar panels and $S(\boldsymbol{x})$ is the surface area of the solar panels.

3.7 Optimisation with multiple conditions

QASTRO is designed to consider multiple flight conditions during the optimisation. Usually, we want to optimise an aircraft for cruise conditions but require that the aircraft not fail in some high-load conditions. The approach in QASTRO is to create multiple (aero-) structural models representing the different (flight) conditions. For the SLSQP and SLSQP KS optimisation approaches, there are no difficulties in adding constraint functions from different models because we compute the derivative of every constraint function anyway. If we consider n_{cond} different conditions, we need to solve n_{cond} times the number of adjoint equations as listed in table 3.1. However, in theory, it is still possible to solve a single adjoint equation for the ALM approach, although we consider multiple conditions. This section explains the difficulties by considering multiple conditions using the ALM optimisation approach if we want to solve the adjoint equation only once. It will help to understand this section while keeping the workflow of QASTRO, depicted in figure B.1, in mind.

For the ALM approach, it becomes more challenging to define our desired augmented Lagrangian function, which reads

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right\|_2^2 + \left\| \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle \right\|_2^2 \right), \tag{3.89}$$

where the equality and inequality constraint functions are all constraint equations for all conditions. However, the construction of the augmented Lagrangian function within QASTRO is fixed at the Fortran level, limiting flexibility. A composite function is proposed, computed on the Python level, formulated as a weighted sum of individual augmented Lagrangian functions

of each considered condition, to circumvent this:

$$\mathcal{A}(\boldsymbol{x}) = \sum_{i}^{n_{cond}} w_{\mathcal{A},i}\mathcal{A}_i, \tag{3.90}$$

where $w_{\mathcal{A},i}$ is the weight of the augmented Lagrangian function and each $\mathcal{A}_i$ is defined as

$$\mathcal{A}_i = w_{f,i}f_i(\boldsymbol{x}) + \frac{1}{2}\rho_{k,i}\left(\left\|\boldsymbol{h}_i(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_{k,i}}{\rho_{k,i}}\right\|_2^2 + \left\|\left\langle \boldsymbol{g}_i(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_{k,i}}{\rho_{k,i}}\right\rangle\right\|_2^2\right) \tag{3.91}$$

$$= w_{f,i}f_i(\boldsymbol{x}) + \frac{1}{2}\rho_{k,i}\left(\left\|\tilde{\boldsymbol{h}}_i(\boldsymbol{x},\boldsymbol{\lambda}_{k,i})\right\|_2^2 + \left\|\tilde{\boldsymbol{g}}_i(\boldsymbol{x},\boldsymbol{\mu}_{k,i})\right\|_2^2\right) \tag{3.92}$$

If we set $w_{\mathcal{A},i} = 1, \forall i$, $w_{f,1} = 1$ and $w_{f,i} = 0, \forall i \neq 1$, we get the desired augmented Lagrangian function presented in equation 3.89.

Next, let us focus on the gradient evaluation. Theoretically, when we aggregate the conditions, we revert to a singular augmented Lagrangian function. This consolidation suggests that, in an ideal scenario, only one residual function remains to solve to get the adjoint variables to compute the gradient $\nabla\mathcal{A}$, simplifying the process significantly. Despite the theoretical simplification, QASTRO's programming structure introduces significant practical limitations. In QASTRO, the augmented Lagrangian functions for individual conditions are defined at the Fortran level, offering high computational efficiency. Furthermore, by defining the augmented Lagrangian functions on the Fortran level we can utilise Tapanade AD to get the derivatives of the augmented Lagrangian functions. However, the operation to sum these functions and form the singular augmented Lagrangian function occurs at the Python level. This separation between the computational levels necessitates solving an adjoint residual function for each condition to obtain the required adjoint variables, contrary to the theoretical model of solving a single adjoint equation.

Currently, we get the gradient of the augmented Lagrangian function $\nabla\mathcal{A}$ by summing up all gradients $\nabla\mathcal{A}_i$ of all conditions. In the following, we would like to show that this approach leads indeed to the correct gradient of the augmented Lagrangian function (eq. 3.89), which considers all conditions. We consider only equality constraints to increase readability - the extension to inequality constraints is left to the reader.

$$\nabla\mathcal{A}(\boldsymbol{x}) = \sum_{i=1}^{n_{cond}} \nabla\mathcal{A}(\boldsymbol{x})_i \tag{3.93}$$

$$=\nabla f_1(\boldsymbol{x}) + \rho_k\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{x}}\tilde{\boldsymbol{h}}_1(\boldsymbol{x},\boldsymbol{\lambda}_k) + \ldots + \rho_k\frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{x}}\tilde{\boldsymbol{h}}_i(\boldsymbol{x},\boldsymbol{\lambda}_k) + \ldots + \rho_k\frac{\partial \boldsymbol{h}_{n_{cond}}}{\partial \boldsymbol{x}}\tilde{\boldsymbol{h}}_{n_{cond}}(\boldsymbol{x},\boldsymbol{\lambda}_k)$$

which we can simplify to

$$\nabla\mathcal{A}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}}f_1(\boldsymbol{x}) + \rho_k\left(\boldsymbol{J_{h_x}}^T\left(\boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k}\right)\right) = \nabla_{\boldsymbol{x}}f_1(\boldsymbol{x}) + \rho_k\left(\boldsymbol{J_{h_x}}^T\tilde{\boldsymbol{h}}(\boldsymbol{x},\boldsymbol{\lambda}_k)\right), \tag{3.94}$$

where $\boldsymbol{J_{h_x}}^T$ is the transpose of the Jacobian of $\boldsymbol{h}$ with respect to $\boldsymbol{x}$ and is defined as

$$\boldsymbol{J_{h_x}}^T = \begin{bmatrix}\frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{x}} & \frac{\partial \boldsymbol{h}_2}{\partial \boldsymbol{x}} & \ldots & \frac{\partial \boldsymbol{h}_{n_{cond}}}{\partial \boldsymbol{x}}\end{bmatrix} = \begin{bmatrix}\frac{\partial \boldsymbol{h}_1}{\partial x_1} & \frac{\partial \boldsymbol{h}_2}{\partial x_1} & \ldots & \frac{\partial \boldsymbol{h}_{n_{cond}}}{\partial x_1} \\ \frac{\partial \boldsymbol{h}_1}{\partial x_2} & \frac{\partial \boldsymbol{h}_2}{\partial x_2} & \ldots & \frac{\partial \boldsymbol{h}_{n_{cond}}}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \boldsymbol{h}_1}{\partial x_k} & \frac{\partial \boldsymbol{h}_2}{\partial x_k} & \ldots & \frac{\partial \boldsymbol{h}_{n_{cond}}}{\partial x_k}\end{bmatrix}, \tag{3.95}$$

where we consider k design variables. Comparing this result with the gradient of the aug-

mented Lagrangian function given in the Theoretical Background chapter (eq. A.83) shows that equation 3.93 gives the correct gradient of the augmented Lagrangian function given in equation 3.89.

In summary, this section showed that it is possible to consider multiple (flight) conditions in QASTRO with the ALM approach. However, we still do not use the full potential of the ALM approach because we need to solve as many residual functions as conditions to get the adjoint variables at every iteration.

3.8 Outline of comparison strategy for optimisation approaches

In this thesis, we want to compare the performance of the SLSQP, SLSQP KS and the ALM approach. This section aims to give the reader an outline of the comparison strategy. There are various aspects which we want to compare. The optimisation results in the following chapter are wisely structured to focus on different aspects of the comparison sequentially. We want to constantly increase the complexity throughout the result chapter, starting with structural optimisation and ending with aerostructual optimisation with two flight conditions. We will increase the number of inequality constraints by increasing the model's discretisation. Structural optimisations focus on comparing optimisation durations, whereas aerostructural optimisations concentrate on the optimisation results themselves.

For the structural optimisations, we first analyse the three optimisation approaches on the Helios aircraft model with 60 beam-truss elements. For the SLSQP KS approach, the main focus lies on the influence of the ρ_{KS} value. The parameter ρ_{KS} is critical for effectively approximating inequality constraints. We want to answer two questions: How large does ρ_{KS} need to be that we can consider the SLSQP KS solution close enough to the SLSQP solution, and what is the influence of ρ_{KS} on the optimisation time?

For the ALM approach, we want to analyse if the ALM approach and the SLSQP give identical results. Considering that both methods do not approximate the inequality constraints, one can expect that both methods give the same result. Furthermore, we want to analyse the influence on the optimisation duration of each ALM optimisation parameter: ρ_0, γ, and r.

Our investigation journey will continue with an analysis of the usage of B-splines. B-splines can reduce the number of design variables and, hence, reduce the complexity of the optimisation problem. We want to answer how much B-splines can reduce the optimisation time.

In aerostructural optimisations, emphasis shifts to comparing the outcomes, $f(\boldsymbol{x}^*)$, rather than the duration of optimisations. The aim is to progressively increase model complexity and evaluate the results of different optimisation methods, particularly examining the effects of ρ_{KS} in the SLSQP KS method and the impacts of ρ_0, γ, and r in the ALM approach.

First, we will consider a simple Helios aircraft model with 14 beam-truss elements and 20 horseshoe vortices and the reduced set of design variables. Then, we will increase the number of design variables. In this comparison, we want to focus on two separate analyses. We want to analyse how the three optimisation approaches can handle the increased number of design variables and analyse the potential difficulties. Furthermore, we want to analyse how the optimisation approaches deal with an increasing number of inequality constraint functions.

The final comparison of the three optimisation approaches will be with two flight conditions. By considering two flight conditions, we decrease the feasible region, which increases the complexity of the optimisation problem. Furthermore, each flight condition introduces its inequality constraints to the optimisation problem, resulting in a constraint-rich aerostructural optimisation. We will increase the discretisation to analyse which optimisation approach best handles a large number of inequality constraints considering a highly complex optimisation problem.

4 Results

In this chapter, we explore the results of our study, building upon the methodologies outlined in Chapter 3. We thoroughly examine the results, specifically examining and comparing the SLSQP, SLSQP KS, and ALM optimisation approaches. All optimisations are executed on a Macbook Pro M2, 8 GB. Table 4.1 summarises the optimality conditions of each optimisation approach. The optimality condition $f(\boldsymbol{x})_{tol} \leq 10^{-6}$ implies that

$$|f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k-1})| \leq 10^{-6}. \tag{4.1}$$

The optimality condition of the main optimisation problem of the ALM is

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \rho)_{tol} = |\mathcal{A}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) - \mathcal{A}(\boldsymbol{x}_{k-1}, \boldsymbol{\lambda}_{k-1}, \boldsymbol{\mu}_{k-1}, \rho_{k-1})| \leq 10^{-6}, \tag{4.2}$$

whereas the optimality condition of the sub-problem is

$$\mathcal{A}(\boldsymbol{x})_{tol} = |\mathcal{A}(\boldsymbol{x}_j, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) - \mathcal{A}(\boldsymbol{x}_{j-1}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)| \leq 10^{-6}, \tag{4.3}$$

where j is the index of the sub-optimisation problem.

SLSQP	$f(\boldsymbol{x})_{tol} \leq 10^{-6}$	
SLSQP KS	$f(\boldsymbol{x})_{tol} \leq 10^{-6}$	
ALM	Main	$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \rho)_{tol} \leq 10^{-6}$ and $f(\boldsymbol{x})_{tol} \leq 10^{-6}$
	Sub	$\mathcal{A}(\boldsymbol{x})_{tol} \leq 10^{-6}$

TABLE 4.1 – Overview of optimality condition of each optimisation approach.

4.1 Structural optimisation problem

Section 4.1 delves into the structural optimisation of the Helios aircraft. It is crucial to note that here, aerodynamic and structural models are not coupled; instead, we estimate aerodynamic loads independently using Schrenk's model. Please note that, due to symmetry, we model only half of the aircraft for the structural optimisations. In this section, we will compare the SLSQP, the SLSQP KS and the ALM optimisation approaches. Each method's impact on the optimisation process, particularly in terms of solution quality and efficiency, is thoroughly analysed, offering crucial insights into their respective advantages and limitations in structural optimisation scenarios. The main focus lies on the performance of the optimisation methods regarding the discretisation of the Helios aircraft model.

It is important to note that the optimisation time is not measured but calculated. The methodology regarding the time computation can be found in Appendix B.5.

This section is split into two subsections. In the first subsection, we optimise the thickness of the beam-truss elements directly. We will refer to the beam-truss elements as the frame elements of the aircraft. In the second subsection, we use B-splines to describe the thickness variation of the frame elements along the wingspan. Instead of optimising the thickness of the

frame elements, we optimise the control points of the B-splines. Section A.1.8 in the Theoretical Background chapter explains how QASTRO uses B-splines for optimisation.

4.1.1 Structural optimisation without B-splines

Section 4.1.1 delves into the structural optimisation results of the Helios aircraft without the use of B-splines. This means that the design variables for the optimisers are the thicknesses of each frame element. The number of design variables equals the number of frame elements in the model. In the following three subsections, we will investigate the performance of each optimisation approach individually. Furthermore, we will compare each optimisation approach with each other.

4.1.1.1 SLSQP approach analysis

The SLSQP approach serves as a baseline for our analysis because it considers each constraint separately, therefore not suffering from the approximations generated by the aggregation approach. This makes the SLQP approach a reference point for comparing other optimisation strategies. However, before diving into the performance of the optimisation algorithm, the author wants to familiarise the reader with the solution of the structural optimisation from the Helios aircraft.

The optimisation formulation is stated in equation 3.79 in the Methodology chapter. The design variables are solely the thickness $\boldsymbol{t}$ of the tubular wing spar. The objective of the optimisation procedure is to reduce the weight of the Helios aircraft. Figure 4.1 presents the SLSQP-optimised structure of the Helios wing using a 60-frame element model. Notably, the thickness distribution varies significantly along the wingspan. The largest thickness is observed at the wing root, a structural necessity to support the higher stress concentrations common in this area. Conversely, towards the wing tips, the thickness reaches its lower bound, reflecting the reduced stress and the necessity for lighter construction to maintain aerodynamic efficiency and structural integrity. This gradation in thickness from root to tip is integral to understanding the wing's structural optimisation.

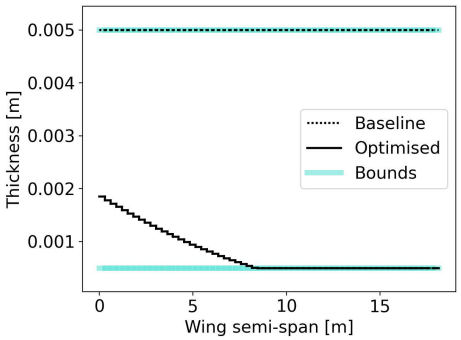
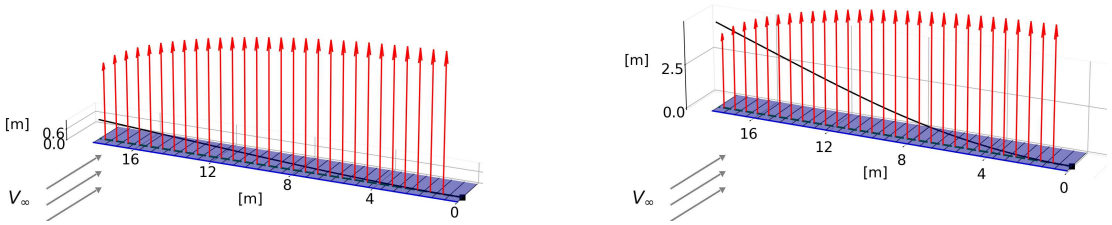


FIGURE 4.1 – Helios aircraft tubular wing spar thickness, before and after structural weight optimisation by SLSQP method (without KS function) or ALM. Helios aircraft wing model with 60 frame elements.

Figure 4.2 compares the Helios aircraft's wing displacement before and after optimisation using the SLSQP method. The red arrows indicate the lift distribution according to Schrenk's model. The lift forces act on the quarter chord line, indicated by the dashed line. The solid black line illustrates the deformed quarter chord line of the wing. Please note that the lift forces act on the undeformed wing, not the deformed one. The lift forces act on the undeformed wing because the aerodynamic model is not coupled with the structural model. The figure distinctly shows

the substantial deformation of the optimised wing compared to the baseline, underlining the significant structural changes achieved through optimisation. Please note that the deformation in figure 4.2b is the maximum wing deformation in the high-load case before failure. The wing tip displacement is 5.84% in the baseline and 26.78% in the optimised aircraft of the semi-span.

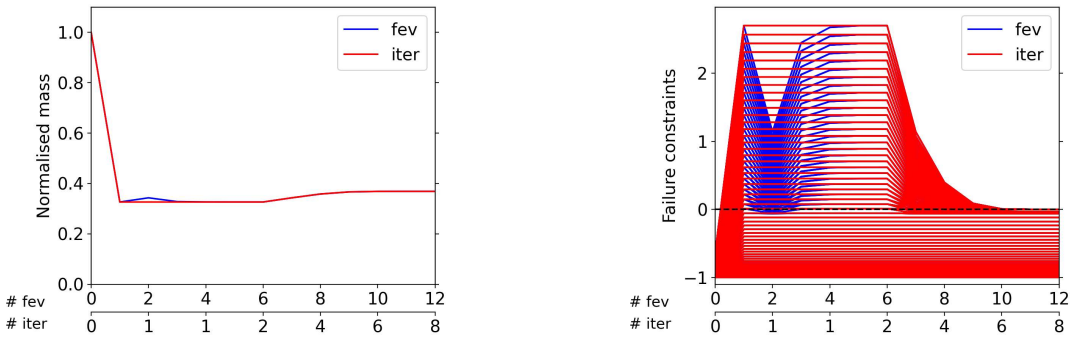


(a) Baseline (b) Optimised

FIGURE 4.2 – Helios aircraft wing displacement, before and after structural weight optimisation by SLSQP approach (without KS function) or ALM.

The optimisation process using the SLSQP method, particularly with 60 frame elements, is depicted in figure 4.3. 'fev' and 'iter' indicate the function evaluations and optimisation iterations, respectively. Figure 4.3a illustrates the normalised mass of the entire aircraft, including both wings and fixed masses. A notable reduction in the wing mass is observed right from the first iteration, despite the initial infeasibility shown in figure 4.3b. As the optimisation progresses, the structure regains feasibility, and the normalised mass stabilises at less than 40% of the baseline.

Notably, the SLSQP method achieved convergence within eight iterations and only twelve function evaluations, signifying efficiency in reaching the solution. However, it is crucial to note that the optimisation time for the SLSQP approach, particularly for the 60-frame element model, was considerably long, totalling 270.97 seconds. This extended duration is attributed to the time-consuming process of computing derivatives for all $4 \cdot 60 = 240$ constraint functions during iterations.



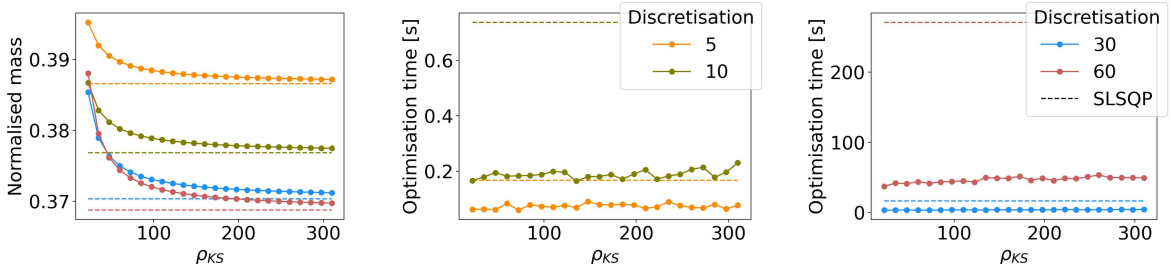
(a) Objective function (b) Constraint functions

FIGURE 4.3 – Helios aircraft SLSQP structural weight optimisation with 60 frame elements. Optimisation time: 270.97 seconds.

4.1.1.2 SLSQP KS approach analysis

The performance of the SLSQP KS approach, which utilises the KS function to aggregate inequality constraints for structural optimisation, is critically analysed. Figure 4.4 illustrates the comparative efficiency and the impact of ρ_{KS} on optimisation, respectively. A higher ρ_{KS}

enhances the solution quality by reducing the normalised mass and aligning closer to the SLSQP solution. However, the optimisation time, as shown in figures 4.4b and 4.4c, is significantly lower for the SLSQP KS approach, particularly at higher discretisation levels. While at low discretisation, the time savings are minimal, at higher complexities, such as with 60 frame elements in the Helios aircraft model, the SLSQP KS method markedly outperforms the SLSQP in terms of optimisation time, achieving results in less than 50 seconds compared to over 250 seconds. Despite these advantages, excessively high ρ_{KS} values should be avoided to prevent the KS function from becoming a less desirable piecewise differential *max* function, underscoring the importance of a balanced ρ_{KS} selection for effective constraint approximation and optimisation efficiency.



(a) Influence of ρ_{KS} on $f(\boldsymbol{x}^*)$

(b) Influence of ρ_{KS} on optimisation time

(c) Influence of ρ_{KS} on optimisation time

FIGURE 4.4 – Helios aircraft SLSQP structural weight optimisation with KS function for 5, 10, 30 & 60 frame elements.

In the comparative analysis of the SLSQP and SLSQP KS approaches, a specific margin of tolerance is defined to evaluate the closeness of the SLSQP KS results to the SLSQP method. This margin of 0.25% with respect to the baseline equates to about $1.35\,kg$ in terms of weight reduction. The margin is crucial for determining the effectiveness of the SLSQP KS approach in approximating the SLSQP results. Figure 4.5 categorises the SLSQP KS results based on this tolerance level. Optimisations falling within this defined tolerance are considered successful. The selection of the ρ_{KS} parameter in the SLSQP KS approach is pivotal. The analysis reveals that lower ρ_{KS} values often lead to outcomes outside the acceptable tolerance, especially at higher discretisations. Conversely, a ρ_{KS} value around 160 or higher consistently yields results within the tolerance range, irrespective of the model's discretisation in our considered range. This is further illustrated in Figure 4.6a, where the relationship between ρ_{KS} value, discretisation, and adherence to tolerance criteria is clearly depicted.

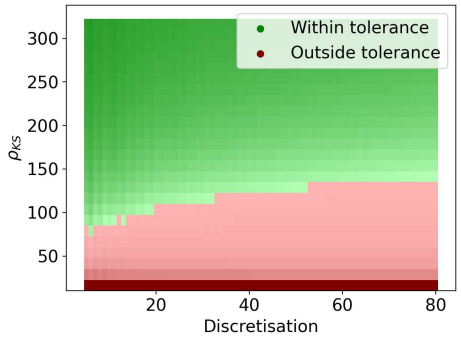
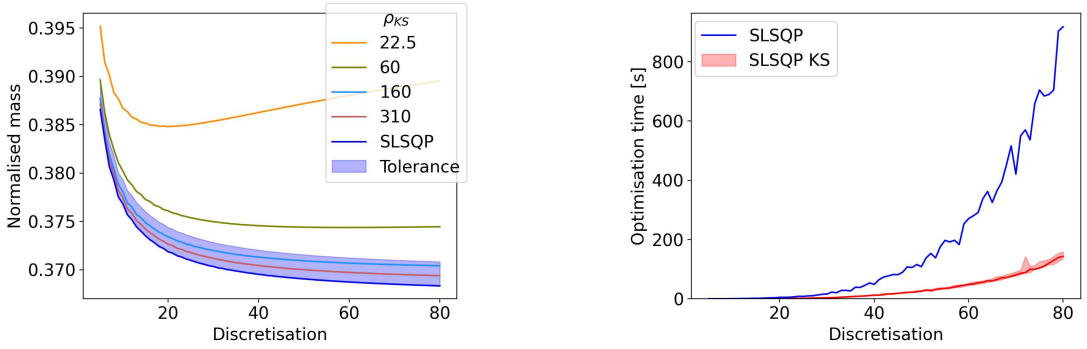


FIGURE 4.5 – Categorised Helios aircraft structural weight optimisations in within and outside tolerance sorted by ρ_{KS}. The green colour indicates that the optimisation result of the SLSQP KS approach is within tolerance. The optimisations coloured red are outside the tolerance.

Figure 4.6a critically examines the normalised mass as a function of discretisation for both

the SLSQP and SLSQP KS methods across varying ρ_{KS} values. This comparison highlights the significance of choosing an appropriate ρ_{KS} value for optimal outcomes. The figure incorporates a blue shaded area, symbolising the tolerance level of 0.25%, which serves as a benchmark for assessing the closeness of the SLSQP KS results to the SLSQP method. A notable observation from the figure is the impact of different ρ_{KS} values on solution quality. For instance, a ρ_{KS} value of 22.5 leads to suboptimal results at higher discretisations. This finding is somewhat counterintuitive, as one might expect better solutions with more frame elements. However, it underscores that without a judicious selection of ρ_{KS}, the expected improvement in solutions may not materialise, emphasising the criticality of a strategic ρ_{KS} value choice in structural optimisation. Figure 4.6a emphasises again that when the ρ_{KS} value is set to 160, the optimisation results of the SLSQP KS approach consistently fall within the defined tolerance across the considered range of discretisation. This consistency reinforces the efficacy of selecting a ρ_{KS} value of 160 for achieving optimal results within the acceptable margin, thereby providing a reliable guideline for parameter selection in (aero-) structural optimisation scenarios.



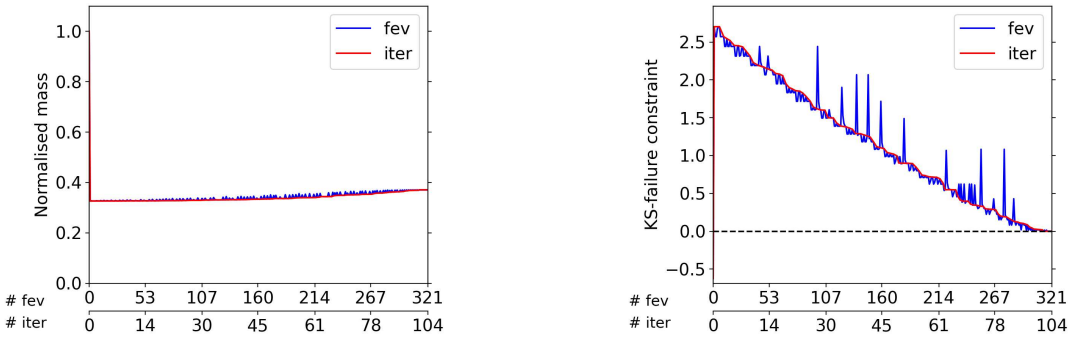
(a) $f(\boldsymbol{x}^*)$ for selected ρ_{KS} values.

(b) Optimisation time range for all ρ_{KS} values.

FIGURE 4.6 – Both figures compare the SLSQP Helios aircraft weight optimisation with and without using a KS function. The time range in figure (b) comprises optimisations within the tolerance indicated in figure (a). Shaded area in figure (b) indicates optimisation time range depending on ρ_{KS} selection.

The optimisation history of the SLSQP KS approach using $\rho_{KS} = 160$ for 60 frame elements is documented in figure 4.7. The figure comprises two parts: figure 4.7a, showcasing the objective function's progression, and figure 4.7b, depicting the KS function aggregating all inequality constraints. This approach required 104 iterations and 321 function evaluations to reach convergence, initially navigating through highly infeasible solutions before gradually shifting towards feasible ones. Interestingly, the normalised mass, serving as the objective function, experiences only a marginal increase throughout this convergence trajectory. This analysis substantiates the choice of $\rho_{KS} = 160$, highlighting its ability to consistently yield solutions within the tolerance range while ensuring optimisation time remains relatively short of only 48.79 seconds.

Figure 4.6b offers a crucial comparison of optimisation times between the SLSQP and SLSQP KS approaches, focusing on different levels of discretisation. This comparison is restricted to those SLSQP KS results falling within the predefined tolerance, ensuring fairness. The red shaded area indicates the optimisation time range of all ρ_{KS} values, which let the optimisation converge within the tolerance. The analysis reveals a significant disparity in optimisation times at higher discretisations, with the SLSQP method exhibiting a more pronounced quadratic increase in time compared to the SLSQP KS approach. The solid red line in figure 4.6b represents the optimisation with $\rho_{KS} = 160$. A critical aspect of the SLSQP KS approach's efficiency is its method of aggregating inequality constraints using the KS function. This aggregation significantly reduces the computational cost, as the SLSQP KS only requires derivatives of the objective function and the aggregated KS constraint function. In contrast, the SLSQP approach must compute derivatives for each individual constraint function. With the number



(a) Objective function

(b) Constraint functions

FIGURE 4.7 – Helios aircraft structural weight optimisation with SLSQP KS. Optimisation with $\rho_{KS} = 160$ and 60 frame elements. Optimisation time: 48.79 seconds.

of constraint functions equaling four times the number of frame elements, this difference in derivative computation requirements considerably impacts the optimisation time. Consequently, this is a critical factor in the faster optimisation times observed with the SLSQP KS approach for higher discretisations. For a discretisation of 80 frame elements, the SLSQP KS is more than five times faster than the SLSQP approach.

In summary, the SLSQP KS approach, particularly with a judicious choice of the ρ_{KS} value, demonstrates superior efficiency in optimisation time, significantly as model complexity increases. This contrast in time efficiency, alongside the ability of the SLSQP KS method to produce results within a reasonable tolerance of the SLSQP approach, underlines its potential advantages in (aero-) structural optimisation scenarios involving complex models.

4.1.1.3 ALM approach analysis

In this section, we want to draw our attention to the performance of the ALM optimisation approach. The option to select various parameters highlights the approach's versatility and complexity. These include initial Lagrange multipliers, the initial penalty factor value ρ_0, the factor γ for scaling the penalty factor, and the threshold value r for increasing the penalty. Table 4.2 summarises the parameters the user can choose and the meaning of each parameter. This flexibility in parameter selection offers significant control over the optimisation process. However, it also introduces the challenge of parameter choice, which can be daunting for users unfamiliar with optimal values. This dual nature of flexibility and complexity is a defining characteristic of the ALM approach.

Value	Description
ρ_0	Initial value for the penalty factor
γ	Factor by which the penalty factor is multiplied to increase it
r	Threshold value determining when to increase the penalty factor
$\boldsymbol{\mu}_0$	Initial values for Lagrange multipliers

TABLE 4.2 – Parameters the user needs to choose for the ALM approach.

Table 4.3 presents the chosen optimisation parameters for ALM optimisation, including initial penalty factors (ρ_0) set at 0.005, 0.2525, and 0.5 to dictate the outset of penalisation, reflecting minimal to significant initial constraint emphasis. The rate of penalty increase (γ) options are 1.1, 2.05, and 3, indicating a slow to fast penalty escalation. The threshold r value, crucial for modulating penalty factor adjustments, varies between 0 and 1. A r value of 1 delays

penalty increases until the solution's feasibility deteriorates from one iteration to the next, promoting gradual refinement. On the contrary, a r value of 0 triggers penalty increases at every iteration, fostering an aggressive push towards meeting constraints. The table's colour coding, with red highlighting conservative values, assists in parameter selection according to optimisation needs and model complexity. Highly conservative parameters prioritise satisfying constraints, while slightly conservative parameters prioritise minimising the objective function. Please note that we excluded the possibility of selecting different initial Lagrange multipliers. We will not investigate the influence of the initial guess for the Lagrange multipliers because, in our case, the structure is initially feasible, which means that we know the initial Lagrange multipliers. The Lagrange multipliers of feasible inequality constraints are zero. Furthermore, at the solution $\boldsymbol{x}^*$, most inequality constraints are inactive and inactive inequality constraints have an associated Lagrange multiplier equal to zero. Therefore, assuming that all inequality constraints are not active initially is an excellent guess.

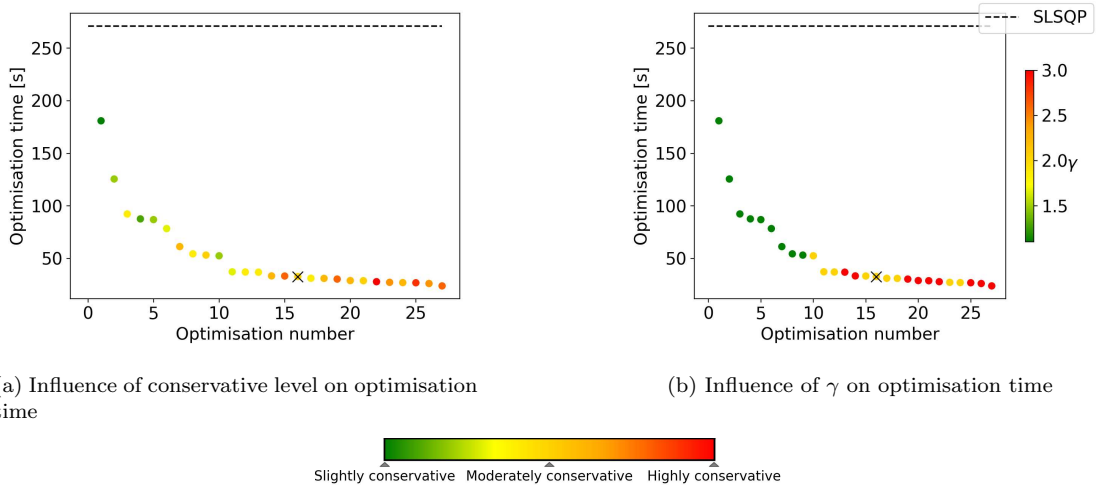
Parameter	**Values**		
ρ_0	0.005	0.2525	0.5
γ	1.1	2.05	3
r	1.0	0.5	0.0
	Slightly conservative	Moderately conservative	Highly conservative

TABLE 4.3 – Optimisation parameters values for ALM. Red are highly conservative parameters, yellow parameters are moderately conservative, and green parameters are slightly conservative parameters.

Figure 4.8 illustrates the impact of ALM parameter combinations on optimisation duration for 60 frame elements, using colour coding (green for slightly conservative, red for highly conservative, and orange/yellow for moderately) to denote conservativeness levels. The black cross indicate the moderately conservative parameters of table 4.3. The collective interaction of parameters, rather than individual ones, determines optimisation time, with highly conservative settings generally leading to quicker convergence due to an early focus on feasibility. Notably, a low γ value, especially $\gamma = 1.1$, prolongs optimisation, indicating that higher γ values may better support optimisation speed. In the 60-frame element model, despite varied times across parameters, all ALM scenarios outperform the SLSQP method, showcasing ALM's overall efficiency. Moreover, ALM's advantage grows with higher discretisation levels, consistently surpassing SLSQP and achieving shorter optimisation times, underscoring ALM's effectiveness and scalability with increased discretisation.

While choosing more conservative ALM parameters for faster optimisation might be appealing, caution is advised. Overly conservative parameters risk transforming the ALM method into a penalty optimisation method, undermining the benefits of Lagrange multipliers. This analysis underscores the importance of strategic parameter selection in the ALM approach, highlighting the need to balance between achieving quick feasibility and overall optimisation efficiency.

Building upon the analysis in figure 4.8, we delve into the ALM optimisation with two parameter sets: moderately conservative and slightly conservative. The former balances feasibility and efficiency, while the latter focuses on the Lagrange multiplier update instead of increasing the penalty factor to reach feasibility. Figures 4.9 and 4.10 highlight the ALM's performance under moderately conservative parameters. 'sub-iter' and 'alm-iter' indicate the sub-iterations and the ALM iterations of the algorithm. Figure 4.9 illustrates the interplay between objectives and constraints, showing the ALM's progression towards an optimal solution and effective constraint management. Figure 4.10 provides insight into the ALM's adaptive mechanisms, showcasing dynamic adjustments of the Lagrange multipliers and penalty factors, crucial for constraint handling and solution refinement. The ALM algorithm achieves convergence by scaling up the

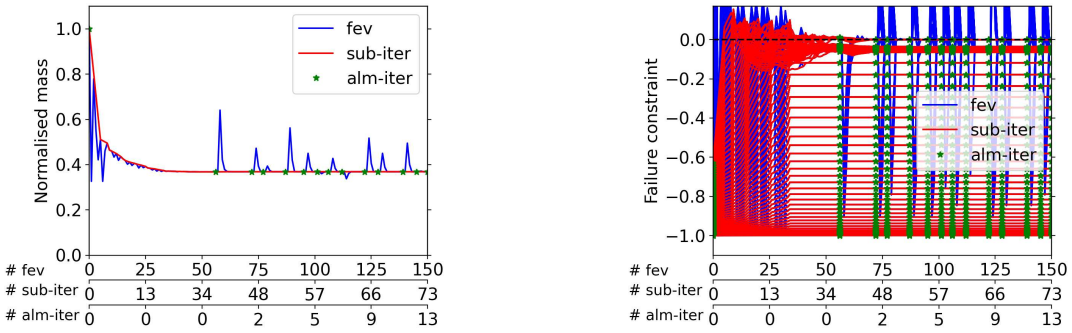


(a) Influence of conservative level on optimisation time

(b) Influence of γ on optimisation time

FIGURE 4.8 – Helios aircraft structural weight optimisation with ALM and 60 frame elements. Influence of conservative level and γ on optimisation times.

penalty factor to a few times its initial value, ρ_0, without necessitating extreme multiplicative enhancements. This behaviour is desirable to avoid ill-conditioning. Instead of increasing the penalty factor, the algorithm updates the Lagrange multipliers to achieve feasibility.

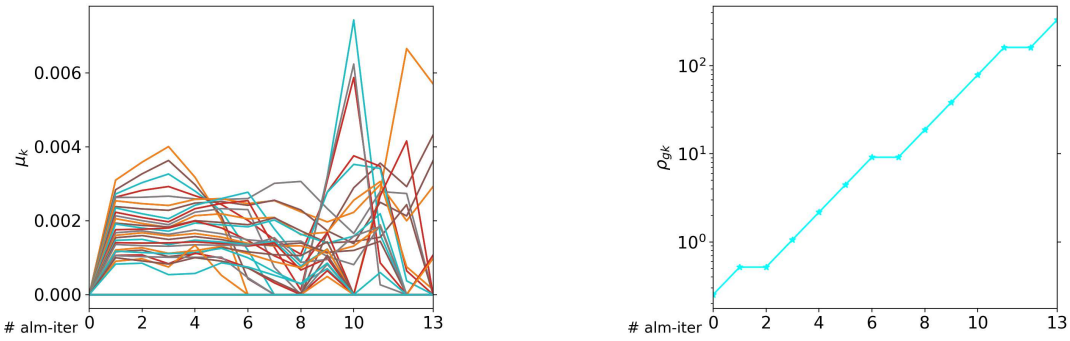


(a) Objective function

(b) Constraint functions

FIGURE 4.9 – Helios aircraft structural weight optimisation with ALM and 60 frame elements. Parameters: $\rho_0 = 0.2525$, $\gamma = 2.05$, $r = 0.5$ and $\boldsymbol{\mu}_0 = \mathbf{0}$. The figure shows desirable convergence behaviour. Optimisation time: 32.75 seconds.

Examining the ALM with slightly conservative parameters, figure 4.11 shows an example of a long optimisation time of 180.95 seconds due to slightly conservative ALM parameters. The figures show an initial decrease in normalised mass, followed by stability. Despite this early progress, the optimisation is not complete. The penalty factor's dynamics, illustrated in figure 4.11b, remain unchanged for the first 20 ALM iterations, contributing to the lengthy optimisation process due to slow constraint satisfaction. The critical insights, however, come from figure 4.11c, which uniquely presents two perspectives of the constraint satisfaction process: a zoomed-out view on the left and a zoomed-in view on the right. The zoomed-out view shows the overall trend of constraint satisfaction nearing completion. In contrast, the zoomed-in view reveals the intricate details of this process, highlighting that it takes a significantly longer time for the algorithm to drive all active constraints precisely to zero. This detailed view underscores the meticulous approach taken by the algorithm under slightly conservative parameters, striving to achieve exact constraint satisfaction, albeit at the cost of a prolonged optimisation timeline. Given the observed slow converging behaviour under the γ value of 1.1, it has been decided to exclude this particular setting from further comparative analysis. The rationale behind this

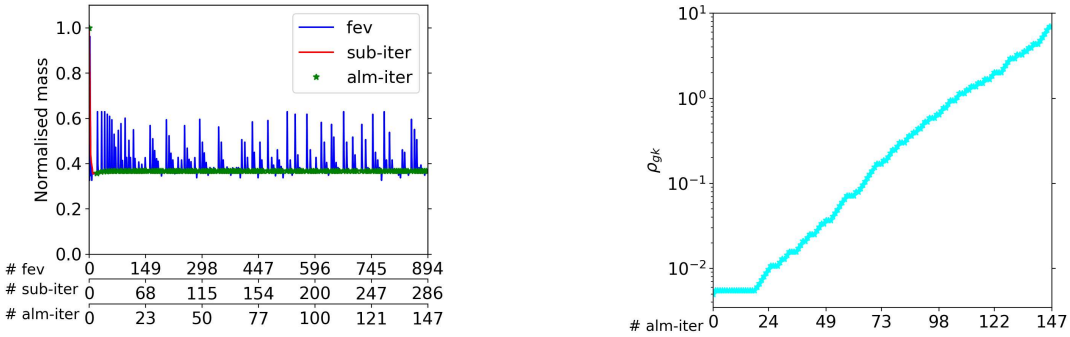


(a) Lagrange multipliers μ

(b) Penalty factor ρ_k

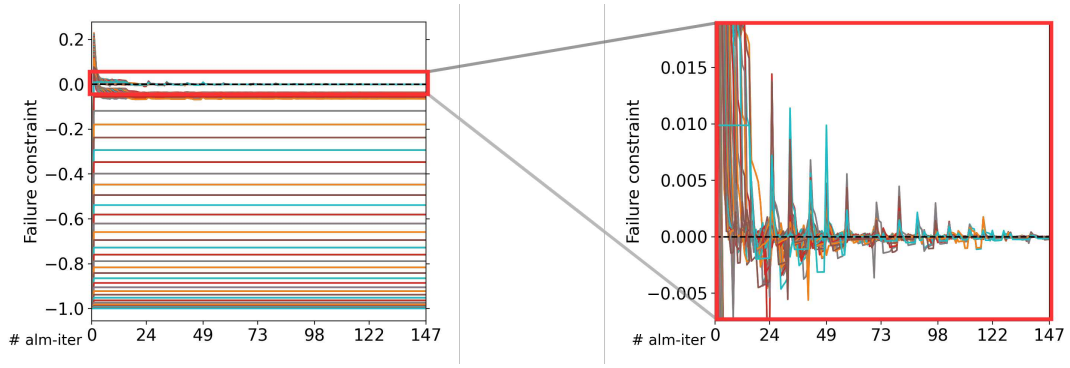
FIGURE 4.10 – Helios aircraft structural weight optimisation with ALM and 60 frame elements. Parameters: $\rho_0 = 0.2525$, $\gamma = 2.05$, $r = 0.5$, and $\boldsymbol{\mu}_0 = \mathbf{0}$. The figure shows desirable convergence behaviour. Optimisation time: 32.75 seconds.

decision is to focus on more effective parameter combinations that ensure a more balanced trade-off between convergence speed and optimisation domain exploration.



(a) Objective function

(b) Penalty factor ρ_k

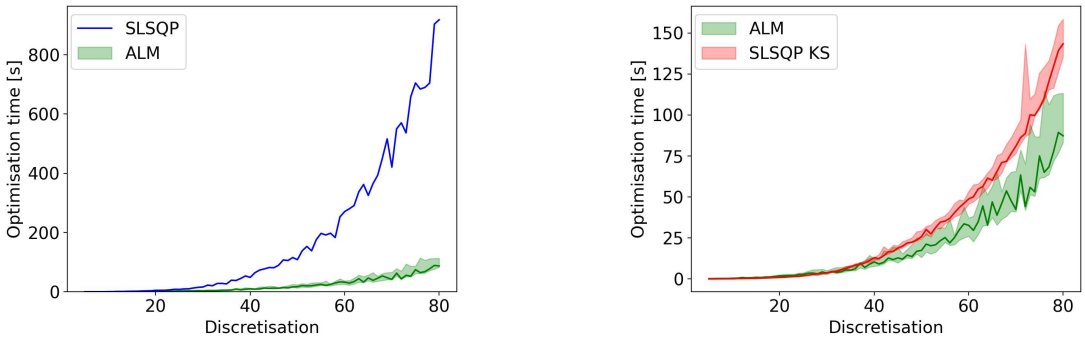


(c) Constraint functions

FIGURE 4.11 – Helios aircraft structural weight optimisation with ALM and 60 frame elements. Parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $r = 1.0$, and $\boldsymbol{\mu}_0 = \mathbf{0}$. The figure shows slow convergence due to low penalisation. Optimisation time: 180.95 seconds.

Continuing our investigation of the optimisation methods, figure 4.12 provides a crucial comparison of optimisation times between the three optimisation approaches. In this comparison, it is essential to note that optimisations, where γ is set at 1.1, have been excluded due to their previously discussed inefficient convergence behaviour. The green and red shaded areas indicate the optimisation time range depending on the optimisation parameter selection of the ALM and the SLSQP KS approaches. Figure 4.12a reveals a significant finding: the optimisation time for

the ALM is consistently lower than that of the SLSQP approach across various discretisations. This is particularly evident when employing moderately conservative parameters for the ALM, such as $\rho_0 = 0.2525$, $\gamma = 2.05$, and $r = 0.5$. The green solid line indicates the optimisation of those parameters. For a highly discretised model, the ALM is up to ten times faster than the SLSQP approach. In figure 4.12b, the focus shifts to comparing the ALM and SLSQP KS approaches. Again, the red shaded area indicates the optimisation time range of all ρ_{KS} values, which let the optimisation converge within the tolerance. The results illustrate that, particularly at lower discretisations, the optimisation times for both methods are strikingly similar. However, as the number of discretisations increases, the ALM demonstrates a more pronounced advantage in terms of optimisation time over the SLSQP KS approach. This trend underscores the effectiveness of the ALM approach, especially for more complex models with higher discretisation. For a discretisation of 80 frame elements, the ALM approach's optimisation time with moderate conservative parameters is 40% lower than that of the SLSQP KS approach with $\rho_{KS} = 160$. Depending on the optimisation parameters of both approaches, the ALM approach can be up to almost twice as fast as the SLSQP KS approach. The reason behind this efficiency becomes apparent when considering the nature of the SLSQP KS approach. As the number of frame elements—and consequently, the number of constraint functions—increases, the task of aggregating these constraints into a single KS function becomes increasingly challenging, leading to longer optimisation times. The ALM, however, handles the constraints differently by integrating them into its formulation, which, as results suggest, is a more time-efficient approach. This method's capability to handle constraints within its formulation without aggregating them leads to faster optimisation, especially as the model complexity escalates.



(a) SLSQP and ALM comparison

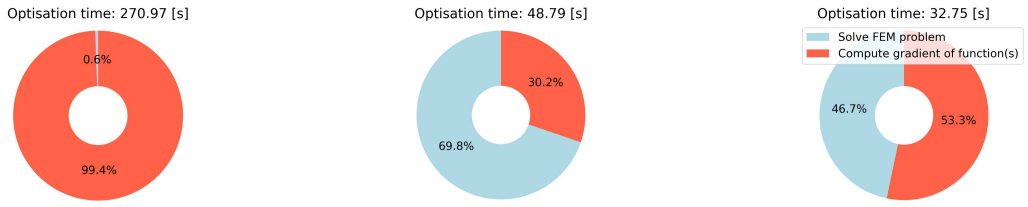
(b) SLSQP KS and ALM comparison

FIGURE 4.12 – Helios aircraft structural weight optimisation time comparison of all three approaches. Solid red line: SLSQP KS with $\rho_{KS} = 160$. Solid green line: ALM with $\rho_0 = 0.2525$, $\gamma = 2.05$, $r = 0.5$, and $\boldsymbol{\mu}_0 = \mathbf{0}$. Shaded areas indicate the optimisation time range depending on the selection of the optimisation parameters.

In conclusion, figure 4.12 provides a compelling argument for the superiority of the ALM over both the SLSQP and SLSQP KS approaches in terms of optimisation time, particularly for models with high discretisation. This superiority, coupled with the ALM's ability to converge consistently to better solutions than the SLSQP KS approach, makes it a preferred choice in scenarios where optimisation time is a critical factor.

Figure 4.13 shows a comparative analysis of time allocation across the three optimisation approaches. SLSQP primarily spends its optimisation time on gradient computations; only a minor portion of the optimisation time is necessary to solve for the state variables. This observation is confirmed by the rapid convergence with a few iterations shown in figure 4.3. In contrast, SLSQP KS predominantly invests time in problem-solving, benefiting from simplified derivative calculations due to its focus on a singular constraint and objective function, thereby reducing the derivative computation load. ALM equally divides its time between problem-solving and gradient computation. The ALM approach avoids the direct derivative calculations for con-

straints and the objective function. However, the ALM's sub-optimisation process, driven by the need to satisfy the curvature condition (eq. 3.62), demands frequent gradient evaluations, thus extending optimisation duration. Notably, optimisation time could be significantly reduced for SLSQP by accelerating derivative computations and for SLSQP KS by expediting residual function evaluations. For ALM, improvements in either aspect yield benefits due to its balanced time allocation. The distribution of optimisation time remains consistent for all three optimisation approaches across different discretisation levels.



(a) SLSQP (b) SLSQP KS (c) ALM

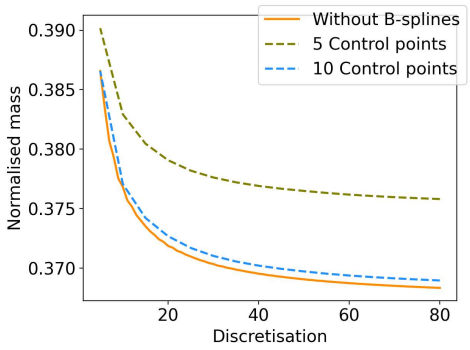
FIGURE 4.13 – Helios aircraft structural weight optimisation: Optimisation time of optimisation approaches for 60 frame configuration. SLSQP KS with $\rho_{KS} = 160$. The ALM optimisation parameters are $\rho_0 = 0.2525$, $\gamma = 2.05$, $r = 0.5$, and $\boldsymbol{\mu}_0 = \mathbf{0}$.

4.1.2 Structural optimisation with B-splines

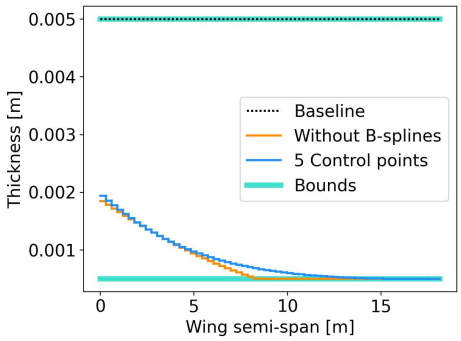
In the structural optimisation of the Helios aircraft, employing B-splines facilitates a reduction in design variables by optimising their control points rather than the direct thicknesses of each frame element. The B-splines describe the multiplicative change to the initial design state. Section A.1.8 in the Appendix explains how QASTRO employs B-splines for optimisation. While this approach generally leads to suboptimal outcomes, it significantly simplifies the representation of these thicknesses. Specifically, the thicknesses are modelled through the mathematical framework of B-splines, with their control points serving as the adjustable parameters. This method not only delineates the frame elements' dimensions but also streamlines the design process. The study confines itself to third-order B-splines, adhering to previous research findings suggesting that higher-order B-splines do not enhance results (ALMEIDA, 2021).

Illustrated in figure 4.14a is a depiction of the trade-off associated with the use of B-splines. The objective function, represented by the normalised mass, is examined in relation to discretisation. The solid line signifies the normalised mass without incorporating B-splines, while the dashed lines represent the objective function when utilising 5 and 10 control points, respectively. It is evident that the reduction in design variables, achieved through the use of B-splines, corresponds to a compromised solution, indicated by a higher normalised mass. By using 60 frame elements and 5 control points, the mass is 2% higher with respect to the baseline than for the optimisation without B-splines. In the case of 10 control points, the relative difference reduces to only 0.19% with respect to the baseline. Notably, with more control points, the solution approaches proximity to the one obtained without B-splines. When enough control points are used, the optimisations with and without B-splines give identical results.

In figure 4.14b, the tubular wing spar's thickness is depicted as a function of the wing semi-span, utilising the SLSQP method with five control points. This representation allows for visual comparison with the solution obtained without B-splines, as showcased in figure 4.1. An observation from figure 4.14b is the smoother reduction in thickness for the tubular wing spar when B-splines are employed, as opposed to the solution where B-splines were not utilised (figure 4.1). This visualisation provides insight into the functioning of B-splines and their ability to approximate a function more smoothly. Furthermore, it underscores the importance of caution



(a) Comparison of $f(\boldsymbol{x}^*)$ with and without B-splines with SLSQP approach (ALM gives identical results).



(b) Helios aircraft tubular wing spar thickness, before and after structural weight optimisation by SLSQP or ALM approach using B-spline with 5 control points.

FIGURE 4.14 – Helios aircraft structural weight optimisation modelled by 60 frame elements and optimised with third-order B-spline functions. The figure shows the influence of B-spline on optimisation results.

in selecting an appropriate number of control points. The figure demonstrates that opting for too few control points may yield suboptimal results, reinforcing the notion that an optimal balance must be struck to achieve effective results in structural optimisation using B-splines. This observation emphasises the significance of judiciously determining the number of control points to ensure the efficacy of the optimisation process. In the ensuing section, a more comprehensive discussion will ensue regarding the optimal selection of the number of control points for the optimisation process. This analysis aims to give the reader a relation between the number of control points and the optimisation time.

Subsequently, we turn our attention to analysing the impact of using B-splines on the optimisation time for both ALM and the SLSQP KS optimisation approaches. Given that both methods exhibit significantly shorter optimisation times than the SLSQP approach, we focus our comparative analysis solely on the ALM and SLSQP KS approaches, as they present relatively comparable optimisation times. Notably, even with incorporating B-splines, the SLSQP approach, without aggregating the constraint functions, continues to exhibit the most extended optimisation times.

Figure 4.15 portrays the optimisation time relative to discretisation, considering 5, 40, and 80 control points. This figure aligns with the representation in Figure 4.12b, where no B-splines were employed. Observing figure 4.15, a striking reduction in optimisation time for the SLSQP KS approach is evident when using just five control points. In contrast, the augmented Lagrangian approach's optimisation time remains relatively constant compared to the scenario without B-splines. However, we can observe for the ALM approach that the number of control points affects the influence of the ALM parameters on the optimisation time, with a smaller influence on the optimisation time for an increasing number of control points. Nevertheless, as the number of control points increases, the optimisation time for the SLSQP KS approach rises, as depicted in figures 4.15b and 4.15c, where 40 and 80 control points are employed, respectively. This trend suggests that the number of control points notably influences the optimisation time, especially for the SLSQP KS approach, highlighting the need for careful consideration in selecting an optimal number of control points for effective (aero-) structural optimisation.

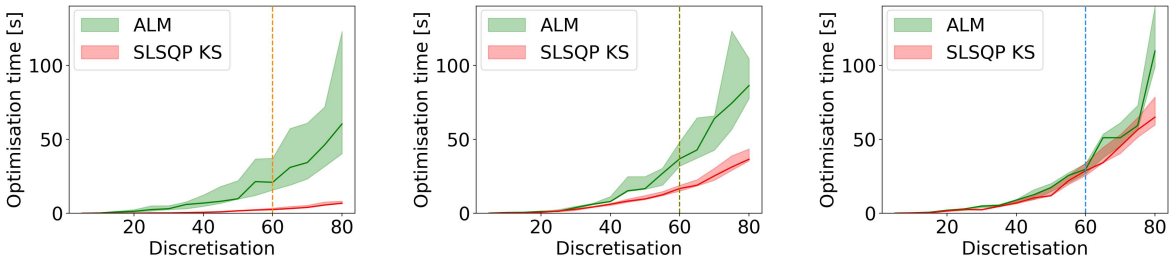
The perplexing question arises: Why does the optimisation time of the augmented Lagrangian approach not exhibit a substantial decrease with the utilisation of B-splines, while the SLSQP KS approach is significantly influenced in terms of optimisation time when employing B-splines? The key lies in understanding that the reduction of design variables achieved through using B-splines inherently diminishes the complexity of the optimisation problem. Consequently,

the SLSQP KS approach benefits from a quicker convergence to a solution, as observed in the decreased optimisation times. Furthermore, the B-splines normalise the design variables, which helps the optimiser converge faster. Contrastingly, the augmented Lagrangian approach does not experience a commensurate reduction in optimisation time when B-splines are employed. The reason for this lies in the fact that the augmented Lagrangian approach necessitates the update of hyperparameters at every augmented Lagrangian iteration. These hyperparameters correspond to the Lagrange multipliers, which are subject to updating. Notably, the crucial point is that the number of hyperparameters remains unaltered, unaffected by using B-splines. Specifically, the number of hyperparameters equals the number of inequality constraints. In practical terms, the augmented Lagrangian approach maintains four times as many hyperparameters as frame elements, resulting in a substantial quantity. Consequently, the optimisation time fails to decrease significantly because the number of hyperparameters requiring updating remains constant, and it is this factor that contributes to the sustained optimisation time observed in the ALM, irrespective of the utilisation of B-splines.

Table 4.4 illustrates the number of design variables and hyperparameters for a 60-frame elements model. The SLSQP KS approach needs to update only the design variables, whereas the ALM approach updates the design variables and the hyperparameters. For a small number of control points, thus design variables, the hyperparameter quantity is considerably more prominent compared to the design variables quantity. Furthermore, table 4.4 highlights that the ALM approach updates a large unaltered amount of hyperparameters, although the usage of B-splines.

Control points	**Design variables**	**Hyperparameters**	**Design variables + Hyperparameters**
5	5	$60 \cdot 4 = 240$	245
40	40	$60 \cdot 4 = 240$	280
80	80	$60 \cdot 4 = 240$	320
	SLSQP KS	↑Stays unaltered	ALM

TABLE 4.4 – Overview of the number of design variables and hyperparameters for 60-frame elements model. We can effectively reduce the design variables using B-splines, but the number of hyperparameters stays unaltered. The ALM approach needs to update the hyperparameters, which increases the optimisation time.



(a) 5 control points (b) 40 control points (c) 80 control points

FIGURE 4.15 – Helios aircraft structural weight optimisation time comparison of SLSQP KS and ALM approaches with B-splines. Solid red line: SLSQP KS with $\rho_{KS} = 160$. Solid green line: ALM with $\rho_0 = 0.2525$, $\gamma = 2.05$, $r = 0.5$, and $\boldsymbol{\mu}_0 = \mathbf{0}$. Shaded areas indicate the optimisation time range depending on optimisation parameter selection.

Continuing the discussion, let us focus on figure 4.16 to provide a visual representation of the optimisation time for the ALM and the SLSQP KS optimisation approach with varying numbers of control points while keeping the number of frame elements fixed at 60. In figure 4.16, the vertical lines represent the control points 5, 40 and 80, as indicated in figure 4.15,

providing a reference for the reader to comprehend the significance of the graph. Notably, figure 4.16 reinforces the earlier explanation that increasing the number of control points leads to an increase in the optimisation time for the SLSQP KS approach. In contrast, the optimisation time for the augmented Lagrangian approach remains relatively constant. It is essential to acknowledge that the choice of using more control points than discretisation, as shown in figure 4.16, might not always be optimal. However, the subsequent sections will delve into determining an optimal number of control points for effective optimisations.

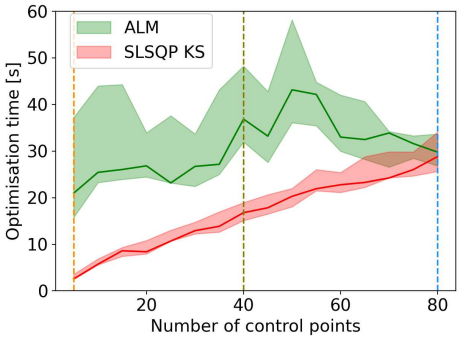


FIGURE 4.16 – Helios aircraft structural optimisation time as a function of number of control points for Helios aircraft with 60 frame elements. Shaded areas indicate the optimisation time range depending on optimisation parameter selection.

The overarching conclusion is that employing B-splines is advisable. Primarily, this recommendation stems from the normalisation of design variables facilitated by B-splines, along with the significant reduction in the optimisation time of the SLSQP KS approach—an advantageous outcome. Although there is a marginal reduction in the objective function value when B-splines are used, it becomes apparent that, with an adequate number of control points, the solutions with and without B-splines closely align. Furthermore, B-splines aid the optimiser in navigating the design space by restricting unrealistic changes in thickness between neighbouring frame elements. This feature prevents the exploration of design spaces that lack practical feasibility. While this effect is not explicitly illustrated in the current analysis, it is noteworthy as a supplementary point highlighting the multifaceted advantages of B-splines. A secondary observation is drawn from this section: the SLSQP KS approach enjoys an advantage in terms of optimisation time over the ALM approach, especially when the number of control points is small. However, as the number of control points increases, the optimisation time of the ALM approach becomes smaller compared to the SLSQP KS approach. This nuanced perspective underscores the context-dependent efficiency of the optimisation methods and the need for careful consideration of factors such as control points in achieving optimal outcomes.

4.2 Aerostructural optimisation problems

The aerostructural optimisation results focus on the result $f(\boldsymbol{x}^*)$ of the three different optimisation approaches and not anymore on the optimisation time as in the structural optimisation result section 4.1. This is because, in the structural optimisation, all optimisation approaches converged within a small tolerance to the same solution (for SLSQP KS only if ρ_{KS} was high enough). However, for the aerostructural analysis, this is not the case anymore. In this section, we will analyse to which solution the optimisation approaches converge and make a relationship to the optimisation parameters. This chapter is important because it shows how we can get the best optimisation result with each optimisation approach. Furthermore, this chapter investigates the influence of the discretisation. For the aerostructural model, we discretise the structural

model in finite beam-truss (i.e. frame) elements and the aerodynamic model in horseshoe vortices along the quarter-chord (i.e. panels). It is essential to mention that for the aerostructural optimisation, it was necessary to model the Helios aircraft with around 20 horseshoe vortices or higher. QASTRO could no longer compute the state variables for a lower lifting-line discretisation. For this reason, the Helios aircraft is modelled with at least 20 horseshoe vortices in this section. In this section, all design variables are described by third-order B-splines. Also, note that QASTRO does not allow users to apply symmetry conditions for aerostructural optimisations. Therefore, the entire aircraft is modelled for the aerostructural optimisations. For all aerostructural optimisations we take $\boldsymbol{\lambda}_0 = \mathbf{0}$, similar to the initial inequality Lagrange multipliers $\boldsymbol{\mu}_0$ in the structural optimisations.

The aerostructural optimisation result section is divided into two parts. The first subsection describes the results with a single flight condition, and the following subsection describes the solution with two flight conditions.

4.2.1 Single flight condition

In this subsection, we will focus on the optimisation results with a single flight condition. The optimisation problem with a single flight condition is described in equation 3.82. This subsection is further subdivided into a subsection where we use the reduced set of design variables and the complete set of design variables.

4.2.1.1 Reduced set of design variables

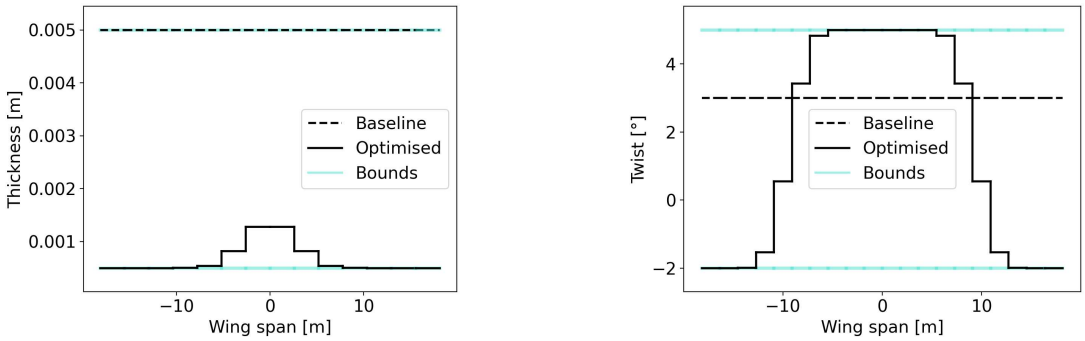
In this section, we examine the results of optimising the Helios aircraft under a single-flight condition, focusing on a reduced set of design variables. The reduced set of design variables are:

$$\boldsymbol{x}_{\text{reduced}} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{\alpha}_0 & \alpha \end{bmatrix}^T \tag{4.4}$$

This analysis is structured into several parts, each corresponding to the different optimisation approaches employed: SLSQP, SLSQP KS and ALM approach.

SLSQP approach analysis

Figures 4.17 and 4.18 illustrate the outcome of the SLSQP approach with a FEM and LLT discretisation of 14 and 20, respectively. The optimised mass is 38.10% of the baseline. Figure 4.17 demonstrates the distribution of the tubular wing spar $\boldsymbol{t}$, along with the wing twist $\boldsymbol{\alpha}_0$. It is evident from these results that the distribution of thickness mirrors the outcomes observed in the structural optimisation problem, albeit with a slight overall reduction in thickness. This reduction is attributed to the inclusion of both twist $\boldsymbol{\alpha}_0$ and angle of attack α as design variables. Because the optimiser can control each panel's local angle of attack, the optimiser can control the lift distribution, which leads to a lighter wing structure. In the structural optimisation, we could only use an approximation of the lift distribution. Figure 4.18 presents the aircraft model incorporating the thickness and twist parameters as depicted in figure 4.17. The red arrows indicate the lift distribution of the aircraft. One can see that the lift distribution deviates from the elliptical distribution, indicating that the induced drag is high in the optimised state. By generating more lift at the wing root, the optimiser can further alleviate the bending moment applied at the structure. We could achieve a similar effect by reducing the wing span. However, please note that this stands in contrast to maximising endurance and the power production over power consumption ratio. Later in this chapter, we will discuss the results of the optimisations further.



(a) Thickness distribution

(b) Twist distribution

FIGURE 4.17 – Objective function: Normalised weight; reduced set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α; single flight condition. Aerostructural optimisation result of SLSQP with 14 control points with FEM discretisation 14 and LLT discretisation 20. Result of other design variables: $\alpha = 1.42\,^{\circ}$, $V_\infty = 28.62\,m/s$.

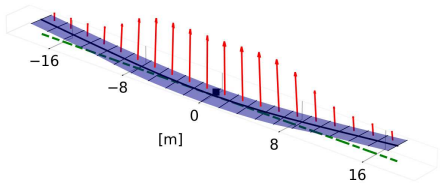
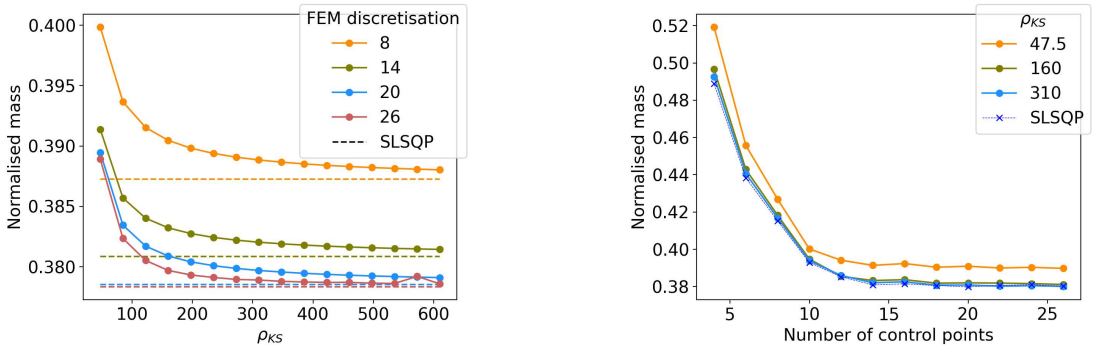


FIGURE 4.18 – Objective function: Normalised weight; reduced set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α; single flight condition. Aerostructural optimisation result of SLSQP with 14 control points with FEM discretisation 14 and LLT discretisation 20.

Comparison of SLSQP and SLSQP KS approaches

Figure 4.19 compares the SLSQP, and the SLSQP KS approaches. This comparison focuses on the effect of varying the ρ_{KS} value in the SLSQP KS approach. It is observed that with increasing ρ_{KS} values, the SLSQP KS solution tends to converge towards the SLSQP solution. This result reflects the same conclusion as the structural result section. However, an excessively high ρ_{KS} value may impede this convergence, as evidenced in figure 4.19a, where the objective function value deviates from the SLSQP solution for a FEM discretisation of 26. Later in this section, we will focus more on this discrepancy.



(a) Influence of ρ_{KS} on $f(\boldsymbol{x})$ using 14 control points and LLT discretisation of 20 horseshoe vortices.

(b) Influence of number of control points. Fixed FEM and LLT discretisation of 14 and 20, respectively.

FIGURE 4.19 – Objective function: Normalised weight; reduced set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α; aerostructural optimisation with single flight condition.

Figure 4.19b emphasises the relationship between the number of control points used for the B-splines and the final objective function value (normalised weight). Note that B-splines are used to describe both twist and thickness. This relationship is analysed under a fixed number of FEM and LLT discretisations, set at 14 and 20, respectively. The analysis reveals that an increase in the number of control points correlates with a lower objective function value up to the point where the number of control points equals the discretisation number. Beyond this, additional control points do not significantly reduce the objective function value. The reason that we need so many control points to obtain a solution with a low objective function value can be found in figure 4.17b. The twist changes rapidly at around 10 metres of the wing span. It requires many control points that the B-spline can capture this behaviour. Because of this observation, we will use as many control points as horseshoe vortices for the LLT discretisation in many analyses. Additionally, figure 4.19b shows that higher ρ_{KS} values lead to improved solutions, converging towards the SLSQP solution. The same conclusion which we got already from figure 4.19a, just from a different perspective.

ALM approach analysis

Table 4.5 lists the analysed optimisation parameters for the ALM optimisation approach. The values coloured in red are highly conservative parameters, the values coloured in yellow are moderately conservative parameters, and the values coloured in green are slightly conservative parameters. Highly conservative parameters penalise infeasibility significantly, and slightly conservative parameters allow the algorithm to explore infeasible solutions.

Parameter	**Values**		
ρ_0	0.005	0.2525	0.5
γ	1.1	2.05	3
r	1.0	0.5	0.0
	Slightly conservative	Moderately conservative	Highly conservative

TABLE 4.5 – Optimisation parameters values for ALM. Red are highly conservative parameters, yellow parameters are moderately conservative, and green parameters are slightly conservative parameters.

Figure 4.20 explores the ALM approach, focusing on the impact of varying levels of conservatism in the optimisation parameters. Highly conservative parameters, which seek feasible solutions rapidly, are contrasted with slightly conservative parameters, which allow gradual updates of the hyperparameters. The findings suggest that overly conservative parameters may prevent convergence to the optimal solution achieved by the SLSQP method. In contrast, slightly conservative parameters appear more likely to reach this optimal solution without becoming entrapped in local minima. Figure 4.20 shows that the ALM can converge to worse solutions with highly conservative optimisation parameters where the normalised weight is the objective function. However, this behaviour could also be shown with the endurance or the power ratio as the objective function. The author chose to present the bad converging behaviour with the normalised weight as an objective function because, in this case, the ALM optimisation approach behaved the worst. In the following sections, we will see that the other optimisation approaches also have the most difficulties in optimising the aircraft's weight.

In summary, for section 4.2.1.1, it is discerned that all three optimisation approaches are viable with a reduced set of design variables. However, it is crucial to employ slightly conservative optimisation parameters for the ALM approach to avoid suboptimal solutions. Please note that slightly conservative parameters increase the optimisation time as concluded from section 4.1

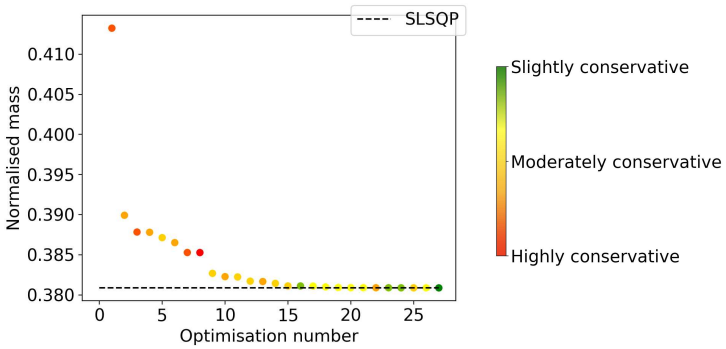


FIGURE 4.20 – Objective function: Normalised weight; reduced set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α; Aerostructural optimisation with single flight condition. Influence of conservative level of ALM optimisation parameters on $f(\boldsymbol{x}^*)$. FEM discretisation 14, LLT discretisation 20 and 14 control points.

4.2.1.2 Complete set of design variables

In this subsection, we delve deeper into the optimisation of the Helios aircraft under single-flight conditions, employing a complete set of design variables, including the thickness of the frame element $\boldsymbol{t}$, wing twist $\boldsymbol{\alpha}_0$, angle of attack α, chord lengths $\boldsymbol{c}$, free-stream velocity V_∞ and the frame element location $\boldsymbol{x}_{\text{Beam}}$ along the chord length.

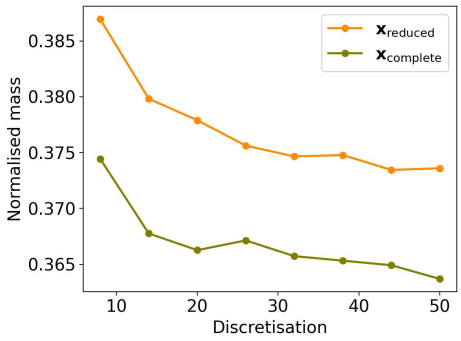
$$\boldsymbol{x}_{\text{complete}} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{\alpha}_0 & \alpha & \boldsymbol{c} & V_\infty & \boldsymbol{x}_{\text{Beam}} \end{bmatrix}^T \tag{4.5}$$

The frame element location in the x - direction might not seem to influence the optimisation result, but this is not the case. The strategic adjustment of the FEM nodes in the x-direction, parallel to the chord length, is particularly crucial. This adjustment allows for nuanced control over the wing's twist in response to lift forces, crucially impacting its aerodynamic efficiency and structural integrity due to the created moments.

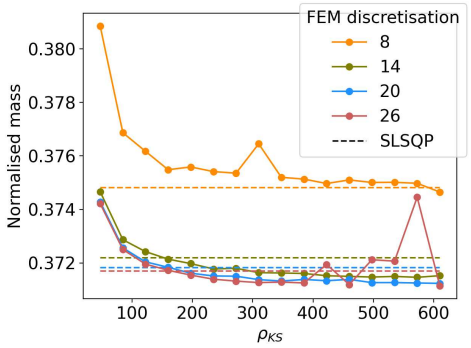
Comparison of SLSQP and SLSQP KS approaches

Before presenting the results with the complete set of design variables, the author wants to emphasise that including more groups of design variables can reduce the objective function. Figure 4.21a offers a detailed illustration of the benefits of using the complete set of design variables. The figure shows the normalised mass as a function of the discretisation using the SLSQP approach. For every discretisation, the final normalised mass is about 1% lower using the complete set of design variables compared to using the reduced set of design variables. A reduction of 1% corresponds to a notable decrease of about $5.42\,kg$. Figure 4.21a shows one more important observation. The objective function value is higher for a discretisation of 26 than for a discretisation of 20 using the complete set of design variables. This shows that the SLSQP method has difficulties to converge to the global minimum, which can be confirmed by figure 4.21b.

The result of the SLSQP KS approach, shown in figure 4.21b, indicates that the SLSQP KS approach can converge to solutions which are better than the SLSQP approach. This observation contradicts the previous results, where the SLSQP KS always converged to the SLSQP approach. It seems that it helps to simplify the optimisation problem by aggregating the inequality constraint functions to avoid the SLSQP method converging to local minima. Thus, selecting the SLSQP KS optimisation approach over the SLSQP approach for complex optimisation problems is advisable to converge to a better solution. Note that we can not adjust any parameters for the SLSQP approach to overcome this problem, except changing the optimisation's starting point $\boldsymbol{x}_0$. The lack of flexibility showcases a big disadvantage of the SLSQP approach. However,



(a) Difference between reduced and complete set of design variables on $f(\boldsymbol{x}^*)$



(b) Aerostructural optimisation results. Influence of ρ_{KS} on $f(\boldsymbol{x}^*)$ with complete set of design variables. Fixed LLT discretisation of 20 horseshoe vortices.

FIGURE 4.21 – Objective function: Normalised weight; aerostructural optimisation with single flight condition. Left figure: For discretisation < 20, LLT discretisation is 20 horseshoe vortices. For discretisation ≥ 20, LLT and FEM discretisations are equal. The number of control points is equal to the LLT discretisation.

choosing the ρ_{KS} value is critical to avoid convergence problems. If ρ_{KS} is chosen too high, the optimisation approach has difficulty converging to the best possible solution. High ρ_{KS} increases the difficulties in computing the gradient of the KS function, leading to suboptimal convergence behaviour. E.g. this can be seen in figure 4.21b for a discretisation of 26 frame elements. We can see that for $\rho_{KS} > 310$, the approximation of the inequality constraints by the KS function increases only slightly, but we increase the risk of converging to sub-optimal solutions due to a decrease in differentiability of the KS function. Consequently, $\rho_{KS} = 310$ shows a good trade-off between solution quality and differentiability.

ALM approach analysis

The previous subsection concluded that choosing slightly conservative optimisation parameters for the ALM approach is advisable. However, in this subsection, the author wants to emphasise the potential difficulties associated with penalisation that is too low and the strategies to overcome them. A key challenge observed with insufficiently conservative parameters is the emergence of oscillating behaviour in the optimisation process. This oscillation often manifests as a back-and-forth fluctuation between the bounds of design variables, mainly when the constraint violation penalties are insufficient to guide the optimiser towards a stable solution. Such oscillatory tendencies are clearly illustrated in figure 4.22, where we optimised for the negative normalised endurance. The reason for this oscillating behaviour is the insufficient penalisation of the constraints. Increasing the r value, which determines if the penalty factor becomes increased if the new design state is infeasible enough, efficiently overcomes this problem. Figure 4.23 illustrates the better performance with a lower r value. Regarding this observation, we will choose $r = 0.75$ to optimise the endurance. Also, we encounter a slow convergence with $r = 1.0$ for the power ratio as the objective function. For this reason, we will choose $r = 0.75$ when optimising concerning the power ratio, too. Table 4.6 summarises the chosen optimisation parameters for the ALM optimisation approach.

Objective function	**Weight**	**Endurance**	**Power ratio**
ρ_0	0.005	0.005	0.005
γ	1.1	1.1	1.1
r	1.0	0.75	0.75

TABLE 4.6 – Chosen ALM optimisation parameters for aerostructural optimisations.

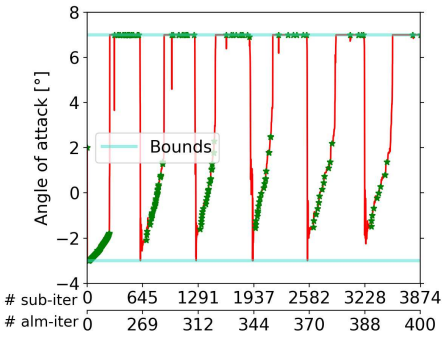
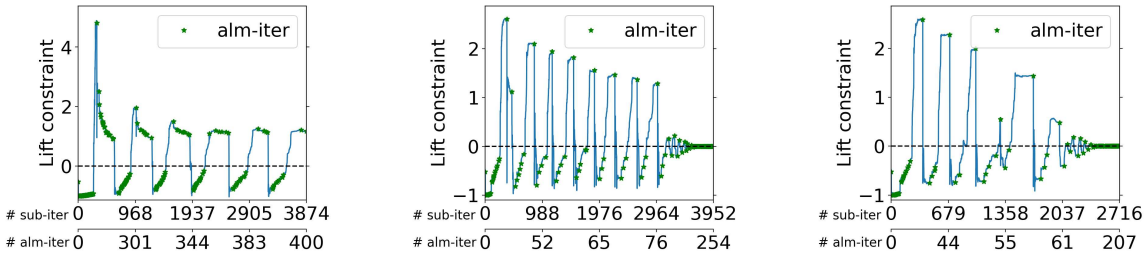


FIGURE 4.22 – Objective function: Negative normalised endurance; aerostructural optimisation with single flight condition and complete set of design variables. Illustration of oscillating design variable due to low penalisation with 20 control points and a FEM and LLT discretisation of 20. Optimisation parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $r = 1$, $\lambda_0 = 0$, $\boldsymbol{\mu}_0 = \mathbf{0}$ and separate penalty factors for equality and inequality constraints.



(a) $r = 1.0$ (b) $r = 0.75$ (c) $r = 0.5$

FIGURE 4.23 – Objective function: Negative normalised endurance; aerostructural optimisation single flight condition and complete set of design variables. Illustration of oscillating behaviour due to low penalisation with 20 control points and a FEM and LLT discretisation of 20. Optimisation parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $\lambda_0 = 0$, $\boldsymbol{\mu}_0 = \mathbf{0}$ and separate penalty factors for equality and inequality constraints. $r = 1$ was terminated after the 400^{th} ALM iteration. $r = 0.75$ and $r = 0.5$ converged to the same solution.

Next, the ALM optimisation approach also faced difficulties converging to symmetric solutions. Figure 4.24 illustrates the outcome when employing the ALM approach with a discretisation of 26. This figure reveals an asymmetry in the chord length distribution along the wingspan resulting from the ALM using a single penalty factor. The asymmetrical solution leads to a suboptimal normalised mass, indicating that the optimiser, in this setup, converges to an unbalanced solution that does not adequately represent the optimal aerodynamic efficiency of the aircraft.

Theoretically, the optimiser should never converge to an asymmetrical solution because the initial design state is symmetric. The symmetry of the initial state ensures that the gradients with respect to each design variable group, such as $\boldsymbol{\alpha}_0$, are also symmetric. This symmetry in gradients should, in turn, lead to a symmetric solution in subsequent iterations. However, the emergence of an asymmetrical final result can be attributed to the methodology employed by QASTRO in computing the derivatives, as detailed in section 3.5. QASTRO's approach involves solving the linear system of equation 3.76 to obtain the adjoint variables, which in turn provide the derivatives with respect to the design variables. Solving for the adjoint variables introduces minor uncertainties, potentially by convergence issues and defined tolerances. Those uncertainties can lead to asymmetrical derivatives and, consequently, asymmetrical design evaluations.

The SLSQP optimisation method has fewer difficulties with asymmetric derivatives, which can be seen by the fact that the SLSQP and SLSQP KS approaches converge less frequently to asymmetrical solutions. It is important to note that it could improve the ALM performance if we

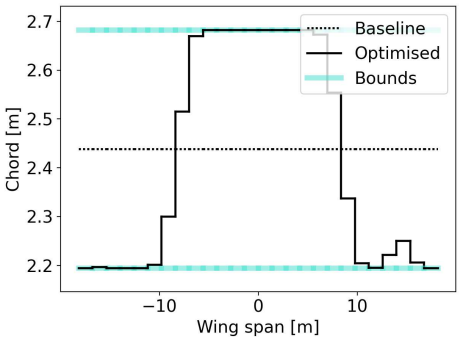
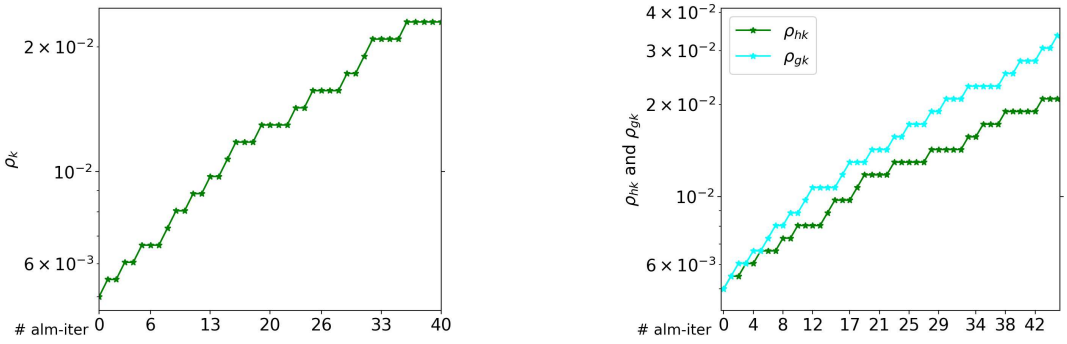


FIGURE 4.24 – Objective function: Normalised weight; aerostructural optimisation single flight condition and complete set of design variables. Illustration of asymmetrical solution of design variables with 26 control points and a FEM and LLT discretisation of 26. Optimisation parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $r = 1.0$, $\lambda_0 = 0$, $\boldsymbol{\mu}_0 = \mathbf{0}$ and single penalty factor for equality and inequality constraints.

select a different optimisation method for solving the ALM approach's sub-problems since other optimisation methods can better handle the inaccuracies of the computed gradients. Nevertheless, one possibility to help the ALM optimisation approach converge to symmetric solutions is using two separate penalty factors, one for equality and one for inequality constraints. The reason why using two separate penalty factors can increase the performance of the ALM algorithm can be explained by figure 4.25.



(a) Single penalty factor

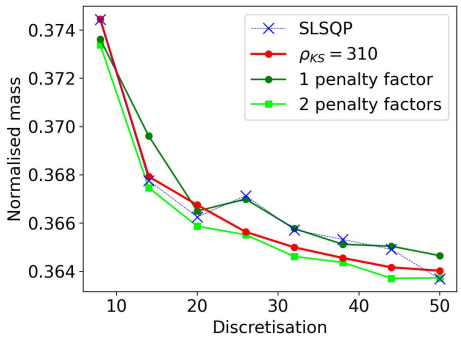
(b) Separate penalty factors

FIGURE 4.25 – Objective function: Normalised weight; aerostructural optimisation single flight condition and complete set of design variables. Illustration of the difference between a single penalty factor and separate penalty factors with 26 control points and a FEM and LLT discretisation of 26. Optimisation parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $r = 1.0$, $\lambda_0 = 0$, $\boldsymbol{\mu}_0 = \mathbf{0}$. Optimisation terminated successfully after 40 ALM iterations with a single penalty factor and after 217 ALM iterations with two penalty factors (not all ALM iterations are shown in figure (b)).

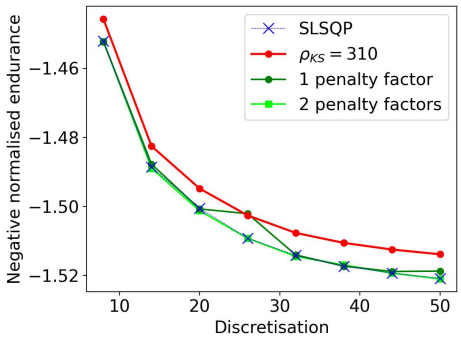
Figure 4.25 illustrates the increase of the penalty factors using single and separate ones. In this figure, we observe that when a single penalty factor is applied, the optimiser's ability to differentiate and effectively manage the equality and inequality constraints is limited, often leading to unbalanced and asymmetric outcomes. Conversely, with two separate penalty factors, the algorithm can independently adjust and fine-tune the penalisations for each type of constraint. This targeted approach allows the optimiser to navigate the complexities of the problem more effectively, reducing the likelihood of asymmetric solutions and improving the overall balance and optimality of the results. Figure 4.25b shows that most of the time the ALM algorithm either increase the penalty factor of the equality or inequality constraints, but not both at the same time. By this, the algorithm can reduce the complexity of the optimisation problem, focusing sequentially on the equality and inequality constraints. Using two penalty factors increases

the chances of converging to symmetric and potentially more aerodynamically efficient designs and enhances the optimisation process's robustness in navigating complex design landscapes. However, it is important to note that employing two separate penalty factors is not a panacea. While it offers a more refined approach to managing constraints, it also introduces another layer of complexity in the form of another hyperparameter that needs to be managed throughout the optimisation process. This complexity underscores the need for careful consideration and calibration of these factors to ensure they contribute positively to the optimisation process rather than adding unnecessary complications. In many other optimisations, using two penalty factors had a negative effect.

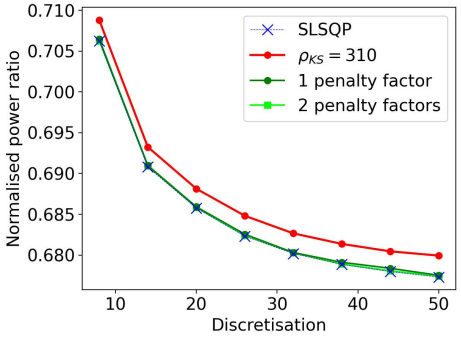
Figure 4.26 offers a comprehensive comparative analysis of the three optimisation approaches — ALM, SLSQP KS, and SLSQP — when optimising for normalised mass, negative normalised endurance, and normalised power ratio. The figure presents the objective functions as a function of discretisation, providing insight into each method's performance across different discretisation levels. Notably, the trend in the figure indicates that beyond a certain point, particularly at a discretisation level of 50, further increases in discretisation yield diminishing improvements in the objective functions. This observation suggests that a discretisation level of 50 is sufficiently detailed for the optimisation process, as it does not lead to a significant decrease in the objective functions compared to higher discretisation levels. Therefore, while higher discretisation provides a more refined model, it does not necessarily translate into substantial enhancements in the optimisation results beyond this level.



(a) Objective function: Normalised weight



(b) Objective function: Negative normalised endurance



(c) Objective function: Normalised power ratio

FIGURE 4.26 – Aerostructural optimisation with single flight condition and complete set of design variables. Objective functions as a function of discretisation. For discretisation < 20, LLT discretisation is 20 horseshoe vortices. For discretisation ≥ 20, LLT and FEM discretisations are equal. The number of control points is equal to the LLT discretisation.

In figure 4.26a, it is particularly noteworthy that the SLSQP approach has considerable difficulties in converging to the same solution as the SLSQP KS and the ALM with two penalty

factors. We could already observe the convergence difficulty of the SLSQP approach in figure 4.21b. The ALM optimisation approach using a single penalty factor yields less favourable results when optimising for normalised mass, similar to the SLSQP approach. The suboptimal ALM results are due to the symmetry issue addressed earlier. As shown in figure 4.26a, using two penalty factors increases the performance of the ALM approach. However, for the other two objective functions—negative normalised endurance and normalised power ratio—the outcomes using single and separate penalty factors are almost identical. This observation highlights that the impact of penalty factor configuration can vary significantly depending on the optimisation objective. When optimising for the normalised power ratio, all methods show a robust convergence performance, suggesting that the choice of optimisation approach may be less crucial for this particular objective, provided the discretisation is handled appropriately.

Table 4.7 lists the final objective function value $f(\boldsymbol{x}^*)$ of all three optimisation approaches with a discretisation of 50. The table compares the final objective function values with the two condition optimisations. In the following subsection, we will discuss in detail the optimisation results by comparing them with the two flight conditions described in the following subsection.

Furthermore, figure 4.27 showcases the final optimised aircraft results for the three objective functions at a discretisation level of 50 utilising the ALM approach. This figure visually demonstrates the tangible outcomes of the optimisation processes, enabling a direct comparison of how different objectives impact the final aircraft design. The visualisation in figure 4.27 is integral for understanding the practical implications of optimisation choices in aerostructural design. Figures 4.30 - 4.32 illustrate the design variables of the optimised aircraft with respect to the weight, endurance and power ratio.

Please note that when we optimise the aircraft's weight, the lift distribution deviates from the elliptical lift distribution. Employing the complete set of design variables, it becomes evident that for the section of the aircraft's wing extending beyond a 10-meter span from the fuselage to the wingtip, there is a significant reduction in lift generation. This results in a minimal aerodynamic lift within this outer portion of the wing and an increase in the induced drag. Note that we obtain a lift distribution with a lower induced drag when optimising the endurance and power ratio.

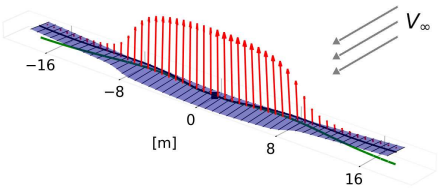
In summary, figures 4.26 and 4.27 collectively provide a deep dive into the optimisation behaviour under varying objectives and discretisation levels. They offer valuable insights for future optimisation efforts in similar aerostructural scenarios, highlighting the importance of method selection, discretisation levels, and the configuration of penalty factors.

4.2.2 Two flight conditions

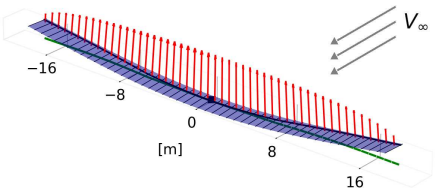
This section presents the optimisation results for the Helios aircraft under two flight conditions: cruise and high-load. The high-load condition demands the aircraft to produce lift forces three times greater than in cruise condition, which is a stricter constraint than the constraint in the single flight condition optimisation. Addressing this challenge requires utilising the complete set of design variables, which are

$$\boldsymbol{x}_{\text{complete 1, 2}} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{\alpha}_0 & \alpha_1 & \alpha_2 & \boldsymbol{c} & V_{\infty 1} & V_{\infty 2} & \boldsymbol{x}_{\text{Beam}} \end{bmatrix}^T, \tag{4.6}$$

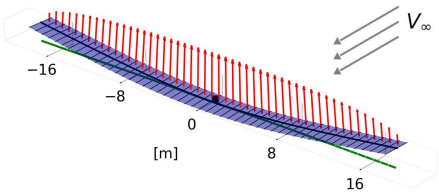
where the subscripts 1 and 2 indicate the design variable of the first and second flight conditions (cruise and high load flight conditions), respectively. Using the reduced set of design variables is insufficient to obtain a feasible solution. This analysis explores how adding a flight condition affects the aircraft's aerodynamic and structural design, highlighting the complexities of meeting the additional high-load model constraints.



(a) Objective function: Normalised weight



(b) Objective function: Negative normalised endurance



(c) Objective function: Normalised power ratio

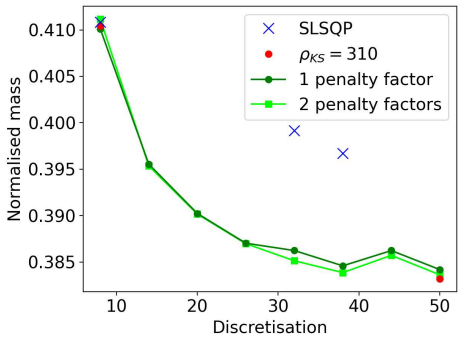
FIGURE 4.27 – Aerostructural ALM optimisation with single flight condition and complete set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α, $\boldsymbol{c}$, V_∞, $\boldsymbol{x}_{\text{Beam}}$. Optimisation result of ALM with 50 control points with FEM and LLT discretisation of 50. Optimisation parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $\lambda_0 = 0$, $\boldsymbol{\mu}_0 = \mathbf{0}$ and separate penalty factors for equality and inequality constraints. $r = 1$ for normalised weight optimisation. $r = 0.75$ for endurance and power ratio optimisation.

4.2.2.1 Comparing of SLSQP, SLSQP KS and ALM approach performances

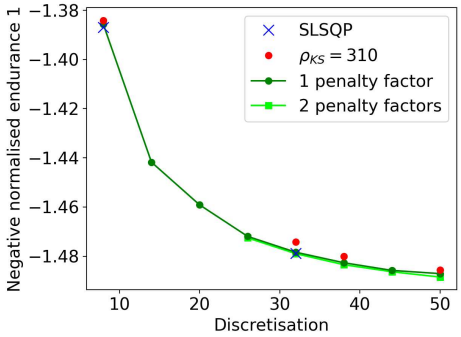
In figure 4.28, the effects of discretisation on the optimisation results for the three objective functions — normalised weight, negative normalised endurance, and normalised power ratio — are analysed under the two flight conditions. This figure highlights a key trend: a higher discretisation generally leads to better optimisation outcomes. As the discretisation level increases, the results for each objective function show a convergence towards a consistent solution.

As one might notice, figure 4.28 misses many optimisation results of the SLSQP and SLSQP KS approach. The reason for this is because those optimisations failed. The unsuccessful termination is not directly related to the optimisation approaches. The optimisations failed because QASTRO could not solve for the state variables anymore at the design points during the optimisation process. The solver for the state variables is described in the Methodology chapter in section 3.3. QASTRO uses the results of the previous state variables as the initial guess for the state variables at the new design state. A possible reason why the non-linear solver of QASTRO is not capable anymore of solving for the state variables is that the SLSQP and the SLSQP KS make too big steps during the optimisation process, which makes the previous result of the state variables an inadequate initial guess. In scenarios where the solver fails repeatedly, the optimisation process is terminated. The significant disadvantage of the SLSQP and the SLSQP KS is the lack of parameters to adjust to overcome this problem potentially.

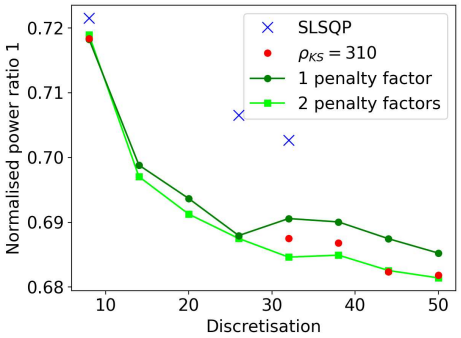
Figure 4.28 illustrates that if the optimisation did not fail the SLSQP approach converged many times to a local minima. Furthermore, we see that the SLSQP KS optimisation approaches are capable of finding solutions at very low and high discretisation levels, but they encounter significant challenges at intermediate levels. Even if we choose a different ρ_{KS} value, the SLSQP KS method has difficulty converging. The best converging behaviour was observed with $\rho_{KS} = 310$. We can reduce the convergence difficulties with very low ρ_{KS} values, but in this case, the optimiser converges to unsatisfactory solutions. The reason for the convergence to unsatisfactory solutions with lower ρ_{KS} values can be seen in figure 4.6a. The higher the number of inequality constraints we wish to approximate using the KS function, the greater the ρ_{KS} value must be



(a) Objective function: Normalised weight



(b) Objective function: Negative normalised endurance



(c) Objective function: Normalised power ratio

FIGURE 4.28 – Aerostructural optimisation with two flight conditions and complete set of design variables. Objective functions as a function of discretisation. For discretisation < 20, LLT discretisation is 20 horseshoe vortices. For discretisation ≥ 20, LLT and FEM discretisations are equal. The number of control points is equal to the LLT discretisation.

to ensure convergence to a satisfactory solution. If we loosen the tight constraint requirements, e.g., by reducing the load factor, the SLSQP and the SLSQP KS approaches have less difficulty converging towards a solution. However, the optimisation approach should not determine the aircraft's requirements.

Luckily, the ALM optimisation approach is more robust and shows, in general, a successful converging behaviour for increasing discretisation. The big advantage of the ALM optimisation approach is that even if an optimisation fails, the user can select different optimisation parameters and try to run the optimisation again. This flexibility is not given for the SLSQP approach and is just in a limited range for the SLSQP KS approach. Nevertheless, even without selecting multiple optimisation parameters, the ALM approach with a single penalty factor did not fail for a single discretisation. This is impressive regarding the complexity of the aircraft model. However, in the context of optimising for weight, the ALM approach also demonstrates its limitations, particularly at higher discretisation levels. As seen in figure 4.28a, at a discretisation level of 44, the ALM approach does not yield the most optimal results. For lower discretisation, the ALM approach found a better solution than for a discretisation level of 44, indicating a potential difficulty in navigating the solution space effectively at this level of model detail.

In optimising endurance, the ALM approach demonstrates a relatively consistent performance advantage. As evidenced in figure 4.28b, the objective function values consistently decrease with an increase in the number of discretisation elements, indicating the method's effectiveness in steadily improving the optimisation results as the model detail is enhanced. This trend suggests that the ALM approach is adept at navigating the endurance optimisa-

tion landscape, particularly as the complexity of the aerostructural model increases with higher discretisation. However, a notable challenge arises with the ALM approach when two penalty factors are used at lower discretisation levels. Specifically, the method struggles at discretisations below 26, and QASTRO non-linear solvers fails to compute the state variables, which makes the optimisation approach fail.

When optimising for the power ratio, figure 4.28c reveals an intriguing aspect of the ALM approach: it converges to different solutions depending on whether single or separate penalty factors are used for discretisations higher than 26. This distinction is crucial as it demonstrates the method's varied response to penalty factor configuration, affecting the optimisation's trajectory and final outcomes. Notably, using a single penalty factor in the ALM approach does not result in asymmetrical solutions, contrary to previous results. This outcome suggests that while separate penalty factors provide a more targeted approach to constraint management, a single penalty factor can still yield balanced and symmetric solutions in the context of power ratio optimisation. This finding underscores the nuanced role of penalty factor configurations in the ALM optimisation process, particularly in achieving efficient and effective solutions for different objective functions. The differing solutions achieved by the ALM approach in the power ratio optimisation are further illustrated in figure 4.33. This figure visually compares the outcomes using single and separate penalty factors, highlighting the distinct paths taken by the optimisation process under each setup. A more detailed analysis of these results, especially in the context of optimising the power ratio, will be discussed later, offering more profound insights into the implications of these varied solutions.

Table 4.7 lists the final objective function values of each optimisation approach with a FEM and LLT discretisation of 50. The table compares the results of the single and two flight conditions. One can see that considering two flight conditions reduces the feasible region, which leads to higher final objective function values. Considering two flight conditions, all SLSQP optimisations at a discretisation of 50 failed. Nevertheless, the SLSQP KS optimisations did not fail for the discretisation of 50 and showed comparable results to the ALM optimisation results.

	Normalised weight		**Neg. normalised endurance**		**Normalised power ratio**	
	1 cond.	2 cond.	1 cond.	2 cond.	1 cond.	2 cond.
SLSQP	0.3637	-	−1.521	-	0.6773	-
SLSQP KS	0.3640	0.3832	−1.514	−1.486	0.6800	0.6818
ALM	0.3637	0.3836	−1.521	−1.488	0.6774	0.6814

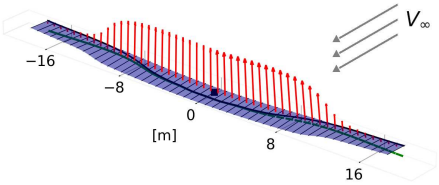
TABLE 4.7 – Final objective function values of optimisation approaches with two flight conditions and FEM and LLT discretisation of 50. ALM approach with two penalty factors.

4.2.2.2 Analysis of optimisation results

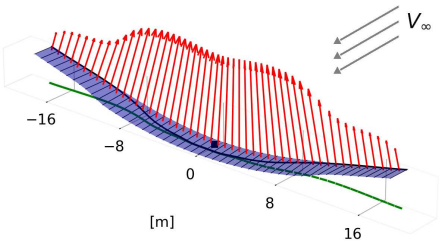
Figure 4.29 shows the aircraft design optimised regarding the weight, endurance and power ratio. The figure shows the result of the ALM approach with a discretisation of 50. The lift distribution of the aircraft closely resembles that of an optimised aircraft under a single condition, as depicted in figure 4.27, upon initial observation. However, a detailed review will show some essential differences.

Figure 4.30 reveals critical differences in the Helios aircraft's design when optimising for weight under single and two-flight conditions. Notably, the tubular wing spar's thickness increases under two-flight conditions (figure 4.30a), a necessary change to support higher lift forces. The chord length also extends in the two-flight scenario (figure 4.30b), aiding in greater lift generation. Figure shows 4.30c a similar twist distribution as from the previous results,

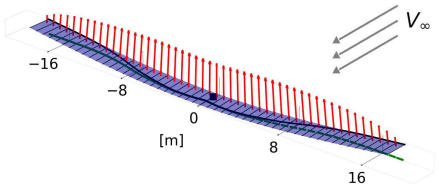
Cruise condition | **High load condition**



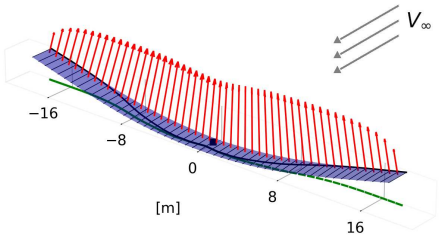
(a) Objective function: Normalised weight



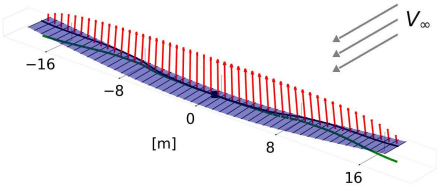
(b) Objective function: Normalised weight



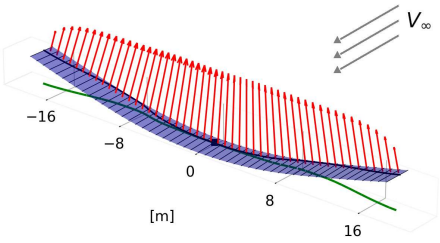
(c) Objective function: Negative normalised endurance



(d) Objective function: Negative normalised endurance



(e) Objective function: Normalised power ratio



(f) Objective function: Normalised power ratio

FIGURE 4.29 – Aerostructural ALM optimisation with two flight conditions and complete set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α_1, α_2, $\boldsymbol{c}$, $V_{\infty\,1}$, $V_{\infty\,2}$, $\boldsymbol{x}_{\text{Beam}}$. Figures show left the cruise condition and right the high load condition. Optimisation result of ALM with 50 control points with FEM and LLT discretisation of 50. Optimisation parameters: $\rho_0 = 0.005$, $\gamma = 1.1$, $\boldsymbol{\lambda}_0 = \mathbf{0}$, $\boldsymbol{\mu}_0 = \mathbf{0}$ and separate penalty factors for equality and inequality constraints. $r = 1$ for normalised weight optimisation. $r = 0.75$ for endurance and power ratio optimisation.

shown in figure 4.17b. However, more horseshoe vortices have a higher twist angle when considering two flight conditions. The dashed line indicates the total aerodynamic angle at the high load condition. Additionally, the beam's location shifts closer to the leading edge at the wing tips in the two-flight conditions (figure 4.30d), resulting in the leading edge twisting downwards due to the lift forces, effectively reducing the angle of attack at the wing tips. At the high load condition, the aerodynamic load shifts towards the root, which enhances structural resistance to bending. These adaptations underscore the intricate design modifications needed to balance weight optimisation with structural and aerodynamic demands in varying flight conditions. When optimising with two flight conditions, we could reduce the weight of the Helios aircraft from an initial weight of $541.52\,kg$ to $208.01\,kg$. However, please keep in mind that the initial design is not feasible. Moreover, please note that regardless of whether we optimise with a single or two flight conditions, the weight-optimised solution shows that the incidence angles are higher at the wing roots compared to the wing tips with an abrupt change in the twist angle. This leads to the non-elliptical lift distribution shown in figure 4.27a and 4.29a. This lift distribution can make the weight optimisation questionable.

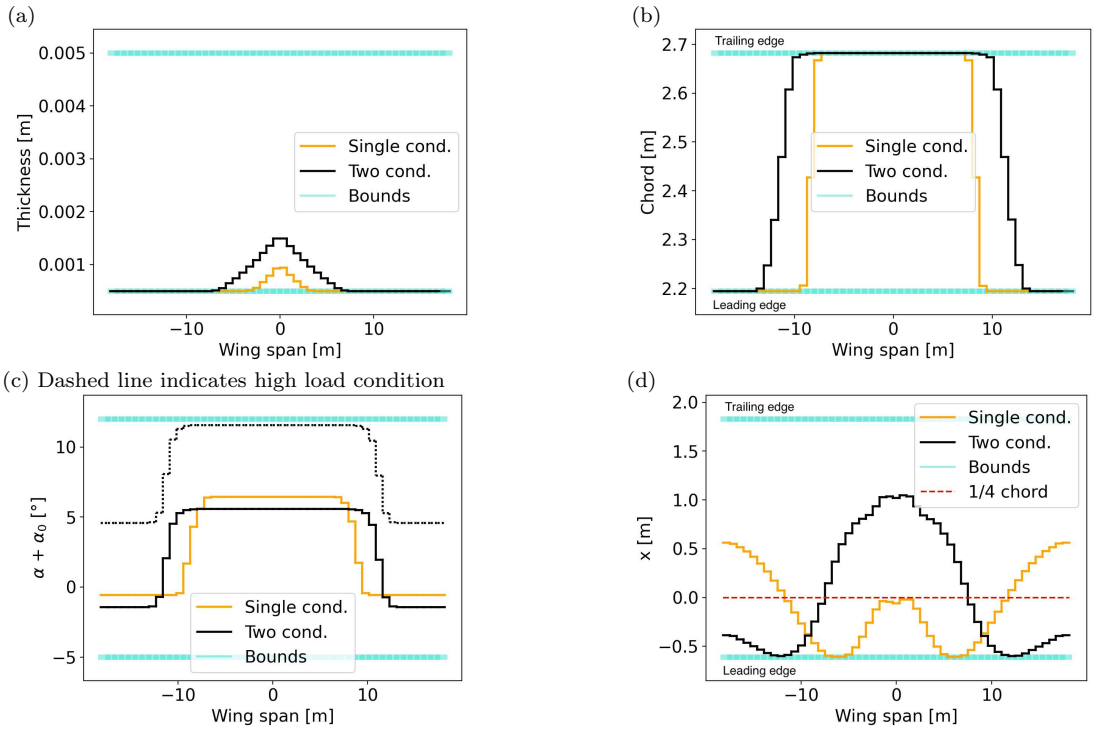


FIGURE 4.30 – Objective function: Normalised weight; complete set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α_1, α_2, $\boldsymbol{c}$, $V_{\infty 1}$, $V_{\infty 2}$, $\boldsymbol{x}_{\text{Beam}}$; aerostructural ALM optimisation comparison between single and two flight conditions. The optimisation results with 50 control points, FEM and LLT discretisation of 50 and two independent penalty factors. Result of the remaining design variable with single condition: $V_\infty = 31.52\,m/s$ (upper bound). Result of other design variables with two conditions: $V_{\infty 1} = 28.62\,m/s$, $V_{\infty 2} = 31.52\,m/s$ (upper bound).

Figure 4.31 presents the design variable results when optimising the Helios aircraft for endurance. One can see that for both optimisation models, the chord length is at its lower bound. This is because a shorter chord length increases, the aspect ratio and thus reduces the aircraft's drag and consequently increases the endurance. A crucial observation is the similar total aerodynamic angle across single and two-flight conditions, indicating consistent aerodynamic profiles despite varied operational demands. Notably, in the two-flight condition, there is a restraint on increasing the twist angle, which is essential to prevent exceeding the stall angle. Differences in frame element location are also evident; the frame element is positioned closer to the leading edge in the two-flight conditions, particularly at the wingtips. This strategic placement counters excessive upward bending of the wings, which is crucial for maintaining optimal aerodynamic performance under cruise condition requirements. When optimising with two flight conditions, we could increase the endurance of the Helios aircraft on the batteries from 2.64 hours to 3.93 hours.

Figure 4.32 showcases the design variable outcomes when optimising the Helios aircraft for the power ratio. Notably, the tubular wing spar thickness remains consistent across single and two flight conditions, underscoring similar structural demands. The chord length is at its upper limit in both scenarios, maximising the solar panel area for enhanced energy production. Variations in the total geometric angle of attack are observed, with higher angles in the two-flight condition. One might interpret the result as the aircraft requiring more lift in cruise conditions when optimising with two flight conditions. However, this is not the case. The reason for the lower aerodynamic angle result when optimising with a single flight condition can be seen in figure 4.32d. In the single-condition optimisation, the frame element location did not change significantly. In contrast, when optimising with two flight conditions, the frame element location changed such that the wing had a stronger resistance against upward bending. Consequently, the optimised wing using only a single flight condition will twist more than the optimised wing

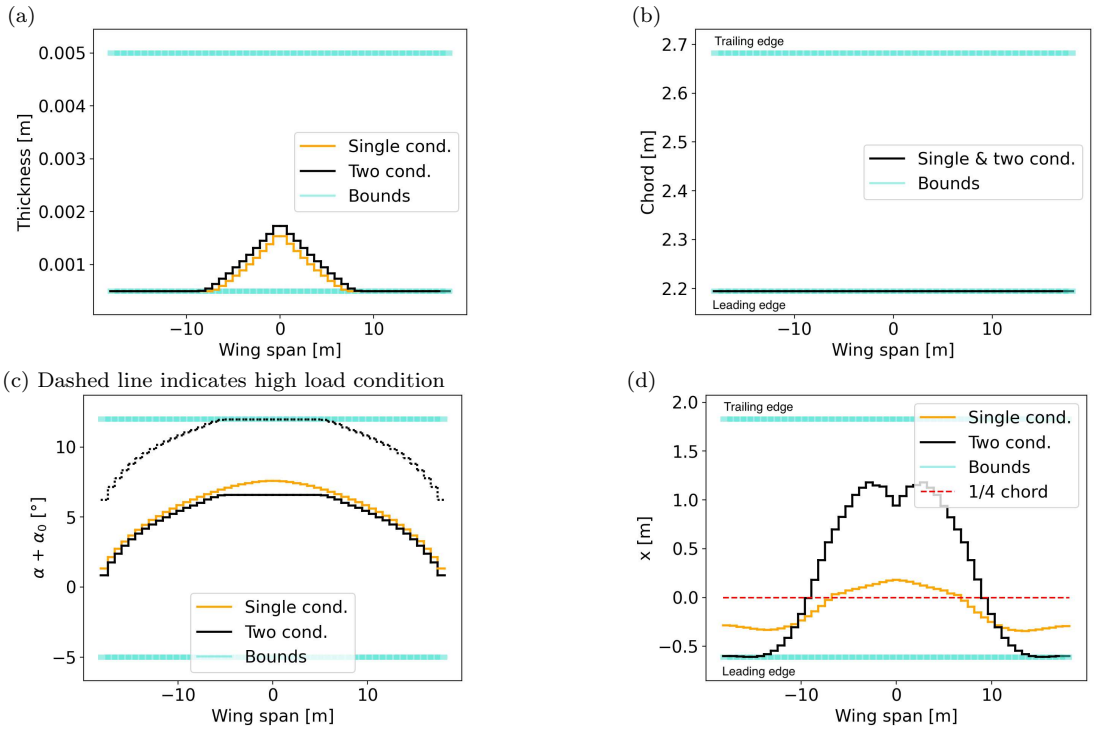


FIGURE 4.31 – Objective function: Negative normalised endurance; complete set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α_1, α_2, $\boldsymbol{c}$, $V_{\infty\,1}$, $V_{\infty\,2}$, $\boldsymbol{x}_{\text{Beam}}$; aerostructural ALM optimisation comparison between single and two flight conditions. The optimisation results with 50 control points, FEM and LLT discretisation of 50 and two independent penalty factors. Result of the remaining design variable with a single condition: $V_\infty = 25.79\,m/s$ (lower bound). Result of other design variables with two conditions: $V_{\infty\,1} = 25.79\,m/s$ (lower bound), $V_{\infty\,2} = 31.52\,m/s$ (upper bound).

using two flight conditions for optimising. This additional twist increases the angle of attack, providing more lift. The frame element location differences reflect the structural adaptations necessary for maintaining aerodynamic performance, particularly under the increased demands of the two-flight condition. These adjustments highlight the intricate design considerations crucial for optimising the power ratio, balancing energy production, and aerodynamic efficiency.

The Helios aircraft's wings are approximately 75% covered with solar panels. If we assume that the solar panles can produce $c_{\text{Solar panel}} = 205.83\,W/m^2$, which is in a realistic range for modern solar panels (RIBAH; RAMAYANTI, 2018), then we could decrease the power ratio from 0.17 to 0.12 when considering two flight conditions. This means we increase power production over the power consumption from 5.78 to 8.48. According to Flittie and Curtin (1998), the power ratio of the Pathfinder configuration at take-off is 4 and even higher in cruise conditions. Our optimised Helios aircraft shows an improvement in the power ratio, significantly surpassing the benchmarks set by the Pathfinder configuration. Please note that this ratio will only be achieved under optimal conditions, usually not given.

Figure 4.33 illuminates the differing results of the ALM optimisation for the power ratio under single and separate penalty factor configurations. A key distinction lies in the frame element location: with separate penalty factors, the frame element is nearer the leading edge, enhancing resistance to wing bending and affecting aerodynamic properties. Conversely, using a single penalty factor, the beam's positioning allows more upward bending of the wing. This bending increases the angle of attack, enabling sufficient lift generation without maxing out the angles. Consequently, the angle of attack plus twist does not reach its upper limit in the single penalty factor solution. This dynamic, especially evident in a discretisation of higher than 26, showcases how varying penalty factor setups in the ALM lead to notably different optimisation outcomes for the power ratio, reflecting the complexities of balancing energy production and aerodynamic efficiency.

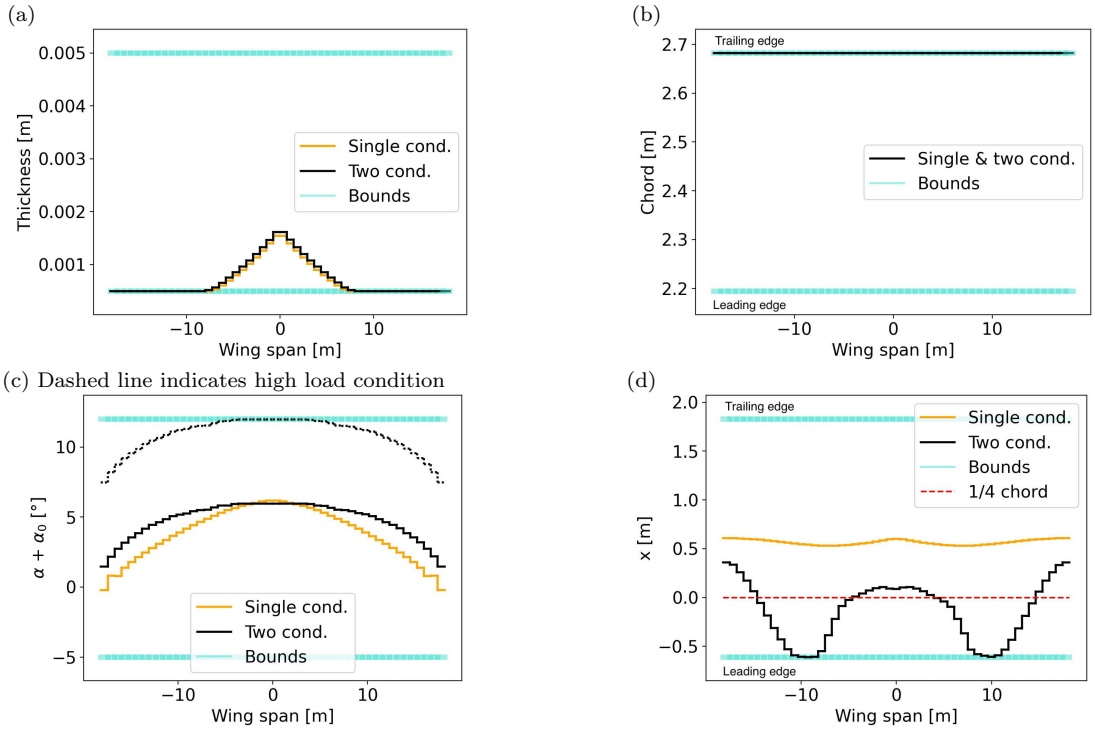


FIGURE 4.32 – Objective function: Normalised power ratio; complete set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α_1, α_2, $\boldsymbol{c}$, $V_{\infty\,1}$, $V_{\infty\,2}$, $\boldsymbol{x}_{\text{Beam}}$; aerostructural ALM optimisation comparison between single and two flight conditions. Optimisation result with 50 control points, FEM and LLT discretisation of 50 and two independent penalty factors. Result of the remaining design variable with a single condition: $V_\infty = 25.79\,m/s$ (lower bound). Result of other design variables with two conditions: $V_{\infty\,1} = 25.79\,m/s$ (lower bound), $V_{\infty\,2} = 31.52\,m/s$ (upper bound).

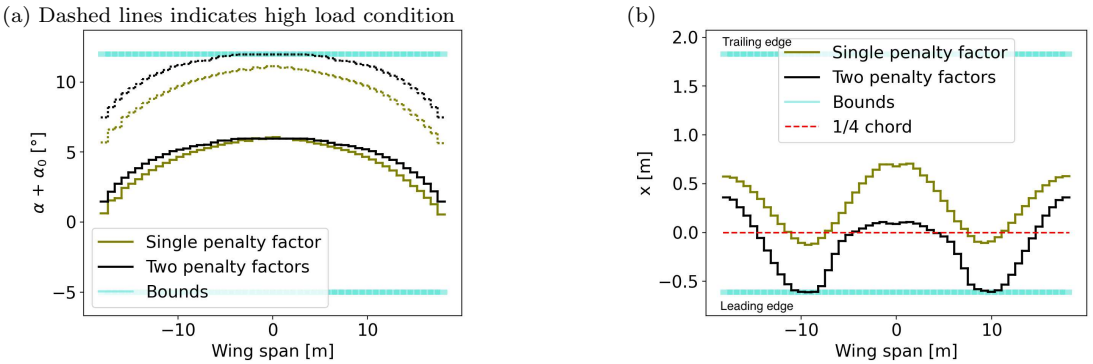


FIGURE 4.33 – Objective function: Normalised power ratio; complete set of design variables: $\boldsymbol{t}$, $\boldsymbol{\alpha}_0$, α_1, α_2, $\boldsymbol{c}$, $V_{\infty\,1}$, $V_{\infty\,2}$, $\boldsymbol{x}_{\text{Beam}}$; aerostructural ALM optimisation with two flight conditions. The optimisation results with 50 control points, FEM and LLT discretisation of 50. Comparison of results with single and two penalty factors. Results of other design variables are identical with single and two penalty factors: $V_{\infty\,1} = 25.79\,m/s$ (lower bound), $V_{\infty\,2} = 31.52\,m/s$ (upper bound).

5 Discussion

In Chapter 5, we critically evaluate and interpret the findings presented in Chapter 4, with a particular focus on the methodologies and results of the optimisation techniques. The chapter is structured to provide an in-depth discussion of these outcomes, aiding in the understanding of their implications and relevance. This chapter is divided into two separate sections. The first section discusses the findings of the structural optimisation, and the second section elaborates on the outcomes of the aerostructural results.

5.1 Structural optimisations

The section 4.1 focuses on structural optimisation results, emphasising the significance of the underlying assumptions and methodological choices. A pivotal assumption in the model is using aluminium for the wing spar, deviating from the actual diverse material composition like Kevlar of the wing spar of the Helios aircraft. This simplification for structural modelling, by assuming a tubular wing spar represents all wing properties, may not capture the full complexities of actual wing structures. Moreover, a linear FEM model is not appropriate for highly-flexible wings such as for the Helios aircraft. Additionally, the study concentrates solely on structural optimisation, excluding aerostructural aspects. This methodological choice simplifies the analysis but overlooks the crucial interaction between structural and aerodynamic models, vital in real-world scenarios for comprehensive understanding of aerostructural dynamics. The decision to distinguish structural optimisation from aerostructural analysis aims to clarify the structural optimisation landscape, evaluating three optimisation approaches (SLSQP, SLSQP KS, and ALM) within a simplified structural model context. While these findings are informative, they represent an idealised scenario, and their real-world applicability should be considered in light of these methodological constraints. Future analysis and interpretation of these optimisation results must remain cognisant of these foundational methodology aspects and inherent assumptions. It is essential to highlight that this study focuses on comparing optimisers' performance as the problem complexity scales.

5.1.1 Discussion on SLSQP KS approach

In our study of the SLSQP KS approach, the flexibility of the ρ_{KS} parameter was pivotal. Higher ρ_{KS} values improved optimisation outcomes, consistent with the anticipated behaviour of inequality constraint approximation in this approach. The SLSQP KS notably converged towards SLSQP solutions, validating its effectiveness and showing alignment with our initial hypothesis. A key observation was the significant reduction in optimisation time when employing the KS function for constraint aggregation, underscoring the approach's efficiency, especially in time-sensitive scenarios. Regarding the ρ_{KS} parameter specifics, we limited its range to a maximum of 310, aiming to balance optimisation time against result quality. In contexts where time efficiency is crucial, further increasing ρ_{KS} may not correspondingly enhance result accuracy. For setting

the tolerance for optimisation results, we established a criterion that considers outcomes to be acceptable if they deviate by up to a quarter of a percent from the objective function values of the SLSQP and ALM methods, with this deviation measured against the initial design, i.e., the baseline. This threshold, translating to about a 1.35-kilogram difference in our study, met our optimisation goals and allowed for effective comparisons between the three approaches. However, there is room for discussion on this threshold. One could set this threshold appropriately to their optimisation goals.

5.1.2 Discussion on ALM approach

In examining the ALM approach, an essential aspect of our analysis was exploring the wide range of available parameter choices. Notably, the γ parameter considerably affected the optimisation time, which deviated from our initial expectations; $\gamma = 1.1$ exhibited extended durations. This suggests that a higher penalty factor is necessary to achieve convergence, contrary to our assumption that the Lagrange multipliers would effectively drive all active constraints to zero. Furthermore, we anticipated an increase in optimisation time with overly conservative parameters, as the ALM approach tends to morph into a penalty approach, diminishing the benefits of updating Lagrange multipliers. However, this anticipated trend was not distinctly evident in the result data, presenting a notable discrepancy between expected and observed outcomes. The choice of initial penalty factor, ρ_0, was a critical decision in our methodology. We kept ρ_0 at a maximum value of 0.5 based on the understanding that this setting moderates the influence of constraint gradients in the initial optimisation phases. This was a strategic choice to prevent constraints from dominating the optimisation process prematurely. Regarding the decision to increase the penalty factor, our approach diverged from the common practice of incrementing it in each iteration. Instead, we opted to increase it only when there was inadequate progress towards a feasible solution. This methodology, inspired by the findings of Birgin and Martínez (2019), proved effective in our case, indicating that a less frequent increase in the penalty factor can be beneficial in specific scenarios. Additionally, the ALM approach showed shorter optimisation times for high discretisation compared to both the SLSQP and SLSQP KS approaches. While it was anticipated to outperform the SLSQP in terms of time efficiency, its superiority over the SLSQP KS approach indicates the effectiveness of the ALM.

5.1.3 Discussion on the usage of B-Splines

Integrating B-splines in optimisation, especially in SLSQP KS and ALM approaches, yields diverse outcomes. In SLSQP KS, B-splines significantly reduce optimisation time, mainly due to fewer design variables, challenging prior assumptions about optimisation efficiency. Conversely, their application in the ALM approach does not expedite optimisation, indicating a possible lesser sensitivity to design variable quantity in ALM or a more significant influence of other factors like hyperparameter adjustments. The SLSQP KS approach updates only the design variables at each iteration, whereas the ALM approach needs to update the design variables and the hyperparameters. Effectively, the ALM updates quantitatively more parameters than the SLSQP KS approach due to the Lagrange multipliers updates. This finding emphasises the role of hyperparameter management in ALM's optimisation efficiency. These observations underscore the need for further investigation into the interplay between design variables, hyperparameter adjustments, and different optimisation methods. The variable impact of B-splines on different optimisation strategies highlights their versatility and suggests a tailored approach to optimisation tool selection based on specific problem characteristics. Overall, B-splines' role in structural optimisation demonstrates a complex interaction of factors affecting efficiency, offering valuable insights for optimising tool utilisation in various scenarios.

5.2 Aerostructual optimisation

In this analysis, we critically examine the methodology underlying our aerostructural optimisation results, which is essential for contextualising the findings within the broader optimisation framework. The methodology incorporates chord length and freestream velocity variations, impacting the Reynolds number and lift curve. In our model, we assume that the parameters defining the lift and drag curves remain constant across the entire spectrum of Reynolds numbers considered. Our investigation indicates that alterations up to 10% in these parameters have negligible effects on lift and drag, affirming the approximation's suitability for the specified conditions. However, there is room for discussion to select a higher or lower percentage to alter those parameters.

A significant assumption in our approach is that changes in chord length do not affect the aircraft's weight, primarily due to the simplified representation of the aircraft structure with a tubular wing spar. Furthermore, we assumed that the location of the tubular wing spar could vary anywhere along the chord length of the wing. This assumption is unrealistic, as the wing spar's diameter at specific points exceeds the wing thickness. A better solution is to establish a coupling between the wing thickness and the diameter of the tubular wing spar to maintain structural feasibility. Additionally, the study imposes bounds on twist angles between 5° and -2°, which can be questioned. The authors chose to balance the optimisation benefits, which could be obtained by loosening the tight bound constraints on the twist angle against potential stall risks and pilot control limitations.

Moreover, considering structural twist bending, which is not confined by the bounds of the twist angle, adds another layer of complexity to the aerostructural model. Structural twist bending can increase the angle of attack. This additional increase can result in an angle of attack higher than 12° and thus in stall behaviour. When combined with the downwash, which reduces the angle of attack, this effect presents a balancing act within the model. This means that even if the twist bending increases the angle of attack by more than 12°, the downwash will, on the other hand, reduce the angle of attack, which can prevent the stall behaviour. However, it is essential to acknowledge that this balance is an assumption within the model, and the precise interplay of these factors could vary in real-world flight conditions.

Next, our decision to exclude wingspan as a design variable was motivated by the complex integration challenges within QASTRO. Including the wingspan as a design variable would have introduced additional computational complexities, detracting from the study's primary focus areas.

Lastly, choosing a relatively high load factor of three in our optimisation model warrants discussion. While NASA's paper (FLITTIE; CURTIN, 1998) suggest that the Helios aircraft is capable of withstanding accelerations up to 3.2 g, we consciously opted for a load factor of three to ensure feasibility in our optimisation solutions. It is important to contextualise this decision: using a load factor as high as three, albeit lower than the aircraft's maximum capability, was driven by the need to challenge its design within realistic operational limits while still achieving viable optimisation results. This load factor, though seemingly high, was carefully chosen to strike a balance between testing the structural integrity of the aircraft under demanding conditions and the practical constraints of the optimisation process.

While characterised by certain simplifications and assumptions, the methodology applied in our aerostructural optimisation study was critical in navigating the complexities inherent in integrating structural and aerodynamic models. These choices reflect a balance between computational feasibility and the pursuit of realistic and applicable optimisation results, laying the groundwork for the subsequent analysis of optimisation approaches and their outcomes.

5.2.1 Discussion on SLSQP approach

In the aerostructural optimisation context (section 4.2), the performance of the SLSQP method diverged notably from initial expectations. Initially selected based on its efficacy in structural optimisation (section 4.1), the SLSQP approach faced challenges in more complex aerostructural scenarios. While it excelled in more straightforward optimisations involving a limited set of design variables and a single flight condition, its limitations became evident by introducing a complete set of variables and multiple conditions. Contrary to our expectations, SLSQP struggled to find the global minimum, often becoming trapped in local minima. This unexpected outcome highlights the method's constraints in handling the increased complexity of aerostructural optimisation.

5.2.2 Discussion on SLSQP KS approach

For complex aerostructural optimisation, the SLSQP KS approach generally converged to lower objective function values than the SLSQP approach. This result deviates from our hypothesis, considering the SLSQP KS approach's foundational assumption of aggregating all inequality constraints into a single scalar function. Contrary to conventional expectations, this approximation proved beneficial, assisting the optimiser in avoiding local minima. This outcome highlights the SLSQP KS approach's capacity to effectively navigate complex optimisation landscapes, a significant insight into its potential utility in aerostructural optimisation.

5.2.3 Discussion on ALM approach

In the context of aerostructural optimisation, the ALM approach provided some key observations. Notably, the necessity for slightly conservative parameters to converge to optimal solutions was unexpected. This requirement underscores the uniqueness of the aerostructural optimisation problem and suggests that different optimisation challenges might demand varying parameter strategies.

A crucial element of our ALM methodology was using two independent penalty factors for equality and inequality constraints. While potentially more complex than simply scaling constraints, this approach offered distinct advantages. It addressed the varying challenges posed by different types of constraints, depending on the objective function. For instance, optimising for weight made satisfying the lift constraint more challenging, while objectives like endurance or power ratio presented more significant difficulties with structural constraints. Independent penalty factors allowed the optimiser to focus on different constraint types sequentially, effectively reducing the problem's complexity in specific scenarios. However, this strategy opens up new considerations. The idea of using separate penalty factors for each constraint leads to a potential trade-off. While it offers more nuanced control over the optimisation process, it also adds complexity. In fact, our analysis revealed that the inclusion of more penalty factors could detrimentally increase this complexity. This observation was echoed in the results of the structural optimisations, particularly in the section on B-splines, where it was noted that a significant portion of optimisation time was consumed by managing numerous hyperparameters. Thus, there's a balance to be struck between the level of control provided by multiple penalty factors and the increased complexity they introduce.

6 Conclusions & Recommendations

In Chapter 6, we combine the essential findings and insights from our investigation into constrained optimisation methods for aerostructural design. For our analysis, we utilised a program called QASTRO. This chapter is pivotal as it crystallises the research outcomes into conclusive statements and forward-looking recommendations. It is structured into two distinct sections: firstly, the conclusions, and secondly, the recommendations. This chapter aims to provide a clear, comprehensive encapsulation of the study, highlighting its contributions to the field of aerostructural optimisation.

6.1 Conclusions

In section 6.1, we comprehensively evaluate the research findings in light of the hypotheses set forth at the beginning of this thesis. Central to our investigation was the comparative assessment of three constrained optimisation approaches – SLSQP, SLSQP KS, and ALM – within the context of aerostructural design. Our hypotheses posited that the SLSQP approach would yield the most favourable results regarding objective function values but at the expense of longer optimisation times. For the SLSQP KS approach, we anticipated slightly lower objective function values with increased efficiency, while the ALM approach was hypothesised to offer both speed and accuracy in its results. This section systematically revisits these hypotheses, aligning them with the results gathered to determine the extent to which our research validated or refuted them.

6.1.1 Conclusion of SLSQP approach

The SLSQP approach, when applied to structural and aerostructural optimisation, presented a nuanced performance profile that both confirmed and contradicted our initial expectations. SLSQP effectively minimised objective function values in more straightforward structural optimisation scenarios, demonstrating its robustness in line with our hypotheses. It showed particular strength in contexts with fewer variables and constraints, affirming its predicted dominance in less complex optimisation tasks. However, its optimisation time is considerably longer than the SLSQP KS and ALM approaches.

However, the approach encountered significant challenges in more intricate aerostructural scenarios, especially at higher discretisation levels. Contrary to our expectations, SLSQP struggled to consistently identify the global minimum, revealing limitations in its capacity to navigate the complexities of aerostructural optimisation. A particularly notable issue was the failure of the SLSQP approach to converge to a solution in several complex aerostructural optimisations. This failure was attributed to the inability of the non-linear Python solvers within QASTRO to compute the state variables during the optimisation process. This limitation severely hindered the effectiveness of the SLSQP approach in these scenarios, marking a substantial impediment to its application in complex aerostructural design tasks. Additionally, the SLSQP approach's

tendency to become ensnared in local minima, mainly due to its individual handling of inequality constraints and lack of adjustable parameters, emerged as a significant barrier. This aspect further limited its adaptability and manoeuvrability in intricate optimisation landscapes.

In conclusion, while the SLSQP approach affirmed its strengths in more straightforward settings, its challenges in complex aerostructural optimisations, including convergence issues, offered more profound insights into its suitability for various optimisation needs within the realm of aerostructural design.

6.1.2 Conclusions of SLSQP KS approach

The SLSQP method combined with the KS function for aggregating the inequality constraints in structural and aerostructural optimisation brought to light several vital conclusions, some aligning with our hypotheses and others revealing new insights.

A critical finding was that for more straightforward problems, including structural optimisations and aerostructural optimisations with fewer design variables, the SLSQP KS approach converged to results similar to the SLSQP approach. This corroborated our hypothesis and highlighted the effectiveness of the KS function in approximating inequality constraints. The quality of this approximation, and consequently the convergence to the SLSQP solution, was contingent on the ρ_{KS} value. A higher ρ_{KS} enhanced the approximation quality, allowing the optimiser to potentially converge to better solutions. For structural optimisations, we could observe with $\rho_{KS} = 160$ a consistent convergence to objective function values not higher than 0.25% (of the baseline) compared to the SLSQP or ALM solution. Moreover, we observed a substantially reduced optimisation time with the SLSQP KS approach compared to SLSQP. For a highly discretised structural optimisation with 80 frame elements, and thus a constraint-rich optimisation, the SLSQP KS is more than five times faster than the SLSQP approach. This efficiency gain stemmed from the simplification offered by representing all inequality constraints within a single scalar KS function, reducing the complexity of derivative computations. However, the SLSQP KS approach exhibited limitations in more complex aerostructural optimisation scenarios, mainly when ρ_{KS} values were set too high. In such cases, the KS function approximates a *max* function, which is only piecewise differentiable and poses challenges in gradient computation. This issue was evident in complex problems with many design variables and high discretisation levels, where the approach struggled to converge to the global solution. In constraint-rich optimisations, a $\rho_{KS} = 310$ demonstrated a favourable balance between solution quality and the differentiability of the KS function. Interestingly, in complex aerostructural optimisations, the SLSQP KS approach often outperformed the SLSQP regarding the solution quality $f(\boldsymbol{x}^*)$. This superior performance is attributed to its problem simplification, aiding the optimiser in navigating towards the global minimum more effectively. The significant impact of B-Splines on reducing optimisation time with the SLSQP KS approach further indicated that its efficiency is closely tied to the number of design variables. However, like SLSQP, the SLSQP KS approach faced difficulties in complex scenarios, often failing to converge to a solution. Like SLSQP, this failure was linked to the non-linear Python solvers within QASTRO. The solvers could not compute state variables during the optimisation process, suggesting that this issue is not mitigated merely by the simplification provided by the KS function.

These findings paint a comprehensive picture of the SLSQP KS approach's capabilities and limitations, offering valuable insights into its applicability and performance in varying contexts of aerostructural optimisation.

6.1.3 Conclusions of ALM approach

Evaluating the ALM approach across structural and aerostructural optimisation provided insightful conclusions about its performance and parameter sensitivities. In structural optimisation problems, independent of the parameter choice, the ALM approach consistently converged to the same solutions as the SLSQP approach, affirming its effectiveness. The choice of parameters notably influenced the optimisation time; highly conservative parameters led to quicker convergence, while slightly conservative settings resulted in more extended optimisation periods. This was particularly evident in the role of the penalty factor increase rate (γ), where a lower γ value, leading to a gradual increase in the penalty factor (ρ), extended the optimisation time. This need for a higher penalty factor to precisely drive active constraints to zero underlined the importance of penalty factor management in the ALM approach. In scenarios with high discretisation, the ALM approach showed remarkable efficiency, mainly due to its requirement to compute derivatives only for the augmented Lagrangian function, unlike the multiple derivative computations needed in SLSQP and SLSQP KS approaches. In a highly discretised Helios aircraft model (e.g. 80 frame elements for one wing), the ALM with moderate conservative parameters is ten times faster than the SLSQP and about 40% faster than the SLSQP KS approach with $\rho_{KS} = 160$. With more conservative ALM parameters, the ALM approach can be up to almost twice as fast as the SLSQP KS approach. For structural optimisations, moderately conservative parameters, such as $\rho_0 = 0.2525$, $\gamma = 2.05$, and $r = 0.5$ showed a good trade-off between convergence speed and constraint management.

Incorporation of B-splines had minimal impact on ALM optimisation time, highlighting hyperparameter updates, especially those of the Lagrange multipliers, as the main time-consuming component. However, increasing control points led to diminished effects of ALM parameters on optimisation time.

In the more complex realm of aerostructural optimisation, the ALM approach's best results were achieved with slightly conservative parameters, such as $\rho_0 = 0.005$, $\gamma = 1.1$, $r = 1.0$. Conversely, highly conservative parameters frequently resulted in local minima convergence, suggesting a need for balanced parameter selection. Instances of low penalisation, meaning minimal conservatism, sometimes led to oscillatory behaviour in the optimiser, impacting the optimisation time without necessarily degrading the solution quality. This oscillatory tendency could be effectively countered by reducing the r value, thereby enhancing the optimisation process's efficiency. The ALM approach occasionally converged to asymmetric solutions, a challenge addressed by employing two separate penalty factors for equality and inequality constraints. This strategy allowed the optimiser to sequentially focus on each constraint type, reducing problem complexity and aiding in achieving more symmetric solutions.

A standout aspect of the ALM approach was its robust convergence behaviour, especially in highly complex scenarios involving a complete set of design variables and two flight conditions. Its strategy of dividing the optimisation into multiple unconstrained sub-optimisations contributed to this robustness, showcasing the method's potential as a powerful tool in complex optimisation scenarios, contingent upon careful parameter selection and management.

In summary, the ALM approach demonstrated a high degree of effectiveness and adaptability in both structural and aerostructural optimisation, with its performance intricately linked to the judicious selection and management of its parameters.

6.1.4 Overall findings

Incorporating multiple optimisation approaches within QASTRO marks an advancement in the field of aerostructural optimisation, offering enhanced flexibility and choice for users.

The addition of the SLSQP and ALM approaches to the previously sole SLSQP KS approach represents a notable expansion of the tool's capabilities.

One key implication of this expansion is the increased user flexibility in addressing various optimisation challenges. Users can now opt for the straightforward SLSQP approach without delving into complex parameter settings for simple optimisation tasks where time efficiency is not a primary concern. This simplification is particularly beneficial for users focused on less intricate problems, providing a hassle-free optimisation process. Another significant advantage is the ability to switch between optimisation approaches, which is especially useful when there are concerns about convergence to local minima. This flexibility enables users to explore alternative solutions and verify the robustness of their optimisation results, enhancing the reliability of the outcomes. An important insight relevant to users of QASTRO and similar platforms is the impact of discretisation on the optimisation results. Our research suggests that starting with a lower discretisation level and gradually increasing it can be a strategic approach to ensure convergence towards the global solution. Particularly in complex aerostructural optimisations, discretisation plays a crucial role in the solution's quality and optimisation time. It was observed that increasing discretisation beyond a certain threshold (e.g., 50) does not significantly enhance the objective function value but leads to longer optimisation times. This finding is vital for users in managing the trade-off between solution accuracy and computational efficiency.

In essence, these findings underscore the importance of approach selection and parameter management in aerostructural optimisation. The enhanced options in QASTRO, coupled with insights into discretisation and optimisation management, provide users with a more robust and versatile framework for tackling diverse aerostructural challenges.

6.2 Recommendations

In section 6.2, we provide a series of recommendations derived from the findings and analyses of our research. These suggestions are designed to enhance the application and effectiveness of the three optimisation approaches within QASTRO and to inform future research and practical applications in aerostructural optimisation. The section is methodically divided to address each key aspect of our study, offering specific guidance on QASTRO's overall development, as well as the SLSQP, SLSQP KS, and ALM approaches. We aim to contribute actionable insights that can facilitate the ongoing evolution and refinement of aerostructural optimisation techniques.

6.2.1 Recommendations for QASTRO

In the context of enhancing QASTRO, several recommendations emerge from our research. These aim to broaden the software's capabilities and improve its efficiency in handling complex aerostructural optimisation problems.

Firstly, the incorporation of buckling analysis in QASTRO is recommended for more intricate structural models. While the current model does not account for buckling, its consideration could be critical in analysing complex structures. However, given the complexity of implementing buckling analysis, this enhancement should be pursued if buckling is deemed a significant factor in the analysed structures. Furthermore, we recommend exploring more efficient solutions for solving linear equations in structural optimisations. For structural analysis, the residual function for solving the displacements is linear. Currently, QASTRO employs a non-linear solver for solving for the displacements, which, while effective, may not be the most efficient method. There is potential for significant improvements in computational efficiency by adopting advanced linear solvers. Similar efficiency gains can be achieved in solving for adjoint variables. The process, which is always linear regardless of whether the optimisation is structural or aerostructural,

currently utilises the same non-linear solver. Adopting more specialised linear solvers for adjoint variable computations could enhance the overall efficiency and speed of the optimisation process in QASTRO. In light of our experiences, there are currently no reliable linear Python solvers; we strongly recommend pursuing further research and development in this area.

An additional vital recommendation for QASTRO is the implementation of symmetry conditions. Currently, the inability to apply symmetry in QASTRO limits the optimisation process, especially in scenarios where symmetrical solutions are desired or more efficient. Including symmetry conditions would not only reduce the complexity of the problems but also significantly improve the performance of all optimisation approaches. This feature is particularly crucial for the ALM optimiser, which has shown difficulties in converging to symmetrical solutions. Enabling symmetry conditions could be a game-changer for the ALM approach, offering substantial benefits in terms of both efficiency and solution accuracy.

Additionally, a critical area for further investigation within QASTRO pertains to the challenges encountered in computing state variables during complex optimisation tasks, particularly under the SLSQP and SLSQP KS approaches. Our research identified frequent instances of failure, particularly in complex problem scenarios and when considering two flight conditions. This can be attributed to the inadequacies of the non-linear Python solver employed by QASTRO in computing state variables. The optimisation problem 3.34 describes the state variable computation. This issue not only hinders the successful application of these optimisation approaches but also raises concerns about the solvers' capacity to handle highly intricate aerostructural problems. Therefore, conducting in-depth research and development efforts to resolve this computational challenge is strongly recommended. Addressing this issue will not only enhance the robustness and reliability of QASTRO in advanced optimisation scenarios but also significantly expand its applicability to a broader range of complex aerostructural designs.

In computing the state variables, the full Jacobian matrix is required in the optimisation problem 3.34. However, only the Jacobian-vector product, which can be efficiently computed using AD in a single execution, is necessary. Computing the full Jacobian matrix is computationally expensive, requiring as many executions as there are rows or columns in the matrix. This cost is significant, as the number of rows in the Jacobian is six times the number of FEM nodes plus the number of horseshoe vortices. We suggest modifying the Python solver used to solve equation 3.34 to allow users to input the directional derivative, i.e., the Jacobian-vector product. Though not the focus of this thesis, providing the directional derivative can significantly reduce optimisation time in aerostructural optimisations, particularly with the SLSQP and ALM approaches. These methods spend a considerable amount of time computing gradients, as illustrated in figure 4.13.

These recommendations are proposed to make QASTRO a more versatile and powerful tool for aerostructural optimisation, capable of handling a more comprehensive range of design considerations and computational challenges.

6.2.2 Recommendations for SLSQP approach

In the context of refining the SLSQP approach within QASTRO, our study leads to two primary recommendations to enhance its efficiency and effectiveness.

The first recommendation pertains to the method of derivative computation employed in the SLSQP approach. Currently, the adjoint method and backward-mode AD are utilised primarily for their advantages when the number of design variables exceeds the number of functions, as is often the case with the SLSQP KS and ALM approaches. However, in the SLSQP approach, the scenario may often be reversed, with more functions than design variables. This is particularly evident when considering that each beam-truss element in the structural model yields four

inequality constraint functions against a single design variable, such as beam-truss thickness. In such cases, the direct method and forward-mode AD are more beneficial, reducing the computational cost and enhancing the efficiency of the SLSQP optimisation process. A shift to these methods would align the derivative computation with the specific requirements of the SLSQP approach, especially in scenarios where functions outnumber design variables.

The second recommendation focuses on improving the handling of bound constraints in the SLSQP method. The current approach trims the search direction to fit within the bounds, sometimes resulting in less progress as the search direction may become significantly shortened. A more practical alternative could be the adoption of the projection approach, similar to that used in the ALM sub-optimisation method. This approach could potentially enhance the progress made by the optimiser, avoiding the limitations of the current trimming strategy and thereby improving the overall efficiency and effectiveness of the SLSQP method in dealing with bound constraints.

6.2.3 Recommendations for SLSQP KS approach

In enhancing the SLSQP KS approach within QASTRO, our study suggests several targeted recommendations to improve its application and effectiveness. Firstly, the recommendation regarding handling bound constraints in the SLSQP approach is equally applicable to the SLSQP KS approach since the SLSQP KS utilises the same underlying SLSQP method.

A key recommendation for the SLSQP KS approach involves the strategic management of the ρ_{KS} value. We observed that an excessively low ρ_{KS} value hinders the optimiser's ability to converge to a satisfactory solution, while a very high value complicates gradient computations. A sequential increase in the ρ_{KS} value could offer a balanced approach, starting with a lower value to facilitate easier derivative computations and then gradually increasing it as the optimiser nears the solution. This method would enable the optimiser to initially navigate the solution space more efficiently and then refine its approach as it gets closer to the optimal solution. This approach was already suggested by Poon and Martins (2007).

Additionally, a hybrid approach is proposed, combining the strengths of the SLSQP KS approach with either the SLSQP or the ALM approach. This hybrid method would initially employ the SLSQP KS approach to approximate a solution close to the global minimum, leveraging its effective convergence behaviour when aggregating inequality constraints. Following this initial approximation, the optimisation could switch to either the ALM or the SLSQP approach with tight bounds to refine and fully converge to the global minimum. It is crucial to balance the tightness of these bounds to avoid hindering the optimiser's progress. If the optimisation ends at the bounds, a slight relaxation of the bounds followed by re-optimisation might be necessary to achieve the best possible solution.

6.2.4 Recommendations for the ALM approach

In optimising the ALM approach within QASTRO, several key recommendations can be made to enhance its efficacy and efficiency based on our research findings.

Firstly, the reader wants to emphasise again that currently, the ALM approach does not use its full potential when multiple flight conditions are considered. The author recommends revisiting the programming structure regarding implementing multiple conditions with the ALM approach such that we need to solve just one adjoint equation, regardless of the number of flight conditions we consider. For more details, the reader is referred to section 3.7.

Secondly, exploring different methods for solving the sub-optimisation problems within the ALM framework is advisable. Our exploration indicated that alternative methods, such as the

SLSQP method, could offer better and faster convergence behaviour for these sub-problems. The current method, L-BFGS-B, requires derivative evaluations at each function evaluation to verify the Wolfe conditions (eqs. 3.61 and 3.62). A more efficient strategy could be to initially check only the Armijo condition (eq. 3.61) and require gradient evaluations only if necessary. The SLSQP method, for example, employs an approach to adjust the step size by solving a one-dimensional optimisation problem (eq. 3.49), which seems more effective due to fewer derivative evaluations. However, a comprehensive analysis of various methods for solving the ALM's sub-problems is needed to validate this hypothesis. The initial choice of the L-BFGS-B method was driven by its practical projection approach for handling bounds, but alternative methods might offer better performance.

Another area for improvement in the ALM approach is the strategy for solving adjoint variables. Currently, the approach for solving adjoint variables in the ALM is similar to that in the SLSQP and SLSQP KS approaches. The only difference is that the SLSQP and SLSQP KS approaches use the previous adjoint variables as initial guesses. Using the previous adjoint variables as an initial guess is not optimal for the ALM approach, especially when the penalty factor is increased. An increase in the penalty factor significantly alters the derivatives, making the previous adjoint variables less suitable as initial guesses. Therefore, it is suggested that further research be conducted to develop a more effective initial guess strategy for adjoint variables in the ALM approach. This could involve using the previous adjoint variables as initial guesses only when the penalty factor remains unchanged and finding alternative strategies when the penalty factor is adjusted.

6.2.5 Recommendation for new optimisation approach

The main advantage of the ALM approach is that we do not explicitly need to compute the derivative of the objective function and all constraint functions. However, we only need to compute the derivative of the augmented Lagrangian function $\mathcal{A}$. We could apply the same idea to the SQP method. The SQP method solves multiple sub-problems of the form

$$\nabla^2\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)\begin{Bmatrix}\boldsymbol{q}_k\\ \boldsymbol{\gamma}_k\end{Bmatrix} = -\nabla\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k) \tag{6.1}$$

$$\begin{bmatrix}\nabla^2_{\boldsymbol{xx}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k) & \nabla^2_{\boldsymbol{x\lambda}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)\\ \nabla^2_{\boldsymbol{\lambda x}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\lambda}) & \nabla^2_{\boldsymbol{\lambda\lambda}}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)\end{bmatrix}\begin{Bmatrix}\boldsymbol{q}_k\\ \boldsymbol{\gamma}_k\end{Bmatrix} = -\begin{Bmatrix}\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)\\ \nabla_{\boldsymbol{\lambda}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)\end{Bmatrix} \tag{6.2}$$

$$\begin{bmatrix}\nabla^2_{\boldsymbol{xx}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k) & \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}_k)\\ \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}_k)^T & \boldsymbol{0}\end{bmatrix}\begin{Bmatrix}\boldsymbol{q}_k\\ \boldsymbol{\gamma}_k\end{Bmatrix} = -\begin{Bmatrix}\nabla_{\boldsymbol{x}}f(\boldsymbol{x}_k)+\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}_k)\boldsymbol{\lambda}\\ \boldsymbol{h}(\boldsymbol{x}_k)\end{Bmatrix}, \tag{6.3}$$

see section A.1.4.6 in the Theoretical Background chapter in the Appendix. $\boldsymbol{\gamma}_k$ determines the Lagrange multiplier update at iteration k. We can get $\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)$ by the adjoint method and backward-mode AD. $\boldsymbol{h}(\boldsymbol{x}_k)$ is given at every iteration. Thus, we know $\nabla\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)$ by only solving a single residual function to get the adjoint variables. Next, we need to know the Hessian matrix $\nabla^2\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)$ which we can approximate with the aid of $\nabla\mathcal{L}(\boldsymbol{x}_k,\boldsymbol{\lambda}_k)$ by the BFGS - formula proposed by Powell (1978b). Alternatively, one could even approximate the inverse of the Hessian. In the final step, we must solve the linear system for the search direction $\boldsymbol{q}_k$ and the multiplier update $\boldsymbol{\gamma}_k$. This process can be repeated until convergence. One can see that we just need to solve one residual function for each sub-problem to get the adjoint variables and thus the gradients for the SQP method. In contrast, the SLSPQ method requires solving one residual function for the objective function and one for each constraint function to determine the adjoint variables.

Bibliography

AKGÜN, M. A.; HAFTKA, R. T.; WU, K. C.; WALSH, J. L.; GARCELON, J. H. Efficient structural optimization for multiple load cases using adjoint sensitivities. **AIAA Journal**, v. 39, n. 3, p. 620–626, 2001.

ALMEIDA, L. d. S. **Propeller slipstream model coupled with an aerodynamic optimization program**. Dissertation (Mestrado) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 2021.

ANDERSON, J. D. **Fundamentals of Aerodynamics**. Mcgraw-hill series in aeronautical and aerospace engineering. New York: McGraw-Hill Education, 2017. (13). ISBN 978-1-259-12991-9.

ARMAND, P.; OMHENI, R. A globally and quadratically convergent primal-dual augmented lagrangian algorithm for equality constrained optimization. **Optimization Methods and Software**, v. 32, p. 1–21, 2017.

ARMAND, P.; OMHENI, R. A mixed logarithmic barrier-augmented lagrangian method for nonlinear optimization. **Journal of Optimization Theory and Applications**, v. 173, p. 523–547, 2017.

ARORA, J. S.; CHAHANDE, A. I.; PAENG, J. K. Multiplier methods for engineering optimization. **International Journal for Numerical Methods in Engineering**, John Wiley & Sons Ltd., v. 32, p. 1485–1525, 1991.

ARRECKX, S.; LAMBE, A.; MARTINS, J. R. R. A.; ORBAN, D. A matrix-free augmented lagrangian algorithm with application to large-scale structural design optimization. **Optimization and Engineering**, v. 17, p. 359–384, 2016.

BEER, G.; MARUSSIG, B.; DUENSER, C. Basis functions, B-splines. *In*: THE ISOGEOMETRIC BOUNDARY ELEMENT METHOD. **Springer International Publishing**. Cham, 2020. p. 35–71. ISBN 978-3-030-23339-6. Available at: https://doi.org/10.1007/978-3-030-23339-6_3.

BERTSEKAS, D. P. **Constrained Optimization and Lagrange Multiplier Methods**. New York: Academic Press, 1982.

BERTSEKAS, D. P. **Constrained Optimization and Lagrange Multiplier Methods**. 1st. ed. [*S.l.*]: Athena Scientific, 1996.

BERTSEKAS, D. P. **Nonlinear Programming**. 2nd. ed. [*S.l.*]: Athena Scientific, 1999.

BIRGIN, E.; CASTILLO, R.; MARTÍNEZ, J. Numerical comparison of augmented lagrangian algorithms for nonconvex problems. **Computational Optimization and Applications**, Springer Science + Business Media, v. 31, n. 1, p. 31–55, 2005. Available at: https://doi.org/10.1007/s10589-005-1066-7.

BIRGIN, E. G.; MARTÍNEZ, J. M. Complexity and performance of an augmented lagrangian algorithm. **Universidade de São Paulo**, July 2019.

BIROS, G.; GHATTAS, O. Parallel lagrange–newton–krylov–schur methods for pde-constrained optimization. part i: the krylov–schur solver. **SIAM Journal on Scientific Computing**, v. 27, n. 2, p. 687–713, 2005.

BROYDEN, C. G. A class of methods for solving nonlinear simultaneous equations. American Mathematical Society, v. 19, n. 92, p. 577–593, 1965. Available at: https://www.jstor.org/stable/2003941.

BRYSON, A.; HO, Y.-C. **Applied Optimal Control: Optimization, Estimation, and Control**. New York: Wiley, 1975.

BUYS, J. D. **Dual algorithms for constrained optimization problems**. Thesis (Doutorado) — University of Leiden, Netherlands, 1972.

BYRD, R. H.; LU, P.; NOCEDAL, J.; ZHU, C. A limited memory algorithm for bound constrained optimization. **Society for Industrial and Applied Mathematics**, v. 16, n. 5, p. 1190–1208, September 1995.

BYRD, R. H.; NOCEDAL, J.; SCHNABEL, R. B. Representations of quasi-newton matrices and their use in limited memory methods. **Mathematical Programming**, Springer-Verlag, v. 63, p. 129—156, 1994.

CONN, A. R.; HARING, R. A.; VISWESWARIAH, C.; WU, C. W. Circuit optimization via adjoint lagrangians. **IEEE**, iBM T. J. Watson Research Center, Yorktown Heights, NY 10598, 1997. ISSN 1092-3152.

COOK, R. D.; MALKUS, D. S.; PLESHA, M. E.; WITT, R. J. **Concepts and applications of Finite Element Analysis**. 4th. ed. New York: John Wiley & Sons. Inc., 2001. ISBN 978-0-471-35605-9.

COUCEIRO, I.; PARíS, J.; RAMíREZ, L.; NAVARRINA, F. A new formulation for transient response optimization of structures based on constraint aggregation functions. **Engineering Optimization**, Taylor & Francis, v. 54, n. 9, p. 1600–1616, 2022.

CRUZ, A. C. B. da. **Box wing optimization with a 3D lifting line theory**. Dissertation (Mestrado) — Instituto Tecnológico de Aeronáutica, São José dos Campos, 2020.

CURRY, M. Pathfinder photo gallery contact sheet. Feb 2002. Available at: https://www.dfrc.nasa.gov/Gallery/Photo/Pathfinder/HTML/index.html.

DODT, T. **Introducing the 787 (Presentation)**. Lisbon, 2011. Available at: https://www.isasi.org/Documents/library/technical-papers/2011/Introducing-787.pdf. Accessed on: 04.01.2024.

DRELA, M. Xfoil: An analysis and design system for low reynolds number airfoils. Springer Berlin Heidelberg, Berlin, Heidelberg, p. 1–12, 1989.

DRELA, M. **Flight Vehicle Aerodynamics**. Cambridge, Massachusetts: MIT Press, 2014. ISBN 978-0-262-52644-9.

European Commission. **Reducing emissions from aviation**. 2023. `https://climate.ec.europa.eu/eu-action/transport/reducing-emissions-aviation_en`. Accessed on: 08.12.23.

FLETCHER, R. Methods related to lagrangian functions. *In*: NUMERICAL METHODS FOR CONSTRAINED OPTIMIZATION. **Academic Press**. London, 1974. p. 219–239.

FLETCHER, R. An ideal penalty function for constrained optimization. **J. Inst. Maths applies**, v. 15, p. 319–342, 1975.

FLITTIE, K.; CURTIN, B. Pathfinder solar-powered aircraft flight performance. 1998.

GHAZLANE, I.; CARRIER, G.; DUMONT, A.; MARCELET, M.; DéSIDéRI, J.-A. Aerostructural optimization with the adjoint method. *In*: **EUROGEN 2011**. **Proceedings** [...]. Capua, Italy: [*s.n.*], 2011. Hal-00645837.

GILL, P. E.; MURRAY, W.; SAUNDERS, M. A.; WRIGHT, M. H. **Some Theoretical Properties of an Augmented Lagrangian Merit Function**. [*S.l.*], Sep 1986. Technical Report.

GIUNTA, A. A.; DUDLEY, J. M.; NARDUCCI, R.; GROSSMAN, B.; TTAFTKA, R. T.; MASON, W. H.; WATSON, L. T. Noisy aerodynamic response and smooth approximations in hsct design. *In*: AIAA. **Proc. 5th AIAA Multidisciplinary Analysis and Optimization Syrup. Proceedings** [...]. Panama City, FL, 1994. p. 1117–1128. AIAA Paper No. 94-4376.

GRIVA, I.; NASH, S. G.; SOFER, A. **Linear and nonlinear optimization**. 2nd. ed. Fairfax, Virginia: Society for Industrial and Applied Mathematics, 2009. ISBN 978-0-898716-61-0.

GROSSMAN, B.; GURDAL, Z.; STRAUCH, G.; EPPARD, W.; HAFTKA, R. Integrated aerodynamic/structural design of a sailplane wing. **Journal of Aircraft**, v. 25, n. 9, p. 855–860, 1988.

HAARHOFF, P. C.; BUYS, J. D. A new method for the optimization of nonlinear function subject to nonlinear constraints. **The Computer Journal**, v. 13, p. 178–184, 1970.

HAFTKA, R. T.; GüRDAL, Z. Constrained optimization. *In*: ELEMENTS OF STRUCTURAL OPTIMIZATION. **Springer Dordrecht**. 3. ed. Reston, 1992. p. 159–207. (Solid Mechanics and Its Applications, v. 11). ISBN 978-0-7923-1505-6.

HAN, S. A globally convergent method for nonlinear programming. **Journal of Optimization Theory and Applications**, Springer-Verlag, v. 3, p. 297–309, 1977.

HASCOËT, L.; PASCUAL, V. The Tapenade Automatic Differentiation tool: Principles, Model, and Specification. **ACM Transactions On Mathematical Software**, v. 39, n. 3, 2013. Available at: http://dx.doi.org/10.1145/2450153.2450158http://dx.doi.org/10.1145/2450153.2450158.

HESTENES, M. R. Multiplier and gradient methods. **Journal of Optimization Theory and Applications**, v. 4, p. 303–320, 1969.

HICKS, R.; HENNE, P. Wing design by numerical optimization. *In*: **AIAA Paper**. **Proceedings** [...]. [*S.l.: s.n.*], 1977. Paper 77-1247.

HICKS, R.; MURMAN, E.; VANDERPLAATS, G. **An Assessment of Airfoil Design by Numerical Optimization**. [*S.l.*], July 1974.

JAMESON, A. Aerodynamic design via control theory. **Journal of Scientific Computing**, Plenum Publishing Corporation, v. 3, n. 3, p. 233–260, 1988.

KATZ, J.; PLOTKIN, A. **Low-Speed Aerodynamics**. 2nd. ed. [*S.l.*]: Cambridge University Press, 2001. (13). ISBN 978-0-521-66552-0.

KENNEDY, G. J.; HICKEN, J. E. Improved constraint-aggregation methods. **Computational Methods in Applied Mechanics and Engineering**, Elsevier, v. 289, p. 332–354, 2015.

KRAFT, D. A software package for sequential quadratic programming. **Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt**, July 1988.

LAMBE, A. B.; KENNEDY, G. J.; MARTINS, J. R. R. A. An evaluation of constraint aggregation strategies for wing box mass minimization. **Structural and Multidisciplinary Optimization**, v. 55, p. 257–277, 2017.

LARSSON, T.; RÖNNQVIST, M. Simultaneous structural analysis and design based on augmented lagrangian duality. **Structural Optimization**, Springer-Verlag, v. 9, p. 1–11, 1995.

LAWSON, C. L.; HANSON, R. J. *In*: SOLVING LEAST SQUARES PROBLEMS. **Society for Industrial and Applied Mathematics**. [*S.l.: s.n.*], 1995. (Classics in Applied Mathematics, v. 15). ISBN 978-0-89871-356-5.

LOGAN, D. L. **A First Course in the Finite Element Method**. 5th. ed. 200 First Stamford Place, Suite 400 Stamford, CT 06902: CENGAGE Learning, 2012. ISBN 978-0-495-66825-1.

LOH, C. **How Much Can The Boeing 787's Wings Flex?** 2022. `https://simpleflying.com/boeing-787-wings-flex/`. Accessed: 14.12.23.

LUENBERGER, D. G.; YE, Y. **Linear and Nonlinear Programming**. 3rd. ed. Stanford, CA, USA: Springer, 2008. ISBN 978-0-387-74502-2.

LUND, E. **Finite Element Based Design Sensitivity Analysis and Optimization**. Thesis (Ph.D. Dissertation) — Aalborg University, Aalborg, Denmark, 1994.

MADER, C. A.; KENWAY, G. K.; MARTINS, J. R. A framework for high-fidelity aerostructural optimization of aircraft configurations. *In*: **2008 Asia Simulation Conference - 7th International Conference on System Simulation and Scientific Computing**. **Proceedings** [...]. [*S.l.: s.n.*], 2008. p. 488–493.

MARTINS, J.; ALONSO, J.; REUTHER, J. High-fidelity aero-structural design optimization of a supersonic business jet. *In*: **AIAA Paper Series**. **Proceedings** [...]. [*S.l.: s.n.*], 2004. Paper 2002-1483.

MARTINS, J. R.; ALONSO, J. J.; REUTHER, J. J. A coupled-adjoint sensitivity analysis method for high-fidelity aero-structural design. Toronto, ON, Canada; Stanford, CA, USA; Moffett Field, CA, USA, received August 19, 2002; Revised September 24, 2003, 2005.

MARTINS, J. R. R. A.; KENNEDY, G. J. Enabling large-scale multidisciplinary design optimization through adjoint sensitivity analysis. **Structural and Multidisciplinary Optimization**, Springer, v. 64, p. 2959–2974, 2021.

MARTINS, J. R. R. A.; NING, A. **Engineering Design Optimization**. 1st. ed. New York: Cambridge University Press, 2021. ISBN 9781108833417.

MATWEB. **ASM Material Data Sheer**. 2023. `https://asm.matweb.com/search/SpecificMaterial.asp?bassnum=ma2024t4`. Accessed: 01.08.23.

MAUTE, K.; ALLEN, M. Conceptual design of aeroelastic structures by topology optimization. **Structural and Multidisciplinary Optimization**, v. 27, p. 27–42, 2004.

MORE, J. J.; GARBOW, B. S.; HILLSTROM, K. E. User guide for minpack-1. [in fortran]. 1980. Available at: https://www.osti.gov/biblio/6997568.

MORSE, W. L. Wing lift distribution. **AIRCRAFT ENGINEERING**, p. 136–138, maio 1944.

NASA. Helios prototype: The forerunner of 21st century solar-powered "atmospheric satellites". **NASA Facts**, August 2002. Available at: https://www.nasa.gov/wp-content/uploads/2021/09/120318main_FS-068-DFRC.pdf. Accessed on: 19 Nov. 2023.

NEWMAN, J. C. I.; TAYLOR, A. C. I.; BARNWELL, R. W.; NEWMAN, P. A.; HOU, G. J.-W. Overview of sensitivity analysis for complex aerodynamic and shape optimization configurations. **Journal of Aircraft**, submitted for publication in the Special MDA & O Issue, June 1998.

NOCEDAL, J.; WRIGHT, S. J. **Numerical Optimization**. 2nd. ed. 233 Spring Street, New York, NY 10013, USA: Springer Science + Business Media, 2000. ISBN 978-0387-30303-1.

PETER, J. E.; DWIGHT, R. P. Numerical sensitivity analysis for aerodynamic optimization: A survey of approaches. **Computers & Fluids**, Elsevier, Chatillon, France; Delft, The Netherlands, v. 39, p. 373–391, 2010. Available at: https://doi.org/10.1016/j.compfluid.2009.09.013.

PHILLIPS, W.; SNYDER, D. O. Modern adaption of prandtl's classic liftign-line theroy. **Journal of Aircraft**, v. 37, n. 4, p. 662–670, 2000.

PIERRE, D. A.; LOWE, M. J. **Mathematical Programming via Augmented Lagrangians: An Introduction with Computer Programs**. Massachusetts: Addison-Wesley, 1975.

PILLO, G. D.; GRIPPO, L. A new augmented lagrangian function for inequality constraints in nonlinear programming problems. **Journal of Optimization Theory and Applications**, Plenum Publishing Corporation, v. 36, n. 4, p. 495–519, 1982. Available at: https://doi.org/10.1007/BF00940544.

POLAK, E.; SANGIOVANNI-VINCENTELLI, A. Theoretical and computational aspects of the optimal design centering, tolerancing and tuning problem. **IEEE Transactions on Circuits and Systems**, v. 26, p. 795–813, 1979.

POLYAK, B.; TRETYAKOV, N. The method of penalty estimates for conditional extremum problems. **USSR Computational Mathematics and Mathematical Physics**, v. 13, p. 42–58, 1974.

POON, N. M. K.; MARTINS, J. R. R. A. An adaptive approach to constraint aggregation using adjoint sensitivity analysis. **Structural and Multidisciplinary Optimization (2007)**, Springer-Verlag, v. 34, p. 61–73, December 2006.

POON, N. M. K.; MARTINS, J. R. R. A. An adaptive approach to constraint aggregation using adjoint sensitivity analysis. **Structural and Multidisciplinary Optimization**, v. 34, p. 61–73, 2007.

POWELL, M. J. D. A method for nonlinear constraints in minimization problems. *In*: FLETCHER, R. (Ed.). **Optimization**. London and New York: Academic Press, 1969. p. 283–298.

POWELL, M. J. D. A hybrid method for nonlinear equations. *In*: RABINOWITZ, P. (Ed.). **Numerical Methods for Nonlinear Algebraic Equations**. [*S.l.*]: Gordon and Breach, 1970.

POWELL, M. J. D. Algorithms for nonlinear constraints that use lagrangian functions. **Mathematical Programming**, v. 14, p. 224–248, 1978.

POWELL, M. J. D. A fast algorithm for nonlinearly constrained optimization calculations. *In*: NUMERICAL ANALYSIS. **Springer Berlin Heidelberg**. Berlin, Heidelberg, 1978. p. 144–157. (Lecture Notes in Mathematics, v. 630). ISBN 978-3-540-35972-2.

PRANDTL, L. Über Tragflügel kleinsten induzierten Widerstandes. **Zeitschrift für Flugtechnik und Motorluftschiffahrt**, v. 24, p. 305–306, 1933.

QIN, J.; NGUYEN, D. T. Generalized exponential penalty function for nonlinear programming. **Journal of Computational and Applied Mathematics**, Elsevier, v. 50, p. 263–273, 1994.

RASPANTI, C.; BANDONI, J.; BIEGLER, L. New strategies for flexibility analysis and design under uncertainty. **Computers and Chemical Engineering**, v. 24, p. 2193–2209, 2000.

RIBAH, A. Z.; RAMAYANTI, S. Power produced analysis of solar arrays in nadir pointing mode for low-earth equatorial micro-satellite conceptual design. **IOP Conference Series: Earth and Environmental Science**, IOP Publishing, v. 284, p. 012048, 2018.

ROCKAFELLAR, R. The multiplier method of hestenes and powell applied to convex programming. **Journal of Optimization Theory and Applications**, Plenum Publishing Corporation, v. 12, n. 6, 1973. Available at: https://doi.org/10.1007/BF00934777.

ROCKAFELLAR, R. T. A dual approach to solving nonlinear programming problems by unconstrained optimization. **Mathematical Programming**, v. 5, p. 354–373, 1973.

ROONEY, W.; BIEGLER, L. **Optimal Process Design with Model Parameter Uncertainty and Process Variability**. Pittsburgh, PA, 2002.

SCHITTKOWSKI, K. The Nonlinear Programming Method of Wilson, Han, and Powell with an Augmented Lagrangian Type Line Search Function, Part 1: Convergence Analysis. **Numerische Mathematik**, v. 38, p. 83–114, 1981.

SCHITTKOWSKI, K. The Nonlinear Programming Method of Wilson, Han, and Powell with an Augmented Lagrangian Type Line Search Function, Part 2: An Efficient Implementation with Linear Least Squares Subproblems. **Numerische Mathematik**, v. 38, p. 115–127, 1981.

SCHRENK, O. Ein einfaches näherungsverfahren zur ermittlung von auftriebsverteilungen langs der tragflügelbreite. **Luftwissen**, v. 7, n. 4, p. 118–120, abr. 1940.

SENHORA, F. V.; GIRALDO-LONDOÑO, O.; MENEZES, I. F. M.; PAULINO, G. H. Topology optimization with local stress constraints: A stress aggregation-free approach. **Structural and Multidisciplinary Optimization**, Springer, v. 62, p. 1639–1668, 2020.

SOBIESZCZANSKI-SOBIESKI, J. Sensitivity of complex internally coupled systems. **AIAA Journal**, v. 28, n. 1, p. 153–160, 1990.

SOBIESZCZANSKI-SOBIESKI, J.; HAFTKA, R. Multidisciplinary aerospace design optimization: survey of recent developments. **Structural Optimization**, Springer-Verlag, v. 14, p. 1–23, 1997.

STETTNER, M.; SCHRAGE, D. An approach to tiltrotor wing aeroservoelastic optimization. *In*: **Proceedings of the 4th AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization**. **Proceedings** [...]. Cleveland, Ohio: [*s.n.*], 1992.

TOROPOV, V.; MARKINE, V. The use of simplified numerical models as mid-range approximations. *In*: **Proc. 6th AIAA/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization**. **Proceedings** [...]. Bellevue, WA: [*s.n.*], 1996. p. 952–958. AIAA Paper No. 96-4088.

VANDERPLAATS, G.; HICKS, R. **Numerical Airfoil Optimization Using Reduced Number of Design Coordinates**. [*S.l.*], July 1976.

VIRTANEN, P.; GOMMERS, R.; OLIPHANT, T. E.; HABERLAND, M.; REDDY, T.; COURNAPEAU, D.; BUROVSKI, E.; PETERSON, P.; WECKESSER, W.; BRIGHT, J.; van der Walt, S. J.; BRETT, M.; WILSON, J.; MILLMAN, K. J.; MAYOROV, N.; NELSON, A. R. J.; JONES, E.; KERN, R.; LARSON, E.; CAREY, C. J.; POLAT, İ.; FENG, Y.; MOORE, E. W.; VanderPlas, J.; LAXALDE, D.; PERKTOLD, J.; CIMRMAN, R.; HENRIKSEN, I.; QUINTERO, E. A.; HARRIS, C. R.; ARCHIBALD, A. M.; RIBEIRO, A. H.; PEDREGOSA, F.; van Mulbregt, P.; SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. **Nature Methods**, v. 17, p. 261–272, 2020.

WILSON, R. B. **A simplicial algorithm for concave programming. Doctoral dissertation.** 1963. 193 p. Thesis (Doctor of Business Administration) — Harvard Business School, Harvard, 1963.

WITTEL, H.; JANNASCH, D.; VOßIEK, J.; SPURA, C. **Roloff/Matek Maschinenelemente: Normung, Berechnung, Gestaltung**. 24. ed. Wiesbaden: Springer Vieweg, 2019. ISBN 978-3-658-26280-8.

WUNDERLICH, T. F.; DÄHNE, S.; REIMER, L.; SCHUSTER, A. Global aero-structural design optimization of composite wings with active manoeuvre load alleviation. **CEAS Aeronautical Journal**, v. 13, p. 639–662, 2022.

XIA, Q.; SHI, T. Stiffness optimization of geometrically nonlinear structures and the level set based solution. **International Journal of Simulation Multidiscience and Design Optimization**, EDP Sciences, v. 7, 2016.

ZHANG, K.; HAN, Z.; GAO, Z. *et al.* Constraint aggregation for large number of constraints in wing surrogate-based optimization. **Structural and Multidisciplinary Optimization**, v. 59, p. 421–438, 2019.

ZHU, C.; BYRD, R. H.; LU, P.; NOCEDAL, J. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound- constrained optimization. **ACM Transactions on Mathematical Software**, v. 23, n. 4, p. 550–560, December 1997.

A Theoretical Background

The Theoretical Background chapter plays a crucial role in any academic work, particularly in the field of optimisation. Different definitions of theories that are equally valid exist in the literature, but it is essential to be consistent with those theories. This chapter provides the reader with the definition of all theories used in this Master's Thesis. Furthermore, this chapter establishes a framework for understanding the research question and the significance of the study. Ultimately, the Theoretical Background chapter is essential for ensuring the validity and rigour of the research and for demonstrating the author's expertise and contribution to the field of (aero-) structural optimisation. The Theoretical Background chapter is divided into two parts. The first part gives the reader background information regarding optimisation; the second part is about the fundamental parts of aerodynamics.

A.1 Optimisation

A.1.1 Notation

This subsection shows the notation used in this Master's Thesis. All vectors or matrices are written in bold, e.g. $\boldsymbol{x}$. All other variables are scalar values or scalar functions. A vector of scalar functions $\boldsymbol{f}(\boldsymbol{x})$ $\mathbb{R}^n \rightarrow \mathbb{R}^m$ is defined to be

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{Bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \\ f_3(\boldsymbol{x}) \\ \vdots \\ f_n(\boldsymbol{x}) \end{Bmatrix}, \tag{A.1}$$

where the domain is in $\mathbb{R}^n$ and the range is in $\mathbb{R}^m$. Vectors are always column vectors if not mentioned otherwise. The gradient of a scalar function $f(\boldsymbol{x})$ $\mathbb{R}^n \rightarrow \mathbb{R}$ is a column vector of the partial derivatives of the function with respect to all variables $x_1, x_2 ... x_n$.

$$\nabla f(\boldsymbol{x}) = \begin{Bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{Bmatrix} \tag{A.2}$$

The gradient of a vector of scalar functions $\boldsymbol{f}(\boldsymbol{x})$ $\mathbb{R}^n \rightarrow \mathbb{R}^m$ is a matrix where each column is the gradient vector of one function.

$$\nabla \boldsymbol{f}(\boldsymbol{x}) = \frac{\partial \boldsymbol{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_1}{\partial x_n} & \frac{\partial f_2}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \tag{A.3}$$

The transpose of $\nabla \boldsymbol{f}(\boldsymbol{x})$ is called the Jacobian matrix and is defined as

$$\boldsymbol{J_f} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}. \tag{A.4}$$

The second derivative of a scalar function $f(\boldsymbol{x})$ $\mathbb{R}^n \to \mathbb{R}$ is a matrix, too. This matrix is referred to as the Hessian matrix and is defined as

$$\boldsymbol{H}_f = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}. \tag{A.5}$$

The norm used in this Master's Thesis is the Euclidean norm, which is defined as

$$\|\boldsymbol{h}\|_2^2 = \boldsymbol{h}^T \boldsymbol{h}. \tag{A.6}$$

A.1.2 Definition of optimisation problem

Formally, an optimisation problem can be formulated as

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{subjected to (s.t.)} \quad & \boldsymbol{h}(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{0} \\ & \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{p}) \leq \boldsymbol{0} \\ & \boldsymbol{l} \leq \boldsymbol{x} \leq \boldsymbol{u} \\ \text{while solving (w.s.)} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0} \\ \text{for} \quad & \boldsymbol{p}. \end{aligned} \tag{A.7}$$

where f is called the objective function. The objective function could, for example, describe the fuel consumption, weight of a structure or the drag of a wing and depends on the design variables $\boldsymbol{x}$ and the state variables $\boldsymbol{p}$. The design variables $\boldsymbol{x}$ are, for example, the chord length of a wing or the fuselage diameter. The state variables $\boldsymbol{p}$ are, for example, the wing displacements. We want to optimise f with respect to the design variables $\boldsymbol{x}$, or in other words, we want to find the lowest value of the objective function. Underneath the word 'min' is indicated with respect to which variable the objective function becomes minimised. Suppose we assume that f is bounded, which means that the range of f is finite and that there are no constraints, then the lowest value of the objective function is at a minimum. It is more complicated than it seems to find this minimum because the objective function f is usually not an explicit analytical expression. Hence, it is difficult to symbolically determine the derivative of f and set the derivative to zero to find the extreme points of the objective function. Additionally, f is generally computationally expansive to evaluate at a certain design state. Thus, evaluating f at multiple points in the design range is also inefficient in finding its lowest value. Gradient-based optimisation is an

iterative process determining the function values and their gradient at multiple design points. The optimisation procedure converges to the minimum of the objective function. It should be noted that an optimisation algorithm could not converge to the global minimum but to a local minimum of f. In most cases, we want to optimise under certain conditions, which are called constraints in optimisation terms. In this case, we minimise f with respect to $\boldsymbol{x}$ subjected to the constraint functions. A condition or constraint could, for example, be the number of seats in an aeroplane, the speed of an aeroplane or the flight altitude. There are three types of constraints: equality, inequality and bound constraints. All equality constraints are combined in one vector-valued function called $\boldsymbol{h}$, and all inequality constraints are combined in one vector-valued function called $\boldsymbol{g}$. Both constraints are formulated such that they equal zero (equality constraint) or that they are greater than zero (inequality constraint). For example, suppose we want to minimise the fuel consumption of an aeroplane. In that case, we can prescribe the flight altitude by $h = x_1 - x_{1_{desirable}} = 0$, where x_1 is the design variable for the flight altitude and $x_{1_{desirable}}$ is the desirable flight altitude. The bound constraint defines that there is an upper and a lower limit for all design variables.

Furthermore, we can require that while solving the problem, an equation is satisfied. In the optimisation problem (A.7) we solve the equation $\boldsymbol{r}(\boldsymbol{p};\boldsymbol{x}) = \boldsymbol{0}$ to get the state variables $\boldsymbol{p}$. The semicolon indicates that the design variables $\boldsymbol{x}$ are fixed when the functions $\boldsymbol{r}$ are solved for the state variables $\boldsymbol{p}$. Usually, those functions $\boldsymbol{r}$ are called residuals. In this Master's Thesis, the residuals ensure that the system follows the physical laws (MARTINS; NING, 2021).

A.1.3 Optimality conditions

It is essential to define what we consider as the solution to an optimisation problem since we are trying to find it. We check for optimality conditions at every design stage to verify if we found the solution to the optimisation problem. Although many algorithms only check the first-order conditions. Furthermore, it is only possible to reach the optimality conditions within a specific computer precision ϵ in numerical optimisation. However, in many engineering applications, reaching the optimality conditions within a range higher than the computer's precision is sufficient. So ϵ is usually set to a higher value than the computer precision. It is important to note that the optimality condition can only be checked if we find a local solution but not if we find a global solution. Furthermore, we assume that all the functions are twice continuously differentiable.

A.1.3.1 Unconstrained

Let us consider an optimisation problem without any constraints.

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}) \tag{A.8}$$

The solution to an optimisation problem is called $\boldsymbol{x}^*$. For an unconstrained optimisation problem, the solution $\boldsymbol{x}^*$ is the (global) minimum of the objective function $f(\boldsymbol{x})$. It is well known that at $\boldsymbol{x}^*$, the gradient of the objective function is zero, $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$, which is a first-order necessary condition. First-order refers to the first derivative of the condition. However, the first-order necessary condition is also satisfied at a maximum of $f(\boldsymbol{x})$. Hence, we need an additional necessary condition to check for optimality. One can derive from the Taylor series that $\nabla^2 f(\boldsymbol{x}^*)$ is positive semidefinite, the second necessary optimality condition. However, both necessary conditions are still insufficient because a saddle point satisfies both conditions but is not a minimiser of $f(\boldsymbol{x})$. A sufficient condition is that if $\nabla^2 f(\boldsymbol{x})$ is positive definite, then $\boldsymbol{x} = \boldsymbol{x}^*$ (GRIVA *et al.*, 2009). The box below summarises the necessary and sufficient optimality

conditions of an unconstrained optimisation problem.

Optimality conditions unconstrained optimisation

Necessary conditions

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0} \tag{A.9}$$

$$\nabla^2 f(\boldsymbol{x}^*) \text{ is positive semidefinite} \tag{A.10}$$

Sufficient conditions

$$\nabla f(\boldsymbol{x}^*) = \mathbf{0} \tag{A.11}$$

$$\nabla^2 f(\boldsymbol{x}^*) \text{ is positive definite} \tag{A.12}$$

A.1.3.2 Equality constraints

Next, let us consider an optimisation problem with equality constraints.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0} \end{aligned} \tag{A.13}$$

It becomes more difficult when equality constraints are also considered. In constraint optimisation, there is a different first-order necessary condition. Let us assume that we are considering a regular point. The regularity assumption assumes that the rows of $\nabla \boldsymbol{h}(\boldsymbol{x}^*)$ are linearly independent. Without diving too deep inside the theory, Lagrange showed that a first-order necessary condition for constrained optimisation is that the gradient of the objective function $f(\boldsymbol{x}^*)$ needs to be a multiple of the gradient of the constraint function $\boldsymbol{h}(\boldsymbol{x}^*)$. Mathematically, the condition reads

$$\nabla f(\boldsymbol{x}^*) = -\boldsymbol{\lambda}^T \nabla \boldsymbol{h}(\boldsymbol{x}^*), \tag{A.14}$$

where $\boldsymbol{\lambda}$ are the Lagrange multipliers. Note that the minus sign is arbitrary. A second first-order necessary condition is that the constraints are not violated, hence $\boldsymbol{h}(\boldsymbol{x}^*) = \mathbf{0}$. Both of those first-order necessary conditions can be expressed by a so-called Lagrangian function, which is a function of the design variables $\boldsymbol{x}$ and the Lagrange multipliers $\boldsymbol{\lambda}$ and is defined as

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}). \tag{A.15}$$

If the gradient of the Lagrangian function equals zero, both first-order optimality conditions stated above are satisfied.

$$\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \begin{Bmatrix} \nabla_{\boldsymbol{x}} \mathcal{L} \\ \nabla_{\boldsymbol{\lambda}} \mathcal{L} \end{Bmatrix} = \begin{Bmatrix} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^*) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}^*)\, \boldsymbol{\lambda} \\ \boldsymbol{h}(\boldsymbol{x}^*) \end{Bmatrix} = \begin{Bmatrix} \mathbf{0} \\ \mathbf{0} \end{Bmatrix} \tag{A.16}$$

Hence, to check if we found the optimum of our constrained optimisation problem, we do not check if $\nabla f(\boldsymbol{x}) = \mathbf{0}$, but we check if $\nabla \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \mathbf{0}$. $\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}$ is our first necessary condition for optimality. One could think that the second necessary condition is $\nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)$ is positive semidefinite, but this is not true. It is out of the scope of this Master's Thesis to derive the second-order necessary condition because it requires a deep understanding of the optimisation theory. Nevertheless, the second-order necessary condition is that $\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\, \boldsymbol{Z}(\boldsymbol{x}^*)$ is positive semidefinite, where $\boldsymbol{Z}$ is the null-space matrix of the Jacobian of the equality constraints at $\boldsymbol{x}^*$, i.e. $\boldsymbol{J}_{\boldsymbol{h}(\boldsymbol{x}^*)}$. The condition is derived by defining feasible curves and a tangent vector at $\boldsymbol{x}^*$. The full derivation can be found in (GRIVA *et al.*, 2009). However, the necessary conditions are not sufficient. The sufficient conditions are that $\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0}$ and that $\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\, \boldsymbol{Z}(\boldsymbol{x}^*)$

is positive definite (GRIVA *et al.*, 2009). The necessary and sufficient conditions are summarised in the box below.

Optimality conditions with equality constraints

Necessary conditions

$$\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0} \tag{A.17}$$

$$\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{Z}(\boldsymbol{x}^*) \text{ is positive semidefinite} \tag{A.18}$$

Sufficient conditions

$$\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \mathbf{0} \tag{A.19}$$

$$\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) \boldsymbol{Z}(\boldsymbol{x}^*) \text{ is positive definite} \tag{A.20}$$

$\boldsymbol{Z}$ is the null-space matrix of the Jacobian of the equality constraints at $\boldsymbol{x}^*$, i.e. $\boldsymbol{J}_{\boldsymbol{h}(\boldsymbol{x}^*)}$.

A.1.3.3 Inequality constraints

Moving forward, let us consider an optimisation problem with only inequality constraints.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \end{aligned} \tag{A.21}$$

The optimality conditions for inequalities differ slightly from those with equality conditions, but extra attention must be taken. We call the Lagrange multipliers of the inequality constraints $\boldsymbol{\mu}$ to distinguish them from the Lagrange multipliers of the equality constraints. The key point to understanding inequality optimality conditions is that we only need to consider *active* constraints. An constraint is active if $g_i(\bar{\boldsymbol{x}}) = 0$, where $\bar{\boldsymbol{x}}$ is any feasible point of the optimisation problem. All constraints which are at $\boldsymbol{x}^*$ not active do not influence the solution of the optimisation problem and can be neglected. The active constraints at $\boldsymbol{x}^*$ can be treated as equality constraints because at $\boldsymbol{x}^*$ holds that $g_i(\boldsymbol{x}^*) = 0$. Thus, the necessary conditions for an optimisation problem with inequality constraints are the same as for equality constraints, but we need to ensure two things:

1. The Lagrange multiplier of g_i needs to be positive. The objective function decreases towards the feasible region if the Lagrange multiplier is negative.

$$\mu_i^* \leq 0 \tag{A.22}$$

2. We only consider active inequality constraints. We can do this by setting λ_i to zero for inactive constraints and require

$$\boldsymbol{\mu}^{*T} \boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0}, \tag{A.23}$$

which means that a constraint is inactive ($\mu_i = 0$) and/ or the inequality constraint is equal to zero ($\boldsymbol{g}(\boldsymbol{x}^*) = 0$). This condition is called *complementary slackness condition*.

The two additional conditions, together with the necessary conditions defined for the equality constraints, are the necessary optimality conditions for an optimisation problem with only inequality constraints, which are also called the Karush-Kuhn-Tucker (KKT) conditions. Please note that we again assume that $\boldsymbol{x}^*$ is a regular point, which means for inequality constraints that the gradients of the active constraints at $\boldsymbol{x}^*$, $\{\nabla g_i(\boldsymbol{x}^*) : g_i(\boldsymbol{x}^*) = 0\}$, are linearly independent.

For the sufficient conditions of an optimisation problem with inequality constraints, extra care must be taken with the condition $\boldsymbol{\mu}^{*T}\boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0}$. It can happen that $\lambda_i = 0$ and $g_i(\boldsymbol{x}) = 0$, but $\boldsymbol{x} \neq \boldsymbol{x}^*$. Hence, we need to require that not both μ_i and $g_i(\boldsymbol{x})$ equal zero. This condition is called *strict complementary condition*. Adding the strict complementary condition to the necessary optimality and requiring that $\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\, \boldsymbol{Z}(\boldsymbol{x}^*)$ is positive definite, we get the sufficient optimality conditions. However, a better way to define the sufficient optimality conditions is to require that $\boldsymbol{Z}_+(\boldsymbol{x}^*)^T \nabla^2\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*)\, \boldsymbol{Z}_+(\boldsymbol{x}^*)$ is positive definite, instead requiring the strict complementary condition. $\boldsymbol{Z}_+$ is the basis for the null-space of $\boldsymbol{J}_{\boldsymbol{g}(\boldsymbol{x}^*)}$ with only active constraints. Again, the derivation of the second-order conditions is out of the scope of this Master's Thesis. However, one can prove the second-order conditions by defining feasible arcs. The full derivation is given by Griva *et al.* (2009).

Optimality conditions with inequality constraints

Necessary conditions (KKT)

$$\nabla\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\mu}^*) = \mathbf{0} \tag{A.24}$$

$$\boldsymbol{\mu}^* \leq \mathbf{0} \tag{A.25}$$

$$\boldsymbol{\mu}^{*T}\boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0} \tag{A.26}$$

$$\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\mu}^*)\, \boldsymbol{Z}(\boldsymbol{x}^*) \text{ is positive semidefinite} \tag{A.27}$$

Sufficient conditions

$$\nabla\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\mu}^*) = \mathbf{0} \tag{A.28}$$

$$\boldsymbol{\mu}^* \leq \mathbf{0} \tag{A.29}$$

$$\boldsymbol{\mu}^{*T}\boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0} \tag{A.30}$$

$$\boldsymbol{Z}_+(\boldsymbol{x}^*)^T \nabla^2\mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\mu}^*)\, \boldsymbol{Z}_+(\boldsymbol{x}^*) \text{ is positive definite} \tag{A.31}$$

$\boldsymbol{Z}$ is the null-space matrix of the Jacobian of the inequality constraints at $\boldsymbol{x}^*$, i.e. $\boldsymbol{J}_{\boldsymbol{g}(\boldsymbol{x}^*)}$. $\boldsymbol{Z}_+$ is the basis for the null-space of $\boldsymbol{J}_{\boldsymbol{g}(\boldsymbol{x}^*)}$ with only active constraints.

A.1.3.4 Equality and inequality constraints

Ultimately, let us define the optimality conditions for the optimisation problem with equality and inequality constraints.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0} \\ & \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \end{aligned} \tag{A.32}$$

The Lagrangian function with equality and inequality constraints is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T\boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\mu}^T\boldsymbol{g}(\boldsymbol{x}). \tag{A.33}$$

It is straightforward to combine the optimality conditions for equality and inequality constraints. Equality constraints can be seen as inequality constraints, which are always active (GRIVA *et al.*, 2009). The box below combines the necessary and sufficient conditions of an optimisation problem with equality and inequality constraints.

Optimality conditions with equality and inequality constraints

Necessary conditions (KKT)

$$\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0} \tag{A.34}$$

$$\boldsymbol{\mu}^* \leq \mathbf{0} \tag{A.35}$$

$$\boldsymbol{\mu}^{*T} \boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0} \tag{A.36}$$

$$\boldsymbol{Z}(\boldsymbol{x}^*)^T \nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \boldsymbol{Z}(\boldsymbol{x}^*) \text{ is positive semidefinite} \tag{A.37}$$

Sufficient conditions

$$\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) = \mathbf{0} \tag{A.38}$$

$$\boldsymbol{\mu}^* \leq \mathbf{0} \tag{A.39}$$

$$\boldsymbol{\mu}^{*T} \boldsymbol{g}(\boldsymbol{x}^*) = \mathbf{0} \tag{A.40}$$

$$\boldsymbol{Z}_+(\boldsymbol{x}^*)^T \nabla^2 \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\mu}^*) \boldsymbol{Z}_+(\boldsymbol{x}^*) \text{ is positive definite} \tag{A.41}$$

where $\boldsymbol{Z}$ is the null-space matrix of the Jacobian of the constraints at $\boldsymbol{x}^*$. $\boldsymbol{Z}_+$ is the basis for the null space of the Jacobian of the constraints with all equality constraints but only active inequality constraints.

A.1.4 Optimisation methods

This subsection explains a few optimisation methods. There are many more, but only the methods below are used in this Master's Thesis.

A.1.4.1 Newton's method for minimising

The Newton method for minimising is a method to solve unconstrained optimisation problems. It is based on Newton's method to find a root of $\boldsymbol{f}(\boldsymbol{x})$, where $\boldsymbol{f}(\boldsymbol{x})$ is a vector of single value functions $f_i(\boldsymbol{x})$. First, let us review the Newton method for finding a root. The Taylor series approximation for the function $\boldsymbol{f}$ at the point $\boldsymbol{x}_k$ is

$$\boldsymbol{f}(\boldsymbol{x}_k + \boldsymbol{q}) \approx \boldsymbol{f}(\boldsymbol{x}_k) + \nabla \boldsymbol{f}(\boldsymbol{x}_k)^T \boldsymbol{q}, \tag{A.42}$$

where $\boldsymbol{x}_k$ is our estimation of the solution (root). We want to find the point $\boldsymbol{x}^*$, where $\boldsymbol{f}(\boldsymbol{x}^*) = \mathbf{0}$. So, we can set the Taylor series approximation to zero and solve for $\boldsymbol{q}$.

$$\boldsymbol{q} = -\nabla \boldsymbol{f}(\boldsymbol{x})^{-T} \boldsymbol{f}(\boldsymbol{x}) \tag{A.43}$$

The idea is to interpret $\boldsymbol{q}$ as a step we need to walk to find the root of $\boldsymbol{f}$. Hence, our new estimate of the solution is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{q} = \boldsymbol{x}_k - \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-T} \boldsymbol{f}(\boldsymbol{x}_k). \tag{A.44}$$

Newton's method for minimising is based on the same logic. However, we are not trying to find the root of $\boldsymbol{f}(\boldsymbol{x})$ but of $\nabla f(\boldsymbol{x})$. Please note that $\nabla f(\boldsymbol{x})$ is also a vector of functions. We are trying to find $\nabla f(\boldsymbol{x}) = \mathbf{0}$ because it is one necessary optimality condition; see equation (A.9). Following the same procedure for $\nabla f(\boldsymbol{x})$ instead of $\boldsymbol{f}(\boldsymbol{x})$ we get the Taylor series approximation

$$\nabla f(\boldsymbol{x}_k + \boldsymbol{q}) \approx \nabla f(\boldsymbol{x}_k) + \nabla^2 f(\boldsymbol{x}_k) \boldsymbol{q}, \tag{A.45}$$

and the search direction $\boldsymbol{q}$ becomes

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{q} = \boldsymbol{x}_k - \left[\nabla^2 f(\boldsymbol{x}_k)\right]^{-1} \nabla f(\boldsymbol{x}_k). \tag{A.46}$$

Please note that $\nabla^2 f(\boldsymbol{x})$ is symmetric. However, usually, the search direction $\boldsymbol{q}$ is obtained by solving the linear system

$$\left[\nabla^2 f(\boldsymbol{x}_k)\right] \boldsymbol{q} = -\nabla f(\boldsymbol{x}_k). \tag{A.47}$$

An optimisation algorithm solves the linear system of equation A.47 to determine the search direction $\boldsymbol{q}$ at every iteration. The next iteration is at $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{q}$. Those steps are repeated until the algorithm converges to $\boldsymbol{x}^*$. Newton's method is a prevalent optimisation method due to its quadratic convergence rate. The Taylor series approximation can prove the quadratic convergence rate and is left to the reader (GRIVA *et al.*, 2009). The box below summarises the steps of Newton's methods for minimising. Please note that this is the simplest form of Newton's method. Usually, algorithms are more advanced.

Newton's method algorithm steps

1.Check for optimality: If $\left\|\nabla f(\boldsymbol{x}_k)\right\| \leq \epsilon$, then stop.

2.Solve linear system for $\boldsymbol{q}$:

$$\left[\nabla^2 f(\boldsymbol{x}_k)\right] \boldsymbol{q} = -\nabla f(\boldsymbol{x}_k)$$

3.Make step

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{q}$$

A.1.4.2 Quasi-Newton method (BGFS)

Quasi-Newton methods are methods to solve unconstrained optimisation problems. The disadvantage of Newton's method is that it requires determining the Hessian matrix $\boldsymbol{H} = \nabla^2 f(\boldsymbol{x})$ at every iteration $\boldsymbol{x}_k$, which is very computationally expansive. Quasi-Newton methods approximate the Hessian matrix (or the inverse of the Hessian) by the gradient $\nabla f(\boldsymbol{x})$ at the current and previous iterations. This subsection will only discuss the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. The approximation of the Hessian of Quasi-Newton methods is based on the Taylor series approximation of equation (A.45). Since $\boldsymbol{q} = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k$ we can rewrite the Taylor series approximation to

$$\nabla f(\boldsymbol{x}_k + \boldsymbol{x}_{k+1} - \boldsymbol{x}_k) \approx \nabla f(\boldsymbol{x}_k) + \nabla^2 f(\boldsymbol{x}_k)\,(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) \tag{A.48}$$

$$\nabla f(\boldsymbol{x}_{k+1}) \approx \nabla f(\boldsymbol{x}_k) + \nabla^2 f(\boldsymbol{x}_k)\,(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k), \tag{A.49}$$

and rewrite the equation to

$$\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k) \approx \nabla^2 f(\boldsymbol{x}_k)\,(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k). \tag{A.50}$$

Let us call $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$ and $\boldsymbol{s}_k = (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k)$ and we get

$$\boldsymbol{y}_k \approx \nabla^2 f(\boldsymbol{x}_k)\,\boldsymbol{s}_k. \tag{A.51}$$

This approximation above is the foundation of the approximation of the Hessian matrix $\boldsymbol{H} = \nabla^2 f$. Let us call the approximation of the Hessian $\boldsymbol{B}$ and define that the approximation needs to satisfy

$$\boldsymbol{y}_k = \boldsymbol{B}_{k+1}\,\boldsymbol{s}_k. \tag{A.52}$$

By using some properties of the Hessian matrix, we can define $\boldsymbol{B}_{k+1}$ as

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k} \tag{A.53}$$

(LUENBERGER; YE, 2008). The box below lists the steps of how an algorithm uses the approximation of the Hessian $\boldsymbol{B}_{k+1}$ to solve an optimisation problem.

Quasi-Newton's method algorithm steps (Hessian approx.)

1.Check for optimality: If $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$, then stop.

2.Solve linear system for $\boldsymbol{q}$:

$$\boldsymbol{B}_k \boldsymbol{q} = -\nabla f(\boldsymbol{x}_k)$$

3.Perform line search to get α_k

4.Set $\boldsymbol{s}_k = \alpha_k \boldsymbol{q}_k$ and make step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$

5.Calculate $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$

6.Update approximation of Hessian

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}$$

Please note that the user needs to give an initial guess for $\boldsymbol{B}_k$. The identity matrix $\boldsymbol{I}$ is a common choice because this boils down to the gradient descent method in the first iteration. One can see that the algorithm uses a so-called line search to get α_k. α_k is a positive scalar which determines how long the step in the search direction should be. There are different line search methods, but they are not discussed in this Thesis. The Newton method does not need a line search because $\alpha_k = 1$ is mostly the best choice.

The algorithm above needs to solve a linear system at every iteration k (step 2). It would be better to have an updated formula for the inverse of the Hessian because then we do not need to solve the linear system at every iteration. One can use the Sherman-Morrison formula to calculate the inverse of $\boldsymbol{B}_{k+1}$.

$$\boldsymbol{B}_{k+1}^{-1} = \boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{(\boldsymbol{s}_k^T \boldsymbol{y}_k + \boldsymbol{y}_k^T \boldsymbol{H}_k \boldsymbol{y}_k)(\boldsymbol{s}_k \boldsymbol{s}_k^T)}{(\boldsymbol{s}_k^T \boldsymbol{y}_k)^2} - \frac{\boldsymbol{H}_k \boldsymbol{y}_k \boldsymbol{s}_k^T + \boldsymbol{s}_k \boldsymbol{y}_k^T \boldsymbol{H}_k}{\boldsymbol{s}_k^T \boldsymbol{y}_k}, \tag{A.54}$$

where $\boldsymbol{H}_k \equiv \boldsymbol{B}_k^{-1}$ (LUENBERGER; YE, 2008). The box below shows the steps of an algorithm using the inverse approximation of the Hessian matrix.

Quasi-Newton's method algo. steps (inv. Hessian approx.)

1.Check for optimality: If $\|\nabla f(\boldsymbol{x}_k)\| \leq \epsilon$, then stop.

2.Calculate search direction $\boldsymbol{q}$:

$$\boldsymbol{q} = -\boldsymbol{H}_k \nabla f(\boldsymbol{x}_k)$$

3.Perform line search to get α_k

4.Set $\boldsymbol{s}_k = \alpha_k \boldsymbol{q}_k$ and make step $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{s}_k$

5.Calculate $\boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)$

6.Update approximation of the inverse of the Hessian

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{(\boldsymbol{s}_k^T \boldsymbol{y}_k + \boldsymbol{y}_k^T \boldsymbol{H}_k \boldsymbol{y}_k)(\boldsymbol{s}_k \boldsymbol{s}_k^T)}{(\boldsymbol{s}_k^T \boldsymbol{y}_k)^2} - \frac{\boldsymbol{H}_k \boldsymbol{y}_k \boldsymbol{s}_k^T + \boldsymbol{s}_k \boldsymbol{y}_k^T \boldsymbol{H}_k}{\boldsymbol{s}_k^T \boldsymbol{y}_k}$$

A.1.4.3 Limited memory Quasi-Newton method (L-BGFS)

The limited memory Quasi-Newton method is an optimisation method to minimise an unconstrained optimisation problem. This subsection only describes the limited memory method for the Quasi-Newton method using the BGFS update formula (GRIVA *et al.*, 2009). This algorithm is not used in the Master's Thesis, but understanding it will help the reader understand more advanced algorithms in this Master's Thesis. Please note that there are also different limited memory methods for other optimisation methods. The disadvantage of the Quasi-Newton methods is that they require storing matrices, and it is necessary to perform matrix-vector multiplications. The limited memory Quasi-Newton methods only require storing vectors instead of matrices. Let us look at the updated formula of the approximation of the inverse of the Hessian again according to the BGFS update formula.

$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{(\boldsymbol{s}_k^T \boldsymbol{y}_k + \boldsymbol{y}_k^T \boldsymbol{H}_k \boldsymbol{y}_k)(\boldsymbol{s}_k \boldsymbol{s}_k^T)}{(\boldsymbol{s}_k^T \boldsymbol{y}_k)^2} - \frac{\boldsymbol{H}_k \boldsymbol{y}_k \boldsymbol{s}_k^T + \boldsymbol{s}_k \boldsymbol{y}_k^T \boldsymbol{H}_k}{\boldsymbol{s}_k^T \boldsymbol{y}_k} \tag{A.55}$$

$$= - \left[\boldsymbol{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right] \boldsymbol{H}_k \left[\boldsymbol{I} - \frac{\boldsymbol{y}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right] - \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} \tag{A.56}$$

One can see that $\boldsymbol{H}_{k+1}$ is only defined by vectors and the previous approximation of the Hessian $\boldsymbol{H}_k$. The idea of the limited memory method is to describe $\boldsymbol{H}_{k+1}$ only by $\boldsymbol{s}_k$ and $\boldsymbol{y}_k$ and $\boldsymbol{H}_k$. But instead of storing the matrix $\boldsymbol{H}_k$, $\boldsymbol{H}_k$ is described by $\boldsymbol{s}_{k-1}$ and $\boldsymbol{y}_{k-1}$ and $\boldsymbol{H}_{k-1}$. And $\boldsymbol{H}_{k-1}$ is also not stored but described by $\boldsymbol{s}_{k-2}$ and $\boldsymbol{y}_{k-2}$ and $\boldsymbol{H}_{k-2}$ and so on. That means that, in theory, we need to store all vectors $\boldsymbol{s}$ and $\boldsymbol{y}$ since the first iteration. However, in practice, the user specifies an integer n_{mc}, which defines how many previous iterations are considered to calculate $\boldsymbol{H}_{k+1}$. Usually, it is sufficient to choose n_{mc} between 3 and 5. However, that means that $\boldsymbol{H}_{k+1-n_{mc}}$ needs to be somehow initialised because the information to determine $\boldsymbol{H}_{k+1-n_{mc}}$ is not stored anymore. A simple approach is to choose $\boldsymbol{H}_{k+1-n_{mc}} = \boldsymbol{I}$, but there are more sophisticated methods.

Let us have a look at how an algorithm calculates the new search direction $\boldsymbol{q}_{k+1}$ by only storing vectors. The new search direction is

$$\boldsymbol{q}_{k+1} = \boldsymbol{H}_{k+1} \nabla f(\boldsymbol{x}_k) \tag{A.57}$$

$$= - \left[\boldsymbol{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right] \boldsymbol{H}_k \left[\boldsymbol{I} - \frac{\boldsymbol{y}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k}\right] \nabla f(\boldsymbol{x}_k) - \frac{\boldsymbol{s}_k \boldsymbol{s}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} \nabla f(\boldsymbol{x}_k), \tag{A.58}$$

Next, we define the three scalars t_0, t_1 and t_2 and rewrite the search direction to

$$\boldsymbol{q}_{k+1} = - \left[\boldsymbol{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k}{t_0}\right] \boldsymbol{H}_k \left[\nabla f(\boldsymbol{x}_k) - \boldsymbol{y}_k t_2\right] - \boldsymbol{s}_k \, t_2, \tag{A.59}$$

where

$$t_0 \equiv \boldsymbol{y}_k^T \boldsymbol{s}_k \tag{A.60}$$

$$t_1 \equiv \boldsymbol{s}_k^T \nabla f(\boldsymbol{x}_k) \tag{A.61}$$

$$t_2 \equiv t_1 / t_0 \tag{A.62}$$

Next we can define $\boldsymbol{u} \equiv \nabla f(\boldsymbol{x}_k) - \boldsymbol{y}_k t_2$ and get

$$\boldsymbol{q}_{k+1} = -\left[\boldsymbol{I} - \frac{\boldsymbol{s}_k \boldsymbol{y}_k}{t_0}\right] \boldsymbol{H}_k \boldsymbol{u} - \boldsymbol{s}_k \, t_2. \tag{A.63}$$

We must compute the matrix-vector multiplication $\boldsymbol{H}_k \boldsymbol{u}$. However, $\boldsymbol{H}_k$ is not explicitly known. Instead, we perform the matrix-vector multiplication in the same manner as we multiply the matrix $\boldsymbol{H}_{k+1}$ with the vector $\nabla f(\boldsymbol{x}_k)$ at the moment. (This requires starting with the computation of intermediate results of iteration $k - n_{mc}$). After this process we reassign $\boldsymbol{u} \leftarrow \boldsymbol{H}_k \boldsymbol{u}$ and further simplify $\boldsymbol{q}_{k+1}$ to

$$\boldsymbol{q}_{k+1} = -\boldsymbol{u}(t_4 - t_2)\boldsymbol{s}_k, \tag{A.64}$$

where

$$t_3 = \boldsymbol{y}_k^T / \boldsymbol{u} \tag{A.65}$$

$$t_4 = t_3 / t_0. \tag{A.66}$$

As we can see, it is possible to avoid storing any matrices and use only vector multiplications to compute the new search direction $\boldsymbol{q}_{k+1}$. However, please note that the BGFS update formula only used information from the last n_{mc} iterations. If the optimisation process is terminated before the n_{mc}^{th} iteration, the result of the L-BGFS method is equivalent to that of the BGFS method. The L-BGFS algorithm steps are the same as the algorithm steps of the BGFS method. The only difference is in step 2, which describes how the search direction is computed.

A.1.4.4 Penalty method

The penalty method is a method to solve a constrained optimisation problem. The idea of the penalty method is as simple as ingenious. We are trying to find the minimum value of a function subjected to certain equality constraints. Hence, we try to solve

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}. \end{aligned} \tag{A.67}$$

The idea of the penalty method is to transform a constrained optimisation problem into an unconstrained optimisation problem which is 'easy' to solve. The unconstrained optimisation problem, which we will get from the penalty method, can be solved by any unconstrained optimisation method. Thus, for example, with Newton's method or with Quasi-Newton methods. Suppose we do not satisfy the constraint at a certain iteration step k during the optimisation. We can 'force' the algorithm to seek a feasible solution by adding a value to the objective function. This additional term is called the *penalty term*. A common choice for the penalty term is $\frac{1}{2}\rho_k \boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{h}(\boldsymbol{x})$. The more the constraint is violated, the higher the penalty term is. Hence, the penalty method converts a constraint optimisation into an unconstrained optimisation problem, which reads

$$\min_{\boldsymbol{x}} \quad \pi(\boldsymbol{x}, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{h}(\boldsymbol{x}). \tag{A.68}$$

The larger ρ_k, the more attention is given to the feasibility. However, if ρ_k is too large, the problem might become ill-conditioned. The procedure is to increase ρ_k at (almost) every iteration. Hence, we solve multiple sub-problems until we reach convergence (GRIVA *et al.*, 2009). The box below shows the steps for a penalty method algorithm.

Penalty method algorithm steps

1.Check for optimality: If $\left\|\nabla\pi(\boldsymbol{x},\rho_k)\right\| \leq \epsilon$, then stop.

2.Solve unconstrained sub-problem:

$$\min_{\boldsymbol{x}} \quad \pi(\boldsymbol{x},\rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{h}(\boldsymbol{x})$$

3.Increase ρ_k

$$\rho_{k+1} \geq \rho_k$$

A.1.4.5 Augmented Lagrangian method

The ALM is a method to solve constrained optimisation problems. This subsection first only considers equality constraints, and later, the theory is extended to inequality constraints to solve the general optimisation problem stated in problem (A.32).

We saw that a first-order optimality condition of a constrained optimisation problem is that $\nabla\mathcal{L}(\boldsymbol{x}^*,\boldsymbol{\lambda}^*) = \boldsymbol{0}$, which is a very similar first-order condition to the unconstrained optimisation problem ($\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$). A question may arise: Why do we not minimise $\mathcal{L}$ and not bother about the constraints anymore? Unfortunately, this is not possible because then we assume that $\mathcal{L}(\boldsymbol{x}^*,\boldsymbol{\lambda}^*)$ is a minimiser, which is, in general, not the case. Usually $\mathcal{L}(\boldsymbol{x}^*,\boldsymbol{\lambda}^*)$ is a saddle point. The ALM augments the Lagrangian function by an additional penalty term to overcome this problem. The penalty term comes from the penalty method, which is explained in the subsection A.1.4.4.

ALM equality constraints

The idea of the ALM is to combine the Lagrangian function with the penalty method. If we add a penalty term to the Lagrangian function shown in equation (A.15), we get the augmented Lagrangian formulation, which is defined as

$$\mathcal{A}(\boldsymbol{x},\boldsymbol{\lambda},\rho_k) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T\boldsymbol{h}(\boldsymbol{x}) + \frac{1}{2}\rho_k \boldsymbol{h}(\boldsymbol{x})^T\boldsymbol{h}(\boldsymbol{x}). \tag{A.69}$$

The penalty term of the augmented Lagrangian ensures that $\mathcal{A}(\boldsymbol{x}^*,\boldsymbol{\lambda}^*,\rho_k)$ is a minimiser. We can increase ρ_k at (almost) every iteration and solve multiple unconstrained sub-problems, similar to the penalty method. However, a question remains: How do we know the values of the Lagrange multipliers? Initially, we need to guess the Lagrange multipliers. However, we update $\boldsymbol{\lambda}_k$ at every iteration in order to get a better guess of the Lagrange multipliers $\boldsymbol{\lambda}^*$ at the solution $\boldsymbol{x}^*$. If we assume that we will reach convergence at the next iteration, we say that

$$\nabla\mathcal{A}(\boldsymbol{x}_{k+1},\boldsymbol{\lambda}_k,\rho_k) = \boldsymbol{0}, \tag{A.70}$$

or

$$\nabla\mathcal{A}(\boldsymbol{x}_{k+1},\boldsymbol{\lambda}_k,\rho_k) = \boldsymbol{0} = \nabla f(\boldsymbol{x}_{k+1}) + \nabla\boldsymbol{h}(\boldsymbol{x}_{k+1})\boldsymbol{\lambda}_k + \rho_k\nabla\boldsymbol{h}(\boldsymbol{x}_{k+1})^T\boldsymbol{h}(\boldsymbol{x}_{k+1}), \tag{A.71}$$

which can be rearranged as

$$\begin{aligned}\nabla\mathcal{A}(\boldsymbol{x}_{k+1},\boldsymbol{\lambda}_k,\rho_k) = \boldsymbol{0} &= \nabla f(\boldsymbol{x}_{k+1}) + \nabla\boldsymbol{h}(\boldsymbol{x}_{k+1})\boldsymbol{\lambda}_k + \rho_k\nabla\boldsymbol{h}(\boldsymbol{x}_{k+1})^T\boldsymbol{h}(\boldsymbol{x}_{k+1}) &\text{(A.72)}\\ &= \nabla f(\boldsymbol{x}_{k+1}) + \nabla\boldsymbol{h}(\boldsymbol{x}_{k+1})[\boldsymbol{\lambda}_k + \rho_k\boldsymbol{h}(\boldsymbol{x}_{k+1})]. &\text{(A.73)}\end{aligned}$$

Hence, a logical choice to update $\boldsymbol{\lambda}$ is

$$\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \rho_k \boldsymbol{h}(\boldsymbol{x}_{k+1}), \tag{A.74}$$

because then

$$\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}) = \nabla f(\boldsymbol{x}_{k+1}) + \nabla \boldsymbol{h}(\boldsymbol{x}_{k+1}) \boldsymbol{\lambda}_{k+1} = \boldsymbol{0}, \tag{A.75}$$

which is an optimality condition for a constrained optimisation problem. The algorithm will terminate when $\nabla \mathcal{L}(\boldsymbol{x}_{k+1}, \boldsymbol{\lambda}_{k+1}) = \boldsymbol{0}$ or $\left\| \nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \right\| \leq \epsilon$. An ALM algorithm can be divided into three steps: 1. Check for optimality, 2. Solve unconstrained sub-problem, 3. Update $\boldsymbol{\lambda}_k$ and ρ_k. Those three steps are repeated until convergence is achieved (GRIVA *et al.*, 2009). The three steps are described in the box below.

Augmented Lagrangian method algorithm steps

1.Check for optimality: If $\left\| \nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \right\| \leq \epsilon$, then stop.

2.Solve unconstrained sub-problem:

$$\min_{\boldsymbol{x}} \quad \mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \rho_k) = f(\boldsymbol{x}) + \boldsymbol{\lambda}_k^T \boldsymbol{h}(\boldsymbol{x}) + \frac{1}{2} \rho_k \boldsymbol{h}(\boldsymbol{x})^T \boldsymbol{h}(\boldsymbol{x})$$

which gives $\boldsymbol{x}_{k+1}$.

3.Update $\boldsymbol{\lambda}_k$ and ρ_k $\boldsymbol{\lambda}_{k+1} = \boldsymbol{\lambda}_k + \rho_k \boldsymbol{h}(\boldsymbol{x}_{k+1})$

$$\rho_{k+1} \geq \rho_k$$

ALM equality and inequality constraints

The previous subsection describes solving a constrained optimisation problem with the ALM. However, the method described above can only handle equality constraints. This subsection describes how the theory is extended to inequality constraints. There are multiple methods to include inequality constraints, see, e.g. (GILL *et al.*, 1986), (PILLO; GRIPPO, 1982), (BIRGIN *et al.*, 2005), (ROCKAFELLAR, 1973a). Nevertheless, in this Master's Thesis, the method of Fletcher (1975) will be used. The augmented Lagrangian formulation of equation (A.69) can be rewritten to an equivalent formulation which is [1]

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2} \rho_k \left\| \boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right\|_2^2 \tag{A.76}$$

$$= f(\boldsymbol{x}) + \frac{1}{2} \rho_k \left\| \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) \right\|_2^2, \tag{A.77}$$

where $\tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \rho_k) = \boldsymbol{h}(\boldsymbol{x}) + \boldsymbol{\lambda}_k / \rho_k$. Please note the similarity to the penalty method (equation (A.68)), where $\left\| 1/2 \, \rho_k \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) \right\|_2^2$ can be seen as a penalty term which gets added to the objective function. Fletcher (1975) proposed to add the inequality constraints $\boldsymbol{g}(\boldsymbol{x})$ to the augmented Lagrangian function in a similar way as the equality constraints. However, the inequality constraints should be ignored if the current iteration is feasible enough. One method to do so is by using a *max* function. Fletcher proposed augmented Lagrangian function is

[1] Actually the rewritten formulation is $\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2} \rho_k \left\| \boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right\|_2^2 + C$, where $C = -1/2 \left(\boldsymbol{\lambda_k} / \rho_k \right)^T \left(\boldsymbol{\lambda_k} / \rho_k \right)$ but since C is a constant it does not influence the solution $\boldsymbol{x}^*$. A constant shifts a curve up or down, but the location of the minimiser stays unaltered.

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right\|_2^2 + \left\| \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle \right\|_2^2 \right) \tag{A.78}$$

$$= f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) \right\|_2^2 + \left\| \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k) \right\|_2^2 \right), \tag{A.79}$$

where $\tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k, \rho_k) = \left\langle \boldsymbol{g}(\boldsymbol{x}) + \boldsymbol{\mu}_k/\rho_k \right\rangle$ and $\langle a \rangle = \max(0, a)$. This implies that $\tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k, \rho_k)$ is a vector where all elements are greater or equal to zero. $\boldsymbol{\mu}_k$ are the estimated Lagrange multipliers associated with the inequality constraints. Please note that in the expression above, the equality and inequality terms get penalised by the same penalisation factor ρ_k. However, choosing two separate penalty factors for the equality and inequality constraints is also possible. Next, instead of using the norm notation, we can express the augmented Lagrangian function by

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) \right\|_2^2 + \left\| \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k) \right\|_2^2 \right) \tag{A.80}$$

$$= f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k)^T \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) + \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k)^T \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k) \right) \tag{A.81}$$

$$= f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left(\boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right)^T \left(\boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right) + \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle^T \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle \right), \tag{A.82}$$

to make the differentiation in the next step more intuitive.

$$\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \rho_k \left(\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x})\, \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{g}(\boldsymbol{x})\, \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k) \right) \tag{A.83}$$

$$= \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \rho_k \left(\boldsymbol{J}_{\boldsymbol{h}_{\boldsymbol{x}}}{}^T\, \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) + \boldsymbol{J}_{\boldsymbol{g}_{\boldsymbol{x}}}{}^T\, \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k) \right) \tag{A.84}$$

$$= \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) + \rho_k \left(\boldsymbol{J}_{\boldsymbol{h}_{\boldsymbol{x}}}{}^T \left(\boldsymbol{h}(\boldsymbol{x}) + \frac{\boldsymbol{\lambda}_k}{\rho_k} \right) + \boldsymbol{J}_{\boldsymbol{g}_{\boldsymbol{x}}}{}^T \left\langle \boldsymbol{g}(\boldsymbol{x}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle \right), \tag{A.85}$$

where $\boldsymbol{J}_{\boldsymbol{h}_{\boldsymbol{x}}}$ and $\boldsymbol{J}_{\boldsymbol{h}_{\boldsymbol{x}}}$ are the Jacobian matrices with respect to $\boldsymbol{x}$ associated with $\boldsymbol{h}(\boldsymbol{x})$ and $\boldsymbol{g}(\boldsymbol{x})$, respectively. $\nabla_{\boldsymbol{x}} \mathcal{A}(\boldsymbol{x}, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k, \rho_k)$ can be used to solve the unconstrained sub-problem. A logical choice to update the Lagrange multipliers of the inequality constraints is

$$\boldsymbol{\mu}_{k+1} = \max \left\{ 0, \boldsymbol{\mu}_k \, + \, \rho_k \, \boldsymbol{g}(\boldsymbol{x}_{k+1}) \right\}. \tag{A.86}$$

Please note that by using the updating formula above, it is not possible that the Lagrange multipliers for inequality constraints are negative. This makes sense because a negative Lagrange multiplier implies that the objective function decreases towards the feasible region.

Augmented Lagrangian method (Fletcher) algorithm steps

1.Check for optimality: If $\left\| \nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k, \boldsymbol{\mu}_k) \right\| \leq \epsilon$, then stop.

2.Solve unconstrained sub-problem:

$$\min_{\boldsymbol{x}} \quad \mathcal{A}(\boldsymbol{x}) = f(\boldsymbol{x}) + \frac{1}{2}\rho_k \left(\left\| \tilde{\boldsymbol{h}}(\boldsymbol{x}, \boldsymbol{\lambda}_k) \right\|_2^2 + \left\| \tilde{\boldsymbol{g}}(\boldsymbol{x}, \boldsymbol{\mu}_k) \right\|_2^2 \right)$$

which gives $\boldsymbol{x}_{k+1}$.

3.Update $\boldsymbol{\lambda}_k$, $\boldsymbol{\mu}_k$ and ρ_k

$$\begin{aligned}
\boldsymbol{\lambda}_{k+1} &= \boldsymbol{\lambda}_k + \rho_k \boldsymbol{h}(\boldsymbol{x}_{k+1}) \\
\boldsymbol{\mu}_{k+1} &= \max\left\{0, \boldsymbol{\mu}_k + \rho_k \boldsymbol{g}(\boldsymbol{x}_{k+1})\right\} \\
\rho_{k+1} &\geq \rho_k
\end{aligned}$$

A.1.4.6 Sequential quadratic programming method

The sequential Quadratic Programming (SQP) method is a method to solve constraint optimisation problems. Suppose we want to solve the following optimisation problem.

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\
\text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{0}
\end{aligned} \tag{A.87}$$

The Lagrangian function of this problem is

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \boldsymbol{\lambda}^T \boldsymbol{h}(\boldsymbol{x}). \tag{A.88}$$

According to equation (A.17), one first-order necessary optimality condition of the stated problem is

$$\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \boldsymbol{0}. \tag{A.89}$$

Let us try to find $\nabla \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*) = \boldsymbol{0}$ by Newtons method explained in subsection A.1.4.1. Hence, we need to solve the linear system

$$\nabla^2 \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix} = -\nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k), \tag{A.90}$$

for the search direction $\begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix}$. At the next iteration we can update $\boldsymbol{x}_k$ and the Lagrange multipliers $\boldsymbol{\lambda}_k$ by

$$\begin{Bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{x}_k \\ \boldsymbol{\lambda}_k \end{Bmatrix} + \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix}. \tag{A.91}$$

We can repeat this process until the algorithm converges to $\boldsymbol{x}^*$ and $\boldsymbol{\lambda}^*$ (GRIVA *et al.*, 2009). But why does this method work? Let us analyse the linear system carefully again.

$$\nabla^2 \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix} = -\nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \tag{A.92}$$

$$\begin{bmatrix} \nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) & \nabla^2_{\boldsymbol{x\lambda}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \\ \nabla^2_{\boldsymbol{\lambda x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}) & \nabla^2_{\boldsymbol{\lambda\lambda}} (\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \end{bmatrix} \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix} = - \begin{Bmatrix} \nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \\ \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \end{Bmatrix} \tag{A.93}$$

$$\begin{bmatrix} \nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) & \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \\ \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T & \boldsymbol{0} \end{bmatrix} \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix} = - \begin{Bmatrix} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k) \boldsymbol{\lambda} \\ \boldsymbol{h}(\boldsymbol{x}_k) \end{Bmatrix} \tag{A.94}$$

The linear system represents the first-order necessary condition of the optimisation problem

$$\begin{aligned}
\min_{\boldsymbol{q}} \quad & d(\boldsymbol{q}) = \frac{1}{2} \boldsymbol{q}^T \left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \right] \boldsymbol{q} + \boldsymbol{q}^T \left[\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \right] \\
\text{s.t.} \quad & \boldsymbol{h}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = \boldsymbol{0},
\end{aligned} \tag{A.95}$$

which is a quadratic approximation of the Lagrangian function $\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ and a linear approximation of the constrain $\boldsymbol{h}(\boldsymbol{x})$ at the point $\boldsymbol{x}_k$ and $\boldsymbol{\lambda}_k$ (GRIVA *et al.*, 2009). It may not be that trivial to see that the linear system is the first-order necessary condition of the abovementioned optimisation problem. So, let us derive the first-order necessary conditions of the optimisation problem. The first-order necessary condition states that the gradient of the objective function $d(\boldsymbol{q})$ is a multiple of the gradient of the constraint $\boldsymbol{h}(\boldsymbol{q})$. Thus,

$$\nabla_{\boldsymbol{q}} d(\boldsymbol{q}^*) + \nabla_{\boldsymbol{q}} \boldsymbol{h}(\boldsymbol{q}^*)\,\boldsymbol{\gamma} = \boldsymbol{0} \tag{A.96}$$

$$\left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\right] \boldsymbol{q} \;+\; \left[\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\right] + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)\,\boldsymbol{\gamma} = \boldsymbol{0} \tag{A.97}$$

$$\left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\right] \boldsymbol{q} \;+\; \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)\,\boldsymbol{\gamma} = -\left[\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\right] \tag{A.98}$$

$$\left[\nabla^2_{\boldsymbol{xx}} \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\right] \boldsymbol{q} \;+\; \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)\,\boldsymbol{\gamma} = -\nabla_{\boldsymbol{x}} f(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)\boldsymbol{\lambda}, \tag{A.99}$$

where $\boldsymbol{\gamma}$ are the Lagrange multipliers of the optimisation problem stated in equation (A.95). Please note that the gradient with respect to $\boldsymbol{q}$ of the constraint $\boldsymbol{h}(\boldsymbol{x}_k) + \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q}$ is equal to the gradient of $\boldsymbol{h}(\boldsymbol{q})$ at $\boldsymbol{x}_k$. Thus, $\nabla_{\boldsymbol{q}} \boldsymbol{h}(\boldsymbol{q}^*) = \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)$. The second first-order condition is that the constraint is not violated. The constraint is satisfied if

$$\boldsymbol{h}(\boldsymbol{x}_k) \;+\; \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = \boldsymbol{0} \tag{A.100}$$

$$\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}_k)^T \boldsymbol{q} = -\boldsymbol{h}(\boldsymbol{x}_k). \tag{A.101}$$

Combining equation (A.99) and (A.101) gives the linear system obtained by applying Newton's method to the optimisation problem stated in (A.87). Consequently, the SQP method works because, at every iteration, we solve a linear system which will satisfy the first-order necessary optimality condition of a quadratic approximation of the Lagrangian subjected to a linear approximation of the constraints. Repeating this process will usually converge to the solution $\boldsymbol{x}^*$. The box below shows the steps of an SQP algorithm. Note that we need to guess $\boldsymbol{\lambda}_0$ to determine the Lagrangian function (equation A.88) at the first iteration.

Sequential quadratic programming method algorithm steps

1.Check for optimality: If $\left\|\nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\right\| \leq \epsilon$, then stop.

2.Solve linear system:

$$\nabla^2 \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k) \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix} = -\nabla \mathcal{L}(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)$$

3.Make step and update $\boldsymbol{\lambda}$

$$\begin{Bmatrix} \boldsymbol{x}_{k+1} \\ \boldsymbol{\lambda}_{k+1} \end{Bmatrix} = \begin{Bmatrix} \boldsymbol{x}_k \\ \boldsymbol{\lambda}_k \end{Bmatrix} + \begin{Bmatrix} \boldsymbol{q}_k \\ \boldsymbol{\gamma}_k \end{Bmatrix}$$

A.1.5 Aggregation functions

Suppose we want to solve the optimisation problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & g_i(\boldsymbol{x}) \leq 0 \quad i = 1, 2, ..., n_g \end{aligned} \tag{A.102}$$

by a gradient-based optimisation method where n_g is a large integer. Hence, it is an optimisation problem with many inequality constraints. For all active constraints at $\boldsymbol{x}_k$, which means for all the constraints $g_i(\boldsymbol{x}_k)$ for which holds $g_i(\boldsymbol{x}_k) = 0$, we need to calculate the gradient. If the optimisation problem has many inequality constraints, it becomes computationally expensive to compute the gradient of all active constraints at $\boldsymbol{x}_k$. Aggregation functions can overcome this problem by combining all inequality constraints in one single scalar function. The simplest aggregation function is the *max* function

$$\bar{g}_{\max} = \max(\boldsymbol{g}(\boldsymbol{x})), \tag{A.103}$$

which returns the maximum value of all constraints. However, in practice, the *max* function performs very poorly, and the function is not differentiable, which makes it unsuitable for gradient-based optimisation. A better and differentiable aggregation function is the KS function and is defined as

$$\bar{g}_{KS}(\rho_{KS}, \boldsymbol{x}) = \frac{1}{\rho_{KS}} \ln \sum_{j=1}^{n_g} \exp\left(\rho_{KS}\, g_j(\boldsymbol{x})\right), \tag{A.104}$$

where ρ_{KS} is a scalar value which determines the accuracy of $\bar{g}_{KS}$ to the largest inequality constraint. The advantage of the KS function is that the contribution of inactive constraints is multiple orders of magnitude smaller compared to active constraints (POON; MARTINS, 2006). The KS function has several properties (HAFTKA; GüRDAL, 1992),(RASPANTI *et al.*, 2000):

1. $\bar{g}_{\max} < \bar{g}_{KS}(\rho_{KS}, \boldsymbol{x}) < \bar{g}_{\max} + \frac{\ln(n_g)}{\rho_{KS}}$
2. $\bar{g}_{KS}(\rho_{KS}, \boldsymbol{x}) \geq \bar{g}_{\max}$ for all $\rho > 0$
3. $\lim_{\rho_{KS} \to \infty} \bar{g}_{KS}(\rho_{KS}, \boldsymbol{x}) = \bar{g}_{\max}$
4. $\bar{g}_{KS}(\rho_{KS2}, \boldsymbol{x}) \geq \bar{g}_{KS}(\rho_{KS1}, \boldsymbol{x})\, \forall \boldsymbol{x}$ such that $\rho_{KS1} > \rho_{KS2}$
5. The KS function $\bar{g}_{KS}(\rho_{KS}, \boldsymbol{x})$ is insensitive to ρ_{KS} as ρ_{KS} becomes large
6. The gradient of the KS function with respect to $\boldsymbol{x}$ is independent of $\bar{g}_{\max}$
7. $\bar{g}_{KS}(\rho_{KS}, \boldsymbol{x})$ is convex if and only if all constraints are convex

Figure A.1 shows how the KS function approximates the maximum value of the three inequality functions

$$g_1(x) \leq \frac{1}{5}x + 1 \tag{A.105}$$

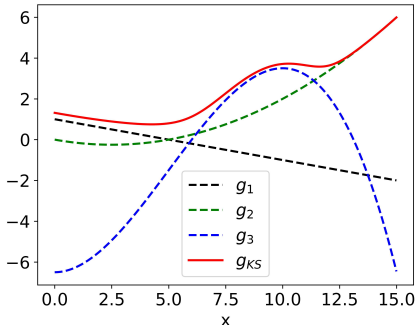
$$g_2(x) \leq \frac{4}{100}x^2 - \frac{1}{5}x \tag{A.106}$$

$$g_3(x) \leq -\frac{2}{100}x^3 + \frac{3}{10}x^2 - \frac{13}{2}. \tag{A.107}$$

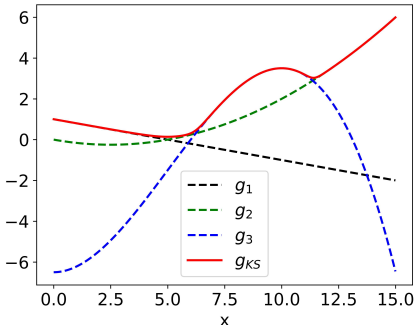
One can see that the larger ρ_{KS}, the better the KS function approximates the maximum value of all inequality constraints. However, if ρ_{KS} is too large, the optimisation problem can become ill-conditioned.

A.1.6 Automatic differentiation

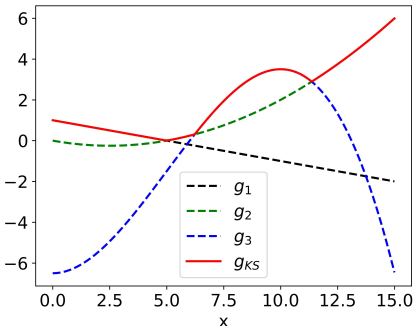
As the previous subsections showed, knowledge of the gradient is essential in optimisation. Usually, the constraint and objective functions are not explicitly known but are coded in a



(a) $\rho_{KS} = 1$



(b) $\rho_{KS} = 5$



(c) $\rho_{KS} = 50$

FIGURE A.1 – Influence of ρ_{KS} on accuracy of KS function.

computer script. There are several methods to get the gradient of a script, and AD is one of them. AD is a popular choice to get the gradient due to its precision, which is of the same order as the machine precision (NOCEDAL; WRIGHT, 2000). In a script, we use several variables; in this subsection, we will call those variables x_k. x_1 is the first assigned variable, x_2 is the second and so on until we assign x_n, where n is the total number of variables assigned in the script. AD is based on the chain rule. Two different chain rules exist that are equally valid. The first one is

$$\frac{dx_i}{dx_j} = \sum_{k=j}^{i-1} \frac{\partial x_i}{\partial x_k} \frac{dx_k}{dx_j}, \tag{A.108}$$

which is used for the *forward-mode AD*. The second chain rule is

$$\frac{dx_i}{dx_j} = \sum_{k=j+1}^{i} \frac{\partial x_k}{\partial x_j} \frac{dx_i}{dx_k}, \tag{A.109}$$

which is used for the *backward-mode AD* (MARTINS; NING, 2021). The second chain rule is less intuitive and less known but gives the same result as the first one. Both chain rules give the derivative of variable x_i with respect to the variable x_j, which could be any variable of our script. In both chain rules, the indices i and j are fixed, and we sum over the index k. One can see that the difference between both chain rules is that only the partial and total derivatives are interchanged. Let us consider the function

$$f(x_1, x_2) = x_5(x_2, x_3, x_4) = x_2 \cdot x_3(x_1) \cdot x_4(x_1, x_2), \tag{A.110}$$

where

$$x_3(x_1) = x_1^2 \tag{A.111}$$
$$x_4(x_1, x_3) = x_1 + x_3(x_1) \tag{A.112}$$

and suppose we want to know the derivative of $f = x_5$ with respect to x_1 and x_2. Thus, $i = 5$ and $j = 1, 2$. The chain rule used by the forward-mode AD gives

$$\frac{dx_5}{dx_1} = \sum_{k=j=1}^{i-1=4} \frac{\partial x_5}{\partial x_k} \frac{dx_k}{dx_1} \tag{A.113}$$
$$= \frac{\partial x_5}{\partial x_1} \frac{dx_1}{dx_1} + \frac{\partial x_5}{\partial x_2} \frac{dx_2}{dx_1} + \frac{\partial x_5}{\partial x_3} \frac{dx_3}{dx_1} + \frac{\partial x_5}{\partial x_4} \frac{dx_4}{dx_1} \tag{A.114}$$
$$= x_2(3x_1^2 + 4x_1^3). \tag{A.115}$$

and

$$\frac{dx_5}{dx_2} = \sum_{k=j=2}^{i-1=4} \frac{\partial x_5}{\partial x_k} \frac{dx_k}{dx_2} \tag{A.116}$$
$$= \frac{\partial x_5}{\partial x_2} \frac{dx_2}{dx_2} + \frac{\partial x_5}{\partial x_3} \frac{dx_3}{dx_2} + \frac{\partial x_5}{\partial x_4} \frac{dx_4}{dx_2} \tag{A.117}$$
$$= x_1^3 + x_1^4. \tag{A.118}$$

The chain rule of the backward-mode AD gives the same answer. So, how can we use both chain rules to calculate the derivatives of a script? A computer breaks down complex equations in simple operations like additions, multiplications, etc. and elementary functions like the sinus or exponential function. The script organises the simple operations and elementary functions in a computational graph. Figure A.2 shows the computational graph for our example. Every circle is called a *node*. The Forward- and backward-mode AD use this computational graph to determine the derivative. Both AD methods only calculate the partial derivatives and assemble them by the corresponding chain rule to get the total derivative.

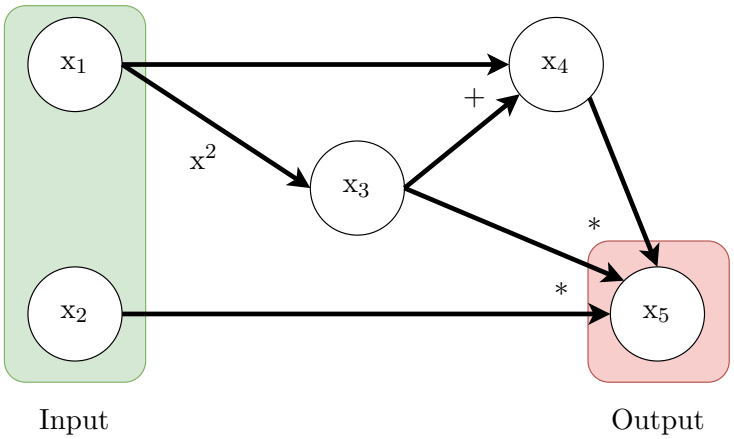


FIGURE A.2 – Computational graph for $f(\boldsymbol{x})$ of equation A.110.

A.1.6.1 Forward-mode AD

Forward-mode AD gets its name from the computational flow direction of the derivatives in the computational graph. The derivatives are determined in the computational graph from left to right in a forward direction. In a script are n_{in} input variables and n_d dependent variables, which depend on the input variables. Furthermore, there are n_{out} output variables, which are at the same time also dependent variables. The forward-mode AD calculates the directional

derivative of the dependent variables along the *seed vector* $\dot{\boldsymbol{v}} = \begin{pmatrix} \dot{x}_1 & \dot{x}_2 & \dots & \dot{x}_{n_{in}} \end{pmatrix}^T$. The seed vector is a unit vector whose elements are called *seeds*. Mathematically, the forward-mode calculates

$$\nabla_{\dot{\boldsymbol{v}}} \boldsymbol{x}_d = \nabla_{\boldsymbol{x}_{in}} \boldsymbol{x}_d^T \cdot \dot{\boldsymbol{v}} \tag{A.119}$$

$$\nabla_{\dot{\boldsymbol{v}}} \boldsymbol{x}_d = \boldsymbol{J}_{\boldsymbol{x}_d} \cdot \dot{\boldsymbol{v}}. \tag{A.120}$$

where $\boldsymbol{x}_d$ are all dependent variables, $\boldsymbol{x}_{in}$ are all input variables, $\boldsymbol{J}_{\boldsymbol{x}_d}$ is the Jacobian of the dependent variables with respect to the input variables and $\nabla_{\dot{\boldsymbol{v}}} \boldsymbol{x}_d$ is the directional derivative of the dependent variables with respect to the input variables along the direction of the vector $\dot{\boldsymbol{v}}$. The vector $\dot{\boldsymbol{v}}$ is an input of the forward-mode AD. However, we are usually only interested in the derivatives of the outputs of our script. Hence, we can write the result of the forward-mode AD as

$$\nabla_{\dot{\boldsymbol{v}}} \boldsymbol{x}_{out} = \boldsymbol{J}_{\boldsymbol{x}_{out}} \cdot \dot{\boldsymbol{v}}. \tag{A.121}$$

$\boldsymbol{J}_{\boldsymbol{x}_{out}}$ is a sub-matrix of $\boldsymbol{J}_{\boldsymbol{x}_d}$, which is a sub-matrix of $\boldsymbol{J}_{\boldsymbol{x}}$. In the most simple case, $\dot{\boldsymbol{v}}$ is the j^{th} canonical basis vector of the input space. In that case, the forward-mode AD determines the Jacobian matrix's j^{th} column, as shown below.

$$\boldsymbol{J}_{\boldsymbol{x}} = \begin{bmatrix} \frac{dx_1}{dx_1} & \frac{dx_1}{dx_2} & \cdots & \frac{dx_1}{dx_j} & \cdots & \frac{dx_1}{dx_n} \\ \frac{dx_2}{dx_1} & \frac{dx_2}{dx_2} & \cdots & \frac{dx_2}{dx_j} & \cdots & \frac{dx_2}{dx_n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{dx_j}{dx_1} & \frac{dx_j}{dx_2} & \cdots & \frac{dx_j}{dx_j} & \cdots & \frac{dx_j}{dx_n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{dx_n}{dx_1} & \frac{dx_n}{dx_2} & \cdots & \frac{dx_n}{dx_j} & \cdots & \frac{dx_n}{dx_n} \end{bmatrix} \tag{A.122}$$

Suppose we want to know the derivative dx_4/dx_1 of our example above. That means we need to calculate the column $j = 1$ of the Jacobian matrix because dx_4/dx_1 is the last element of the first column of the Jacobian matrix. So, we need to choose the first canonical basis vector as the seed vector, which is the unit vector along the x_1-axis. Thus, $\dot{\boldsymbol{v}} = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$. The forward-mode AD calculates at every node i the total derivative with respect to the first input node x_1, defined as $\dot{x}_i = dx_i/dx_1$. The total derivative of the input nodes are the elements of the seed vector. In our example the seed value of node $i = 1$ is $\dot{x}_1 = dx_1/dx_1 = 1$ and the seed value of node $i = 2$ is $\dot{x}_2 = dx_2/dx_1 = 0$. Next we move to the third node $i = 3$ and calculate $\dot{x}_3 = dx_3/dx_1$ by the chain rule of equation (A.108)

$$\frac{dx_3}{dx_1} = \dot{x}_3 = \sum_{k=j=1}^{i-1=2} \frac{\partial x_3}{\partial x_k} \frac{dx_k}{dx_1} \tag{A.123}$$

$$= \frac{\partial x_3}{\partial x_1} \frac{dx_1}{dx_1} + \frac{\partial x_3}{\partial x_2} \frac{dx_2}{dx_1} \tag{A.124}$$

$$= \frac{\partial x_3}{\partial x_1} \dot{x}_1 + \frac{\partial x_3}{\partial x_2} \dot{x}_2 \tag{A.125}$$

$$= 2x_1, \tag{A.126}$$

where we only need to calculate the partial derivatives of node 3 with respect to the two parent nodes 1 and 2 because we already know the total derivatives $\dot{x}_1$ and $\dot{x}_2$. Next, we can move to node $i = 4$ and calculate $\dot{x}_4$.

$$\frac{dx_4}{dx_1} = \dot{x}_4 = \sum_{k=j=1}^{i-1=3} \frac{\partial x_4}{\partial x_k} \frac{dx_k}{dx_1} \tag{A.127}$$

$$= \frac{\partial x_4}{\partial x_1} \frac{dx_1}{dx_1} + \frac{\partial x_4}{\partial x_2} \frac{dx_2}{dx_1} + \frac{\partial x_4}{\partial x_3} \frac{dx_3}{dx_1} \tag{A.128}$$

$$= \frac{\partial x_4}{\partial x_1} \dot{x}_1 + \frac{\partial x_4}{\partial x_2} \dot{x}_2 + \frac{\partial x_4}{\partial x_3} \dot{x}_3 \tag{A.129}$$

$$= 1 + 2x_1 \tag{A.130}$$

Ultimately, we can move to the output node $i = 5$ to calculate our desired derivative dx_5/dx_1.

$$\frac{dx_5}{dx_1} = \dot{x}_5 = \sum_{k=j=1}^{i-1=4} \frac{\partial x_5}{\partial x_k} \frac{dx_k}{dx_1} \tag{A.131}$$

$$= \frac{\partial x_5}{\partial x_1} \frac{dx_1}{dx_1} + \frac{\partial x_5}{\partial x_2} \frac{dx_2}{dx_1} + \frac{\partial x_5}{\partial x_3} \frac{dx_3}{dx_1} + \frac{\partial x_5}{\partial x_4} \frac{dx_4}{dx_1} \tag{A.132}$$

$$= \frac{\partial x_5}{\partial x_1} \dot{x}_1 + \frac{\partial x_5}{\partial x_2} \dot{x}_2 + \frac{\partial x_5}{\partial x_3} \dot{x}_3 + \frac{\partial x_5}{\partial x_4} \dot{x}_4 \tag{A.133}$$

$$= x_2(3x_1^2 + 4x_1^3) \tag{A.134}$$

One can see that the forward-mode AD walks along the computational tree from left to right and determines the partial derivatives on its way. The forward-mode AD reuses the previously determined total derivatives to calculate the total derivative at the current node. Once the forward-mode AD reaches the computational tree's right side, the desired directional derivative along the seed vector $\dot{\boldsymbol{v}}$ is known. Suppose that not only x_5 is an output of the script, but also x_4. In this case, we do not need to apply the forward-mode AD again because dx_4/dx_1 is an intermediate result of the process. However, suppose we want the derivative dx_5/dx_2. In this case, we need to choose as the seed vector the second canonical basis vector, e.g. $\dot{\boldsymbol{v}} = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$, and perform the forward-mode AD again. Consequently, the computational price to get the directional derivative of the function outputs along the canonical basis vectors of the input space by the forward-mode AD is independent of the number of output functions but scales with the input variables. This becomes clear by observing the Jacobian matrix. Let us consider the Jacobian matrix of our example.

$$\boldsymbol{J}_{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \frac{dx_3}{dx_1} & \frac{dx_3}{dx_2} & 1 & 0 & 0 \\ \frac{dx_4}{dx_1} & \frac{dx_4}{dx_2} & \frac{dx_4}{dx_3} & 1 & 0 \\ \frac{dx_5}{dx_1} & \frac{dx_5}{dx_2} & \frac{dx_5}{dx_3} & \frac{dx_5}{dx_4} & 1 \end{bmatrix} \tag{A.135}$$

Our chosen seed vector is $\dot{\boldsymbol{v}} = \begin{pmatrix} 1 & 0 \end{pmatrix}^T$ which is encircled in orange. Because our seed vector is the first canonical basis vector, the forward-mode AD determined the Jacobian matrix's first column, shown in blue. The Jacobian matrix of all dependent variables is encircled in green. The Jacobian matrix of the output $\boldsymbol{J}_{\boldsymbol{x}_{out}}$ nodes is encircled in red. As someone can see, the forward-mode AD calculates the directional derivative along the x_1-axis of all variables in the script. However, please note that the output of the forward-mode AD is just $\nabla_{\dot{\boldsymbol{v}}} \boldsymbol{x}_{out} = \boldsymbol{J}_{\boldsymbol{x}_{out}} \cdot \dot{\boldsymbol{v}} = dx_5/dx_1$, which is the intersection of the blue and red circles. All other derivatives encircled in blue are intermediate results. Additionally, one can observe that we need to perform the forward-mode AD again to get dx_5/dx_2.

A.1.6.2 Backward-mode AD

The backward-mode AD gets its name from the computational flow direction of the derivatives. The result of the backward-mode AD can be expressed as

$$\bar{\boldsymbol{x}} = \boldsymbol{J}_{\boldsymbol{x}_{out}}^{T} \cdot \bar{\boldsymbol{v}}, \tag{A.136}$$

where $\boldsymbol{J}_{\boldsymbol{x}_{out}}$ is the Jacobian matrix of the output variables with respect to the input variables, and $\bar{\boldsymbol{v}}$ is a vector spanned in the output space. $\bar{\boldsymbol{v}}$ is also referred as the *reversed seed vector*. In the simplest case, $\bar{\boldsymbol{v}}$ is the $(n_x - j + 1)^{th}$ canonical basis vector in the output space. In that case, the backward-mode AD determines the Jacobian matrix's j^{th} row as shown below.

$$\boldsymbol{J}_{\boldsymbol{x}} = \begin{bmatrix} \frac{dx_1}{dx_1} & \frac{dx_1}{dx_2} & \cdots & \frac{dx_1}{dx_j} & \cdots & \frac{dx_1}{dx_n} \\ \frac{dx_2}{dx_1} & \frac{dx_2}{dx_2} & \cdots & \frac{dx_2}{dx_j} & \cdots & \frac{dx_2}{dx_n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{dx_j}{dx_1} & \frac{dx_j}{dx_2} & \cdots & \frac{dx_j}{dx_j} & \cdots & \frac{dx_j}{dx_n} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ \frac{dx_n}{dx_1} & \frac{dx_n}{dx_2} & \cdots & \frac{dx_n}{dx_j} & \cdots & \frac{dx_n}{dx_n} \end{bmatrix} \tag{A.137}$$

The backward-mode AD consists of two sweeps: a forward sweep and a backward sweep. The forward sweep walks the computational tree from the left to the right, and the backward sweep from the right to the left. In the forward sweep, all numerical values of all nodes are determined. Additionally, the forward sweep determines and stores all partial derivatives on its way. Next, the backward sweep gets initialised. The backward-mode AD assigns to every node an *adjoint variable*. The adjoint variables of the output nodes are the values in the reversed seed vector, which the user specifies. The adjoint variables of all other nodes are defined as $\bar{x}_j = dx_n/dx_j$, where x_n is the last assigned variable in the script (which is consequently also an output of the script). The backward-mode AD determines the adjoint variables $\bar{x}_j$ by the second chain rule of equation (A.109). Let us consider our example again. Figure A.3 shows the computational tree of our example after the forward sweep where we determined the numerical values of x_1 - x_5 and all numerical values of the partial derivatives shown in the figure. Our output space is only one-dimensional. Hence, we can only choose $\bar{\boldsymbol{v}}$ as a scalar. So, let us choose $\bar{\boldsymbol{v}} = 1$ to get the last row of the Jacobian matrix. Hence, $\bar{x}_5 = 1$. Next, we start the backward sweep at node $j = 4$ and determine the adjoint

$$\bar{x}_4 = \frac{dx_5}{dx_4} = \sum_{k=j+1=5}^{i=5} \frac{\partial x_k}{\partial x_4} \frac{dx_5}{dx_k} \tag{A.138}$$

$$= \frac{\partial x_5}{\partial x_4} \frac{dx_5}{dx_5} \tag{A.139}$$

$$= x_2 x_1^2. \tag{A.140}$$

Next, we move to node $j = 3$ and determine the adjoint $\bar{x}_3$.

$$\frac{dx_5}{dx_3} = \bar{x}_3 = \sum_{k=j+1=4}^{i=5} \frac{\partial x_k}{\partial x_3} \frac{dx_5}{dx_k} \tag{A.141}$$

$$= \frac{\partial x_4}{\partial x_3} \frac{dx_5}{dx_4} + \frac{\partial x_5}{\partial x_3} \frac{dx_5}{dx_5} \tag{A.142}$$

$$= \frac{\partial x_4}{\partial x_3} \bar{x}_4 + \frac{\partial x_5}{\partial x_3} \bar{x}_5 \tag{A.143}$$

$$= x_2(2x_1^2 + x_1), \tag{A.144}$$

where $\partial x_4/\partial x_3$, $\bar{x}_4$, $\partial x_5/\partial x_3$ and $\bar{x}_5$ are known. Moving on to node $j = 2$ and determine by the chain rule $\bar{x}_2$

$$\frac{dx_5}{dx_2} = \bar{x}_2 = \sum_{k=j+1=3}^{i=5} \frac{\partial x_k}{\partial x_2} \frac{dx_5}{dx_k} \tag{A.145}$$

$$= \frac{\partial x_3}{\partial x_2} \frac{dx_5}{dx_3} + \frac{\partial x_4}{\partial x_2} \frac{dx_5}{dx_4} + \frac{\partial x_5}{\partial x_2} \frac{dx_5}{dx_5} \tag{A.146}$$

$$= \frac{\partial x_3}{\partial x_2} \bar{x}_3 + \frac{\partial x_4}{\partial x_2} \bar{x}_4 + \frac{\partial x_5}{\partial x_2} \bar{x}_5 \tag{A.147}$$

$$= x_1^3 + x_1^4, \tag{A.148}$$

where we use again the intermediate results $\bar{x}_3$ and $\bar{x}_4$. Ultimately, we can calculate the derivative $\bar{x}_1$ by

$$\frac{dx_5}{dx_1} = \bar{x}_1 = \sum_{k=j+1=2}^{i=5} \frac{\partial x_k}{\partial x_1} \frac{dx_5}{dx_k} \tag{A.149}$$

$$= \frac{\partial x_2}{\partial x_1} \frac{dx_5}{dx_2} + \frac{\partial x_3}{\partial x_1} \frac{dx_5}{dx_3} + \frac{\partial x_4}{\partial x_1} \frac{dx_5}{dx_4} + \frac{\partial x_5}{\partial x_1} \frac{dx_5}{dx_5} \tag{A.150}$$

$$= \frac{\partial x_2}{\partial x_1} \bar{x}_2 + \frac{\partial x_3}{\partial x_1} \bar{x}_3 + \frac{\partial x_4}{\partial x_1} \bar{x}_4 + \frac{\partial x_5}{\partial x_1} \bar{x}_5 \tag{A.151}$$

$$= x_2(4x_1^3 + 3x_1^2). \tag{A.152}$$

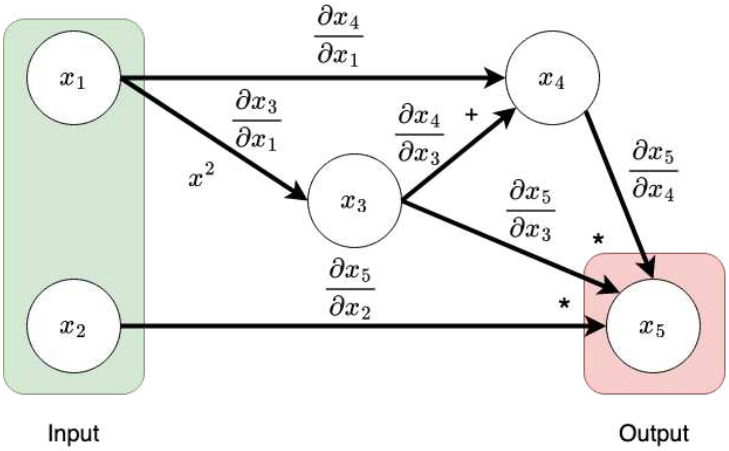


FIGURE A.3 – Computational graph after forward sweep. In the forward sweep, the partial derivatives are determined.

As you can see, we determine the derivative by walking the computational tree backwards and exploiting how the second chain rule is defined. Furthermore, one can see that $\bar{x}_2 = dx_5/dx_2$ is an intermediate result. So, we do not need to repeat the backward-mode AD to get the derivative of the output function with respect to a different input variable. Nevertheless, we need to repeat

the backward-mode AD to know the total derivative of a different output function. That means the backward-mode AD is beneficial if the script has many input variables and only a few output functions, as is usually the case in optimisation problems. Let us observe the Jacobian matrix of our example again and see which derivatives the backward-mode AD determines.

$$\boldsymbol{J}_{\boldsymbol{x}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \frac{dx_3}{dx_1} & \frac{dx_3}{dx_2} & 1 & 0 & 0 \\ \frac{dx_4}{dx_1} & \frac{dx_4}{dx_2} & \frac{dx_4}{dx_3} & 1 & 0 \\ \frac{dx_5}{dx_1} & \frac{dx_5}{dx_2} & \frac{dx_5}{dx_3} & \frac{dx_5}{dx_4} & 1 \end{bmatrix} \tag{A.153}$$

Our seed vector is the scalar value 1 and is in the lower right corner of the Jacobian matrix in orange. In blue, we can see which derivatives the backward-mode AD determines. In red is $\boldsymbol{J}_{\boldsymbol{x}_{out}}$, thus the derivatives of the output function with respect to the input variables, which is at the same time the result of the backward-mode AD. As in the forward-mode AD, the result of the backward-mode AD is the intersection of the blue and the red circles. One can see that we can get the gradient of the output function $f = x_5$ with one backward-mode AD execution.

A.1.7 Direct and adjoint method

Suppose we want to solve the optimisation problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}. \end{aligned} \tag{A.154}$$

In non-mathematical terms, does this problem state that we want to minimise the objective function f while we need to satisfy $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$. The semicolon indicates that the equation $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x})$ is solved for $\boldsymbol{p}$ while $\boldsymbol{x}$ is fixed. Hence, $\boldsymbol{p}$ is a function of $\boldsymbol{x}$, i.e. $\boldsymbol{p}(\boldsymbol{x})$. The objective function could, for example, be the drag of a wing, which depends on the design variables $\boldsymbol{x}$ of the wing and the state variables $\boldsymbol{p}$ (e.g. displacement of the wing). In this case, $\boldsymbol{r}$ are the residual functions ensuring the model follows physical laws.

We can see $\boldsymbol{r} = \boldsymbol{0}$ as an equality constraint and use any constrained optimisation method described in subsection A.1.4 to solve this problem. However, we can do better than this. $\boldsymbol{r} = \boldsymbol{0}$ describes the relationship between the design and state variables, and the objective function is a function of both the design and state variables. The idea is to give the optimiser a derivative, which already considers the implicit relationship between the design and state variables. For this, we give the optimiser a derivative of f, which is along a linear approximation of $\boldsymbol{r} = \boldsymbol{0}$. We can do that by treating $\boldsymbol{p}$ as a function of $\boldsymbol{x}$ and define the optimisation problem as

$$\min_{\boldsymbol{x}} \quad f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) \tag{A.155}$$

where the residual function $\boldsymbol{r}$ describes $\boldsymbol{p}(\boldsymbol{x})$. The only disadvantage is that it is more complicated to calculate the derivative of the objective function. Using the chain rule, the total derivative of f is

$$\frac{d}{d\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = \frac{\partial f}{\partial \boldsymbol{p}} \frac{d\boldsymbol{p}}{d\boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{x}} \frac{d\boldsymbol{x}}{d\boldsymbol{x}} \tag{A.156}$$

$$= \frac{\partial f}{\partial \boldsymbol{p}} \frac{d\boldsymbol{p}}{d\boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{x}}, \tag{A.157}$$

which requires to know $d\boldsymbol{p}/d\boldsymbol{x}$. Figure A.4 shows the geometrical interpretation of equation (A.157). One can see that the derivative of $f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x}))$ is along the curve $\boldsymbol{r} = \boldsymbol{0}$.

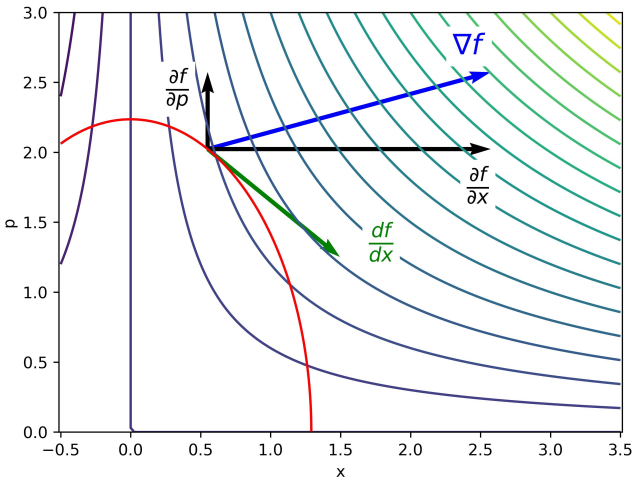


FIGURE A.4 – Contour lines of objective function $f(x, p) = x\,p$ and residual function $r = 3x^2 + p^2 - 5 = 0$. The plot shows the gradient of the objective function in blue and the total derivative of $f(x, p(x))$ in green.

Next, we can get $d\boldsymbol{p}/d\boldsymbol{x}$ from the total derivative of $\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x}))$ (please note that we defined that $\boldsymbol{p}$ is a function of $\boldsymbol{x}$) which is

$$\frac{d}{d\boldsymbol{x}}\boldsymbol{r}(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = \boldsymbol{0} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\frac{d\boldsymbol{p}}{d\boldsymbol{x}} + \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}\frac{d\boldsymbol{x}}{d\boldsymbol{x}} \tag{A.158}$$

$$= \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\frac{d\boldsymbol{p}}{d\boldsymbol{x}} + \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} \tag{A.159}$$

$$-\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\frac{d\boldsymbol{p}}{d\boldsymbol{x}}. \tag{A.160}$$

Please note that $d\boldsymbol{r}/d\boldsymbol{x}$ is equal to zero because $\boldsymbol{p}$ is a function of $\boldsymbol{x}$. To show this is left to the reader. However, a different way to see that $d\boldsymbol{r}/d\boldsymbol{x} = \boldsymbol{0}$ is to realise that $\boldsymbol{r}$ is zero everywhere. Thus a small perpetuation in $\boldsymbol{r}$ is zero, i.e. $d\boldsymbol{r} = \boldsymbol{0}$. Next, let us rearrange equation (A.160) to get $d\boldsymbol{p}/d\boldsymbol{x}$

$$\frac{d\boldsymbol{p}}{d\boldsymbol{x}} = -\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^{-1}\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}. \tag{A.161}$$

where we assumed that $\partial\boldsymbol{r}/\partial\boldsymbol{p}$ is invertible. Since we got an expression for $d\boldsymbol{p}/d\boldsymbol{x}$ we can plug it in equation (A.157) which gives

$$\frac{d}{d\boldsymbol{x}}f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = -\frac{\partial f}{\partial \boldsymbol{p}}\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^{-1}\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{x}}. \tag{A.162}$$

Please note that we only need to compute partial derivatives to get the total derivative of the objective function. However, it is computationally expensive to invert a matrix. There are two methods to avoid calculating the inverse of $\partial\boldsymbol{r}/\partial\boldsymbol{p}$, which are the *direct method* and the *adjoint method.*

A.1.7.1 Direct method

The direct method introduces a matrix $\boldsymbol{\Phi}$ which is defined to be

$$\boldsymbol{\Phi} \equiv \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^{-1} \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}. \tag{A.163}$$

Hence, we can determine $\boldsymbol{\Phi}$ by solving the linear system

$$\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}} \boldsymbol{\Phi} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}. \tag{A.164}$$

$\boldsymbol{\Phi}$ has the dimensions $n_p \times n_x$. We need to solve for every column of $\boldsymbol{\Phi}$ one linear system; thus, we must solve n_x linear systems. Consequently, the direct method is beneficial if only a few design variables $\boldsymbol{x}$ exist. The equation below visualises the dimensions of the matrices in case $n_x > n_p$.

$$\underset{(n_p \times n_p)}{\boxed{\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}}} \quad \underset{(n_p \times n_x)}{\boxed{\boldsymbol{\Phi}}} = \underset{(n_p \times n_x)}{\boxed{\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}}}$$

Next, let us plug in $\boldsymbol{\Phi}$ in equation (A.162) and we get

$$\frac{d}{d\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = -\frac{\partial f}{\partial \boldsymbol{p}} \underbrace{\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^{-1} \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}}_{\boldsymbol{\Phi}} + \frac{\partial f}{\partial \boldsymbol{x}} \tag{A.165}$$

$$= -\frac{\partial f}{\partial \boldsymbol{p}} \boldsymbol{\Phi} + \frac{\partial f}{\partial \boldsymbol{x}}. \tag{A.166}$$

The box below shows the steps on how to use the direct method.

Direct method steps

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0} \end{aligned}$$

1. Solve $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$ for $\boldsymbol{p}$
2. Solve linear system for $\boldsymbol{\Phi}$

$$\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}} \boldsymbol{\Phi} = \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}}$$

3. Calculate total derivative

$$\frac{d}{d\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = \frac{\partial f}{\partial \boldsymbol{p}} \boldsymbol{\Phi} + \frac{\partial f}{\partial \boldsymbol{x}}$$

4. Use the total derivative for any optimisation method to get the new design variables $\boldsymbol{x}$

A.1.7.2 Adjoint method

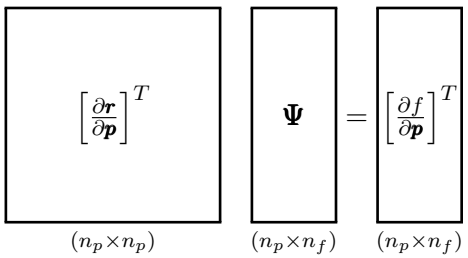
The adjoint method defines a matrix $\boldsymbol{\Psi}$ as

$$\boldsymbol{\Psi}^T \equiv \frac{\partial f}{\partial \boldsymbol{p}} \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^{-1}. \tag{A.167}$$

So we can calculate $\boldsymbol{\Psi}$ by solving the linear system

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial f}{\partial \boldsymbol{p}}\right]^T . \tag{A.168}$$

Please note that $\boldsymbol{\Psi}$ has the dimensions $n_p \times n_f$. Again, we need to solve for every column of $\boldsymbol{\Psi}$ one linear system. In case there are only a few functions of interest, as is usually the case in optimisation, the adjoint method is beneficial. The equation below shows the dimensions of the matrices in case $n_p > n_f$.



Similar to the direct method, we can plug in $\boldsymbol{\Psi}$ in equation (A.162).

$$\frac{d}{d\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = -\underbrace{\frac{\partial f}{\partial \boldsymbol{p}} \left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^{-1}}_{\boldsymbol{\Psi}^T} \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{x}} \tag{A.169}$$

$$= -\boldsymbol{\Psi}^T \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{x}} \tag{A.170}$$

The box below shows the steps of the adjoint method.

Adjoint method steps

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}, \boldsymbol{p}) \\ \text{w.s.} \quad & \boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0} \end{aligned}$$

1. Solve $\boldsymbol{r}(\boldsymbol{p}; \boldsymbol{x}) = \boldsymbol{0}$ for $\boldsymbol{p}$
2. Solve linear system for $\boldsymbol{\Psi}$

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}}\right]^T \boldsymbol{\Psi} = \left[\frac{\partial f}{\partial \boldsymbol{p}}\right]^T$$

3. Calculate total derivative

$$\frac{d}{d\boldsymbol{x}} f(\boldsymbol{x}, \boldsymbol{p}(\boldsymbol{x})) = -\boldsymbol{\Psi}^T \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} + \frac{\partial f}{\partial \boldsymbol{x}}$$

4. Use the total derivative for any optimisation method to get the new design variables $\boldsymbol{x}$

A.1.8 Introduction to B-splines

Basic splines, or in short B-splines, are built from piecewise polynomial functions, also called *basis functions*. B-splines are widely used in computer graphics, computer-aided design (CAD) and other numerical analysis. This section explains the fundamental principles of the splines and explores how they are employed in optimisation processes.

Suppose we want to minimise the weight of a beam loaded by a vertical force as shown in figure A.5. The design variable is the beam's radius, and we require that the beam does not fail.

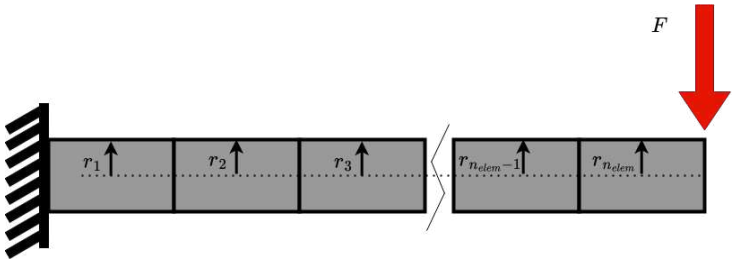


FIGURE A.5 – Beam subjected to vertical force, an example where B-splines are useful in optimising. B-splines could describe the radius of the beam.

We can divide the beam into finite parts and optimise of every section the radius as indicated in figure A.5. It is evident that the more beams we use for modelling, the closer we get to the optimal solution. The disadvantage of increasing the number of beams in the model is that we increase the number of design variables, which increases the optimisation cost. We introduce B-splines in the optimisation process to decrease the number of design variables. Instead of defining the thickness of the beam at finite points, we use a B-spline to describe the thickness as a function of the length of the beam. The design variables become the weights of each basic function of the B-spline. The weights determine the contribution of each basic function to the B-spline. Mathematically, we can describe the B-spline as

$$f_B = \sum_{i}^{n_B} B_{i,p}(\xi)\, c_i, \tag{A.171}$$

where $B_{i,p}$ is the i^{th} basic function of degree p, n_B is the number of basic functions and c_i is the weight of the i^{th} basic function. All weights can be collected in the weight vector $\boldsymbol{c}$. To create a basic function, one must choose a degree p of the basis function and a knot vector $\boldsymbol{\Xi}$. The knot vector contains a series of non-decreasing values of parametric coordinates called *knots*. The knot vector reads

$$\boldsymbol{\Xi} = \begin{bmatrix} \xi_0, & \xi_1, & \xi_i, & \dots & \xi_{n_{\boldsymbol{\Xi}}} \end{bmatrix}, \tag{A.172}$$

where $\xi_i \leq \xi_{i+1}$ and $n_{\boldsymbol{\Xi}}$ is the total number of knots. Two neighbouring knots span an interval, which is known as a *knot span*. The i^{th} knot span goes from ξ_i up to and not including ξ_{i+1}, or mathematically $[\xi_i, \xi_{i+1})$. Given the degree p and the knot vector $\boldsymbol{\Xi}$ the B-splines can be determined by the Cox–de Boor recursion formula, which reads for $p = 0$

$$B_{i,0}(\xi) = \begin{cases} 1 & \text{if } \xi_i \leq \xi < \xi_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \tag{A.173}$$

and for $p > 0$

$$B_{i,p}(\xi) = \frac{\xi - \xi_i}{\xi_{i+p} - \xi_i} B_{i,p-1}(\xi) + \frac{\xi_{i+p+1} - \xi}{\xi_{i+p+1} - \xi_{i+1}} B_{i+1,p-1}(\xi). \tag{A.174}$$

Please note that the higher-order basic functions ($p > 0$) are defined based on the lower-order basis functions. What is interesting to observe is that we will get $n_B = n_{\boldsymbol{\Xi}} - p - 1$ basic functions. The B-splines have two essential properties. They are always positive, and the sum of all basis functions together is one on the parameter space $[a, b]$.

$$B_{i,p}(\xi) \geq 0, \;\; \sum_{i=0}^{n_B} B_{i,p}(\xi) = 1, \;\; \forall \zeta \in [a, b] \tag{A.175}$$

For simplicity, we will choose the parameter space $[0, 1]$, which we can always be achieved by

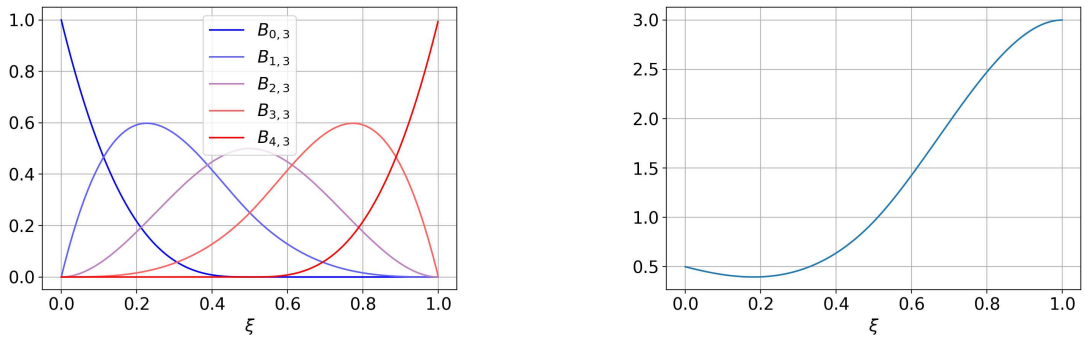
scaling. Furthermore, we only consider B-splines with knot vectors where the first and the last vector element is $p+1$ times repeated. Those knot vectors are called *open knot vectors* and have the advantage that the B-splines are clamped at a and b. Clamped means that the B-spline will have the value of the first weight c_0 at a and the value of the last weight c_{n_B} at b. To give an impression how the B-splines look, figure A.6a shows the basic functions for the for $p = 3$ and the knot vector

$$\boldsymbol{\Xi} = \begin{bmatrix} 0 & 0, & 0, & 0, & 1/2, & 1, & 1, & 1, & 1 \end{bmatrix} \tag{A.176}$$

Next, if we choose the weight vector

$$\boldsymbol{c}_B = \begin{bmatrix} 1/2, & 1/3, & 1/4, & 3, & 3 \end{bmatrix} \tag{A.177}$$

we get the B-spline shown in figure A.6b. The B-spline is clamped, as we can see by the fact that its first and final values are the values of the first and final weight. The B-spline is adjustable by changing the weights, making the weights excellent design variables in an optimisation process (BEER *et al.*, 2020).



(a) Basis functions of third degree

(b) B-spline

FIGURE A.6 – Basis functions and B-spline with knot vector $\boldsymbol{\Xi} = \begin{bmatrix} 0 & 0, & 0, & 0, & 1/2, & 1, & 1, & 1, & 1 \end{bmatrix}$ and with weight vector $\boldsymbol{c} = \begin{bmatrix} 1/2, & 1/3, & 1/4, & 3, & 3 \end{bmatrix}$.

In QASTRO, the user provides the initial design parameters (e.g. the radius at finite points of a beam), which is the baseline. The user has two options; the B-splines can represent either 'additive' or 'multiplicative' changes to the initial design parameters, which correspond to relative changes rather than their absolute values. When the 'multiplicative' option is chosen, the B-spline is initially a flat line, which means that all weights are equal to one. As the optimiser runs, it modifies the B-spline weights. The B-spline becomes evaluated at finite points where the design parameters are defined, and those values are the factors which increase or decrease the baseline geometry. On the other hand, when the 'additive' option is selected, the B-spline is initially a flat line, which means that all weights are zero. As optimisation proceeds, the B-spline is modified, adding or subtracting values to the baseline geometry. In this Master's Thesis, we use only the 'multiplicative' option.

A.2 Aerodynamics

The methodology chapter describes a finite-wing model based on Prandlt's finite-wing model. It is essential to be familiar with the finite wing model of Prandlt to understand the methodology in this Master's Thesis. Both finite wing models are based on the potential flow theory described in the first subsection. The second subsubtion explains Prandlt's finite-wing model.

A.2.1 Introduction to potential flow

The potential flow theory is an aerodynamics concept that describes fluid behaviour in a flow. The theory makes the important assumption that the fluid is inviscid, incompressible and irrotational. The flow velocity at any point in the flow field can be determined by a potential function $\Phi(\boldsymbol{r})$, which is a function of the location $\boldsymbol{r} = \Big(x, y, z\Big)^T$. $\Phi(\boldsymbol{r})$ is chosen such that its gradient gives us the velocity field. Thus,

$$\nabla\Phi(\boldsymbol{r}) = \boldsymbol{V}(\boldsymbol{r}), \tag{A.178}$$

where $\boldsymbol{V}(\boldsymbol{r})$ is the flow velocity at the location $\boldsymbol{r}$. Because we assume that the flow is incompressible, we know that

$$\nabla \cdot \boldsymbol{V}(\boldsymbol{r}) = 0, \tag{A.179}$$

which means that

$$\nabla \cdot \nabla\Phi(\boldsymbol{r}) = 0 \tag{A.180}$$

$$\nabla^2\Phi(\boldsymbol{r}) = 0. \tag{A.181}$$

Consequently, we need to solve the Laplace equation $\nabla^2\Phi = 0$ for the scalar function $\Phi(\boldsymbol{r})$, which will give us the flow's velocity field. We have two boundary conditions to solve the Laplace equation:

1. Zero flow condition: There is no airflow through a solid object in a flow. In our case, the object in the flow will be a wing. Hence, we will define that there is no flow normal to the wing surface. Mathematically, this boundary condition reads

$$\nabla\Phi(\boldsymbol{r}) \cdot \boldsymbol{n} = \boldsymbol{0} \tag{A.182}$$

2. Decay of disturbance: The object, or in our case the wing, disturbs the flow. This disturbance should decay far away from the object. That means that

$$\lim_{r\to\infty} (\nabla\Phi - \boldsymbol{V}_{rel}) = \boldsymbol{0}, \tag{A.183}$$

where $\boldsymbol{V}_{rel}$ is the relative velocity between the undisturbed fluid and the object, and r is the distance to the object.

The challenge is to find the scalar function $\Phi(\boldsymbol{r})$. Luckily, we can use the principle of superposition for the Laplace equation, which states that if $\Phi_1, \Phi_2, \dots \Phi_{n_\Phi}$ are solutions to the Laplace equation, then

$$\Phi = \sum_{k=1}^{n_\Phi} c_k \Phi_k \tag{A.184}$$

is a solution to the Laplace equation, too. c_k is an arbitrary constant.

The procedure to find $\Phi(\boldsymbol{r})$ is to combine multiple elementary solutions (e.g. sources, doublets, vortexes), which satisfy the Laplace equation such that the zero flow boundary condition is satisfied. Those elementary solutions mostly satisfy the decay of disturbance, which is the second boundary condition.

A.2.1.1 Circulation free elementary solutions

Polynomials

The most straightforward solutions to the Laplace equation are polynomials. For example, we can describe the free stream velocity by the potential function

$$\Phi = Ax + By + Cz \tag{A.185}$$

which gives the velocity field

$$\boldsymbol{V} = \begin{Bmatrix} A \\ B \\ C \end{Bmatrix} = \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}. \tag{A.186}$$

Also, different polynomial potential functions can be used to describe a flow field.

Point source/sink

A point source/ sink located at point $\boldsymbol{r}_0$ has the potential function

$$\Phi(\boldsymbol{r}) = -\frac{\sigma}{4\,\pi|\boldsymbol{r} - \boldsymbol{r}_0|}, \tag{A.187}$$

where σ is the strength of the source/ sink. If σ is positive, we call it a source; if σ is negative, it's called a sink. The velocity due to the point source/ sink is

$$\boldsymbol{V} = \frac{\sigma|\boldsymbol{r} - \boldsymbol{r}_0|}{4\,\pi|\boldsymbol{r} - \boldsymbol{r}_0|^3}. \tag{A.188}$$

Please note that at $\boldsymbol{r}_0$, the conservation of mass is violated, and we need to exclude the point $\boldsymbol{r}_0$ of the solution.

Point doublet

Let us consider a point source and a point sink of strength σ with opposite sign. The vector $\boldsymbol{e}_l$ with a length l connects the sink and the source. Suppose we let l go to zero ($l \to 0$), we let σ go to infinity ($\sigma \to \infty$) but remain the product of the distance l and the strength σ constant ($\sigma l \to \mu$). In this case, we get a point doublet with a potential function

$$\Phi = -\frac{\mu\,\boldsymbol{e}_l \cdot \boldsymbol{r}}{4\,\pi\,|\boldsymbol{r} - \boldsymbol{r}_0|^3}. \tag{A.189}$$

The doublet has a directional component $\boldsymbol{e}_l$. If we orientate $\boldsymbol{e}_l$ along the x-,y- or z-axis we get the potential functions

$$\Phi_{\boldsymbol{e}_x}(x,y,z) = -\frac{\mu}{4\,\pi}(x-x_0)\left[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2\right]^{-3/2} \tag{A.190}$$

$$\Phi_{\boldsymbol{e}_y}(x,y,z) = -\frac{\mu}{4\,\pi}(y-x_0)\left[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2\right]^{-3/2} \tag{A.191}$$

$$\Phi_{\boldsymbol{e}_z}(x,y,z) = -\frac{\mu}{4\,\pi}(z-x_0)\left[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2\right]^{-3/2}. \tag{A.192}$$

The velocity field $\boldsymbol{V} = \Big(u,v,w\Big)^T$ of a point doublet orientated along the x-axis is

$$u = -\frac{\mu}{4\,\pi}\,\frac{(y-y_0)^2+(z-z_0)^2-2(x-x_0)^2}{\left[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2\right]^{5/2}} \tag{A.193}$$

$$v = -\frac{3\mu}{4\,\pi}\,\frac{(x-x_0)(y-y_0)}{\left[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2\right]^{5/2}} \tag{A.194}$$

$$v = -\frac{3\mu}{4\,\pi}\,\frac{(x-x_0)(z-z_0)}{\left[(x-x_0)^2+(y-y_0)^2+(z-z_0)^2\right]^{5/2}}. \tag{A.195}$$

The velocity field of $\Phi_{\boldsymbol{e}_y}$ and $\Phi_{\boldsymbol{e}_z}$ are left to the reader.

A.2.1.2 Circulation elementary solutions

Understanding the concept of vorticity and circulation is important before we move on to other elementary solutions to the Laplace equation. Let us consider a particle in a flow. The *vorticity* ζ is twice the angular velocity of the particle in the flow. Thus,

$$\boldsymbol{\zeta} \equiv 2\,\boldsymbol{\omega} = \nabla \times \boldsymbol{V}. \tag{A.196}$$

The *circulation* is the line integral of the tangential velocity component about a closed curve fixed in the flow. Thus,

$$\Gamma \equiv \oint_c \boldsymbol{V}\cdot d\boldsymbol{l}. \tag{A.197}$$

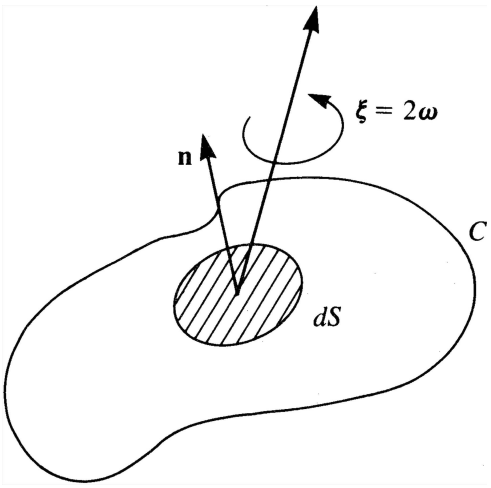


FIGURE A.7 – Relationship between vorticity and circulation by Stokes' Theorem (KATZ; PLOTKIN, 2001).

With Stokes' Theorem, we can write the circulation in terms of the vorticity. Let us consider an open surface S with a closed curve C on its boundary. The circulation of the enclosed surface S is given by

$$\Gamma \equiv \oint_c \boldsymbol{V} \cdot d\boldsymbol{l} = \int_S \boldsymbol{\zeta} \cdot \boldsymbol{n}\, dS = \int_S \nabla \times \boldsymbol{V} \cdot \boldsymbol{n}\, dS, \tag{A.198}$$

where $\boldsymbol{n}$ is normal to the surface S and $2\boldsymbol{\omega}$ is the vorticity vector. Hence, the circulation around a closed contour is the sum of the enclosed vorticity. Consequently, we need vorticity in the flow to get a circulation in the flow because equation (A.198) shows that if $\boldsymbol{\zeta} = \boldsymbol{0}$, then $\Gamma = 0$. With the aid of the concept of vorticity and circulation, we can describe flow fields with only tangential velocity components. We will see that those elements are essential to describe the flow around wings and to generate the lift. However, there is a problem because the potential flow needs to be irrotational. Irrotational means that the flow particles can not spin around their own axis or, in other words, that the angular velocity is zero, which implies that $\boldsymbol{\zeta} = \boldsymbol{0}$. The potential flow needs to be irrotational because it is irrotational by definition:

$$\zeta = \nabla \times \boldsymbol{V} = \nabla \times (\nabla \Phi) = \boldsymbol{0} \tag{A.199}$$

$\nabla \times (\nabla \Phi)$ is a vector identity and is always zero. Thus, we need to define elementary solutions for the potential flow where we exclude the rotational part of the solution. By excluding the rotational part of the potential flow solution, we can have a circulation in the potential flow solution. Circulation means that the fluid particles can have a rotational motion around a closed path, e.g. around an airfoil (ANDERSON, 2017).

General vortex element (3D)

The general vortex element in three dimensions is given by the Biot & Savart law. The Biot & Savart law does not give a potential function but directly the flow field. Let us consider an infinitesimal piece of the vorticity filament $\boldsymbol{\zeta}$, shown in figure A.8. The Biot & Savart law states that the velocity field induced by the vorticity filament is given by

$$\boldsymbol{V}(\boldsymbol{r}) = \frac{\Gamma}{4\,\pi} \int \frac{d\boldsymbol{l} \times \boldsymbol{r}}{|\boldsymbol{r}|^3} \tag{A.200}$$

where $d\boldsymbol{l}$ is a vector along the vorticity filament and $\boldsymbol{r} = \boldsymbol{r}_0 - \boldsymbol{r}_1$. $\boldsymbol{r}_1$ is a point on the vorticity filament. The vorticity vector $\boldsymbol{\zeta}$ is located on the vortex segment dS. The circulation of the vortex segment is Γ, which is according to Stokes' Therom $\Gamma = \int_S \boldsymbol{\zeta} \cdot \boldsymbol{n}\, dS$.

Straight vortex segment

We want to determine the induced velocity in point P by a straight vortex segment with the circulation Γ according to the Biot & Savart law. Figure A.9a shows the straight vortex segment. A vortex segment can not start or end in a fluid. However, we will determine the induced velocity in point P by the vortex segment, which starts at point 1 and ends at point 2. The distance to the point P is $\boldsymbol{r}_0 - \boldsymbol{r}_1 = \boldsymbol{r}$.

The induced velocity by the vortex segment $1 \rightarrow 2$ is given by the differential form of the Biot & Savart law, which is

$$\Delta \boldsymbol{V} = \frac{\Gamma}{4\,\pi} \frac{d\boldsymbol{l} \times \boldsymbol{r}}{r^3}. \tag{A.201}$$

We can write the differential version of the Biot & Savart law in a scalar form

$$\Delta V_\theta = \frac{\Gamma}{4\pi} \frac{\sin\beta}{r^2}\, dl \tag{A.202}$$

$$= \frac{\Gamma}{4\pi d} \sin\beta\, d\beta \tag{A.203}$$

because we consider a straight vortex segment, which will only induce a tangential velocity. Next, we can integrate along the vortex segment

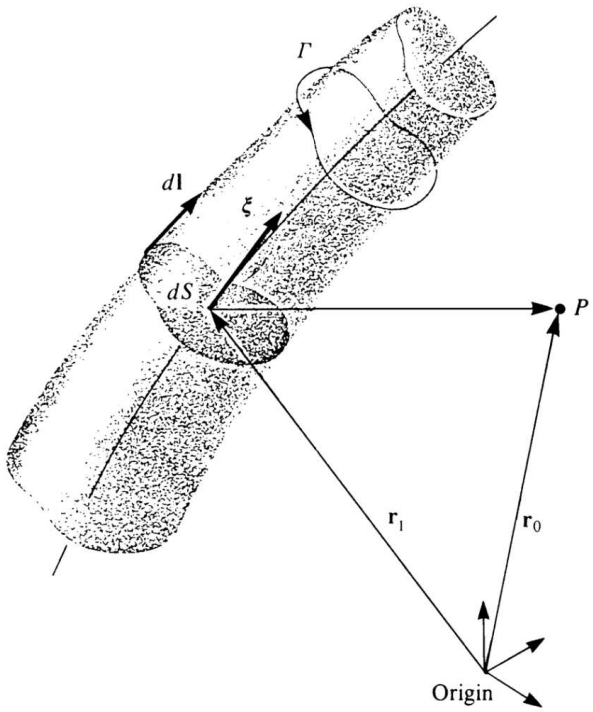
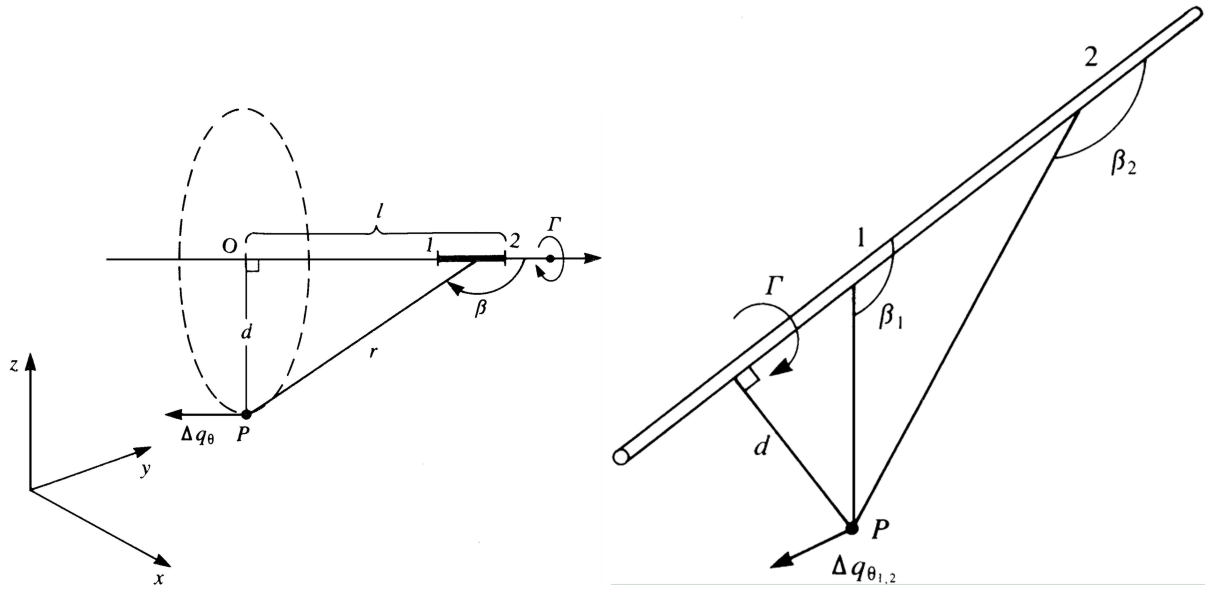


FIGURE A.8 – Vorticity filament of Biot & Savart law. The Biot & Savart law determines the velocity induced by the vortex segment dS in point P (KATZ; PLOTKIN, 2001).



(a) Induced velocity in point P by straight vortex segment

(b) Angles of straight vortex segment

FIGURE A.9 – From Katz and Plotkin (2001).

$$V_\theta = \frac{\Gamma}{4\pi d}\int_{\beta_1}^{\beta_2} \sin\beta\, d\beta = \frac{\Gamma}{4\pi d}(\cos\beta_1 - \cos\beta_2) \tag{A.204}$$

to get the induced velocity by the vortex segment in point P. Figure A.9b shows the angles β_1 and β_2. However, we can also define the velocity vector $\boldsymbol{V}$ in terms of $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ by

$$\boldsymbol{V} = \frac{\Gamma}{4\pi}\frac{\boldsymbol{r}_1 \times \boldsymbol{r}_2}{|\boldsymbol{r}_1 \times \boldsymbol{r}_2|}\boldsymbol{r}_0 \cdot \left(\frac{\boldsymbol{r}_1}{r_{\ 1}} - \frac{\boldsymbol{r}_2}{r_2}\right), \tag{A.205}$$

where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the vectors from point 1 and 2 to point P, respectively.

If we choose $\beta_1 = \pi/2$ and $\beta_2 = \pi$ we get the induced velocity

$$U_\theta = \frac{\Gamma}{4\pi d} \tag{A.206}$$

in point P by a semi-infinite vortex line starting at point O.

An infinite vortex segment has the angles $\beta_1 = 0$ and $\beta_2 = \pi$ which gives the induced velocity

$$U_\theta = \frac{\Gamma}{2\pi d}. \tag{A.207}$$

The infinite vortex segment can be seen as a two-dimensional vortex as shown in figure A.10.

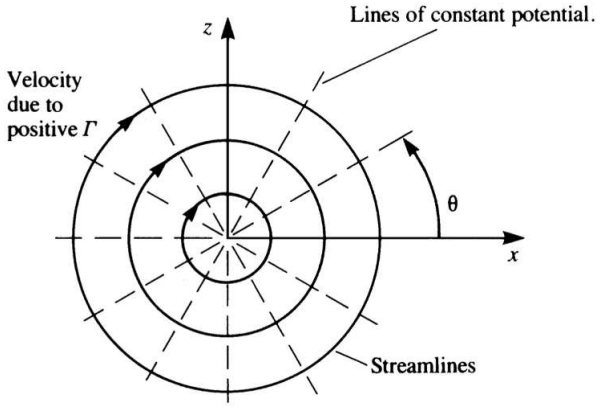


FIGURE A.10 – Infinite vortex segment, which is a two-dimensional vortex segment. Figure shows the streamlines in two dimensions, where the vortex is located at the origin (KATZ; PLOTKIN, 2001).

For the two-dimensional vortex, the potential function is

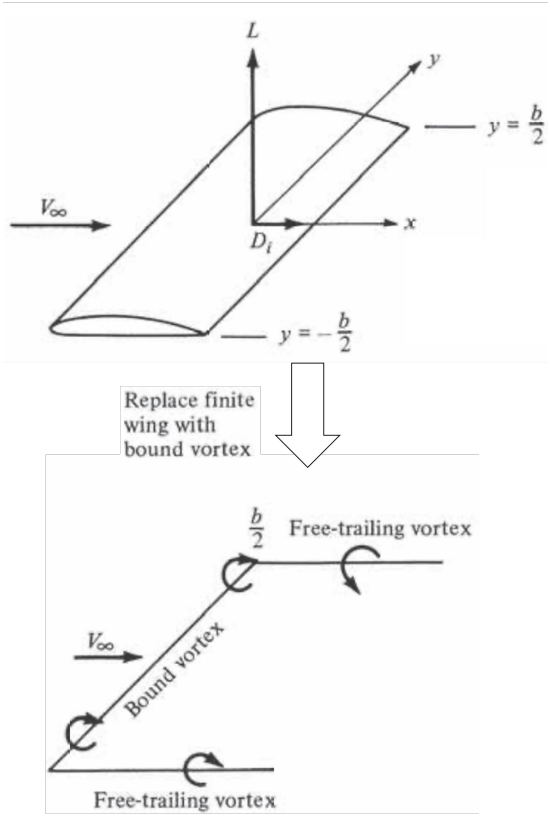
$$\Phi = -\frac{\Gamma}{2\pi} \tan^{-1}\left(\frac{z - z_0}{x - x_0}\right) \tag{A.208}$$

where the vortex is located at $\boldsymbol{r}_0 = \begin{Bmatrix} x_0 & z_0 \end{Bmatrix}^T$. Please note that inside the vortex filament is vorticity, which means that we need to exclude the filament from the solution because the potential flow must remain irrotational.

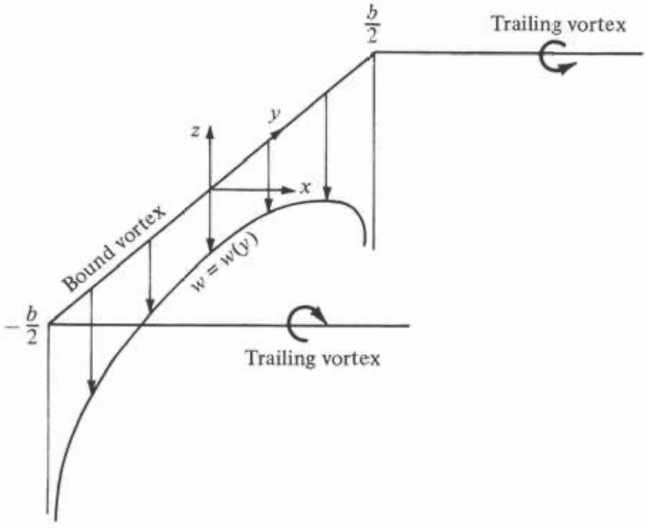
A.2.2 Prandtl's classical lifting-line theory

Prandtl developed a lifting-line theory to model a finite wing. He knew that circulation in the flow was essential to generate lift; thus, he chose to use straight vortex segments to model the finite wing. However, Helmholtz's theorem states that a vortex filament cannot start nor end in a fluid, which makes it difficult to model a finite wing. Prandtl's idea was to create a "Horseshoe vortex", which consists of a straight vortex segment along the span b of the wing. At the wing tips, at $b/2$ and $-b/2$, the vortex segment does not end but continues in the wake of the wing. The straight trailing vortex segments continue until infinity.

Figure A.11a shows the basic idea of Prandtl. The aim is to find the vortex strength Γ of the horseshoe, which describes a wing with span width b and chord width c. This model needs to satisfy two conditions. The Kutta condition states that the vorticity component parallel to the trailing edge needs to be zero and the zero flow through solid bodies condition of equation A.182. One can show that we satisfy the Kutta condition if we place the vortex line at the wing's quarter-chord line. The zero flow condition is more challenging to achieve because the vortex



(a) Prandlt's finite wing model with a single horseshoe. Adapted from Anderson (2017)



(b) Downwash of Prandlt's wing model with single horseshoe (ANDERSON, 2017)

FIGURE A.11 – Basic idea of Prandtl's finite wing model with single horseshoe.

line is placed inside the wing. However, we can define a control point on the airfoil with a zero flow condition. Prandtl put this control a half chord length behind the vortex line, effectively at a three-quarter line of the wing. Figure A.12 shows the zero flow boundary condition of Prandtl's finite wing model. Mathematically, we can write the boundary condition as

$$w_{wing} + w_{wake} + V_\infty \alpha = 0, \tag{A.209}$$

where w_{wing} is the velocity induced by the straight vortex segment along the span inside the wing and w_{wake} is the induced velocity by the wake vortex segments along the x-axis, $V_\infty = \|\boldsymbol{V}_\infty\|$ and α is the geometric angle of attack.

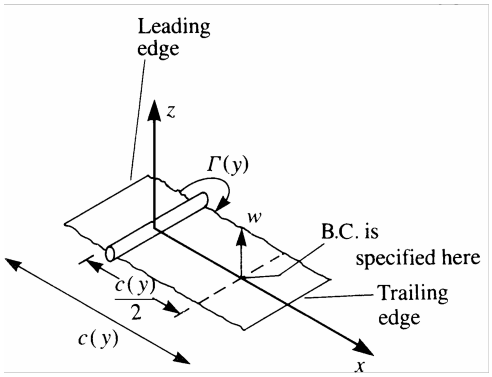


FIGURE A.12 – Boundary condition for Prandlt's finite wing model from Katz and Plotkin (2001).

The finite wing is attached to the origin, and the free stream velocity $\boldsymbol{V}_\infty$ is in the positive

x-axis. The trailing vortex segments create a so-called downwash flow in the x-y plane. The downwash velocity of a semi-infinite vortex line is given by (A.206). The distance d in (A.206) for the vortex line at $y = b/2$ is $d_r = b/2 - y$. For the vortex line at $y = -b/2$, the distance is $d_l = b/2 + y$. If we plug those distances in (A.206) and add both induced velocities of the right and the left vortex line together, we get the total induced downwash velocity

$$w(y) = -\frac{\Gamma}{4\pi d_l} - \frac{\Gamma}{4\pi d_r} = -\frac{\Gamma}{4\pi(b/2+y)} - \frac{\Gamma}{4\pi(b/2-y)} = -\frac{\Gamma}{4\pi}\frac{b}{(b/2)^2 - y^2} \tag{A.210}$$

in the negative z-direction. The negative sign in (A.210) ensures that the downwash velocity is positive in the negative z-direction. Figure A.11b shows the resulting downwash due to the trailing vortex segments. One can see that this model has the problem that the downwash velocity at the wing tips goes to infinity. The model gives little flexibility to change this behaviour, and here is where the idea of Prandtl's lifting-line starts.

We can not change the vorticity along the horseshoe because this violates Helmholtz's theorem, which states that the strength of a vortex line is constant along its length. However, we can include multiple horseshoes with each a different vorticity strength $d\Gamma$. Figure A.13a shows a finite wing model with in total three vortex horseshoes. The first vortex horseshoe goes from point A to point F along the span width b of the wing, similar to the model with a single horseshoe in figure A.11a. The second horseshoe goes from point B to point E and the third goes from point C to point D. The vortex segments along the wing span coincide with each other. The vortex strength of all vortex lines coinciding with each other becomes summed up. The vortex strength $d\Gamma_1$ of the first element is along the whole wing span. Along the second vortex line is a total vortex strength of $d\Gamma_1 + d\Gamma_2$. In the middle is the vortex strength of all vortex horseshoes together: $d\Gamma_1 + d\Gamma_2 + d\Gamma_3$. The vortex segments in the wake of all horseshoes are parallel. Please note that the vortex strength of the vortex segments in the wake is always equal to the change in vortex strength along the span. For example, the vortex strength of the wake vortex line at point E is the jump in the vortex strength along the span at point E. The vortex jump at point E is from $d\Gamma_1$ to $d\Gamma_1 + d\Gamma_2$ which is $d\Gamma_2$. Thus, the strength of the wake vortex line at point E is $d\Gamma_2$.

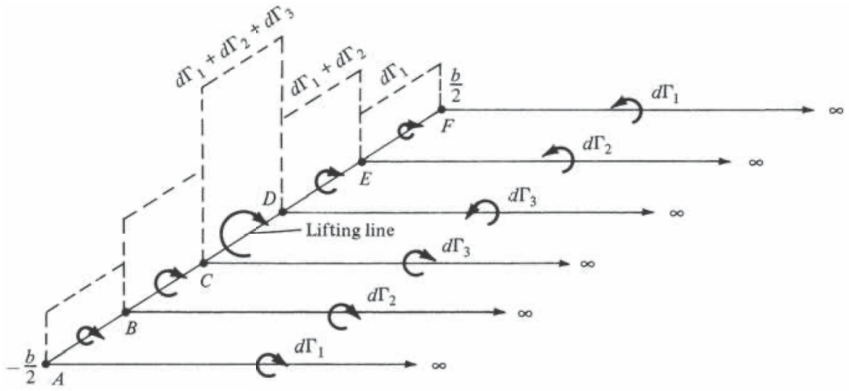
By orientating the horseshoes in the manner as shown in figure A.13a, we create a vortex line along the wing span, which varies without violating Helmholtz's theorem. We do not violate Helmholtz's theorem because we shed the local circulation change along the wing span into the wake. Now, let us increase the number of vortex shoes to infinity. In this case, we create a *continuous varying vortex strength* $\Gamma(y)$ along the wing span and a *continuous vortex sheet* in the x-y plane. Figure A.13b visualises infinity horseshoes along the wing span. The vortex line along the wing span is called the *lifting-line*. Let us consider an infinitesimal small segment dy on the y-axis. At this location, the vortex strength is $\Gamma(y)$, and the change in vorticity along dy is $d\Gamma(y) = (d\Gamma(y)/dy)dy$. Consequently, the wake vortex line segment at point y has the strength $d\Gamma(y) = (d\Gamma(y)/dy)dy$. The induced downwash at point y_0 of the wake vortex line at y is

$$dw(y_0) = -\frac{(d\Gamma(y)/dy)dy}{4\pi(y_0 - y)}. \tag{A.211}$$

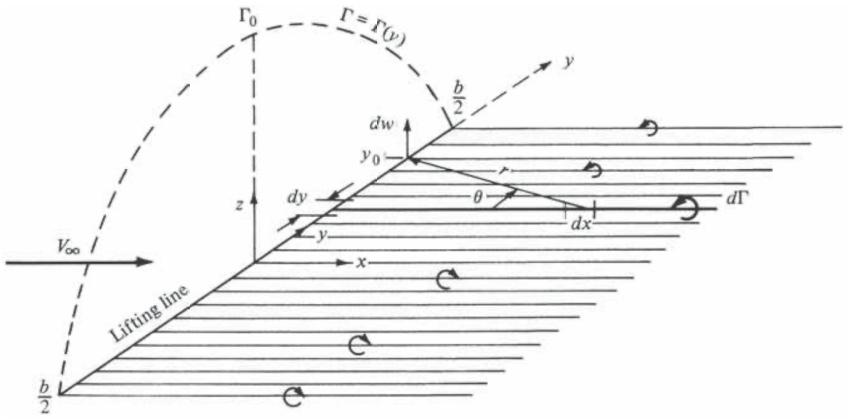
Please note that $dw(y_0)$ is constant along the x-axis and positive upward. The induced velocity of all wake vortex lines is given by integrating over the span of the wing.

$$w(y_0) = w_{wake} = -\frac{1}{4\pi}\int_{-b/2}^{b/2} \frac{(d\Gamma(y)/dy)dy}{4\pi(y_0 - y)} \tag{A.212}$$

Equation (A.212) gives the downwash w_{wake} of the vortex sheet in the x-y plane, thus through our control points. Next, we must calculate the induced downwash w_{wing} by the straight vortex line along the wing span. Let us consider a small piece of the lifting-line. This small piece



(a) Prandlt's finite wing model with multiple horseshoes



(b) Prandlt's finite wing model with infinite horseshoes

FIGURE A.13 – From Anderson (2017).

creates a downwash

$$\Delta w_{wing}(d) = -\frac{\Delta\Gamma}{4\pi d}(\cos\beta_1 - \cos\beta_2) \tag{A.213}$$

$$\Delta w_{wing}(c(y)) = -\frac{\Delta\Gamma}{4\pi c(y)/2}\left[\frac{y+y_0}{\sqrt{(c/2)^2+(y+y_0)^2}} + \frac{y_0-y}{\sqrt{(c/2)^2+(y_0-y)^2}}\right] \tag{A.214}$$

$$\approx -\frac{\Delta\Gamma}{\pi c(y)} \tag{A.215}$$

according to equation (A.204) and if the aspect ratio is large enough, we can neglect $(c/2)^2$. Note that w_{wing} is defined in a positive z-direction. Thus, the downwash is negative due to the lifting-line. All small pieces of the lifting-line together create a downwash

$$w_{wing}(y) = -\frac{\Gamma}{\pi c(y)}. \tag{A.216}$$

Next, we can plug the downwash velocity w_{wing} due to the wing (equation (A.212)) and the downwash velocity w_{wake} due to the wake (equation (A.216)) in the zero flow equation (A.209) and divide it by the free stream velocity V_∞.

$$w_{wing} + w_{wake} + V_\infty \alpha = 0 \tag{A.217}$$

$$-\frac{\Gamma}{\pi c(y)} - \frac{1}{4\pi}\int_{-b/2}^{b/2} \frac{(d\Gamma(y)/dy)dy}{4\pi(y_0 - y)} + V_\infty \alpha = 0 \tag{A.218}$$

$$-\frac{\Gamma}{\pi c(y) V_\infty} - \frac{1}{4\pi V_\infty}\int_{-b/2}^{b/2} \frac{(d\Gamma(y)/dy)dy}{4\pi(y_0 - y)} + \alpha = 0 \tag{A.219}$$

Equation A.217 is Prandtl's lifting-line integrodifferential equation and is only a function of the unknown vortex distribution $\Gamma(y)$. The equation can be solved with the boundary condition that the pressure difference at the wing tips must be zero.

$$\Gamma(y = \pm b/2) = 0 \tag{A.220}$$

However, from an engineering perspective, it is worth rearranging equation (A.217). The integrodifferential equation can be seen as a combination of the geometric angle of attack α, the induced angle of attack α_i and the effective angle of attack α_{eff} shown in figure A.14.

$$\underbrace{-\frac{\Gamma}{\pi c(y) V_\infty}}_{\alpha_{eff}} \underbrace{-\frac{1}{4\pi V_\infty}\int_{-b/2}^{b/2} \frac{(d\Gamma(y)/dy)dy}{4\pi(y_0 - y)}}_{\alpha_i} + \alpha = 0 \tag{A.221}$$

$$-\alpha_{eff} - \alpha_i + \alpha = 0 \tag{A.222}$$

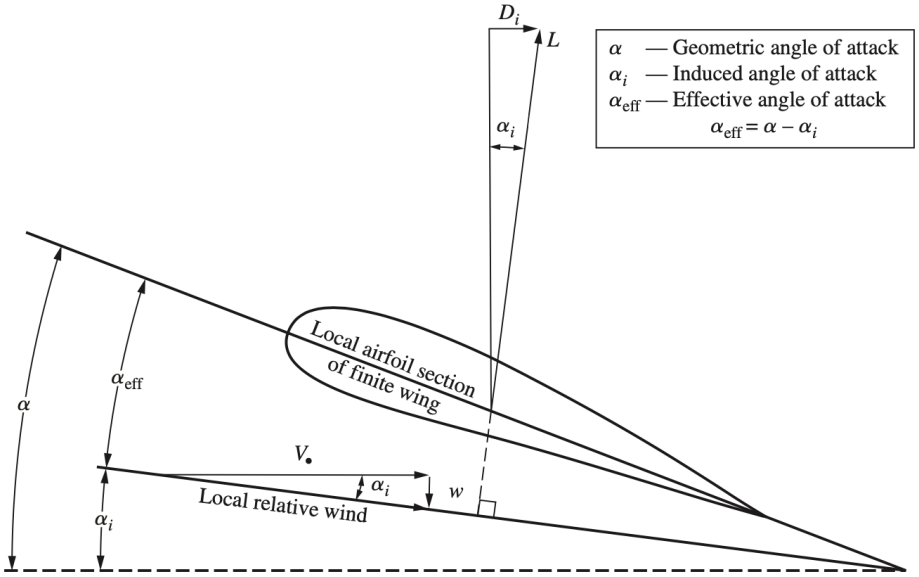


FIGURE A.14 – Effect of downwash on the local flow over a local airfoil section of a finite wing (ANDERSON, 2017).

The downwash reduces the geometric angle of attack α to the effective angle of attack α_{eff}. To generate lift is not for "free"; you need to "pay" it by the induced angle of attack α_i.

What is if the wing has a twist angle $\alpha_{twist}(y)$ along the chord? In this case, the geometric angle α also becomes a function of y. Thus, it might be better to rewrite equation (A.217) in terms of the twist angle $\alpha_{twist}(y)$. The geometric angle $\alpha(y)$ is the twist angle $\alpha_{twist}(y)$ minus the zero-lift angle $\alpha_{L=0}(y)$. Therefore, it is convenient to rewrite the integrodifferential equation in terms of the twist angle $\alpha_{twist}(y)$ and the zero-lift-angle $\alpha_{L=0}(y)$.

$$\boxed{-\frac{\Gamma}{\pi c(y) V_\infty} - \frac{1}{4\pi V_\infty} \int_{-b/2}^{b/2} \frac{(d\Gamma(y)/dy)dy}{4\pi(y_0 - y)} + \alpha_{twist}(y) - \alpha_{L=0}(y) = 0} \tag{A.223}$$

A.2.2.1 Aerodynamic loads

This section will compute the aerodynamic loads acting on a wing. The Kutta-Joukowski theorem states that the lift at y_0 of an airfoil section is

$$L'(y_0) = \rho_\infty V_\infty \Gamma(y_0). \tag{A.224}$$

Once the vortex distribution $\Gamma(y)$ of the lifting-line is known, we can compute the total lift of a wing by integrating equation (A.224) over the wing.

$$L = \int_{-b/2}^{b/2} L'(y)\, dy = \int_{-b/2}^{b/2} \rho_\infty V_\infty \Gamma(y_0)\, dy \tag{A.225}$$

The lift coefficient is defined as

$$C_L = \frac{L}{\frac{1}{2}\rho_\infty V_\infty S}, \tag{A.226}$$

where $S = b\,c$, which means that the lift coefficient in terms of $\Gamma(y)$ becomes

$$C_L = \frac{2}{V_\infty S} \int_{-b/2}^{b/2} \Gamma(y_0)\, dy \tag{A.227}$$

Figure A.14 shows that the induced drag per unit span is

$$D_i'(\alpha) = L'(y)\, \sin\alpha_i \approx L'(\alpha)\, \alpha_i(y), \tag{A.228}$$

where α_i is given in equation (A.221). Again, we can integrate the drag per unit span over the whole wing to get the total induced drag of the wing.

$$D_i = \int_{-b/2}^{b/2} L'(y)\alpha_i(y)\, dy = \rho_\infty V_\infty \int_{-b/2}^{b/2} \Gamma(y)\alpha_i(y)\, dy \tag{A.229}$$

The drag coefficient in terms of the circulation of the lifting-line becomes

$$C_{D,i} = \frac{D}{\frac{1}{2}\rho_\infty V_\infty S} = \frac{2}{V_\infty S} \int_{-b/2}^{b/2} \Gamma(y)\alpha_i(y)\, dy. \tag{A.230}$$

B Advanced Methodology Topics

Appendix B is intended as a resource for those interested in the technical aspects of our research methods, providing a deeper dive into the specialised techniques employed. It enhances transparency and facilitates future replication or extension of our work, but the thesis is designed to be fully accessible without a detailed engagement with these advanced methodologies.

The advanced methodology covers QASTRO's implementation, the boundary condition implementation in QASTRO, the limited storage procedure for the ALM-suboptimisations, indicates that for structural optimisations, the derivative computation can be simplified and elaborates on the optimisation time computing.

B.1 QASTRO's workflow and implementation

This appendix section illuminates the technical intricacies of QASTRO. As an integrated aerostructural optimisation tool, it harnesses the computational speed of Fortran and the user-friendly interface of Python, capitalising on the respective strengths of each programming language. The Fortran codes are wrapped into Python. The following paragraphs will detail how the FEM and LLT, introduced in Sections 3.1 and 3.2, are synergistically combined to form the comprehensive aerostructural model explained in section 3.3. This segment is crafted to offer a succinct yet thorough overview of QASTRO's functional architecture, providing clarity on its dynamic operational processes for advanced aerostructural analysis.

Figure B.1 illustrates the workflow of QASTRO. The figure illustrates which tasks the Fortran and Python levels perform. The LLT model, as described in section 3.3, takes as an input the state variables (circulation strengths $\boldsymbol{\Gamma}$), the design variables and the displaced aerodynamic mesh, which takes as an input the displacement of the structural model. For simplicity, figure B.1 illustrates only the twist angles $\boldsymbol{\alpha}_0$ as design variables for the LLT model. Given the displaced aerostructural mesh and the state and design variables, the LLT model computes the LLT residual function $\boldsymbol{r}_{aero}$ and the aerodynamic forces $\boldsymbol{F}_{LLT}$.

On the structural side, the FEM, detailed in section 3.1, takes as input the displacements $\boldsymbol{d}$, the design variables, e.g. the beam-truss element thicknesses $\boldsymbol{t}$ and the transformed aerodynamic forces acting on the aeroplane structure $\boldsymbol{F}_{FEM}$. The FEM model returns the structural residual function $\boldsymbol{r}_{struc}$.

The LLT and the FEM residual functions are given to a non-linear solver in Python. The details of the non-linear solver are given in section3.3. The non-linear solver updates the state variables $\boldsymbol{p}^T = \begin{bmatrix} \boldsymbol{d}^T & \boldsymbol{\Gamma}^T \end{bmatrix}$ until both residual functions $\boldsymbol{r}_{aero}$ and $\boldsymbol{r}_{struc}$ are zero.

Upon completion of the aerostructural analysis, the output functions such as lift, drag, weight, and margins are known. QASTRO computes the objective and constraint functions based on the output functions. In figure B.1 the objective function is based on the drag of the aircraft, but the user can choose different objective functions. The KS function aggregates all margins, and the objective and the constraint functions are combined into the augmented Lagrangian function.

These functions — the objective function, the ALM function, and the KS function — are then passed on to the optimisation algorithms. These optimisers reside within the Python layer of QASTRO, taking advantage of Python's user-friendly environment and the powerful optimisation tools available within the SciPy library (VIRTANEN *et al.*, 2020). They iteratively adjust the design variables (e.g. $\boldsymbol{t}$ and $\boldsymbol{\alpha}_0$) to find the optimal solution that minimises the objective function while satisfying all the imposed constraints.

This integrated approach allows for a robust optimisation process, leveraging the strengths of both Fortran and Python to solve complex aerostructural problems efficiently.

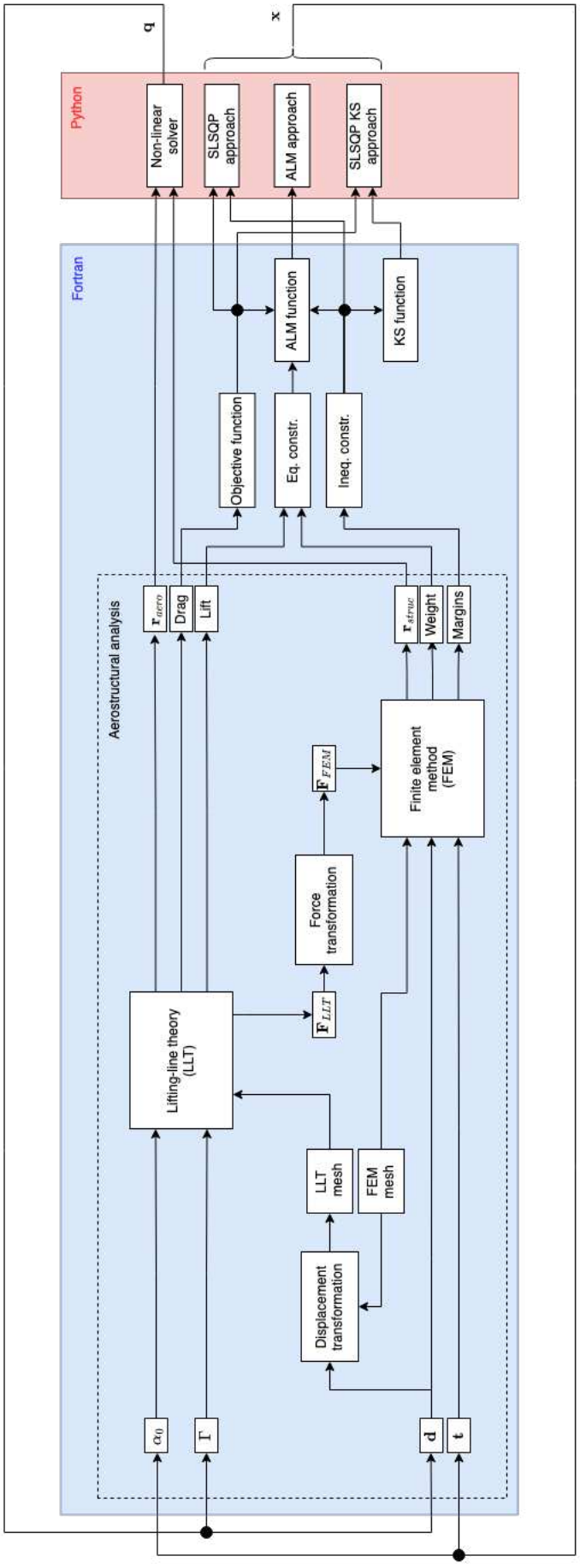


FIGURE B.1 – QASTRO's workflow.

B.2 Structural boundary conditions implementation

This section explains to the reader the boundary condition implementation in QASTRO. The stiffness of one beam-truss element is $\boldsymbol{k}_{\text{local}} =$

$$
\begin{array}{c}
\begin{array}{cccccccccccc}
u_1 & v_1 & w_1 & \phi_{1x} & \phi_{1y} & \phi_{1z} & u_1 & v_2 & w_2 & \phi_{2x} & \phi_{2y} & \phi_{2z}
\end{array} \\
\begin{bmatrix}
\frac{AE}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{AE}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} & 0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & \frac{6EI_z}{L^2} \\
0 & 0 & \frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 & 0 & 0 & -\frac{12EI_y}{L^3} & 0 & -\frac{6EI_y}{L^2} & 0 \\
0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 \\
0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 \\
0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} \\
-\frac{AE}{L} & 0 & 0 & 0 & 0 & 0 & \frac{AE}{L} & 0 & 0 & 0 & 0 & 0 \\
0 & -\frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} & 0 & \frac{12EI_z}{L^3} & 0 & 0 & 0 & -\frac{6EI_z}{L^2} \\
0 & 0 & -\frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 & 0 & 0 & \frac{12EI_y}{L^3} & 0 & \frac{6EI_y}{L^2} & 0 \\
0 & 0 & 0 & -\frac{GJ}{L} & 0 & 0 & 0 & 0 & 0 & \frac{GJ}{L} & 0 & 0 \\
0 & 0 & -\frac{6EI_y}{L^2} & 0 & \frac{2EI_y}{L} & 0 & 0 & 0 & \frac{6EI_y}{L^2} & 0 & \frac{4EI_y}{L} & 0 \\
0 & \frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{2EI_z}{L} & 0 & -\frac{6EI_z}{L^2} & 0 & 0 & 0 & \frac{4EI_z}{L}
\end{bmatrix}
\end{array}
$$

where E, G, J, I_y, I_z, A, and L are the elastic modulus, shear modulus, second moment of area around the x-axis, y-axis, and z-axis, cross-sectional area, and length of a beam-truss element, respectively (LOGAN, 2012).

The local stiffness matrices of all elements are assembled in one stiffness matrix $\boldsymbol{K}$ according to their connectivity. QASTRO modifies the stiffness matrix $\boldsymbol{K}$ to encounter the boundary conditions. We need to modify $\boldsymbol{K}$ according to the boundary conditions to be able to solve equation 3.8 for the displacements $\boldsymbol{d}$. Two options exist to modify $\boldsymbol{K}$ (COOK *et al.*, 2001). Suppose the structure has n_{DoF} DoFs, and the DoF $i = 3$ is fixed. The first option is to remove all rows and columns of the fixed DoFs in $\boldsymbol{K}$. Thus, we need to remove the third row and third column of $\boldsymbol{K}$ in our example. The second option is to replace all vectors and columns of fixed DoFs with a zero vector, which contains a single 1. This 1 is located at the intersection of the rows and columns, which got replaced by the zero vector. Hence, in our example, we replace the third column and the third row with a zero vector and put a 1 at $\boldsymbol{K}(3,3)$. So the system which remains to solve is

$$
\begin{bmatrix}
k_{1,1} & k_{1,2} & 0 & k_{1,4} & \dots & k_{1,\,n_{DoF}} \\
k_{2,1} & k_{2,2} & 0 & k_{2,4} & \dots & k_{2,\,n_{DoF}} \\
0 & 0 & 1 & 0 & \dots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
k_{n_{DoF}-1,\,1} & k_{n_{DoF}-1,\,2} & 0 & k_{n_{DoF}-1,\,4} & \dots & k_{n_{DoF}-1,\,n_{DoF}} \\
k_{n_{DoF},\,1} & k_{n_{DoF},\,2} & 0 & k_{n_{DoF},\,4} & \dots & k_{n_{DoF},\,n_{DoF}}
\end{bmatrix}
\begin{Bmatrix}
d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n_{DoF}-1} \\ d_{n_{DoF}}
\end{Bmatrix}
=
\begin{Bmatrix}
F_1 \\ F_2 \\ 0 \\ \vdots \\ F_{n_{DoF}-1} \\ F_{n_{DoF}}
\end{Bmatrix},
\tag{B.1}
$$

where n_{DoF} is the number of degrees of freedom. QASTRO uses the second option to apply the boundary conditions because it is less computationally expensive and easier to implement.

B.3 Limited storage procedure of ALM sub-optimisation

The ALM uses a limited storage BFGS algorithm from the SciPy library (VIRTANEN *et al.*, 2020) to solve the sub-problems. The limited storage algorithm is based on the paper of Byrd *et*

al. (1994) and avoids storing matrices and avoids computing matrix-vector multiplications. This section explains how the limited storage algorithm works. Section A.1.4.3 exemplifies a limited storage algorithm and can help to understand this limited storage procedure.

For the ALM sub-problem we approximate the augmented Lagrangian function by a quadratic model m_k at $\bar{\boldsymbol{x}}_k$ for which we need to compute $(\boldsymbol{x} - \bar{\boldsymbol{x}}_k)^T \boldsymbol{B}_k (\boldsymbol{x} - \bar{\boldsymbol{x}}_k)$. In this section, we show how the algorithm computes these vector-matrix multiplications by the gradients of n_{mc} previous iterations. According to equation A.53, the update formula of the Hessian approximation $\boldsymbol{B}_k$ is

$$\boldsymbol{B}_{k+1} = \boldsymbol{B}_k + \frac{\boldsymbol{y}_k \boldsymbol{y}_k^T}{\boldsymbol{y}_k^T \boldsymbol{s}_k} - \frac{\boldsymbol{B}_k \boldsymbol{s}_k \boldsymbol{s}_k^T \boldsymbol{B}_k}{\boldsymbol{s}_k^T \boldsymbol{B}_k \boldsymbol{s}_k}. \tag{B.2}$$

Let us define

$$\boldsymbol{s}_k = \begin{bmatrix} \boldsymbol{s}_0 & \boldsymbol{s}_1 & \dots & \boldsymbol{s}_{k-1} \end{bmatrix}, \qquad \boldsymbol{y}_k = \begin{bmatrix} \boldsymbol{y}_0 & \boldsymbol{y}_1 & \dots & \boldsymbol{y}_{k-1} \end{bmatrix}. \tag{B.3}$$

Byrd *et al.* (1994) shows the proof that after the k^{th} update, the Hessian approximation can be written as

$$\boldsymbol{B}_k = \boldsymbol{B}_0 - \begin{bmatrix} \boldsymbol{B}_0 \boldsymbol{S}_k & \boldsymbol{Y}_k \end{bmatrix} \begin{bmatrix} \boldsymbol{S}_k^T \boldsymbol{B}_0 \boldsymbol{S}_k & \boldsymbol{L}_k \\ \boldsymbol{L}_k^T & -\boldsymbol{D}_k \end{bmatrix}^{-1} \begin{Bmatrix} \boldsymbol{S}_k^T \boldsymbol{B}_0 \\ \boldsymbol{Y}_k^T \end{Bmatrix}, \tag{B.4}$$

$$= \sigma_k \boldsymbol{I} - \begin{bmatrix} \sigma_k \boldsymbol{S}_k & \boldsymbol{Y}_k \end{bmatrix} \begin{bmatrix} \sigma_k \boldsymbol{S}_k^T \boldsymbol{S}_k & \boldsymbol{L}_k \\ \boldsymbol{L}_k^T & -\boldsymbol{D}_k \end{bmatrix}^{-1} \begin{Bmatrix} \sigma_k \boldsymbol{S}_k^T \\ \boldsymbol{Y}_k^T \end{Bmatrix}, \tag{B.5}$$

where if we only consider the last n_{mc} iterations, $\boldsymbol{L}_k$ is a $n_{mc} \times n_{mc}$ matrix defined as

$$(\boldsymbol{L}_k)_{i,j} = \begin{cases} \boldsymbol{s}_{k-n_{mc}-1+i}^T \boldsymbol{y}_{k-n_{mc}-1+j} & \text{if } i > j, \\ 0 & \text{otherwise.} \end{cases} \tag{B.6}$$

and

$$\boldsymbol{D}_k = \operatorname{diag} \begin{bmatrix} \boldsymbol{s}_0^T \boldsymbol{y}_0 & \boldsymbol{s}_0^T \boldsymbol{y}_0 & \dots & \boldsymbol{s}_{k-1}^T \boldsymbol{y}_{k-1} \end{bmatrix} \tag{B.7}$$

and $\boldsymbol{B}_0 = \sigma_k \boldsymbol{I}$ is the initialised matrix. The matrix

$$\begin{bmatrix} \boldsymbol{S}_k^T \boldsymbol{B}_0 \boldsymbol{S}_k & \boldsymbol{L}_k \\ \boldsymbol{L}_k^T & -\boldsymbol{D}_k \end{bmatrix} \tag{B.8}$$

is indefinite. Nevertheless, its inverse can be factorised by the Cholesky factorisation of a related matrix

$$\begin{bmatrix} -\boldsymbol{D}_k & \boldsymbol{L}_k^T \\ \boldsymbol{L}_k & \boldsymbol{S}_k^T \boldsymbol{B}_0 \boldsymbol{S}_k \end{bmatrix} = \begin{bmatrix} \boldsymbol{D}_k^{1/2} & \boldsymbol{0} \\ -\boldsymbol{L} \boldsymbol{D}_k^{-1/2} & \boldsymbol{J}_k \end{bmatrix} \begin{bmatrix} -\boldsymbol{D}_k^{1/2} & \boldsymbol{D}_k^{-1/2} \boldsymbol{L}_k^T \\ \boldsymbol{0} & \boldsymbol{J}_k^T \end{bmatrix}, \tag{B.9}$$

where $\boldsymbol{J}_k$ satisfies

$$\boldsymbol{J}_k \boldsymbol{J}_k^T = \boldsymbol{S}_k^T \boldsymbol{B}_0 \boldsymbol{S}_k + \boldsymbol{L}_k \boldsymbol{D}_k^{-1} \boldsymbol{L}_k^T. \tag{B.10}$$

Using this observation, Byrd *et al.* (1994) provides the direct Hessian approximation

$$\boldsymbol{B}_k = \sigma_k \boldsymbol{I} - \begin{bmatrix} \boldsymbol{Y}_k & \sigma_k \boldsymbol{S}_k \end{bmatrix} \begin{bmatrix} -\boldsymbol{D}_k^{1/2} & \boldsymbol{D}_k^{-1/2} \boldsymbol{L}_k^T \\ \boldsymbol{0} & \boldsymbol{J}_k^T \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{D}_k^{1/2} & \boldsymbol{0} \\ -\boldsymbol{L}_k \boldsymbol{D}_k^{-1/2} & \boldsymbol{J}_k \end{bmatrix}^{-1} \begin{Bmatrix} \boldsymbol{Y}_k^T \\ \sigma_k \boldsymbol{S}_k^T \end{Bmatrix}. \tag{B.11}$$

We would like to compute $\boldsymbol{v}^T \boldsymbol{B}_k \boldsymbol{v}$, where $\boldsymbol{v} = (\boldsymbol{x} - \bar{\boldsymbol{x}}_k)$, thus

$$\boldsymbol{v}^T\boldsymbol{B}_k\boldsymbol{v} = \sigma_k\boldsymbol{v}^T\boldsymbol{v} - \boldsymbol{v}^T\boldsymbol{W}_k^T\begin{bmatrix} -\boldsymbol{D}_k^{1/2} & \boldsymbol{D}_k^{-1/2}\boldsymbol{L}_k^T \\ \boldsymbol{0} & \boldsymbol{J}_k^T \end{bmatrix}^{-1}\begin{bmatrix} \boldsymbol{D}_k^{1/2} & \boldsymbol{0} \\ -\boldsymbol{L}_k\boldsymbol{D}_k^{-1/2} & \boldsymbol{J}_k \end{bmatrix}^{-1}\boldsymbol{W}_k\boldsymbol{v} \tag{B.12}$$

where

$$\boldsymbol{W}_k = \begin{Bmatrix} \boldsymbol{Y}_k^T \\ \sigma_k\boldsymbol{S}_k^T \end{Bmatrix} \tag{B.13}$$

$\boldsymbol{v}^T\boldsymbol{B}_k\boldsymbol{v}$ will be computed in four steps. Those four steps are indicated in equation B.14. Every encircled number represents one step.

$$\boldsymbol{v}^T\boldsymbol{B}_k\boldsymbol{v} = \underbrace{\sigma_k\boldsymbol{v}^T\boldsymbol{v}}_{④} - \underbrace{\overbrace{\boldsymbol{v}^T\boldsymbol{W}_k^T}^{\boldsymbol{Q}_k^T}\underbrace{\overbrace{\begin{bmatrix} -\boldsymbol{D}_k^{1/2} & \boldsymbol{D}_k^{-1/2}\boldsymbol{L}_k^T \\ \boldsymbol{0} & \boldsymbol{J}_k^T \end{bmatrix}^{-1}\begin{bmatrix} \boldsymbol{D}_k^{1/2} & \boldsymbol{0} \\ -\boldsymbol{L}_k\boldsymbol{D}_k^{-1/2} & \boldsymbol{J}_k \end{bmatrix}^{-1}}^{\boldsymbol{A}_k^{-1}}\underbrace{\overbrace{\boldsymbol{W}_k\boldsymbol{v}}^{\boldsymbol{Q}_k}}_{①}}_{②}}_{③} \tag{B.14}$$

First, we calculate the products $\boldsymbol{Y}_k^T\boldsymbol{v}$ and $\sigma_k\boldsymbol{S}_k^T\boldsymbol{v}$, which gives us the matrix-vector product $\boldsymbol{Q}_k = \boldsymbol{W}_k\boldsymbol{v}$. In the second step, we need to compute $\boldsymbol{A}_k^{-1}\boldsymbol{Q}_k$ where $\boldsymbol{A}_k^{-1}$ represents both inverted matrices. Computing $\boldsymbol{A}_k^{-1}\boldsymbol{Q}_k$ is equivalent to solve $\boldsymbol{A}_k\boldsymbol{t} = \boldsymbol{Q}_k$ for $\boldsymbol{t}$. Because we applied the Cholesky factorisation to the matrix in equation B.12, we can solve the system at the cost of $\mathcal{O}(n_{mc}^2)$. The solution of the system is a $2n_{mc} \times 1$ vector, which can be multiplied in the third step by the vector $\boldsymbol{Q}_k^T = \boldsymbol{v}^T\boldsymbol{W}_k^T$. Please note that we already computed $\boldsymbol{Q}_k = \boldsymbol{W}_k\boldsymbol{v}$ in the first step. For the fourth step, we compute $\sigma_k\boldsymbol{v}^T\boldsymbol{v}$. Lastly, we need to subtract the result from step three of the result of step four. This procedure is repeated at every iteration k to determine $m_k(\boldsymbol{x})$.

B.4 Simplified structural derivative computation

In this section, we would like to show that the derivative computation of the structural optimisations can be simplified. We can streamline the derivative computation by introducing the analytic result of partial derivatives. In the case of structural optimisation, those partial derivatives are not complex to compute in contrast to aerostructural optimisations.

In the following, we will show how the derivative computation for the structural optimisation can be simplified for the augmented Lagrangian function. The simplified derivative computation for the objective function and the inequality constraint functions are intermediate derivation results. In structural optimisation, we deal only with inequality constraints; hence, our augmented Lagrangian function is

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{\mu}_k) = f(\boldsymbol{x}) + \frac{1}{2}\left\langle \boldsymbol{g}(\boldsymbol{d}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle^T \left\langle \boldsymbol{g}(\boldsymbol{d}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle, \tag{B.15}$$

where $\langle a \rangle = max(0, a)$. The total derivative of the augmented Lagrangian function using the adjoint method is

$$\frac{d\mathcal{A}}{d\boldsymbol{x}} = \frac{\partial\mathcal{A}}{\partial\boldsymbol{x}} + \boldsymbol{\psi}_{\mathcal{A}}^T\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{x}}, \tag{B.16}$$

where $\boldsymbol{\psi}_{\mathcal{A}}$ is determined by

$$\left[\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{d}}\right]^T \boldsymbol{\psi}_{\mathcal{A}} = \left[\frac{\partial \mathcal{A}}{\partial \boldsymbol{d}}\right]^T. \tag{B.17}$$

QASTRO uses a matrix-free approach to solve for the adjoint variables $\boldsymbol{\psi}_{\mathcal{A}}$, and it is precisely this step which can be simplified. For the structural optimisation, the Jacobian matrix $\partial \boldsymbol{r}/\partial \boldsymbol{d}$ is the stiffness matrix, which is already known.

$$\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{d}} = \boldsymbol{K} \tag{B.18}$$

That means we only need to execute the backward-mode AD script once to get $\partial \mathcal{A}/\partial \boldsymbol{d}$, and we can solve for the adjoint variables. However, let's analyse the partial derivative of the augmented Lagrangian function with respect to the state variables:

$$\frac{\partial \mathcal{A}}{\partial \boldsymbol{d}} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{d}} + \rho_k \, \tilde{\boldsymbol{g}}(\boldsymbol{d}) \frac{\partial \tilde{\boldsymbol{g}}(\boldsymbol{d})}{\partial \boldsymbol{d}} \tag{B.19}$$

$$= \rho_k \, \tilde{\boldsymbol{g}}(\boldsymbol{d}) \frac{\partial \tilde{\boldsymbol{g}}(\boldsymbol{d})}{\partial \boldsymbol{d}} \tag{B.20}$$

where $\tilde{\boldsymbol{g}}(\boldsymbol{d}) = \left\langle \boldsymbol{g}(\boldsymbol{d}) + \frac{\boldsymbol{\mu}_k}{\rho_k} \right\rangle = max\left(0, \boldsymbol{g}(\boldsymbol{d}) + \frac{\boldsymbol{\mu}_k}{\rho_k}\right)$. $\partial f(\boldsymbol{x})/\partial \boldsymbol{d} = \boldsymbol{0}$ because the objective function, the mass of the structure is independent of the structure's displacements. Please note that if the constraints are not violated, $\tilde{\boldsymbol{g}}(\boldsymbol{d}) = \boldsymbol{0}$, implies that $\partial \mathcal{A}/\partial \boldsymbol{d} = \boldsymbol{0}$ because $\mathcal{A}$ is not a function of $\boldsymbol{d}$ if the constraints are not violated. We just need to execute the backward-mode AD script to get $\partial \mathcal{A}/\partial \boldsymbol{d}$ if the constraints are violated. In all other cases, we can compute the adjoint variables without a single backward-mode AD execution. Also, note that the adjoint variables of the non-violated augmented Lagrangian function are in the null space of the stiffness matrix. Because the zero vector is included in the null space, the zero vector is a valid solution for the adjoint variables. Thus, if the constraints are not violated, it holds that $d\mathcal{A}/\partial \boldsymbol{x} = \partial \mathcal{A}/\partial \boldsymbol{x}$.

After the adjoint variables are known, QASTRO executes one more time the backward-mode AD script to get the total derivative $d\mathcal{A}/d\boldsymbol{x}$. QASTRO does not explicitly compute the Jacobian matrix $\partial \boldsymbol{r}/\partial \boldsymbol{x}$, but exploits the way the backward-mode AD is defined to get with a single backward-mode AD execution $d\mathcal{A}/d\boldsymbol{x}$. For details, see section 3.5. However, we will show that the final backward-mode AD execution is redundant, too. For the structural optimisation, the partial derivative of the augmented Lagrangian function with respect to the design variables can be simplified:

$$\frac{\partial \mathcal{A}}{\partial \boldsymbol{x}} = \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}} + \rho_k \, \tilde{\boldsymbol{g}}(\boldsymbol{d}) \frac{\partial \tilde{\boldsymbol{g}}(\boldsymbol{d})}{\partial \boldsymbol{x}} \tag{B.21}$$

$$= \frac{\partial f(\boldsymbol{x})}{\partial \boldsymbol{x}}, \tag{B.22}$$

$\partial \tilde{\boldsymbol{g}}(\boldsymbol{d})/\partial \boldsymbol{x} = \boldsymbol{0}$ because the stress in the structure does not depend directly on the design variables $\boldsymbol{x}$. But note that the stress depends on the stiffness matrix $\boldsymbol{K}$, which is dependent on the design variables $\boldsymbol{x}$. Hence, there is only an implicit dependency between $\tilde{\boldsymbol{g}}(\boldsymbol{d})$ and $\boldsymbol{x}$ (MARTINS; NING, 2021). Consequently, the partial derivative of the augmented Lagrangian function is equal to the partial derivative of the objective function with respect to the design variables. Our objective function is the mass of the structure, which is

$$f(\boldsymbol{x} = \boldsymbol{t}) = \frac{1}{m_{w0}} \sum_{i}^{n_{\text{beams}}} \rho_{\text{mat}\,i} \, L_i \, A_i = \frac{1}{m_{w0}} \sum_{i}^{n_{\text{beams}}} 2\pi \rho_{\text{mat}\,i} \, L_i \, r_i \, t_i. \tag{B.23}$$

Thus, the partial derivative of the objective function with respect to the design variables is

$$\frac{\partial f}{\partial \boldsymbol{x}} = \frac{\partial f}{\partial \boldsymbol{t}} = \frac{1}{m_{w0}} \sum_{i}^{n_{\text{beams}}} 2\pi \rho_{\text{mat}\,i} \, L_i \, r_i. \tag{B.24}$$

The only partial derivative, which is missing, is $\partial \boldsymbol{r}/\partial \boldsymbol{x}$, but in the structural optimisation, we can compute $\partial \boldsymbol{r}/\partial \boldsymbol{x}$ analytically:

$$\frac{\partial \boldsymbol{r}}{\partial \boldsymbol{x}} = \frac{\partial \boldsymbol{K}}{\partial \boldsymbol{x}} \boldsymbol{d}, \tag{B.25}$$

We need to know the partial derivative of the stiffness matrix with respect to the thickness of the beam-truss elements, which is easy to compute. The stiffness matrix consists of a few repetitive entries. One of them is, for example, AE/L, where $A = \pi t(t+2r)$ is the cross-sectional area of the beam-truss element, E is the elasticity modulus, L is the length of the beam-truss element, r is the inner radius and t is the thickness. The partial derivative of AE/L with respect to the beam-truss thickness is $\pi(2t + 2r)E/L$. We can obtain similar simple derivatives for all other entries in the stiffness matrix. Thus, we can compute $\partial \boldsymbol{r}/\partial \boldsymbol{x}$ analytically. In the final step, we can get the total derivative of the augmented Lagrangian function by equation B.16. Please note that in this simplified approach, we need to compute matrix-vector operations, which are unnecessary in the derivative computation described in section 3.5.

To summarise, we can compute the derivative of the augmented Lagrangian function in the context of structural optimisation with a maximum of a single backward-mode AD script execution. However, in this master thesis, we reject the use of this simplified derivative computation.

B.5 Computational time measurement

In this section, we delineate the methodology employed to ascertain the optimisation time for the structural optimisation outcomes. We opt for a calculated approach over direct measurement due to inherent challenges in standardising computational time across different hardware and execution conditions.

The crux of this research lies in juxtaposing three distinct optimisation approaches, with a primary emphasis on evaluating their efficiency through computational duration for structural optimisation and on the optimisation outcomes for aerostructural optimisation. Given the pivotal role of optimisation time in gauging the performance of structural optimisation methods, a reliable measure of this metric is paramount. However, the direct measurement of computational time is fraught with variability due to hardware dependencies and inherent fluctuations in execution time, rendering it an unreliable metric for comparative analysis, especially when optimisation durations are minimal.

To circumvent these impediments, we adopt a calculated methodology to derive optimisation time, ensuring consistency and enabling extrapolation of optimisation durations across disparate hardware environments. This approach also allows for the exclusion of initialisation and unrelated computational durations, focusing solely on the optimisation-related computations.

The computational effort in structural optimisation bifurcates into two segments: determining state variables and acquiring derivatives. These components are subject to variability contingent upon the design state, attributed to the integration of nested optimisation (eq. 3.33) within the primary optimisation process.

The nested optimisation processes are crucial in obtaining both the state variables and the derivatives. To deduce the state variables, it's imperative to compute the residual function. The execution time for this computation remains constant across different design states, making it a reliable metric for estimating the time required to solve for the state variables. We can thus measure the execution time of the residual function and use this as a basis to calculate the time involved in determining the state variables.

Similarly, when it comes to computing derivatives, the solution hinges on executing the

backward-mode AD script. This script is integral to the nested optimisation process that yields the necessary derivatives by solving the residual equation 3.72. Notably, the execution time of the backward-mode AD script is also design state-independent. By assessing the duration required to execute this script, we can accurately compute the time needed to obtain the derivatives. This approach systematically quantifies the computational efforts involved in both aspects of the optimisation process, free from the variabilities tied to different design states.

Hence, by meticulously tracking the frequency of residual function evaluations and backward-mode AD script executions within these nested optimisation scenarios and understanding the invariant computational demand of each evaluation, we can precisely calculate the total optimisation time. This calculated time is devoid of the variabilities and external dependencies inherent in direct measurements, providing a robust and reliable metric for comparative analysis of different optimisation methodologies within this thesis.

Let us draw our attention on the time computation for the structural optimisation of the Helios aircraft model. The optimisation time is calculated based on the measured time to get the residual function to determine the displacements and rotations $\boldsymbol{d}$ and the measured time it takes to call the backward-mode AD code to get the derivatives. Both times are dependent on the number of beam-truss elements used to model the Helios aircraft. The dots in figure B.2 show the measured times for $5-80$ beam-truss elements on a Macbook Pro M2, 8 GB with OS 13.5.1. The solid line is a quadratic approximation based on the measured times. It is unsurprising that the time to get the residual and to call the backward-mode AD increases quadratically. The residual function is obtained by a matrix vector operation as shown in equation 3.9. The number of elements in the stiffness matrix $\boldsymbol{K}$ increases by $\mathcal{O}(n_{\text{beams}}^2)$. The backward-mode AD code is a script which determines the derivative line by line according to the chain rule. Hence, its computational price also increases quadratic with the number of beam-truss elements, as the main script also does so. Furthermore, figure B.2 shows that the computational cost to obtain the residual function and to call the backward-mode AD script is very similar.

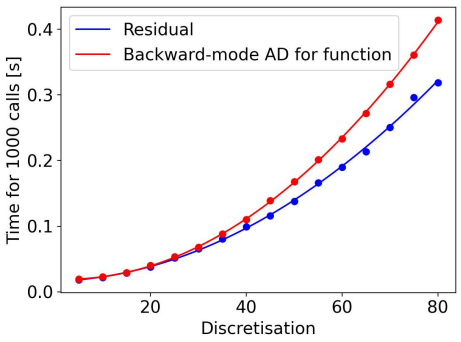


FIGURE B.2 – Time measurement and interpolation for the Helios aircraft model. Time to get residual function and to run the backward-mode AD script for a function of interest. Times are shown as function of number of beam-truss elements used to discretise the wing.

Please note that the Helios aircraft is two dimensional, but modelled in three dimensions. In QASTRO it is not possible to optimise two dimensional models. Thus, we get in total $(n_{\text{beams}} - 1) \cdot 6$ DoFs.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
DM	2024		88

5. TÍTULO E SUBTÍTULO:

Comparison of constrained optimisation methods for aerostructural design

6. AUTOR(ES):

Thorben Fabian Koch

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Multidisciplinary optimisation; Aerostrucutral Analysis; Constrained Optimisation

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

10. APRESENTAÇÃO: ( ) **Nacional** (**X**) **Internacional**

ITA, São José dos Campos. Curso de Mestrado. Programa de Pós-Graduação em Engenharia Aeronáutica e Mecânica. Área de Projeto Aeronáutico, Estruturas e Sistemas Aeroespaciais. Orientador: Prof. Dr. Ney Rafael Secco. Coorientador: Dr. ir. Arne van Garrel em 2024. Publicada em 2024.

11. RESUMO:

In the pursuit of sustainable aviation, the demand for highly efficient and environmentally friendly aircraft takes precedence. This imperative brings to light the complexities of aerostructural coupling, underscoring the necessity for Multidisciplinary Design Optimisation (MDO) and gradient-based optimisation techniques.

Central to aerostructural optimisation is the intricate problem of managing numerous inequality constraints, such as structural stresses, which traditional optimisation methods often struggle to handle efficiently. The current methods, such as using *max* functions, the Kreisselmeier-Steinhauser (KS) function, and compliance approaches, encounter challenges in efficiently managing the constraints as the number of constraint functions increases. This thesis explores the potential of the Augmented Lagrangian method (ALM) to transcend these limitations by offering a more effective approach to managing inequality constraints.

This thesis explores aerostructural optimisation through a comprehensive study that employs a finite element model for structural analysis and an adaptation of Prandtl's lifting line theory for aerodynamics. Central to this investigation is the comparison of the ALM approach with both a non-aggregating optimisation approach, which directly handles all inequality constraints, and an aggregating approach that employs a KS function to consolidate these constraints.

Focusing on a model of the Helios Pathfinder Plus aircraft, the study highlights the effectiveness of the ALM, particularly in the context of derivative calculation via the adjoint method. This contrasts with the complexities and computational demands associated with the non-aggregating optimisation approach, especially in handling complex structural models. The aggregating approach, while time-efficient, does not achieve the same level of accuracy and computational efficiency as the ALM.

In highly discretised structural weight optimisations, the ALM significantly outperforms competing approaches, achieving speeds nearly twice as fast as the aggregating approach and ten times faster than the non-aggregating approach. Moreover, in the context of highly complex aerostructural optimisations, where both the non-aggregating and aggregating approaches struggled to achieve convergence, the ALM consistently demonstrated robust and reliable convergence behavior. The research underscores the significance the fine-tuning of ALM parameters to balance exploration of the optimisation landscape and the pursuit of feasible solutions. Overly conservative parameter settings may lead the ALM algorithm to suboptimal solutions by prioritising rapid feasibility over thorough exploration.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**