MASTER OF SCIENCE THESIS

Terminal Area Energy Management Trajectory Optimization using Interval Analysis

Nuno Ricardo Salgueiro Filipe

December 4, 2008

Faculty of Aerospace Engineering

Delft University of Technology

Terminal Area Energy Management Trajectory Optimization using Interval Analysis

MASTER OF SCIENCE THESIS

Nuno Ricardo Salgueiro Filipe

December 4, 2008

Faculty of Aerospace Engineering · Delft University of Technology

Delft University of Technology

DELFT UNIVERSITY OF TECHNOLOGY

DEPARTMENT OF

CONTROL AND SIMULATION

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled "**Terminal Area Energy Management Trajectory Optimization using Interval Analysis**" by **Nuno Ricardo Salgueiro Filipe** in partial fulfilment of the requirements for the degree of **Master of Science**.

Dated: December 4, 2008

Supervisor:

prof. dr. ir. J. A. Mulder

Readers:

dr. Q. P. Chu

ir. A. M. Barrio

ir. E. van Kampen

ir. E. de Weerdt

Abstract

Optimization is the problem of minimizing (or maximizing) a certain quantity, normally referred to as the cost function, by correctly choosing the values of a certain set of variables - static optimization - or by tuning the shapes of a certain set of time functions - dynamic optimization.

Optimization problems are transversal to many manifestly different disciplines: physics, chemistry, engineering, economics, management, biology, just to name a few, which makes it a field of paramount importance.

Gradient-based methods, like Nonlinear Programming, and Genetic Algorithms are currently the most popular and widespread optimization techniques. These methods cannot guarantee convergence to the global minimum.

Through a branch and bound strategy, interval analysis is able to yield guaranteed and rigorous bounds on the global minimum. It does so by using intervals instead of real numbers and interval arithmetics instead of real arithmetics. Even numerical roundoff errors are considered and do not affect the rigour of the solution.

Optimization based on interval analysis is a relatively recent topic. For example, the application of interval methods to high dimensional (more than a few hundred variables) static optimization problems is still too time consuming. Likewise, the development of interval methods for solving dynamic optimization problems is still just starting.

In this thesis, a static and a dynamic global optimization algorithm are developed using interval analysis. It is shown that their efficiency and correctness are superior to a pre-existent algorithm.

The dynamic global optimization algorithm is applied to the optimization of the Terminal Area Energy Management (TAEM) phase trajectory of a Reusable Launch Vehicle (RLV). During the TAEM phase, the kinematic and potential energies of the vehicle must be regulated in order to assure a safe landing. At the same time, the heading of the vehicle must be aligned with the runway. The aerodynamic model of HORUS-2B was used during the tool's prototyping.

Two versions of the TAEM trajectory optimization tool were developed. The first one assumes no coupling between the longitudinal and lateral motions of the RLV. A Monte Carlo method was used to validate it. It was shown that the tool is able to find the global optimum of a cost function with several local minima.

The second version takes into account the complete fully coupled equations of motion. VNODE-LP, a free interval Ordinary Differential Equations (ODE) solver, was used to rigorously bound the time histories of the states. It was shown that VNODE-LP cannot handle wide interval parameters, and is therefore inadequate for global optimization. Promising alternative interval ODE solvers were identified and recommended.

Keywords: *interval analysis, interval arithmetics, reliable computing, trajectory optimization, TAEM phase, HORUS-2B, static global optimization, dynamic global optimization, VNODE-LP, VSPODE, INTLAB*

Acknowledgments

First of all, I would like to express my gratitude to dr. Q. P. Chu (TU Delft) for being my mentor during this entire time. It was he who first pointed me this line of research. It was also thanks to him that I was able to finish my thesis at ESA/ESTEC.

I would also like to thank ir. E. van Kampen and ir. E. de Weerdt (TU Delft) for their ideas and suggestions on interval analysis and for the patience they have demonstrated during our (sometimes) long meetings.

A special thank you to prof. dr. ir. J. A. Mulder (TU Delft) for accepting me into the Control and Simulation Department and for his long-lasting support to Portuguese students.

I am also very thankful to ir. A. M. Barrio (TEC-ECM, ESA/ESTEC) for his eye-opening advices and for lending me all the literature he has gathered about the TAEM phase.

This thesis would not be possible without the help of dr. G. Ortega, the Head of the Dynamics and Mathematical Analysis Unit (TEC-ECM) of ESA/ESTEC. I would like to thank him for his unconditional support (even after our more or less heated discussions) and for giving me the opportunity to work for TEC-ECM. I would like to commend him for foreseeing the potential of interval analysis.

During my stay at TEC-ECM I had the pleasure of working with some amazing people that disserve to be mentioned here: dr. ir. S. Erb, ir. F. Castellini, ir. J. Melman, A. Riccardi and ir. C. Tienda.

My special thanks to dr. ir. E. Mooij (TU Delft) for providing me the aerodynamic model of the HORUS-2B.

I am also in debt towards dr. ir. H. G. Visser (TU Delft) for finding the time to explain to me the foundations of static and dynamic optimization in a comprehensible manner.

My sincere thanks to ir. T. Lombaerts (TU Delft) for his bright spirit and friendship. I will never forget the wonderful experience it was to be the student assistant of Automatic Flight Control System Design.

I have to give my thanks to dr. N. S. Nedialkov (McMaster University, Canada), the developer of VNODE-LP, for his quick responses to all my questions.

My gratitude also goes to prof. dr. S. M. Rump (Hamburg University of Technology), the developer of INTLAB, for his prompt responses.

I would not be able to finish this thesis (at least not with the same mental sanity) if it weren't for my dear friends. I would like to thank: Fred (for being always there), Tatiana (for never stop talking), Mário (for his friendship), Inês (for her craziness), Boavida (for his attitude towards life), Bruno (for his jokes), Maria (for her Portuguese dishes), Christina (for being my dance partner), Mathilde (*oh la la*), Sören (for never forgetting), Pieter (for being a great guy), Daan (for the Belgian beers) and Paulo (for being an animal).

Last but not the least, I want to thank my mother, my father, my grandmother and my grandfather for making me the man I am today. Thank you!

Por último, quero agradecer à minha mãe, ao meu pai, à minha avó e ao meu avô. Sem vocês nada disto seria possível. Muito obrigado por serem quem são!

Table of Contents

List of Figures

List of Tables

Nomenclature

Delimiters

$\lvert\cdot\rvert$	magnitude
$[a,b]$	set of all real numbers comprehended between (and including) a and b

Greek Symbols

α	angle-of-attack
$\alpha_{sub,\min}$	minimum α during subsonic regime
$\alpha_{sub,\max}$	maximum α during subsonic regime
β	sideslip angle
χ	heading angle
χ_r	heading angle relative to the runway centreline
$\chi_{r,weight}$	x-position cost function weight
δ	deflection angle
Δ	increment
$\Delta E_{rel,sub-phases}$	sub-phases relative length
Δt_{CPU}	CPU time
ε	user-defined stopping criteria
ϕ	end-cost term
Φ	extended end-cost term
γ	flight-path angle
λ_e	equality Lagrangian multipliers
λ_i	inequality Lagrangian multipliers
$\lambda(t)$	influence functions
μ	multiplier parameters
μ_k	flight-path bank angle
μ_1	flight-path bank angle (interval) during sub-phase 1
μ_2	flight-path bank angle (interval) during sub-phase 3

ρ	air density
ρ_0	air density at the reference altitude h_0
ν	multiplier parameters
$\psi(.)$	functions of the final state

Latin Symbols

a	speed of sound
$\vec{a}$	acceleration vector
$\arg(.)$	argument
$C(.)$	inequality constraints on the control variables
C_D	aerodynamic drag coefficient
C_L	aerodynamic lift coefficient
$\vec{D}$	drag force
E	normalized energy
$\bar{f}$	smallest upper bound
$\underline{F}$	minimum of the infima
$\bar{F}$	maximum of the suprema
F^*	bounds on the minimum
$\vec{F}$	vector sum of all external forces
G	centre of mass
h	altitude
h_0	reference altitude
H	Hamiltonian; scale height
i	general index
$\inf(.)$	infimum
j	general index
J	cost function or performance index
$\bar{J}$	smallest calculated upper bound
$\underline{\mathrm{J}}$	minimum of the infima
$\bar{\mathrm{J}}$	maximum of the suprema
J^I	bounds on the minimum

k	general index
l	number of parameters
L	Lagrangian
L/D	lift-over-drag ratio
$\vec{L}$	lift force
m	number of columns; number of inequality constraints; number of control functions; mass
$m(.)$	midpoint or centre
$mag(.)$	magnitude
$\max(.)$	maximum
$mig(.)$	mignitude
$\min(.)$	minimum
M	Mach number
$M_{subsonic}$	maximum subsonic M
n	number of dimensions; number of rows; number of state equations; load factor
n_{split}	number of components split
N	number of equations solvable by HC
$N(.)$	tangency constraints
$\bar{N}$	largest floating point number representable by the computer
$N_{clusters}$	number of clusters
N_{ineq}	number of time subintervals
N_{int}	number of integration steps
$N_{remaining}$	number of remaining boxes
$N_{shrinking}$	number of subintervals in which each parameter is divided during the shrinking phase
N_{split}	number of new boxes per old box
N_{μ_1}	number of subintervals in which μ_1 is split
N_{μ_2}	number of subintervals in which μ_2 is split
p	point inside the remaining boxes; integration order
$p(.)$	inequality constraint
P	parameter

P^I	bounds on the solution point
q	number of functions of the final state; dynamic pressure
$q(.)$	equality constraint
$u(t)$	control function
r	number of equality constraints
$round(.)$	round floating numbers towards closest integers
$R(.)$	inequality constraint
$\sup(.)$	supremum
s	number of inequality constraints on functions of the state variables
$\mathrm{sum}(.)$	summation
S	reference area
$S(.)$	inequality constraints on functions of the state variables
t	time
$t_{in/out}$	time instants at which the system’s trajectory enters or leaves the path constraints
T	time interval
U	set of admissible controls
$\vec{V}$	velocity vector
$\vec{V}_a$	airspeed velocity vector
$\vec{V}_k$	ground velocity or flight-path velocity vector
$w(.)$	width or diameter function
W	width
$\vec{W}$	gravitational force
x	x-position in the runway reference frame
x_b	x-position in the body-fixed reference frame
x_k	x-position in the flight-path reference frame
x_{weight}	x-position cost function weight
$x(t)$	state
X^*	bounds on the solution point
X^I	bounds on the final state
y	y-position in the runway reference frame

y_b	y -position in the body-fixed reference frame
y_k	y -position in the runway reference frame
y_{weight}	y -position cost function weight
Y	aerodynamic side force
z	z -position in the runway reference frame
z_b	z -position in the body-fixed reference frame
z_k	z -position in the runway reference frame

Subscripts

0	initial
abs	absolute
eq	equality
f	final
$feas$	feasible
$fixed$	fixed parameters
$free$	free parameters
$(i), i \in \mathbb{N}$	general identifier
$i, i \in \mathbb{N}$	general identifier; initial
l	left
$j, j \in \mathbb{N}$	general identifier
m	midpoint
$\max$	maximum
mid	midpoint
$\min$	minimum
nom	nominal
r	rudder
rel	relative
$thres$	threshold
$\underline{X}$	lower bound of the interval X
z	along the negative z -body axis

Superscripts

$\dot{x}$	first time derivative of x
'	general identifier
''	general identifier
$*$	global optimum
C	Chu's parameterization
F	Filipe's parameterization
$(i), i \in \mathbb{N}$	general identifier
$initial$	initial box
I	interval
j	dimensional index
m	number of inequality constraints
n	number of dimensions
p	solution point
(q)	q-th total time derivative
r	number of equality constraints
T	transpose
$\bar{X}$	upper bound of the interval X

Other Symbols

$\bullet$	any of the four elementary operations: addition, subtraction, multiplication or division
$\varnothing$	empty set
?	unknown

Acronyms

ALI	Auto-Landing Interface
BB	Branch and Bound
BC	Box Consistency
CPU	Central Processing Unit
DOF	Degrees Of Freedom
ESA	European Space Agency
ECM	Electrical Control Mathematics
ESTEC	European Space Research and Technology Centre
GA	Genetic Algorithms
HAC	Heading Alignment Cylinder/Cone
HH	Hull Consistency
HORUS-2B	Hypersonic Orbital Research and Utilization System
INTLAB	INTerval LABoratory
IVP	Initial Value Problem
M.Sc	Master of Science
MBB	Messerschmitt-Bölkow-Blohm
N/A	Not Applicable or Not Available
NASA	National Aeronautics and Space Administration
NEP	Nominal Entry Point
NLP	Nonlinear Programming
ODE	Ordinary Differential Equation
OS	Operating System
RAM	Random Access Memory
RLV	Reusable Launch Vehicle
TACAN	TACtical Air Navigation
TAEM	Terminal Area Energy Management
TEC-ECM	Dynamics and Mathematical Analysis Unit
TEG	Terminal Entry Gate
TEP	Terminal Entry Point
TPBVP	Two-Point Boundary-Value Problem
TSTO	Two-Stage-To-Orbit
TU Delft	Delft University of Technology
USA	United States of America

Chapter 1 | Introduction

1.1 Background

Optimization is the problem of minimizing (or maximizing) a certain quantity, referred to in mathematics as the cost function, by correctly choosing the values of a certain set of variables - static optimization - or by tuning the shapes of a certain set of time functions - dynamic optimization.

Global optimization is the problem of finding the global minimum of a cost function with several local minima. Finding an arbitrary local minimum is called local optimization.

Optimization problems are transversal to many manifestly different disciplines: physics, chemistry, engineering, economics, management, biology, just to name a few, which makes it a field of paramount importance. While the applications may differ, the mathematical formulation of the problems is pretty much the same, which allows the same optimization techniques to be used in multiple disciplines. In particular, several apparently independent design problems connected with different disciplines may be formulated into a single optimization problem. This constitutes a so-called *multidisciplinary design optimization problem*.

The most popular optimization technique currently available is *Nonlinear Programming* (NLP). As a curiosity, the term *programming* is not related with computer programming but with the word *program* used by the United States military to refer to a schedule. Military logistics programmes were the first application of optimization programming [Wikipedia, 2008b] back in the 1940s.

Gradient-based optimization techniques like NLP require an initial guess for the optimal solution in order to start. Depending on that initial guess, the global minimum might or might not be found. In general, the more the local minima, the harder it will be for the NLP algorithm to find the global solution. Moreover, in many problems, generating a reasonable initial guess is a hard and time consuming task. Many authors consider it an art.

Genetic Algorithms (GA) are a non-gradient-based optimization technique also popular nowadays. They are inspired in evolutionary biology. Like NLP algorithms, they might or might not converge to the global optimum. They have several tuning parameters.

One of the applications of dynamic optimization is trajectory optimization. Trajectory optimization is the process of designing a trajectory that minimizes or maximizes a certain quantity [Wikipedia, 2008c]. As it will be seen shortly, one of the goals of this thesis is to optimize the trajectory described by a Reusable Launch Vehicle (RLV) during the Terminal Area Energy Management (TAEM) phase. The TAEM phase is the second of three phases that constitute the descent flight of a RLV: re-entry, TAEM and landing. The TAEM phase is in charge of controlling the vehicle's total energy and of aligning it with the runway. The manoeuvres performed during this phase are vital for a safe landing.

1.2 Problem statement

There is a need for an optimization technique capable of finding the global optimum in a systematic way. With it, the user would no longer be required to go through several optimization loops. The global solution could be calculated in one go without the need for initial guesses or extensive fine tuning. There is an optimization technique capable of all of this: interval analysis.

Through a branch and bound strategy, interval analysis is able to yield guaranteed and rigorous bounds on the global minimum. It does so by using intervals instead of real numbers and interval arithmetics instead of real arithmetics. Even the roundoff errors introduced by computers do not affect the rigour of the solution. For a more in-depth discussion about interval analysis the reader is referred to Chapter 2.

In particular, interval analysis is a promising alternative to gradient-based methods (e.g., [Chartres *et al.*, 2005b]) and to genetic algorithms (e.g., [Yokoyama and Suzuki, 2005]) for optimizing the descent trajectory of a RLV. So far, these two latter optimization techniques have been unable to guarantee convergence to the global optimum trajectory.

Optimization based on interval analysis is a relatively recent topic. For example, the application of interval methods to high dimensional (more than a few hundred variables) static optimization problems is still too time consuming. Likewise, the development of interval methods for solving dynamic optimization problems is still just starting.

1.3 Thesis goal

In her Master of Science thesis: *Interval Analysis Applied to Re-Entry Flight Trajectory Optimization* [Chu, 2007], Weiwei Chu applied interval analysis to the optimization of the re-entry trajectory of an Reusable Launch Vehicle (RLV). During the course of her thesis, she developed a static and a dynamic global optimization algorithm. She was partially successful: her trajectory optimization algorithm was so time demanding that she was not able to carry out the complete optimization. Moreover, inconsistencies were identified in her algorithms.

The end goal of this thesis is to develop generic static and dynamic global optimization algorithms capable of competing with currently available static and dynamic optimization tools. Chu’s work will be used as a starting point. With this main goal in view, the following intermediate goals were established:

1) Investigate the main virtues and drawbacks of interval analysis.

2) Investigate the basic concepts of interval arithmetics.

3) Develop and test a static global optimization algorithm less time demanding than Chu’s.

4) Develop and test a dynamic global optimization algorithm less time demanding than Chu’s.

5) Apply the dynamic global optimization algorithm to the Terminal Area Energy Management (TAEM) trajectory optimization problem.

1.4 Thesis Outline

This thesis is divided in two parts. The first part was written between August 2007 and March 2008 in the Delft University of Technology, Faculty of Aerospace Engineering, Control and Simulation Department. It is constituted by the author’s Preliminary Master of Science Thesis [Filipe, 2008] published in February 16, 2008. It describes the knowhow acquired during the aforementioned period in static and dynamic global optimization using interval analysis.

The second part of this thesis was written between April and November 2008 in the European Space Research and Technology Centre (ESTEC), the largest site of the European Space Agency (ESA), within the Dynamics and Mathematical Analysis Unit (TEC-ECM). In it, the theory developed in Part I is applied and extended to the design of optimal TAEM trajectories.

1.4.1 Outline of Part I

The first part of this thesis is constituted by four chapters.

The first chapter (Chapter 2) introduces interval analysis and its main virtues and drawbacks. It was written thinking about someone that never heard of interval analysis before.

The second chapter (Chapter 3) gives the mathematical foundations of interval arithmetics and explains some important concepts such as outward rounding, dependency and the wrapping effect. The fundamental theorem of interval analysis is also stated.

The third chapter (Chapter 4) is devoted to static global optimization using interval analysis. The underlying mathematical problem is defined. A static global interval optimization algorithm is designed and compared with Chu's. The fundamental building blocks of the algorithm are carefully explained.

The fourth and last chapter of Part I (Chapter 5) is devoted to dynamic global optimization using interval analysis. The underlying mathematical problem is defined. A dynamic global interval optimization algorithm is designed and compared with Chu's. The fundamental building blocks of the algorithm are carefully explained.

1.4.2 Outline of Part II

The second part of this thesis is constituted by two chapters.

The first chapter (Chapter 6) describes the state-of-the-art in TAEM trajectory design. The 3 degrees of freedom equations of motion are derived.

The second and last chapter of Part II (Chapter 7) recovers the theory explained in Chapter 5 and adapts it to the TAEM optimal trajectory design problem. Two methodologies are investigated: one assuming no coupling between the longitudinal and lateral motions and another one considering the full model.

The conclusions and recommendations for the future research are presented in Chapter 8 and Chapter 9, respectively.

Part I | Static and Dynamic Global Optimization using Interval Analysis

Chapter 2 | Interval Analysis – An Introduction

The conventional arithmetic taught all around the world is based on real numbers. It is the core of almost every computational algorithm ever developed. Since the precision of computers is finite, using real arithmetic will most surely produce rounding errors. How big these rounding errors are is a question many people have tried to answer in the past. It is a fundamental question to assess the reliability of a result.

The basic idea behind interval analysis is to use intervals (of real numbers) instead of real numbers. The purpose is to find guaranteed lower and upper bounds on the exact solution. This gives insight about the effect rounding errors have on the computed result.

Interval algorithms calculate intervals that unquestionably contain the exact solution. The importance of this result cannot be stressed enough. Floating-point algorithms can never guarantee that the exact solution is within some range of the calculated result. This makes interval algorithms extremely more reliable than their floating-point counterparts.

In most problems, bounding the exact solution is of little importance unless the interval bounds are narrow. An interval is said to be *sharp* [Hansen and Walster, 2004] if it is as narrow as possible. Of course that, in the limit, an interval can not be narrower than the shortest distance between two consecutive numbers representable by the computer.

People started using intervals for bounding rounding errors in the 1950's but interval analysis is said to have begun in 1966 with the book *Interval Analysis* [Moore, 1966] written by R. E. Moore. Nowadays, interval analysis is also called *reliable computing* and a journal with the same name exists completely dedicated to the field.

2.1 The virtues of interval analysis

The virtues of interval analysis are far from being limited to bounding rounding errors ([Hansen and Walster, 2004]):

1) When a problem has uncertain parameters it is said that the problem is perturbed. Uncertain parameters can be for example measured quantities subject to experimental errors. If the uncertain parameter is known to lie within a certain interval, any algorithm based on interval arithmetic will rigorously bind all possible solutions.

2) In other problems all the parameters are exactly known. However, it could be of interest to know how the solution of a problem changes when some parameters change. This is called sensitivity analysis. By substituting the parameters by intervals bounding the range of interest, sensitivity analysis can be performed using interval arithmetic.

3) A wide solution interval is a warning that round-off and/or catastrophic cancellation have occurred. These kind of numerical errors are harder to detect with floating-point algorithms than with their interval counterparts.

4) Problems like global optimization can not be solved with noninterval methods. The best floating-point algorithms can do is to find a local optimum and hope that it is indeed the global one. Conversely, interval algorithms provide always guaranteed bounds on the global optimum. Moreover, if there are multiple solutions, all will be found and correctly bounded.

5) According to [Hansen and Walster, 2004], interval methods are in general more reliable in the sense that some interval iterative methods always converge, while their noninterval counterparts do not, e.g., the Newton method for finding the zeros of a nonlinear equation.

6) Interval algorithms have natural stopping criteria: there is no point in iterating further when the bounds on the solution are sufficiently narrow or no further reduction occurs.

7) Interval algorithms for solving systems of nonlinear equations can sometimes prove the existence and uniqueness of a simple, i.e. nonmultiple, zero inside an interval without any extra computations.

8) **All** the solutions of a nonlinear system of equations inside a given initial interval vector, also called a box, will be found and correctly bounded by an interval algorithm.

9) Interval arithmetic can be extended to form a *closed system*, i.e., a system where there are no undefined arithmetic operations or operands. This includes division by zero and undetermined forms like $0/0$, $\infty-\infty$, $0\times\infty$ and ∞/∞ which are normally undefined in real arithmetic.

2.2 The drawbacks of interval analysis

Interval analysis is not without some problems:

1) **Slowness**: since each interval is represented by a lower and upper bound (two real numbers) even the simplest interval operations, like adding or subtracting two intervals, are certain to take longer than their real counterparts, at least, if common hardware is used. This is the price to pay for guaranteed error bounds. However, as Moore's law predicted, computer performance keeps on doubling approximately every two years. Parallel processing, for example, has yet to show all of its potential, as more and more processors are put together. Moreover, different computer companies already have specially developed hardware for computing with intervals. This makes it likely that in a matter of a few years, the relative effort of performing operations on intervals or on real numbers will no longer be an issue.

Also worth mentioning is the fact some kind of problems, like finding all the roots of a polynomial, and in some areas, like robust control, interval algorithms are actually faster [Hansen and Walster, 2004].

2) **Lack of sharpness**: sometimes the initial bounds on a solution might be far from sharp. That can happen due to *dependency* and/or to the *wrapping effect* (both will be carefully explained in Chapter 3). Even though there is no general solution to these problems, there are simple techniques to substantially reduce their significance. Sometimes, simply rewriting an expression is all that takes to get sharp results. Moreover, a lot of research is being done right now to improve interval arithmetic even further.

2.3 Interval arithmetics software toolboxes

It was decided in the beginning that all the algorithms would be implemented in MATLAB. This decision was made taking into account the unrivalled portability and simplicity offered by MATLAB. On the other hand, MATLAB has the theoric drawback of being relatively slower than lower level software languages like C or FORTRAN. However, it will be shown when comparing the results obtained in Chapter 4 and Chapter 5 with the ones obtained with FORTRAN in [Chu, 2007] that the relative speed of MATLAB is not a significant issue. At least, not when compared with the optimality of the algorithms.

By default, MATLAB does not come with a toolbox to work with intervals. However, a list of available interval software for MATLAB is available in [Computer Science, 2007], as well as for other platforms. In there, one can find the interval toolbox INTLAB[1] developed by Siegfried M. Rump from the Hamburg University of Technology in Germany (see [Rump, 2007]). INTLAB is totally free for private or academic purposes and it is completely written in MATLAB. Version 5.4, the one used in Part I, includes, between other features, interval arithmetic operations for working with interval vectors and matrices and rigorous evaluation of standard real functions (e.g., the functions sine and cosine) when intervals are used as arguments.

All the interval software for MATLAB listed in [Computer Science, 2007] requires INTLAB for running and/or is partially written in other languages making INTLAB the obvious choice.

For a quite helpful INTLAB tutorial see [Hargreaves, 2002].

All the results presented in Part I were obtained using a computer equipped with a Pentium 4 processor running at 2.4 GHz and with 1.25 GB of RAM. The operating system running on the machine was Microsoft Windows XP Professional with version 7.3.0 (R2006b) of MATLAB installed.

[1] *INTLAB stands for INTerval LABoratory.*

Chapter 3 | Interval Arithmetic

In this chapter, the most basic and fundamental concepts of interval arithmetic will be explained. For a more in-depth analysis the reader is referred to Chapter 2 of [Hansen and Walster, 2004].

As previously stated, intervals are for interval arithmetic as real numbers are for real arithmetic. An *interval* X with endpoints a and b, where a and b are real numbers satisfying $a \le b$ will be represented as:

$$X = [a,b] \tag{3-1}$$

i.e., X is the set of all real numbers comprehended between (and including) a and b:

$$X = \{x \in \mathbb{R} \mid a \le x \le b\} \tag{3-2}$$

3.1 Outward rounding

Imagine X is an intermediate result calculated by some interval algorithm. It can happen that the endpoints a and b are not exactly representable by the computer:

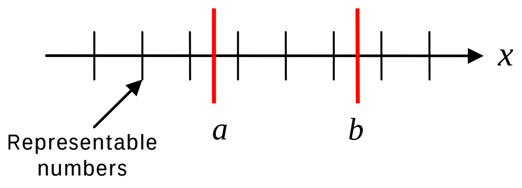

Figure 3.1: *Endpoints not representable by the computer.*

To guarantee that the entire computed interval is saved, *outward rounding* must be used. To explain how outward rounding works, assume $X' = [a', b']$ is the interval saved by the computer. Then, a' should be the largest machine-representable number smaller than a and b' should be smallest machine-representable number larger than b, as illustrated in **Figure 3.2**.

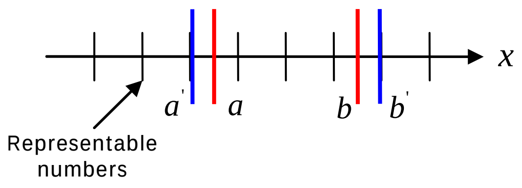

Figure 3.2: *Outward rounding.*

Outward rounding is crucial to guarantee that the exact solution is never deleted. Luckily, all computers since the 1980's have the option to perform *directed rounding*, i.e., the user can specify if a number is to be rounded upwards or downwards. Direct rounding is used to carry out the so needed outward rounding in interval arithmetic.

The mathematical rigour of interval analysis cannot be claimed, if a single roundoff error has not been taken into account [Nedialkov, 2006a]. Therefore, it is of paramount importance that all roundoff errors are identified and correctly incorporated in the final interval result.

3.2 Basic definitions

The following definitions establish the basis for interval arithmetic:

1) **Infimum and supremum:** the lower and upper endpoints of an interval $X=[a,b]$ will be denoted as $\underline{X}=a$ and $\overline{X}=b$. $\underline{X}=\inf(X)$ and $\overline{X}=\sup(X)$ are also called the *infimum* and the *supremum* of X, respectively.

2) **Positive, negative, nonnegative and nonpositive intervals:** an interval is said to be *positive* if $a>0$ and *negative* if $b<0$. Similarly, an interval is said to be *nonnegative* if $a\geq 0$ and *nonpositive* if $b\leq 0$.

3) **Equal intervals:** two intervals, $X=[a,b]$ and $Y=[c,d]$, are said to be *equal* if and only if:

$$\begin{cases} a=c \\ b=d \end{cases} \tag{3-3}$$

4) **Ordering:** an interval X is said to be *smaller* than Y, $X<Y$, if and only if $b<c$.

5) **Thin and thick intervals:** if $a=b$ the interval is said to be *degenerate*, *thin* or a *point interval* since it is equivalent to the real number a. If $a<b$, then the interval is said to be *thick*.

6) **Containment:** an interval X is said to be *strictly contained* in Y, $X\subset Y$, if and only if:

$$\begin{cases} \underline{X}>\underline{Y} \\ \overline{X}<\overline{Y} \end{cases} \tag{3-4}$$

In the same way, X is said to be *contained* in Y, $X\subseteq Y$, if and only if:

$$\begin{cases} \underline{X}\geq\underline{Y} \\ \overline{X}\leq\overline{Y} \end{cases} \tag{3-5}$$

7) **Image of the union and of the intersection of two intervals**: consider two intervals X and Y and a function $f:X\to Y$. If X_1 and X_2 are subintervals of X then it can be easily proven that ([Jaulin *et al.*, 2001]):

$$f(X_1\cap X_2)\subset f(X_1)\cap f(X_2) \tag{3-6}$$

$$f(X_1\cup X_2)=f(X_1)\cup f(X_2) \tag{3-7}$$

3.3 Real functions of intervals

The following functions are repeatedly used in algorithms based on interval analysis:

1) The ***diameter*** or ***width*** of an interval is defined as:

$$w(X) = \overline{X} - \underline{X} \qquad \textit{(3-8)}$$

2) The ***midpoint*** or ***centre*** of an interval, as the name suggests, is given by:

$$m(X) = \frac{\underline{X} + \overline{X}}{2} \qquad \textit{(3-9)}$$

3) The ***magnitude*** of an interval is calculated as follows:

$$mag(X) = \max\left(\left|\underline{X}\right|, \left|\overline{X}\right|\right) = |X| \qquad \textit{(3-10)}$$

while a similar expression is used to calculate the ***mignitude*** of X :

$$mig(X) = \begin{cases} \min\left(\left|\underline{X}\right|, \left|\overline{X}\right|\right) & ,0 \notin X \\ 0 & ,0 \in X \end{cases} \qquad \textit{(3-11)}$$

3.4 The four elementary interval arithmetic operations

Using the concepts of addition $(+)$, subtraction $(-)$, multiplication $(\times)$ and division $(\div)$ from real arithmetic, the corresponding interval arithmetic operation can be defined as:

$$X \bullet Y = \{x \bullet y \mid x \in X, y \in Y\} \qquad \textit{(3-12)}$$

where $\bullet$ symbolizes any of the four operations. In words, (3-12) means that $X \bullet Y$ contains all possible results of $x \bullet y$ for any $x \in X$ and $y \in Y$.

Using (3-12) as reference, the following expressions are used in practice to calculate the endpoints of $X \bullet Y$, with $X = [a,b]$ and $Y = [c,d]$:

$$X + Y = [a+c, b+d] \qquad \textit{(3-13)}$$

$$X - Y = [a-d, b-c] \qquad \textit{(3-14)}$$

$$X \times Y = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)] \qquad \textit{(3-15)}$$

$$\frac{1}{Y} = \left[\frac{1}{d}, \frac{1}{c}\right] \quad ,0 \notin Y \qquad \textit{(3-16)}$$

$$\frac{X}{Y} = X \times \frac{1}{Y} \quad ,0 \notin Y \tag{3-17}$$

Division by an interval containing zero is possible if extended interval arithmetic is used. The rules for doing that can be found in Section 2.1 of [Hargreaves, 2002] and in Section 4.6.1 of [Hansen and Walster, 2004]. Extended interval arithmetic is not properly supported by INTLAB. It is left to the user to implement it if desired. During this thesis, no need was felt as satisfactory results could be obtained without the extra programming. However, that might be necessary in a near future.

Also useful to avoid dependency in multiplication (see Section 3.5 for a detailed explanation of dependency) is the following rule (this one implemented in INTLAB) given in [Hansen and Walster, 2004]:

$$X^n = \begin{cases} [1,1] & ,if\ n=0 \\ \left[a^n, b^n\right] & ,if\ a \geq 0\ or\ if\ n\ is\ odd \\ \left[b^n, a^n\right] & ,if\ b \leq 0\ and\ n\ is\ even \\ \left[0, \max\left(a^n, b^n\right)\right] & ,if\ a \leq 0 \leq b\ and\ n > 0\ is\ even \end{cases} \tag{3-18}$$

3.5 Dependency

Dependency is probably the single most significant problem of interval analysis. To explain it, an example given in [Hansen and Walster, 2004] will be used. Consider the interval $X = [a,b]$. While most people would say that $X - X$ equals zero, applying (3-14) yields:

$$X - X = [a-b, b-a] \tag{3-19}$$

which is only zero if $a = b$. The solution is a thick interval and not a thin one. This widening of computed intervals is called dependency.

Dependency occurs in the preceding example because by definition $X - X$ is given by:

$$X - X = \left\{x - y \mid x \in X, y \in X\right\} \tag{3-20}$$

and <u>not</u> by:

$$X - X = \left\{x - x \mid x \in X\right\} \tag{3-21}$$

In other words, $X - X$ is computed as if it were $X - Y$ with Y numerically equal but independent of X.

Another example from [Hansen and Walster, 2004] shows why (3-18) is useful to avoid dependency in multiplication. Consider the operation $[-1,2]^2$. Using (3-15) one gets:

$$[-1,2]^2 = [-1,2] \times [-1,2] = [-2,4] \tag{3-22}$$

while (3-18) yields:

$$[-1,2]^2=[0,4] \tag{3-23}$$

The latter result is not affected by dependency and, therefore, is sharper.

An important note about dependency is given in [Moore, 1966]: if an interval variable appears only once in a given expression, then it will not produce dependency. In particular, if every interval variable appears only once in a given expression, dependency will not occur. Take the following example from [Hansen and Walster, 2004]:

$$f_{(1)}(X,Y)=\frac{X-Y}{X+Y} \tag{3-24}$$

Expression (3-24) can be alternatively written as:

$$f_{(2)}(X,Y)=1-\frac{2}{1+\frac{X}{Y}} \tag{3-25}$$

The subscripts are only used for identification purposes. Despite analytically equivalent, when analysed in terms of dependency, the two forms are different. Since X and Y only appear once in (3-25), dependency will not occur. But the same can not be said about (3-24). Taking $X=[1,2]$ and $Y=[3,4]$, one gets:

$$f_{(1)}(X,Y)=[-0.7501,-0.1666] \tag{3-26}$$

$$f_{(2)}(X,Y)=[-0.6001,-0.1999] \tag{3-27}$$

As foreseen, $f_{(2)}(X,Y)\subset f_{(1)}(X,Y)$ as a consequence of dependency. Note, however, that if $0\in Y$, expression (3-25) can not be applied without extended interval arithmetic. In that case, a deeper analysis would have to be done for figuring out which form is better.

Dependency can also be used to explain why the distributive law does not hold in general when multiplying intervals:

$$X(Y+Z)\neq XY+XZ \tag{3-28}$$

In the left hand side of (3-28), X appears only once while in the right hand side it appears twice, making the form $XY+XZ$ subject to dependency. Hence, in interval arithmetic the so-called *sub-distributive* law of multiplication holds:

$$X(Y+Z)\subseteq XY+XZ \tag{3-29}$$

This law will be extensively used in the upcoming chapters to reduce the dependency effect.

As for the associative and commutative laws of addition and multiplication, they still hold in interval arithmetic:

$$X+Y=Y+X \tag{3-30}$$

$$X\times Y=Y\times X \tag{3-31}$$

$$(X+Y)+Z=X+(Y+Z) \tag{3-32}$$

$$(X \times Y) \times Z = X \times (Y \times Z) \tag{3-33}$$

While it is true that dependency can cause catastrophic numerical instability, it is also true that one of the virtues of interval analysis is to produce wide solution intervals when this happens, warning the user about what is going on. On the contrary, floating-point algorithms can hide catastrophic numerical instability from the user.

3.6 The wrapping effect

The wrapping effect and dependency are two completely unrelated problems. The bounds on a solution might be far from sharp exclusively due to the wrapping effect without any contribution from dependency.

A good example explaining the wrapping effect can be found in [Jaulin *et al.*, 2001]. Take:

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \tag{3-34}$$

and:

$$X = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} [-1,0] \\ [1,2] \end{bmatrix} \tag{3-35}$$

By using (3-15) one gets:

$$A.X = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} . \begin{bmatrix} [-1,0] \\ [1,2] \end{bmatrix} = \begin{bmatrix} [0,2] \\ [1,2] \end{bmatrix} \tag{3-36}$$

Note that:

$$\begin{bmatrix} 0 \\ 2 \end{bmatrix} \in A.X \tag{3-37}$$

but:

$$\begin{bmatrix} 0 \\ 2 \end{bmatrix} \notin \{ A.x \mid x \in X \} \tag{3-38}$$

where $\{ A.x \mid x \in X \}$ is the true *solution set* of the problem. **Figure 3.3** shows geometrically what is happening.

The red area represents the true solution set: $\{ A.x \mid x \in X \}$, while the dashed rectangle is the result calculated in (3-36). This result is the interval vector containing the true solution set with the narrowest possible interval components. It is the so-called *hull* of the solution set. The elementary interval arithmetic operations defined in Section 3.4 are only able to calculate the hull of the solution set and not the solution set itself. In this way, the computed solutions contain points that do not belong to the true solution sets, as is the case with the point $[0 \quad 2]^T$ in the former example[2]. This phenomenon is called *wrapping effect*.

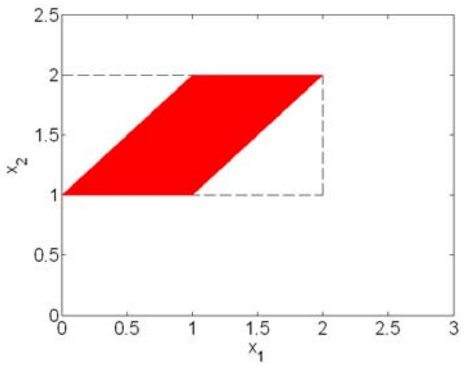

Figure 3.3: *Geometric representation of the wrapping effect.*

The operation most susceptible to the wrapping effect is integration. Integration is a recurrent procedure when optimizing dynamic systems. The rectangular method is the simplest technique to perform numerical integration and the use of interval arithmetics makes it even more straightforward. Its accuracy is directly related to the number of subintervals in which the initial interval is divided, N_{int}.

The rectangular method was chosen due to its simplicity, despite its accuracy limitations. Taylor models could be used to improve this aspect.

Figure 3.4 and expression (3-39) illustrate the idea behind the rectangular method:

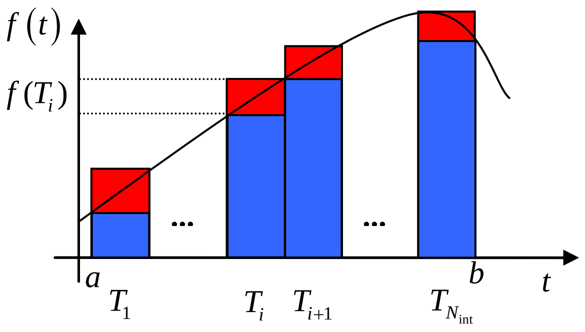

Figure 3.4: *Rectangular integration and the wrapping effect.*

$$\int_a^b f(t)\,dt = \sum_{i=1}^{N_{\text{int}}} f(T_i).w(T_i) \tag{3-39}$$

A very important remark should be done here: the widths $w(T_i)$ might not be machine representable numbers. Therefore, $w(T_i)$ in (3-39) should be understood as the smallest intervals containing the true values of $w(T_i)$.

As explained in [Berz, 2007], the wrapping effect in verified integration algorithms is caused by the need to enclose intermediate results in rectangular boxes. This can be seen in **Figure**

[2] X^T *represents the transpose of* X .

3.4 as the red boxes wrapping $f(t)$. This can lead to integration results that are far from sharp.

The only way to counteract the wrapping effect in the rectangular method is by increasing N_{int}, which has obvious consequences on the computational speed.

Apart from the wrapping effect, the integration results can also be affected by dependency coming from the calculation of $f(T_i)$.

3.7 The fundamental theorem of interval analysis

One of the most important theorems in interval analysis was first demonstrated by Moore in [Moore, 1966]. Various authors recurrently call it the *fundamental theorem of interval analysis*. The subsequent version can be found in [Hargreaves, 2002] and is particularly easy to grasp:

Theorem 3.1: *If the function* $f(Z_1,...,Z_n)$ *is an expression with a finite number of intervals* $Z_1,...,Z_n \in \mathbb{R}$ *and interval operations* $(+,-,\times,\div)$*, and if:*

$$X_1 \subseteq Z_1,....,X_n \subseteq Z_n \tag{3-40}$$

then:

$$f(X_1,...,X_n) \subseteq f(Z_1,...,Z_n). \tag{3-41}$$

For a analytically sound proof of this theorem the reader is referred to Section 3.2 of [Hansen and Walster, 2004].

In particular, **Theorem 3.1** guarantees that:

$$x \in X \Rightarrow f(x) \in f(X) \tag{3-42}$$

where x and X denote a real number and an interval, respectively.

As mentioned in [Hansen and Walster, 2004], one of the far reaching consequences of the fundamental theorem is that it makes global optimization possible.

3.8 Interval vectors and matrices

In the forthcoming chapters, interval vectors and interval matrices will recurrently be used. Hence, the following definitions (based on the definitions given in [Hansen and Walster, 2004]) are in order:

1) **Interval vector or box:** an *interval vector* or a *box* is a vector whose components are intervals. The term *box* comes from the fact that an n-dimensional interval vector represents geometrically an n-dimensional parallelepiped with sides parallel to the coordinate axes.

2) **Interval matrix:** an *interval matrix* is a matrix whose elements are intervals.

3) **Containment:** let x be a real vector and X an interval vector, both with n components. It is said that x is contained in X, $x \in X$, if and only if $x_i \in X_i$ for $i = 1, \ldots, n$. Similarly, if A is a real matrix and A^I is an interval matrix, both with n rows and m columns, $A \in A^I$ if and only if $A_{ij} \in A^I_{ij}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

Now let X and Y be n-dimensional interval vectors. $X \subset Y$ if and only if $X_i \subset Y_i$ for $i = 1, \ldots, n$. Similarly, if A and B represent two interval matrices with n rows and m columns, $A \subset B$ if and only if $A_{ij} \subset B_{ij}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, m$.

4) **Midpoint of an interval vector:** the midpoint of an interval vector is defined as the real vector whose components are the midpoints of the corresponding components of the interval vector.

5) **Midpoint of an interval matrix:** the midpoint of an interval matrix is defined as the real matrix whose elements are the midpoints of the corresponding elements of the interval matrix.

6) **Width of an interval vector:** the width of an interval vector is defined as the real vector whose components are the widths of the corresponding components of the interval vector.

7) **Width of an interval matrix:** the width of an interval matrix is defined as the real matrix whose elements are the widths of the corresponding elements of the interval matrix.

8) **Regular interval matrix:** an interval matrix A^I is said to be *regular* if every real matrix $A \in A^I$ is nonsingular. Otherwise, the interval matrix is said to be *irregular*.

INTLAB comes incorporated with two different algorithms to multiply matrices with thick intervals. One built for speed and the other for sharpness. The latter one was chosen using the command *intvalinit*.

Chapter 4 | Static Global Optimization

Weiwei Chu in her Master of Science thesis: *Interval Analysis Applied to Re-Entry Flight Trajectory Optimization* (see [Chu, 2007]) proved, to a certain extent, the suitability of interval analysis for finding guaranteed bounds on the global solution of static optimization problems, i.e., problems which do not involve dynamic equations. She did it using FORTRAN and the interval library INTLIB[3]. Her computer had a Pentium 4 processor running at 2.6 GHz and 512 MB of RAM.

The objective will now be to solve the same static optimization problems discussed in [Chu, 2007] with at least the same level of accuracy using less CPU time. Obviously, by accuracy one should always understand sharpness.

Although Chu was using FORTRAN, which in theory should be faster than MATLAB, and a faster processor, she was also using a computer with less RAM. This makes it hard to compare the forthcoming results with hers in terms of CPU time. Nevertheless, it will be assumed that substantial improvements[4] in computational time can only be justified by the developments made in the global optimization algorithm. Weighting all the pros and cons, the two configurations are too similar to justify significant differences in CPU time.

The static global optimization algorithm discussed in the forthcoming sections is based on [Hansen and Walster, 2004] and [Chu, 2007].

4.1 The problem

A static global optimization problem can be formulated in the following way: given a scalar function $f(x)$, hereafter designated as the *objective function*, with $x \in \mathbb{R}^n$, find the (global) minimum value of f, f^*, and the point or points where this minimum value occurs, x^*. Since maximizing $f(x)$ is the same as minimizing $-f(x)$, all maximization problems can be reformulated as minimization problems.

To start the interval algorithm, an initial box or boxes containing x^* must be given. If the initial boxes overlap and the minimum point occurs in more than one initial box, than the optimum solution will be independently calculated from each of these initial boxes, wasting valuable CPU time. This is the only inconvenience of prescribing several initial boxes. However, to simplify what follows, it is assumed that only one initial box is provided.

If no information whatsoever is known about the location of the minimum, the initial box, $X_{initial}$, can be defined component-wise as $\left[-\bar{N}, \bar{N}\right]$ where $\bar{N}$ is the largest floating point number representable by the computer. However, one should keep in mind that the wider the initial box, the longer it will take to calculate sharp results.

Since the search for the global minimum is restricted to the initial box, the problem is actually constrained:

[3] *More information about the interval standard function library INTLIB can be found in [Kearfott et al., 2007].*

[4] *What is a substantial improvement or not is something that will be discussed separately in each example.*

$$\begin{aligned} &\min f(x) \\ &subject\ to: x \in X_{initial} \end{aligned} \tag{4-1}$$

However, unless explicit equality and/or inequality constraints are given, problem (4-1) will be treated as unconstrained. In other words, it is assumed that the initial box is so wide that the minimum occurs in a stationary point of $f(x)$ and not in a nonstationary point on the boundary of $X_{initial}$.

Let inequality constraints be denoted by $p(x) \le 0$ with $p(x): \mathbb{R}^n \to \mathbb{R}^m$ and equality constraints by $q(x) = 0$ with $q(x): \mathbb{R}^n \to \mathbb{R}^r$. Then, the static global constrained optimization problem with equality and inequality constraints is formulated as follows:

$$\begin{aligned} &\min f(x) \\ &subject\ to: x \in X_{initial} \\ &\qquad p(x) \le 0 \\ &\qquad q(x) = 0 \end{aligned} \tag{4-2}$$

4.2 The algorithm

The support column of the developed static global optimization algorithm is a common *branch and bound* (BB) procedure. As explained in [Wikipedia, 2007], the general branch and bound method can be divided in two main parts, one executed after the other:

1) *Branching* (or *splitting*): each remaining box is divided in two or more subboxes;

2) *Bounding* and *pruning*: for each subbox X_i, bounds on $f(X_i) = \left[\underline{f_i}, \overline{f_i}\right]$ are calculated. Let $\overline{f}$ denote the smallest calculated upper bound (this notation will be kept throughout the text). Since $f^* < \overline{f}$, any box wherein $\underline{f_i} > \overline{f}$ can be safely deleted[5].

If no initial upper bound on f^* is known, $\overline{f}$ is set to $+\infty$ before the algorithm is started.

In addition to the upper bound on the minimum, other techniques for deleting boxes are used in the developed algorithm. These techniques are explained one by one in Section 4.2.2. The developed algorithm also has procedures capable of shrinking the boxes, i.e., capable of reducing their width. In the limit, these latter procedures can even delete entire boxes. They are treated in Section 4.2.1.

A major difference between the current algorithm and the algorithm given in [Hansen and Walster, 2004] is that while in [Hansen and Walster, 2004] only one box is processed at a time, here, all the boxes are processed at the same time. For example, while [Hansen and Walster, 2004] would apply hull consistency and box consistency to the same box before moving to another one, here, hull consistency is applied to all the boxes and then box consistency is applied to all the remaining boxes. This is done to take advantage of fact that interval vector and interval matrix operations are very fast in INTLAB, as highlighted in [Rump,

[5] *This is true for unconstrained problems. For constrained ones, further considerations need to be made. Section 4.2.2.2 deals with this issue.*

2007]. Also in [Rump, 2007], it is said that loops may slow down the system considerably, which supports the idea that the boxes should not be processed one at a time. Nevertheless, further investigation about which is the best approach is required.

The overall structure of the developed algorithm is quite straightforward. It is mainly a big cycle where the first step is always branching, followed by different, independent and contained modules, each implementing a different procedure designed to shrink and/or delete boxes. The order in which each module appears inside the cycle was roughly chosen to optimize speed. For example, fast and efficient procedures like hull consistency, i.e., that delete a lot of boxes and are relatively cheap in terms of CPU time, were placed higher in the cycle, when there are still a lot of boxes being processed simultaneously. Slower and less efficient procedures, like box consistency, were thrown to lower positions.

The order suggested in **Figure 4.1**, where a flowchart describing the static global optimization algorithm is given, is probably not the best possible way of ordering the different modules. It is just a gross attempt to optimize speed based on a simple trial and error procedure. It is dubious if trying to find an optimum sequence is even worth the effort. As many other aspects of interval analysis, the best way of ordering the modules is in all probability problem dependent, making it impossible to find an unique sequence yielding the best results in every case. Instead, for each problem, a reasonable sequence should be sought.

Only some of the procedures mentioned in **Figure 4.1** are used to solve a particular problem. As a matter of fact, no example solved in the forthcoming sections has exactly the same flowchart as the one represented in **Figure 4.1**. Most of the times, a particular method is not applied to a particular problem simply because it is not defined for that particular problem. Other times, by introducing a determined procedure, more CPU time is required to reach the same accuracy. What methods are used in a particular example is something that will be treated later on when discussing the examples separately. At the same time, implementation details will be given for each method.

The good thing about this modular approach is that modules can be inserted and removed from the optimization algorithm without affecting the remaining code. In particular, if a new procedure for shrinking and/or deleting boxes is developed, it is straightforward to integrate it into the existing code.

4.2.1 Procedures to shrink the boxes

In this section, two procedures capable of shrinking the boxes are described and compared: hull consistency and box consistency. In the limit, these procedures can even delete entire boxes.

4.2.1.1 Hull consistency

The following description of *hull consistency* (HC) is based on [Hansen and Walster, 2004]. Especial care will be taken in describing it since it is one of the techniques that most contribute for the forthcoming results.

Start by considering a general function $f(x)$ with a scalar variable x. This function should not be confused with the objective function. Actually, in static optimization problems, $f(x)$ will always be an equality constraint. Assume $f(x)$ can be written as:

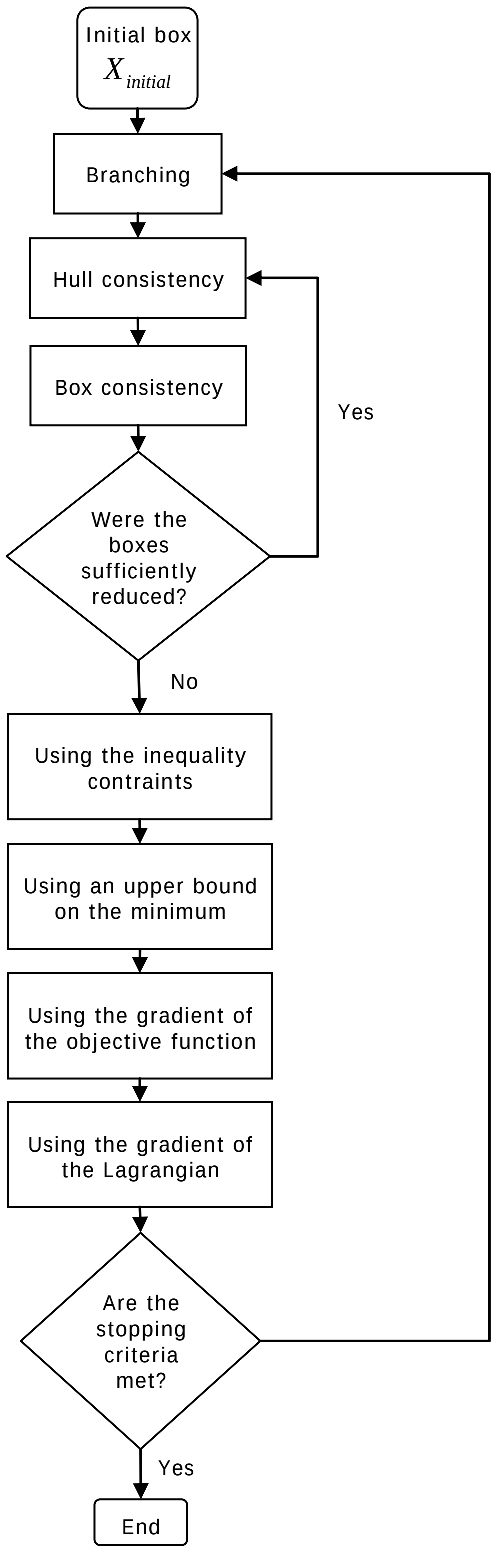
Initial box
$X_{initial}$
Branching
Hull consistency
Box consistency
Yes
Were the boxes sufficiently reduced?
No
Using the inequality contraints
Using an upper bound on the minimum
Using the gradient of the objective function
Using the gradient of the Lagrangian
Are the stopping criteria met?
No
Yes
End

Figure 4.1: *Flowchart of the developed static global optimization algorithm.*

$$f(x) = g(x) - h(x) \tag{4-3}$$

and that the equation $f(x) = 0$ needs to be solved. If x^* is a solution of $f(x) = 0$, then:

$$x^* = g^{-1}\left(h\left(x^*\right)\right) \tag{4-4}$$

If an interval X containing x^* is known, then:

$$x^* \in g^{-1}\left(h\left(X\right)\right) \tag{4-5}$$

Relation (4-5) is a consequence of the fundamental theorem of interval analysis. Therefore, the following iterative process can be defined:

$$\begin{aligned} X' &= g^{-1}\left(h\left(X\right)\right) \\ X'' &= X \bigcap X' \end{aligned} \tag{4-6}$$

If x^* is included in the initial interval X, then (4-5) guarantees that x^* will also be included in X''. This way, the solution will never be eliminated. On the other hand, if X'' is smaller than X, which is normally the case, (4-6) defines a way to sharpen the bounds on x^*. If $X'' = \varnothing$, then X was eliminated and it has been proved that $x^* \notin X$.

HC can be more or less efficient depending on what function $g(x)$ is chosen and how dependency affects the calculation of $h(X)$.

Normally, there are so many alternatives for $g(x)$ that it is hard to choose one. The suggestion given by [Hansen and Walster, 2004] is that $g(x)$ should be the term of $f(x)$ that dominates the other terms for some portion of X. For example, take $f(x) = x^4 + x^3 + x^2 + x$. If the objective is to delete boxes where $|x|$ is large, then the term x^4 should be chosen as $g(x)$. Conversely, if the objective is to delete boxes where $|x| < 1$, then the dominant term is x and $g(x)$ should be chosen as such.

Any values of the range of $h(X)$ that are outside the domain of g^{-1} are deleted when calculating $g^{-1}\left(h\left(X\right)\right)$. INTLAB can be configured to do this automatically by means of the command *intvalinit('RealStdFctsExcptnIgnore')*.

Consider now a system of m equations in n unknowns: $f(x): \mathbb{R}^n \to \mathbb{R}^m$. HC is applied to one equation and one variable at a time. All the remaining variables are replaced by their interval bounds. For example, applying HC to the i-th equation and to the j-th variable results in:

$$q\left(x_j\right) = f_i\left(X_1, ..., x_j, ..., X_n\right) = 0 \tag{4-7}$$

All the previous comments now hold for $q\left(x_j\right)$.

Which is the best way to cycle between the equations and the variables is still a matter being researched. The following procedure is suggested in [Hansen and Walster, 2004]: order the equations in some arbitrary way. Then, cycle through the variables in such a way that each variable is solved once for each equation. When using a particular equation, the variable with the smallest index that has not yet been used with the aforementioned equation should be chosen.

Last but not the least, HC can be used to prove the existence of a solution of a system of nonlinear equations inside a box X. This result is extremely important for Section 4.2.2.2. It is expressed by the following theorem:

Theorem 4.1: *Let the components of a function $f(x):\mathbb{R}^n \to \mathbb{R}^n$ be of the form:*

$$f_i(x) = g_i(x) - h_i(x) = 0 \quad ,i = 1,\dots,n \tag{4-8}$$

Suppose a new box X' is computed from X using:

$$X_i' = g_i^{-1}\left[h_i(X)\right] \quad ,i = 1,\dots,n \tag{4-9}$$

Then, if $X' \subset X$, there is a solution of $f(x)=0$ in X (and also in X').

For a proof, the reader is referred to Section 10.12 of [Hansen and Walster, 2004].

Expression (4-9) states that each new component of X' is calculated using the initial box X. In practice, as soon as a new component X_i' is calculated, it is used to calculate X_{i+1}'. **Theorem 4.1** can be proved using this more efficient form.

It is said that a box X is *feasible*, if there is at least one solution of $f(x)=0$ inside X.

4.2.1.2 Box consistency

Due to the reasons explained in Section 4.2.1.3, *box consistency* (BC) plays a secondary role in the developed algorithm when compared to HC. For this reason, BC will not be as thoroughly described as HC. For a more deep discussion about BC, the reader is referred to Section 10.2 of [Hansen and Walster, 2004].

Consider a function $f(x):\mathbb{R}^n \to \mathbb{R}$. The intention is to find a solution to $f(x)=0$ inside a box X. Start by replacing all the variables except one by their interval bounds:

$$q(x_i) = f(X_1,\dots,x_i,\dots,X_n) = 0 \tag{4-10}$$

The idea behind BC is that if $0 \notin q(X_i')$, where X_i' is a subinterval of X_i, then the subbox $X' = (X_1,\dots,X_i',\dots,X_n)$ does not contain a solution of $f(x)=0$ and can be safely deleted, shrinking the initial box X.

How to choose the subinterval X_i' from X_i and how to delete subintervals of X_i' even when X_i' can not be entirely deleted, are some of the questions answered in [Hansen and Walster, 2004].

4.2.1.3 Comparing hull consistency with box consistency

Both consistency methods are designed to shrink the region of search of a particular problem. However, they have different virtues and drawbacks. The following analysis refers only to the HC and BC algorithms given in [Hansen and Walster, 2004] and is not applicable to other HC and BC algorithms available in literature.

In order for BC to be applied, the function $f(x)$ needs to be continuously differentiable, while for HC, $f(x)$ can even be discontinuous. Another even more important advantage of HC is that it reaches its result faster.

Nevertheless, sharper bounds can usually (but not always) be achieved with BC, if enough iterations are allowed.

A unique characteristic of HC is its ability to isolate and bound different solutions of $f(x)=0$ that might exist in a given box. BC is merely able to reduce the region of search to a box containing all the solutions.

A few tests were done to see if implementing BC after HC was worth the extra programming. In every test, to achieve the same level of accuracy, HC by itself was faster. As a consequence, BC was not implemented in the optimization algorithm. However, the author is aware that this decision is based on a finite set of tests and that a different set could yield different results.

4.2.2 Procedures to eliminate boxes

In this section, four techniques for deleting boxes are described. They are based on:

1) the inequality constraints;

2) an upper bound on the minimum;

3) the gradient of the objective function;

4) the gradient of the Lagrangian.

4.2.2.1 Using the inequality constraints

Consider an inequality constraint $p(x) \leq 0$, where x is a real vector with n components. Now, assume that $p(x)$ is evaluated over a box X using the interval arithmetic operations defined in Section 3.4, resulting in $p(X)=\left[\underline{p}(X), \overline{p}(X)\right]$. It is said that:

1) X is (*certainly) feasible* if $\overline{p}(X) \leq 0$ and that X is *(certainly) strictly feasible* if $\overline{p}(X) < 0$.

2) X is (*certainly) infeasible* if $\underline{p}(X) > 0$.

Any box X that is certainly infeasible can be safely deleted.

Alternatively, instead of evaluating $p(X)$ to test if X is infeasible, HC can be used to solve $p(X) = [-\infty, 0]$, which is equivalent to $p(X) \leq 0$. With essentially the same amount of computing needed to test for infeasibility, HC yields the same or more information. While a feasibility test can only delete or not delete X, HC can also reduce the width of X. The only setback is that extended interval analysis must be used to solve $p(X) = [-\infty, 0]$. As already mentioned, INTLAB does not support extended interval analysis.

4.2.2.2 Using an upper bound on the minimum

How an upper bound on the global minimum can be used to delete boxes was already explained in Section 4.2 when talking about the bounding and pruning phase of a general branch and bound algorithm. While the basic idea is always the same, each kind of constraint needs to be treated in a special way. In the end, everything comes down to the fact that $\overline{f}$ can only be updated using a box X, after a feasible solution point with respect to the inequality and/or equality constraints has been proved to exist inside X. There are four different cases that need to be treated independently: the unconstrained case, the inequality constrained case, the equality constrained case and the inequality and equality constrained case.

4.2.2.2.1 The unconstrained case

Let X_i be a box generated during the branching phase of the algorithm. In the unconstrained case, every point $x \in X_i$ is a feasible point due to the lack of constraints. Hence, instead of evaluating $f(x)$ over the entire box X_i to calculate an upper bound on the minimum, resulting in $f(X_i) = \left[\underline{f_i}, \overline{f_i}\right]$, a single point of X_i can be used for that effect. As in [Hansen and Walster, 2004], the midpoint of X_i is used in this thesis:

$$f\left(m(X_i)\right) = \left[\underline{f}_{i,mid}, \overline{f}_{i,mid}\right] \qquad \textit{(4-11)}$$

Since:

$$\overline{f}_{i,mid} \leq \overline{f_i} \qquad \textit{(4-12)}$$

a sharper upper bound on the minimum is obtained. This means that in general more boxes will be deleted.

4.2.2.2.2 The inequality constrained case

When there are only inequality constraints, before evaluating $f\left(m\left(X_i\right)\right)=\left[\underline{f}_{i,mid},\overline{f}_{i,mid}\right]$, it has to be guaranteed that $m\left(X_i\right)$ is a certainly feasible (thin) box, i.e., it has to be guaranteed that:

$$\overline{p}\left(m\left(X_i\right)\right)\le 0 \qquad \textit{(4-13)}$$

If (4-13) holds, then $\overline{f}$ can be updated using (4-11).

4.2.2.2.3 The equality constrained case

Dealing with equality constraints is relatively harder than dealing with inequality constraints. Suppose a box X_i is known to be a feasible box with regard to the equality constraints, i.e., a solution x of $q\left(x\right)=0$ is known to exist inside X_i. Then, an upper bound on the minimum is the supremum of $f\left(X_i\right)=\left[\underline{f_i},\overline{f_i}\right]$. Note that evaluating $f\left(m\left(X_i\right)\right)=\left[\underline{f}_{i,mid},\overline{f}_{i,mid}\right]$ and taking $\overline{f}_{i,mid}$ as an upper bound on the minimum is no longer valid since $m\left(X_i\right)$ might not be (and probably is not) a feasible box. This comes from the fact that the actual position of the solution of $q\left(x\right)=0$ is not known. The only undisputable fact is that $x\in X$. Therefore, $f\left(x\right)$ must be evaluated over the entire box X_i.

The existence of a solution inside a box can be proven using **Theorem 4.1**. However, **Theorem 4.1** requires the number of variables, n, to be same as the number of equality constraints, r. In general, that is not the case. If $n=r$, then the minimization problem becomes somewhat awkward, since in principle, the r equality constraints can be solved with respect to the n variables. Thus, the most common situation is $r<n$.

Therefore, to apply **Theorem 4.1**, $n-r$ variables must be fixed. As in [Hansen and Walster, 2004], they are made equal to the midpoints of the respective intervals. An intricate method for deciding which are the best variables to fix is given in Section 15.5 of [Hansen and Walster, 2004]. Since the problems solved in this thesis are relatively simple, these variables will be selected manually. However, it should be highlighted that properly choosing which variables to fix can significantly change the results, as will become apparent when solving **Example 4** in Section 4.6.

Denote Z_i as the box obtained from X_i after fixing $n-r$ variables of X_i. If **Theorem 4.1** is satisfied, that means that a box $Z_i^{'}\subset Z_i$ containing a feasible solution is known. Then, to get the smallest possible upper bound on f^*, $f\left(Z_i^{'}\right)$ should be evaluated . Evaluating $f\left(Z_i\right)$ or $f\left(X_i\right)$ would not be as efficient since both Z_i and X_i are wider than $Z_i^{'}$. Note, however, that a box Y_i can only be deleted if $\underline{f\left(Y_i\right)}>\overline{f}$, i.e., $f\left(x\right)$ must be evaluated over the entire box Y_i.

4.2.2.2.4 The inequality and equality constrained case

When a problem has simultaneously inequality and equality constraints, the following two conditions must be proven:

1) X_i must be a feasible box with regard to the equality constraints. This can be proven using the method described in Section 4.2.2.2.3.

2) The inequality constraints must be satisfied over the entire box X_i, i.e., X_i must be a certainly feasible box.

If both conditions are met, $\overline{f_i}$, resulting from $f(X_i)=\left[\underline{f_i}, \overline{f_i}\right]$, is an upper bound on the global minimum.

4.2.2.3 Using the gradient of the objective function

If the objective function $f(x):\mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and the problem is unconstrained, then the gradient function of $f(x)$, $g(x):\mathbb{R}^n \to \mathbb{R}^n$, is equal to zero at x^*. Of course, $g(x)$ is also equal to zero at local minima, local maxima and saddle points.

Consider a box X produced during the branching phase of the algorithm. If $0 \notin g(X)$, then X cannot contain the global minimum and can be safely deleted. Evaluating $g(X)$ is a simple task with INTLAB since a first-order automatic differentiation routine is provided. Thus, the analytical form of $g(x)$ does not even need to be calculated.

However, as already mentioned in Section 4.2.2.1, applying HC to the equation $g(X)=0$ is generally better than simply evaluating $g(X)$. With essentially the same amount of computing, HC has the advantage of sometimes being able to reduce the width of X.

In this thesis, HC is not applied to the equation $g(X)=0$. $g(X)$ is evaluated instead. Further upgrading the algorithm was deemed unnecessary: every time the gradient of the objective function could be applied to solve a specific problem, the desired levels of accuracy were achieved in less than one second. Nonetheless, HC should be used as a rule.

4.2.2.4 Using the gradient of the Lagrangian

According to [Visser, 2007] and [Bryson Jr. and Ho, 1975], to analytically solve the constrained static optimization problem formulated in (4-2), the so-called *Lagrangian function* of the problem must be formed:

$$L(x,\lambda_e,\lambda_i)= f(x)+\lambda_e^T q(x)+\lambda_i^T p(x) \qquad \textit{(4-14)}$$

where λ_e and λ_i are the so-called *Lagrangian multipliers*. L is sometimes also called the *augmented objective function*, for obvious reasons. Necessary conditions for a local solution of (4-2) are then:

$$\frac{\partial L(x,\lambda_e,\lambda_i)}{\partial x}=\frac{\partial f(x)}{\partial x}+\lambda_e^T\frac{\partial q(x)}{\partial x}+\lambda_i^T\frac{\partial p(x)}{\partial x}=0 \tag{4-15}$$

$$q(x)=0 \tag{4-16}$$

$$p(x)\le 0 \tag{4-17}$$

$$\begin{cases}\lambda_{i_j}\ge 0 & ,p_j(x)=0\\ \lambda_{i_j}=0 & ,p_j(x)<0\end{cases}\quad ,j=1,\dots,m \tag{4-18}$$

Conditions (4-15) through (4-18) are known as the *first-order Kuhn-Tucker conditions* [Visser, 2007]. The j-th inequality constraint is said to be *active* if $p_j(x^*)=0$ and *inactive* if $p_j(x^*)<0$.

Solving these optimality conditions is not an easy task. It would require an interval algorithm capable of solving a system of nonlinear equations. If this algorithm was available, it could be used as another procedure for shrinking the boxes. For a step by step procedure on how to implement one, the reader is referred to [Hansen and Walster, 2004].

However, a procedure for deleting boxes can still be devised using the first-order Kuhn-Tucker conditions. It can only be applied to problems with a single inequality constraint ($m=1$) and no equality constraints ($r=0$). It is based on a similar procedure described in [Chu, 2007]. The condition that m should be equal to 1 and that r should be equal to zero is never made in [Chu, 2007]. This should be taken as a lapse.

Rewriting the first order Kuhn-Tucker conditions for $m=1$ and $r=0$ yields:

$$\frac{\partial L(x,\lambda_i)}{\partial x}=\frac{\partial f(x)}{\partial x}+\lambda_i\frac{\partial p(x)}{\partial x}=0 \tag{4-19}$$

$$p(x)\le 0 \tag{4-20}$$

$$\begin{cases}\lambda_i\ge 0 & ,p(x)=0\\ \lambda_i=0 & ,p(x)<0\end{cases} \tag{4-21}$$

where λ_i is now a scalar. Note that expression (4-19) is the same as saying the vectors $\frac{\partial f(x)}{\partial x}$ and $\frac{\partial p(x)}{\partial x}$ must be parallel and have different directions.

Consider a box X. The objective is to devise methods capable of proving that $x^*\notin X$, so X can be safely deleted. Three such methods can be derived from (4-19)-(4-21):

1) If $p(X) > 0$, condition (4-20) is disrespected and X can be immediately deleted. This is the case already discussed in Section 4.2.2.1.

2) If $0 \in p(X)$, the Lagrange multiplier λ_i can be calculated through (4-19) as the intersection of the following intervals:

$$\lambda_i^j(X) = \left(- \frac{\partial f(x)}{\partial x_j} \Big/ \frac{\partial p(x)}{\partial x_j} \right) \Bigg|_{x=X} \quad , j = 1, \ldots, n \tag{4-22}$$

If $\lambda_i(X) = \varnothing$, then X can be safely deleted as (4-19) can not be satisfied. In this case, proving the existence of a solution of $p(x) = 0$ in X is not required. If $x^* \in X$, then λ_i has to be at least a point interval. It can never be an empty set. Therefore, there is no danger of deleting the optimal solution when $\lambda_i(X) = \varnothing$.

Also, if $\lambda_i(X) < 0$, X can be safely deleted as a result of (4-21). Again, proving the existence of a solution of $p(x) = 0$ in X is not necessary. If $x^* \in X$, then $\lambda_i(X) < 0$ is impossible, meaning that the optimal solution is always preserved.

3) The third and final possibility is $p(X) < 0$. In this case, $\lambda_i = 0$ from (4-21). Applying this result to (4-19) leads to:

$$\frac{\partial f(x)}{\partial x} = 0 \tag{4-23}$$

X can be safely deleted if $0 \notin \left. \frac{\partial f(x)}{\partial x} \right|_{x=X}$.

4.2.3 Branching

The branching phase of the algorithm has been mentioned many times but its implementation has not yet been explained. Suppose the objective is to split a box X_i with n components. The algorithm allows the user to specify which components of X_i should be split and in how many subintervals each one of those components should be divided. All the boxes are then split according to these criteria.

For simplicity's sake, assume each component is divided into N_{split} subintervals and that n_{split} components of X are split. Then, for every old box, ${N_{split}}^{n_{split}}$ subboxes will be generated. Even if $N_{split} = 2$ (the smallest possibility), dividing three components of X would result in eight new subboxes. According to [Hansen and Walster, 2004], and to the author's experience, eight new subboxes per old box is the maximum a normal personal computer can handle. This should be taken as a rule of thumb.

In [Hansen and Walster, 2004], when only consistency methods are used to reduce the width of the boxes (which is the case), the branching is performed by always dividing the three widest components into two subintervals each. However, they assume that this can be a poor choice. In fact, no procedure exists today telling which is the best way to split the remaining boxes. As many other things about interval analysis, it is still something very problem dependent and under investigation.

Therefore, to allow the user to experiment different branching configurations, the choice was made not to implement a rigid splitting procedure.

4.2.4 Interpretation of the results

As important as calculating results is interpreting them. Not taking this into account is probably the biggest shortcoming of [Chu, 2007], as will become apparent in the following sections.

Consider the general case when more than one box remains after termination. What makes interval optimization algorithms so unique is their ability to give guaranteed bounds on (all) the solution point(s) of a minimization problem. Any interval optimization algorithm should be able to yield such bounds.

To simplify, it will be assumed that all the remaining boxes are *contiguous*, i.e., there are not any gaps inside the region defined by the remaining boxes. If this is not the case, the following results can be easily adapted.

At the end of the algorithm, the only certain fact is that the global minimum has to be contained inside one or more of the remaining boxes. In which ones, however, it is not known. Therefore, the only guaranteed fact is that:

$$\min_i\left(\inf\left(X_i^j\right)\right) \le x_j^* \le \max_i\left(\sup\left(X_i^j\right)\right) \tag{4-24}$$

where x_j^* is the j-th component of the solution point(s) and X_i^j represents the j-th component of the i-th remaining box. Expression (4-24) means that the solution point(s) of the minimization problem are undoubtedly contained inside the <u>hull</u> of the solution set.

As a consequence of (4-24), the guaranteed bounds on f^* are:

$$\underline{F} \le f^* \le \min\left(\overline{f}, \overline{F}\right) \tag{4-25}$$

where $\underline{F}$ and $\overline{F}$ are given by:

$$\underline{F} = \min_i\left(\underline{f}\left(X_i\right)\right) \tag{4-26}$$

$$\overline{F} = \max_i\left(\overline{f}\left(X_i\right)\right) \tag{4-27}$$

The reader might be asking why the upper bound on f^* is $\min\left(\overline{f}, \overline{F}\right)$. The reason is simple: imagine that the box last used to update $\overline{f}$ has been deleted (for example, by the using the gradient of the objective function), together with a substantial part of the initial search region.

In this case, in all probability, $\overline{F} \le \overline{f}$. If this is not the case, however, $\overline{f}$ is sure to be smaller than $\overline{F}$.

4.2.5 Stopping criteria

Now that all the steps of the algorithm have been explained, an important issue remains: how to stop the main cycle?

The developed algorithm has two different stopping criteria:

1) $$\max_i\left(\sup\left(X_i^j\right)\right)-\min_i\left(\inf\left(X_i^j\right)\right) \le \varepsilon_{X_j} \quad , j=1,\dots,n \tag{4-28}$$

where X_i^j represents the j-th component of the i-th box, and:

2) $$\min\left(\overline{f},\overline{F}\right)-\underline{F} \le \varepsilon_f \tag{4-29}$$

where $\underline{F}$ and $\overline{F}$ are given by (4-26) and (4-27), respectively. Both ε_{X_j} (n parameters) and ε_f (1 parameter) are given by the user.

As can be seen by comparing (4-28)-(4-29) with (4-24)-(4-25), if both stopping conditions are satisfied, the guaranteed bounds on x^* and f^* will be sharper than ε_X and ε_f, respectively. As a rule, however, if one of the stopping conditions is met, the main loop should be stopped. Why? Consider the objective function represented in **Figure 4.2**. The global solution point of this objective function is not really a point but an interval: $x^* = [-1,1]$. If both conditions were to be met simultaneously and if the given value of ε_X was smaller than $w\left(x^*\right)$, than the main cycle would run indefinitely.

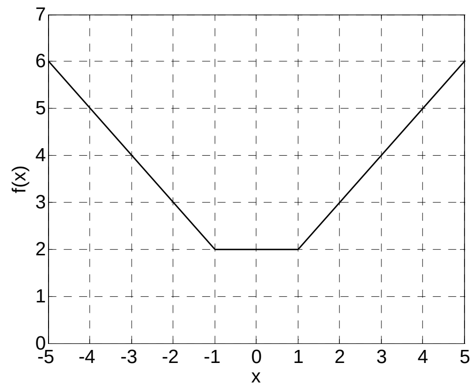

Figure 4.2: *Solution interval instead of a solution point.*

Even when satisfying (4-29) is enough to stop the cycle, some precautions should be taken when choosing ε_f. If ε_f is chosen so small that the wrapping effect and dependency prevent (4-29) from ever being fulfilled, one might end up with an infinite cycle anyway.

However, in cases where it is known from the beginning that the solution point is indeed a point and not an interval, it is preferable only to stop the cycle when both conditions are met.

In that case, after termination, the guaranteed bounds on x^* and f^* will be indubitably sharper than ε_X and ε_f. The stopping conditions suggested in [Hansen and Walster, 2004] are not able to give this guarantee.

Another advantage of requiring both stopping conditions to be met at the same time is that, by selecting ε_X or ε_f relatively large, the criterion that will effectively stop the algorithm can be chosen.

All the examples discussed in this chapter and in Chapter 5 have indeed a single solution point. Therefore, the main loop is only stopped when (4-28) and (4-29) are satisfied simultaneously.

Besides the main loop, the optimization algorithm also has an inner cycle, as can be seen in **Figure 4.1**. The objective of this cycle is to repeatedly apply HC and BC as long as their efficiency is sufficiently high. The inner loop is stopped when the following two conditions are met:

1) No box was deleted by the consistency methods.

2) The maximum (with respect to all the components of all the boxes) absolute width reduction of a component was smaller than ε_{abs} or the maximum relative width reduction of a component was smaller than ε_{rel}. The maximum relative width reduction is defined as the maximum absolute width reduction divided by the initial width of the maximally reduced component. The parameters ε_{abs} and ε_{rel} are both given by the user.

Increasing ε_{abs} and ε_{rel} has both virtues and drawbacks. Bigger values of ε_{abs} and ε_{rel} mean a quicker inner cycle, at least initially. However, as the number and width of the boxes at a particular iteration of the algorithm increase (since the consistency methods have less time to converge), the total time needed to achieve a predetermined accuracy might also increase. Therefore, choosing good values for ε_{abs} and ε_{rel} can be tricky and should be taken seriously.

Six static optimization problems will now be solved using the aforementioned algorithm.

4.3 Example 1

The first problem solved in [Chu, 2007] is an unconstrained problem taken from [Hansen and Walster, 2004]. Its formulation can be seen in **Table 4.1**:

Table 4.1: *Formulation of* ***Example 1****.*

Objective function	$f(x) = x^4 - 4x^2$
Initial box	$X_{initial} = [0,3]$
Solution point	$x^* = \sqrt{2} \approx 1.4142136$
Global minimum	$f^* = -4$

In **Figure 4.3**, a plot of the objective function for $x \in X_{initial}$ is shown.

The objective function can and should be rewritten as:

$$f(x) = x^4 - 4x^2 = x^2(x^2 - 4) \qquad (4\text{-}30)$$

to reduce the effect of dependency. This is not done in [Chu, 2007].

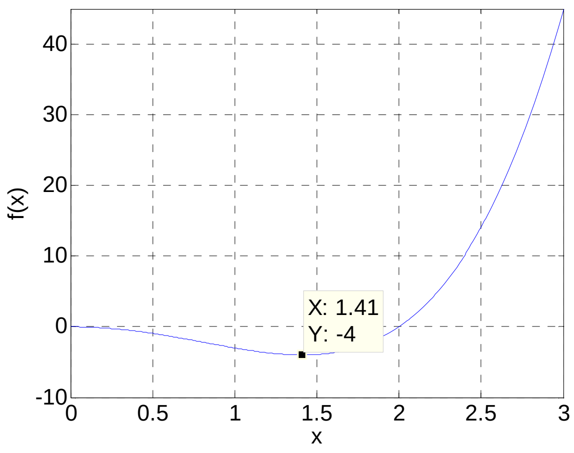

Figure 4.3: *Objective function of* ***Example 1****.*

The following list shows which modules were used to solve this particular problem. The modules are identified by their respective sections.

a) Section 4.2.2.2 - Using an upper bound on the minimum

b) Section 4.2.2.3 - Using the gradient of the objective function

The remaining modules can not be applied to this problem, as can be easily seen by the reader.

The inputs given to the algorithm, besides the objective function and the initial box, are listed in **Table 4.2**. The value of N_{split} was chosen rather arbitrarily. Note, however, that the choice of N_{split} can have a great influence on the algorithm's performance. More considerations about this will be given in **Example 6**.

Table 4.2: *User inputs in* ***Example 1****.*

Designation	**Symbol**	**Value**
Initial upper bound on the minimum	$\overline{f}$	$+\infty$
New boxes per old box	N_{split}	10
Stopping criteria	ε_X	3e-6
	ε_f	6.8e-5

The parameters ε_X and ε_f were chosen equal to the widths of the "guaranteed bounds" calculated in [Chu, 2007] for x^* and f^*. This guarantees that, after termination, the obtained results will be at least as sharp as the results obtained in [Chu, 2007]. The two algorithms can then be compared in terms of speed.

In **Table 4.3**, the results produced by the developed algorithm are compared with the results presented in [Chu, 2007].

Two different algorithms are used in [Chu, 2007] to solve this problem: a so-called *non-gradient method* and a *gradient method*. The only difference between them is the fact that, in the former, procedure **b)** is not used, while the gradient method uses both **a)** and **b)** (like done here). Therefore, the results from [Chu, 2007] listed in **Table 4.3** are the most accurate results given in [Chu, 2007] for the gradient method.

The developed algorithm wins both in terms of speed and accuracy. More importantly, $x^* = \sqrt{2} \approx 1,4142136$ is not correctly bounded by the solution interval taken from [Chu, 2007]. Despite being a serious mistake, the problem is probably not in the algorithm itself but in the interpretation of the results (at least, for this particular example). As discussed in Section 4.2.4, interval arithmetic guarantees that the global solution is unquestionably contained in the hull of the solution set. However, in [Chu, 2007], the interval *that gives the minimum cost function* is taken as the global solution. While it is unclear which interval is actually being picked, the simple fact that a single interval is taken as the global solution (instead of the hull of the solution set) justifies the wrong result.

Giving a single interval instead of the hull of the solution set also makes the results from [Chu, 2007] misleadingly sharp. If the true solution was given, the widths of F^* and X^* would almost certainly increase.

Table 4.3: *Comparative results for* ***Example 1****.*

		The developed algorithm		**Chu's algorithm (Gradient Method)**	
Designation	**Symbol**	**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	F^*	$[-4.00001, -3.99999]$	1.7e-6	$[-4.00004, -3.99996]$	6.8e-5
Bounds on the solution point	X^*	$[1.414213, 1.414214]$	6.0e-7	$[1.414216, 1.414219]$	3e-6
CPU time	$\Delta t_{CPU}\,[s]$	0.42	N/A	0.69	N/A
Number of remaining boxes	$N_{remaining}$	2	N/A	?[6]	N/A

As a side note, outward rounding was used to reduce the number of decimal cases of the lower and upper bounds given in **Table 4.3**. Conventional rounding was used for the widths. That is why the presented widths and intervals do not completely agree. The given values for the widths should be taken as a better approximation for the widths than the actual widths of the given intervals.

4.4 Example 2

Example 2 is also an unconstrained problem. It was taken from [Liberti, 2006]. The problem is stated in **Table 4.4**.

Table 4.4: *Formulation of* ***Example 2****.*

Objective function	$f(x) = 0.25x + \sin(x)$
Initial box	$X_{initial} = [-3, 6]$
Solution point	$x^* = -\cos^{-1}(-0.25) \approx -1.823477$
Global minimum	$f^* = f(x^*) \approx -1.424115$

[6] *? = Not given.*

A plot of the objective function for $x \in X_{initial}$ is shown in **Figure 4.4**.

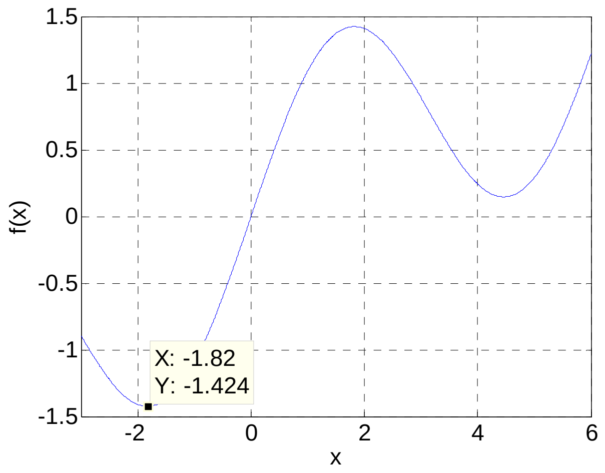

Figure 4.4: *Objective function of **Example 2**.*

Unlike before, the dependency problem can not be reduced now by using the sub-distributive law of multiplication.

The same modules used to solve **Example 1** were again used to solve **Example 2**. This should not come as a surprise since both **Example 1** and **Example 2** are unconstrained optimization problems. In **Table 4.5**, the inputs leading to the results listed in **Table 4.6** are given.

Table 4.5: *User inputs in **Example 2**.*

Designation	Symbol	Value
Initial upper bound on the minimum	$\overline{f}$	$+\infty$
New boxes per old box	N_{split}	10
Stopping criteria	ε_X	9.0e-6
	ε_f	4.6e-6

Table 4.6: *Comparative results for **Example 2**.*

		The developed algorithm		Chu's algorithm (Gradient Method)	
Designation	**Symbol**	**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	F^*	$[-1.4241153, -1.4241149]$	2.3e-7	$[-1.4241173, -1.4241127]$	4.6e-6
Bounds on the solution point	X^*	$[-1.823477, -1.823475]$	9.0e-7	$[-1.823484, -1.823475]$	9.0e-6
CPU time	$\Delta t_{CPU}\,[s]$	0.35	N/A	63.78	N/A
Number of remaining boxes	$N_{remaining}$	1	N/A	?	N/A

To get sharper results for X^* and F^*, the developed algorithm needs 0.35s, while [Chu, 2007] needs 63.78s. The algorithm presented here is approximately 181 times faster. The main reason for this is that while a BB approach is used here, [Chu, 2007] is simply dividing $X_{initial}$ in a high number of subintervals and then applying procedures **a)** and **b)** to all of them. Achieving sharp bounds with this latter technique is a lot more time demanding than with a BB approach.

Since the interval *that gives the minimum cost function* was still taken as the global solution in [Chu, 2007], the fact that here $x^* \in X^*$ must be understood as a mere coincidence.

4.5 Example 3

This example is relatively harder to solve than the two previous ones. It is a 2-dimensional problem with both inequality and equality constraints, as can be seen in **Table 4.7**. This problem was taken from [Hansen and Walster, 2004]. The analytical solution was calculated using the procedure described in Section 4.2.2.4. The steps for analytically calculating x^* are omitted since showing how to solve static optimization problems by hand is not the objective of this work. Mind, however, that the values given for x^* in [Chu, 2007] are wrong.

Note that the bounds on the objective function and on the inequality constraint cannot be affected by dependency since the variables appear only once in $f(x)$ and in $p(x)$.

Table 4.7: *Formulation of* ***Example 3****.*

Objective function	$f(x) = x_1$
Initial box	$X_{initial} = \begin{bmatrix} [-1,0] \\ [0,1] \end{bmatrix}$
Inequality constraints	$p(x) = x_1^2 + x_2^2 - 1 \le 0$
Equality constraints	$q(x) = x_1^2 - x_2 = 0$
Solution point	$x^* = \begin{bmatrix} -\sqrt{-0.5+0.5\sqrt{5}} \\ -0.5+0.5\sqrt{5} \end{bmatrix} \approx \begin{bmatrix} -0.78615138 \\ 0.61803399 \end{bmatrix}$
Global minimum	$f^* = -\sqrt{-0.5+0.5\sqrt{5}} \approx -0.78615138$

To computationally solve this problem, the following modules were used:

a) Section 4.2.1.1 - Hull consistency: applying the theory discussed in Section 4.2.1.1 to the equality constraint:

$$x_1^2 - x_2 = 0 \tag{4-31}$$

yields the following iterative subroutine:

$$X_1^{'} = -\sqrt{X_2} \tag{4-32}$$

$$X_1 = X_1 \cap X_1^{'} \tag{4-33}$$

$$X_2^{'} = X_1^2 \tag{4-34}$$

$$X_2 = X_2 \cap X_2^{'} \tag{4-35}$$

Only the negative root is considered in (4-32) since $X_{initial_1} = [-1, 0]$.

In [Chu, 2007], only (4-34) and (4-35) are used.

b) Section 4.2.2.1 - Using the inequality constraints: this is also done in [Chu, 2007].

c) SECTION 4.2.2.2 - Using the upper bound on the minimum: as discussed in Section 4.2.2.2.4, before using a box X to update $\overline{f}$, X must be proven to be a feasible box with respect to the equality and inequality constraints. Here, the details of how to do that for the equality constraint will be given (dealing with the inequality constraint is relatively easier).

Example 3 has a 2-dimensional objective function subject to a single equality constraint. That means that to apply **Theorem 4.1**, one of the two variables needs to be fixed. By arbitrarily choosing X_1, one ends up with the following procedure:

1) Make X_1 a thin interval: $X_{1,m} = m(X_1)$.

2) Calculate new bounds on X_2 using the equality constraint: $X_2^{'} = X_{1,m}^2$. Interval arithmetic must be used to calculate $X_2^{'}$. Because of rounding errors, $X_2^{'}$ will in general be an interval.

3) If $X_2^{'} \subset X_2$, then **Theorem 4.1** guarantees that there is a solution of (4-31) inside $X^{'} = \begin{bmatrix} X_{1,m} & X_2^{'} \end{bmatrix}^T$. Thus, $X^{'}$ can be used to update $\overline{f}$, provided that the inequality constraint is also satisfied for every $x \in X^{'}$.

In **Table 4.8** and in **Table 4.9**, respectively, the inputs and the outputs of the developed algorithm are given.

Table 4.8: *User inputs in* ***Example 3***.

Designation	Symbol		Value
Initial upper bound on the minimum	$\overline{f}$		$+\infty$
New boxes per old box	N_{split}		$[10 \quad 0]^T$
Inner loop stopping criteria	ε_{abs}		2
	ε_{rel}		1.01
Stopping criteria	ε_X	ε_{X_1}	1.0e-6
		ε_{X_2}	1.6e-5
	ε_f		1.0e-6

High values for the inner loop stopping criteria were intentionally chosen to optimize the computing time. Once again, while this is true for this particular example, it can not be generalized to other problems.

Note that N_{split} is now a column vector. Each component of N_{split} represents the number of subintervals in which each corresponding interval component of X is spit.

Note that in [Chu, 2007], the lower bound of F^* is bigger than the upper bound. Here, these bounds have been switched.

Once again, the developed algorithm was able to calculate sharper bounds in less time. However, the time difference is not as significant as in **Example 2**. This is explained by the fact that [Chu, 2007] is now also using a branch and bound approach.

Table 4.9: *Comparative results for* ***Example 3****.*

			The developed algorithm		Chu's algorithm (Gradient Method)	
Designation	**Symbol**		**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	F^*		$[-0.78615141, -0.78615135]$	5.0e-8	$[-0.786150, -0.786151]$	1.0e-6
Bounds on the solution point	X^*	X_1^*	$[-0.78615141, -0.78615130]$	1.0e-7	$[-0.786151, -0.786150]$	1.0e-6
		X_2^*	$[0.618033, 0.618035]$	1.6e-7	$[0.618317, 0.618334]$	1.6e-5
CPU time	$\Delta t_{CPU}\,[s]$		0.45	N/A	0.73	N/A
Number of remaining boxes	$N_{remaining}$		1	N/A	?	N/A

The developed algorithm gives different upper bounds for F^* and X_1. This may seem strange since $f(x) = x_1$. However, there is nothing strange about it. As explained in Section 4.2.4, the bounds on f^* are calculated using expression (4-25). What is happening here is simply that $\overline{f} < \overline{F}$, and therefore $\overline{f}$ is taken as the upper bound on f^*. Moreover, in this particular example, X_1^* could be made sharper by equalizing it to F^*. However, since this is something that cannot be done in general, this was not done here.

As a final remark, the reader is asked to check that $x^* \notin X^*$, where X^* is the result calculated in [Chu, 2007]. Numerous errors were found in Chu's algorithm:

1) As mentioned before, the interval *that gives the minimum cost function* is taken as the global solution, instead of the hull of the solution set.

2) The boxes are not checked for feasibility with respect to the equality constraints.

3) The procedure described in Section 4.2.2.4 about using the gradient of the Lagrangian when there is only one inequality constraint is applied here, even though an equality constraint exists as well.

Any one of these mistakes is reason enough for $x^* \notin X^*$.

4.6 Example 4

Example 4 consists on a 4-dimensional objective function with two equality constraints, as can be seen in **Table 4.10**. This problem and its analytical solution were taken from [Hansen and Walster, 2004].

Note that the bounds on the objective function cannot be affected by dependency since x_1 only appears once in $f(x)$.

Table 4.10: *Formulation of* ***Example 4****.*

Objective function	$f(x) = -x_1$
Initial box	$X_{initial} = \begin{bmatrix} [-1.1, -0.7] \\ [-1.2, 1] \\ [0, 2] \\ [0, 1.6] \end{bmatrix}$
Equality constraints	$x_1^3 - x_2 + x_3^2 = 0$ $x_1^2 - x_2 - x_4^2 = 0$
Solution point	$x^* = \begin{bmatrix} -0.7 & 0.49 & \sqrt{0.833} \approx 0.912688 & 0 \end{bmatrix}^T$
Global minimum	$f^* = 0.7$

To computationally solve this problem, the following modules were used:

a) Section 4.2.1.1 - Hull consistency: applying the theory discussed in Section 4.2.1.1 to the equality constraints:

$$x_1^3 - x_2 + x_3^2 = 0 \tag{4-36}$$

$$x_1^2 - x_2 - x_4^2 = 0 \tag{4-37}$$

yields the following iterative subroutine:

$$X_1^{'} = -\left(-X_2 + X_3^2\right)^{1/3} \tag{4-38}$$

$$X_1 = X_1 \cap X_1^{'} \tag{4-39}$$

$$X_1^{'} = -\sqrt{X_2 + X_4^2} \tag{4-40}$$

$$X_1 = X_1 \cap X_1^{'} \tag{4-41}$$

$$X_2^{'} = X_1^3 + X_3^2 \tag{4-42}$$

$$X_2 = X_2 \cap X_2^{'} \tag{4-43}$$

$$X_2^{'} = X_1^2 + X_4^2 \tag{4-44}$$

$$X_2 = X_2 \cap X_2^{'} \tag{4-45}$$

$$X_3^{'} = \sqrt{X_2 - X_1^3} \tag{4-46}$$

$$X_3 = X_3 \cap X_3^{'} \qquad (4\text{-}47)$$

$$X_4^{'} = \sqrt{X_1^2 - X_2} \qquad (4\text{-}48)$$

$$X_4 = X_4 \cap X_4^{'} \qquad (4\text{-}49)$$

Expressions (4-38)-(4-49) were written taking into consideration that $X_{initial_1} \leq 0$, $X_{initial_3} \geq 0$ and $X_{initial_4} \geq 0$.

b) Section 4.2.2.2 - Using the upper bound on the minimum: as discussed in Section 4.2.2.2.3, before using a box X to update $\overline{f}$, X must be proven to be a feasible box with respect to the equality constraints.

Example 4 has a 4-dimensional objective function subject to two equality constraints. This means that in order to apply **Theorem 4.1**, two of the four variables need to be fixed.

The chances that $X^{'} \subset X$ are enhanced if $X^{'}$ is much smaller than X. Taking this simple remark into account, one can see that by substituting X_1 and X_2 by their midpoints, expressions (4-46) and (4-48) yield point intervals for $X_3^{'}$ and $X_4^{'}$, apart from rounding errors. This makes it highly probable that $X_3^{'} \subset X_3$ and $X_4^{'} \subset X_4$. The following steps for proving the existence of a solution of the equality constraints inside a box X can then be devised:

1) Substitute X_1 by $X_{1,m} = m(X_1)$ and X_2 by $X_{2,m} = m(X_2)$ in (4-46) and (4-48).

2) Calculate $X_3^{'}$ and $X_4^{'}$.

3) If $X_3^{'} \subset X_3$ and $X_4^{'} \subset X_4$, then **Theorem 4.1** guarantees that there is a solution of (4-36) and of (4-37) inside $X^{'} = \begin{bmatrix} X_{1,m} & X_{2,m} & X_3^{'} & X_4^{'} \end{bmatrix}^T$. Thus, $X^{'}$ can be used to update $\overline{f}$.

The inputs and the outputs of the developed algorithm are given in **Table 4.11** and in **Table 4.12**, respectively.

Note that only the first and second component of X are split during the branching phase. The algorithm relies exclusively on hull consistency to reduce the widths of X_3 and X_4.

Some remarks should be given about the algorithm used by [Chu, 2007] to solve this problem:

1) The branch and bound technique was not used. This is the main reason why the developed algorithm only needs about 5% of the time spent by Chu's algorithm to achieve sharper results.

2) HC is also applied to some extent in [Chu, 2007]. Expressions (4-46) through (4-49) are used but there is no inner loop. This makes it harder for [Chu, 2007] to calculate accurate results.

3) The boxes are not checked for feasibility with respect to the equality constraints.

4) In this example, the so-called non-gradient method is used instead of the gradient method. The different name only means that neither the gradient of the objective function nor the gradient of the Lagrangian are used to eliminate boxes. Indeed, neither of them can be used in this example.

Table 4.11: *User inputs in* ***Example 4****.*

Designation	Symbol		Value
Initial upper bound on the minimum	$\overline{f}$		$+\infty$
New boxes per old box	N_{split}		$\begin{bmatrix}2 & 2 & 0 & 0\end{bmatrix}^T$
Inner loop stopping criteria	ε_{abs}		1e-3
	ε_{rel}		5e-2
Stopping criteria	ε_X	ε_{X_1}	4e-4
		ε_{X_2}	2.2e-4
		ε_{X_3}	4.4e-4
		ε_{X_4}	9.0e-3
	ε_f		4e-4

Table 4.12: *Comparative results for* ***Example 4****.*

			The developed algorithm		Chu's algorithm (Non-Gradient Method)	
Designation	Symbol		Result	Width	Result	Width
Bounds on the minimum	F^*		$[0.6999, 0.7001]$	6.9e-5	$[0.7000, 0.7004]$[7]	4e-4
Bounds on the solution point	X^*	X_1^*	$[-0.7001, -0.6999]$	8.9e-5	$[-0.7004, -0.7000]$	4e-4
		X_2^*	$[0.48999, 0.49013]$	1.2e-4	$[0.49048, 0.49070]$	2.2e-4
		X_3^*	$[0.91268, 0.91283]$	1.4e-4	$[0.91295, 0.91340]$	4.4e-4
		X_4^*	$[0.0000, 0.0058]$	5.8e-3	$[0.0000, 0.0090]$	9.0e-3
CPU time	$\Delta t_{CPU}\ [s]$		48	N/A	1629 (≈27m)	N/A
Number of remaining boxes	$N_{remaining}$		6	N/A	?	N/A

5) The interval *that gives the minimum cost function* is taken as the global solution, instead of the hull of the solution set. This together with 3) implies that, in all probability, the global solution will not be correctly bounded. Indeed, one can see that $x^* \notin X^*$ in [Chu, 2007].

6) Three different solutions are given in [Chu, 2007] for this example. The one that gives the sharpest intervals for X_2^*, X_3^* and X_4^* was chosen.

Coming back to the algorithm developed during this thesis, one can see that the widths of X_3^* and especially of X_4^* are bigger than the widths of X_1^* and X_2^*. This is the price one has to

[7] *The interval actually given is* $[-0.7004, -0.7000]$*, probably due to a typing mistake.*

pay for only splitting X_1 and X_2 during the branching phase. Other ways of splitting the boxes were tested, namely:

i) $N_{split} = \begin{bmatrix} 0 & 0 & 2 & 2 \end{bmatrix}^T$

ii) $N_{split} = \begin{bmatrix} 2 & 2 & 2 & 2 \end{bmatrix}^T$

iii) $N_{split} = \begin{bmatrix} 10 & 0 & 0 & 0 \end{bmatrix}^T$

iv) $N_{split} = \begin{bmatrix} 0 & 10 & 0 & 0 \end{bmatrix}^T$

v) Picking the widest interval of a particular variable from all the boxes and dividing in two each one of the two variables with the biggest widths.

However, the original way of splitting the boxes turned out to be the fastest way of achieving the required accuracy.

4.7 Example 5

The fifth example solved in [Chu, 2007] is essentially another unconstrained one-dimensional problem, now taken from [Deb, 1999]. The problem statement and its analytical solution are given in following table:

Table 4.13: *Formulation of* ***Example 5****.*

Objective function	$f(x) = 2 - \exp\left(-\left(\frac{x-0.1}{0.004}\right)^2\right) - 0.8\exp\left(-\left(\frac{x-0.9}{0.4}\right)^2\right)$
Initial box	$X_{initial} = [0,1]$
Solution point	$x^* = 0.1$
Global minimum	$f^* = f(x^*) \approx 0.9853475$

A plot of the objective function for $x \in X_{initial}$ can be seen in **Figure 4.5**.

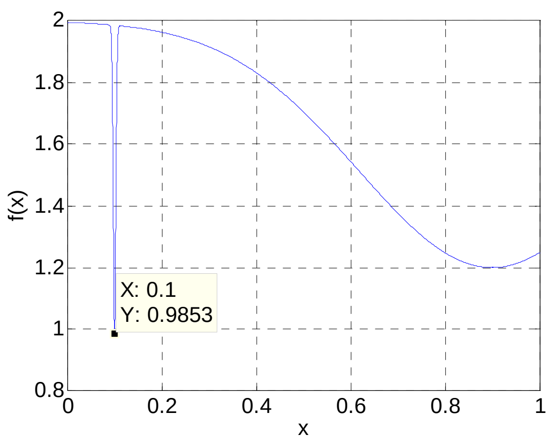

Figure 4.5: *Objective function of* ***Example 5****.*

Again, it is difficult (if not impossible) to rewrite the objective function in order to eliminate dependency.

The same modules used for solving **Example 1** and **Example 2** are now also used to solve **Example 5**.

The algorithm inputs are given in **Table 4.14** while the outputs are given in **Table 4.15**.

Table 4.14: *User inputs in* ***Example 5****.*

Designation	Symbol	Value
Initial upper bound on the minimum	$\overline{f}$	$+\infty$
New boxes per old box	N_{split}	10
Stopping criteria	ε_X	$+\infty$
	ε_f	6.3e-4

Since no information about X^* is given in [Chu, 2007], ε_X was set to $+\infty$. This way, the main loop is stopped when condition (4-29) is fulfilled.

Table 4.15: *Comparative results for* ***Example 5****.*

		The developed algorithm		Chu's algorithm (Non-Gradient Method)	
Designation	**Symbol**	**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	F^*	$[0.98533, 0.98550]$	1.6e-4	$[0.98533, 0.98596]$	6.3e-4
Bounds on the solution point	X^*	$[0.00000, 0.10011]$	1.0e-4	?	?
CPU time	$\Delta t_{CPU}\,[s]$	0.37	N/A	1.66	N/A
Number of remaining boxes	$N_{remaining}$	1	N/A	?	N/A

Note that the so-called non-gradient method is used here, i.e., the only module applied by [Chu, 2007] to solve this problem is the module explained in Section 4.2.2.2.1. The BB technique is used however.

Unfortunately, no information about X^* is given in [Chu, 2007]. Therefore, only F^* can be analysed. By looking at **Table 4.15**, one can see that the developed algorithm has a smaller Δt_{CPU} and a sharper F^*.

Even if $f^* \in F^*$ in [Chu, 2007], this does not prove that $x^* \in X^*$. In fact, it is said in [Chu, 2007] that: *all search results are kept and the smallest one will be used as the global minimum*. This makes it doubtful that $x^* \in X^*$. On the other hand, the developed algorithm yields $x^* \in X^*$ like one would expect.

4.8 Example 6

The sixth and final static optimization problem solved in [Chu, 2007] was taken from [Than *et al.*, 2003]. **Table 4.16** summarizes the problem and its solution. The objective function is three-dimensional and it has no constraints. For the time being, x_1 and x_3 will be assumed as thin intervals. Assuming x_1 and x_2 as thin intervals would give the same results.

Table 4.16: *Formulation of **Example 6**.*

Objective function	$f(x)=\frac{1}{x_1}\left\{1+\left(x_2^2+x_3^2\right)^{0.25}\left[\sin^2\left(50\left(x_2^2+x_3^2\right)^{0.1}\right)+1\right]\right\}$
Initial box	$X_{initial}=\begin{bmatrix}[1,1]\\ [-100,100]\\ [0,0]\end{bmatrix}$
Solution point	$x^*=\begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$
Global minimum	$f^*=1$

What is particularly interesting about this example is that the objective function has several local minima, especially near the global minimum. **Figure 4.6** shows this strange behaviour.

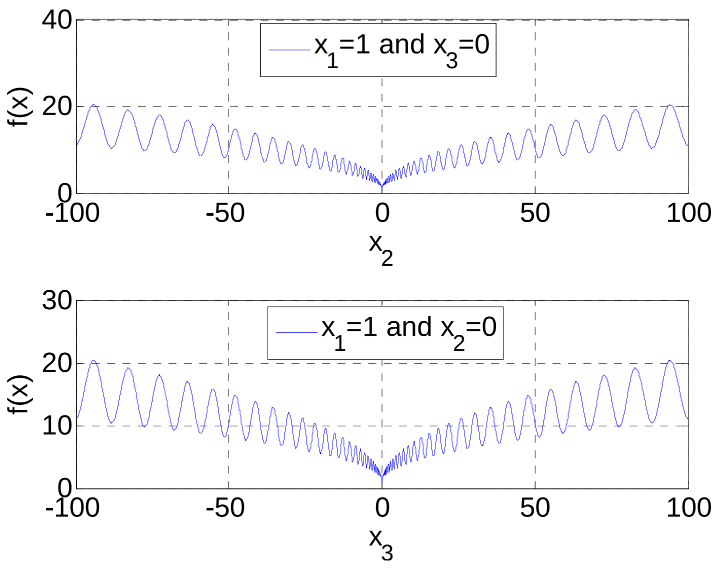

Figure 4.6: *Objective function of Example 6.*

It does not seem possible to further reduce dependency since the objective function is already in a factorized form.

Example 6 was solved using the same modules used in **Example 1**, **Example 2** and **Example 5**. However, there is something unique about this problem. By calculating the gradient of $f(x)$, one can see that the objective function is not differentiable at x^*. Therefore, to guarantee that the global solution point is not eliminated by the optimization algorithm, boxes where the gradient of $f(x)$ is not defined, must not be deleted.

Table 4.17 and **Table 4.18** show the inputs and outputs of the developed algorithm, respectively.

***Table 4.17:** User inputs in **Example 6**.*

Designation	Symbol		Value
Initial upper bound on the minimum	$\overline{f}$		$+\infty$
New boxes per old box	N_{split}		$\begin{bmatrix}0 & 9 & 0\end{bmatrix}^T$
Stopping criteria	ε_X	ε_{X_1}	$+\infty$
		ε_{X_2}	$+\infty$
		ε_{X_3}	$+\infty$
	ε_f		3.4e-1

Once again, no information about X^* is given in [Chu, 2007]. Hence, only (4-29) is used to stop the main loop. In [Chu, 2007], $N_{split} = \begin{bmatrix}0 & 10 & 0\end{bmatrix}^T$.

Chu's algorithm is based on the aforementioned non-gradient method supported by a BB approach. **Table 4.18** shows that it is slower than the developed algorithm.

***Table 4.18:** Comparative results for **Example 6**.*

			The developed algorithm		Chu's algorithm (Non-Gradient Method)	
Designation	Symbol		Result	Width	Result	Width
Bounds on the minimum	F^*		$[1.000,1.020]$	2.0e-2	$[1.000,1.336]$	3.4e-1
Bounds on the solution point	X^*	X_1^*	$[1,1]$	0	$[1,1]$	0
		X_2^*	$[-12,12]$	22	?	?
		X_3^*	$[0,0]$	0	$[0,0]$	0
CPU time	$\Delta t_{CPU}\,[s]$		0.20	N/A	0.27	N/A
Number of remaining boxes	$N_{remaining}$		1	N/A	?	N/A

The algorithm developed during this thesis yields curious results: while the width of F^* is 2.0e-2, the width of X_2^* is 22. To explain this difference, note that the objective function is being evaluated at the midpoint of each box, as explained in Section 4.2.2.2.1. Since the midpoint of $X_{initial_2}$ is $x_2^* = 0$ and each interval X_2 is being divided in an odd number of subintervals, the objective function is being evaluated at x^* every cycle. Therefore, in the beginning of the algorithm, the upper bound on minimum, $\overline{f}$, will be considerably smaller than $\overline{F}$. At the end of the first cycle, condition (4-29) is already satisfied (due to $\overline{f}$) but the algorithm did not have time to sharpen the bounds on x_2^*. This is why the results shown in **Table 4.18** can seem strange.

This problem shows how important it is to properly choose N_{split}. If the intervals X_2 were divided in an even number of subintervals, the objective function would never be evaluated at x^*. To achieve bounds on f^* as sharp as before, the algorithm would need more time. For example, using the same inputs shown in **Table 4.17** except for $N_{split} = \begin{bmatrix}0 & 10 & 0\end{bmatrix}^T$, the time required for the algorithm to stop would be 0.80s. Using $N_{split} = \begin{bmatrix} 0 & 8 & 0 \end{bmatrix}^T$ would even take longer: 1.28s.

Just to demonstrate all the potential of interval analysis, the complete three-dimensional problem will now be solved. Since this was not done in [Chu, 2007], the results cannot be compared. The point here is just to show that interval analysis can successfully solve multidimensional static optimization problems with objective functions as complicated as the one represented in **Figure 4.6**.

The inputs shown in **Table 4.19** were given to the algorithm.

Table 4.19: *User inputs for the complete multidimensional problem.*

Designation	Symbol		Value
Initial box	$X_{initial}$		$\begin{bmatrix} [0.1,1] \\ [-100,100] \\ [-100,100] \end{bmatrix}$
Initial upper bound on the minimum	$\overline{f}$		$+\infty$
New boxes per old box	N_{split}		$\begin{bmatrix} 3 & 3 & 3 \end{bmatrix}^T$
Stopping criteria	ε_X	ε_{X_1}	2.0e-2
		ε_{X_2}	2.0e-2
		ε_{X_3}	2.0e-2
	ε_f		3.4e-1

The corresponding outputs are shown in **Table 4.20**.

Table 4.20: *Results for the complete multidimensional problem.*

Designation	Symbol		Result	Width
Bounds on the minimum	F^*		$[1.000,1.017]$	1.7e-2
Bounds on the solution point	X^*	X_1^*	$[0.9833,1.0000]$	1.7e-2
		X_2^*	$[-0.0052, 0.0051]$	1.0e-2
		X_3^*	$[-0.0052, 0.0051]$	1.0e-2
CPU time	$\Delta t_{CPU}\,[s]$		55	N/A
Number of remaining boxes	$N_{remaining}$		365	N/A

Note that it took the algorithm 55s to satisfy the stopping conditions. Moreover, the number of remaining boxes is the highest so far: 365. Nevertheless, both x^* and f^* were correctly bounded.

Chapter 5 | Dynamic Global Optimization

Dynamic optimization is a far more complex problem than static optimization. While static optimization problems are finite-dimensional, dynamic optimization problems are infinite-dimensional. In static optimization problems, the objective is to optimize a finite set of parameters. In dynamic optimization problems, the objective is to optimize a time dependent function $u(t)$, from an *initial time* t_0 to a *final time* t_f. Since $u(t)$ must be optimized for every $t \in \left[t_0, t_f\right]$, this constitutes an infinite-dimensional problem.

Weiwei Chu proved to a certain extent in [Chu, 2007] that interval analysis can be used to solve dynamic global optimization problems. She did it by solving three distinct problems with known solutions from literature. However, her bounds on the global solutions were sometimes far from sharp. In other cases, the global solution was not correctly bounded.

The objective will now be to improve Weiwei's algorithm. The first two problems solved in [Chu, 2007] will be used as benchmarks. The third one is left as future work. It will be shown that the developed dynamic global optimization algorithm is able to calculate sharper solution intervals than [Chu, 2007] in just a fraction of the time, while correctly bounding the global minima.

As far as the author knows, there is no written book about dynamic global optimization using interval analysis. Therefore, the optimization algorithm was built using as only reference the work previously done by Weiwei in [Chu, 2007]. For the analytical way of solving these problems, the reader is referred to [Visser, 2007] and to [Bryson Jr. and Ho, 1975].

5.1 The problem

In simple terms, the purpose of dynamic optimization is to find the optimal *control function* (or functions) $u^*(t)$ where the *performance index* (sometimes also called cost function) $J\left(x(t),u(t),t\right):\mathbb{R}^\infty \to \mathbb{R}$ has its global minimum J^*. Note that the control functions $u(t)$ exist in an infinite-dimensional space, which makes this an infinite-dimensional problem.

The most general way of writing the performance index J is as follows:

$$J\left(x(t),u(t),t\right) = \phi\left(x\left(t_f\right),t_f\right) + \int_{t_0}^{t_f} L\left(x(t),u(t),t\right)dt \tag{5-1}$$

The function vector $x(t):\mathbb{R} \to \mathbb{R}^n$ represents the *states* of the dynamic system being optimized. The states can be calculated from the control function vector $u(t):\mathbb{R} \to \mathbb{R}^m$ by means of the nonlinear ordinary differential equations modelling the system:

$$\dot{x}(t) = f\left(x(t),u(t),t\right) \tag{5-2}$$

Please note that there is no relation between the state equations $f\left(x,u,t\right)$ and the objective functions $f\left(x\right)$ minimized in Chapter 4.

The term $\phi\left(x\left(t_f\right),t_f\right)$ in (5-1) is called the *end-cost term*. It depends exclusively on the final state and on the final time. The other term $\int_{t_0}^{t_f} L\left(x(t),u(t),t\right)dt$ is called the *path-cost term*. It adds the integrated effect of $L\left(x(t),u(t),t\right)$ to the performance index. The scalar function L is commonly referred to as the *Lagrangian*. No relation exists between this function and the Lagrangian function discussed in Section 4.2.2.4.

Without any loss of generality, all dynamic optimization problems can be treated as minimization problems since maximizing $J\left(x(t),u(t),t\right)$ is equivalent to minimizing $-J\left(x(t),u(t),t\right)$.

5.1.1 Necessary conditions for a solution

A step by step description on how to analytically solve dynamic optimization problems can be found in [Visser, 2007] and [Bryson Jr. and Ho, 1975]. Hereafter, the necessary conditions for the existence of a stationary point of J when there are no terminal constraints and the final time is fixed will be summarized:

$$\dot{x}(t) = f\left(x(t),u(t),t\right) \tag{5-3}$$

$$\dot{\lambda}^T = -\frac{\partial H}{\partial x} \tag{5-4}$$

$$\frac{\partial H}{\partial u} = 0 \tag{5-5}$$

where the scalar function H is called the *Hamiltonian* and is given by:

$$H\left(x(t),u(t),t,\lambda(t)\right) = L\left(x(t),u(t),t\right) + \lambda^T(t)\, f\left(x(t),u(t),t\right) \tag{5-6}$$

The functions $\lambda(t)$ are the so-called *influence functions*.

To solve the $2n$ differential equations established by (5-3) and (5-4), $2n$ boundary conditions are required. Half of them are given by:

$$\lambda^T\left(t_f\right) = \left.\left(\frac{\partial \phi}{\partial x}\right)\right|_{t=t_f} \tag{5-7}$$

and the other half by the initial state which is assumed to be known:

$$x_i\left(t_0\right) \quad ,i=1,...,n \tag{5-8}$$

Since n boundary conditions are given at $t=t_0$ and the other n at $t=t_f$, this is called a *two-point boundary-value problem* (TPBVP).

Equations (5-4), (5-5) and (5-7) are known as the *Euler-Lagrange equations* in the calculus of variations.

If L and f are not explicit functions of time, then it can be proved that H will remain constant along an optimal trajectory, i.e.:

$$\dot{H}^* = \dot{H}\left(u^*(t)\right) = 0 \Leftrightarrow H^* = constant \tag{5-9}$$

Additionally, if the k-th initial state is not known, then the k-th boundary constraint in (5-8) is replaced by:

$$\lambda_k(t_0) = 0 \tag{5-10}$$

5.1.2 Necessary conditions for a solution with functions of the final state specified at a fixed final time

If there are q functions of the final state that need to be fulfilled:

$$\psi\left(x\left(t_f\right), t_f\right) = 0 \tag{5-11}$$

with:

$$\begin{cases} q \le n & , L \ne 0 \\ q \le n-1 & , L = 0 \end{cases} \tag{5-12}$$

then (5-7) is substituted by:

$$\lambda^T\left(t_f\right) = \left(\frac{\partial \Phi}{\partial x}\right)\Bigg|_{t=t_f} = \left(\frac{\partial \phi}{\partial x} + \nu^T \frac{\partial \psi}{\partial x}\right)\Bigg|_{t=t_f} \tag{5-13}$$

where Φ is defined as:

$$\Phi = \phi + \nu^T \psi \tag{5-14}$$

The q multiplier parameters ν are used to adjoin the q functions of the final state to the optimization problem. If $q = n$ for $L = 0$, than the final state could be calculated from (5-11) and used to determine the optimal value of J from (5-1), making it an odd optimization problem.

5.1.3 Necessary conditions for a solution with functions of the final state specified at an unspecified final time

If the final time t_f is not specified, then t_f becomes a control parameter to be chosen together with the control functions $u(t)$. The previous necessary conditions still apply but an extra equation is added to the problem:

$$\left.\left(\frac{\partial\phi}{\partial t}+\nu^T\frac{\partial\psi}{\partial t}+\left(\frac{\partial\phi}{\partial x}+\nu^T\frac{\partial\psi}{\partial x}\right)f+L\right)\right|_{t=t_f}=0 \qquad (5\text{-}15)$$

A performance index of special interest is the time needed to change the state of the system from a known initial state to a specified final state, constituting a *minimum-time problem*. In this case, one has:

$$\begin{cases}\phi=0\\ L=1\end{cases}\Rightarrow J=t_f-t_0 \qquad (5\text{-}16)$$

and expression (5-15) can be simplified into:

$$\left.\left(\nu^T\frac{\partial\psi}{\partial t}+\nu^T\frac{\partial\psi}{\partial x}f\right)\right|_{t=t_f}+1=0 \qquad (5\text{-}17)$$

Of course that, for the problem to make sense, at least one state has to be specified both at $t=t_0$ and at $t=t_f$.

As already seen, if L and f are not explicit functions of time, then H will remain constant along an optimal trajectory. If the final time is not specified as well, then that constant will be zero, i.e.:

$$H^*=H\left(u^*(t)\right)=0 \qquad (5\text{-}18)$$

along an optimal trajectory.

5.1.4 Necessary conditions for a solution with inequality constraints on the control variables

Inequality constraints on the control variables $u(t)$:

$$C\left(u(t),t\right)\le 0 \quad ,t_0\le t\le t_f \quad \left(c\ constraints\right) \qquad (5\text{-}19)$$

make the optimization problems even more complex. Equation (5-5) can no longer be applied to determine the optimal control $u^*(t)$. Instead, it is replaced by the *Minimum Principle of Pontryagin*:

H *must be minimized over the set of all <u>possible</u>* $u(t)$ [Bryson Jr. and Ho, 1975].

By denoting this set of *admissible controls* as U, the Minimum Principle of Pontryagin can be mathematically expressed as:

$$u^*(t) = \arg\min_{u(t)\in U}\{H\} \tag{5-20}$$

An alternative way of dealing with inequality constraints on the control variables is to redefine the Hamiltonian as:

$$H = L + \lambda^T f + \mu^T C \tag{5-21}$$

with the additional requirement that:

$$\begin{cases} \mu_i = 0 & ,C_i < 0 \\ \mu_i \geq 0 & ,C_i = 0 \end{cases} \quad ,i = 1,\dots,c \tag{5-22}$$

By doing this, equation (5-5) remains applicable.

5.1.5 Necessary conditions for a solution with inequality constraints on functions of the state variables

There can also be inequality constraints on functions of the state variables:

$$S(x,t) \leq 0 \quad ,t_0 \leq t \leq t_f \quad (s\ constraints) \tag{5-23}$$

To keep the notation simple, only one constraint ($s=1$) and one control function ($m=1$) will be assumed. The generalization for $s \geq 2$ and $m \geq 2$ is straightforward.

Start by doing successive total time derivatives of S, while substituting $\dot{x}$ by $f(x,u,t)$, until the control function u appears. Assume that the first total time derivative containing u is the q-th one and denote it by $S^{(q)}(x,u,t)$. The Hamiltonian is then rewritten as follows:

$$H = L + \lambda^T f + \mu S^{(q)} \tag{5-24}$$

where:

$$\begin{cases} \mu \geq 0 & ,S = 0 \Leftrightarrow S^{(q)} = 0 \\ \mu = 0 & ,S < 0 \end{cases} \tag{5-25}$$

Equations (5-4) and (5-5) are still necessary conditions for a solution, but the Hamiltonian is now given by (5-24).

Besides the aforementioned conditions, dynamic optimization problems subject to inequality constraints on functions of the state variables must also satisfy the following *tangency constraints*:

$$N\left(x\left(t_{in/out}\right),t_{in/out}\right)\triangleq\begin{bmatrix} S\left(x\left(t_{in/out}\right),t_{in/out}\right) \\ S^{(1)}\left(x\left(t_{in/out}\right),t_{in/out}\right) \\ \vdots \\ S^{(q-1)}\left(x\left(t_{in/out}\right),t_{in/out}\right) \end{bmatrix}=0 \qquad (5\text{-}26)$$

where $t_{in/out}$ are the time instants at which the system's path enters or leaves the constraint boundary.

According to [Bryson Jr. and Ho, 1975] and [Speyer and Bryson Jr., 1968], by defining the Hamiltonian as (5-24), the entry points or the exit points may be arbitrarily picked as the place to satisfy these interior boundary conditions, as they will be automatically satisfied at the other end. Note that HC can be applied to both end points.

As a side note, the influence functions $\lambda(t)$ and the Hamiltonian $H(t)$ are in general discontinuous at junction points between constrained and unconstrained arcs [Bryson Jr. and Ho, 1975].

5.1.6 Direct versus indirect numerical dynamic optimization methods

Numerical dynamic optimization methods which calculate the optimal control function $u(t)$ by solving a multi-point boundary-value problem, i.e., by solving the aforementioned necessary conditions, are called indirect methods [Erb, 2008].

The advantage of indirect methods is that the control functions can have any shape and the solutions are analytically guaranteed to be local minima of J. On the other hand, multi-point boundary-value problems are often extremely difficult (if not impossible) to solve [Wikipedia, 2008a]. In addition to their mathematically complexity, indirect methods suffer from several other difficulties: the initial guess is often hard to obtain as the convergence radius is often small; the initial guess often does not have an intuitive physical meaning [Chu, 2007]; new constraints and most model changes require the structure of the equations to be changed; complex constraints are hard or even impossible to implement. Nevertheless, the few problems that can actually be solved by indirect methods are done so very accurately and fast.

Nowadays, direct methods have substituted indirect methods as the most popular technique. Direct methods are based on the parameterization of the control functions and/or states; the necessary conditions are completely disregarded. Since the shape of $u(t)$ and/or $x(t)$ is restricted, additional constraints are added to the optimization problem, which limits the accuracy of the solutions. On the other hand, it is easier in general to optimize the parameters representing $u(t)$ and/or $x(t)$ with an NLP tool than to solve a multi-point boundary-value problem. As a consequence, the range of problems that can be solved with direct methods is significantly larger than the range of problems that can be solved with indirect methods. Therefore, direct methods are said to be more general and robust than indirect methods.

Due to the aforementioned reasons, the algorithm developed here is based on direct methods, as it will become obvious in Section 5.2. Nevertheless, it would be interesting to investigate in the future if interval analysis can also be used to make indirect methods yield global minima.

5.2 The algorithm

Needless to say that finding the global solution of a dynamic optimization problem via the necessary conditions described in Section 5.1 is, in general, not a very easy task. A commonly used technique to transform this otherwise infinite-dimensional problem into a finite-dimensional one is to parameterize the control function $u(t)$ [Chu, 2007]:

$$u(t) \xrightarrow{Parametrization} u(P_1, ..., P_l, t) \qquad (5\text{-}27)$$

Thus, the problem is no longer to find the optimal control function $u^*(t)$ that minimizes the performance index $J(u(t),t): \mathbb{R}^\infty \to \mathbb{R}$ but the l parameters P_j^* ($j = 1, ..., l$) that minimize $J(P_1, ..., P_l, t): \mathbb{R}^{l+1} \to \mathbb{R}$. The thin box formed by these l parameters will be denoted by P^*. A parameterization with l parameters will be called a l-th order parameterization.

The biggest problem of parameterizing the control function is that the global solution might never be found if the true shape of $u^*(t)$ is not known. Take for example $u^*(t) = \exp(2t)$ and the following two control function parameterizations:

1) $u_1(t) = P_1 + P_2 t$

2) $u_2(t) = \exp(P_1 t)$

If the first parameterization $u_1(t)$ is chosen, then it is obvious that the global solution $u^*(t)$ will never be found. The best a direct method can do is to find the global solution of the initial problem subject to an additional constraint on the shape of $u(t)$. Note, however, that there might not even be a solution to this constrained problem. For example, assuming the initial state is known, a certain prescribed final state might not be reachable using a certain parameterization. Even if it is, the solution might not satisfy the necessary conditions (5-4) and (5-5) for the existence of a stationary point of J. On the other hand, if $u_2(t)$ is picked, then the global solution will be correctly bounded by the interval optimization algorithm.

If the true shape of $u^*(t)$ is not known, then finding the true global minimum of a dynamic optimization problem will be quite hard, if not impossible. All the examples at the end of this chapter have well-known analytical solutions from literature, which makes things easier.

According to [Chu, 2007], an especially flexible way of parameterizing the control function is through the use of neural networks. In this thesis, however, only simple standard functions will be used to approximate $u(t)$, as this is already a sufficiently challenging task.

By parameterizing the control function, the initial infinite-dimensional dynamic optimization problem is transformed into a finite-dimensional *quasi-static* optimization problem: the objective is no longer to find the optimal function $u^*(t)$ but to find the optimal parameters P_j^* ($j = 1, ..., l$). It that sense, dynamic and static optimization are similar. Therefore, the reader will probably find some similarities between what follows and Chapter 4.

To start the interval algorithm, an l-dimensional initial box or boxes $P_{initial}$ containing P^* must be given. If the initial boxes overlap and the minimum point occurs in more than one initial box, then the optimum solution will be independently calculated from each of these initial boxes, wasting valuable CPU time. This is the only inconvenience of prescribing several initial boxes. However, to simplify what follows, it is assumed that only one initial box is provided.

In the limit, if there is not sufficient information available, the initial box $P_{initial}$ can be defined component-wise as $\left[-\overline{N}, \overline{N}\right]$ where $\overline{N}$ is the largest floating point number representable by the computer. However, one should keep in mind that the wider the initial box, the longer it will take to calculate sharp results.

The support column of the developed dynamic global optimization algorithm is again a common BB procedure. Now, however, instead of evaluating the objective function $f\left(x\right)$, as in Chapter 4, the performance index $J\left(x,u,t\right)$ will be evaluated. As a reminder, the two steps of the BB procedure are:

1) Branching (or splitting): each remaining box is divided in two or more subboxes;

2) Bounding and pruning: for each subbox P_i satisfying all the constraints, bounds on $J\left(P_i,t\right)=\left[\underline{J}_i,\overline{J}_i\right]$ are calculated. Let $\overline{J}$ denote the smallest calculated upper bound (this notation will be kept throughout the text). Since $J^* < \overline{J}$, any box wherein $\underline{J}_i > \overline{J}$ can be safely deleted.

If no initial upper bound on J^* is known, $\overline{J}$ is set to $+\infty$ before the algorithm is started.

In addition to the upper bound on the minimum, the developed algorithm has other techniques for deleting boxes. These techniques are explained one by one in Section 5.2.2. The developed algorithm also has procedures capable of shrinking the boxes, i.e., capable of reducing their width. In the limit, these latter procedures can even delete entire boxes. They are treated in Section 5.2.1.

As in Chapter 4, all the boxes are processed at the same time and not one by one as in [Hansen and Walster, 2004]. Although it is the author's conviction that the former approach is faster, this subject still needs to be further investigated.

The overall structure of the dynamic and static optimization algorithms is very similar, as can be seen by comparing **Figure 4.1** with **Figure 5.1**.

Indeed, the concepts behind each block are the same as before. The only block that is completely new is the *shrinking phase* block. This procedure is suggested in [Chu, 2007] as a tool to chop off considerable portions of the initial box. However, the initial search domain will not be reduced into a thin box just by it.

According to [Chu, 2007], the shrinking phase is especially helpful when the number of parameters is high. When it is low, the shrinking phase does not improve the performance of the algorithm and, therefore, should not be used.

Hereafter, the shrinking phase will sometimes be referred to as the *first phase* of the algorithm and the remaining blocks as the *second phase* of the algorithm. It is intuitive to make this distinction since both phases have approximately the same number of code lines.

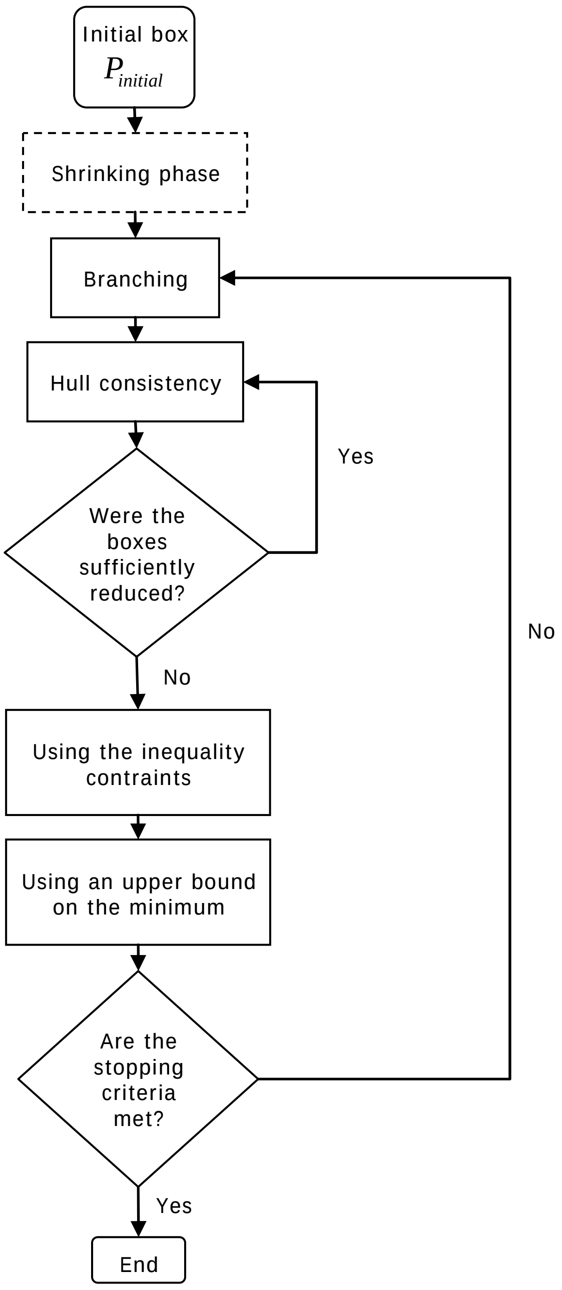
Initial box
$P_{initial}$
Shrinking phase
Branching
Hull consistency
Yes
Were the boxes sufficiently reduced?
No
No
Using the inequality contraints
Using an upper bound on the minimum
Are the stopping criteria met?
Yes
End

Figure 5.1: *Flowchart of the developed dynamic global optimization algorithm.*

In every one of the forthcoming dynamic optimization examples, the only visible effect of adding a shrinking phase to the algorithm was the increase of the execution time. That is why a dashed line was used to represent the shrinking phase block in **Figure 5.1**. This does not mean that the shrinking phase concept does not work. As will be seen in Section 5.2.3, sometimes it does.

The static optimization algorithm discussed in Chapter 4 can also be augmented with a shrinking phase. In Section 5.2.3, **Example 4** (the one with highest number of variables) will be redone using a shrinking phase.

The shrinking phase is analysed in depth in Section 5.2.3.

The dynamic optimization algorithm follows the same modular structure of the static optimization algorithm. This structure makes it easier to modify the code and to adapt it to new problems. The order of the different modules was once again roughly chosen in order to optimize the algorithm’s speed.

It might not be useful or even possible to apply a particular procedure to a particular problem, as it will become apparent when discussing each example individually.

The reader might have noticed that there are fewer procedures for deleting boxes in **Figure 5.1** than in **Figure 4.1**, e.g., **Figure 5.1** does not have any gradient method for deleting boxes. This reflects how immature and undeveloped the field of dynamic optimization using interval analysis is, when compared to static optimization.

5.2.1 Procedures to shrink the boxes

Due to the reasons presented in Section 4.2.1.3, HC is the only procedure used by the dynamic optimization algorithm to shrink the boxes. BC is not used.

5.2.1.1 Hull consistency

Hull consistency was already thoroughly explained in Section 4.2.1.1. All the observations given then for the static case are still applicable now to the dynamic case.

Depending on the problem being solved, the following equality constraints might be present:

1) Initial and final state constraints;

2) Continuity constraints on the states;

3) Continuity constraints on the control functions;

4) Tangency constraints at entry and exit points;

5) Necessary conditions for the existence of a stationary point of J ;

6) Other equality constraints involving the states and/or the control functions;

Whenever possible, these constraints should be converted into equations solvable by HC, to take advantage of its shrinking capabilities.

If there are N equations solvable by HC, then it does not make sense to parameterize the control function $u(t)$ with less than N parameters. Otherwise, one would only need to solve a system of N nonlinear equations to determine the optimal control function $u^*(t)$, that is, assuming that there is a solution. In other words, instead of solving a real optimization problem, one would only need to solve a system of nonlinear equations using interval analysis. Therefore, to take advantage of all available information and to make $u(t)$ as wide-ranging as possible, the following lower bound on the number of parameters P_j ($j=1,...,l$) should be imposed:

$$l > N \tag{5-28}$$

A natural choice when no information is available is to choose the control function as an *interval polynomial*, i.e., as a polynomial with interval coefficients. For example, for $m=1$:

$$u(t) = P_1 + P_2 t + P_3 t^2 + \ldots + P_l t^l \tag{5-29}$$

Therefore, according to (5-28), the bigger the N, the bigger the l, allowing a more general representation of the control function.

5.2.2 Procedures to eliminate boxes

In this section, two techniques for deleting boxes are described. They are based on the inequality constraints and on an upper bound on the minimum. Unlike in Section 4.2.2, no gradient-based method is used to delete boxes.

5.2.2.1 Using the inequality constraints

Assume a general inequality constraint of the form:

$$R\left(x(t),u(t),t\right) \le 0 \quad ,t_0 \le t \le t_f \tag{5-30}$$

By parameterizing $u(t)$ and by solving (5-2), the inequality constraint can be rewritten as:

$$R\left(P_1,\ldots,P_l,t\right) \le 0 \quad ,t_0 \le t \le t_f \tag{5-31}$$

which is equivalent to:

$$R\left(P_1,\ldots,P_l,T\right) \le 0 \tag{5-32}$$

where T is defined as:

$$T \triangleq \left[t_0,t_f\right] \tag{5-33}$$

Assume $R\left(P_1,\ldots,P_l,T\right)$ is evaluated over a box P, resulting in:

$$R(P,T) = \left[\underline{R(P,T)},\overline{R(P,T)}\right] \tag{5-34}$$

It is said that:

1) P is (*certainly) feasible* if $\overline{R(P,T)} \le 0$ and that P is *(certainly) strictly feasible* if $\overline{R(P,T)} < 0$.

2) P is (*certainly) infeasible* if $\underline{R(P,T)} > 0$.

Any box P that is certainly infeasible can be safely deleted.

Once more, if extended interval analysis is supported, instead of evaluating $R(P,T)$ to check if P is feasible or not, a better alternative is to solve $R(P,T)=[-\infty,0]$ through HC.

Dependency can have a major role in the evaluation of $R(P,T)$, especially due to T. If T appears more than once in the expression of $R(P,T)$, even after rearranging the terms, it will be extremely difficult to prove feasibility or infeasibility of a certain box P: while the width of the boxes P will be reduced as the algorithm progresses (due to the branching and shrinking) the interval T will always have the same width:

$$w(T)=t_f-t_0 \tag{5-35}$$

In the limit, it might happen that the boxes P are already point intervals, but the bounds on $R(P,T)$ are still too wide to prove feasibility or infeasibility, all due to the dependency effect and to the constant nonzero width of T.

Being able to prove that a certain box is feasible is a critical task: only feasible boxes can be used to update $\overline{J}$. Being able to prove infeasibility helps, but the BB algorithm can still work without it.

This dependency problem was overcome by dividing T in N_{ineq} subintervals T_i given by:

$$T_i=\left[t_0+(i-1)\frac{t_f-t_o}{N_{ineq}},t_0+i\frac{t_f-t_o}{N_{ineq}}\right] \quad ,i=1,\ldots,N_{ineq} \tag{5-36}$$

and by evaluating $R(P,T_i)$ for $i=1,\ldots,N_{ineq}$ instead of $R(P,T)$. The bigger the N_{ineq}, the smaller the width of the subintervals T_i, and bigger the changes of proving that a certain box is feasible or infeasible, at the expense of a higher computational time.

The definitions of feasibility and infeasibility can be rewritten taking into account this new way of evaluating the inequality constraints:

1) P is *(certainly) feasible* if $\overline{R(P,T_i)}\le 0$ for **all** $i=1,\ldots,N_{ineq}$ and P is *(certainly) strictly feasible* if $\overline{R(P,T_i)}<0$ for **all** $i=1,\ldots,N_{ineq}$.

2) P is *(certainly) infeasible* if $\underline{R(P,T_i)}>0$ for **any** $i=1,\ldots,N_{ineq}$.

5.2.2.2 Using an upper bound on the minimum

As mentioned before, any box P_i for which $\underline{J_i}>\overline{J}$, where $\underline{J_i}$ is calculated from $J(P_i,t)=\left[\underline{J_i},\overline{J_i}\right]$, can be safely deleted. The only condition is that $\overline{J}$ can only be updated using boxes that contain at least one point satisfying all equality and inequality constraints, i.e., feasible boxes.

In Section 4.2.2.2, the four possible combinations of equality and inequality constraints were individually analysed for the static case. Now, the same will be done for the dynamic case. Since there are no major differences between the two, this will be done rather quickly.

Note that all the examples solved at the end of this chapter have at least one equality constraint, namely, the initial and final states.

5.2.2.2.1 The unconstrained case

When a problem has no constraints, the performance index can be evaluated at the midpoint of P_i, $J\left(m\left(P_i\right),t\right)=\left[\underline{J}_{i,mid},\overline{J}_{i,mid}\right]$. The performance index does not need to be evaluated over the entire box P_i, $J\left(P_i,t\right)=\left[\underline{J}_i,\overline{J}_i\right]$. Since:

$$\overline{J}_{i,mid}\le\overline{J}_i \qquad (5\text{-}37)$$

a smaller upper bound on the minimum can in general be calculated by using the midpoints.

5.2.2.2.2 The inequality constrained case

When there are only inequality constraints, before evaluating $J\left(m\left(P_i\right),t\right)=\left[\underline{J}_{i,mid},\overline{J}_{i,mid}\right]$, one has to guarantee that $m\left(P_i\right)$ is a certainly feasible (thin) box, i.e., one has to guarantee that:

$$\overline{R\left(m\left(P_i\right),T\right)}\le 0 \qquad (5\text{-}38)$$

If (5-38) holds, than $\overline{J}$ can be updated using the box P_i.

5.2.2.2.3 The equality constrained case

Dealing with equality constraints is relatively harder than dealing with inequality constraints.

The performance index can no longer be evaluated solely at the midpoint of P_i. It has to be evaluated over the entire box P_i.

The existence of a solution inside a box can be proven using **Theorem 4.1**, as properly explained in Section 4.2.2.2.3.

5.2.2.2.4 The inequality and equality constrained case

When a problem has simultaneously inequality and equality constraints, the following two conditions must be proven:

1) P_i must be a feasible box with respect to the equality constraints. This can be proven using the method described in Section 4.2.2.2.3.

2) The inequality constraints must be satisfied over the entire box P_i, i.e., P_i must be a certainly feasible box.

If both conditions are met, $\overline{J}_i$, resulting from $J\left(P_i,t\right)=\left[\underline{J}_i,\overline{J}_i\right]$, is an upper bound on the global minimum.

5.2.3 The shrinking phase

The flowchart of the shrinking phase can be seen in **Figure 5.2**. Note that the procedures used to shrink the search region are the same used in the second part of the algorithm: HC, the inequality constraints and the upper bound on the minimum. The only aspect that is truly different is the way the boxes are split. In brief, in the shrinking phase, the parameters are split sequentially, one at a time.

The shrinking phase has an inner loop and an outer loop. One cycle of the inner loop consists on first splitting a particular parameter in $N_{shrinking}$ subintervals, while keeping the other parameters fixed. Then, the search region is shrunk using the already mentioned procedures. Finally, the hull of the remaining boxes is calculated and used as the initial box in the next cycle. The number of boxes at any particular time is, therefore, always smaller or equal to $N_{shrinking}$. Thus, the maximum time an inner cycle can take is always the same during the entire shrinking phase. That is not the case in the second part of the algorithm where, in general, the number of boxes and the run time increases every cycle.

Before beginning the inner loop, the width of the initial box, W_i, is calculated. Denote the width of the final box at the end of the inner loop as W_f. W_i and W_f are in general vectors. If:

$$\max\left(W_i - W_f\right) > \varepsilon_{shrinking} \qquad \textit{(5-39)}$$

where $\varepsilon_{shrinking} > 0$ is a parameter given by the user, the inner loop is repeated. That is, if the initial box was sufficiently reduced in the last inner loop, the inner loop is repeated. In [Chu, 2007], the number of times the inner loop is repeated is fixed by the user before the algorithm is started.

As the reader might have noticed, (5-39) is similar to the HC loop stopping condition explained in Section 4.2.5.

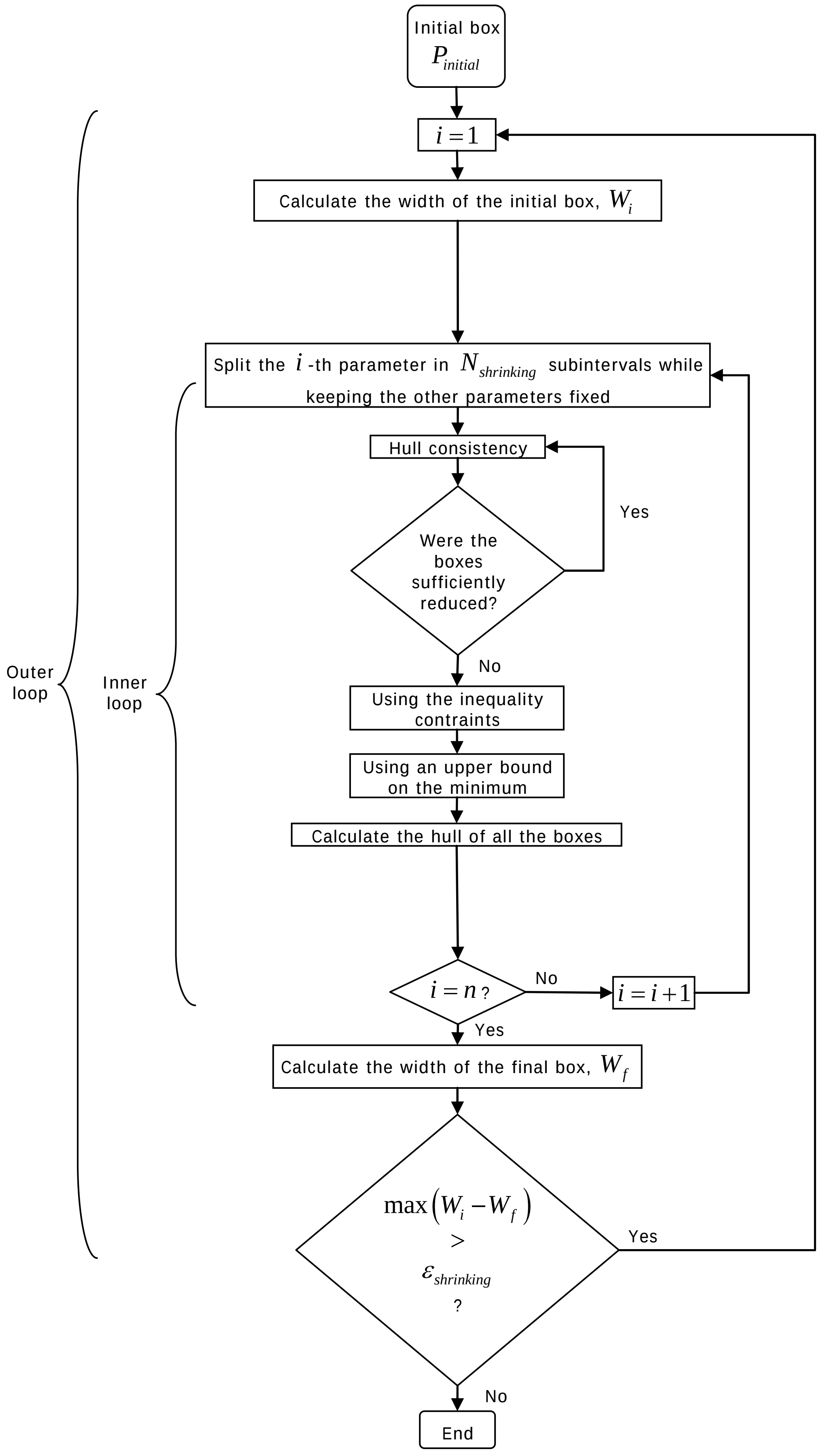
Initial box $P_{initial}$
$i=1$
Calculate the width of the initial box, W_i
Split the i-th parameter in $N_{shrinking}$ subintervals while keeping the other parameters fixed
Hull consistency
Were the boxes sufficiently reduced?
Yes
No
Using the inequality contraints
Using an upper bound on the minimum
Calculate the hull of all the boxes
$i=n$?
No
$i=i+1$
Yes
Calculate the width of the final box, W_f
$\max(W_i - W_f) > \varepsilon_{shrinking}$?
Yes
No
End
Outer loop
Inner loop

Figure 5.2: *Flowchart of the shrinking phase.*

As already mentioned, the main objective of the shrinking phase is to quickly chop off significant portions of the initial box $P_{initial}$. According to [Chu, 2007], the bigger the number of parameters, the more useful it is. The argument is the following: since dividing more than three parameters in the second phase of the algorithm is extremely hard, as pointed out in Section 4.2.3, the fact that in the shrinking phase all the parameters are divided, independently of how many they are, can be extremely helpful in problems where l is big. Also according to [Chu, 2007], when l is small (smaller or equal to two) only using the second phase of the algorithm can be faster than using both phases. The results obtained during this thesis show that this is true even for bigger values of l.

In any case, the user should not expect the initial search domain to be reduced into a thin box during the shrinking phase: its only goal is to reduce $P_{initial}$ so that the second phase of the algorithm can (supposedly) run faster.

Changing the order of the parameters may lead to different results at the end of the shrinking phase. How to decide which order is the best is not an easy task. However, according to [Chu, 2007], parameters representing time should be processed first.

As promised, **Example 4** of Chapter 4 will now be used to demonstrate the capabilities of the shrinking phase. In **Table 5.2**, the results with and without the shrinking phase are compared. The results without the shrinking phase are the same shown in **Table 4.12**. The user inputs in **Table 4.11** are used in both configurations. The user inputs related to the shrinking phase are presented in **Table 5.1**.

Table 5.1: *User inputs related to the shrinking phase.*

Designation	Symbol	Value
Number of subintervals in which each parameter is divided during the shrinking phase	$N_{shrinking}$	100
Shrinking phase stopping criterion	$\varepsilon_{shrinking}$	0.1

Table 5.2: *Results with and without the shrinking phase.*

			Without the shrinking phase		With the shrinking phase	
Designation	Symbol		Result	Width	Result	Width
Bounds on the minimum	F^*		$[0.6999, 0.7001]$	6.9e-5	$[0.6999, 0.7001]$	7.7e-5
Bounds on the solution point	X^*	X_1^*	$[-0.7001, -0.6999]$	8.9e-5	$[-0.7001, -0.6999]$	9.2e-5
		X_2^*	$[0.48999, 0.49013]$	1.2e-4	$[0.48999, 0.49015]$	1.4e-4
		X_3^*	$[0.91268, 0.91283]$	1.4e-4	$[0.91268, 0.91284]$	1.5e-4
		X_4^*	$[0.0000, 0.0058]$	5.8e-3	$[0.0000, 0.0051]$	5.0e-3
CPU time	$\Delta t_{CPU}\,[s]$		48	N/A	26	N/A
Number of remaining boxes	$N_{remaining}$		6	N/A	17	N/A

The desired accuracy is achieved in approximately half the time by adding the shrinking phase.

Also interesting is analysing what happens when the parameter $N_{shrinking}$ is varied. **Table 5.3** shows the elapsed CPU time and the $\mathrm{sum}(\Delta W)$ for different values of $N_{shrinking}$. The

$\mathrm{sum}(\Delta W)$ is a measure of how much the initial search domain was reduced due to the shrinking phase. ΔW is calculated as follows:

$$\Delta W = w\left(P_{final}\right) - w\left(P_{initial}\right) \tag{5-40}$$

where P_{final} is the final box at the end of the shrinking phase. $N_{shrinking}$ was the only varied input.

The minimal elapsed time is achieved when $N_{shrinking}$ is equal to 100. As a general rule, the bigger the value of $N_{shrinking}$, the more the shrinking phase reduces the initial search domain. For small values of $N_{shrinking}$ ($N_{shrinking} = 1$ and $N_{shrinking} = 10$), this reduction is not big enough to decrease significantly the total elapsed CPU time. For big values of $N_{shrinking}$ ($N_{shrinking} = 1000$), the shrinking phase gives slightly sharper results than for intermediate values of $N_{shrinking}$ ($N_{shrinking} = 100$), but this does not compensate the extra time spent on the shrinking phase. As a result, a compromise must be made: an intermediate value of $N_{shrinking}$ should be selected. In the forthcoming examples, $N_{shrinking}$ is always set to 100.

Table 5.3: *Effect of* $N_{shrinking}$.

$N_{shrinking}$	$\Delta t_{CPU}\,[s]$	$\mathrm{sum}(\Delta W)$
1	41	4.174
10	41	4.663
100	26	4.817
1000	61	4.867

In the next section, a new branching routine for the second part of the algorithm is introduced. With it, **Example 4** can be solved faster without a shrinking phase. From the author's experience, this is generally the case: it is always possible to improve the second part of the algorithm to a point where using a shrinking phase is no longer beneficial. Therefore, it is the author's opinion that more effort should be put on optimizing the second part of the algorithm than on the shrinking phase.

5.2.4 Branching

The branching routine executed in the second part of the dynamic optimization algorithm follows the same directives explained in Section 4.2.3 for the static case. Only the notation is different: instead of splitting boxes X with n components, it now splits boxes P with l components.

In fact, the first working branching routine implemented in the dynamic optimization algorithm was copy/pasted from the static case. That would have worked nicely if it was not for the fact that dynamic optimization problems are inherently harder to solve: in general, a lot more boxes need to be processed simultaneously. In this context, the main bottleneck slowing down the optimization algorithm was the branching routine.

The old branching routine was based on l *FOR* cycles, each one covering all the boxes. That made the branching routine extremely slow, for two reasons:

1) INTLAB does not work very well with loops, as advocated in [Rump, 2007]. Therefore, whenever possible, loops should be replaced by matrix operations.

2) As the number of boxes grew, covering all of them became harder and harder.

Therefore, a new branching routine was created based on matrix operations. In it, there are no longer *FOR* cycles covering all the boxes.

Table 5.4 shows the time saved with the new branching routine in **Example 4**. The first two entries of **Table 5.4** were taken from **Table 4.12**.

The only difference between the first and the last case is the branching routine. With the new branching routine, the static optimization algorithm is approximately 16 times faster. When compared to Chu's algorithm, it is approximately 543 times faster.

Table 5.4: *Different ways of solving **Example 4**.*

	CPU time $\Delta t_{CPU}\left[s\right]$
With the old branching routine	48
Chu's algorithm	1629 (≈27m)
With the new branching routine	3

As already mentioned, the new branching routine makes the second part of the algorithm so fast that applying the shrinking phase to **Example 4** is no longer advantageous: with the shrinking phase, the CPU time increases from 3 to approximately 5 seconds ($N_{shrinking}=100$ and $\varepsilon_{shrinking}=0.1$).

As a general rule, *FOR* cycles covering all the boxes should always be replaced by matrix operations.

The new branching routine also checks if the diameter of a component is equal to zero. If it is, that component is not divided.

As explained in Section 4.2.2.2.3, in order to apply **Theorem 4.1**, some parameters P_j ($j=1,...,l$) must be fixed. Denote these parameters by P_{fixed} and the parameters kept free by P_{free}. As also explained in Section 4.2.2.2.3, the objective is to prove that $P_{free}^{'}\subset P_{free}$, where $P_{free}^{'}$ is calculated from P_{free} through HC. Analogously to the static case, the chances that $P_{free}^{'}\subset P_{free}$ are higher if $P_{free}^{'}$ is much smaller than P_{free}. Therefore, when deciding which parameters to split during the branching phase, the user should take into account that parameters forming P_{free} should not be divided, typically. Otherwise, as the algorithm progresses, the boxes P_{free} will become so sharp that proving $P_{free}^{'}\subset P_{free}$ will be almost impossible.

For the same reason, the parameters with the sharpest initial bounds (relatively speaking), or in other words, the sharpest components of $P_{initial}$, should not belong to P_{free}, typically. Otherwise, it will be extremely hard to prove $P_{free}^{'}\subset P_{free}$.

Moreover, when rearranging interval expressions to reduce the dependency effect, the parameters not divided during the branching phase, which in general will be wider, should be factored out first.

Assume that there is only one solution P^* inside $P_{initial}$. If there are as many equations as parameters ($l = N$), then there is no need to branch the boxes, i.e., N_{split} can be defined as an empty array, $N_{split} = [-]$, since in most cases, sharp bounds on P^* can be determined just by applying the HC loop illustrated in **Figure 4.1** and in **Figure 5.1**. If the number of parameters is higher than the number of equations, then the author suggests, as a rule of thumb, splitting:

$$l - N \tag{5-41}$$

parameters. According to the author's experience, this is the ideal number to make the algorithm as fast as possible. In Chapter 4, a trial-and-error procedure was used to optimize N_{split}. The results obtained then are in agreement with (5-41).

The decision of how many parameters to split can be based on (5-41). However, the user still has to decide which parameters to split.

In Section 4.2.3, it was said that eight new subboxes per old box is the maximum a normal personal computer can handle. This rule of thumb takes supremacy over expression (5-41).

If there is more than one solution P^* inside $P_{initial}$, then to bound all of them, the author suggests replacing (5-41) by:

$$l - N + 1 \tag{5-42}$$

5.2.5 Interpretation of the results

As in Section 4.2.4, assume multiple contiguous boxes remain at the end of the algorithm. If that is not the case, the following results can be easily adapted.

Analogously to Section 4.2.4, interval analysis guarantees that:

$$\min_i \left(\inf \left(P_i^j \right) \right) \le P_j^* \le \max_i \left(\sup \left(P_i^j \right) \right) \quad , j = 1, \ldots, l \tag{5-43}$$

i.e., the j-th component of the solution point(s) P^* must be contained inside that component's hull. P_i^j represents the j-th component of the i-th remaining box.

As a consequence of (5-43), just like in Section 4.2.4, the guaranteed bounds on J^* are:

$$\underline{\mathrm{J}} \le J^* \le \min\left(\overline{J}, \overline{\mathrm{J}} \right) \tag{5-44}$$

where $\underline{\mathrm{J}}$ and $\overline{\mathrm{J}}$ are given by:

$$\underline{\mathrm{J}} = \min_i \left(\underline{J\left(P_i, t\right)} \right) \tag{5-45}$$

$$\overline{\mathrm{J}} = \max_i \left(\overline{J\left(P_i,t\right)} \right) \tag{5-46}$$

If $\overline{J} < \overline{\mathrm{J}}$, a new HC equation is suggested here based on (5-44) to shrink the search region. Take $J\left(P_i,t\right) = \left[\underline{J_{P_i}}, \overline{J_{P_i}} \right]$ as the bounds on the performance index associated with a certain box P_i. Then, it should be possible to use the following relation:

$$J\left(P_i,t\right) = \left[\underline{J_{P_i}}, \overline{J} \right] \tag{5-47}$$

as a HC equation to shrink the box P_i since, by definition, $\overline{J} \le \overline{J_{P_i}}$. The same concept can be applied to the static global optimization algorithm discussed in Chapter 4. This idea still needs to be properly validated and tested and is only left here as a recommendation for future work. As far as the author knows, no other author has suggested this procedure before. In particular, it is not used or even mentioned in [Hansen and Walster, 2004] or in [Chu, 2007].

As a final remark, assume p is a point inside one of the remaining boxes. Interval analysis guarantees that:

$$\underline{\mathrm{J}} \le J\left(p,t\right) \le \overline{\mathrm{J}} \tag{5-48}$$

but not that:

$$\underline{\mathrm{J}} \le J\left(p,t\right) \le \overline{J} \tag{5-49}$$

As a consequence, if $\overline{J} < \overline{\mathrm{J}}$, which is normally the case, if a point needs to be selected from the final interval solution, the best course of action, according to the author, is to select a point, for example the midpoint, of the box where the last upper bound on the minimum $\overline{J}$ was calculated. By selecting such a point, one can guarantee that (5-49) is satisfied. Note, however, that p is in all probability not a feasible point with respect to the equality constraints (but certainly feasible with respect to the inequality constraints). But since a feasible solution exists inside the box containing p, p should be *almost feasible*. According to [Hansen and Walster, 2004], *no algorithm can assure that a single point is certainly feasible because of rounding errors in evaluating the equality constraints*. Therefore, an *almost feasible* point is the best one can hope for.

5.2.6 Stopping criteria

As in Section 4.2.5, the dynamic optimization algorithm has two stopping criteria:

1) $$\max_i\left(\sup\left(P_i^j\right)\right) - \min_i\left(\inf\left(P_i^j\right)\right) \le \varepsilon_{P_j} \quad , j = 1,\dots,l \tag{5-50}$$

where P_i^j represents the j-th component of the i-th box, and:

2) $$\min\left(\overline{J}, \overline{\mathrm{J}}\right) - \underline{\mathrm{J}} \le \varepsilon_J \tag{5-51}$$

where $\underline{\mathrm{J}}$ and $\overline{\mathrm{J}}$ are given by (5-45) and (5-46), respectively. Both ε_{P_j} (l parameters) and ε_J (1 parameter) are given by the user.

As can be seen by comparing (5-50)-(5-51) with (5-43)-(5-44), if both stopping conditions are satisfied, the guaranteed bounds on P^* and J^* will be sharper than ε_P and ε_J, respectively. As a rule, however, if one of the stopping conditions is met, the main loop should be stopped. The reasons for this are explained in Section 4.2.5.

Some precautions should be taken when choosing ε_J. If ε_J is chosen so small that the wrapping effect and dependency prevent (5-51) from ever being fulfilled, one might end up with an infinite cycle if $w\left(P^*\right) > \varepsilon_P$.

However, in cases where it is known from the beginning that the solution point is indeed a point and not an interval, i.e., that $w\left(P^*\right) = 0$, it is preferable only to stop the cycle when both conditions are met simultaneously. In that case, after termination, the guaranteed bounds on P^* and J^* will be indubitably sharper than ε_P and ε_J.

Another advantage of requiring both stopping conditions to be met at the same time is that, by selecting ε_P or ε_J relatively large, the criterion that will effectively stop the algorithm can be chosen.

All the examples discussed in this chapter have indeed a single solution point. Therefore, the main loop is only stopped when (5-50) and (5-51) are satisfied simultaneously.

The conditions described in Section 4.2.5 for stopping the HC loop are kept.

The stopping condition used in [Chu, 2007] is based on the width of the bounds on the final state. No mathematical expression is given however. For more information, the reader is referred to [Chu, 2007]. In Section 5.5.3, a possible additional stopping criterion based on the final state is also suggested.

Four dynamic optimization problems will now be solved using the aforementioned algorithm.

5.3 Example 1

The first dynamic optimization problem solved in [Chu, 2007] is an unconstrained one-dimensional problem taken from [Visser, 2007]. It is described by:

Performance index:
$$J\left(x(t),u(t),t\right)=\int_0^1\left(0.5u^2(t)+x(t)\right)dt \qquad (5\text{-}52)$$

State dynamics:
$$\dot{x}(t)=u(t) \qquad (5\text{-}53)$$

Initial state:
$$x(0)=0 \qquad (5\text{-}54)$$

Final state:
$$x(1)=1 \qquad (5\text{-}55)$$

The solution is also given in [Visser, 2007]:

Optimal control function: $$u^*(t) = t + 0.5 \tag{5-56}$$

Global minimum: $$J^* = {}^{23}\!/_{24} \approx 0.958333 \tag{5-57}$$

Since this is the first example, exceptionally, the optimal control function $u^*(t)$ and the global minimum J^* will be analytically calculated.

The first step to determine $u^*(t)$ is to write out the Hamiltonian using definition (5-6):

$$H(x,u,t) = L(x,u,t) + \lambda f(x,u,t) = (0.5u^2 + x) + \lambda.(u) \tag{5-58}$$

As explained in Section 5.1.1, the necessary conditions for a solution are:

$$\dot{x} = f(x,u,t) = u \tag{5-59}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial x} = -1 \tag{5-60}$$

and:

$$\frac{\partial H}{\partial u} = 0 \Leftrightarrow u + \lambda = 0 \Leftrightarrow u = -\lambda \tag{5-61}$$

Differentiating (5-59) and combining the result with (5-61) and (5-60) yields:

$$\ddot{x} = \dot{u} = -\dot{\lambda} = 1 \tag{5-62}$$

which integrated results in:

$$x(t) = 0.5t^2 + At + B \tag{5-63}$$

Two boundary conditions are required to determine the constants A and B. In this case, the initial and final states are known:

$$x(0) = 0 \Leftrightarrow B = 0 \tag{5-64}$$

$$x(1) = 1 \Leftrightarrow A = 0.5 \tag{5-65}$$

Finally, the optimal control function is given by:

$$u^*(t) = \dot{x}^*(t) = t + 0.5 \tag{5-66}$$

the optimal state trajectory by:

$$x^*(t) = 0.5t^2 + 0.5t \tag{5-67}$$

and the global minimum by:

$$J^*\left(x^*(t),u^*(t)\right)=\int_0^1\left(0.5\left(u^*(t)\right)^2+x^*(t)\right)dt={}^{23}\!/\!_{24} \qquad \textit{(5-68)}$$

As predicted in Section 5.1.1, since L and f are not explicit functions of time, the Hamiltonian is constant along an optimal trajectory, i.e.:

$$H^*=0.5\left(u^*\right)^2+x^*+\lambda u^*=0.125\Leftrightarrow \dot{H}^*=0 \qquad \textit{(5-69)}$$

In this example, two different parameterizations of the control function are tested: a second-order parameterization and a first-order parameterization.

5.3.1 Second-order parameterization

The control function parameterization used in [Chu, 2007] will also be used here. It is given by:

$$u(t)=P_2t+P_1 \qquad \textit{(5-70)}$$

Note that the global minimum J^* can be reached using this parameterization, as can be easily seen by comparing (5-70) with (5-66). The optimal solution point is given by:

$$P^*=\begin{bmatrix}0.5\\1\end{bmatrix} \qquad \textit{(5-71)}$$

5.3.1.1 Description of the developed algorithm

This quasi-static optimization problem can be solved just with HC. In other words, the parameters P_1 and P_2 can be determined from two interval equations.

The first equation follows from the initial and final state constraints: the state trajectory $x(t)$ as a function of P_1 and P_2 can be calculated from the state dynamics (5-53) as follows:

$$\dot{x}=u=P_2t+P_1\Leftrightarrow x(t)=\frac{P_2}{2}t^2+P_1t+A \qquad \textit{(5-72)}$$

After that, it is just a matter of applying the initial and the final state constraints:

$$x(0)=0\Leftrightarrow A=0 \qquad \textit{(5-73)}$$

$$x(1)=1\Leftrightarrow \boxed{\frac{P_2}{2}+P_1=1} \qquad \textit{(5-74)}$$

Expression (5-74) is the first equation. The second one is a consequence of the necessary conditions for a solution (5-4) and (5-5):

$$\frac{\partial H}{\partial u} = 0 \Leftrightarrow u + \lambda = 0 \Leftrightarrow \lambda = -u \Leftrightarrow \lambda = -P_2 t - P_1 \tag{5-75}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial x} = -1 \Leftrightarrow -P_2 = -1 \Leftrightarrow \boxed{P_2 = 1} \tag{5-76}$$

Expression (5-76) is the second equation.

Determining P_1 and P_2 from (5-74) and (5-76) is now extremely easy using HC (or even by hand). The following iterative subroutine was implemented following the instructions given in Section 4.2.1.1:

$$P_2^{'} = 1 \tag{5-77}$$

$$P_2 = P_2 \cap P_2^{'} \tag{5-78}$$

$$P_1^{'} = 1 - \frac{P_2^{'}}{2} \tag{5-79}$$

$$P_1 = P_1 \cap P_1^{'} \tag{5-80}$$

$$P_2^{'} = 2\left(1 - P_1\right) \tag{5-81}$$

$$P_2 = P_2 \cap P_2^{'} \tag{5-82}$$

From all the procedures shown in **Figure 5.1** to shrink and/or delete boxes, HC is the only one applied here.

5.3.1.2 Description of Chu's algorithm

The algorithm proposed in [Chu, 2007] is different. It has two procedures to delete boxes:

1) After evaluating the final state, which can be done for example through[8]:

$$x(1) = \left[\underline{x(1)}, \overline{x(1)}\right] = \frac{P_2}{2} + P_1 \tag{5-83}$$

any box $x(1) = \left[\underline{x(1)}, \overline{x(1)}\right]$ not containing the desired final state is eliminated. In this thesis, HC is applied to (5-74). With essentially the same amount of computing, HC yields the same or more information since some boxes may be shrunk in the process. This idea that HC is better than a simple feasibility test is defended in [Hansen and Walster, 2004].

2) The procedure of Section 5.2.2.2 based on an upper bound on the minimum is then applied to all the boxes that were not deleted in 1). Chu's algorithm has a crucial flaw: **Theorem 4.1** is not used to check the feasibility of the boxes. All the *non-infeasible* boxes are mistakenly

[8] *There is no reference in [Chu, 2007] about how the final state is actually calculated.*

assumed to be feasible. As the reader should already know, any box used to update $\overline{J}$ has first to be proved feasible with respect to (5-74), for example, by applying **Theorem 4.1**. Otherwise, P^* might not be correctly bounded.

Moreover, at the end, instead of calculating the hull of the solution set, the box yielding the minimal lower bound on the performance index is selected as the optimal solution. Note that:

1) P^* might not be contained inside this box;

2) The hypothetical bounds on P^* will be misleadingly sharp.

5.3.1.3 Comparing the algorithms

The developed algorithm was started using the following inputs:

Table 5.5: *User inputs in **Example 1**.*

Designation	Symbol		Value
Initial box	$P_{initial}$		$\begin{bmatrix}[0,1]\\ [0,1]\end{bmatrix}$
New boxes per old box	N_{split}		$[-]$
Stopping criteria	ε_P	ε_{P_1}	1.0e-3
		ε_{P_2}	1.0e-3
	ε_J		1.0e-3
Hull consistency stopping criteria	ε_{abs}		2
	ε_{rel}		1.01

The initial box given in **Table 5.5** is the same used in [Chu, 2007]. Also like in [Chu, 2007], the shrinking phase is bypassed since the problem is relatively simple.

The HC stopping criteria were chosen deliberately high to optimize the CPU time. In accordance to (5-41), N_{split} was chosen as an empty array.

In **Table 5.6**, the results produced by the developed algorithm are compared with the sharpest results presented in [Chu, 2007].

In this thesis, the final state is evaluated over a box P through (5-83) and the performance index through:

$$\begin{aligned} J &= \int_0^1 \left(0.5u^2 + x\right) dt = \\ &= \int_0^1 \left(0.5\left(P_2 t + P_1\right)^2 + \left(\frac{P_2}{2}t^2 + P_1 t\right)\right) dt = \\ &= \frac{P_2\left(P_2+1\right)}{6} + \frac{P_1\left(P_1+P_2+1\right)}{2} \end{aligned} \qquad (5\text{-}84)$$

Table 5.6: *Comparative results for **Example 1**.*

			The developed algorithm		Chu's algorithm	
Designation	**Symbol**		**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	J^I		$[0.9583, 0.9584]$	1.1e-16	$[0.943, 0.966]$	2.2e-2
Bounds on the solution point	P^I	P_1^I	$[0.5, 0.5]$	0	$[0.500, 0.501]$[9]	1.0e-3
		P_2^I	$[1,1]$	0	$[0.998, 0.999]$	1.0e-3
Bounds on the final state	X^I		$[1,1]$	0	$[0.998, 1.001]$	1.5e-3
CPU time	$\Delta t_{CPU}\,[s]$		0.02	N/A	67	N/A
Number of remaining boxes	$N_{remaining}$		1	N/A	?	N/A

Dependency is always an important issue. The form of (5-84) was somewhat moulded in order to minimize it.

The developed algorithm correctly bounds P^* while producing a thin solution interval P^I. The same cannot be said about Chu's algorithm. In addition, the developed algorithm is 3350 times faster.

As mentioned in Section 5.2.1.1, if there are N equations solvable by HC, then the number of parameters l should be bigger than N. Otherwise, the optimization problem can be solved by solving a system of (nonlinear) equations. Nevertheless, l was chosen here equal to $N = 2$ to allow a direct comparison with [Chu, 2007] and to demonstrate how HC can be used to speed up the algorithms.

Hence, increasing the order of (5-70) would make this problem more interesting. For example, the input $u(t)$ could be defined as:

$$u(t) = P_3 t^2 + P_2 t + P_1 \tag{5-85}$$

This is not done here in pro of **Example 3**.

5.3.2 First-order parameterization

This section analyses what happens if the input $u(t)$ is chosen as a constant, i.e., if:

$$u(t) = P_1 \tag{5-86}$$

Note that the global minimum J^* can <u>not</u> be reached using this parameterization, as can be easily seen by comparing (5-86) with (5-66).

Repeating the steps of Section 5.3.1.1, the state dynamics are first integrated:

[9] P_1^I *and* P_2^I *are given in a different order, probably due to a typing mistake.*

$$\dot{x} = u = P_1 \Leftrightarrow x(t) = P_1 t + A \tag{5-87}$$

Applying the initial and the final state constraints yields:

$$x(0) = 0 \Leftrightarrow A = 0 \tag{5-88}$$

and:

$$x(1) = 1 \Leftrightarrow \boxed{P_1 = 1} \tag{5-89}$$

Therefore, the solution is already known: $P_1 = 1$.

The performance index for $u(t) = 1$ is equal to:

$$J_{u=1} = \int_0^1 \left(0.5u^2 + x\right) dt = \int_0^1 \left(0.5 + t\right) dt = 1 \tag{5-90}$$

Note that:

$$J_{u=1} > J^* = {}^{23}\!/_{34} \tag{5-91}$$

As for the necessary conditions (5-4) and (5-5), notice that:

$$\frac{\partial H}{\partial u} = 0 = u + \lambda \Leftrightarrow \lambda = -u = -P_1 \Leftrightarrow \dot{\lambda} = 0 \tag{5-92}$$

$$\dot{\lambda} = -\frac{\partial H}{\partial x} = -1 \tag{5-93}$$

As the reader can see, (5-92) and (5-93) are incompatible. This means that no stationary point of J can be of the form $u(t) = P_1$ with $P_1 \in \mathbb{R}$. More specifically, $u(t) = 1$ is not a stationary point of J.

This does not mean the solution is wrong. It only means that $u(t) = 1$ is not a stationary point of J. This should not come as a surprise. Constrained optimization problems, like this one, may have their global minima at non-stationary points. Note that it is the shape of $u(t)$ that is being constrained here.

In [Chu, 2007], it is said that the only way to find a solution is to parameterize the control function as $u(t) = P_2 t + P_1$. This is not completely true. As it has been shown, if $u(t)$ is parameterized as $u(t) = P_1$, then $u(t) = 1$ is a valid solution in the sense that both $x(0) = 0$ and $x(1) = 1$ are satisfied. It is true that other parameterizations may yield better solutions, as is obvious from (5-91), but saying that $u(t) = 1$ is not a valid parameterization is not entirely correct.

5.4 Example 2

Example 2 is formulated and solved in Appendix C.

5.5 Example 3

Example 3 is a two-dimensional unconstrained problem taken from [Bryson Jr. and Ho, 1975]. It is described by:

Performance index: $$J\left(x(t),u(t),t\right)=\int_0^1 0.5u^2(t)dt \tag{5-94}$$

State dynamics: $$\dot{x}_1(t)=u(t) \tag{5-95}$$

$$\dot{x}_2(t)=x_1(t) \tag{5-96}$$

Initial state: $$x_1(0)=1 \tag{5-97}$$

$$x_2(0)=0 \tag{5-98}$$

Final state: $$x_1(1)=-1 \tag{5-99}$$

$$x_2(1)=0 \tag{5-100}$$

The solution is:

Optimal control function: $$u^*(t)=-2 \tag{5-101}$$

Global minimum: $$J^*=2 \tag{5-102}$$

Three different parameterizations of the control function are tested here: a first-order, a second-order and a third-order parameterization.

5.5.1 A first-order parameterization

The first parameterization and the only one tested in [Chu, 2007] is based on the assumption that the control function is constant:

$$u(t)=P_1 \tag{5-103}$$

The optimal solution point is obviously:

$$P^*=[-2] \tag{5-104}$$

5.5.1.1 Description of the developed algorithm

It is very simple to solve this problem using HC. Since the state boundary conditions are known, two interval equations can be written. The first one results from integrating (5-95):

$$\dot{x}_1 = u = P_1 \Leftrightarrow x_1(t) = P_1 t + A \tag{5-105}$$

and from (5-97) and (5-99):

$$x_1(0) = 1 \Leftrightarrow A = 1 \tag{5-106}$$

$$x_1(1) = -1 \Leftrightarrow \boxed{P_1 = -2} \tag{5-107}$$

Therefore, the solution is already known: $P_1 = -2$. This solution must also satisfy the second state equation:

$$\dot{x}_2 = x_1 = P_1 t + 1 \Leftrightarrow x_2(t) = \frac{P_1}{2} t^2 + t + B \tag{5-108}$$

$$x_2(0) = 0 \Leftrightarrow B = 0 \tag{5-109}$$

$$x_2(1) = 0 \Leftrightarrow \boxed{P_1 = -2} \tag{5-110}$$

and indeed it does.

Therefore, it was possible to determine P^* without any programming.

No new information is added by (5-4) and (5-5).

As for the global minimum J^*, it is given by:

$$J^* = \int_0^1 0.5\left(u^*\right)^2 dt = 0.5 P_1^2 = 2 \tag{5-111}$$

5.5.1.2 Description of Chu's algorithm

This problem was solved in [Chu, 2007] using the algorithm explained in Section 5.3.1.2 with the modifications 1) and 3) mentioned in Section C.1.1.

The initial box used in [Chu, 2007] is:

$$P_{initial} = [-4, 4] \tag{5-112}$$

5.5.1.3 Comparing the algorithms

In **Table 5.7**, the obtained results are compared with the sharpest results presented in [Chu, 2007].

Table 5.7: *Comparative results for **Example 3**.*

			The developed algorithm		Chu's algorithm	
Designation	**Symbol**		**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	J^I		$[2,2]$	0	$[1.99998, 2.00002]$	2.6e-5
Bounds on the solution point	P_1^I		$[-2,-2]$	0	$[-2.0001,-1.9999]$	1.3e-4
Bounds on the final state	X^I	X_1^I	$[-1,-1]$	0	$[-1.0001,-0.9999]$	1.3e-4
		X_2^I	$[0,0]$	0	$[-0.0001, 0.0001]$	2.0e-4
CPU time	$\Delta t_{CPU}[s]$		0	N/A	0.391	N/A
Number of remaining boxes	$N_{remaining}$		1	N/A	2	N/A

While Chu's algorithm needs some time to converge, the developed algorithm immediately yields the right solution, as demonstrated in Section 5.5.1.1.

The order of the parameterization will now be increased.

5.5.2 A second-order parameterization

In Section 5.5.1.1, it was seen that two interval equations follow from the state dynamics and the boundary conditions. Therefore, it should be possible to determine the two parameters of a second-order parameterization just by applying HC. That is in fact so.

Consider the following second-order parameterization:

$$u(t) = P_2 t + P_1 \tag{5-113}$$

where the optimal solution point is given by:

$$P^* = \begin{bmatrix} -2 \\ 0 \end{bmatrix} \tag{5-114}$$

Integrating (5-95) and imposing (5-97) and (5-99) yields:

$$\dot{x}_1 = u = P_2 t + P_1 \Leftrightarrow x_1(t) = \frac{P_2}{2} t^2 + P_1 t + A \tag{5-115}$$

$$x_1(0) = 1 \Leftrightarrow A = 1 \tag{5-116}$$

$$x_1(1) = -1 \Leftrightarrow \boxed{\frac{P_2}{2} + P_1 + 1 = -1} \tag{5-117}$$

and doing the same with (5-96), (5-98) and (5-100) results in:

$$\dot{x}_2 = x_1 = \frac{P_2}{2}t^2 + P_1 t + 1 \Leftrightarrow x_2(t) = \frac{P_2}{6}t^3 + \frac{P_1}{2}t^2 + t + B \tag{5-118}$$

$$x_2(0) = 0 \Leftrightarrow B = 0 \tag{5-119}$$

$$x_2(1) = 0 \Leftrightarrow \boxed{\frac{P_2}{6} + \frac{P_1}{2} + 1 = 0} \tag{5-120}$$

HC was applied to (5-117) and to (5-120) in the following way:

$$P_1^{'} = -2 - \frac{P_2}{2} \tag{5-121}$$

$$P_1 = P_1 \cap P_1^{'} \tag{5-122}$$

$$P_1^{'} = -2 - \frac{P_2}{3} \tag{5-123}$$

$$P_1 = P_1 \cap P_1^{'} \tag{5-124}$$

$$P_2^{'} = -4 - 2P_1 \tag{5-125}$$

$$P_2 = P_2 \cap P_2^{'} \tag{5-126}$$

$$P_2^{'} = -6 - 3P_1 \tag{5-127}$$

$$P_2 = P_2 \cap P_2^{'} \tag{5-128}$$

Since this problem is very simple, introducing a shrinking phase would not improve the results.

The final state formulas, (5-115) and (5-118), are not affected by dependency, but the formula of the performance index is:

$$J = \int_0^1 0.5u^2 dt = \int_0^1 0.5(P_2 t + P_1)^2 dt = \frac{P_2^2 + 3P_1P_2 + 3P_1^2}{6} \tag{5-129}$$

The results shown in **Table 5.9** were obtained by introducing the inputs given in **Table 5.8**. As expected, HC correctly bounds P^*. No other procedure is necessary. N_{split} was chosen as an empty array, in accordance to (5-41).

The order of the parameterization will now be increased again.

Table 5.8: *User inputs (second-order parameterization).*

Designation	Symbol		Value
Initial box	$P_{initial}$		$\begin{bmatrix}[-4,4]\\ [-4,4]\end{bmatrix}$
New boxes per old box	N_{split}		$[-]$
Stopping criteria	ε_P	ε_{P_1}	1.0e-3
		ε_{P_2}	1.0e-3
	ε_J		1.0e-3
Hull consistency stopping criteria	ε_{abs}		1.0e-3
	ε_{rel}		0.1

Table 5.9: *Results (second-order parameterization).*

Designation	Symbol		Result	Width
Bounds on the minimum	J^I		$[1.9995, 2.0005]$	2.4e-4
Bounds on the solution point	P^I	P_1^I	$[-2.0002, -1.9998]$	4.8e-4
		P_2^I	$[-0.0003, 0.0003]$	5.5e-4
Bounds on the final state	X^I	X_1^I	$[-1.0003, -0.9997]$	4.8e-4
		X_2^I	$[-0.0001, 0.0001]$	2.0e-4
CPU time	$\Delta t_{CPU}\,[s]$		0.34	N/A
Number of remaining boxes	$N_{remaining}$		1	N/A

5.5.3 A third-order parameterization

Consider the next third-order parameterization:

$$u(t) = P_3 t^2 + P_2 t + P_1 \tag{5-130}$$

with:

$$P^* = \begin{bmatrix}-2 & 0 & 0\end{bmatrix}^T \tag{5-131}$$

Now, only two equations are available for finding three parameters. Obviously, this is not enough. A true optimization problem needs to be solved now. The procedure discussed in Section 5.2.2.2 based on an upper bound on the minimum will have to be used.

First, the HC equations need to be deduced. Integrating (5-95) and imposing (5-97) and (5-99) yields:

$$\dot{x}_1 = u = P_3 t^2 + P_2 t + P_1 \Leftrightarrow x_1(t) = \frac{P_3}{3}t^3 + \frac{P_2}{2}t^2 + P_1 t + A \tag{5-132}$$

$$x_1(0)=1 \Leftrightarrow A=1 \tag{5-133}$$

$$x_1(1)=-1 \Leftrightarrow \boxed{\frac{P_3}{3}+\frac{P_2}{2}+P_1+1=-1} \tag{5-134}$$

and doing the same with (5-96), (5-98) and (5-100) results in:

$$\dot{x}_2=x_1=\frac{P_3}{3}t^3+\frac{P_2}{2}t^2+P_1t+1 \Leftrightarrow x_2(t)=\frac{P_3}{12}t^4+\frac{P_2}{6}t^3+\frac{P_1}{2}t^2+t+B \tag{5-135}$$

$$x_2(0)=0 \Leftrightarrow B=0 \tag{5-136}$$

$$x_2(1)=0 \Leftrightarrow \boxed{\frac{P_3}{12}+\frac{P_2}{6}+\frac{P_1}{2}+1=0} \tag{5-137}$$

HC was applied to (5-134) and to (5-137) in the following way:

$$P_1^{'}=-2-\frac{P_2}{2}-\frac{P_3}{3} \tag{5-138}$$

$$P_1=P_1\cap P_1^{'} \tag{5-139}$$

$$P_1^{'}=-2-\frac{P_2}{3}-\frac{P_3}{6} \tag{5-140}$$

$$P_1=P_1\cap P_1^{'} \tag{5-141}$$

$$P_2^{'}=-4-2P_1-\frac{2P_3}{3} \tag{5-142}$$

$$P_2=P_2\cap P_2^{'} \tag{5-143}$$

$$P_2^{'}=-6-3P_1-\frac{P_3}{2} \tag{5-144}$$

$$P_2=P_2\cap P_2^{'} \tag{5-145}$$

$$P_3^{'}=-6-3P_1-\frac{3P_2}{2} \tag{5-146}$$

$$P_3=P_3\cap P_3^{'} \tag{5-147}$$

$$P_3^{'}=-12-6P_1-2P_2 \tag{5-148}$$

$$P_3=P_3\cap P_3^{'} \tag{5-149}$$

Note that (5-138)-(5-149) are not effected by dependency.

As discussed in Section 5.2.2.2.3, before using a box P to update $\overline{J}$, P must be proved feasible with respect to the equality constraints (5-134) and (5-137). Since there are three unknown parameters and just two equations, in order to apply **Theorem 4.1**, one interval parameter must be fixed, that is, one interval parameter must be substituted by its midpoint.

Hence, the next step is to choose two HC equations from (5-138)-(5-149) for applying **Theorem 4.1**. When choosing, one should bear in mind that the chances that $P^{'} \subset P$ are higher if $P^{'}$ is much smaller than P. After some trial-and-error, the following subroutine was devised:

1) $$P_{3,m} = m(P_3) \qquad (5\text{-}150)$$

2) $$P_1^{'} = -2 - \frac{P_2}{2} - \frac{P_{3,m}}{3} \qquad (5\text{-}151)$$

3) $$P_2^{'} = -4 - 2P_1^{'} - \frac{2P_{3,m}}{3} \qquad (5\text{-}152)$$

4) If $P_1^{'} \subset P_1$ and $P_2^{'} \subset P_2$, then **Theorem 4.1** guarantees that there is a solution of (5-134) and of (5-137) inside $P^{'} = \begin{bmatrix} P_1^{'} & P_2^{'} & P_{3,m} \end{bmatrix}^T$. Thus, $P^{'}$ can be used to update $\overline{J}$.

The final state is calculated from (5-132) and (5-135). Dependency does not affect the calculated results. The same cannot be said about the performance index:

$$J = \int_0^1 0.5u^2 dt = \int_0^1 0.5\left(P_3t^2 + P_2t + P_1\right)^2 dt =$$

$$= \frac{30P_1^2 + P_3\left(15P_2 + 20P_1\right) + 10P_2^2 + 30P_1P_2 + 6P_3^2}{60} \qquad (5\text{-}153)$$

$$= \frac{30P_1^2 + P_2\left(30P_1 + 15P_3\right) + 20P_1P_3 + 10P_2^2 + 6P_3^2}{60} \qquad (5\text{-}154)$$

To minimize the dependency effect, the performance index is calculated as the intersection of (5-153) and (5-154).

The results shown in **Table 5.11** were obtained by introducing the inputs given in **Table 5.10**.

The following comments are due:

1) P^* was correctly bounded.

2) The number of remaining boxes is considerably higher than in previous examples. This shows how much more demanding it is to solve (true) dynamic optimization problems compared with static optimization problems.

3) P_1^I, P_2^I and P_3^I are substantially wider than J^I. This is not really a problem since in most real life applications a point must be selected from P^I (the usefulness of interval solutions is still very limited). As explained in Section 5.2.5, when $\overline{J} < \overline{\mathrm{J}}$, which is the case,

the best option is to select the midpoint p of the box where the last upper bound on the minimum $\overline{J}$ was calculated, i.e., the midpoint of the box $P^{'} = \begin{bmatrix} P_1^{'} & P_2^{'} & P_{3,m} \end{bmatrix}^T$. Then, it is guaranteed that:

$$\left| J\left(p,t\right) - J\left(P^*,t\right) \right| \leq \overline{J} - \underline{J} = w\left(J^I\right) \leq \varepsilon_J \qquad (5\text{-}155)$$

Table 5.10: *User inputs (third-order parameterization).*

Designation	Symbol		Value
Initial box	$P_{initial}$		$\begin{bmatrix} [-2,2] \\ [-2,2] \\ [-2,2] \end{bmatrix}$
New boxes per old box	N_{split}		$\begin{bmatrix} 0 & 0 & 2 \end{bmatrix}^T$
Initial upper bound on the minimum	$\overline{J}$		$+\infty$
Stopping criteria	ε_P	ε_{P_1}	$+\infty$
		ε_{P_2}	$+\infty$
		ε_{P_3}	$+\infty$
	ε_J		1.0e-3
Hull consistency stopping criteria	ε_{abs}		0.1
	ε_{rel}		0.05

Table 5.11: *Results (third-order parameterization).*

Designation	Symbol		Result	Width
Bounds on the minimum	J^I		$[1.9994, 2.0003]$	8.7e-4
Bounds on the solution point	P^I	P_1^I	$[-2.0000, -1.9034]$	9.7e-2
		P_2^I	$[-0.5789, 0.0003]$	5.8e-1
		P_3^I	$[-0.0005, 0.5787]$	5.8e-1
Bounds on the final state	X^I	X_1^I	$[-1.0003, -0.9997]$	5.7e-4
		X_2^I	$[-0.0002, 0.0002]$	2.3e-4
CPU time	$\Delta t_{CPU}\,[s]$		4.3	N/A
Number of remaining boxes	$N_{remaining}$		4744	N/A

Therefore, more important than having sharp bounds on P^*, is to have sharp bounds on J^*. The maximal width of J^I can be defined by the user through ε_J.

Note however that, as explained in Section 5.2.5, in all probability, p will not be a feasible point with respect to the equality constraints. That is, in all probability, the final state will not be exactly reached if p is used.

By evaluating the final state at p:

$$x_i\left(p,t_f\right)=\left[\underline{x_{i_f}},\overline{x_{i_f}}\right] \quad ,i=1,\dots,n \tag{5-156}$$

it is guaranteed that:

$$\left|x_i\left(p,t_f\right)-x_i\left(P^*,t_f\right)\right|\le \max\left\{\left(\overline{x_{i_f}}-x_i\left(P^*,t_f\right)\right),\left(x_i\left(P^*,t_f\right)-\underline{x_{i_f}}\right)\right\} \tag{5-157}$$

with $i=1,\dots,n$. Therefore, another possible stopping condition for the dynamic optimization algorithm is:

$$\max\left\{\left(\overline{x_{i_f}}-x_i\left(P^*,t_f\right)\right),\left(x_i\left(P^*,t_f\right)-\underline{x_{i_f}}\right)\right\}\le \varepsilon_{eq_i} \tag{5-158}$$

with $i=1,\dots,n$ and with the n parameters ε_{eq_i} given by the user. A similar stopping condition is suggested in [Hansen and Walster, 2004].

4) P_2^I and P_3^I are substantially wider than P_1^I. To understand why, **Figure 5.3** is given. In it, the performance index J is plotted as a function of P_1, P_2 and P_3 around P^*. As can be seen, the slope of $J\left(P_1\right)$ around P^* is relatively higher than the slope of $J\left(P_2\right)$ and $J\left(P_3\right)$. As a result, it is easier for the BB algorithm to find sharp bounds on P_1^* than on the other parameters.

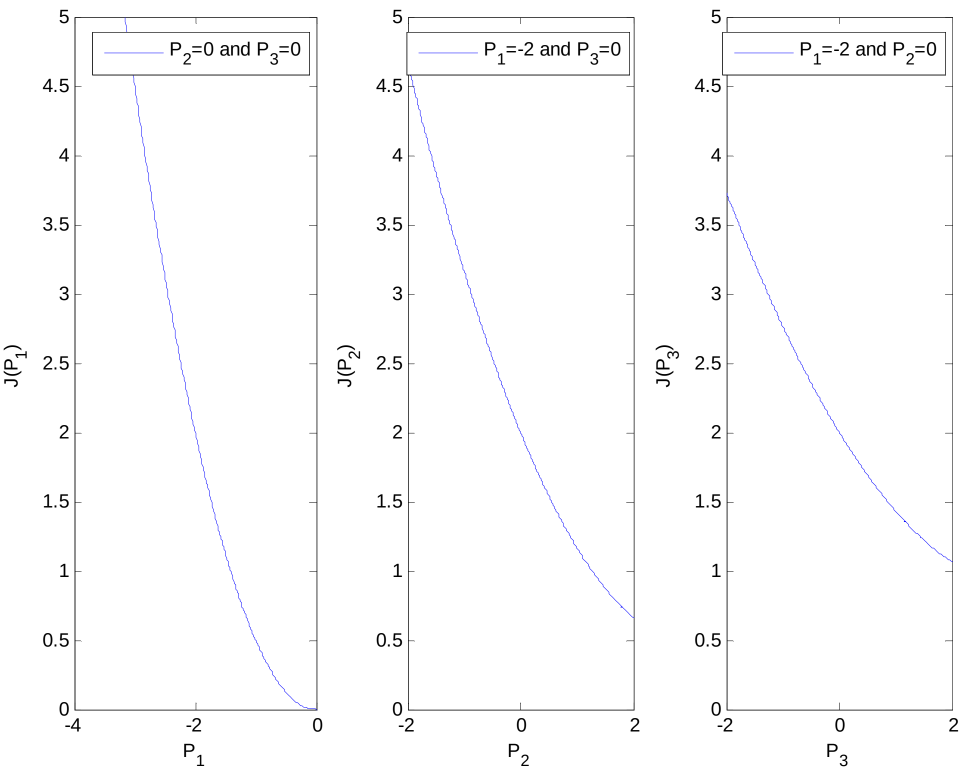

Figure 5.3: *Performance index as a function of* P_1, P_2 *and* P_3.

5) Using the same inputs as before (see **Table 5.10**) together with $N_{shrinking}=100$ and $\varepsilon_{shrinking}=0.1$, adding a shrinking phase results in an increase of the CPU time from

approximately 4.3s to 6.1s, for the same desired accuracy. Even with three parameters, it is still faster not to use a shrinking phase.

6) In **Table 5.12**, Δt_{CPU} is given as a function of N_{split}. All other inputs are kept constant (see **Table 5.10**).

Table 5.12*: CPU time as a function of* N_{split} *in* ***Example 3****.*

Number of parameters being split	**New boxes per old box** N_{split}	**CPU time** $\Delta t_{CPU}\left[s\right]$
1	$\begin{bmatrix}2 & 0 & 0\end{bmatrix}^T$	No convergence after 30s
	$\begin{bmatrix}0 & 2 & 0\end{bmatrix}^T$	3.7
	$\begin{bmatrix}0 & 0 & 2\end{bmatrix}^T$	4.3
2	$\begin{bmatrix}2 & 2 & 0\end{bmatrix}^T$	4.7
	$\begin{bmatrix}2 & 0 & 2\end{bmatrix}^T$	No convergence after 30s
	$\begin{bmatrix}0 & 2 & 2\end{bmatrix}^T$	6.4
3	$\begin{bmatrix}2 & 2 & 2\end{bmatrix}^T$	No convergence after 30s

As predicted by rule of thumb (5-41), splitting just one parameter yields the fastest results.

On the other hand, in Section 5.2.4 it is suggested that in this case, P_3 would be the best parameter to split, as a result of (5-150). However, **Table 5.12** shows that the best results are obtained by splitting P_2 and not P_3. (5-151) and (5-152) show why. Splitting P_2 helps satisfying $P_1^{'} \subset P_1$ but it makes it harder to satisfy $P_2^{'} \subset P_2$. Apparently here, the first effect has predominance. Nevertheless, splitting P_3, as suggested in Section 5.2.4, is almost as fast as splitting P_2, which makes it a good initial guess.

It should be noted that the selection of N_{split} should be done in strict connection with the selection of the HC equations used to prove feasibility, in this case, (5-151) and (5-152). It is not correct to say that a certain N_{split} is always the best; only that a certain N_{split} is the best with respect to a certain set of HC equations.

7) To prove feasibility, two equations had to be chosen from the six HC equations (5-138)-(5-149). Always keeping in mind that the chances of $P^{'} \subset P$ are higher if $P^{'}$ is much smaller than P, the two equations with the smallest coefficients multiplying P_1, P_2 and P_3 were chosen. It would be better if these coefficients were all smaller than one.

5.6 Example 4

Example 4 is formulated and solved in Appendix C.

5.7 Example 5

The only difference between **Example 5** and **Example 4** is that the state inequality constraint (5-166) is more restrictive in **Example 5**.

Performance index: $$J\left(x(t),u(t),t\right)=\int_0^1 0.5u^2(t)dt \qquad (5\text{-}159)$$

State dynamics: $$\dot{x}_1(t)=u(t) \qquad (5\text{-}160)$$

$$\dot{x}_2(t)=x_1(t) \qquad (5\text{-}161)$$

Initial state: $$x_1(0)=1 \qquad (5\text{-}162)$$

$$x_2(0)=0 \qquad (5\text{-}163)$$

Final state: $$x_1(1)=-1 \qquad (5\text{-}164)$$

$$x_2(1)=0 \qquad (5\text{-}165)$$

State inequality constraint: $$x_2(t)\le 0.125 \quad ,0\le t\le 1 \qquad (5\text{-}166)$$

The analytical solution is given in [Bryson Jr. and Ho, 1975]:

Optimal control function: $$u^*(t)=\begin{cases}\frac{128}{9}t-\frac{16}{3} & ,0\le t\le \frac{3}{8}\\ 0 & ,\frac{3}{8}\le t\le \frac{5}{8}\\ -\frac{128}{9}t+\frac{80}{9} & ,\frac{5}{8}\le t\le 1\end{cases} \qquad (5\text{-}167)$$

Global minimum: $$J^*=\frac{32}{9}\simeq 3.55556 \qquad (5\text{-}168)$$

Example 5 has a higher global minimum J^* than **Example 4** because (5-166) is more restrictive.

5.7.1 Description of Chu's algorithm

The following parameterization is suggested in [Chu, 2007]:

$$u(t) = \begin{cases} P_3 t + P_4 & , 0 \le t \le \underline{P_1} \\ (P_3 t + P_4) \cap [-\infty, 0] & , \underline{P_1} < t < \overline{P_1} \\ 0 & , \overline{P_1} \le t \le \underline{P_2} \\ (P_5 t + P_6) \cap [-\infty, 0] & , \underline{P_2} < t < \overline{P_2} \\ P_5 t + P_6 & , \overline{P_2} \le t \le 1 \end{cases} \tag{5-169}$$

with:

$$P_{initial} = \begin{bmatrix} [0.35, 0.40] \\ [0.60, 0.65] \\ [13.0, 15.0] \\ [-8.0, -5.0] \\ [-15.0, -13.0] \\ [8.0, 10.0] \end{bmatrix} \tag{5-170}$$

and:

$$P^* = \begin{bmatrix} \frac{3}{8} & \frac{5}{8} & \frac{128}{9} & -\frac{16}{3} & -\frac{128}{9} & \frac{80}{9} \end{bmatrix}^T \tag{5-171}$$

Note that it might happen that, for a particular box, $P_3 t + P_4 > 0$ for $t \in P_1$ and/or $P_5 t + P_6 > 0$ for $t \in P_2$. In that case, (5-169) will lead to $u(t) = \varnothing$.

This example was solved in [Chu, 2007] using the algorithm explained in Section 5.3.1.2 with the modifications enumerated in Section C.1.1.

5.7.2 Description of the developed algorithm

Without making further assumptions, the 6th order parameterization (5-169) can be rewritten as a 4th order parameterization as follows:

$$u(t) = \begin{cases} \frac{-P_3}{P_1} t + P_3 & , 0 \le t \le P_1 \\ 0 & , P_1 \le t \le P_2 \\ \frac{P_4}{1 - P_2}(t - 1) + P_4 & , P_2 \le t \le 1 \end{cases} \tag{5-172}$$

or in a more detailed form:

$$u(t) = \begin{cases} \frac{-P_3}{P_1}t + P_3 & , 0 \le t \le \underline{P_1} \\ \left(\frac{-P_3}{P_1}t + P_3\right) \cup (0) & , \underline{P_1} \le t \le \overline{P_1} \\ 0 & , \overline{P_1} \le t \le \underline{P_2} \\ (0) \cup \left(\frac{P_4}{1-P_2}(t-1) + P_4\right) & , \underline{P_2} \le t \le \overline{P_2} \\ \frac{P_4}{1-P_2}(t-1) + P_4 & , \overline{P_2} \le t \le 1 \end{cases} \qquad (5\text{-}173)$$

where:

$$P^* = \begin{bmatrix} \frac{3}{8} & \frac{5}{8} & -\frac{16}{3} & -\frac{16}{3} \end{bmatrix}^T \qquad (5\text{-}174)$$

The meaning of P_1, P_2, P_3 and P_4 can be understood from **Figure 5.4**, where they are assumed to be point interval variables.

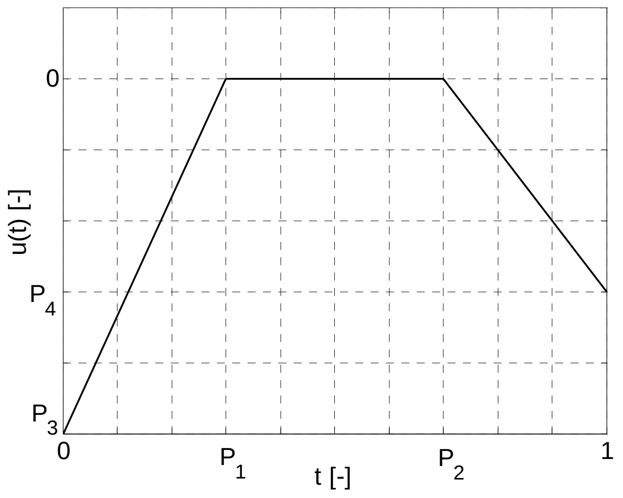

Figure 5.4: *Illustration of the parameterization used to solve* ***Example 5****.*

By looking at (5-169) and (5-172), it is possible to relate Chu's parameterization (identified by the superscript ^{C}) with the one used in this thesis (superscript ^{F}).

$$P_1^F = P_1^C \qquad (5\text{-}175)$$

$$P_2^F = P_2^C \qquad (5\text{-}176)$$

$$u^F(t=0) = u^C(t=0) \Leftrightarrow P_3^F = P_4^C \qquad (5\text{-}177)$$

$$u^F(t=1) = u^C(t=1) \Leftrightarrow P_4^F = P_5^C + P_6^C \qquad (5\text{-}178)$$

For the comparison with [Chu, 2007] to be valid, the initial box used in this thesis must be equivalent to Chu's initial box. Therefore, (5-170) was inserted into (5-175)-(5-178) and the result was taken as the initial box used in this thesis:

$$P_{initial} = \begin{bmatrix} [0.35, 0.4] \\ [0.6, 0.65] \\ [-8, -5] \\ [-7, -3] \end{bmatrix} \qquad (5\text{-}179)$$

Two interval equations can be deduced from the state dynamics and the initial and final state constraints. Integrating (5-160) yields:

$$\begin{cases} \dot{x}_1 = u = \dfrac{-P_3}{P_1} t + P_3 & , 0 \le t \le P_1 \\ \dot{x}_1 = u = 0 & , P_1 \le t \le P_2 \\ \dot{x}_1 = u = \dfrac{P_4}{1-P_2}(t-1) + P_4 & , P_2 \le t \le 1 \end{cases}$$

$$\Leftrightarrow \qquad (5\text{-}180)$$

$$\begin{cases} x_1^{(1)}(t) = \dfrac{-P_3}{2P_1} t^2 + P_3 t + A & , 0 \le t \le P_1 \\ x_1^{(2)}(t) = B & , P_1 \le t \le P_2 \\ x_1^{(3)}(t) = \dfrac{P_4}{2(1-P_2)}(t-1)^2 + P_4 t + C & , P_2 \le t \le 1 \end{cases}$$

The superscripts $^{(1)}$, $^{(2)}$ and $^{(3)}$ are used to identify the different phases.

A, B and C can be determined by imposing the following conditions:

$$x_1^{(1)}(0) = 1 \Leftrightarrow A = 1 \qquad (5\text{-}181)$$

$$x_1^{(1)}(P_1) = x_1^{(2)}(P_1) \Leftrightarrow B = \frac{P_3 P_1}{2} + 1 \qquad (5\text{-}182)$$

$$x_1^{(3)}(1) = -1 \Leftrightarrow C = -1 - P_4 \qquad (5\text{-}183)$$

The first equation follows from equalizing the last two phases of (5-180) at $t = P_2$, i.e.:

$$x_1^{(2)}(P_2) = x_1^{(3)}(P_2) \qquad (5\text{-}184)$$

resulting in:

$$\boxed{q_1(P) \triangleq -\frac{P_3 P_1}{2} + \frac{P_4 P_2}{2} - \frac{P_4}{2} - 2 = 0} \qquad (5\text{-}185)$$

On the other hand, integrating (5-161) yields:

$$\begin{cases} \dot{x}_2 = x_1 = \dfrac{-P_3}{2P_1}t^2 + P_3 t + 1 & ,0 \le t \le P_1 \\ \dot{x}_2 = x_1 = \dfrac{P_3 P_1}{2} + 1 & ,P_1 \le t \le P_2 \\ \dot{x}_2 = x_1 = \dfrac{P_4}{2\left(1-P_2\right)}\left(t-1\right)^2 + P_4 t - 1 - P_4 & ,P_2 \le t \le 1 \end{cases}$$

$$\Leftrightarrow \tag{5-186}$$

$$\begin{cases} x_2^{(1)}\left(t\right) = \dfrac{-P_3}{6P_1}t^3 + \dfrac{P_3}{2}t^2 + t + D & ,0 \le t \le P_1 \\ x_2^{(2)}\left(t\right) = \left(\dfrac{P_3 P_1}{2} + 1\right)t + E & ,P_1 \le t \le P_2 \\ x_2^{(3)}\left(t\right) = \dfrac{P_4}{6\left(1-P_2\right)}\left(t-1\right)^3 + \dfrac{P_4}{2}t^2 + \left(-1-P_4\right)t + F & ,P_2 \le t \le 1 \end{cases}$$

D, E and F can be determined by imposing the following conditions:

$$x_2^{(1)}\left(0\right) = 0 \Leftrightarrow D = 0 \tag{5-187}$$

$$x_2^{(1)}\left(P_1\right) = x_2^{(2)}\left(P_1\right) \Leftrightarrow E = -\frac{P_3 P_1^2}{6} \tag{5-188}$$

$$x_2^{(3)}\left(1\right) = 0 \Leftrightarrow D = 1 + \frac{P_4}{2} \tag{5-189}$$

The second equation follows from equalizing the last two branches of (5-186) at $t = P_2$, i.e.:

$$x_2^{(2)}\left(P_2\right) = x_2^{(3)}\left(P_2\right) \tag{5-190}$$

resulting in:

$$-2P_2 + 1 - \frac{2P_4 P_2}{3} + \frac{P_4 P_2^2}{3} + \frac{P_4}{3} + \frac{P_3 P_1^2}{6} - \frac{P_3 P_1 P_2}{2} = 0 \tag{5-191}$$

By combining (5-185) with (5-191), the latter can be simplified, resulting in:

$$\boxed{q_2\left(P\right) \triangleq 1 - \frac{P_4 P_2}{6} - \frac{P_4 P_2^2}{6} + \frac{P_4}{3} + \frac{P_3 P_1^2}{6} = 0} \tag{5-192}$$

HC was applied to (5-185) and to (5-192) in the following way (the intermediate intersection steps are not shown here):

$$P_1^{'} = \frac{-2 + P_4\left(\dfrac{P_2}{2} - \dfrac{1}{2}\right)}{\dfrac{P_3}{2}} \tag{5-193}$$

$$P_1' = \sqrt{\frac{P_4\left(P_2\left(\frac{1}{6}+\frac{P_2}{6}\right)-\frac{1}{3}\right)-1}{\frac{P_3}{6}}} \quad (5\text{-}194)$$

$$P_2' = \frac{-\frac{P_3 P_1}{2}-2}{-\frac{P_4}{2}}+1 \quad (5\text{-}195)$$

$$P_2' = \frac{1+P_4\left(-\frac{P_2^2}{6}+\frac{1}{3}\right)+\frac{P_3 P_1^2}{6}}{\frac{P_4}{6}} \quad (5\text{-}196)$$

$$P_3' = \frac{P_4\left(P_2-1\right)-4}{P_1} \quad (5\text{-}197)$$

$$P_3' = \frac{-1+P_4\left(P_2\left(\frac{1}{6}+\frac{P_2}{6}\right)-\frac{1}{3}\right)}{\frac{P_1^2}{6}} \quad (5\text{-}198)$$

$$P_4' = \frac{\frac{P_3 P_1}{2}+2}{\frac{P_2}{2}-\frac{1}{2}} \quad (5\text{-}199)$$

$$P_4' = \frac{-1-\frac{P_3 P_1^2}{6}}{P_2\left(-\frac{1}{6}-\frac{P_2}{2}\right)+\frac{1}{3}} \quad (5\text{-}200)$$

The dependency effect was taken into account when writing (5-193)-(5-200).

Since there are four parameters and just two equations, the procedure explained in Section 5.2.2.2.4 based on an upper bound on the minimum has to be used. The first step is to prove the feasibility of a box P with respect to the equality constraints. In order to apply **Theorem 4.1**, two interval parameters have to be fixed. Since the initial bounds on P_1 and P_2 are relatively sharper than the initial bounds on P_3 and P_4 and following the suggestion given in Section 5.2.4, P_1 and P_2 were initially fixed. Then, the following procedure was used to prove feasibility:

1) $$P_{1,m} = m(P_1) \quad (5\text{-}201)$$

2) $$P_{2,m} = m(P_2) \quad (5\text{-}202)$$

3)
$$P_3^{'} = \frac{-1 + P_4\left(P_{2,m}\left(\frac{1}{6} + \frac{P_{2,m}}{6}\right) - \frac{1}{3}\right)}{\frac{P_{1,m}^2}{6}} \quad (5\text{-}203)$$

4)
$$P_4^{'} = \frac{\frac{P_3^{'}P_{1,m}}{2} + 2}{\frac{P_{2,m}}{2} - \frac{1}{2}} \quad (5\text{-}204)$$

5) If $P_3^{'} \subset P_3$ and $P_4^{'} \subset P_4$, then **Theorem 4.1** guarantees that there is a solution of (5-185) and of (5-192) inside $P^{'} = \begin{bmatrix} P_{1,m} & P_{2,m} & P_3^{'} & P_4^{'} \end{bmatrix}^T$. Thus, $P^{'}$ can be used to update $\overline{J}$.

Note that (5-203) and (5-204) are not affected by dependency.

After a few tests, it became apparent that proving feasibility with the above procedure was not possible, reinforcing the idea that the suggestions given in Section 5.2.4 should not be taken as general rules. After some trial-and-error, the following procedure was devised:

1)
$$P_{1,m} = m(P_1) \quad (5\text{-}205)$$

2)
$$P_{3,m} = m(P_3) \quad (5\text{-}206)$$

3)
$$P_4^{'} = \frac{-1 - \frac{P_{3,m}P_{1,m}^2}{6}}{P_2\left(-\frac{1}{6} - \frac{P_2}{2}\right) + \frac{1}{3}} \quad (5\text{-}207)$$

4)
$$P_2^{'} = \frac{-\frac{P_{3,m}P_{1,m}}{2} - 2}{-\frac{P_4^{'}}{2}} + 1 \quad (5\text{-}208)$$

5) If $P_4^{'} \subset P_4$ and $P_2^{'} \subset P_2$, then **Theorem 4.1** guarantees that there is a solution of (5-185) and of (5-192) inside $P^{'} = \begin{bmatrix} P_{1,m} & P_2^{'} & P_{3,m} & P_4^{'} \end{bmatrix}^T$. Thus, $P^{'}$ can be used to update $\overline{J}$.

(5-208) is not affected by dependency but (5-207) is.

Trial-and-error also showed that the above procedure yields good results with $N_{split} = \begin{bmatrix} 0 & 2 & 2 & 0 \end{bmatrix}^T$.

Proving that $P^{'}$ is feasible with respect to the equality constraints (5-185) and of (5-192) is only the first step. $P^{'}$ must also be proven feasible with respect to the inequality constraint (5-166). So sharp bounds on $x_2(t)$ could be calculated, or in other words, to reduce the dependency effect, (5-186) was rewritten as follows:

$$\begin{cases} x_2^{(1)}(T)=T\left(\left(-\frac{T}{6P_1}+\frac{1}{2}\right)P_3T+1\right) & ,T=\left[0,\underline{P_1}\right] \\ x_2^{(2)}(t)=P_1\left(\frac{P_3P_1}{3}+1\right) & ,t=P_1 \\ x_2^{(3)}(t)=\left(\frac{P_3P_1}{2}+1\right)t-\frac{P_3P_1^2}{6} & ,\overline{P_1}\le t\le \underline{P_2} \\ x_2^{(4)}(t)=\left(P_3P_1\left(-\frac{P_1}{6}+\frac{P_2}{2}\right)+P_2\right)\cup\left(P_4\left(\left(\frac{P_2}{3}-\frac{2}{3}\right)P_2+\frac{1}{3}\right)-P_2+1\right) & ,t=P_2 \\ x_2^{(5)}(T)=P_4\left(\frac{(T-1)^3}{6(1-P_2)}+\left(\frac{T}{2}-1\right)T+\frac{1}{2}\right)-T+1 & ,T=\left[\overline{P_2},1\right] \end{cases} \quad (5\text{-}209)$$

Note that $x_2^{(3)}(t)$ is a linear function of t. Therefore, it is not necessary to check if:

$$x_2^{(3)}(t)\le 0.125 \quad (5\text{-}210)$$

for every $\overline{P_1}\le t\le \underline{P_2}$. It is enough to check if $x_2^{(2)}(t)$ and $x_2^{(4)}(t)$ satisfy the inequality constraint.

To reduce the dependency effect due to T in the calculation of $x_2^{(1)}(T)$ and $x_2^{(5)}(T)$, the time intervals $\left[0,\underline{P_1}\right]$ and $\left[\overline{P_2},1\right]$ were divided in N_{ineq} subintervals each, as explained in Section 5.2.2.1.

In **Example 4**, N_{ineq} is chosen as a (big) constant. As a consequence, during the first cycles of the algorithm, when the bounds on the parameters are still wide, an unnecessarily high effort is put on splitting the time intervals T: there is no point in dividing the time intervals T in very sharp subintervals if the bounds on the parameters are still relatively wide. Therefore, N_{ineq} should be dynamically chosen every cycle taking into account how sharp are the bounds on the parameters. While the idea itself makes sense, many different formulas can be suggested for N_{ineq}. After some trial-and-error, N_{ineq} was empirically defined as follows:

$$N_{ineq}=\operatorname{round}\left(\frac{\underset{i}{\operatorname{mean}}\left(w\left(T^i\right)\right)}{\frac{\underset{j}{\max}\left(\underset{i}{\operatorname{mean}}\left(w\left(P_j^i\right)\right)\right)}{10}}\right) \quad ,j=1,\dots,l \quad (5\text{-}211)$$

where the function $\operatorname{round}(x)$ rounds x to the nearest integer. P_j^i represents the j-th parameter of the i-th box. The function $\underset{i}{\operatorname{mean}}$ calculates the mean value with respect to all the boxes.

Example 4 can be solved in 284s, instead of the 554s showed in **Table C.4**, by substituting $N_{ineq} = 1000$ with (5-211) using $T = \left[0, \underline{P_1}\right]$.

Example 5 is solved using (5-211) with $T = \left[0, \underline{P_1}\right]$. The time interval $\left[\overline{P_2}, 1\right]$ is divided in the same number of subintervals as $\left[0, \underline{P_1}\right]$.

(5-209) is used both for proving feasibility of a box P' and for proving infeasibility of a box P with respect to the inequality constraint.

The performance index can be calculated as follows:

$$J = \int_0^1 0.5u^2 dt = \underbrace{\int_0^{\underline{P_1}} 0.5\left(\frac{-P_3}{P_1}t + P_3\right)^2 dt}_{1^{st}\ parcel} + \underbrace{\int_{\underline{P_1}}^{\overline{P_1}} 0.5\left(\left(\frac{-P_3}{P_1}t + P_3\right) \cup (0)\right)^2 dt}_{2^{nd}\ parcel} +$$
$$+ \underbrace{\cancel{\int_{\overline{P_1}}^{\underline{P_2}} 0.5(0)^2 dt}}_{3^{rd}\ parcel} + \underbrace{\int_{\underline{P_2}}^{\overline{P_2}} 0.5\left((0) \cup \left(\frac{P_4}{1-P_2}(t-1) + P_4\right)\right)^2 dt}_{4^{th}\ parcel} + \underbrace{\int_{\overline{P_2}}^{1} 0.5\left(\frac{P_4}{1-P_2}(t-1) + P_4\right)^2 dt}_{5^{th}\ parcel} \qquad (5\text{-}212)$$

The first and fifth parcels were calculated analytically. The second and the fourth were calculated using the rectangular method with $N_{\text{int}} = 1$.

$$J = \underbrace{\frac{\underline{P_1} P_3^2 \left(3P_1^2 - 3P_1 \underline{P_1} + \underline{P_1}^2\right)}{6P_1^2}}_{1^{st}\ parcel} + \underbrace{0.5\left(\frac{-P_3}{P_1}P_1 + P_3\right)^2 P_1}_{2^{nd}\ parcel} + \underbrace{0.5\left(\frac{P_4}{1-P_2}(P_2 - 1) + P_4\right)^2 P_2}_{4^{th}\ parcel} -$$
$$- \underbrace{\frac{\left(\overline{P_2} - 1\right) P_4^2}{6\left(P_2 - 1\right)^2}\left(3P_2^2 - 3P_2\left(\overline{P_2} + 1\right) + \overline{P_2}^2 + \overline{P_2} + 1\right)}_{5^{th}\ parcel} \qquad (5\text{-}213)$$

Note that:

$$0 \in -\frac{P_3}{P_1}t + P_3 \quad , t = P_1 \qquad (5\text{-}214)$$

and:

$$0 \in \frac{P_4}{1-P_2}(t-1) + P_4 \quad , t = P_2 \qquad (5\text{-}215)$$

were used to write (5-213). Dependency was also taken into account when writing (5-213).

5.7.3 Comparing the algorithms

The inputs given to the developed algorithm are shown in **Table 5.13**.

Table 5.13: *User inputs in* ***Example 5****.*

Designation	Symbol		Value
New boxes per old box	N_{split}		$\begin{bmatrix}0 & 2 & 2 & 0\end{bmatrix}^T$
Initial upper bound on the minimum	$\overline{J}$		$+\infty$
Stopping criteria	ε_P	ε_{P_1}	$+\infty$
		ε_{P_2}	$+\infty$
		ε_{P_3}	$+\infty$
		ε_{P_4}	$+\infty$
	ε_J		1.29e-1
Hull consistency stopping criteria	ε_{abs}		0.01
	ε_{rel}		0.05

In **Table 5.14**, the results produced by the developed algorithm are compared with the sharpest results presented in [Chu, 2007].

Table 5.14: *Comparative results for* ***Example 5****.*

			The developed algorithm		Chu's algorithm	
Designation	**Symbol**		**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	J^I		$[3.46, 3.59]$	1.2e-1	$[3.48, 3.62]$	1.3e-1
Bounds on the solution point	P^I	P_1^I	$[0.350, 0.391]$	4.1e-2	$[0.3500, 0.357]$	6.3e-3
		P_2^I	$[0.609, 0.651]$	4.1e-2	$[0.605, 0.650]$	4.5e-2
		P_3^I	$[-5.63, -5.21]$	4.1e-1	$[13.5, 14.8]$	1.2
		P_4^I	$[-5.64, -5.20]$	4.4e-1	$[-5.44, -5.24]$	1.9e-1
		P_5^I	N/A	N/A	$[-13.88, -13.59]$	2.8e-1
		P_6^I	N/A	N/A	$[8.34, 8.63]$	2.8e-1
CPU time	$\Delta t_{CPU}\,[s]$		134 (≈2m)	N/A	4580 (≈1h16m)	N/A
Number of remaining boxes	$N_{remaining}$		3893	N/A	6158	N/A

While the developed algorithm correctly bounds P^*, Chu's algorithm does not. Furthermore, the developed algorithm is approximately 34 times faster.

The following comments are in order:

1) Unlike in **Example 4**, here, the initial bounds on the two intervals representing time, P_1 and P_2, were successfully reduced. Most probably, due to the adequateness of (5-193)-(5-196).

2) Like in **Example 4**, no box P containing P^* can ever be proved feasible with respect to the inequality constraint. To understand why, the limit case is considered: $P = P^*$. By replacing P with P^* in (5-209) for $t = P_1^* = 0.375$, for example, one would obtain:

$$x_2\left(t = P_1^* = 0.375\right) = \left[\underline{X_2}, \overline{X_2}\right] = \left[0.124999, 0.125001\right] \qquad (5\text{-}216)$$

Since the number 0.125 cannot be exactly represented in a computer, outward rounding must be used. (5-216) is the sharpest non-thin interval containing 0.125. Since $\overline{X_2} > 0.125$, $P = P^*$ cannot be proved feasible with respect to the inequality constraint. Moreover, due to fundamental theorem of interval analysis (see (3-42)), no box containing P^* can ever be proved feasible with respect to the inequality constraint. This makes calculating sharp bounds on J^* and P^* a lot harder.

3) Using the same inputs as before (see **Table 5.13**) together with $N_{shrinking} = 100$ and $\varepsilon_{shrinking} = 0.1$, adding a shrinking phase results in an increase of the CPU time from approximately 134s to 408s, for the same desired accuracy. Even with four parameters, it is still faster not to use a shrinking phase.

4) Using the same inputs as before (see **Table 5.13**), using $N_{ineq} = 1000$ instead of (5-211) results in an increase of the CPU time from approximately 134s to 293s, for the same desired accuracy.

5) In **Table 5.15**, Δt_{CPU} is given as a function of N_{split}. All other inputs are kept constant (see **Table 5.13**).

As predicted by rule of thumb (5-41), splitting two parameters yields the fastest results. It is also interesting to see that P_1 and P_2 are the best parameters to split, as they both represent time. This was already seen in **Example 4**, where P_1, the only parameter representing time, must be split to solve the problem as fast as possible, as shown in **Table C.5**.

6) As said before, in most real life applications, a point p must be selected from P^I. This will now be done. Since $\overline{J} < \overline{\mathrm{J}}$, p is chosen as the midpoint of the box where the last upper bound on the minimum $\overline{J}$ was calculated, as explained in Section 5.2.5. This yields the following point:

$$p = \left[0.3766 \quad 0.6304 \quad -5.3269 \quad -5.3940\right]^T \qquad (5\text{-}217)$$

The control function and the states of the system produced by the approximate solution p and by the optimal solution point P^* can be plotted together by substituting P with p and P^* in (5-172), (5-180) and (5-186). This is done in **Figure 5.5**, **Figure 5.6** and **Figure 5.7**.

A first naked eye analysis of the figures shows an apparently good correspondence between the plots generated by p and P^*. A closer examination also shows that:

i) the control function $u\left(p,t\right)$ is continuous for $0 \le t \le 1$. This condition was used to write (5-172).

ii) $x_1(p,t)$ and $x_2(p,t)$ start and end exactly at the prescribed initial and final states. The initial and final states were used to write (5-180) and (5-186).

Table 5.15: *CPU time as a function of* N_{split} *in **Example 5**.*

Number of parameters being split	New boxes per old box N_{split}	CPU time $\Delta t_{CPU}\left[s\right]$
1	$\begin{bmatrix}2 & 0 & 0 & 0\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}0 & 2 & 0 & 0\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}0 & 0 & 2 & 0\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}0 & 0 & 0 & 2\end{bmatrix}^T$	No convergence after 500s
2	$\begin{bmatrix}2 & 2 & 0 & 0\end{bmatrix}^T$	85
	$\begin{bmatrix}2 & 0 & 2 & 0\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}2 & 0 & 0 & 2\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}0 & 2 & 2 & 0\end{bmatrix}^T$	134
	$\begin{bmatrix}0 & 2 & 0 & 2\end{bmatrix}^T$	418
	$\begin{bmatrix}0 & 0 & 2 & 2\end{bmatrix}^T$	No convergence after 500s
3	$\begin{bmatrix}2 & 2 & 2 & 0\end{bmatrix}^T$	198
	$\begin{bmatrix}2 & 2 & 0 & 2\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}2 & 0 & 2 & 2\end{bmatrix}^T$	No convergence after 500s
	$\begin{bmatrix}0 & 2 & 2 & 2\end{bmatrix}^T$	451
4	$\begin{bmatrix}2 & 2 & 2 & 2\end{bmatrix}^T$	No convergence after 500s

iii) $x_1(p,t)$ and $x_2(p,t)$ are continuous at $t = p_1$. This condition was used to write (5-180) and (5-186).

iv) $x_1(p,t)$ and $x_2(p,t)$ are discontinuous at $t = p_2$ since p does not exactly satisfy (5-185) and (5-192). As explained in Section 5.5.3, it is guaranteed that:

$$\left|x_1^{(2)}\left(p,t=p_2\right)-x_1^{(3)}\left(p,t=p_2\right)\right| \le mag\left(q_1\left(p\right)\right) \quad \textit{(5-218)}$$

$$\left|x_2^{(2)}\left(p,t=p_2\right)-x_2^{(3)}\left(p,t=p_2\right)\right| \le mag\left(q_2\left(p\right)\right) \quad \textit{(5-219)}$$

In this particular case:

$$mag\left(q_1\left(p\right)\right)=5.1e-8 \quad \textit{(5-220)}$$

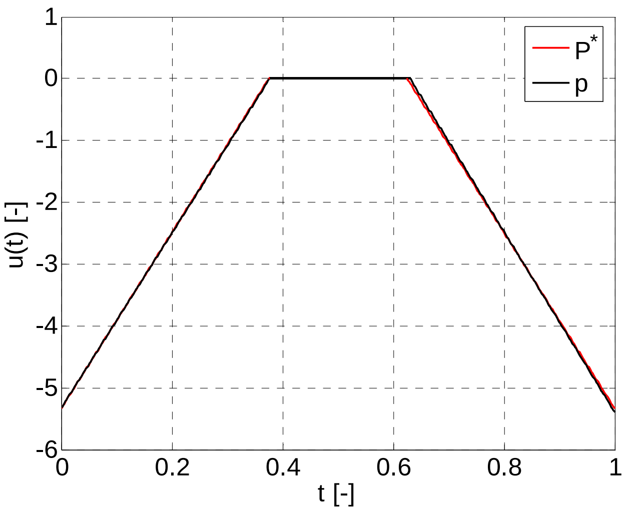

Figure 5.5: *Control function for* p *and* P^* *.*

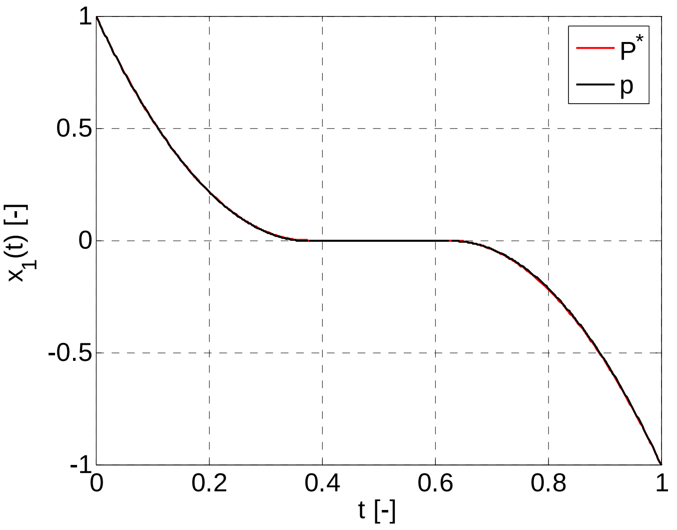

Figure 5.6: *First state for* p *and* P^* *.*

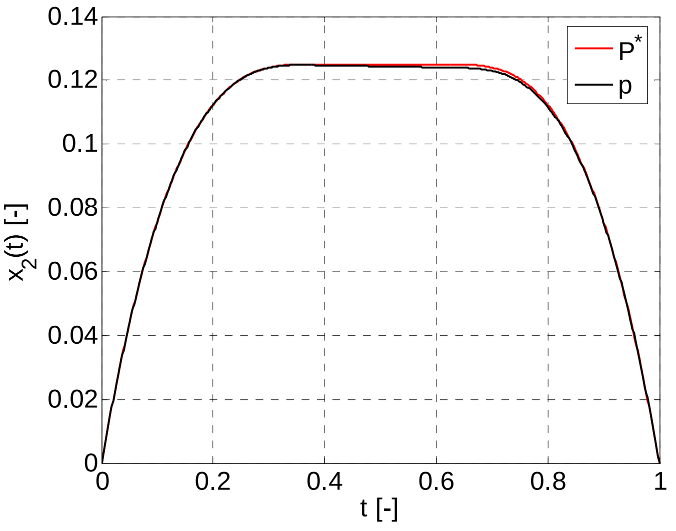

Figure 5.7: *Second state for* p *and* P^* *.*

$$mag\left(q_2\left(p\right)\right) = 2.4e-6 \tag{5-221}$$

Two new stopping conditions for the dynamic optimization algorithm can be suggested based on (5-218) and (5-219):

$$mag\left(q_1\left(p\right)\right) \leq \varepsilon_{eq_1} \tag{5-222}$$

$$mag\left(q_2\left(p\right)\right) \leq \varepsilon_{eq_2} \tag{5-223}$$

where the parameters ε_{eq_1} and ε_{eq_2} would be given by the user.

Now, it should be easier to understand point 5) of Section C.2.3.

v) $x_2\left(p,t\right)$ satisfies the inequality constraint (5-166).

Moreover, it is guaranteed that:

$$\left|J\left(p,t\right)-J^*\right| \leq \overline{J}-\underline{J} = w\left(J^I\right) = 1.2e-1 \leq \varepsilon_J \tag{5-224}$$

This is indeed true as:

$$\left|J\left(p,t\right)-J^*\right| = 1.8e-2 \tag{5-225}$$

Part II | Terminal Area Energy Management Trajectory Optimization

Chapter 6 | The TAEM Phase

In its descent flight, a Reusable Launch Vehicle (RLV) undergoes three phases:

1) The Atmospheric Re-entry Phase;

2) The Terminal Area Energy Management (TAEM) Phase;

3) The Landing Phase.

The initial condition of the Atmospheric Re-entry Phase is normally a low-Earth orbit. During this phase, the RLV decelerates to approximately Mach 2.5. The vehicle's maximum heat load and maximum heat rate are the main design drivers during this phase.

The TAEM phase is in charge of managing the vehicle's kinematic and potential energies and, at the same time, aligning the vehicle with the runway. The start of the TAEM phase is commonly referred to as the *Terminal Entry Point* (TEP) and its end as the *Auto-Landing Interface* (ALI).

At the start of the landing phase, around Mach 0.5, the total energy of the vehicle should already be consistent with a safe landing. During the landing phase, the vehicle transits from a steep glideslope to a shallow glideslope, before finally landing on the runway.

During this study, the focus will be on the TAEM phase. **Figure 6.1** shows how the TAEM phase fits between the re-entry and the landing phase and shows typical altitude (h) and Mach number (M) values at TEP and ALI found in literature, e.g., [Moore, 1991], [Mayanna *et al.*, 2006] and [Chartres *et al.*, 2005b], for different RLVs. These values are shown here to give the reader a feeling of the problem.

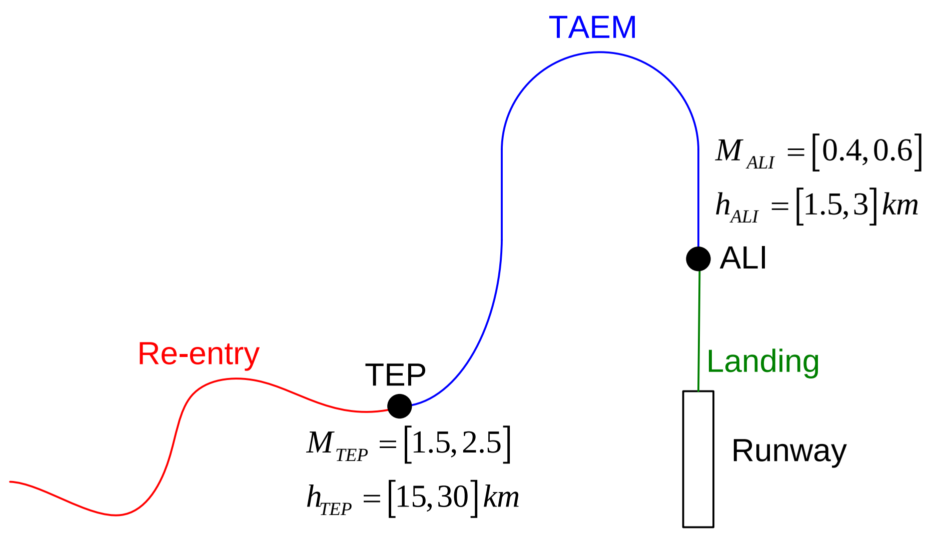

Figure 6.1: *The TAEM phase.*

During the TAEM phase, as the vehicle's velocity decreases from supersonic ($M > 1$) to subsonic ($M < 1$), the RLV undergoes the so-called *transonic flight regime* ($M \approx 1$). The transonic flight regime is characterised by a rapid change of the vehicle's aerodynamic characteristics. This, and the fact that RLVs are typically bad gliders, i.e., they typically have low lift-to-drag ratios (L/D), makes TAEM trajectory design a challenging task [Câmara, 2003].

In this chapter, a brief introduction to the TAEM trajectory design problem is given. First, the Space Shuttle's TAEM guidance strategy and the most recent advances in this field are described. Then, the equations of motion modelling the movement of a RLV during the TAEM phase are derived.

6.1 The Space Shuttle's TAEM Guidance

As far as the author knows, the only TAEM guidance strategy fully operational today is the one of NASA's Space Shuttle. The trajectory is composed by four different branches/phases [Moore, 1991], as depicted in **Figure 6.2.**

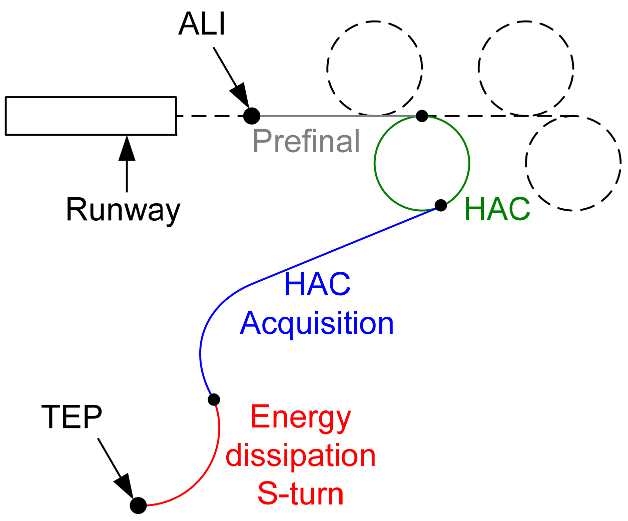

Figure 6.2: *Space Shuttle TAEM guidance phases.*

The Space Shuttle's TAEM guidance scheme is based on four Heading Alignment Cylinders (in earlier versions) or Cones (in more recent versions), also known as HACs. These HACs, which are illustrated in **Figure 6.2**, are classified as *overhead* or *straight-in* and as *nominal entry point* or *minimum energy point*. Depending on the vehicle's energy at TEP, the pilot can choose which HAC to fly.

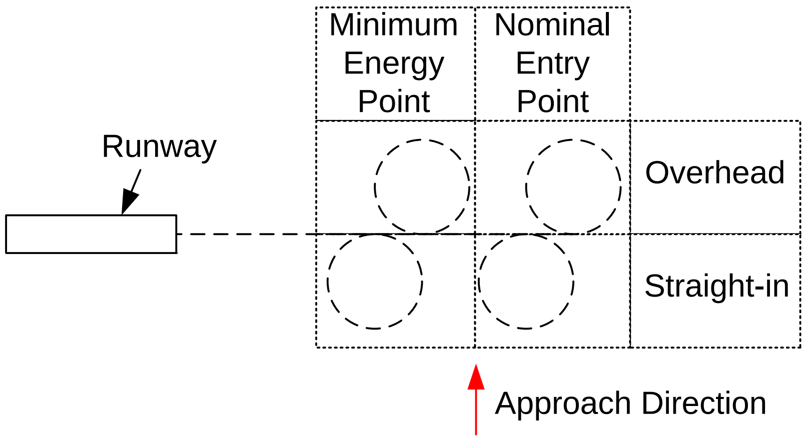

Figure 6.3: *Space Shuttle's HACs.*

The Space Shuttle's TAEM guidance phases are the following:

Phase 1 - Energy dissipation S-turn: if the vehicle's energy at TEP is too high, an initial S-turn is performed to dissipate the excess energy at maximum bank angle.

Phase 2 - HAC acquisition: the vehicle's heading angle, χ, is changed until its tangent to the HAC.

Phase 3 - HAC: the vehicle is aligned with the runway by following a cylinder or a cone. The vehicle describes a spiral trajectory. The radius of the cylinder/cone can be changed.

Phase 4 - Prefinal: straight flight until the auto-landing glide slope is intercepted.

The lateral and the longitudinal motions are treated separately by the Shuttle's TAEM guidance system (only the lateral motion has been mentioned so far). The longitudinal guidance consists basically on predefined altitude and dynamic pressure reference profiles. The altitude reference profile is composed by two linear segments and one cubic segment, while the dynamic pressure reference profile is composed by two limited linear segments, as depicted in **Figure 6.4**:

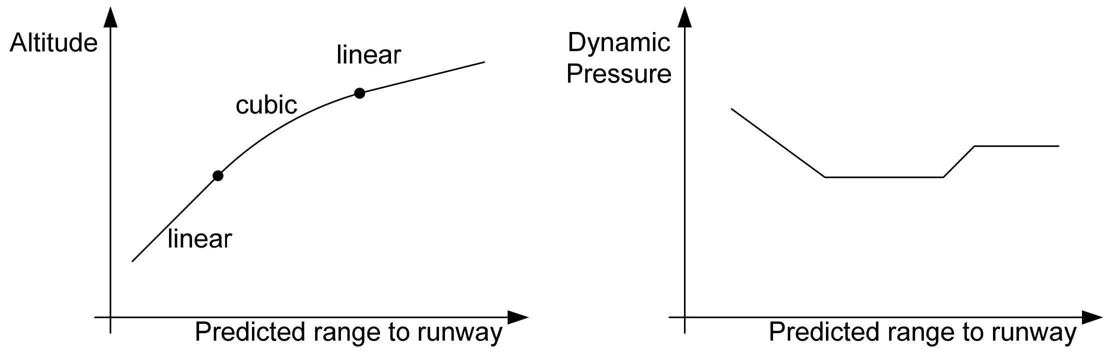

Figure 6.4: *Space Shuttle's TAEM guidance longitudinal reference profiles.*

The flight-path angle reference is obtained by differentiating the altitude reference profile.

Three loops are used by the Space Shuttle's TAEM path controller to follow the longitudinal and the lateral reference profiles:

1) An angle-of-attack controller;

2) A bank angle controller.

3) An energy/velocity/dynamic pressure controller (using the speedbrakes as actuator).

In the calculation of the nominal reference trajectories, the speedbrakes are assumed to be partially deployed during the subsonic flight regime. The speedbrakes deflection can be further increased or decreased in-flight to compensate for disturbances [Costa, 2003].

The reader is referred to [Moore, 1991] for more details about the Space Shuttle's TAEM guidance system.

6.2 State-Of-The-Art

Hereafter, a quick overview of the current status of TAEM trajectory design is presented.

Reference	[Costa, 2003] COSTA, R. R. da, "Studies for Terminal Area GNC of Reusable Launch Vehicles", *American Institute of Aeronautics and Astronautics*, Paper 2003-5438, 2003.

Overview

In [Costa, 2003], Rodrigo da Costa decouples the longitudinal from the lateral motion. The longitudinal reference profile is defined exactly as the mid-corridor between the maximum dive and the maximum glide trajectories. The lateral reference profile is calculated from the constrained optimization of $\frac{\partial \chi}{\partial h}$ parameterized as two cubic polynomials using the final distance to the target ALI and the final heading error as performance index.

This method does not allow the final velocity at ALI to be specified by the user.

Reference	[Mayanna *et al.*, 2006] MAYANNA, Anand, GRIMM, Werner, and WELL, Klaus H., "Adaptive Guidance for Terminal Area Energy Management (TAEM) of Reentry Vehicles", presented as paper AIAA 2006-6037 at the AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado, 21-24 August 2006.

Overview

In [Mayanna *et al.*, 2006], the longitudinal and the lateral motions are also decoupled. The longitudinal reference profile is chosen somewhat between the maximum dive and the maximum glide trajectories. However, unlike [Costa, 2003], the velocity and altitude boundary conditions at ALI are taken into account in the definition of the longitudinal reference profile.

The lateral reference profile is defined by three circular arcs and three straight lines disposed like in **Figure 6.5**.

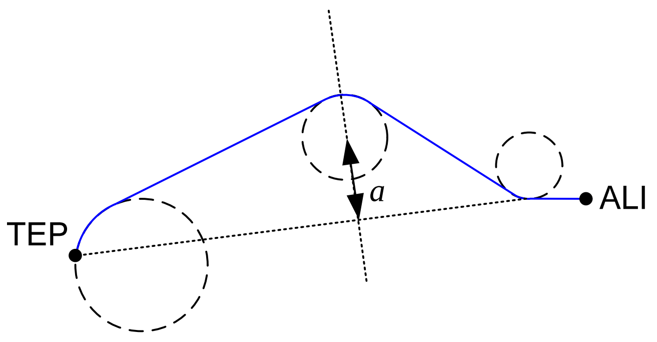

Figure 6.5: *Lateral reference profile suggested in [Mayanna et al., 2006].*

The radius and the position of the three circles are fixed except for the distance a, which is calculated in order to make the length of ground track equal to the downrange of the predefined vertical reference profile.

This algorithm is suitable for real-time on-board use.

Reference	[Chartres *et al.*, 2005b] CHARTRES, J. T. A., GRÄBLIN, M. H., and SCHNEIDER, G., "Optimisation of the Terminal Flight Phase for a Future Reusable Launch Vehicle", presented as paper AIAA 2005-6060 at the AIAA Guidance, Navigation, and Control Conference and Exhibit, San Francisco, 15-18 August 2005.

Overview

In [Chartres *et al.*, 2005b], an optimized reference trajectory is calculated through a Nonlinear Programming (NLP) optimiser.

The optimization problem is characterized by:

- Seven states: the six states from the complete three degrees of freedom equations of motion (longitudinal and lateral motions coupled) and the vehicle mass.
- Three inputs: angle of attack, bank angle and speedbrake setting.
- Several constraints:
    - Limited inputs.
    - Required initial and final states.
    - Limited lateral loads.
    - Limited heat flux and heat load.
    - Limited dynamic pressure.
- Performance index: final state error.

The drawbacks of using a NLP optimizer are:

1) An initial guess must be supplied.
2) The optimizer might converge to a local minimum.
3) As a consequence of 1) and 2), the final solution might be dependent on the initial guess.

[Chartres *et al.*, 2005b] makes an interesting observation: during the optimizations, the solutions seem to follow some characteristic shapes. **Figure 6.6** shows three such shapes taken from [Chartres *et al.*, 2005b].

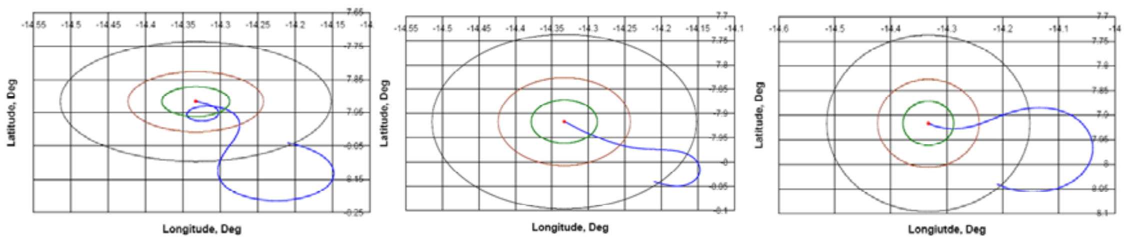

Figure 6.6: *Characteristic shapes taken from [Chartres et al., 2005b].*

The RLV performs only two turns: one left turn followed by a right turn or vice-versa.

Reference	[Chartres et al., 2005a] CHARTRES, J., GRÄBLIN, M., and SCHNEIDER, G., "A New Method For Terminal Area Guidance For Future Reusable Launch Vehicles", presented as paper IAC-05-C1.8.05 at the 56th International Astronautical Congress, 2005.

Overview

In [Chartres *et al.*, 2005a], the authors of [Chartres *et al.*, 2005b] suggest an adaptation to their NLP guidance algorithm to make it suitable for real-time on-board use.

An optimal reference trajectory is first calculated off-line and loaded into the RLV as an initial guess. During the TAEM flight, the on-board software updates the reference trajectory taking into account off-nominal conditions such as atmospheric disturbances, sensor errors and modelling errors. The new trajectory is kept as close as possible to the initial optimal solution.

Reference	[Barton, 2002] BARTON, G. H., "New Methodologies For Onboard Generation Of TAEM Trajectories For Autonomous RLVs", presented at the Core Technologies for Space Systems Conference, 2002.

Overview

In [Barton, 2002], Barton presents two real-time methodologies fully coupling the longitudinal and the lateral motions. The first one is restricted to the subsonic flight regime and is based on a fixed number of geometric shapes. The longitudinal and the lateral geometries are shown in **Figure 6.7**. The required angle of attack and bank angle to keep the vehicle on the reference trajectory are adjusted via an iterative process referred to as *balancing the dynamics*.

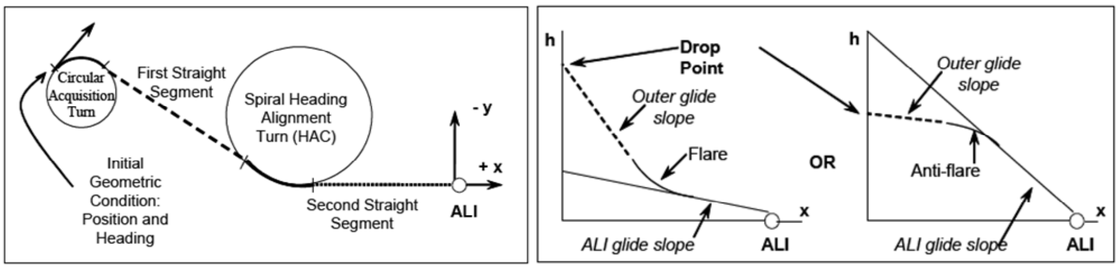

Figure 6.7: *Longitudinal and lateral geometries taken from [Barton, 2002].*

The second methodology does not rely on a predefined fixed geometry and is valid over the entire TAEM phase. First, a suboptimal dynamic pressure profile is selected satisfying both path (maximum dive and maximum glide) and final state constraints (final position and velocity). Then, the three-dimensional trajectory geometry is continuously adjusted from node to node via an iterative process (*balancing the dynamics*) to follow the specified dynamic pressure profile without violating the maximum loads supported by the vehicle. This methodology provides suboptimal solutions and is still in a development phase. The coupling between longitudinal and lateral motions is estimated.

Reference	[Burchett, 2004] BURCHETT, B. T., "Fuzzy Logic Trajectory Design and Guidance for Terminal Area Energy Management", Journal of Spacecraft and Rockets, Vol. 41, No. 3, May-June 2004.

Overview

In [Burchett, 2004], fuzzy logic methods are used to create real-time TAEM guidance systems which, in theory, can even cope with control surfaces failures. Guidance commands are generated for the bank angle and for the negative z -axis acceleration.

Fuzzy logic methods can choose from a continuum of possible trajectory parameter values. They also allow multiple constraints and objectives to be balanced into a single decision.

The lateral motion is based on a typical TACtical Air Navigation (TACAN) approach flown by military aircraft and shown in **Figure 6.8**.

All the turns are designed as circular arcs. The only adjustable parameters are XHAC and YHAC, i.e., the centre of the HAC.

The altitude reference profile is defined as a cubic polynomial with one arbitrary fixed coefficient and one adjustable coefficient. A linear velocity profile is used to predict the velocity at the beginning of the HAC.

Some experience is required to tune the final fuzzy logic system via trial-and-error. The longitudinal and lateral motions are considered completely decoupled.

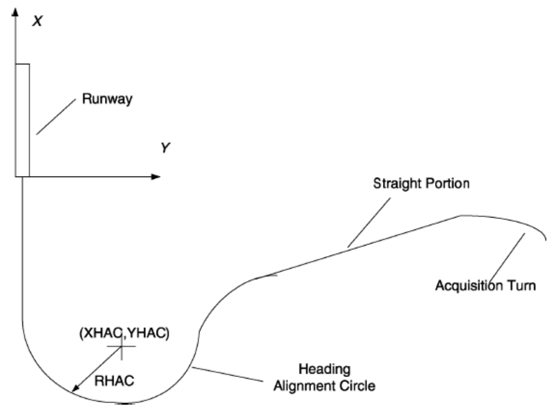

Figure 6.8: *Lateral geometry taken from [Burchett, 2004].*

Reference	[Vernis and Ferreira, 2006] VERNIS, P. and FERREIRA, E., "On-Board Trajectory Planner for the TAEM Guidance of a Winged-Body", EADS SPACE Transportation, 2006.

Overview

The method presented in [Vernis and Ferreira, 2006] is a real-time TAEM guidance system based on Buran's (the Russian space shuttle) reference trajectory geometry shown in **Figure 6.9**. The longitudinal and the lateral motions are considered fully coupled.

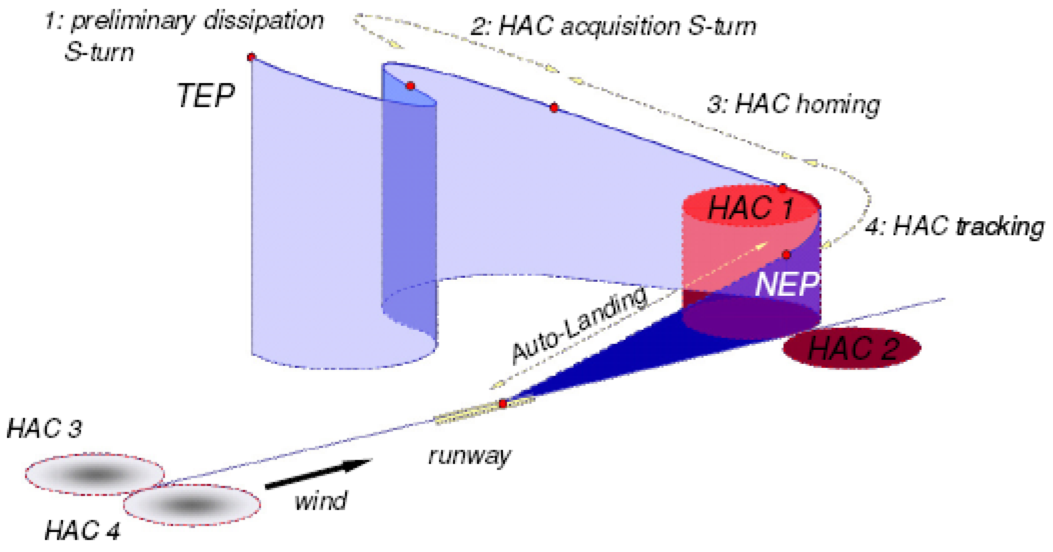

Figure 6.9: *Reference trajectory geometry of the Buran shuttle (taken from [Vernis and Ferreira, 2006]). Here, the acronym NEP (Nominal Entry Point) is used instead of ALI.*

The suggested TAEM planner performs the dynamic placement of a fixed length 3D curve between the TEP and the ALI points using only closed-form computations of some waypoints, i.e., without resorting to optimization techniques. The length of the curve is directly dependent on the energy level that needs to be dissipated.

Two different tools are described in [Vernis and Ferreira, 2006]: one that given the ALI point and the HAC parameters calculates the TAEM entry gate, and another one that given the TEP and ALI points calculates the HAC parameters.

The L/D ratio is assumed constant during each TAEM sub-phase.

6.3 The Equations of Motion

This section is dedicated to the derivation of the equations of motion modelling the movement of a RLV during the TAEM phase. These equations have a set of assumptions associated with them, which are enumerated in Section 6.3. The coordinate systems in which the equations of motion are based are described and illustrated in Section 6.3.2. The models used to characterize the Earth's atmosphere and the RLV's aerodynamic behaviour are specified in Sections 6.3.3 and 6.3.4, respectively. Ultimately, the equations of motion are mathematically derived in Section 6.3.5.

6.3.1 Assumptions

To reduce the computation load, simpler models are normally used during the so-called *Optimization Phase*. Afterwards, more sophisticated and accurate models are used to verify and validate the solutions in what is typically called the *Simulation Phase*.

The equations of motion used during the Simulation Phase take into account the following assumptions:

1) The vehicle is assumed to be a point mass.
2) The gravity acceleration is assumed to be constant.
3) The Earth is assumed to be flat, non-rotating and inertial.
4) All the turns are assumed to be coordinate turns, i.e., there are no aerodynamic side forces since the sideslip angle β is assumed to be zero.
5) The wind velocity is assumed to be negligible.
6) The mass of the vehicle is assumed to be constant.

6.3.2 Coordinate Systems

Before the equations of motion can be derived, four coordinate systems need to be defined: the runway reference frame, the local-horizontal local-vertical reference frame, the kinematic reference frame and the body-fixed reference frame.

6.3.2.1 Runway Reference Frame

The *Runway Reference Frame* $(ALIxyz)$ is illustrated in **Figure 6.10**:

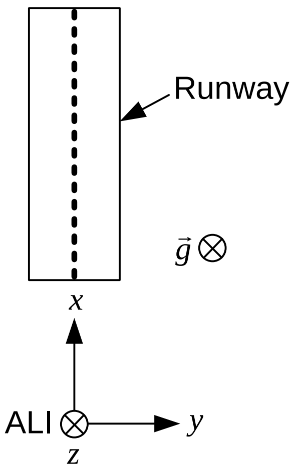

Figure 6.10: *Runway Reference Frame.*

where:

- ALI is the Auto-Landing Interface point;
- x is aligned with the runway's centreline;
- z is aligned with the gravity's acceleration vector;
- y completes the right-orthogonal reference frame.

The runway reference frame inherits the same properties assumed for planet Earth: flat, non-rotating and inertial.

6.3.2.2 Local-Horizontal Local-Vertical Reference Frame

The *Local-Horizontal Local-Vertical Reference Frame* $(Gx_h y_h z_h)$ has the same orientation has the runway reference frame but has its origin at the vehicle's centre of mass G, as illustrated in **Figure 6.11**:

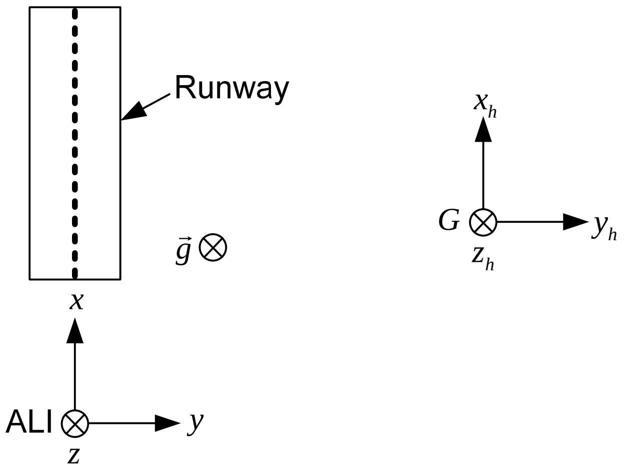

Figure 6.11: *Local-Horizontal Local-Vertical Reference Frame.*

6.3.2.3 Kinematic or Flight-Path Reference Frame

The equations of motion are written with the help of the so-called *Kinematic* or *Flight-Path Reference Frame* $(Gx_k y_k z_k)$ [Mulder *et al.*, 2007]. The flight-path reference frame is associated with the vehicle's *ground velocity* or *flight-path velocity* $\vec{V}_k$, i.e., with the vehicle's velocity relative to the ground. The relation between the flight-path reference frame and the local-horizontal local-vertical reference frame $(Gx_h y_h z_h)$ is illustrated in **Figure 6.12** where:

- x_k has the same direction as $\vec{V}_k$;
- z_k is in the vehicle's symmetry plane and points downwards;
- y_k completes the right-orthogonal reference frame;
- χ_r (or simply χ) is the vehicle's heading relative to the runway centreline;
- γ is the vehicle's flight-path angle;
- μ_k is the flight-path bank angle.

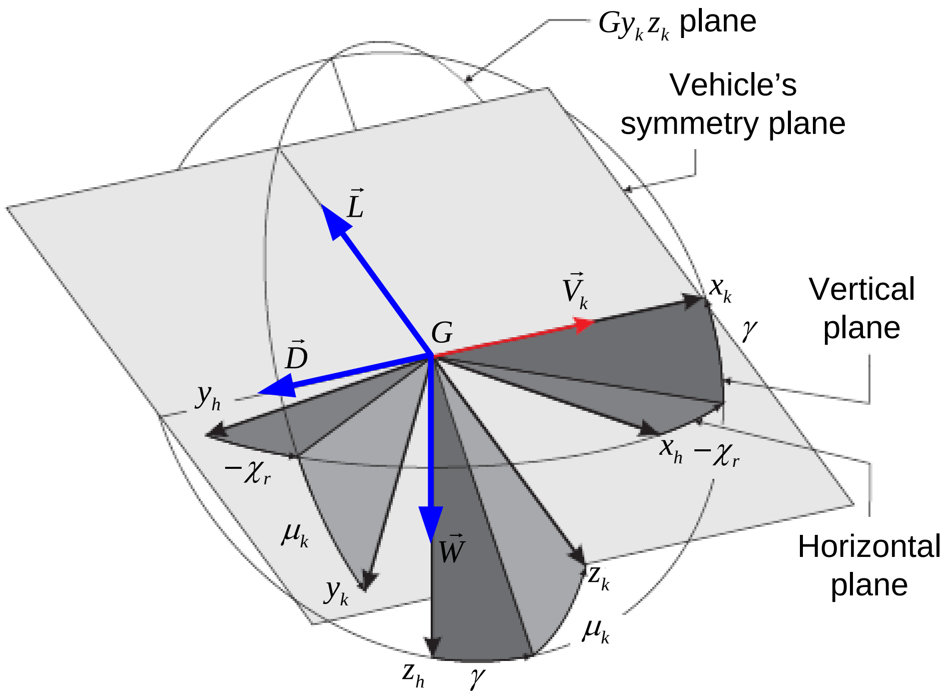

Figure 6.12: *Relation between the Flight-Path Reference Frame and the Local-Horizontal Local-Vertical Reference Frame (figure adapted from [Mulder et al., 2007]).*

6.3.2.4 Body-Fixed Reference Frame

The angle-of-attack (α) and the sideslip angle (β) are defined using the *Body-Fixed Reference Frame* $\left(Gx_b y_b z_b\right)$. The relation between the body-fixed reference frame and the flight-path reference frame $\left(Gx_k y_k z_k\right)$ assuming no wind is illustrated in **Figure 6.13** where:

- x_b is in the vehicle's symmetry plane and points forward;
- z_b is in the vehicle's symmetry plane and points downwards;
- y_b completes the right-orthogonal reference frame.

If there is no wind, the airspeed velocity vector $\vec{V}_a$ is equal to the flight-path velocity vector $\vec{V}_k$. $\vec{V}$ will be used to denote both them.

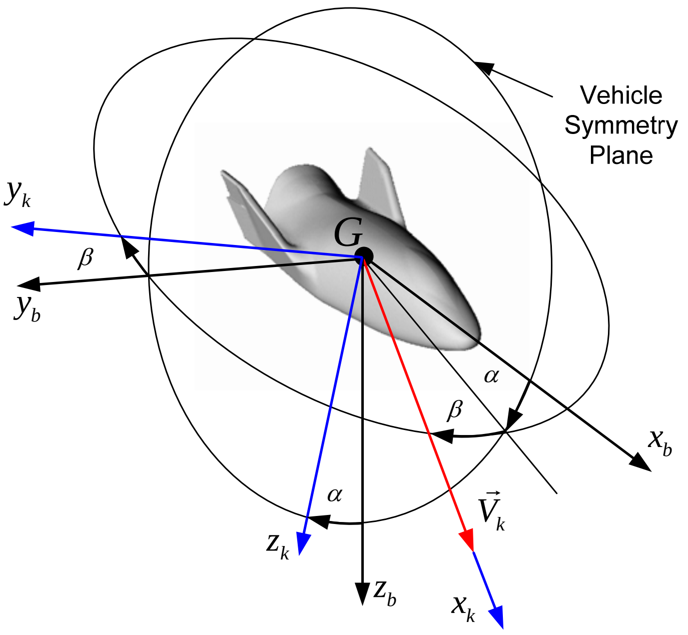

Figure 6.13: *Relation between the Body-Fixed Reference Frame and the Flight-Path Reference Frame.*

6.3.3 Atmospheric Model

To reduce the computational load, the simple and time-invariant *Exponential Atmospheric Model* is used to simulate the atmosphere. The exponential model assumes that the *air density* ρ decreases exponentially with increasing altitude [Vallado, 2007] as follows:

$$\rho\left(h\right)=\rho_0 \exp\left(-\frac{h-h_0}{H}\right) \qquad (6\text{-}1)$$

where ρ_0 is the air density at the reference altitude h_0 and H is the so-called *scale height*. These parameters are given in [Vallado, 2007] for different altitude ranges. The numerical values assumed for ρ_0, h_0 and H are shown in **Table 6.1**. These values are the ones given in [Vallado, 2007] for an altitude range going from 0 to 25 km. This altitude range covers almost completely the altitude range flown during the TAEM phase.

Table 6.1: *Parameters of the Exponential Atmospheric Model.*

Parameter	Symbol	Value
Reference altitude	h_0	0 km
Reference density	ρ_0	1.225 kg/m^3
Scale height	H	7.249

The speed of sound a is assumed to be constant and equal to 340.29 m/s, the value at mean sea level according to the *1976 Standard Atmosphere Model*. Since the speed of sound is strongly dependent on temperature, assuming a constant speed of sound is essentially equivalent to assuming an isothermic atmosphere.

6.3.4 Vehicle Model

In this study, the only aerodynamic forces considered are the *lift* (L) and the *drag* (D). The sideslip angle β is assumed to be zero at all times, i.e., the aerodynamic side forces (Y) along the y_k-axis are assumed to be zero.

Many authors (e.g., [Chu, 2007], [Câmara, 2003], [Vrijbergen, 2004], [Barton and Tragesser, 1999], [Mayanna *et al.*, 2006] and [Costa, 2003]) assume that the *aerodynamic lift* and *drag coefficients*, respectively:

$$C_L = \frac{L}{qS} \tag{6-2}$$

$$C_D = \frac{D}{qS} \tag{6-3}$$

where S is some *reference area*, typically the wing area, and q is the *dynamic pressure*, given by:

$$q = \frac{1}{2}\rho V^2 \tag{6-4}$$

are only dependent on the angle-of-attack α and on the *Mach number* M, given by:

$$M = \frac{V}{a} \tag{6-5}$$

The same assumption is made in this thesis.

In theory, the current TAEM tool is not limited to a specific RLV. The software should be able to calculate proper reference trajectories for any vehicle given their *mass* (m), reference area

(S), aerodynamic coefficients ($C_L(\alpha, M)$ and $C_D(\alpha, M)$) and minimum and maximum trimmable angles-of-attack: $\alpha_{\min}(M)$ and $\alpha_{\max}(M)$, respectively. Additionally, the tool is prepared to receive the following vehicle-dependent inputs:

- the vehicle's maximum supported dynamic pressure: $q_{\max}$;
- the vehicle's minimum supported load factor along the negative z-body axis: $n_{z,\min}$
- the vehicle's maximum supported load factor along the negative z-body axis: $n_{z,\max}$

In practice, the *HORUS-2B* (*Hypersonic Orbital Research and Utilization System*) illustrated in **Figure 6.14** was the only spacecraft used during the tool's prototyping.

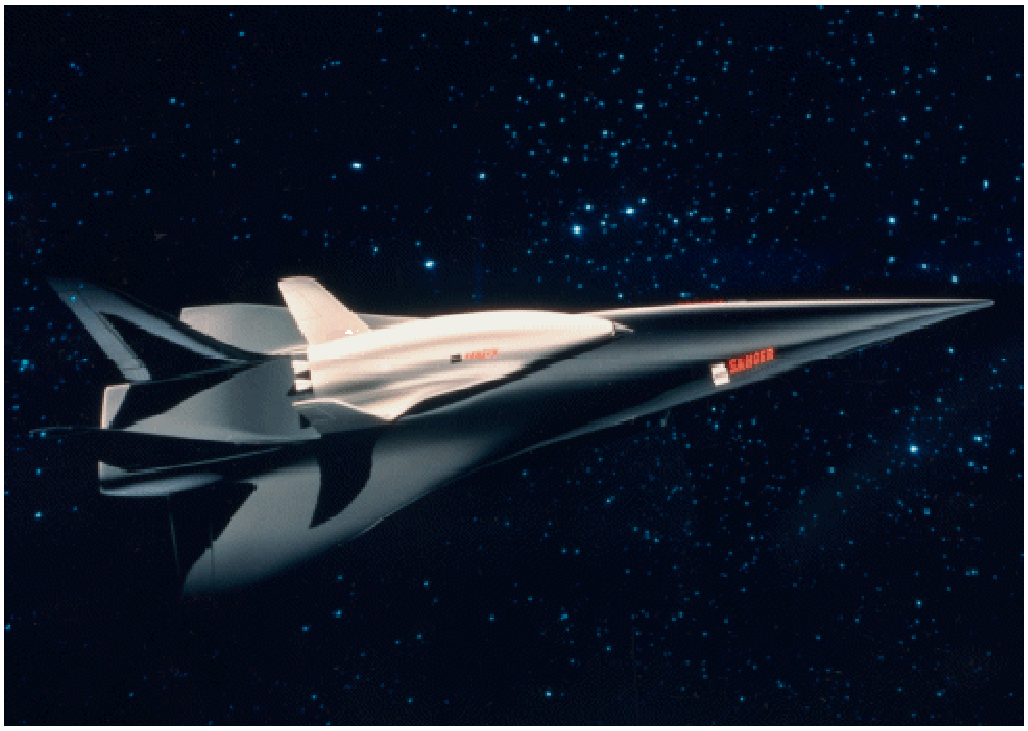

Figure 6.14: *The HORUS spacecraft shown on top of the Sänger (taken from [Space Services Koellen, 2008]).*

The HORUS-2B was originally developed by the German company Messerschmitt-Bölkow-Blohm (MBB) as a fully reusable second stage for the Ariane-5 launcher [Mooij, 1998]. The initial design was essentially unpowered, apart from a deorbitation engine and some attitude-control thrusters. Later on, a rocket engine was added and HORUS became the second stage of *Sänger*, the German TSTO (Two-Stage-To-Orbit) flag vehicle, also represented in **Figure 6.14**. The original winged, unpowered re-entry vehicle design is used in this study.

The HORUS control surfaces are schematically represented in **Figure 6.15**. The rudders can only move outwardly, which means that only one rudder is moved at a time during yaw manoeuvres. Deflecting both rudders at the same time results in the so-called *speedbrake* function.

The aerodynamic database of the HORUS-2B was gently provided by Dr. Erwin Mooij of the Delft University of Technology, Faculty of Aerospace Engineering. It is based on figures shown in [MBB, 1988]. The numerical values of the supersonic and hypersonic regimes are given in [Mooij, 1995].

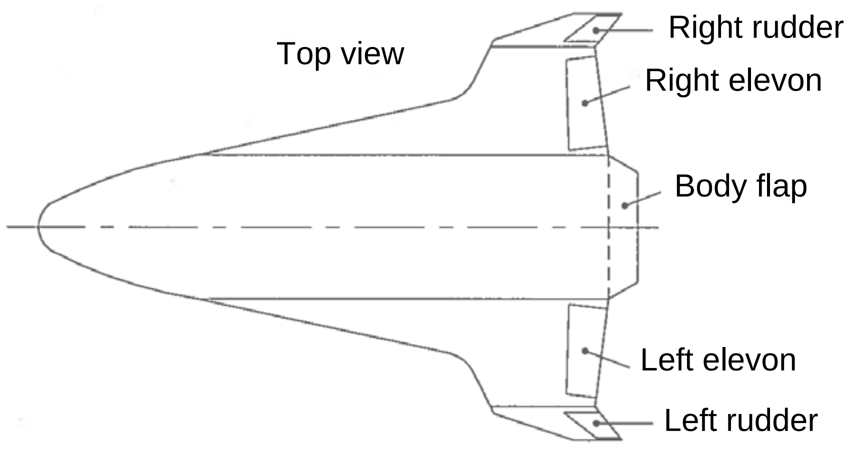

Figure 6.15: *Control surfaces of the HORUS-2B (taken from [Mooij, 1998]).*

In the aerodynamic database provided by Dr. Mooij, the trimmed supersonic, transonic and subsonic aerodynamic lift and drag coefficients are given in the format of lookup tables as a function of α and M. Also given are:

- the subsonic aerodynamic drag coefficient increase due to the deflection of the left rudder as a function of its deflection angle $\delta_{r,l}$ and M;
- the aerodynamic drag coefficient decrement due to altitude as a function of h and M.

These last two effects were not considered as they were deemed neglectable.

In this study, the speedbrake function, which corresponds to deflecting both rudders at the same time by δ_r degrees, is not considered, in order to simplify the trajectory generation problem. Ideally, δ_r should also be treated as a time-variant control function and its optimal time-history should also be calculated by the optimization process.

In **Figure 6.16**, the lift-over-drag ratio $L/D = C_L/C_D$ of the HORUS-2B RLV is plotted as a function of α and M. Linear interpolation will recurrently be used to fill up the gaps.

Figure 6.16 shows why one of the usual requirements imposed on the TAEM phase is a fast sonic transition [Costa, 2003]:

- Like most RLVs, the HORUS-2B has worse gliding capabilities, i.e., a smaller L/D, during the supersonic regime.
- During the transonic regime, it is harder to trim the vehicle, as can be seen from the small α corridor around $M = 1$.

Another component of the HORUS-2B aerodynamic database is the minimum and maximum trimmable α as a function of M. These bounds are illustrated in **Figure 6.17**.

The re-entry mass m and wing area S of the HORUS-2B are given in **Table 6.2** [Mooij, 1998].

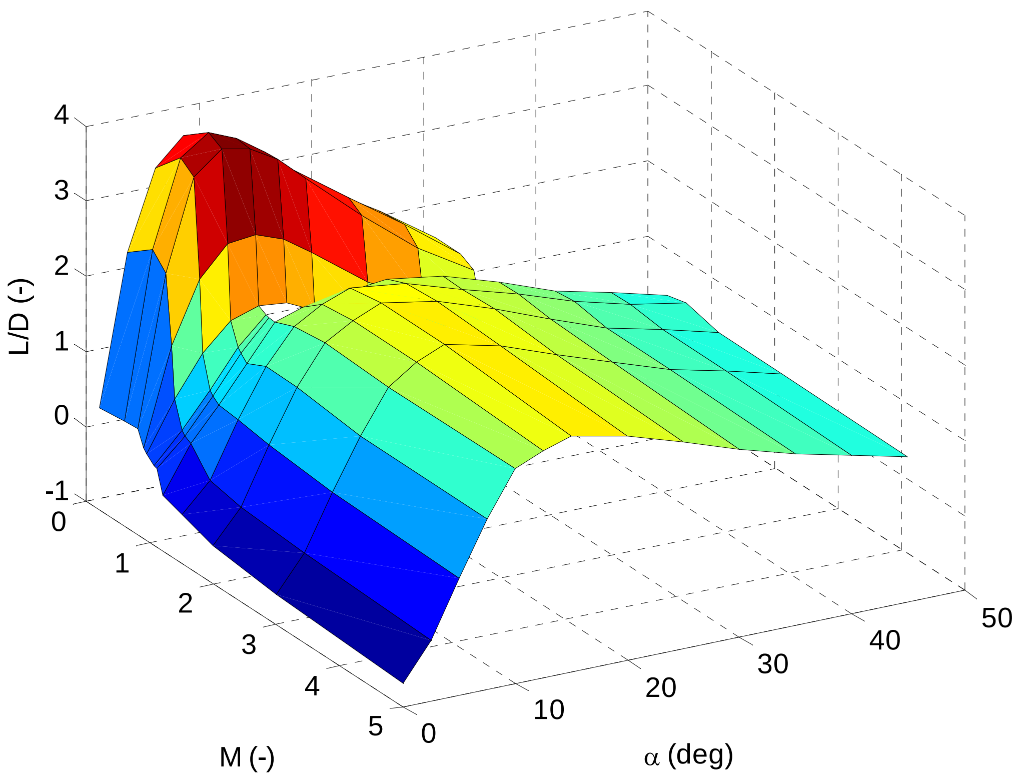

Figure 6.16: *Lift-over-drag ratio of the HORUS-2B.*

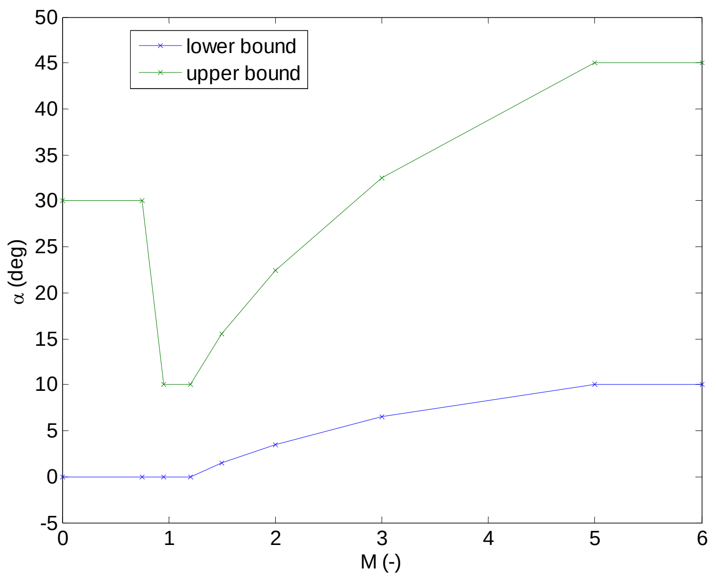

Figure 6.17: *Minimum and maximum allowed angle-of-attack.*

Table 6.2: *HORUS-2B main characteristics.*

Parameter	Symbol	Value
Re-entry mass	m	26,029 kg
Wing area	S	110 m^2

Due to the unavailability of more representative values, the path constraints assumed for the HORUS are the ones specified in [Chartres *et al.*, 2005b] for the *Hopper* RLV. These values are given in **Table 6.3**.

Table 6.3: *Path constraints assumed for the HORUS-2B.*

Parameter	Symbol	Value
Maximum dynamic pressure	$q_{\max}$	40 kPa
Minimum load factor along the negative z-body axis	$n_{z,\min}$	-0.75 g
Maximum load factor along the negative z-body axis	$n_{z,\max}$	2.5 g

6.3.5 Derivation of the Equations of Motion

All the blocks necessary to derive the equations of motion have now been described: the assumptions, the coordinate systems, the atmospheric model and the vehicle model.

Under the assumptions presented in Section 6.3.1, the only external forces acting on the vehicle are the aerodynamic force and the gravitational force $\vec{W}$. The aerodynamic force can be decoupled into two components: one with the same direction as $\vec{V}_a$, the drag force $\vec{D}$, and another one perpendicular to $\vec{V}_a$, the lift force $\vec{L}$. These forces are represented in **Figure 6.18**.

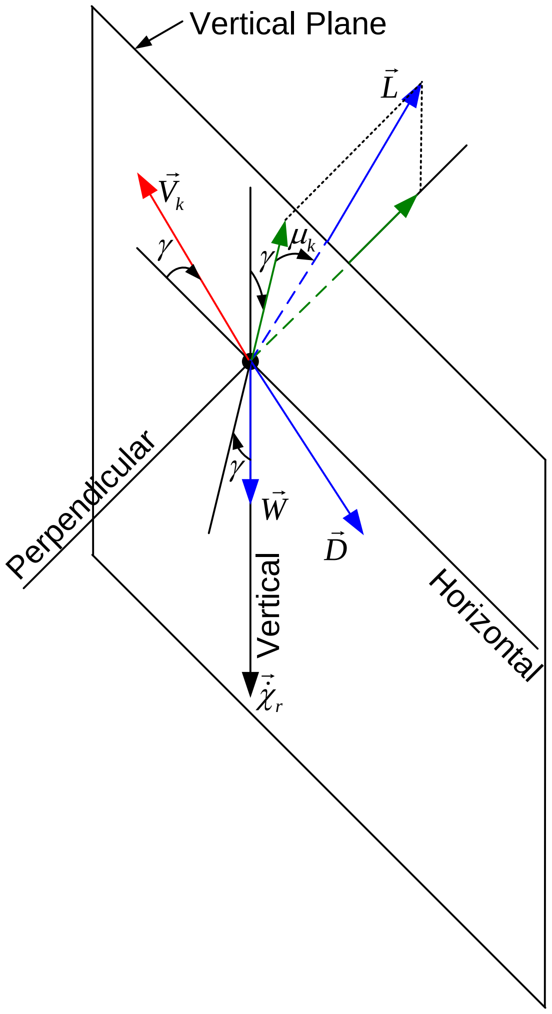

Figure 6.18: *External forces acting on the vehicle.*

Applying Newton's second law:

$$\vec{F} = m\vec{a} \tag{6-6}$$

where $\vec{F}$ is the vector sum of all external forces and $\vec{a}$ is the acceleration of the centre of mass G with respect to the inertial reference frame (the runway reference frame in this case) to the configuration shown in **Figure 6.18** results in three equations describing the *dynamics* of the system:

$$\dot{V} = -\frac{D}{m} - g\sin\gamma \tag{6-7}$$

$$\dot{\gamma} = \frac{1}{V}\left(\frac{L}{m}\cos\mu_k - g\cos\gamma\right) \tag{6-8}$$

$$\dot{\chi}_r = \frac{L\sin\mu_k}{mV\cos\gamma} \tag{6-9}$$

In addition, three *kinematic equations* rise from the coordinate transformation of the velocity vector $\vec{V}$ from the flight-path reference frame to the runway reference frame:

$$\dot{x} = V\cos\gamma\cos\chi_r \tag{6-10}$$

$$\dot{y} = V\cos\gamma\sin\chi_r \tag{6-11}$$

$$\dot{h} = -\dot{z} = V\sin\gamma \tag{6-12}$$

The equations of motion (6-7)-(6-12) can be rewritten in a more useful form by:

1) Substituting L and D by $C_L(\alpha,M)$ and $C_D(\alpha,M)$ via (6-2) and (6-3).

2) Substituting the velocity V by the dynamic pressure q via (6-4). This change is defended in [Barton and Tragesser, 1999], [Costa, 2003] and [Barton, 2002] due to three reasons:

   - The equations of motion become better behaved since dynamic pressure is a more slowly varying parameter than velocity.

   - The dynamic pressure path constraint can be directly imposed on the optimization problem.

   - Controllers based on dynamic pressure are more extensively available in literature than flight path velocity controllers [Costa, 2003].

3) Substituting time by altitude as independent variable via (6-12). This change is defended in [Barton and Tragesser, 1999], [Costa, 2003] and [Barton, 2002] due to two reasons:

   - It is easier to define the integration stopping condition in terms of altitude than in terms of time since it is the altitude and not the duration of the TAEM phase that is limited.

   - The time history of the trajectory is of little concern.

Applying these changes to (6-7)-(6-12) yields the following equations of motion:

$$\frac{dq}{dh}=\left(\frac{d\rho}{dh}\frac{1}{\rho}-\frac{\rho SC_D}{m\sin\gamma}\right)q-g\rho \tag{6-13}$$

$$\frac{d\gamma}{dh}=\frac{\rho}{2\sin\gamma}\left(\frac{C_LS\cos\mu_k}{m}-\frac{g\cos\gamma}{q}\right) \tag{6-14}$$

$$\frac{d\chi_r}{dh}=\frac{\rho SC_L\sin\mu_k}{m\sin(2\gamma)} \tag{6-15}$$

$$\frac{dx}{dh}=\frac{\cos\chi_r}{\tan\gamma} \tag{6-16}$$

$$\frac{dy}{dh}=\frac{\sin\chi_r}{\tan\gamma} \tag{6-17}$$

$$\frac{dt}{dh}=\frac{1}{\sqrt{\frac{2q}{\rho}}\sin\gamma} \tag{6-18}$$

Using altitude as independent variable also has its own problems: a singularity at $\gamma=0$ appears which did not exist in the time formulation. To have the best of both worlds, a new independent variable is suggested in [Chartres *et al.*, 2005b] and [Kluever, 2007]: energy. Since the true velocity and altitude at TEP and the desired altitude and velocity at ALI are known, it is straightforward to define the starting and stopping conditions in terms of energy, which is a monotonically decreasing function of time for an unpowered glider.

In this thesis, the normalized energy E is defined as the sum of the vehicle's potential energy with the vehicle's kinematic energy normalized by the vehicle's mass, i.e.:

$$E=gh+\frac{1}{2}V^2 \tag{6-19}$$

E has the units of J/kg. Differentiating (6-19) with respect to time yields:

$$\frac{dE}{dt}=-\frac{VD}{m} \tag{6-20}$$

Multiplying (6-7)-(6-12) with the inverse of (6-20) and rearranging the equations to reduce the dependency effect yields the final 3 degrees of freedom (DOF) equations of motion describing the vehicle's movement during the TAEM phase and used during the optimization process:

$$\frac{dq}{dE}=\frac{m\sin\gamma}{C_DS}\left(-\frac{d\rho}{dh}\frac{1}{\rho}+\frac{\rho g}{q}\right)+\rho \tag{6-21}$$

$$\frac{d\gamma}{dE}=\frac{\rho}{2q^2C_DS}\left(-C_LqS\cos\mu_k+mg\cos\gamma\right) \tag{6-22}$$

$$\frac{d\chi_r}{dE}=-\frac{\rho C_L\sin\mu_k}{2C_Dq\cos\gamma} \tag{6-23}$$

$$\frac{dx}{dE} = -\frac{m\cos\gamma\cos\chi_r}{C_D qS} \tag{6-24}$$

$$\frac{dy}{dE} = -\frac{m\cos\gamma\sin\chi_r}{C_D qS} \tag{6-25}$$

$$\frac{dh}{dE} = -\frac{m\sin\gamma}{C_D qS} \tag{6-26}$$

where $\rho = \rho\left(h\left(E\right)\right)$, $\frac{d\rho}{dh} = -\frac{\rho\left(h\left(E\right)\right)}{H}$, $C_L = C_L\left(\alpha\left(E\right), M\left(E\right)\right)$ and $C_D = C_D\left(\alpha\left(E\right), M\left(E\right)\right)$.

If of interest, the mission elapsed time can be retrieved from (6-20) through:

$$\frac{dt}{dE} = -\frac{m}{VD} = -\frac{m}{\sqrt{\frac{2q^3}{\rho}}C_D S} \tag{6-27}$$

Finally, the load factor along the negative z-body axis, n_z, can be calculated as follows [Kluever, 2007]:

$$n_z = \frac{L\cos\alpha + D\sin\alpha}{mg} = \frac{C_L qS\cos\alpha + C_D qS\sin\alpha}{mg} \tag{6-28}$$

From (6-21)-(6-27), it is easy to see that the states of the system are:

$$x = \begin{bmatrix} q & \gamma & \chi_r & x & y & h & t \end{bmatrix}^T \tag{6-29}$$

where the last state, t, is optional. The system has two inputs:

$$u = \begin{bmatrix} \alpha & \mu_k \end{bmatrix} \tag{6-30}$$

Chapter 7 | TAEM Trajectory Optimization using Interval Analysis

During the course of this thesis, two different TAEM trajectory design methodologies were developed and tested. The first one assumes no coupling between the longitudinal and lateral motions and it is in a much higher level of operational readiness. The second one is more general in the sense that it works with the fully coupled 3-DOF equations of motion, but it is still in a preliminary stage. Both methodologies are explained in this chapter.

All the results presented in this part (Part II) were obtained using a computer equipped with Intel Core 2 Duo processors running at 2x2.5GHz and with 4GB of RAM. The OS running on the machine was Microsoft Vista Ultimate 32-bit with Service Pack 1 installed. Version 7.6.0.324 (R2008a) of MATLAB and 5.5 of INTLAB were used.

7.1 1-DOF Methodology

In the 1-DOF TAEM trajectory design methodology, the longitudinal motion is assumed to be decoupled from the lateral motion. The basic principle consists on first fixing the longitudinal profile, namely, the q, γ, C_L, C_D, ρ and E time histories[10], and then use them in the optimization of the lateral profile, which is done with interval analysis. The same basic principle is used in [Costa, 2003].

Two different tools were developed using the 1-DOF methodology: one that determines the Terminal Entry Gate (TEG) and another one that designs optimal[11] TAEM trajectories.

The Terminal Entry Gate is defined as the set of all feasible TEP states, i.e., the set of all TEP states from which the suggested guidance scheme is able to steer the RLV safely until an acceptable ALI state.

7.1.1 Terminal Entry Gate Determination

In **Figure 7.1**, a flowchart representing the main blocks of the TEG determination algorithm is given. The blocks are described in the forthcoming sections.

7.1.1.1 Input File

Table 7.1 summarizes all the main inputs of the TEG determination tool and the values used during the subsequent simulations. The exact structure of the input file and an example are given in Appendix B.

[10] *Even though the term* time history *is used here and in other places along the text, the reader should keep in mind that the independent variable is in fact normalized energy and not time, as explained in Section 6.3.5.*

[11] *The TAEM trajectories are optimal in the sense that the final state error is minimized. This subject is readdressed in Section 7.1.2.*

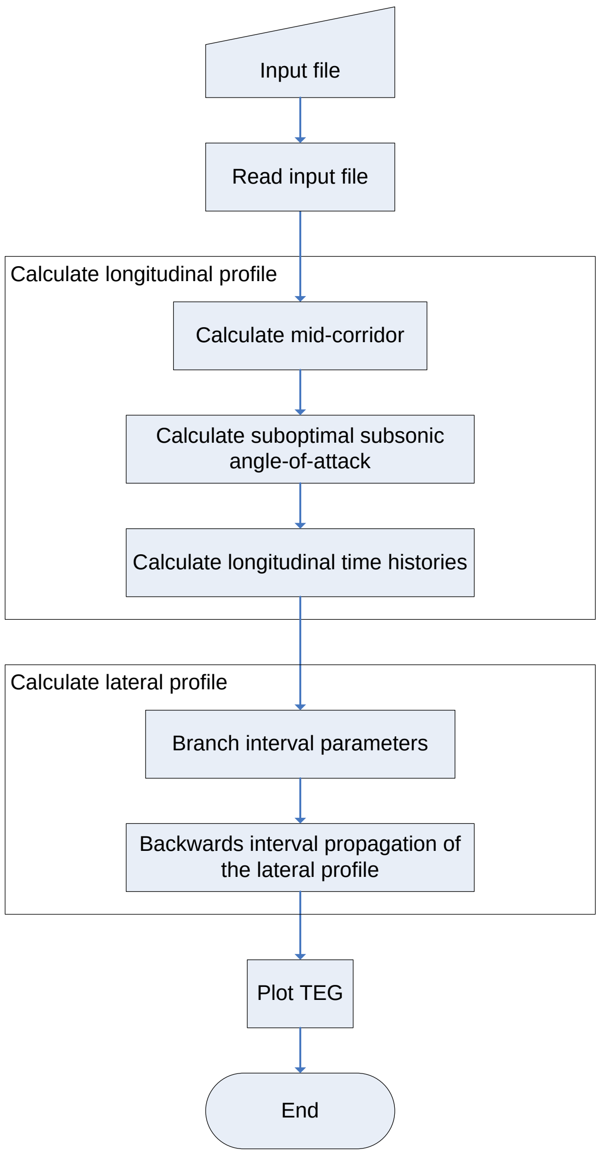
Input file
Read input file
Calculate longitudinal profile
Calculate mid-corridor
Calculate suboptimal subsonic angle-of-attack
Calculate longitudinal time histories
Calculate lateral profile
Branch interval parameters
Backwards interval propagation of the lateral profile
Plot TEG
End

Figure 7.1: *TEG Determination Algorithm Flowchart.*

Table 7.1*: Main inputs of the TEG Determination Tool.*

Input	Symbol	Format	Value	Units
Acceptable relative heading angle (interval) at ALI	$\chi_{r,ALI}$	*interval*	$[-3,3]$	deg
Acceptable x-position (interval) at ALI	x_{ALI}	*interval*	$[-1000,1000]$	m
Acceptable y-position (interval) at ALI	y_{ALI}	*interval*	$[-1000,1000]$	m
Flight-path bank angle (interval) during sub-phase 1	μ_1	*interval*	$[-60,60]$	deg
Flight-path bank angle (interval) during sub-phase 3	μ_2	*interval*	$[-60,60]$	deg
Number of subintervals in which μ_1 is split	N_{μ_1}	*int*	12	-
Number of subintervals in which μ_2 is split	N_{μ_2}	*int*	140	-

Input	Symbol	Format	Value	Units
Reference area of the vehicle	S	*float*	110	m^2
Re-entry mass of the vehicle	m	*float*	26,029	kg
Aerodynamic lift coefficient	$C_L(\alpha, M)$	*float*	N/A	-
Aerodynamic drag coefficient	$C_D(\alpha, M)$	*float*	N/A	-
Minimum and maximum trimmable α as a function of M	$\alpha_{\min}(M)$ and $\alpha_{\max}(M)$	*float*	N/A	deg
Mach number at TEP	M_{TEP}	*float*	1.5	-
Mach number at ALI	M_{ALI}	*float*	0.5	-
Altitude at TEP	h_{TEP}	*float*	15,000	m
Altitude at ALI	h_{ALI}	*float*	3,000	m
Number of integration steps	N_{int}	*int*	3,000	-
Flight-path angle at TEP	γ_{TEP}	*float*	-10	deg
Minimum α during subsonic regime	$\alpha_{sub,\min}$	*float*	0	deg
Maximum α during subsonic regime	$\alpha_{sub,\max}$	*float*	30	deg
Subsonic α search step	$\Delta\alpha$	*float*	1	deg
Maximum subsonic M	$M_{subsonic}$	*float*	0.75	-
Maximum dynamic pressure supported by the vehicle	$q_{\max}$	*float*	40,000	Pa
Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\min}$	*float*	-0.75	-
Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\max}$	*float*	2.5	-
Sub-phases relative length	$\Delta E_{rel,sub-phases}$	*float*	$[0.44 \quad 0.05 \quad 0.5 \quad 0.01]$	- (%)

Most of the variables appearing for the first time in **Table 7.1** are related with the control function parameterization. The next section is dedicated to this subject.

7.1.1.2 Control Function Parameterization

As discussed in Section 6.3.5, the control function, u, of the 3-DOF equations of motion (6-21)-(6-27) is:

$$u = \begin{bmatrix} \alpha & \mu_k \end{bmatrix}^T \tag{7-1}$$

α and μ_k are independently parameterized. This can be done since the longitudinal and lateral motions are assumed to be decoupled.

7.1.1.2.1 Angle-Of-Attack Parameterization

Several authors, e.g., [Costa, 2003] and [Mayanna *et al.*, 2006], define the longitudinal profile of the TAEM phase as some type of average between the *maximum dive* and *maximum glide* profiles.

[Costa, 2003] and [Mayanna *et al.*, 2006] define the *maximum dive* profile as the trajectory producing the shortest covered distance while respecting the vehicle's constraints, namely, the maximum supported dynamic pressure. On the other hand, the *maximum glide* profile is defined as the trajectory producing the longest covered distance, which corresponds to flying straight ($\mu = 0$) at an angle-of-attack yielding maximum L/D throughout the TAEM phase.

The SIMULINK model represented in **Figure 7.2** was used to simulate the maximum glide and the maximum dive profiles of the HORUS-2B under the conditions established in **Table 7.1**. MATLAB's *ode45* solver, set with a relative tolerance of 1e-6, was used to propagate the vehicle's trajectory.

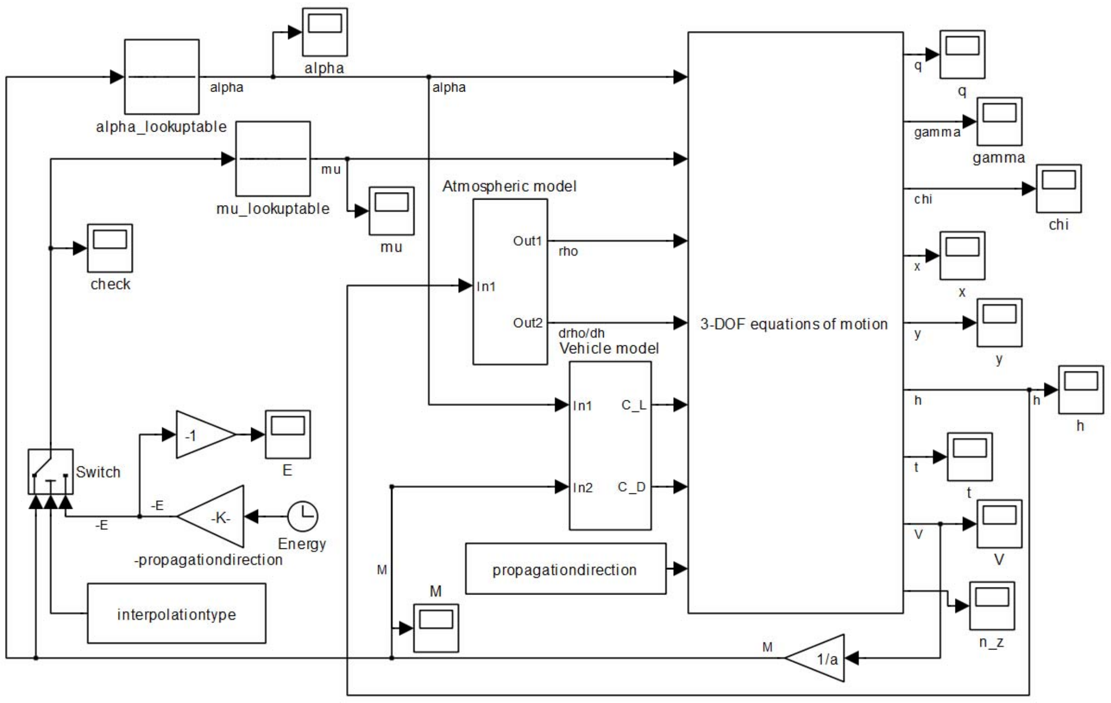

Figure 7.2: *3-DOF SIMULINK model.*

In this thesis, the maximum dive and maximum glide angle-of-attack profiles are computed as the angles-of-attack yielding the minimum and maximum $C_L(\alpha,M) / C_D(\alpha,M)$ for each Mach number, respectively. Linear interpolation is used to fill up the gaps between different Mach numbers. The bank angle μ_k is kept equal to zero.

The seven states of the system and two output variables, M and n_z, associated with the maximum dive and maximum glide profiles, are plotted in **Figure 7.3**. In **Figure 7.4**, the corresponding angle-of-attack profiles as a function of M are given.

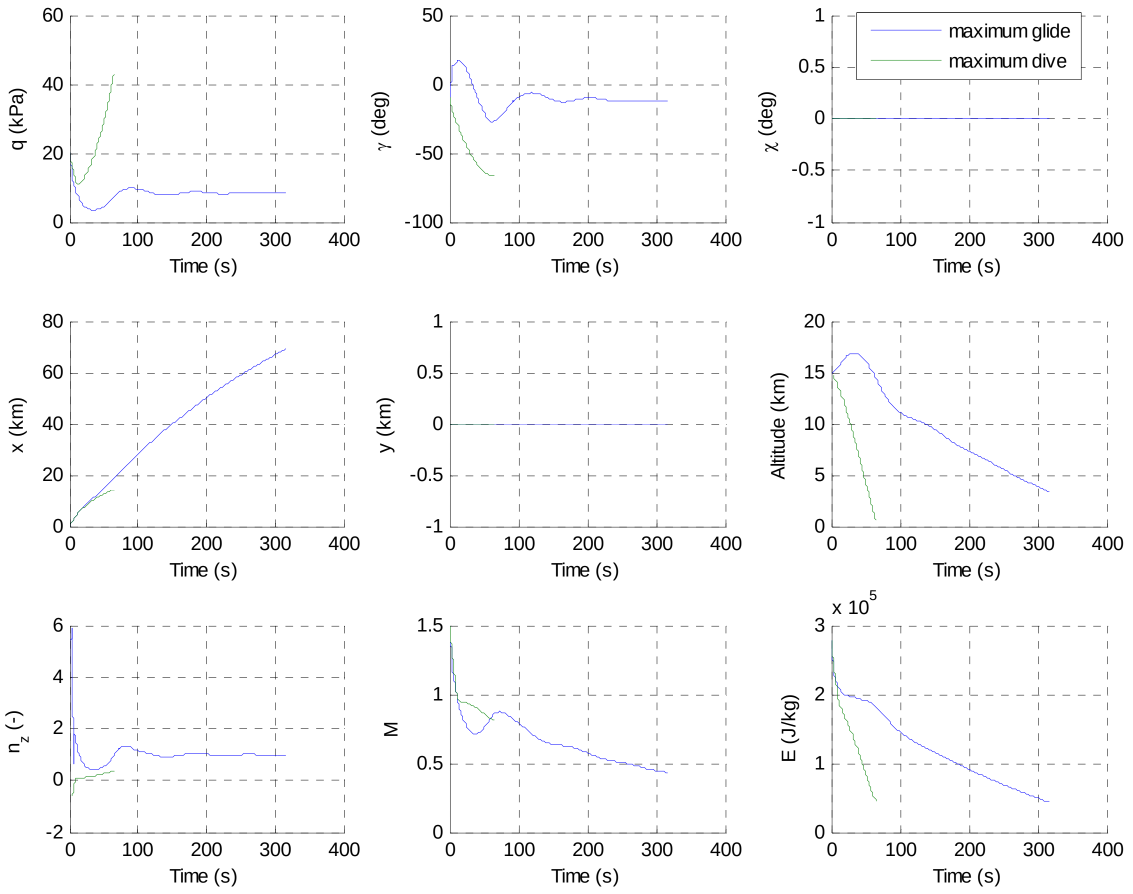

Figure 7.3: *Maximum glide and maximum dive profiles.*

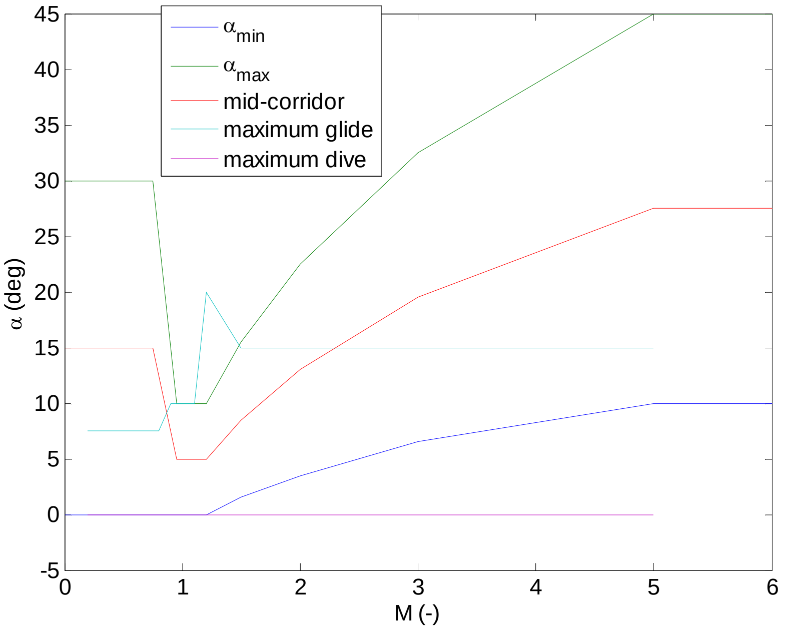

Figure 7.4: *Maximum glide and maximum dive angle-of-attack profiles.*

As the reader can see, the maximum dive profile violates the $q_{\max}$ constraint, while the maximum glide profile violates the $n_{z,\max}$ constraint, due to a sudden initial pull-up manoeuvre. Both of them violate the trimmable angle-of-attack boundaries defined by $\alpha_{\min}$ and $\alpha_{\max}$.

Upon these results and to maximize safety, the longitudinal motion was defined based on the mid-corridor angle-of-attack profile shown in **Figure 7.4** and not as some type of average between the maximum dive and maximum glide profiles, as in [Costa, 2003] and [Mayanna *et al.*, 2006]. The states and outputs associated with the selected solution are given in **Figure 7.5**.

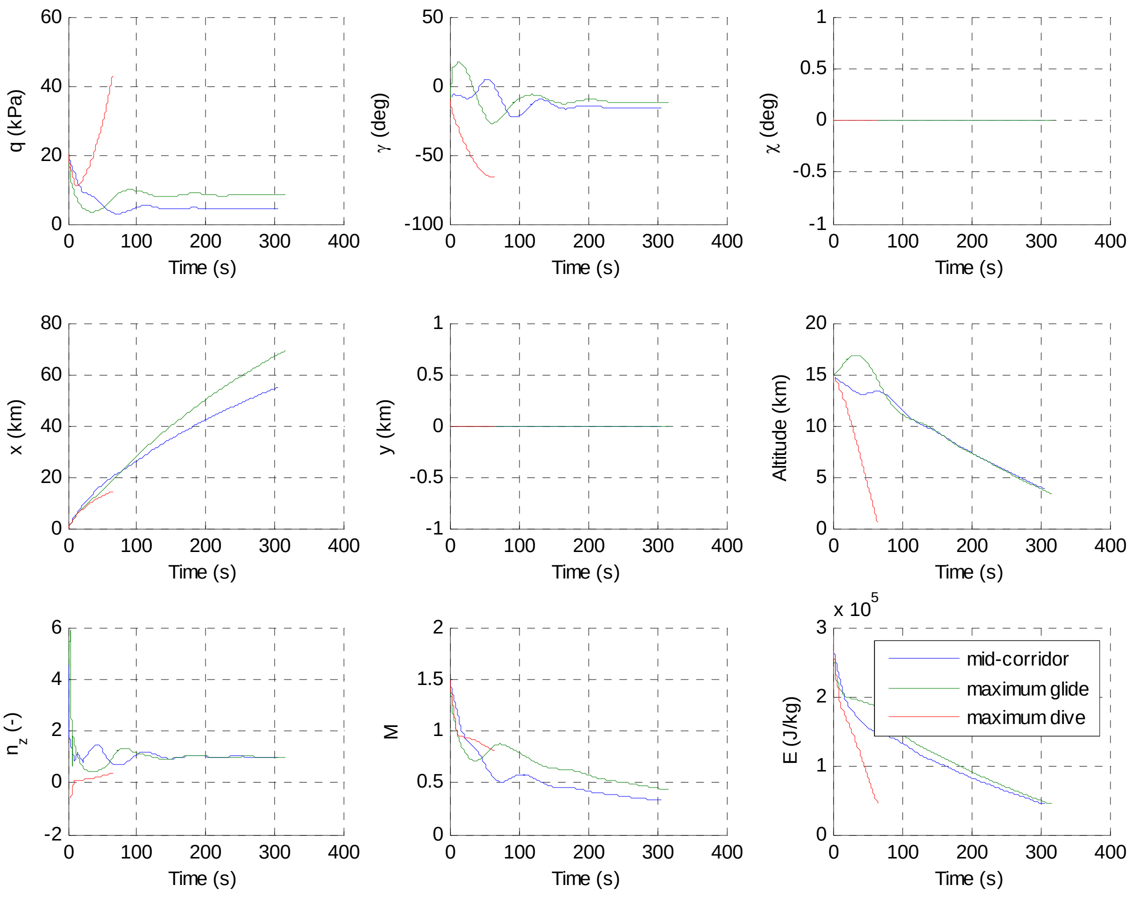

Figure 7.5: *Mid-corridor profile.*

As can be seen from the figure, all path constraints are satisfied and the resulting longitudinal motion is acceptable. However, the altitude and Mach number at ALI are not the desired ones. In order to give the tool the freedom it needs to adjust these two variables, the subsonic angle-of-attack was defined as an optimizable parameter. The respective first-order angle-of-attack parameterization is represented in **Figure 7.6**, where the maximum subsonic Mach number $M_{subsonic}$ and the subsonic angle-of-attack interval:

$$\alpha_{subsonic} = \left[\alpha_{sub,\min}, \alpha_{sub,\max}\right] \tag{7-2}$$

are inputs of the tool. $M_{subsonic}$ is kept free to allow the user to choose where to place the boundary between the subsonic and the transonic flight regimes.

Although **Figure 7.6** might give the idea that the subsonic flight regime is relatively short (in terms of duration), a careful look into **Figure 7.5** shows that, in fact, during the biggest part of the TAEM phase, the RLV is flying under subsonic conditions. Therefore, the idea that the desired Mach number at ALI can be achieved by selecting an adequate subsonic angle-of-attack seems reasonable. This is proved in **Figure 7.7**, where the Mach number at ALI is plotted as a function of $\alpha_{subsonic}$. The path constraints were never violated.

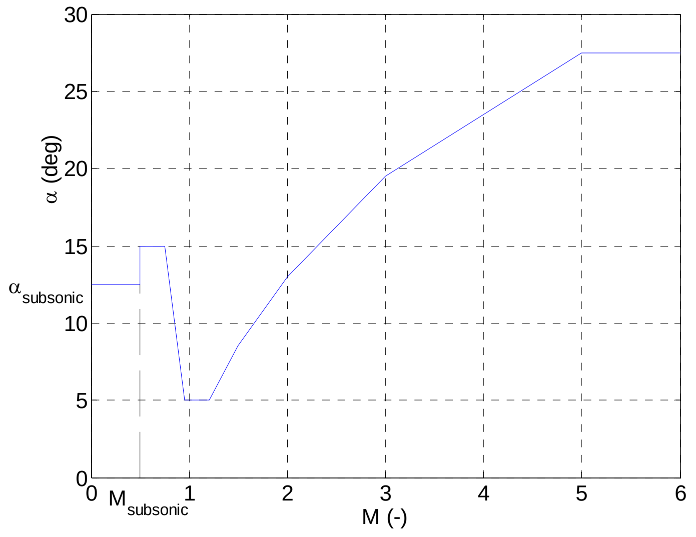

Figure 7.6: *Angle-of-attack parameterization.*

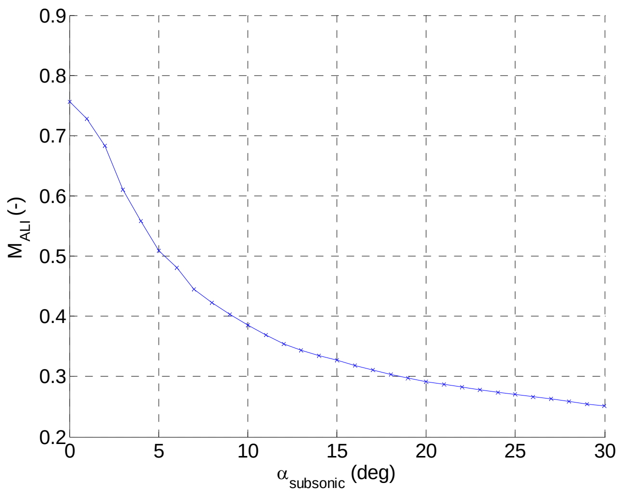

Figure 7.7: *Mach number at ALI as a function of subsonic angle-of-attack.*

Since typical values of M_{ALI} are comprehended between 0.4 and 0.6, as shown in **Figure 6.1**, the suggested first-order angle-of-attack parameterization is acceptable, as enough freedom is given to the optimiser.

Since the stopping condition is defined in terms of normalized energy, which is a function of altitude and Mach number, if the correct Mach number at ALI is attained, the altitude at ALI will also be the correct one.

7.1.1.2.2 Flight-Path Bank Angle Parameterization

The results presented in [Chartres *et al.*, 2005b] (see Section 6.2) were taken into account when parameterizing μ_k. The authors of this article use an NLP optimiser to calculate an optimal TAEM trajectory. This work outstands itself from the rest of the examined literature for being the one that follows the most general approach: the complete 3-DOF equations of motion are considered which means that, for example, no fixed geometry is assumed for the longitudinal and/or lateral profiles of the TAEM phase. Moreover, different path constraints, independent variables and cost functions are tested, making it a very serious and extensive work on TAEM trajectory optimization.

While an NLP optimiser might converge to a local minimum and, therefore, be dependent on the initial guess, it allows higher order parameterizations of the control functions than an interval analysis optimiser. Therefore, it is instructive to analyse the solutions of the former for defining an appropriate parameterization for the latter.

One of the observations made in [Chartres *et al.*, 2005b] is that the NLP solutions seem to follow some characteristic shapes. Three of these shapes are given in the original article. These same shapes are reproduced in **Figure 6.6**. An interesting characteristic of these solutions is that only two turns are performed during the TAEM phase: one left turn followed by a right turn or vice-versa. Taking this result into account, the flight-path bank angle was parameterized according to **Figure 7.8**.

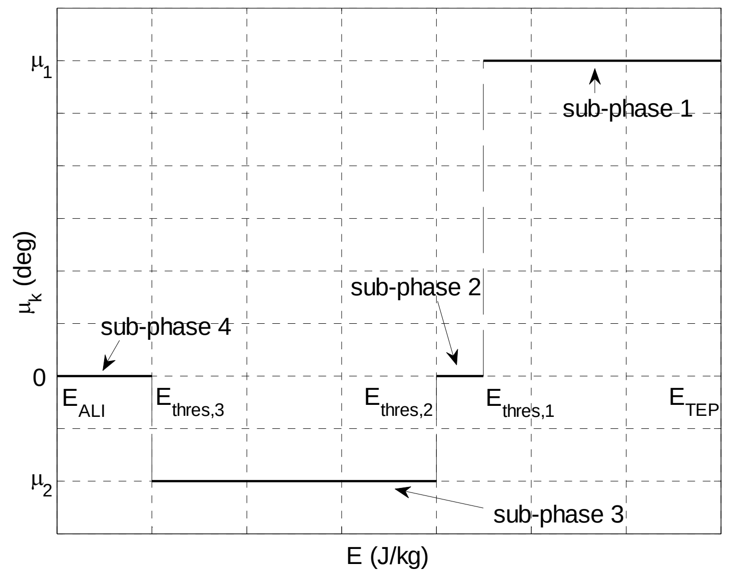

Figure 7.8: *Flight-path bank angle parameterization.*

where μ_1 and μ_2 are interval parameters given directly by the user and $E_{thres,1}$, $E_{thres,2}$ and $E_{thres,3}$ are real variables given indirectly by the user through $\Delta E_{rel,sub-phases}$:

$$\Delta E_{rel,sub-phases} = \left[\frac{E_{TEP} - E_{thres,1}}{E_{TEP} - E_{ALI}} \quad \frac{E_{thres,1} - E_{thres,2}}{E_{TEP} - E_{ALI}} \quad \frac{E_{thres,2} - E_{thres,3}}{E_{TEP} - E_{ALI}} \quad \frac{E_{thres,3} - E_{ALI}}{E_{TEP} - E_{ALI}} \right] \times 100\% \qquad (7\text{-}3)$$

In simple terms, $\Delta E_{rel,sub-phases}$ specifies how much energy should be dissipated in each TAEM sub-phase relatively to the total energy loss taking place during the TAEM phase, ΔE, which, in its turn, is given by:

$$\Delta E = E_{TEP} - E_{ALI} \qquad (7\text{-}4)$$

where E_{TEP} and E_{ALI} are the normalized energy at TEP and ALI, respectively.

The second-order parameterization specified in **Figure 7.8** allows the RLV to perform two independent turns like in the characteristic shapes of [Chartres *et al.*, 2005b]. However, these turns must be performed with a constant μ_k, which is a limitation of the parameterization. Moreover, the length of each sub-phase is fixed in terms of energy loss which is yet another limitation. Eventually, these limitations could be overcome by increasing the order of the parameterization, e.g., by treating $E_{thres,1}$, $E_{thres,2}$ and $E_{thres,3}$ as optimizable parameters. To keep the computational times within manageable values, the bank angle parameterization was chosen as simple as possible.

During sub-phase 2 and sub-phase 4, the bank angle is set to zero to allow a smoother transition between turns, in the first case, and to keep the RLV aligned with the runway during the period that immediately precedes the landing phase, in the second case.

The lengths of sub-phases 2 and 4 should be relatively small when compared with the lengths of sub-phases 1 and 3, to give the optimiser as much freedom as possible. Especially important is the length of sub-phase 3 since it is during this period, i.e., during the subsonic regime, that the vehicle achieves its highest turning capability. This can be seen in **Figure 7.9**.

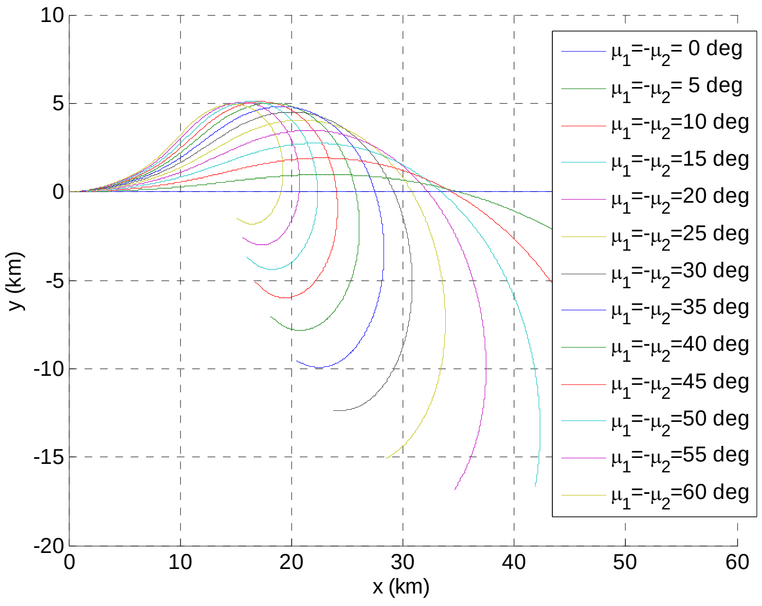

Figure 7.9: *Lateral profile for different values of* μ_1 *and* μ_2.

Another instructive plot is shown in **Figure 7.10** where the states and outputs of the system are given as a function of μ_1 and μ_2. The increase of the absolute value of μ_1 and μ_2 may lead to the violation of the path constraints. The 1-DOF methodology does not take this effect into account as the longitudinal and lateral motions are assumed to be decoupled.

Sometimes, to avoid highly populated areas, it is desirable to approach the landing localizer from one particular side (left or right). Adding this constraint to the optimization problem is relatively simple: a left-side approach corresponds to $\mu_2 \leq 0$ and a right-side approach corresponds to $\mu_2 \geq 0$.

7.1.1.3 Longitudinal Profile Calculation

In the 1-DOF methodology, the longitudinal profile is computed through a simple Monte Carlo optimization. Interval analysis is only used to generate the lateral profile.

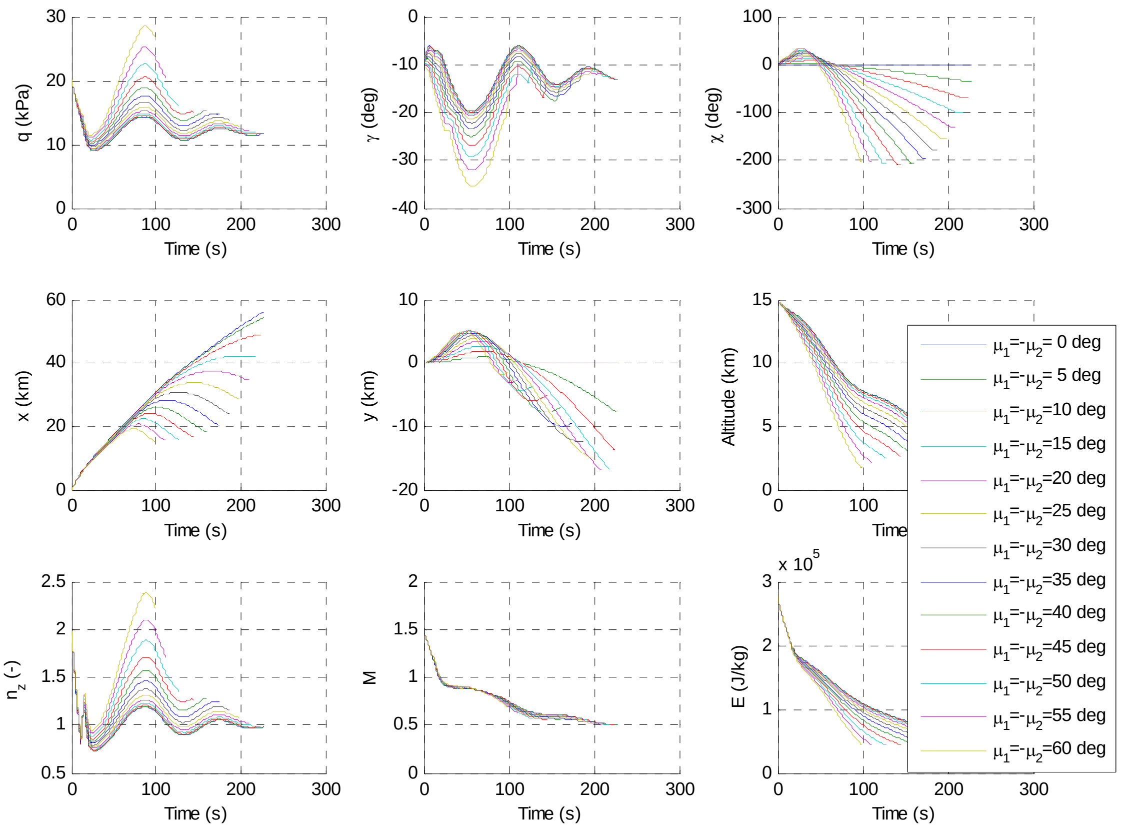

Figure 7.10: *States and outputs for different values of μ_1 and μ_2.*

The following steps describe the longitudinal profile calculation algorithm:

1) The mid-corridor angle-of-attack profile represented in **Figure 7.4** is calculated.

2) The Mach number at ALI is calculated as a function of $\alpha_{subsonic}$, creating a plot similar to **Figure 7.7**. M_{ALI} is evaluated every $\Delta\alpha$ degrees between $\alpha_{sub,\min}$ and $\alpha_{sub,\max}$. Subsonic angles-of-attack which violate the path constraints are not considered.

3) The subsonic angle-of-attack that yields the desired M_{ALI} (and h_{ALI}) is calculated through linear interpolation from **Figure 7.7**.

4) A last simulation is performed with the previously calculated $\alpha_{subsonic}$. The q, γ, C_L, C_D, ρ and E time histories are saved. They are later used in the generation of the lateral profile.

Applying these steps to the values given in **Table 7.1** yields a subsonic angle-of-attack of approximately 5.3 degrees and produces the states and outputs illustrated in **Figure 7.11**. The Mach number and altitude at ALI are 0.4987 and 3008m, respectively. The associated relative error is of 0.26% in both quantities. The algorithm runs in approximately 57 seconds.

The reader might be asking: “why wasn’t the longitudinal motion also optimized with interval analysis? $\alpha_{subsonic}$ is even given as an interval!” The fact is, optimizing the longitudinal motion with interval analysis would require more or less the same effort as optimizing the complete coupled system: an IVP (Initial Value Problem) ODE (Ordinary Differential Equation) problem would have to be solved numerically through interval analysis. Therefore, the decision was made to move directly into the complete problem, i.e., to move directly into 3-DOF methodology, instead of upgrading the 1-DOF methodology with an interval analysis optimization of the longitudinal profile.

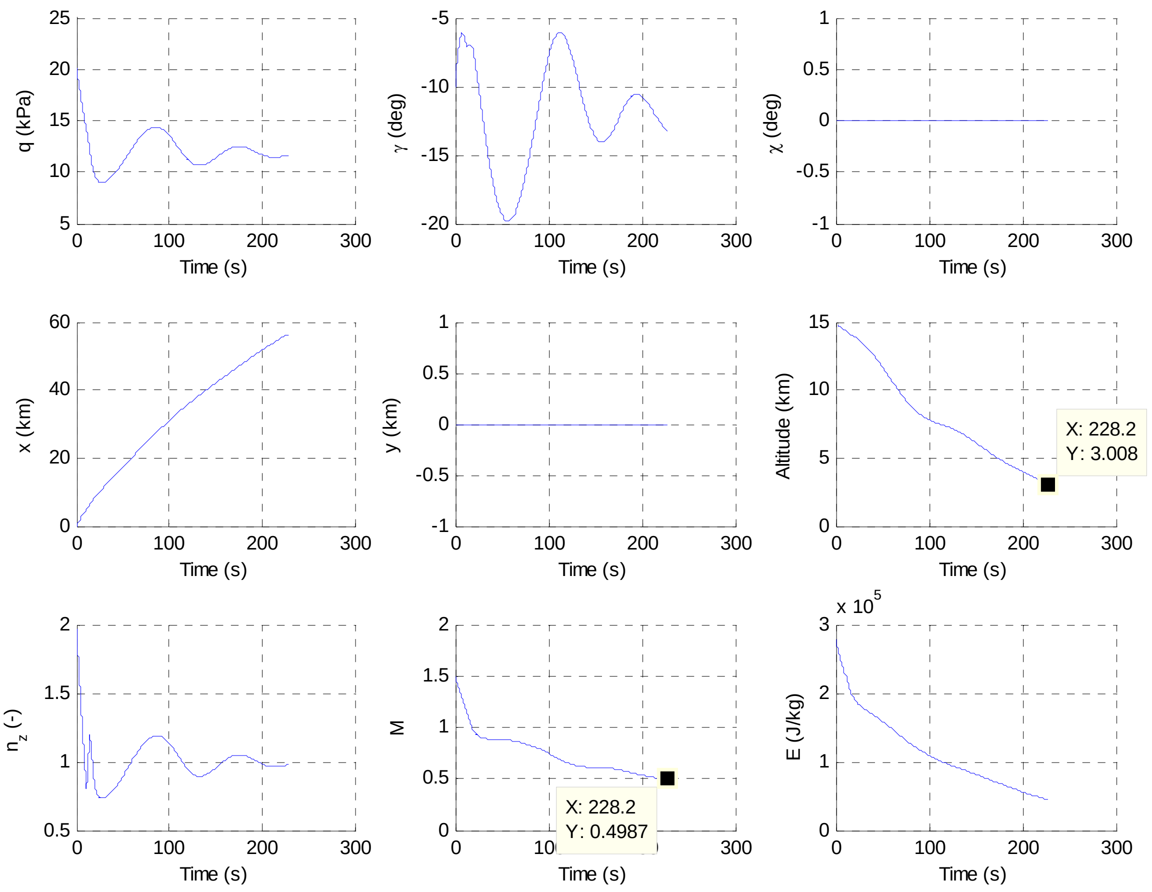

Figure 7.11: *Nominal longitudinal profile.*

A more careful reader might be wondering why γ_{TEP} is asked as an input instead of γ_{ALI}. This is associated with the fact that the longitudinal motion, contrarily to the lateral motion (see Section 7.1.1.4), is propagated forward in time and not backwards. While conceptually it would probably make more sense to propagate the longitudinal motion backwards in time, since the objective is to determine the Terminal Entry Gate, it would be harder for the user to find a viable γ_{ALI}. This idea is better explained with the help of **Figure 7.12** and **Figure 7.13**.

The influence of γ_{TEP} and γ_{ALI} on the time histories of q and γ is shown in **Figure 7.12** and **Figure 7.13**, respectively. In the first figure, the longitudinal motion is propagated forward in time while, in the second one, the longitudinal motion is propagated backwards in time. The previously calculated angle-of-attack profile is used together with the boundary conditions specified in **Table 7.1**. **Figure 7.12** shows that, for a relatively high range of γ_{TEP} (-40 to -10 degrees), the trajectories do not violate the $q_{\max}$ constraint and they all converge to similar values of q_{ALI} and γ_{ALI}. On the other hand, **Figure 7.13** shows that, for a relatively small range of γ_{ALI} (-15 to -10 degrees), the $q_{\max}$ constraint is sometimes violated and q_{TEP} and γ_{TEP} are spread over a large area.

As a consequence, if the longitudinal profile was propagated backwards in time, i.e., if γ_{ALI} was the input and not γ_{TEP}, it would be harder for the tool and for the user to find a matching $\alpha_{subsonic}$ and γ_{ALI} satisfying all path and boundary constraints. Since the main concern here is the lateral profile (which is the one in which interval analysis is applied) the simplest solution was adopted: propagating the longitudinal profile forward in time.

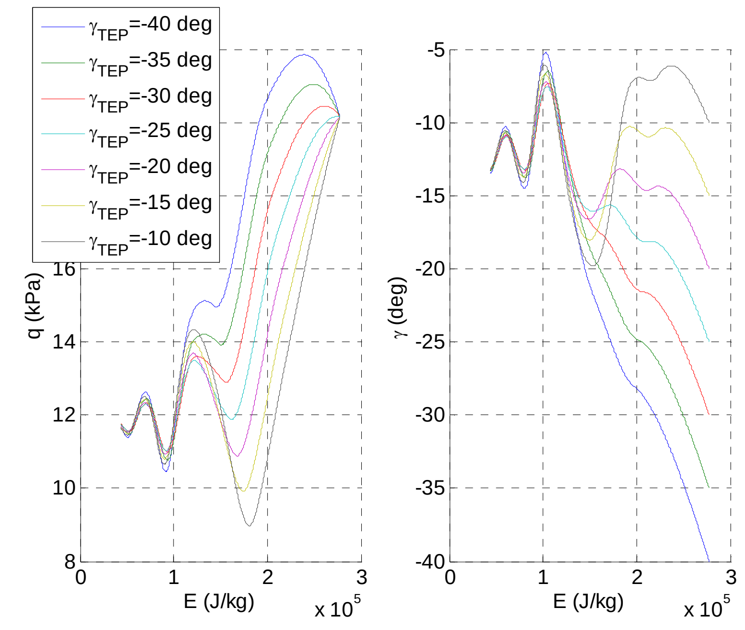

Figure 7.12: *Influence of γ_{TEP} on the time histories of q and γ.*

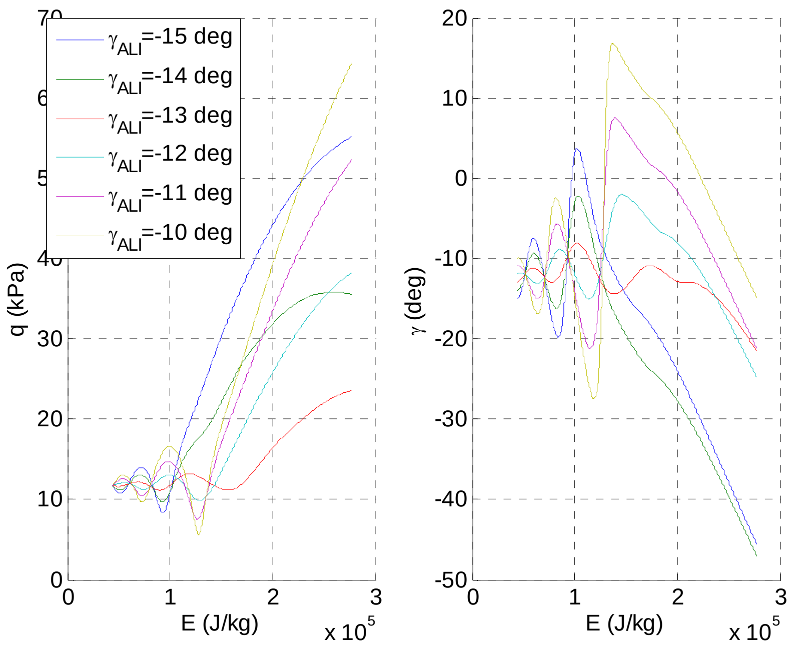

Figure 7.13: *Influence of γ_{ALI} on the time histories of q and γ.*

7.1.1.4 Lateral Profile Calculation

The calculation of the lateral profile is the last step in the determination of the Terminal Entry Gate. This last step is composed by two operations:

1) Branching of the intervals μ_1 and μ_2 in N_{μ_1} and N_{μ_2} subintervals, respectively.

2) Simultaneous backwards interval propagation of the lateral profile associated with each combination of μ_1 and μ_2 subintervals.

The branching algorithm used in this chapter is the same branching algorithm used in Chapter 5.

Since the longitudinal profile is assumed to be decoupled from the lateral profile, the state dynamics equations (6-21), (6-22) and (6-26) are of no interest for the calculation of the lateral profile.

From Section 7.1.1.3, the discrete time histories of $\rho(E)$, $C_L(E)$, $C_D(E)$, $q(E)$ and $\gamma(E)$ are known at the time steps:

$$E_i = E_{TEP} + (i-1)\frac{E_{TEP} - E_{ALI}}{N_{\text{int}}} \quad ,i = 1,...,N_{\text{int}} + 1 \tag{7-5}$$

These points can be connected through linear interpolation, making $\rho(E)$, $C_L(E)$, $C_D(E)$, $q(E)$ and $\gamma(E)$ defined at every $E \in [E_{ALI}, E_{TEP}]$. As a consequence, the right-side of (6-23) becomes completely defined at every $E \in [E_{ALI}, E_{TEP}]$. This means that a numerical interval integration method can be used to calculate $\chi_r(E)$ at every $E \in [E_{ALI}, E_{TEP}]$: an interval IVP ODE solver is not needed to do this. Moreover, with $\chi_r(E)$ for every $E \in [E_{ALI}, E_{TEP}]$ available, $x(E)$ and $y(E)$ can also be numerically integrated from (6-24) and (6-25). Avoiding an interval IVP ODE solver is the main reason behind the 1-DOF simplification.

The developed tool calculates the relative heading angle at TEP, $\chi_{r,TEP}$, from (6-23) using the interval rectangular integration method described in Section 3.6 as follows:

$$\chi_{r,TEP} = \chi_{r,ALI} + \int_{ALI}^{TEP} \frac{d\chi_r}{dE}(E)\,dE = \chi_{r,ALI} + \sum_{i=1}^{N_{\text{int}}} -\frac{\rho(E_i^I)C_L(E_i^I)\sin\mu_k(E_i^I)}{2C_D(E_i^I)q(E_i^I)\cos\gamma(E_i^I)}w(E_i^I) \tag{7-6}$$

where:

$$E_i^I = \left[E_{ALI} + (i-1)\frac{(E_{TEP} - E_{ALI})}{N_{\text{int}}}, E_{ALI} + i\frac{(E_{TEP} - E_{ALI})}{N_{\text{int}}}\right] \quad ,i = 1,...,N_{\text{int}} \tag{7-7}$$

and where $w(E_i^I)$ in (7-6) should be understood as the smallest interval containing the real value of $w(E_i^I)$ since $w(E_i^I)$ might not be a machine-representable number.

In practice, since the same number of integration steps is used to calculate the longitudinal and lateral profiles, the lower and upper bounds of E_i^I are directly the sampling points E_i. Analogously, the lower and upper bounds of $\rho\left(E_i^I\right)$, $C_L\left(E_i^I\right)$, $C_D\left(E_i^I\right)$, $q\left(E_i^I\right)$ and $\gamma\left(E_i^I\right)$ are directly the discrete time history points computed in Section 7.1.1.3. In **Figure 7.14**, this mechanism of extending a discrete time history into an interval time history is exemplified using the dynamic pressure profile shown in **Figure 7.11**.

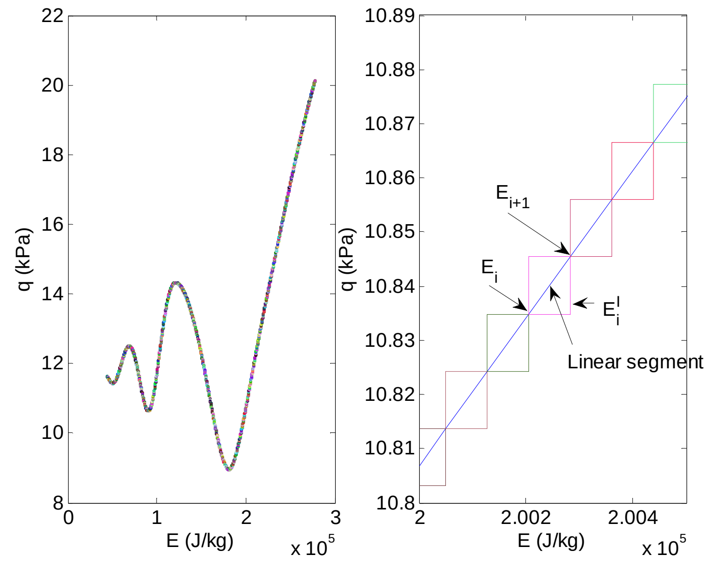

Figure 7.14: *Left: interval wrapping of $q\left(E\right)$. Right: zoom-in.*

Since the longitudinal profiles need to be enclosed inside interval boxes, the results produced by the interval integration algorithm are affected by the wrapping effect, by definition.

The relative heading angle during the normalized energy interval E_i^I is evaluated as follows:

$$\begin{aligned}\chi_r\left(E_i^I\right) &= \chi_{r,TEP} + \\ &+\sum_{j=1}^{i-1} -\frac{\rho\left(E_j^I\right)C_L\left(E_j^I\right)\sin\mu_k\left(E_j^I\right)}{2C_D\left(E_j^I\right)q\left(E_j^I\right)\cos\gamma\left(E_j^I\right)} w\left(E_j^I\right) + \\ &+\left(-\frac{\rho\left(E_i^I\right)C_L\left(E_i^I\right)\sin\mu_k\left(E_i^I\right)}{2C_D\left(E_i^I\right)q\left(E_i^I\right)\cos\gamma\left(E_i^I\right)}\right)\left(E_i^I - \inf\left(E_i^I\right)\right)\end{aligned} \qquad (7\text{-}8)$$

After computing $\chi_r\left(E_i^I\right)$, the x- and y-positions at TEP are calculated in the same way as $\chi_{r,TEP}$:

$$x_{TEP} = x_{ALI} + \int_{ALI}^{TEP} \frac{dx}{dE}\left(E\right)dE = x_{ALI} + \sum_{i=1}^{N_{\text{int}}} -\frac{m\cos\gamma\left(E_i^I\right)\cos\chi_r\left(E_i^I\right)}{C_D\left(E_i^I\right)q\left(E_i^I\right)S} w\left(E_i^I\right) \qquad (7\text{-}9)$$

$$y_{TEP} = y_{ALI} + \int_{ALI}^{TEP} \frac{dy}{dE}(E)\,dE = y_{ALI} + \sum_{i=1}^{N_{\text{int}}} -\frac{m\cos\gamma\left(E_i^I\right)\sin\chi_r\left(E_i^I\right)}{C_D\left(E_i^I\right)q\left(E_i^I\right)S} w\left(E_i^I\right) \quad (7\text{-}10)$$

Note that the form of (6-23), (6-24) and (6-25) is already optimal in terms of dependency.

In order to validate the interval integration algorithm, a new SIMLULINK model was built to calculate the lateral profile based on the 1-DOF methodology. This SIMULINK model is shown in **Figure 7.15**. It linearly interpolates the longitudinal profile values calculated in Section 7.1.1.3 to compute χ_r, x and y, just like its interval counterpart. Like the SIMULINK model of **Figure 7.2**, it uses MATLAB's *ode45* solver with a relative tolerance of 1e-6.

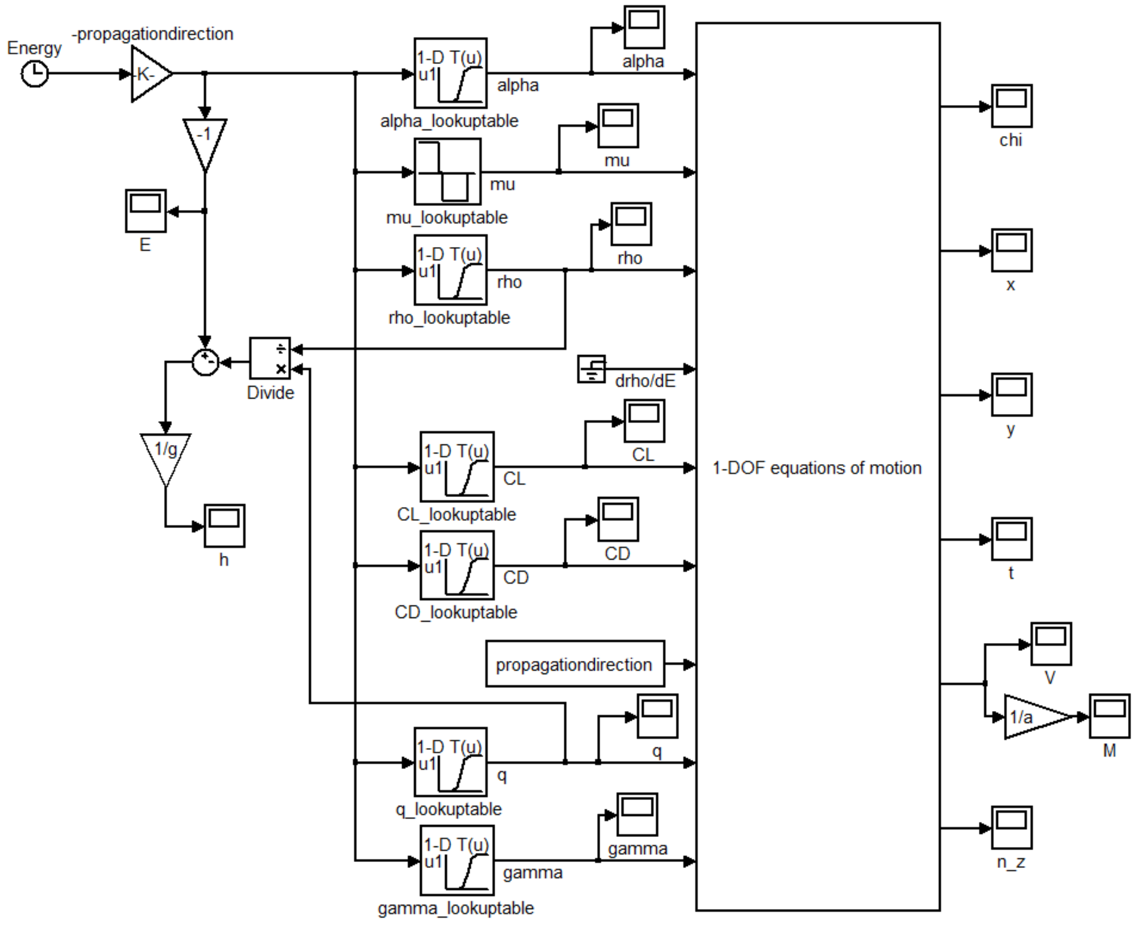

Figure 7.15: *1-DOF SIMULINK model.*

The 3-DOF and the 1-DOF SIMULINK models are compared in **Figure 7.16**. As expected, the two models yield the same states and outputs for $\mu_1 = \mu_2 = 0$ degrees. For $\mu_1 = -\mu_2 = 60$ degrees, the lateral states x, y and χ_r produced by the two models are no longer the same. As expected, the 1-DOF model overestimates the TAEM flight duration since it does not take into account the reduction of the vertical component of the lift force when the vehicle turns.

In **Figure 7.17**, the results produced by the interval integration algorithm are compared with the ones produced by the 1-DOF SIMULINK model for $\chi_{r,ALI} = x_{ALI} = y_{ALI} = 0$ and $\mu_1 = -\mu_2 = 60^{\circ}$. As can be seen, the SIMULINK solution is correctly bounded by the interval solution, verifying the correctness of the interval integration algorithm.

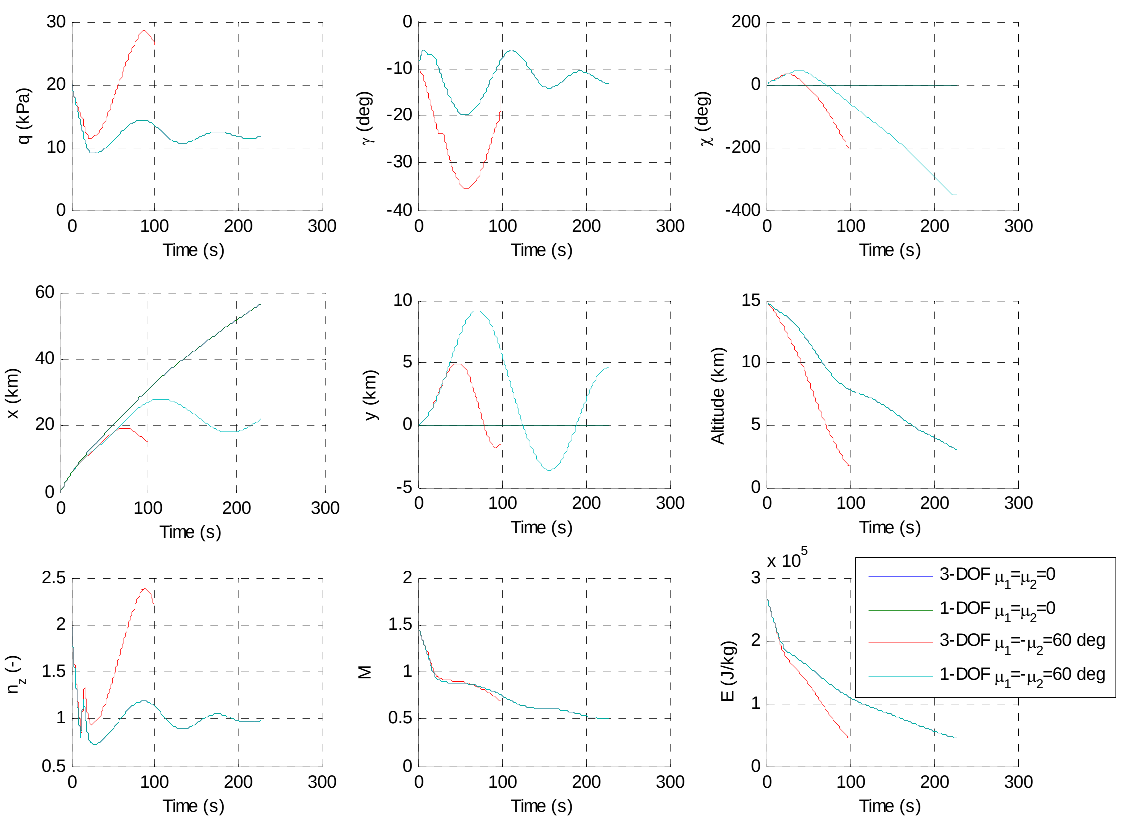

Figure 7.16: *3-DOF versus 1-DOF SIMULINK model.*

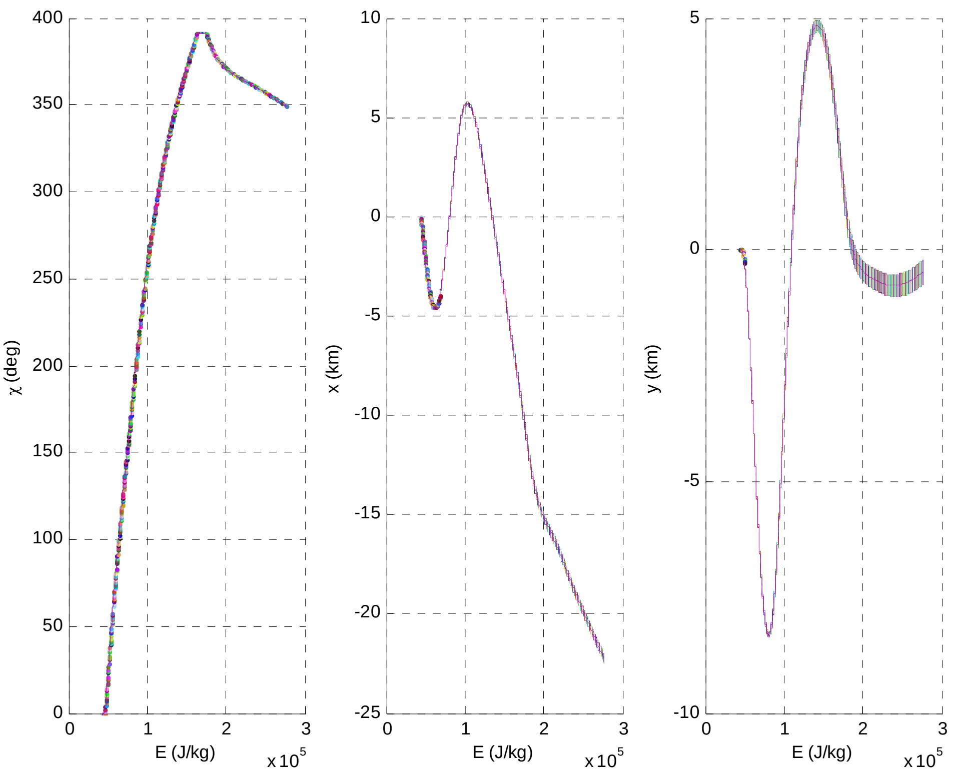

Figure 7.17: *Correctness verification of the interval integration algorithm.*

Mostly due to the wrapping effect, $\chi_{r,TEP}$, x_{TEP} and y_{TEP} are not as sharp as one would like them to be. Their widths are approximately 0.78°, 356m and 520m, respectively. It should be noted that these results were obtained for

$w(\chi_{r,ALI}) = w(x_{ALI}) = w(y_{ALI}) = w(\mu_1) = w(\mu_2) = 0$. As a consequence, the size of the Terminal Entry Gate will be overestimated. There will also be some consequences for the TAEM optimal trajectory design but these will be mentioned in Section 7.1.2.

Sharper results would require more sophisticated interval integration methods than the rectangular method, like the ones explained in [Moore, 1966]. In this thesis, solving the more complex 3-DOF problem and finding an appropriate interval IVP-ODE solver was given priority over this. The 1-DOF methodology was developed more as a feasibility study than as an operational tool. Therefore, upgrading the 1-DOF tool with a more accurate interval integration method was deemed unnecessary and unproductive.

7.1.1.5 Results

Now that the theory has been explained, some results can finally be shown.

The subsequent plots were created using the values given in **Table 7.1**. The lateral profile calculation algorithm runs in approximately 74 seconds.

Figure 7.18 shows the position of the Terminal Entry Gate. The plane represented in **Figure 7.18** is a plane of constant normalized energy, equal to E_{TEP}. Since the longitudinal profile is fixed, the plane represented in **Figure 7.18** is also a plane of constant altitude and Mach number, respectively equal to h_{TEP} and M_{TEP}.

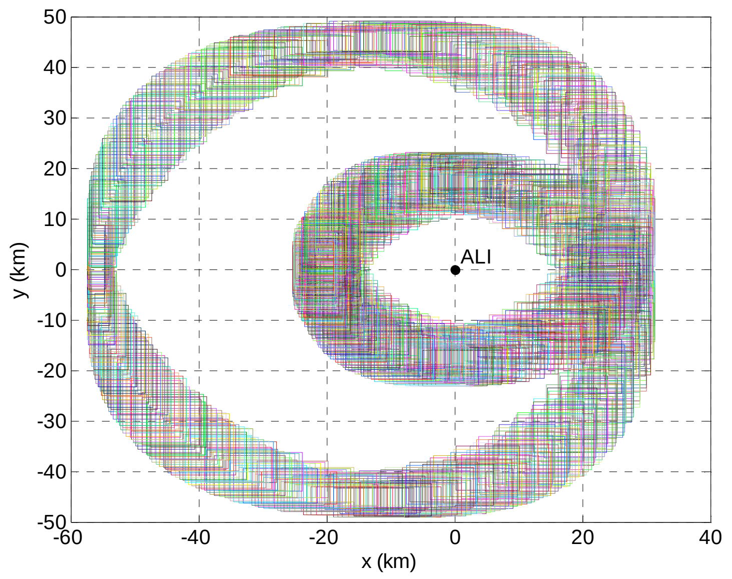

Figure 7.18: *Position of the Terminal Entry Gate.*

Each square represented in **Figure 7.18** is obtained by propagating backwards in time the lateral state at ALI, namely, $\chi_{r,ALI}$, x_{ALI} and y_{ALI}, from E_{ALI} until E_{TEP} using interval integration. Each square is associated with a particular combination of μ_1 and μ_2 angles. Each combination results from splitting the original μ_1 and μ_2 intervals given in **Table 7.1** into N_{μ_1} and N_{μ_2} subintervals, respectively.

In **Figure 7.19**, the squares are transformed into cubes by plotting them against the μ_1 and μ_2 subintervals associated with each of them. There are two reasons why the plot associated with μ_2 looks more like a spiral than the plot associated with μ_1:

1) $N_{\mu_2} > N_{\mu_1}$ and;

2) As already noticed in Section 7.1.1.2.2, the shape of the trajectories is more affected by μ_1 than by μ_2.

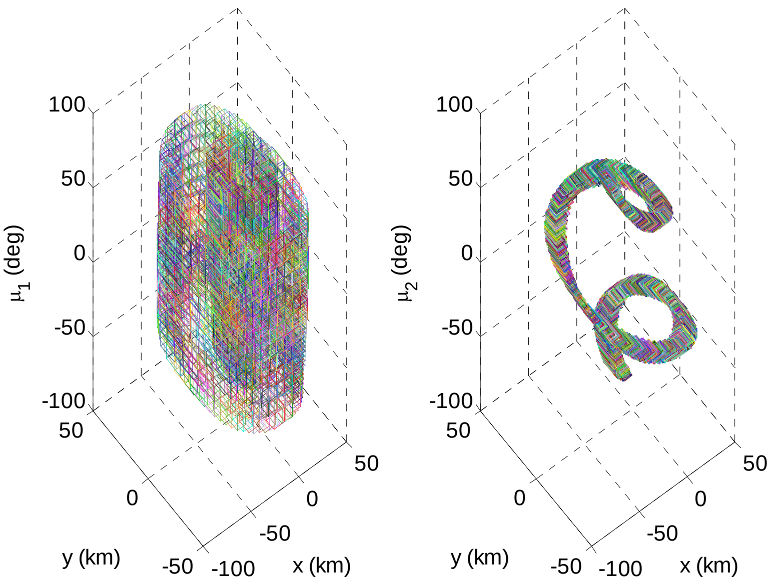

Figure 7.19: *μ_1 and μ_2 subintervals associated with each square.*

There is also an $\chi_{r,TEP}$ interval associated with each square represented in **Figure 7.18**. This basically means that if the RLV starts the TAEM phase in a certain square, the guidance scheme will only be able to correctly steer the vehicle until the auto-landing interface point, if the relative heading angle of the RLV at TEP is within certain bounds. These bounds are given in **Figure 7.20**.

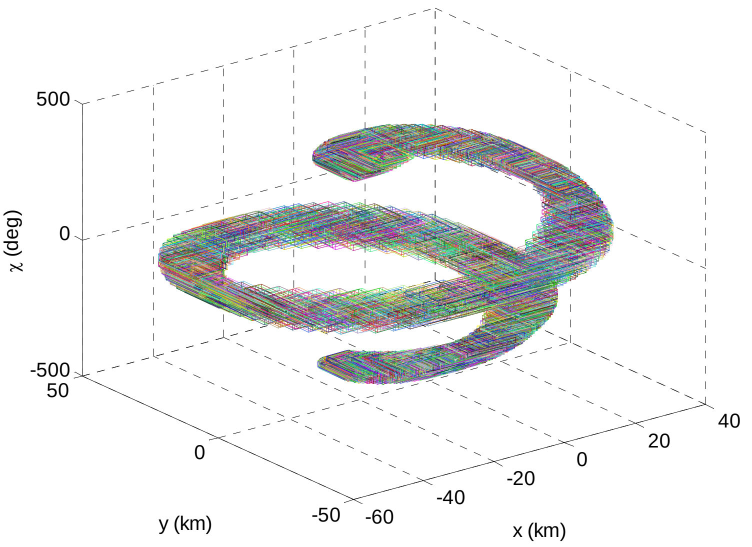

Figure 7.20: *$\chi_{r,TEP}$ intervals associated with each square.*

It should be noted that the squares represented in **Figure 7.18** and the cubes represented in **Figure 7.20** are an overestimation of the true TEG. However, due to the properties of interval analysis, the true TEG is mathematically guaranteed to be contained inside the calculated TEG. This means that, for example, the trajectory design algorithm described in Section 7.1.2 will not able to prove the feasibility of a TEP point laying outside the volume represented in **Figure 7.20**. A feasible TEP point should be understood as a TEP point from where the suggested guidance scheme is able to steer the vehicle until an acceptable ALI point[12] while satisfying all path constraints. But what the trajectory design algorithm can do is sometimes prove the feasibility or infeasibility of a TEP point contained inside the volume represented in **Figure 7.20**.

A Monte Carlo analysis was performed in order to attest how many points inside the calculated TEG shown in **Figure 7.20** could be proved feasible or infeasible. A total of 100 simulations were run. The TEP points ($\chi_{r,TEP}$, x_{TEP} and y_{TEP}) were chosen completely randomly from the calculated TEG gate. Three criteria were used to stop the simulations:

1) feasibility could be proved;

2) infeasibility could be proved;

3) Number of boxes exceeds 1500.

The entire range of μ_1 and μ_2 was searched in each simulation, i.e., $\mu_1 = \mu_2 = [-60, 60]$ degrees, to take advantage of the fact that several cubes in **Figure 7.20** share the same volume. The boxes were branched based on $N_{\mu_1} = 2$ and $N_{\mu_2} = 2$. The results are shown in **Figure 7.21**.

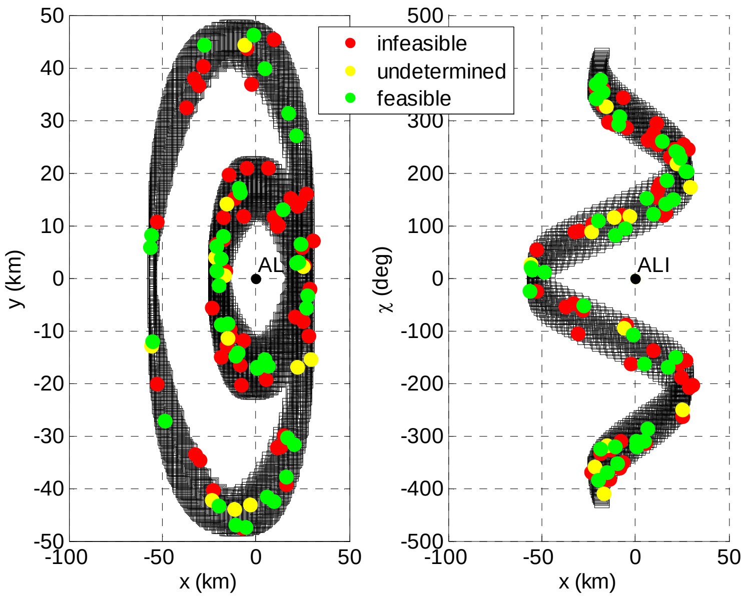

Figure 7.21: *Feasibility Monte Carlo analysis: random TEP points.*

From the 100 tested TEP points, 38% could be proved feasible, 50% could be proved infeasible and 12% could neither be proved feasible or unfeasible.

Another Monte Carlo analysis is shown in **Figure 7.22**. This time, instead of choosing completely random points, only the centres of the cubes represented in **Figure 7.20** were selected as possible starting points. All of them could be proved feasible.

[12] *The meaning of acceptable ALI point is mathematically defined in Section 7.1.2.3.*

Some might defend that calculating a TEG gate in which there are not any infeasible points would be a more significant result. While the current TEG determination methodology cannot give such guarantee, the sharpness of the TEG enclosure can be controlled by:

1) changing the interval integration algorithm: a more sophisticated interval integration method would reduce the wrapping effect and produce sharper bounds on the TEG;

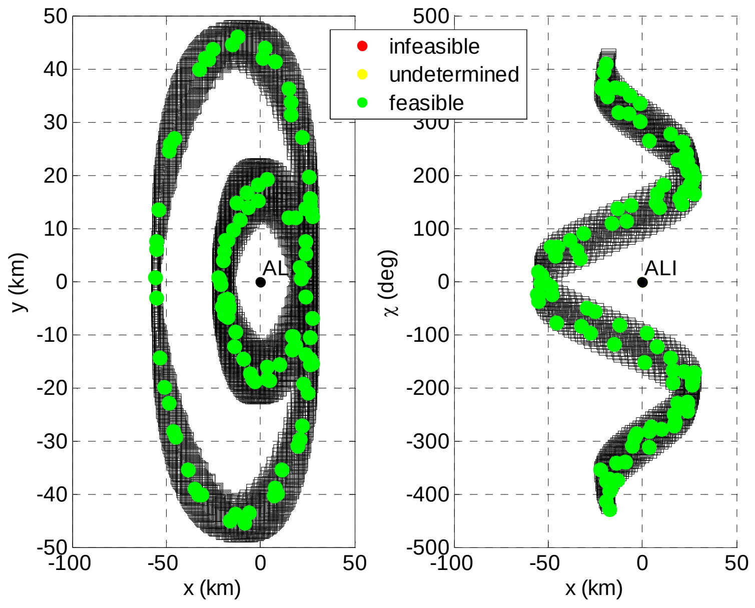

Figure 7.22: *Feasibility Monte Carlo analysis: random midpoints.*

2) increasing N_{int}: increasing N_{int} reduces the wrapping effect inherent to the rectangular interval integration method. For example, for $N_{\text{int}} = 3000$, all the cubes shown in **Figure 7.20** can be enclosed by a bigger cube with a volume of approximately 1.328e11rad.m^2. For $N_{\text{int}} = 5000$, the volume of this cube reduces to 1.326e11rad.m^2, which corresponds to a decrease of approximately 0.2%.

3) increasing N_{μ_1} and/or N_{μ_2}: increasing N_{μ_1} and/or N_{μ_2} decreases the wrapping effect inherent to the discretization in cubes of the total volume of the TEG. In other words, the bigger the number of subintervals in each μ_1 and/or μ_2 are divided, the bigger will be the resolution of **Figure 7.20**, i.e., the smaller will be the cubes shaping the TEG. However, one should take into account that the minimum dimension of each cube is limited by the wrapping effect associated with the interval integration algorithm. For example, for $N_{\mu_2} = 140$, all the cubes shown in **Figure 7.20** can be enclosed by a bigger cube with a volume of approximately 1.328e11rad.m^2. For $N_{\mu_2} = 280$, the volume of this cube reduces to 1.317e11rad.m^2, which corresponds to a decrease of approximately 0.8%.

Another positive feature added by interval analysis to the TEG determination problem is the possibility of defining $\chi_{r,ALI}$, x_{ALI} and y_{ALI} either as intervals or as real values. This simplifies, for example, the execution of sensitivity analysis studies about the size of the ALI gate. As an example, in **Figure 7.23**, the position of the TEG for $\chi_{r,ALI} = x_{ALI} = y_{ALI} = 0$ is plotted. It is easy to see the reduction in area with respect to **Figure 7.18**.

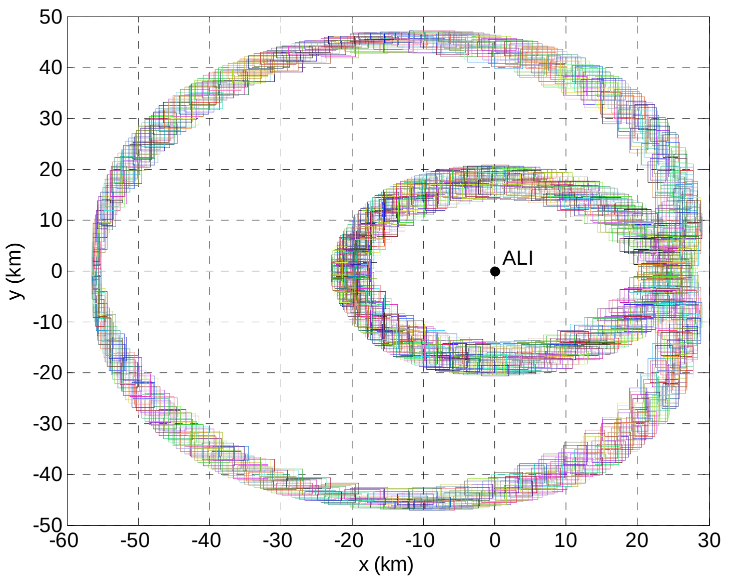

Figure 7.23: *Position of the TEG for* $\chi_{r,ALI} = x_{ALI} = y_{ALI} = 0$.

7.1.2 TAEM Optimal Trajectory Design

The 1-DOF TAEM trajectory design tool described hereafter assumes no coupling between the longitudinal and lateral movements. Like the aforementioned TEG determination tool, it uses a simple Monte Carlo optimization algorithm to fix the longitudinal motion. The lateral motion is then optimized using interval analysis and a performance index based on the final state error.

In **Figure 7.24**, a flowchart representing the main blocks of the trajectory design algorithm is given. For the longitudinal profile calculation algorithm, the reader is referred to Section 7.1.1.4. Likewise, for the lateral profile interval propagation algorithm, the reader is referred to Section 7.1.1.4. The same concepts, apart from some minor sign changes, are used to propagate the lateral profile forward and backwards in time.

The remaining blocks of **Figure 7.24** are described hereafter.

7.1.2.1 Input File

Table 7.2 summarizes all the main inputs of the trajectory design tool and the values used during the subsequent simulations. The exact structure of the input file and an example are given in Appendix B.

Note that $\chi_{r,TEP}$, x_{TEP} and y_{TEP} are defined in **Table 7.2** as point intervals. This is the typical case of interest.

The only variables in **Table 7.2** that are completely new are $\chi_{r,weight}$, x_{weight} and y_{weight}. They are related with the performance index. Their meaning is explained in Section 7.1.2.3.

7.1.2.2 Branching

The algorithm of Section 5.2.4 is used to branch μ_1 and μ_2 with $N_{split} = \begin{bmatrix} N_{\mu_1} & N_{\mu_2} \end{bmatrix}^T$.

Since the state dynamics equations need to be integrated numerically (in Chapter 5 they were integrated analytically) due to their complexity, HC cannot be used to shrink the initial search region of μ_1 and μ_2. In order to do that, both parameters need to be branched, since only box deletion techniques are available.

The influence of N_{split} on the computational time is studied in Section 7.1.2.9.

7.1.2.3 Final State Constraint

In Chapter 5, the final state constraint was used together with HC to shrink the boxes P_i, where i indexes all the boxes created by the branching algorithm during a certain cycle. Since the state dynamics equations are now numerically integrated, applying HC to the final state constraint is no longer an option. However, the final state constraint can still be used to delete certainly infeasible boxes with respect to the final state constraint.

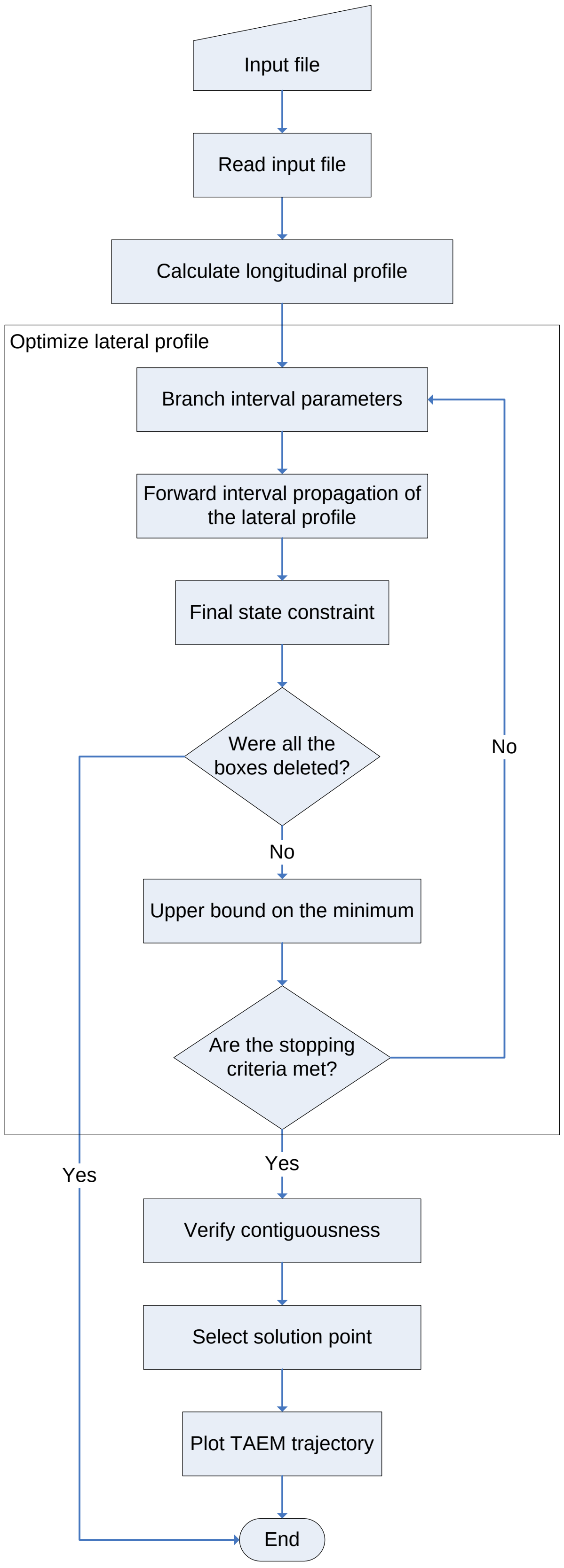
Input file
Read input file
Calculate longitudinal profile
Optimize lateral profile
Branch interval parameters
Forward interval propagation of the lateral profile
Final state constraint
Were all the boxes deleted?
No
Upper bound on the minimum
Are the stopping criteria met?
No
Yes
Yes
Verify contiguousness
Select solution point
Plot TAEM trajectory
End

Figure 7.24: *TAEM Optimal Trajectory Design Algorithm Flowchart.*

***Table 7.2:** Main Inputs of the TAEM Optimal Trajectory Design Tool.*

Input	Symbol	Format	Value	Units
Relative heading angle at TEP	$\chi_{r,TEP}$	*interval*	$[220, 220]$	deg
x-position at TEP	x_{TEP}	*interval*	$[25000, 25000]$	m
y-position at TEP	y_{TEP}	*interval*	$[0, 0]$	m
Flight-path bank angle (interval) during sub-phase 1	μ_1	*interval*	$[-60, 60]$	deg
Flight-path bank angle (interval) during sub-phase 3	μ_2	*interval*	$[-60, 60]$	deg
Number of subintervals in which μ_1 is split	N_{μ_1}	*int*	2	-
Number of subintervals in which μ_2 is split	N_{μ_2}	*int*	4	-
Reference area of the vehicle	S	*float*	110	m^2
Re-entry mass of the vehicle	m	*float*	26,029	kg
Aerodynamic lift coefficient	$C_L(\alpha, M)$	*float*	N/A	-
Aerodynamic drag coefficient	$C_D(\alpha, M)$	*float*	N/A	-
Minimum and maximum trimmable α as a function of M	$\alpha_{\min}(M)$ and $\alpha_{\max}(M)$	*float*	N/A	deg
Mach number at TEP	M_{TEP}	*float*	1.5	-
Mach number at ALI	M_{ALI}	*float*	0.5	-
Altitude at TEP	h_{TEP}	*float*	15,000	m
Altitude at ALI	h_{ALI}	*float*	3,000	m
Number of integration steps	N_{int}	*int*	3,000	-
Flight-path angle at TEP	γ_{TEP}	*float*	-10	deg
Minimum α during subsonic regime	$\alpha_{sub,\min}$	*float*	0	deg
Maximum α during subsonic regime	$\alpha_{sub,\max}$	*float*	30	deg
Subsonic α search step	$\Delta\alpha$	*float*	1	deg
Maximum subsonic M	$M_{subsonic}$	*float*	0.75	-
Maximum dynamic pressure supported by the vehicle	$q_{\max}$	*float*	40,000	Pa

Input	**Symbol**	**Format**	**Value**		**Units**
Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\min}$	*float*	-0.75		-
Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\max}$	*float*	2.5		-
Sub-phases relative length	$\Delta E_{rel,sub-phases}$	*float*	$\begin{bmatrix}0.44 & 0.05 & 0.5 & 0.01\end{bmatrix}$		- (%)
χ_r cost function weight	$\chi_{r,weight}$	*float*	3		deg
x-position cost function weight	x_{weight}	*float*	1,000		m
y-position cost function weight	y_{weight}	*float*	1,000		m
Initial upper bound on the global minimum	$\overline{J}$	*float*	Inf		-
Stopping criterion based on the width of the parameters	ε_P	*float*	$\varepsilon_{\mu_1} = \text{Inf}$	$\varepsilon_{\mu_2} = \text{Inf}$	deg
Stopping criterion based on width of the cost function	ε_J	*float*	0.3		-

The most general case of an equality constraint $f(P) = a$, where $f(P): \mathbb{R}^l \rightarrow \mathbb{R}^r$ and $a \in \mathbb{R}^r$ is a non-degenerate interval vector, $a = [\underline{a}, \overline{a}]$, will be considered. In the case at hand, $r = 3$ ($\chi_{r,ALI}$, x_{ALI} and y_{ALI}) and $l = 2$ (μ_1 and μ_2). After evaluating $f(P)$ over a box P_i, $f(P_i) = \left[\underline{f(P_i)}, \overline{f(P_i)}\right]$, it is said that:

1) P_i is *(certainly) feasible* if $f_j(P_i) \subseteq a_j$ for **all** $j = 1,...,r$ and that P_i is *(certainly) strictly feasible* if $f_j(P_i) \subset a_j$ for **all** $j = 1,...,r$.

2) P_i is (*certainly) infeasible* if $f_j(P_i) \not\subset a_j$ for **any** $j = 1,...,r$.

Any infeasible box P_i can be safely deleted as it does not satisfy the equality constraint.

According to the fundamental theorem of interval analysis, all the points contained inside a feasible box satisfy the equality constraint.

Note that if $a \in \mathbb{R}^r$ is a degenerate interval vector, then the above definition cannot be used to prove that a box P_i is feasible with respect to the equality constraint $f(P) = a$, except in the very unlikely event that $f(P_i)$ is also a degenerate interval vector exactly equal to a.

In Chapter 5, $a \in \mathbb{R}^r$ could be defined as a point interval vector since HC and **Theorem 4.1** were used to prove feasibility with respect to the equality constraints. Since now the equations of motion have to be numerically integrated, **Theorem 4.1** can no longer be applied and there is no choice but to define $\chi_{r,ALI}$, x_{ALI} and y_{ALI} as intervals.

Since most of the time, satisfying a certain ALI point is not a concern, only a certain ALI gate, i.e., the nominal ALI point does not need to be strictly satisfied, only the tolerances prescribed on the final state, there is no major consequence in having to define $\chi_{r,ALI}$, x_{ALI} and y_{ALI} as intervals. Thus, while a certain box cannot be guaranteed to satisfy a certain nominal ALI point, it can be guaranteed to satisfy the prescribed tolerances on the final state.

$\chi_{r,ALI}$, x_{ALI} and y_{ALI} are given in the following format to the tool:

$$\chi_{r,ALI} = \left[-\chi_{r,weight}, \chi_{r,weight}\right] \tag{7-11}$$

$$x_{ALI} = \left[-x_{weight}, x_{weight}\right] \tag{7-12}$$

$$y_{ALI} = \left[-y_{weight}, y_{weight}\right] \tag{7-13}$$

where $\chi_{r,weight}$, x_{weight} and y_{weight} are the tolerances associated with $\chi_{r,ALI}$, x_{ALI} and y_{ALI}, respectively. An acceptable ALI point is therefore a point contained inside (7-11), (7-12) and (7-13).

The wider $\chi_{r,ALI}$, x_{ALI} and y_{ALI} are, the easier it will be to prove feasibility, but the harder it will be to prove infeasibility.

7.1.2.4 Cost Function

In [Chartres *et al.*, 2005b], several cost functions are tested, e.g., maximizing or minimizing range, maximizing or minimizing flight time, minimizing control effort or combinations of these. At the end, their conclusion is that, while the loads and the dynamic pressure should be minimized, the primary concern should be to minimize the final state error, as this is the most difficult constraint to meet.

Therefore, and following the steps of [Costa, 2003] and [Kluever, 2007], the final state error was defined as the cost function as follows:

$$J(P_i) = \left(\frac{\chi_{r,ALI}(P_i) - \chi_{r,ALI,nom}}{\chi_{r,weight}}\right)^2 + \left(\frac{x_{ALI}(P_i) - x_{ALI,nom}}{x_{weight}}\right)^2 + \left(\frac{y_{ALI}(P_i) - y_{ALI,nom}}{y_{weight}}\right)^2 \tag{7-14}$$

where:

- $\chi_{r,ALI}(P_i)$, $x_{ALI}(P_i)$ and $y_{ALI}(P_i)$ are the calculated values of χ_r, x and y at ALI for a certain box P_i;

- $\chi_{r,weight}$, x_{weight} and y_{weight} are used to undimensionalize the cost function and;

- $\chi_{r,ALI,nom} = x_{ALI,nom} = y_{ALI,nom} = 0$ are the nominal values of χ_r, x and y at ALI[13].

The cost function (7-14) is constituted only by an end-cost term, as can be easily seen by comparing (7-14) with (5-1).

An approximate graphical representation of (7-14) is given in **Figure 7.25** based on the values shown in **Table 7.2**. The SIMULINK model of **Figure 7.15** was used to propagate the TAEM trajectory.

As the reader can see, the cost function (7-14) has several local minima. If this problem was solved with a standard gradient-based optimization method, the final solution might have been a local minimum, depending on the initial guess. However, with interval analysis, the global optimum is guaranteed to be correctly bounded.

According to **Figure 7.25**, the global minimum is:

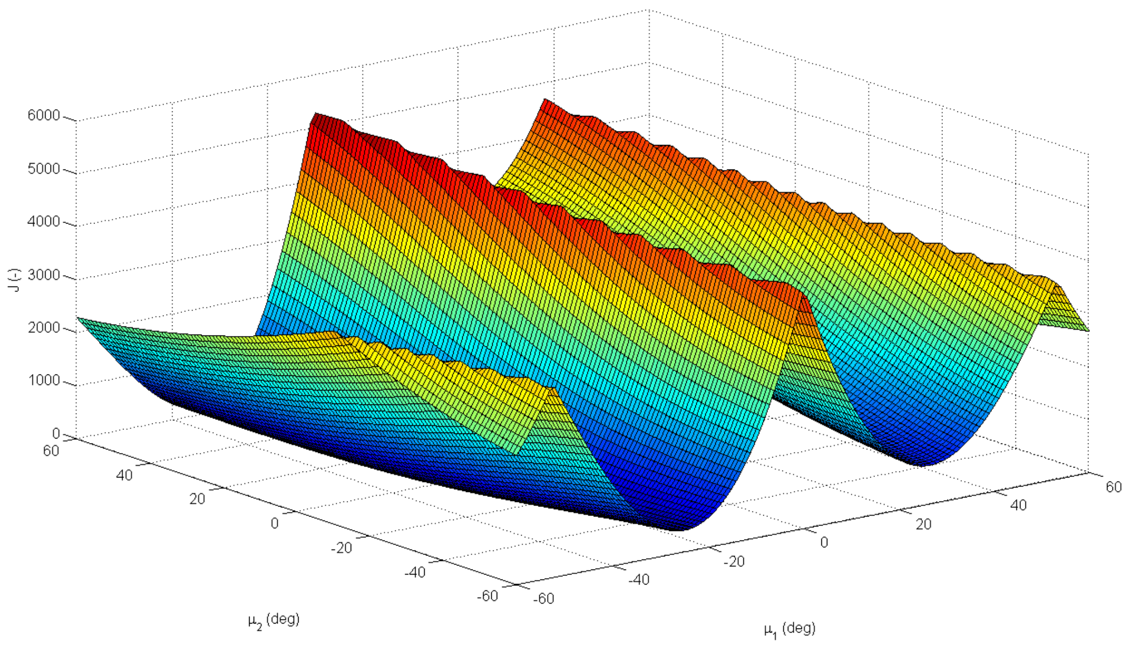

Figure 7.25: *Cost function.*

$$J^* \approx 0.14418 \tag{7-15}$$

the global minimum point is:

$$\mu_1^* \approx 10.41^{\circ} \tag{7-16}$$

$$\mu_2^* \approx -30.50^{\circ} \tag{7-17}$$

and the associated final state is:

$$\chi_r^*\left(E_{ALI}\right) \approx -0.49^{\circ} \tag{7-18}$$

$$x^*\left(E_{ALI}\right) \approx -334m \tag{7-19}$$

[13] *Note that the nominal value of χ_r at ALI is actually equal to $\chi_{r,ALI,nom} = 0 + k2\pi, k \in \mathbb{Z}$.*

$$y^*\left(E_{ALI}\right) \approx -80m \qquad (7\text{-}20)$$

(7-15)-(7-20) will be used in Section 7.1.2.9 to verify the results of the global optimizer.

7.1.2.5 Using an upper bound on the minimum

As mentioned in Sections 4.2.2.2 and 5.2.2.2, any box P_i for which $\underline{J_i} > \overline{J}$, where $\underline{J_i}$ is calculated from $J\left(P_i\right) = \left[\underline{J_i}, \overline{J_i}\right]$, can be safely deleted. The only condition is that $\overline{J}$ can only be updated using boxes that contain at least one point satisfying all equality and inequality constraints, i.e., feasible boxes.

Assume a box P_{feas} has been proved feasible with respect to the final state constraint according to the definition of Section 7.1.2.3. Then, all the points of P_{feas} are guaranteed to satisfy the final state constraint by the fundamental theorem of interval analysis. Since no other constraint exists, any point of P_{feas} can be used to update $\overline{J}$. Just like in an unconstrained problem, evaluating the performance index at the midpoint of P_{feas}, $J\left(m\left(P_{feas}\right)\right) = \left[\underline{J}_{feas,mid}, \overline{J}_{feas,mid}\right]$, instead of over the entire box P_{feas}, $J\left(P_{feas}\right) = \left[\underline{J}_{feas}, \overline{J}_{feas}\right]$, should result in a smaller upper bound on the minimum, i.e.:

$$\overline{J}_{feas,mid} \le \overline{J}_{feas} \qquad (7\text{-}21)$$

The only inconvenience is that the equations of motion need to be integrated twice: a first time using P_i to prove feasibility/infeasibility and second time using $m\left(P_{feas}\right)$ to update $\overline{J}$. If $\overline{J}$ was calculated using the entire boxes P_{feas}, then the equations of motion would only have to be integrated once.

Integrating the equations of motion is by the far the most computationally demanding operation of the optimization algorithm. Therefore, it might not compensate to use the midpoints of P_{feas} to update $\overline{J}$. In the case at hand, it does, as it will be demonstrated in Section 7.1.2.9, but it is doubtful that this conclusion can be generalized to other problems.

7.1.2.6 Stopping Criteria

The optimization algorithm is interrupted when the stopping criteria described in Chapter 5, namely:

1) $$\max_i\left(\sup\left(P_i^j\right)\right) - \min_i\left(\inf\left(P_i^j\right)\right) \le \varepsilon_{P_j} \quad , j = 1,\ldots,l \qquad (5\text{-}49)$$

2) $$\min\left(\overline{J}, \overline{\mathrm{J}}\right) - \underline{\mathrm{J}} \le \varepsilon_J \qquad (5\text{-}50)$$

are simultaneously met.

Since ε_{μ_1} and ε_{μ_2} were always chosen equal to infinite during the optimizations, (5-50) was the only stopping criterion effectively used.

Since the minimum value of $\min\left(\overline{J},\overline{\mathrm{J}}\right)-\underline{\mathrm{J}}$ is limited by dependency and by the wrapping effect, as demonstrated in **Figure 7.17**, some care was taken not to choose ε_J too small.

The stopping criteria (5-49) and (5-50) are based on the width of the guaranteed bounds on P^* and J^*, respectively. Another stopping criterion could be suggested based on the width of the guaranteed bounds on the final state constraint, i.e., based on the width of the guaranteed bounds on $\chi_r^*\left(E_{ALI}\right)$, $x^*\left(E_{ALI}\right)$ and $y^*\left(E_{ALI}\right)$. Using the nomenclature of Section 7.1.2.3, this stopping criterion would be:

$$\max_i\left(\overline{f_j\left(P_i\right)}\right)-\min_i\left(\underline{f_j\left(P_i\right)}\right)\le\varepsilon_{f_j}\quad ,j=1,...,r \tag{7-22}$$

where $\varepsilon_{f_j}>0$ (r parameters) would be given by the user.

In some problems, it might be easier to define ε_{f_j} than ε_J. For example, in the case at hand, the physical meaning of ε_{f_j} is easier to understand than the one of ε_J.

(7-22) still needs to be implemented in the optimization algorithm and duly tested.

7.1.2.7 Contiguousness Verification

At the end of the optimization loop, P^* is guaranteed to be inside a (multidimensional) box with an volume smaller than $\varepsilon_{P_1}\times\varepsilon_{P_2}\times...\times\varepsilon_{P_l}$, i.e., the so-called hull of the solution set.

However, if the hull of the solution set contains (multidimensional) gaps, it might be possible to calculate sharper bounds on P^*. In order to do that, after the optimization loop, the remaining boxes are checked for contiguousness, i.e., the (multidimensional) region defined by the remaining boxes is checked for (multidimensional) gaps. The contiguousness verification routine then returns the hull of any contiguous cluster of boxes contained inside the solution set.

The developed contiguousness verification routine can handle an arbitrary high number of dimensions.

To shed some light on the above, a three-dimensional example is given in **Figure 7.26**. Since P^* must be contained inside one of the remaining boxes, P^* must also be contained inside one of the clusters, which are a smaller enclosure of P^* than the entire hull of the solution set.

7.1.2.8 Solution Point Selection

As mentioned in Section 5.2.5, interval solutions per se are not very useful. For example, in the case at hand, real (and not interval) values of μ_1 and μ_2 are required to feed the HORUS's onboard flight computer.

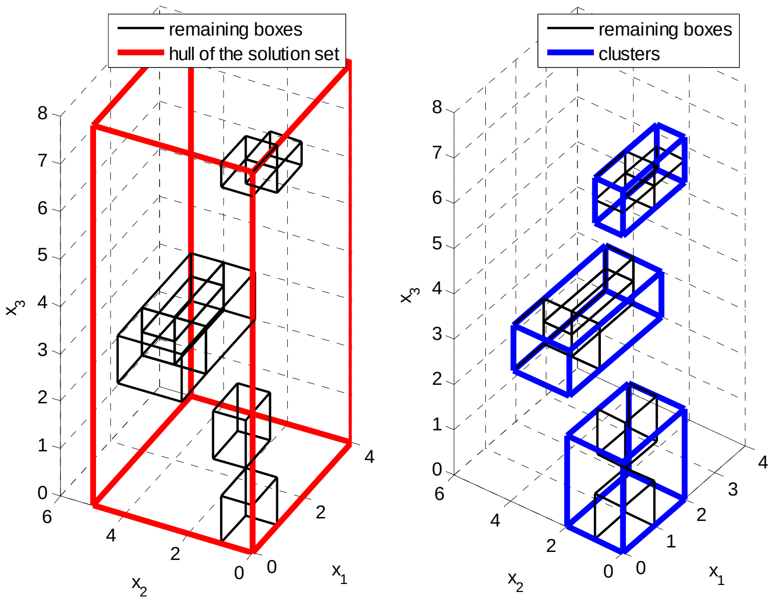

Figure 7.26: *Contiguousness verification.*

Therefore, the following method for selecting a solution point p from the guaranteed bounds on P^* was devised:

1) if $\bar{J}$ has been updated at least once during the optimization loop, the midpoint of the box last used to update $\bar{J}$ is selected. This guarantees that (5-48) is satisfied.

2) otherwise, the midpoint of the first (and possibly only) remaining box is taken as the solution point p. This guarantees that (5-47) is satisfied.

Note that the last box used to update $\bar{J}$ will always be one of the remaining boxes at the end of the optimization loop illustrated in **Figure 7.24**. The same cannot be said about the optimization loops illustrated in **Figure 4.1** and **Figure 5.1**.

7.1.2.9 Results

For the input values given in **Table 7.2**, the optimization results are summarized in **Table 7.3**.

As can be seen, the values of J^*, μ_1^*, μ_2^*, $\chi_r^*(E_{ALI})$, $x^*(E_{ALI})$ and $y^*(E_{ALI})$ given in Section 7.1.2.4 are correctly bounded by J^I, μ_1^I, μ_2^I, $\chi_r^I(E_{ALI})$, $x^I(E_{ALI})$ and $y^I(E_{ALI})$, respectively.

The remaining boxes form only one contiguous cluster. The hull of this cluster is equal to the hull of the solution set P^I .

Table 7.3: *Optimization results.*

Result	Symbol		Value	Width	Units
Bounds on the minimum	J^I		$[0.000, 0.274]$	2.7e-1	-
Bounds on the solution point	P^I	μ_1^I	$[7.50, 13.13]$	5.6	deg
		μ_2^I	$[-31.03, -29.88]$	1.1	deg
Bounds on the final state	X^I	$\chi_r^I(E_{ALI})$	$[-3.74, 3.57]$	7.3	deg
		$x^I(E_{ALI})$	$[-1.39, 0.64]$	2.0	km
		$y^I(E_{ALI})$	$[-1.33, 1.26]$	2.6	km
Optimization loop duration	Δt_{CPU}		217 (≈4m)	N/A	s
Number of remaining boxes	$N_{remaining}$		76	N/A	-
Number of clusters	$N_{clusters}$		1	N/A	-
Selected solution point	p	μ_1^p	≈10.31	N/A	deg
		μ_2^p	≈-30.48	N/A	deg

Several boxes were proved feasible during the course of the optimization loop. As a result, the selected solution point is the midpoint of the box last used to update $\overline{J}$, in accordance with rules expressed in Section 7.1.2.8.

The absolute error between μ_1^p and the global minimum μ_1^* is 0.1 degrees and the absolute error between μ_2^p and the global minimum μ_2^* is 0.02 degrees. Both errors are smaller than the widths of the guaranteed bounds on the solution point, as expected.

In **Figure 7.27**, with the help of the 1-DOF SIMULINK model shown in **Figure 7.15**, the lateral profile produced by the selected flight-path bank angles (μ_1^p and μ_2^p) is verified against the optimal trajectory produced by μ_1^* and μ_2^*.

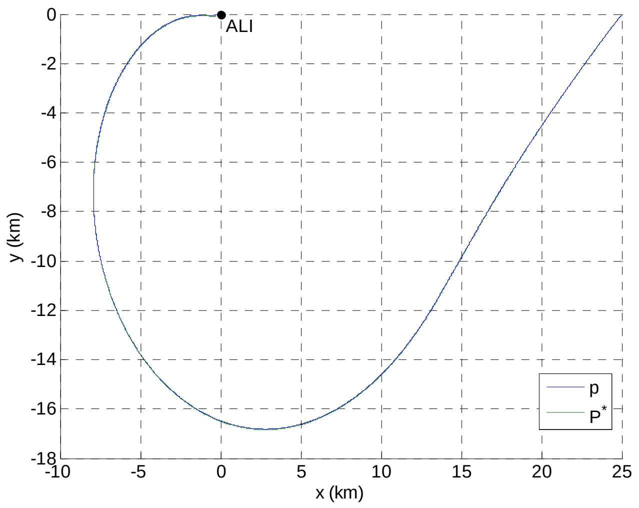

Figure 7.27: *Verification of the optimization results.*

The two trajectories are hardly distinguishable. The final lateral state associated with p is given by:

$$\chi_r^p\left(E_{ALI}\right) \approx -0.45^{\circ} \tag{7-23}$$

$$x^p\left(E_{ALI}\right) \approx 347m \tag{7-24}$$

$$y^p\left(E_{ALI}\right) \approx -47m \tag{7-25}$$

Compared with the optimal final lateral state $\chi_r^*\left(E_{ALI}\right)$, $x^*\left(E_{ALI}\right)$ and $y^*\left(E_{ALI}\right)$, the absolute errors are 0.04 degrees, 13 meters and 33 meters, respectively. These errors are smaller than the widths of the guaranteed bounds on the final state, as expected.

Evaluating the performance index at p yields:

$$J^p \approx 0.14491 \tag{7-26}$$

The absolute error with respect to J^* is 0.00073, which is smaller than the width of the guaranteed bounds on the minimum, as expected.

In **Figure 7.28** and **Figure 7.29**, the 1-DOF and the 3-DOF trajectories generated by p are compared as a validation test. The 3-DOF trajectory is calculated using the SIMULINK model of **Figure 7.2**.

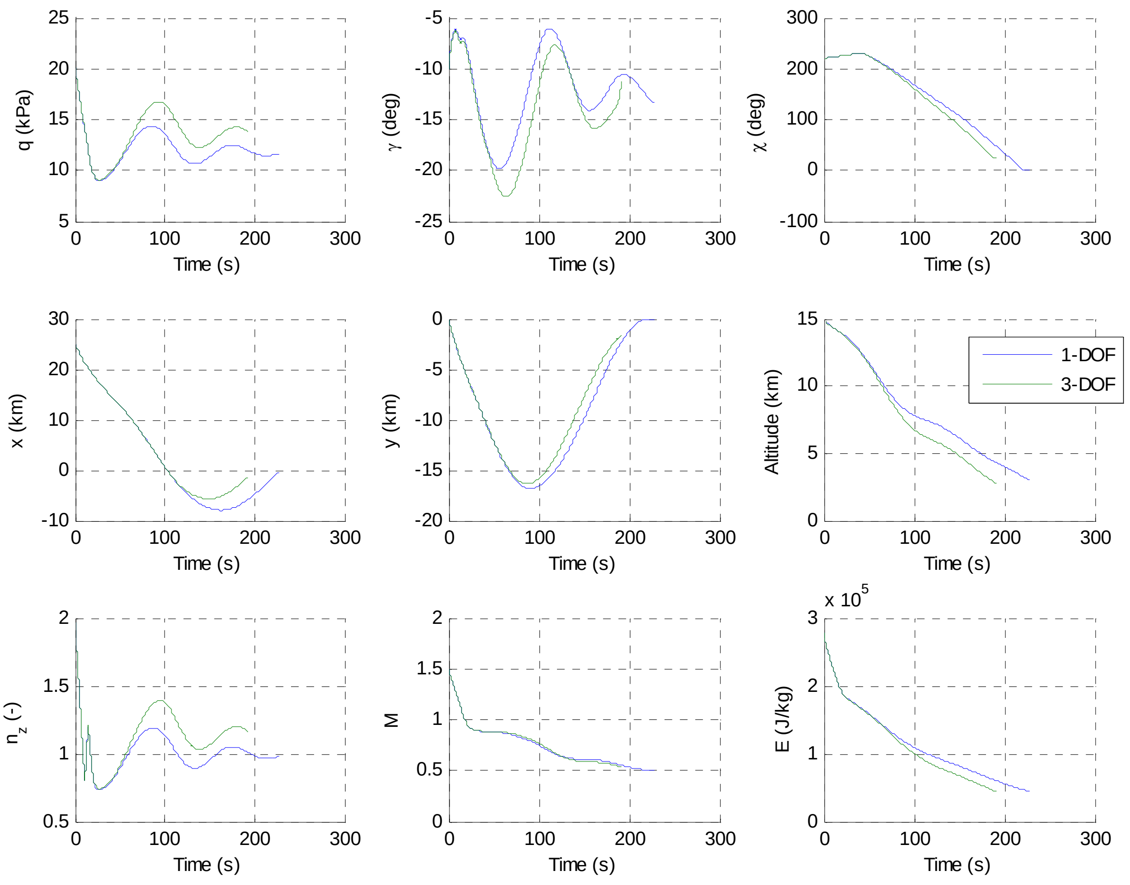

Figure 7.28: *Validation of the optimization results against the 3-DOF model: states and outputs.*

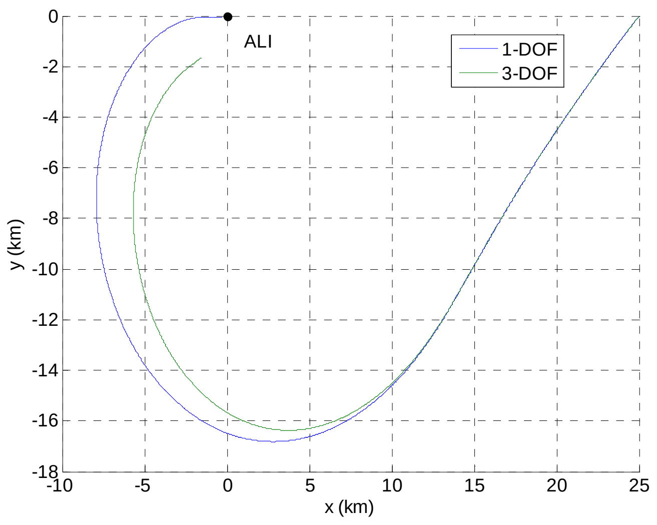

Figure 7.29: *Validation of the optimization results against the 3-DOF model: lateral profile.*

The two trajectories are visibly different. Inserting p into the 3-DOF model yields the following final lateral state:

$$\chi_r^{3-DOF}\left(E_{ALI}\right) \approx 25^{\circ} \tag{7-27}$$

$$x^{3-DOF}\left(E_{ALI}\right) \approx -1.54km \tag{7-28}$$

$$y^{3-DOF}\left(E_{ALI}\right) \approx -1.65km \tag{7-29}$$

which is well outside the ALI gate specified by (7-11)-(7-13). This means that the calculated flight-path bank angles μ_1^p and μ_2^p cannot be used directly to fly the RLV from TEP to ALI.

While it is true that the 1-DOF trajectory illustrated in **Figure 7.29** does satisfy the specified final state constraints (both longitudinal and lateral), it is not guaranteed that an automatic flight control system will be able to follow it without violating the vehicle's path constraints. What one would like to have is directly the flight-path bank angles μ_1^p and μ_2^p the vehicle must follow to fly from TEP to ALI without violating the vehicle's path constraints. For that, the complete 3-DOF equations of motion need to be solved through an interval ODE solver. This topic is addressed in Section 7.2.

In **Table 7.4**, the influence of N_{split} on Δt_{CPU} is analysed.

Table 7.4: *Influence of* N_{split} *on* Δt_{CPU} .

N_{split}	Δt_{CPU}
$[2 \quad 2]^T$	397s (≈7m)
$[4 \quad 2]^T$	Maximum number of boxes exceeded[14] before convergence
$[2 \quad 4]^T$	217s (≈4m)

[14] *If the number of boxes becomes too large, the optimization loop will run out of memory.*

Since the shape of the trajectory is more sensible to variations of μ_2 than of μ_1, as demonstrated in **Figure 7.9**, it makes sense that dividing μ_2 in more boxes than μ_1 leads to a faster convergence of the optimization loop.

In **Table 7.5**, the two methods described in Section 7.1.2.5 for updating $\bar{J}$ are compared in terms of Δt_{CPU} for $N_{split} = \begin{bmatrix} 2 & 2 \end{bmatrix}^T$.

Table 7.5: *Influence of the method used to update* $\bar{J}$ *on* Δt_{CPU} *.*

$\bar{J}$ **updated using:**	Δt_{CPU}
P_{feas}	439s
$m\left(P_{feas}\right)$	397s

As shown, updating $\bar{J}$ with the midpoints of the feasible boxes instead of with the entire feasible boxes leads to a faster convergence of the optimization loop. Like expressed in Section 7.1.2.5, it is doubtful that this conclusion can be generalized to other problems.

7.2 3-DOF Methodology

In the 3-DOF methodology, the longitudinal and lateral motions of the vehicle are assumed fully coupled. Thus, it is no longer possible to optimize each motion separately like in Section 7.1.

The reason why the longitudinal motion is calculated before the lateral motion in Section 7.1 is to allow the relative heading angle to be numerically integrated from (6-23). Without knowing the longitudinal motion beforehand, an interval IVP-ODE solver is required to calculate the lateral (and the longitudinal) motion.

7.2.1 Chu's method

In [Chu, 2007], the equations of motion are propagated not by an guaranteed interval IVP-ODE solver but by a Runge-Kutta method based on real arithmetics[15]. The algorithm itself is not explained in [Chu, 2007], but the ideas behind it were transmitted to the author directly by Weiwei Chu via personal communication [Chu, 2008].

Chu's method starts by assuming that all the states of the equations of motion are monotonic functions. Then, the lower and upper bounds of the initial state are propagated independently using an IVP-ODE solver based on real arithmetics, like for example, a Runge-Kutta method. Without entering in too much detail, this brief description should already be sufficient for the reader to understand why Chu's method will never be able to perform rigorous global optimization.

First of all, it is wrong to assume that all the states of the equations of motion are monotonic functions since in general this is not the case. **Figure 7.9** is given as an example. Secondly, even if they were, an off the shelf IVP-ODE solver based on real arithmetics cannot yield guaranteed bounds on the time histories of the states, e.g., because the numerical errors are not accounted for. Therefore, the time histories enclosures produced by Chu's method are just numerical approximations of the true enclosures and are not guarantee bounds on the latter. Without the assurance of guarantee bounds, the fundamental theorem of interval analysis cannot be applied and global optimization cannot be performed.

7.2.2 Interval ODE solvers

After identifying the need for an interval ODE solver, several options were investigated.

In a first phase, a first-order ($p=1$) and a second-order ($p=2$) interval ODE solver were built based on the theory developed in Chapter 10 of [Moore, 1966]: the so-called traditional method. Integrating the fully coupled 3-DOF equations of motion (6-21)-(6-26) with these solvers revealed itself to be impossible as the guaranteed bounds on the time histories of the states would explode before reaching the final condition, even for relatively small time steps. For details about the traditional method, the reader is referred to [Moore, 1966] and to [Nedialkov, 2006b]. As a side note, even VNODE-LP, a high-order ($3\leq p\leq 50$) interval ODE solver based on the same theory, explodes before reaching the final condition, as will be shown in Section 7.2.3.

[15] *At a certain point in [Chu, 2007], one can even read that dynamic trajectory optimization can be performed without an ODE solver of any kind.*

In a second phase, developing from scratch a high-order interval ODE solver based either on the traditional method or on Taylor models was considered. However, with approximately one month left for coding, implementing and testing such a complex algorithm in the available time slot was deemed unrealistic. For example, interval automatic differentiation based on operator overloading would need to be implemented. It is the author's belief that developing such a solver would require at least the time normally allocated to a Master of Science thesis, just by itself.

Taking the above into consideration, it was decided that an existing high-order interval ODE solver would be used to integrate the equations of motion, to quickly test the capabilities of the existing interval theory for solving IVP-ODE problems.

A survey about high-order interval ODE solvers was realized and several tools identified. **Table 7.6** summarizes the gathered information. Most of it was retrieved from [Nedialkov, 2006b].

Also given in [Nedialkov, 2006b] is a list of past applications of interval ODE solvers. Of these, the following ones are emphasized:

1) rigorous computation of asteroid orbits [Berz *et al.*, 2001];

2) rigorous multibody simulations [Auer *et al.*, 2004];

3) global optimization for parameter estimation in chemical engineering [Lin and Stadtherr, 2006a].

The first two applications were selected due to their relevance to aerospace engineering. The third one was selected due to its relevance to the present work.

In [Lin and Stadtherr, 2006a], a dynamic global optimization problem related with chemical engineering is solved using interval analysis. The theory described there is completely applicable to the current trajectory optimization problem. Like in this thesis, they apply a branch and bound method. They also delete boxes that do not satisfy the equality constraints. However, they introduce a novel theory based on Taylor models capable of shrinking the boxes and of proving feasibility with respect to equality and/or inequality constraints. This theory is very similar to the HC method described in previous chapters but, unlike HC, it does not require the state equations to be analytically solved. The same theory is used in [Lin *et al.*, 2008] to enclose all the solutions of Two-Point Boundary Value Problems (TPBVP). VSPODE is used in both papers to integrate the dynamic equations.

It is the author's conviction that VSPODE is the most suitable high-order interval ODE solver for dynamic global optimization currently available. First of all, it uses Taylor models to integrate the dynamic equations, which according to [Nedialkov, 2006b], are more effective than traditional methods at producing tight enclosures when the interval parameters are wide (which is the case) and the integration intervals are long. As a setback, they can only handle 5 to 10 equations, typically, while traditional methods can deal with a few hundred equations. Since the 3-DOF equations of motion are constituted only by 6 equations, this should not be a problem. Secondly, the HC-type method described in [Lin and Stadtherr, 2006a] has so far only been applied with VSPODE, although, it should work with other interval ODE solvers based on Taylor models.

While VSPODE may seem the obvious choice, VNODE-LP was selected to integrate the 3-DOF equations of motion due the following reasons:

1) Since this thesis was done in collaboration with the European Space Agency (ESA), more particularly, with the TEC-ECM section of ESA/ESTEC (European Space Research and Technology Centre), it was given preference to interval ODE solvers developed in ESA member states. VNODE-LP was developed in Canada, which is an ESA member state.

2) Tools coded in C++ or with a C++ interface were given preference. VNODE-LP is coded is C++.

3) Only available tools were considered. VNODE-LP is free to download.

Table 7.6: *High-order interval ODE solvers.*

Adjacent Theory	Solver	Year	Language	Developer	Availability	Reference
Traditional methods	AWA	1988	Pascal-XSC	R. J. Lohner (Universität Karlsruhe, Germany)	Free	[Lohner, 1988]
	ADIODES	1997	C++	Ole Stauning (Technical University of Denmark, Denmark)	Unavailable	[Stauning, 1997]
	VNODE	2001	C++	N. S. Nedialkov and K. R. Jackson (McMaster University, Canada)	Free	[Nedialkov and Jackson, 2002]
	VODESIA	2003	Fortran-XSC	S. Dietich (University of Karlsruhe, Germany)	Unavailable	[Dietich, 2003]
	VNODE-LP	2006	C++	N. S. Nedialkov (McMaster University, Canada)	Free	[Nedialkov, 2006a]
Taylor models	COSY-VI	2006	Fortran C++ interface	Center for Dynamical Systems, Department of Physics and Astronomy, Michigan State University, USA	Free for non-commercial users	[Berz and Makino, 2006]
Mixture of traditional methods with Taylor models	VSPODE	2005	C++	Y. Lin and M. A. Stadtherr (University of Notre Dame, USA)	By request from the authors	[Lin and Stadtherr, 2006b]
Based on the Jacobian of the state equations	ValEncIA-IVP	2005	C++	E. Auer (University of Duisburg-Essen, Germany) and A. Rauh (University of Rostock, Germany)	By request from the authors	[Rauh and Auer, 2008]

4) VNODE and COSY-IV are the most popular interval ODE solvers currently available, according to [Nedialkov, 2006b]. For example, VNODE is used in [Lin and Stadtherr, 2006b] and in [Rauh and Auer, 2008] to validate other interval ODE solvers.

5) VNODE-LP is an updated version of VNODE.

In Section 7.2.3, it will be shown that VNODE-LP is not suitable for solving the 3-DOF TAEM trajectory optimization problem. It is the author's belief that future attempts to solve this problem should focus on VSPODE.

7.2.3 VNODE-LP

VNODE-LP was developed by Prof. Nedialkov from the McMaster University located in Canada. It is the successor of VNODE, a well-known interval ODE solver referred across literature. For documentation, the reader is referred to [Nedialkov, 2006a].

The following free packages must be installed before VNODE-LP:

1) Interval arithmetics C++ package: FILIB++ [Lerch *et al.*, 2008] or PROFIL/BIAS [Rump, 2008].

2) Linear algebra packages: LAPACK [LAPACK, 2008] and BLAS[16] [ATLAS, 2008].

PROFIL/BIAS was selected over FILIB++ since, to date, the only successful installation of VNODE-LP on Windows has been with PROFIL/BIAS and Cygwin [Cygwin, 2008], a Linux-like environment for Windows. For a step-by-step description of the installation process of VNODE-LP, the reader is referred to [Nedialkov, 2006a].

Additionally to the abovementioned packages, VNODE-LP already comes with FADBAD++, a free C++ automatic differentiation package [Stauning and Bendtsen, 2008].

At each step, VNODE-LP automatically calculates the integration time step. The following three parameters give the user a gross control over it:

1) the order of the integration, p;

2) the absolute error tolerance;

3) the relative error tolerance.

The effect of each parameter is analysed in [Nedialkov, 2006a].

The flowchart of **Figure 7.30** describes how VNODE-LP is used to integrate the 3-DOF equations of motion. First, the trajectory is propagated using a SIMULINK model similar to the one of **Figure 7.2** and configured in the same way. Then, a C++ executable file is called and the equations of motion are integrated with VNODE-LP. The two solutions are then compared within MATLAB.

Especial care must be taken when reading and writing intervals into text files: outward rounding must always be assured. For example, the bounds of an interval are usually written into a text file with less precision than on the computer's memory. Therefore, when copying an interval from the computer's memory into a text file, the bounds have to be outwardly rounded. Otherwise, the mathematical rigor of interval analysis is lost. In the same way, when reading an interval, one of the bounds might not be exactly representable by the computer. In that case, outward rounding must also be enforced. Luckily, both INTLAB and PROFIL/BIAS (the interval arithmetics package used by VNODE-LP) provide functions that allow intervals to be correctly read and written into text files.

Table 7.7 summarizes all the main inputs of the 3-DOF methodology script and the values used during the subsequent simulations. The exact structure of the input file and an example are given in Appendix B.

Unlike in Section 7.1, the time histories of C_L and C_D are no longer calculated from lookup tables but are assumed constant during each TAEM sub-phase. This has to do with one of the limitations of VNODE-LP and is further discussed in Section 7.2.3.1.

[16] *The non-optimized version of BLAS that comes with LAPACK was used. Optimized versions of BLAS are sold by several hardware brands.*

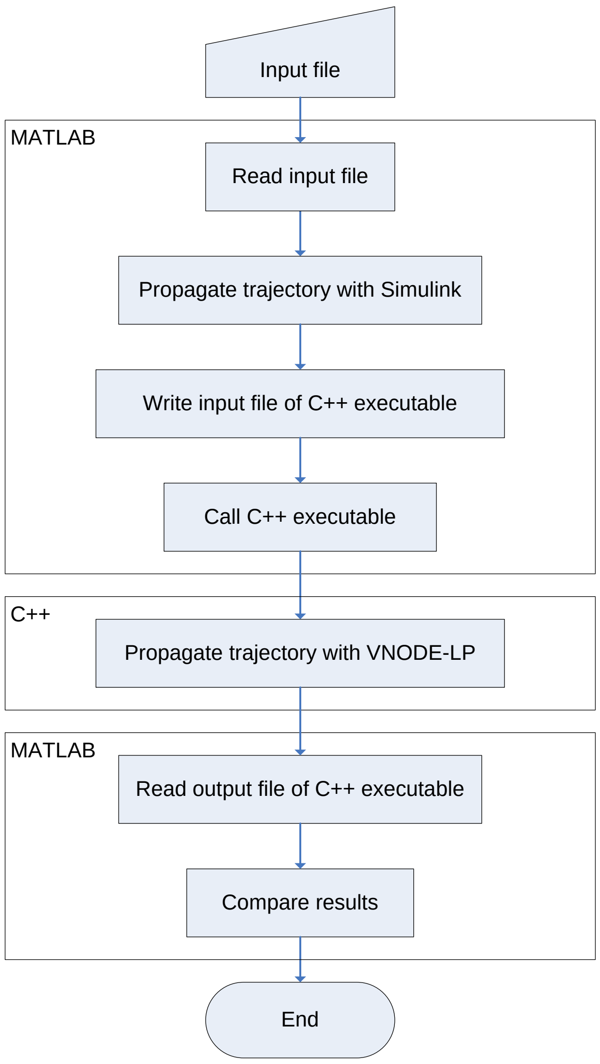
Input file
MATLAB
Read input file
Propagate trajectory with Simulink
Write input file of C++ executable
Call C++ executable
C++
Propagate trajectory with VNODE-LP
MATLAB
Read output file of C++ executable
Compare results
End

Figure 7.30: *Integration of the 3-DOF equations of motion with VNODE-LP.*

Table 7.7: *Main Inputs of the 3-DOF Methodology Script.*

#	Input	Symbol	Format	Value	Units
1	Flight-path bank angle (interval) during sub-phase 1	μ_1	*interval*	$[60, 60]$	deg
2	Flight-path bank angle (interval) during sub-phase 3	μ_2	*interval*	$[-60, -60]$	deg
3	Reference area of the vehicle	S	*float*	110	m^2
4	Re-entry mass of the vehicle	m	*float*	26,029	kg
5	Mach number at TEP	M_{TEP}	*float*	1.5	-
6	Mach number at ALI	M_{ALI}	*float*	0.5	-
7	Altitude at TEP	h_{TEP}	*float*	15,000	m
8	Altitude at ALI	h_{ALI}	*float*	3,000	m

#	Input	Symbol	Format	Value	Units
9	Number of integration steps used by SIMULINK	N_{int}	*int*	3,000	-
10	Flight-path angle at TEP	γ_{TEP}	*float*	-10	deg
11	Relative heading angle at TEP	χ_{TEP}	*float*	0	deg
12	x-position at TEP	x_{TEP}	*float*	-65,000	m
13	y-position at TEP	y_{TEP}	*float*	34,000	m
14	Sub-phases relative lengths	$\Delta E_{rel,sub-phases}$	*float*	$\begin{bmatrix}0.44 & 0.05 & 0.5 & 0.01\end{bmatrix}$	- (%)
15	Aerodynamic lift coefficient per sub-phase	C_L	*float*	$\begin{bmatrix}0.3 & 0.2 & 0.2 & 0.2\end{bmatrix}$	-
16	Aerodynamic drag coefficient per sub-phase	C_D	*float*	$\begin{bmatrix}0.14 & 0.14 & 0.11 & 0.05\end{bmatrix}$	-
17	Integration order used by VNODE-LP	p	*int*	7	-

7.2.3.1 Limitations of VNODE-LP

VNODE-LP suffers from the following limitations:

1) The biggest and foremost problem of VNODE-LP is that it cannot handle wide interval initial conditions/parameters. Moreover, it cannot handle long integration time intervals.

VNODE-LP was designed to produce tight bounds on the solution of IVP-ODE problems with point initial conditions/parameters or with interval initial conditions/parameters with a sufficiently small width, over not very long time intervals. This limitation is common to all interval ODE solvers based on traditional methods. If these conditions are not satisfied, the manual of VNODE-LP suggests using COSY, which is based on Taylor models.

Figure 7.31 and **Figure 7.32** illustrate for the problem at hand, which are the maximum widths of μ_1 and μ_2 that VNODE-LP can handle. In the first figure, μ_1 and μ_2 are defined as point intervals: $\mu_1 = -\mu_2 = 60^{\circ}$. As can be seen, VNODE-LP does not have any problem integrating the equations of motion in this case. The numerical results produced by SIMULINK and by VNODE-LP for $\mu_1 = -\mu_2 = 60^{\circ}$ are presented in **Table 7.8**. The intention is not to directly compare the computational accuracy and time of the two algorithms (which is not a trivial task) but to show that both algorithms yield results of the some order of magnitude.

The interval final state calculated by VNODE-LP is guaranteed to contain the true final state (assuming the algorithm is correct). Moreover, the width of the bounds on the final state can be controlled by, e.g., increasing the integration order. On the other hand, the crisp final state calculated by SIMULINK is only an approximation of the true final state and its accuracy is unknown. While, in theory, decreasing the relative tolerance of the ode45 solver used by SIMULINK should produce more accurate results, the truth is, a certain accuracy can never be guaranteed. This is, according to the author, the biggest advantage of using an interval ODE solver.

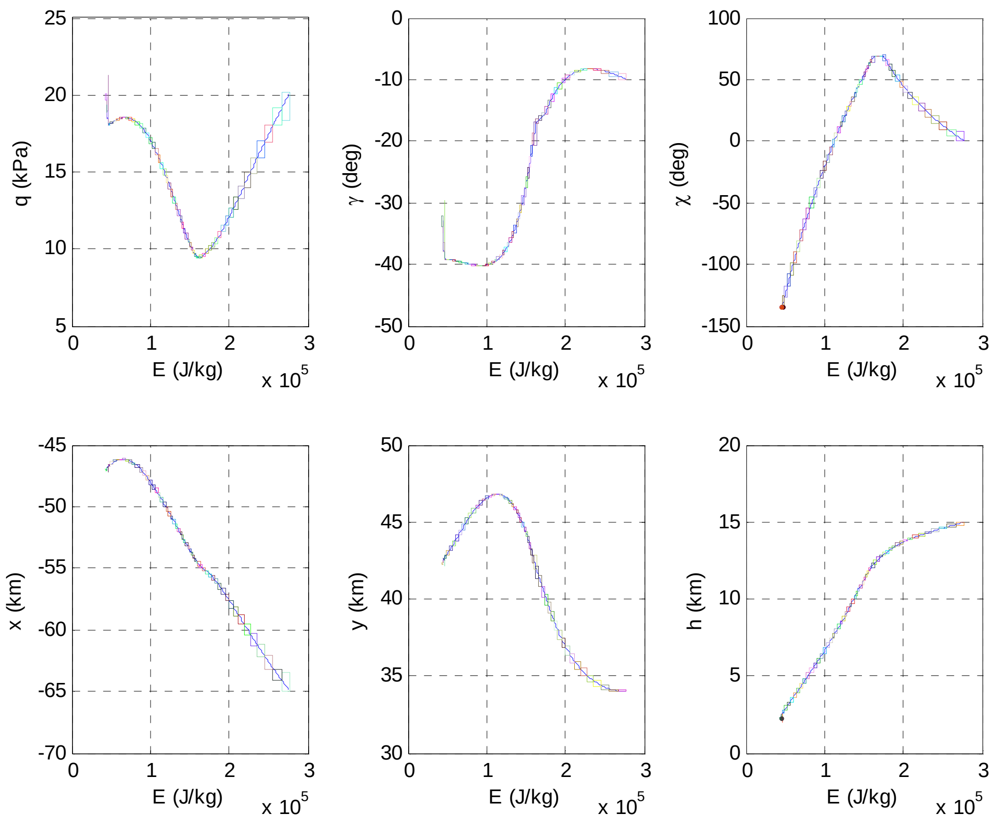

Figure 7.31: *Bounds calculated by VNODE-LP when point interval parameters are used.*

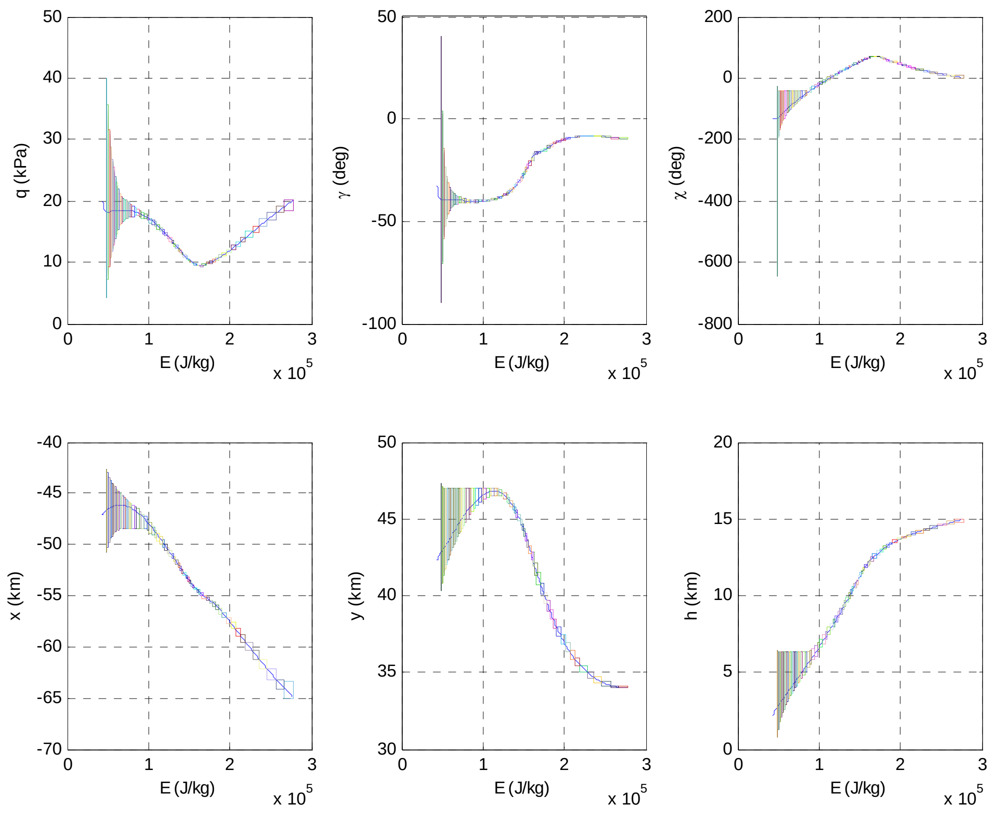

Figure 7.32: *Bounds calculated by VNODE-LP when non-point interval parameters are used.*

Table 7.8: *Final state computed by SIMULINK and by VNODE-LP.*

			Simulink	VNODE-LP	
Result	**Symbol**	**Unit**	**Value**	**Value**	**Width**
Final state	q_{ALI}	kPa	19.8525	$[19.852792, 19.852795]$	2e-6
	γ_{ALI}	deg	-32.86	$[-32.859739, -32.859734]$	4e-4
	χ_{ALI}	deg	-134.57	$[-134.5540, -134.5538]$	1e-4
	x_{ALI}	km	-47.06	$[-47.05986, -47.05983]$	2e-5
	y_{ALI}	km	42.323	$[42.32294, 42.32296]$	2e-5
	h_{ALI}	km	2.2281	$[2.228793, 2.228801]$	6e-6
CPU time	Δt_{CPU}	s	0.04	0.3	N/A

In **Figure 7.32**, μ_1 and μ_2 are defined as $[59.9, 60.0]$ and $[-60.0, -59.9]$ degrees, respectively. As can be seen, the bounds on the states explode before E_{ALI} is reached. Notice the decreasing size of the integration time steps as the integrator explodes. Attempts to overcome this problem by changing the integration order and the error tolerances of VNODE-LP were not successful.

In [Lin and Stadtherr, 2006b], a possible solution is suggested: time-invariant interval parameters (like μ_1 and μ_2) should be treated as additional state variables with zero first-order derivatives. They base their suggestion on the fact that VNODE-LP can handle better interval initial states than parameters. In the case at hand, the visible effects of defining μ_1 and μ_2 as states were that the integrator would explode closer to E_{ALI} but still short of the final condition.

Through personal communication, Prof. Nedialkov [Nedialkov, 2008], the creator of VNODE-LP, has confirmed that VNODE-LP was not designed to deal with wide interval parameters. According to him, the only solution is to split them into smaller subintervals and to integrate each subinterval independently. However, in case at hand, this would be unaffordable in terms of computational time. For example, if integrating a subbox of 0.1x0.1 degrees takes 0.3 seconds (like in **Table 7.8**), covering an initial search box equal to $\mu_1 = \mu_2 = [-60, 60]$ would take approximately 5 days.

As of this time, no solution has been found to overcome this limitation of VNODE-LP.

2) Another problem of VNODE-LP is that it does not allow C_L and C_D to be directly calculated from lookup tables as a function of α and M.

In order to integrate the 3-DOF equations of motion, VNODE-LP requires the right side of (6-21)-(6-26) to be at least p-times differentiable (where p can be as big as 50) during each integration time step and to be completely defined at the time of compilation. Therefore, the only way to define C_L and C_D by branches is to allocate a single branch to each time step. Since the user cannot directly control the size of the time steps and C_L and C_D are a function of M, which depends on q (one of the states of the system), there is no way of allocating a single branch to each time step.

One possible solution would be to approximate C_L and C_D by multidimensional splines calculated from the lookup table points. The entire integration time interval would have to be covered by a single multidimensional spline.

Figure 7.31 and **Figure 7.32** were created assuming C_L and C_D constant during each TAEM sub-phase. The same assumption is done in [Vernis and Ferreira, 2006]. Since the integration stopping condition is defined in terms of E, just like the TAEM sub-phases, it is possible to attribute a constant C_L and C_D to each TAEM sub-phase.

3) VNODE-LP does not allow all the boxes to be integrated at the same time, i.e., in parallel. The boxes have to be integrated sequentially. This is partially due to the fact that VNODE-LP does not exploit PROFIL's matrix and vector operations, which are optimized in terms of minimizing rounding mode switches. If it did, the running time would have been reduced [Nedialkov, 2006b]. This decision was taken in order to allow the user to choose between PROFIL/BIAS and FILIB++.

Chapter 8 | Conclusions

The main conclusions and accomplishments of this Master of Science thesis will not be synthesized.

In Chapter 2, the reader was introduced to the unique characteristics of interval analysis and to the reasons why it is currently a hot topic in the scientific world. Guaranteed global optimization, simplified uncertainty and sensibility analyses and the nonexistence of undefined forms such as $0/0$, $\infty-\infty$, $0\times\infty$ and ∞/∞, make it a very promising technique for the future. As computers become faster and special hardware chips for computing with intervals become available, the slowness of interval analysis becomes less of an issue.

In Chapter 3, outward rounding was shown to be essential for the mathematical rigor of interval analysis. In other words, if a single roundoff error is not taken into account, the mathematical rigor of interval analysis is lost and the fundamental theorem of interval analysis is no longer valid.

The accuracy/sharpness of interval arithmetics is influenced by the dependency problem and the wrapping effect. Dependency can be reduced by rewriting expressions in a different form. The wrapping effect is harder to control and the operation most susceptible to it is integration. Taylor models are normally used to suppress it.

In Chapter 4, a novel static global optimization algorithm based on interval analysis was developed in MATLAB. Further developments of the code will now be expedited by MATLAB's simplicity and portability. The algorithm exploits INTLAB's vector and matrix operations in order to reduce the computational time. Taking advantage of the fact that all the boxes are processed at the same time every iteration, new stopping criteria were conceived that guarantee that, at termination, the widths of the bounds on the global minimum and minimiser are below certain user-specified thresholds.

The interval results produced by the developed algorithm were proved to be rigorous even when the global minimum is surrounded by multiple local minima. At the same time, several flaws were identified in the static global optimization algorithm presented in [Chu, 2007]. For example, the incorrectness of the equality-constraint-feasibility-test was demonstrated. Chu's algorithm was proved to be unsuitable for calculating guaranteed bounds on the global minimum.

By means of several benchmark problems, it was shown that the developed algorithm needs less time than Chu's to reach the same accuracy/sharpness, mostly thanks to hull consistency. In addition to quickening the algorithm, hull consistency is also used to prove feasibility with respect to the equality constraints.

Static global optimization using interval analysis is already in a level of operational readiness where it is a viable and competitive alternative to traditional static optimization methods based on real arithmetics.

In Chapter 5, a novel dynamic global optimization algorithm based on interval analysis was developed. It shares many characteristics with its static older brother: both of them were programmed in MATLAB using INTLAB's vector and matrix operations; both of them are only terminated when the widths of the bounds on the global minimum and minimiser are below certain user-specified thresholds; both of them were proved to yield rigorous results; both of them were proved to be faster than the algorithms presented in [Chu, 2007].

The same mistakes found in the static version of Chu's global optimization algorithm were found in the dynamic version. Moreover, in [Chu, 2007], the time dependent state equations are integrated using a crisp ODE solver and not a guaranteed interval ODE solver.

In Chapter 5, the IVP-ODE problems were solved analytically and not numerically. Hull consistency was used to prove feasibility with respect to the equality constraints and to shrink the search region.

In Chapter 6, the 3-DOF equations of motion describing the movement of a reusable launch vehicle during the TAEM phase were deduced. Normalized energy was proved to be a better independent variable than time and altitude.

In Chapter 7, two different TAEM trajectory design methodologies were developed and tested: a 1-DOF and a 3-DOF methodology.

Two different tools were developed within the 1-DOF methodology: a terminal entry gate (TEG) determination tool and a TAEM optimal trajectory design tool. Interval analysis was only used in the determination/optimization of the lateral motion. The equations of motion were integrated numerically and not analytically like in Chapter 5. HC could not be used to prove feasibility or to shrink the search region. A new equality-constraint-feasibility-test was devised which does not depend on HC.

Due to the properties of interval analysis, the true TEG is mathematically guaranteed to be contained inside the calculated TEG, under the given assumptions and assuming the interval algorithm was encoded correctly.

The TAEM optimal trajectory design tool was able to find the global optimal of a cost function with several local minima.

A survey about high-order interval ODE solvers (the most essential component of a dynamic global optimization algorithm) was realized and several tools identified within the 3-DOF methodology. One of them, VNODE-LP, was used to integrate the 3-DOF equations of motion. It was shown that VNODE-LP and other interval ODE solvers based on traditional methods are unable to cope with the large interval parameters required by dynamic global optimization.

VSPODE, an interval ODE solver based on Taylor models, was identified as the most promising interval ODE solver currently available. Dynamic global optimization was already performed with it in [Lin and Stadtherr, 2006a]. The same paper also explains how the HC techniques developed in Chapter 5 based on analytical integration can be readapted to work with VSPODE and with Taylor-model-based numerical integration.

Chapter 9 | Recommendations

A collection of interesting future research topics in the field of static and dynamic global optimization using interval analysis is given below:

- First and foremost, check if VSPODE is indeed capable of integrating the 3-DOF equations of motion with non-point interval parameters. If so, use VSPODE to solve the 3-DOF TAEM trajectory optimization problem. At the same time, implement the HC techniques described in [Lin and Stadtherr, 2006a] based on Taylor models.
- Investigate other methods of parameterizing the control function, e.g., using neural networks.
- Implement extended interval arithmetics in INTLAB.
- Improve the static global optimization algorithm by solving the John conditions. More information about the John conditions can be found in [Hansen and Walster, 2004].
- Check every box for the existence of gaps and investigate how that can be used to optimize the algorithms. Some suggestions are given in [Hansen and Walster, 2004].
- Investigate if or when processing all the boxes at the same time is better than processing them one by one.
- Investigate if interval analysis can be used to make indirect methods yield global minima.
- Investigate if the HC equation (5-47) can be used to shrink the boxes.
- As a suggestion for a new line of research, develop a global optimization algorithm using interval analysis for solving mixed-integer nonlinear programming problems.

Bibliography

[ATLAS, 2008] ATLAS, "Automatically Tuned Linear Algebra Software (ATLAS)", retrieved October 2008, from http://math-atlas.sourceforge.net/

[Auer *et al.*, 2004] Auer, E., Kecskeméthy, A., Tändl, M., and Traczinski, H., "Interval algorithms in modelling of multibody systems", *Numerical Software with Result Verification*, Vol. 2991 of LNCS, Springer-Verlag, pp. 132-159, 2004.

[Barton, 2002] Barton, G. H., "New Methodologies For Onboard Generation Of TAEM Trajectories For Autonomous RLVs", presented at the Core Technologies for Space Systems Conference, 2002.

[Barton and Tragesser, 1999] Barton, G. H. and Tragesser, S. G., "Autolanding Trajectory Design for the X-34", presented as paper AIAA-99-4161 at the AIAA Atmospheric Flight Mechanics Conference and Exhibit, Portland, Oregon, 9-11 August 1999.

[Berz and Makino, 2006] Berz, M. and Makino, K., "COSY INFINITY 9.0 - Programmer's Manual", Michigan State University, MSU Report MSUHEP 060803, August 2006. Retrieved from: http://bt.pa.msu.edu/index_cosy.htm

[Berz *et al.*, 2001] Berz, M., Makino, K., and Hoefkens, J., "Verified integration of dynamics in the solar system", *Nonlinear Analysis: Theory, Methods & Applications*, Vol. 47, pp. 179-190, 2001.

[Berz, 2007] Berz, Martin, "Verified ODE and DAE integration controlling the wrapping effect", retrieved November 2007, from http://pims.math.ca/science/2001/scicade/18/BerzMartin241.pdf

[Bryson Jr. and Ho, 1975] Bryson Jr., A. E. and Ho, Y.-C., *Applied Optimal Control, Optimization, Estimation and Control*, Revised Printing, Great Britain: Taylor & Francis Group, 1975.

[Burchett, 2004] Burchett, B. T., "Fuzzy Logic Trajectory Design and Guidance for Terminal Area Energy Management", *Journal of Spacecraft and Rockets*, Vol. 41, No. 3, May-June 2004.

[Câmara, 2003] Câmara, F. M. C., "Development and Design of the Terminal Area Energy Management Guidance for a Reusable Launch Vehicle", Master of Science Thesis, Control and Simulation Division, Faculty of Aerospace, Delft University of Technology, Delft, February 2003.

[Chartres *et al.*, 2005a] Chartres, J., Gräblin, M., and Schneider, G., "A New Method For Terminal Area Guidance For Future Reusable Launch Vehicles", presented as paper IAC-05-C1.8.05 at the 56th International Astronautical Congress, 2005a.

[Chartres *et al.*, 2005b] Chartres, J. T. A., Gräblin, M. H., and Schneider, G., "Optimisation of the Terminal Flight Phase for a Future Reusable Launch Vehicle", presented as paper AIAA 2005-6060 at the AIAA Guidance, Navigation, and Control Conference and Exhibit, San Francisco, 15-18 August 2005b.

[Chu, 2007] Chu, W., "Interval Analysis Applied to Re-Entry Flight Trajectory Optimization", Master of Science Thesis, Control and Simulation Division, Faculty of Aerospace, Delft University of Technology, Delft, July 2007.

[Chu, 2008] Chu, Weiwei (private communication), September 2008.

[Computer Science, 2007] Computer Science, University of Texas at El Paso, "Interval and Related Software", retrieved October 2007, from http://www.cs.utep.edu/interval-comp/intsoft.html

[Costa, 2003] Costa, R. R. da, "Studies for Terminal Area GNC of Reusable Launch Vehicles", *American Institute of Aeronautics and Astronautics*, Paper 2003-5438, 2003.

[Cygwin, 2008] Cygwin, "Cygwin", retrieved October 2008, from http://www.cygwin.com/

[Deb, 1999] Deb, K., "Multiobjective genetic algorithms: Problem difficulties and construction of test problem", *Journal of Evolutionary Computation 7(3)*, The MIT Press, pp. 205-230, 1999.

[Dietich, 2003] Dietich, S., "Adaptive verifizierte Lösung gewöhnlicher Differentialgleichungen", PhD thesis, University of Karlsruhe, Karlsruhe, Germany, February 2003.

[Erb, 2008] Erb, Sven, "GESOP Training Course" Presentation Slides, ESA/ESTEC, 20 June 2008.

[Filipe, 2008] Filipe, Nuno, "Static and Dynamic Global Optimization using Interval Analysis", Preliminary Master of Science Thesis, Control and Simulation Department, Delft University of Technology, Faculty of Aerospace Engineering, Delft, February 16, 2008.

[Hansen and Walster, 2004] Hansen, E. and Walster, G. W., *Global Optimization Using Interval Analysis*, Second Edition, Revised and Expanded, United States of America: Marcel Dekker, Inc., 2004.

[Hargreaves, 2002] Hargreaves, G. I., "Interval Analysis in MATLAB", Manchester Centre for Computational Mathematics, University of Manchester, Manchester, Numerical Analysis Reports, No. 416, December 2002.

[Jaulin *et al.*, 2001] Jaulin, L., Kieffer, M., Didrit, O., and Walter, E., *Applied Interval Analysis*, Great Britain: Springer, 2001.

[Kearfott *et al.*, 2007] Kearfott, R. B., Dawande, M., Du, K., and Hu, Ch., "INTLIB: A Portable FORTRAN 77 Interval Standard Function Library", retrieved November 2007, from interval.louisiana.edu/preprints/intlib_toms_algorithm.ps

[Kluever, 2007] Kluever, C. A., "Terminal Guidance for an Unpowered Reusable Launch Vehicle with Bank Constraints", *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 1, January-February 2007.

[LAPACK, 2008] LAPACK, "LAPACK - Linear Algebra PACKage", retrieved October 2008, from http://www.netlib.org/lapack/

[Lerch *et al.*, 2008] Lerch, M., Tischler, G., Gudenberg, J. Wolff von, Hofschuster, W., and Krämer, W., "FILIB++", retrieved October 2008, from http://www.math.uni-wuppertal.de/~xsc/software/filib.html

[Liberti, 2006] Liberti, L., *Introduction to global optimization*, Italy: DEI, Politecnico de Milano, February 2006.

[Lin *et al.*, 2008] Lin, Youdong, Enszer, Joshua A., and Stadtherr, Mark A., "Enclosing all solutions of two-point boundary value problems for ODEs", *Computers & Chemical Engineering*, Vol. 32, Issue 8, pp. 1714-1725, 22 August 2008.

[Lin and Stadtherr, 2006a] Lin, Youdong and Stadtherr, Mark A., "Deterministic Global Optimization for Parameter Estimation of Dynamic Systems", *Industrial & Engineering Chemistry Research*, Vol. 45, pp. 8438-8448, 2006a.

[Lin and Stadtherr, 2006b] Lin, Youdong and Stadtherr, Mark A., "Validated solution of initial value problems for ODEs with interval parameters", Department of Chemical and Biomolecular Engineering, University of Notre Dame, Notre Dame, IN 46556, USA, 2006b.

[Lohner, 1988] Lohner, R. J., "Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen", PhD thesis, 1988, http://www.math.uni-wuppertal.de/~xsc/xsc/pxsc_software.html#awa.

[Mayanna *et al.*, 2006] Mayanna, Anand, Grimm, Werner, and Well, Klaus H., "Adaptive Guidance for Terminal Area Energy Management (TAEM) of Reentry Vehicles", presented as paper AIAA 2006-6037 at the AIAA Guidance, Navigation, and Control Conference and Exhibit, Keystone, Colorado, 21-24 August 2006.

[MBB, 1988] MBB, Space Communication and Propulsion Systems Division, "Study on re-entry guidance and control - Final report", MBB, Munich, ESA report reference: ESA CR (P) 2652, 1988.

[Mooij, 1995] Mooij, E., "The HORUS-2B reference vehicle", Delft University of Technology, Faculty of Aerospace Engineering, Memorandum M-692, 1995.

[Mooij, 1998] Mooij, Erwin, "Aerospace-Plane Flight Dynamics - Analysis of Guidance and Control Concepts", PhD Thesis, Delft University of Technology, June 1998.

[Moore, 1966] Moore, R., *Interval Analysis*, Englewood Cliffs, New Jersey: Prentice-Hall, 1966.

[Moore, 1991] Moore, Thomas E., "Space Shuttle Entry Terminal Area Energy Management", NASA, NASA Technical Memorandum 104744, November 1991.

[Mulder *et al.*, 2007] Mulder, J. A., Staveren, W.H.J.J. van, Vaart, J.C. van der, and Weerdt, E. de, *Flight Dynamics*: Delft University of Technology, February 5 2007.

[Nedialkov, 2006a] Nedialkov, N. S., "VNODE-LP - A Validated Solver for Initial Value Problems in Ordinary Differential Equations", Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada, L8S 4K1, Technical Report CAS-06-06-NN, November 2006a. Retrieved from: http://www.cas.mcmaster.ca/~nedialk/vnodelp/

[Nedialkov, 2008] Nedialkov, N. S. (private communication), September-October 2008.

[Nedialkov and Jackson, 2002] Nedialkov, N. S. and Jackson, K. R., "The design and implementation of a validated object-oriented solver for IVPs for ODEs", Software Quality Research Laboratory, Department of Computing and Software, McMaster University, Hamilton, Canada, L8S 4K1, Technical Report 6, 2002. Retrieved from: http://www.cas.mcmaster.ca/~nedialk/Software/VNODE/VNODE.shtml

[Nedialkov, 2006b] Nedialkov, Nedialko S., "Interval Tools for ODEs and DAEs", Dept. of Computing and Software, McMaster University, Hamilton, ON, L8S 4K1, Canada, CAS 06-09-NN, November 2006b.

[Rauh and Auer, 2008] Rauh, Andreas and Auer, Ekaterina, "ValEncIA-IVP - VALidation of state ENClosures using Interval Arithmetic for Initial Value Problems", retrieved October 2008, from http://valencia-ivp.com/

[Rump, 2007] Rump, S. M., "INTLAB - INTerval LABoratory", retrieved October 2007, from http://www.ti3.tu-harburg.de/rump/intlab/index.html

[Rump, 2008] Rump, S. M., "PROFIL/BIAS", retrieved October 2008, from http://www.ti3.tu-harburg.de/Software/PROFILEnglisch.html

[Space Services Koellen, 2008] Space Services Koellen, "Project Saenger II", retrieved October 2008, from http://www.ottmar-koellen.de/saenger_2/saenger1.html

[Speyer and Bryson Jr., 1968] Speyer, J. L. and Bryson Jr., A. E., "Optimal Programming Problems with a Bounded State Space", *AIAA Journal*, Vol. 6, pp. 1488-1491, 1968.

[Stauning, 1997] Stauning, O., "Automatic Validation of Numerical Solutions", PhD thesis, Technical University of Denmark, DK-2800, Lyngby, Denmark, October 1997, http://www2.imm.dtu.dk/documents/ftp/phdliste/phd36.abstract.html.

[Stauning and Bendtsen, 2008] Stauning, Ole and Bendtsen, Claus, "FADBAD++ - Flexible Automatic differentiation using templates and operator overloading in C++", retrieved October 2008, from http://www.fadbad.com/fadbad.html

[Than *et al.*, 2003] Than, K. C., Khor, E. F., Lee, T. H., and Yang, Y. J., "A Tabu-Based exploratory algorithm for multiobjective optimization", *Artificial Intelligence Review 19*, Kluwer Academic Publishers, pp. 231-260, 2003.

[Vallado, 2007] Vallado, D. A., *Fundamentals of Astrodynamics and Applications*, Third Edition, USA: Space Technology Library, 2007.

[Vernis and Ferreira, 2006] Vernis, P. and Ferreira, E., "On-Board Trajectory Planner for the TAEM Guidance of a Winged-Body", EADS SPACE Transportation, 2006.

[Visser, 2007] Visser, H. G., *Aircraft Performance Optimization - Part 1*, Delft: Faculty of Aerospace, Delft University of Technology, 2007.

[Vrijbergen, 2004] Vrijbergen, M. R., "TAEM Path Controller Design using Feedback Linearization and Constrained Model Predictive Control", Master of Science Thesis, Control and Simulation Division, Faculty of Aerospace, Delft University of Technology, Delft, 2004.

[Wikipedia, 2007] Wikipedia, "Branch and Bound", retrieved November 2007, from http://en.wikipedia.org/wiki/Branch_and_bound

[Wikipedia, 2008a] Wikipedia, "Optimal control", retrieved November 2008, from http://en.wikipedia.org/wiki/Optimal_control

[Wikipedia, 2008b] Wikipedia, "Optimization (mathematics)", retrieved November 2008, from http://en.wikipedia.org/wiki/Optimization_(mathematics)

[Wikipedia, 2008c] Wikipedia, "Trajectory Optimization", retrieved November 2008, from http://en.wikipedia.org/wiki/Trajectory_optimization

[Yokoyama and Suzuki, 2005] Yokoyama, N. and Suzuki, S., "Modified Genetic Algorithm for Constrained Trajectory Optimization", *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 1, January-February 2005.

Appendix A | Some practical issues of INTLAB

The next two sections report two practical problems that may arise during the normal utilization of INTLAB. Possible solutions are presented.

A.1 The difference between intval(0.1) *and* intval('0.1')

INTLAB's command $intval(x)$ can be used to convert a real number x into a thin interval $[\underline{x}, \overline{x}]$.

If x is a machine-representable number, then the commands $intval(x)$ and $intval('x')$ are completely equivalent and:

$$\underline{x} = \overline{x} \tag{A-1}$$

However, if x is not a machine-representable number, like for example $x = 0.1$, then the commands $intval(x)$ and $intval('x')$ yield different outputs.

Long story short, to correctly bound a number like 0.1, the right command is $intval('0.1')$. That will produce an interval $[\underline{x}, \overline{x}]$ satisfying:

$$\underline{x} \le 0.1 \le \overline{x} \tag{A-2}$$

On the other hand, the bounds produced by the command $intval(0.1)$ might not contain 0.1.

With INTLAB, checking if a real number x is machine representable or not is easy. If it is, then:

$$diam\big(intval('x')\big) = 0 \tag{A-3}$$

$diam(x)$ is the INTLAB command to calculate the width of an interval. If it is not, then:

$$diam\big(intval('x')\big) > 0 \tag{A-4}$$

A.2 Rigorous bounds on a result

The next example was taken from MATLAB. To access it, type: *help intval*.

The idea is to calculate rigorous bounds on:

$$x = 0.125 + {}^{1}\!/_{10} \tag{A-5}$$

which is equal to 0.225. Writing (A-5) as:

$$X = intval\left(0.125\right) + {}^{1}\!/_{10} \tag{A-6}$$

will not work. According to the priority rules of real and interval arithmetic, division has priority over addition. Therefore, ${}^{1}\!/_{10}$ will be the first term to be calculated. Since both the numerator and the denominator are real numbers, interval arithmetic will be used to calculate ${}^{1}\!/_{10}$, making it impossible for X to contain 0.225.

A correct way of calculating (A-5) is:

$$X = 0.125 + \frac{intval\left(1\right)}{10} \tag{A-7}$$

Again, $\frac{intval\left(1\right)}{10}$ will be the first term to be calculated. The difference is that now the numerator is an interval, forcing MATLAB to use interval arithmetic. Adding a real number with an interval also requires interval arithmetic. At the end, 0.225 will belong to X .

To make sure the true result is always correctly bounded, the easiest way is to define all quantities in an expression as intervals, i.e.:

$$X = intval\left(0.125\right) + \frac{intval\left(1\right)}{intval\left(10\right)} \tag{A-8}$$

Appendix B | Input Files

During this thesis, three software tools were developed:

1) A TEG determination tool;

2) A TAEM optimal trajectory design tool;

3) A 3-DOF methodology script.

Each tool has its own input file. The structure of these input files is detailed in the subsequent sections.

B.1 TEG Determination Tool Input File

The input file of the TEG determination tool is called *inputfileTEGdimensioning.in* and its structure is explained in **Table B.1**.

Table B.1: *TEG Determination Tool Input File Structure.*

#	Input	Symbol	Format	Units
1	Acceptable relative heading angle (interval) at ALI	$\chi_{r,ALI}$	*interval*	deg
2	Acceptable x-position (interval) at ALI	x_{ALI}	*interval*	m
3	Acceptable y-position (interval) at ALI	y_{ALI}	*interval*	m
4	Flight-path bank angle (interval) during sub-phase 1	μ_1	*interval*	deg
5	Flight-path bank angle (interval) during sub-phase 3	μ_2	*interval*	deg
6	Number of intervals in which μ_1 is split	N_{μ_1}	*int*	-
7	Number of intervals in which μ_2 is split	N_{μ_2}	*int*	-
8	Reference area of the vehicle	S	*float*	m^2
9	Re-entry mass of the vehicle	m	*float*	kg
10	Number of rows/angle-of-attack points of $C_L(\alpha,M)$ and $C_D(\alpha,M)$	N/A	*int*	-
11	Number of columns/Mach number points of $C_L(\alpha,M)$ and $C_D(\alpha,M)$	N/A	*int*	-

#	Input	Symbol	Format				Units
12	Aerodynamic lift coefficient	$C_L(\alpha,M)$	NaN	M ($float$)			-
			α ($float$)	C_L ($float$)			
13	Aerodynamic drag coefficient	$C_D(\alpha,M)$	NaN	M ($float$)			-
			α ($float$)	C_D ($float$)			
14	Number of rows/Mach number points of $\alpha_{\min}(M)$ and $\alpha_{\max}(M)$	N/A	int				-
15	Minimum and maximum trimmable α as a function of M	$\alpha_{\min}(M)$ and $\alpha_{\max}(M)$	M ($float$)	$\alpha_{\min}$ ($float$)	$\alpha_{\max}$ ($float$)		deg
16	Mach number at TEP	M_{TEP}	$float$				-
17	Mach number at ALI	M_{ALI}	$float$				-
18	Altitude at TEP	h_{TEP}	$float$				m
19	Altitude at ALI	h_{ALI}	$float$				m
20	Number of integration steps	N_{int}	int				-
21	Flight-path angle at TEP	γ_{TEP}	$float$				deg
22	Minimum α during subsonic regime	$\alpha_{sub,\min}$	$float$				deg
23	Maximum α during subsonic regime	$\alpha_{sub,\max}$	$float$				deg
24	Subsonic α search step	$\Delta\alpha$	$float$				deg
25	Maximum subsonic M	$M_{subsonic}$	$float$				-
26	Maximum dynamic pressure supported by the vehicle	$q_{\max}$	$float$				Pa
27	Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\min}$	$float$				-
28	Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\max}$	$float$				-
29	Sub-phases relative length	$\Delta E_{rel,sub-phases}$	Sub-phase 1 length ($float$)	Sub-phase 2 length ($float$)	Sub-phase 3 length ($float$)	Sub-phase 4 length ($float$)	- (%)

An example is given in **Figure B.1**:

```
[-3,3]
[-1000,1000]
[-1000,1000]
[-60,60]
[-60,60]
12
140
110
26029
13
12
NaN 0.2 0.6 0.8 0.9 0.95 1.05 1.1 1.2 1.5 2 3 5
0 0.03 0.03 0.03 0.03 0.03 0.03 0.03 -0.08 -0.08 -0.08 -0.08 -0.08
2.5 0.2 0.1 0.2 0.2 0.2 0.13 0.13 0.023 NaN NaN NaN NaN
5 0.19 0.19 0.19 0.19 0.19 0.3 0.3 0.3 0.09 0.07 0.06 0.05
7.5 0.3 0.3 0.3 0.3 0.3 0.4 0.4 0.3 NaN NaN NaN NaN
10 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.36 0.27 0.22 0.19 0.17
12.5 0.45 0.45 0.45 0.5 0.42 NaN NaN NaN NaN NaN NaN NaN
15 0.6 0.6 0.6 0.5 0.5 NaN NaN 0.67 0.5 0.4 0.4 0.3
20 0.7 0.7 0.7 0.6 0.6 NaN NaN 1.0 0.7 0.6 0.5 0.5
25 0.8 0.8 0.7 0.6 0.6 NaN NaN 1.3 1.0 0.8 0.7 0.7
30 0.9 0.8 0.8 0.7 0.6 NaN NaN 1.3 1.3 1.0 0.8 0.8
35 NaN NaN NaN NaN NaN NaN NaN 1.3 1.4 1.1 0.93 0.9
40 NaN NaN NaN NaN NaN NaN NaN 1.3 1.32 1.19 1.03 0.94
45 NaN NaN NaN NaN NaN NaN NaN 1.3 1.31 1.3 1 1
NaN 0.2 0.6 0.8 0.9 0.95 1.05 1.1 1.2 1.5 2 3 5
0 0.06 0.06 0.06 0.2 0.2 0.5 0.5 0.4 0.3 0.2 0.2 0.1
2.5 0.05 0.04 0.05 0.07 0.2 0.3 0.3 0.07 NaN NaN NaN NaN
5 0.06 0.05 0.06 0.08 0.2 0.3 0.25 0.16 0.11 0.09 0.09 0.08
7.5 0.08 0.07 0.07 0.2 0.2 0.3 0.3 0.2 NaN NaN NaN NaN
10 0.1 0.1 0.1 0.2 0.2 0.23 0.2 0.2 0.1 0.1 0.1 0.08
12.5 0.1 0.22 0.23 0.17 0.23 NaN NaN NaN NaN NaN NaN NaN
15 0.1 0.1 0.1 0.2 0.2 NaN NaN 0.3 0.3 0.2 0.2 0.2
20 0.3 0.3 0.3 0.4 0.5 NaN NaN 0.5 0.4 0.3 0.3 0.2
25 0.3 0.3 0.4 0.5 0.5 NaN NaN 0.7 0.5 0.5 0.4 0.3
30 0.5 0.5 0.45 0.5 0.67 NaN NaN 0.8 0.7 0.6 0.5 0.6
35 NaN NaN NaN NaN NaN NaN NaN 1.0 1.0 0.8 0.7 0.7
40 NaN NaN NaN NaN NaN NaN NaN 1.1 1.1 1.1 0.9 0.9
45 NaN NaN NaN NaN NaN NaN NaN 1.3 1.3 1.4 1.2 1.1
9
0 0 30
0.75 0 30
0.95 0 10
1.2 0 10
1.5 1.5 15.5
2 3.5 22.5
3 6.5 32.5
5 10 45
6 10 45
1.5
0.5
15000
3000
3000
-10
0
30
1
0.75
40000
-0.75
2.5
0.44 0.05 0.5 0.01
```

Figure B.1: *TEG Determination Tool Input File Example.*

B.2 TAEM Optimal Trajectory Design Tool Input File

The input file of the TAEM optimal trajectory design tool is called *inputfileHACoptimizer.in* and its structure is explained in **Table B.2**.

Table B.2: *TAEM Optimal Trajectory Design Tool Input File Structure.*

#	Input	Symbol	Format			Units
1	Relative heading angle at TEP	$\chi_{r,TEP}$	*interval*			deg
2	x-position at TEP	x_{TEP}	*interval*			m
3	y-position at TEP	y_{TEP}	*interval*			m
4	Flight-path bank angle (interval) during sub-phase 1	μ_1	*interval*			deg
5	Flight-path bank angle (interval) during sub-phase 3	μ_2	*interval*			deg
6	Number of intervals in which μ_1 is split	N_{μ_1}	*int*			-
7	Number of intervals in which μ_2 is split	N_{μ_2}	*int*			-
8	Reference area of the vehicle	S	*float*			m^2
9	Re-entry mass of the vehicle	m	*float*			kg
10	Number of rows/angle-of-attack points of $C_L(\alpha,M)$ and $C_D(\alpha,M)$	N/A	*int*			-
11	Number of columns/Mach number points of $C_L(\alpha,M)$ and $C_D(\alpha,M)$	N/A	*int*			-
12	Aerodynamic lift coefficient	$C_L(\alpha,M)$	NaN α (*float*)	M (*float*) C_L (*float*)		-
13	Aerodynamic drag coefficient	$C_D(\alpha,M)$	NaN α (*float*)	M (*float*) C_D (*float*)		-
14	Number of rows/Mach number points of $\alpha_{\min}(M)$ and $\alpha_{\max}(M)$	N/A	*int*			-
15	Minimum and maximum trimmable α as a function of M	$\alpha_{\min}(M)$ and $\alpha_{\max}(M)$	M (*float*)	$\alpha_{\min}$ (*float*)	$\alpha_{\max}$ (*float*)	deg
16	Mach number at TEP	M_{TEP}	*float*			-
17	Mach number at ALI	M_{ALI}	*float*			-

#	Input	Symbol	Format				Units
18	Altitude at TEP	h_{TEP}	*float*				m
19	Altitude at ALI	h_{ALI}	*float*				m
20	Number of integration steps	N_{int}	*int*				-
21	Flight-path angle at TEP	γ_{TEP}	*float*				deg
22	Minimum α during subsonic regime	$\alpha_{sub,\min}$	*float*				deg
23	Maximum α during subsonic regime	$\alpha_{sub,\max}$	*float*				deg
24	Subsonic α search step	$\Delta\alpha$	*float*				deg
25	Maximum subsonic M	$M_{subsonic}$	*float*				-
26	Maximum dynamic pressure supported by the vehicle	$q_{\max}$	*float*				Pa
27	Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\min}$	*float*				-
28	Minimum load factor along the negative z-body axis supported by the vehicle	$n_{z,\max}$	*float*				-
29	Sub-phases relative lengths	$\Delta E_{rel,sub-phases}$	Sub-phase 1 length (*float*)	Sub-phase 2 length (*float*)	Sub-phase 3 length (*float*)	Sub-phase 4 length (*float*)	- (%)
30	χ_r cost function weight	$\chi_{r,weight}$	*float*				deg
31	x-position cost function weight	x_{weight}	*float*				m
32	y-position cost function weight	y_{weight}	*float*				m
33	Initial upper bound on the global minimum	$\bar{J}$	*float*				-
34	Stopping criterion based on the width of the parameters	ε_P	ε_{μ_1} (*float*)		ε_{μ_2} (*float*)		deg
35	Stopping criterion based on width of the cost function	ε_J	*float*				-

An example is given in **Figure B.2**:

```
[240,240]
[26000,26000]
[0,0]
[-60,60]
[-60,60]
2
4
110
26029
13
12
NaN 0.2 0.6 0.8 0.9 0.95 1.05 1.1 1.2 1.5 2 3 5
0 0.03 0.03 0.03 0.03 0.03 0.03 0.03 -0.08 -0.08 -0.08 -0.08 -0.08
2.5 0.2 0.1 0.2 0.2 0.2 0.13 0.13 0.023 NaN NaN NaN NaN
5 0.19 0.19 0.19 0.19 0.19 0.3 0.3 0.3 0.09 0.07 0.06 0.05
7.5 0.3 0.3 0.3 0.3 0.3 0.4 0.4 0.3 NaN NaN NaN NaN
10 0.4 0.4 0.4 0.4 0.4 0.5 0.4 0.36 0.27 0.22 0.19 0.17
12.5 0.45 0.45 0.45 0.5 0.42 NaN NaN NaN NaN NaN NaN NaN
15 0.6 0.6 0.6 0.5 0.5 NaN NaN 0.67 0.5 0.4 0.4 0.3
20 0.7 0.7 0.7 0.6 0.6 NaN NaN 1.0 0.7 0.6 0.5 0.5
25 0.8 0.8 0.7 0.6 0.6 NaN NaN 1.3 1.0 0.8 0.7 0.7
30 0.9 0.8 0.8 0.7 0.6 NaN NaN 1.3 1.3 1.0 0.8 0.8
35 NaN NaN NaN NaN NaN NaN NaN 1.3 1.4 1.1 0.93 0.9
40 NaN NaN NaN NaN NaN NaN NaN 1.3 1.32 1.19 1.03 0.94
45 NaN NaN NaN NaN NaN NaN NaN 1.3 1.31 1.3 1 1
NaN 0.2 0.6 0.8 0.9 0.95 1.05 1.1 1.2 1.5 2 3 5
0 0.06 0.06 0.06 0.2 0.2 0.5 0.5 0.4 0.3 0.2 0.2 0.1
2.5 0.05 0.04 0.05 0.07 0.2 0.3 0.3 0.07 NaN NaN NaN NaN
5 0.06 0.05 0.06 0.08 0.2 0.3 0.25 0.16 0.11 0.09 0.09 0.08
7.5 0.08 0.07 0.07 0.2 0.2 0.3 0.3 0.2 NaN NaN NaN NaN
10 0.1 0.1 0.1 0.2 0.2 0.23 0.2 0.2 0.1 0.1 0.1 0.08
12.5 0.1 0.22 0.23 0.17 0.23 NaN NaN NaN NaN NaN NaN NaN
15 0.1 0.1 0.1 0.2 0.2 NaN NaN 0.3 0.3 0.2 0.2 0.2
20 0.3 0.3 0.3 0.4 0.5 NaN NaN 0.5 0.4 0.3 0.3 0.2
25 0.3 0.3 0.4 0.5 0.5 NaN NaN 0.7 0.5 0.5 0.4 0.3
30 0.5 0.5 0.45 0.5 0.67 NaN NaN 0.8 0.7 0.6 0.5 0.6
35 NaN NaN NaN NaN NaN NaN NaN 1.0 1.0 0.8 0.7 0.7
40 NaN NaN NaN NaN NaN NaN NaN 1.1 1.1 1.1 0.9 0.9
45 NaN NaN NaN NaN NaN NaN NaN 1.3 1.3 1.4 1.2 1.1
9
0 0 30
0.75 0 30
0.95 0 10
1.2 0 10
1.5 1.5 15.5
2 3.5 22.5
3 6.5 32.5
5 10 45
6 10 45
1.5
0.5
15000
3000
3000
-10
0
30
1
0.75
40000
-0.75
2.5
0.44 0.05 0.5 0.01
3
1000
1000
Inf
Inf Inf
0.3
```

Figure B.2: *TAEM Optimal Trajectory Design Tool Input File Example.*

B.3 3-DOF Methodology Script Input File

The input file of the 3-DOF methodology script is called *inputfilevnodelp3dof.in* and its structure is explained in **Table B.3**.

Table B.3: *3-DOF Methodology Script Input File Structure.*

#	Input	Symbol	Format				Units
1	Flight-path bank angle (interval) during sub-phase 1	μ_1	*interval*				deg
2	Flight-path bank angle (interval) during sub-phase 3	μ_2	*interval*				deg
3	Reference area of the vehicle	S	*float*				m^2
4	Re-entry mass of the vehicle	m	*float*				kg
5	Mach number at TEP	M_{TEP}	*float*				-
6	Mach number at ALI	M_{ALI}	*float*				-
7	Altitude at TEP	h_{TEP}	*float*				m
8	Altitude at ALI	h_{ALI}	*float*				m
9	Number of integration steps used by SIMULINK	N_{int}	*int*				-
10	Flight-path angle at TEP	γ_{TEP}	*float*				deg
11	Relative heading angle at TEP	χ_{TEP}	*float*				deg
12	x-position at TEP	x_{TEP}	*float*				m
13	y-position at TEP	y_{TEP}	*float*				m
14	Sub-phases relative lengths	$\Delta E_{rel,sub-phases}$	Sub-phase 1 length (*float*)	Sub-phase 2 length (*float*)	Sub-phase 3 length (*float*)	Sub-phase 4 length (*float*)	- (%)
15	Aerodynamic lift coefficient per sub-phase	C_L	C_L at sub-phase 1 (*float*)	C_L at sub-phase 2 (*float*)	C_L at sub-phase 3 (*float*)	C_L at sub-phase 4 (*float*)	-
16	Aerodynamic drag coefficient per sub-phase	C_D	C_D at sub-phase 1 (*float*)	C_D at sub-phase 2 (*float*)	C_D at sub-phase 3 (*float*)	C_D at sub-phase 4 (*float*)	-
17	Integration order used by VNODE-LP	p	*int*				-

An example is given in **Figure B.3**:

```
[60,60]
[-60,-60]
110
26029
1.5
0.5
15000
3000
3000
-10
0
-65000
34000
0.44 0.05 0.5 0.01
0.3 0.2 0.2 0.2
0.14 0.14 0.11 0.05
7
```

Figure B.3: *3-DOF Methodology Script Input File Example.*

Appendix C | More dynamic global optimization examples

In Chapter 5, two dynamic global optimization examples were mentioned but not solved: **Example 2** and **Example 4**. These two examples are solved here.

C.1 Example 2

The only difference between **Example 2** and **Example 1** is that now there are some constraints on the control function:

Performance index: $$J\left(x(t),u(t),t\right)=\int_0^1\left(0.5u^2(t)+x(t)\right)dt \tag{C-1}$$

State dynamics: $$\dot{x}(t)=u(t) \tag{C-2}$$

Initial state: $$x(0)=0 \tag{C-3}$$

Final state: $$x(1)=1 \tag{C-4}$$

Control function constraints: $$0.75\le u(t)\le 1.25 \tag{C-5}$$

According to [Visser, 2007], the solution is:

Optimal control function: $$u^*(t)=\begin{cases}0.75 & ,0\le t\le 0.25\\ t+0.5 & ,0.25\le t\le 0.75\\ 1.25 & ,0.75\le t\le 1\end{cases} \tag{C-6}$$

Global minimum: $$J^*\approx 0.963542 \tag{C-7}$$

Example 2 has a higher global minimum J^* than **Example 1** because of (C-5).

C.1.1 Description of Chu's algorithm

The following parameterization is suggested in [Chu, 2007]:

$$u(t)=\begin{cases} 0.75 & ,0\le t\le 0.2 \\ 0.75\cap\left(P_1\left(t-P_2\right)+0.75\right)\cap\left(P_1\left(t-P_3\right)+1.25\right) & ,0.2<t<0.4 \\ \left(P_1\left(t-P_2\right)+0.75\right)\cap\left(P_1\left(t-P_3\right)+1.25\right) & ,0.4\le t\le 0.6 \\ 1.25\cap\left(P_1\left(t-P_2\right)+0.75\right)\cap\left(P_1\left(t-P_3\right)+1.25\right) & ,0.6<t<0.8 \\ 1.25 & ,0.8\le t\le 1 \end{cases} \quad \text{(C-8)}$$

with:

$$P_{initial}=\begin{bmatrix} [0,1] \\ [0.2,0.4] \\ [0.6,0.8] \end{bmatrix} \quad \text{(C-9)}$$

and:

$$P^*=\begin{bmatrix} 1 \\ 0.25 \\ 0.75 \end{bmatrix} \quad \text{(C-10)}$$

Take $t=0.3$ for example. According to (C-8), the control function $u(t)$ can only be equal to 0.75 or to $\varnothing$. Either way, the result is wrong. Finding P^* with (C-8) is therefore impossible.

As for the algorithm itself, the only differences relatively to Section 5.3.1.2 are that:

1) a shrinking phase is used.

2) the constraint is checked. No details are given about how this is done.

3) J^I, P^I and X^I could be calculated for Chu's algorithm.

As a final remark, it is not explained in [Chu, 2007] how exactly are the final state and the performance index calculated in this particular example.

C.1.2 Description of the developed algorithm

Without making further assumptions, the 3rd order parameterization (C-8) can be rewritten as a 2nd order parameterization as follows:

$$u(t)=\begin{cases} 0.75 & ,0\le t\le P_1 \\ \dfrac{0.5}{P_2-P_1}\left(t-P_1\right)+0.75 & ,P_1\le t\le P_2 \\ 1.25 & ,P_2\le t\le 1 \end{cases} \quad \text{(C-11)}$$

or in a more detailed form:

$$u(t)=\begin{cases} 0.75 & ,0\le t\le \underline{P_1} \\ 0.75\cup\left(\dfrac{0.5}{P_2-P_1}(t-P_1)+0.75\right) & ,\underline{P_1}\le t\le \overline{P_1} \\ \dfrac{0.5}{P_2-P_1}(t-P_1)+0.75 & ,\overline{P_1}\le t\le \underline{P_2} \\ \left(\dfrac{0.5}{P_2-P_1}(t-P_1)+0.75\right)\cup 1.25 & ,\underline{P_2}\le t\le \overline{P_2} \\ 1.25 & ,\overline{P_2}\le t\le 1 \end{cases} \qquad \text{(C-12)}$$

where:

$$P^*=\begin{bmatrix}0.25\\0.75\end{bmatrix} \qquad \text{(C-13)}$$

The meaning of P_1 and P_2 can be understood from **Figure C.1**, where P_1 and P_2 are assumed to be point interval variables.

From **Figure C.1**, an intuitive choice for $P_{initial}$ is:

$$P_{initial}=\begin{bmatrix}[0,0.5]\\ [0.5,1]\end{bmatrix} \qquad \text{(C-14)}$$

Like **Example 1**, this problem is completely solvable by HC. Since there are only two parameters, two interval equations are sufficient.

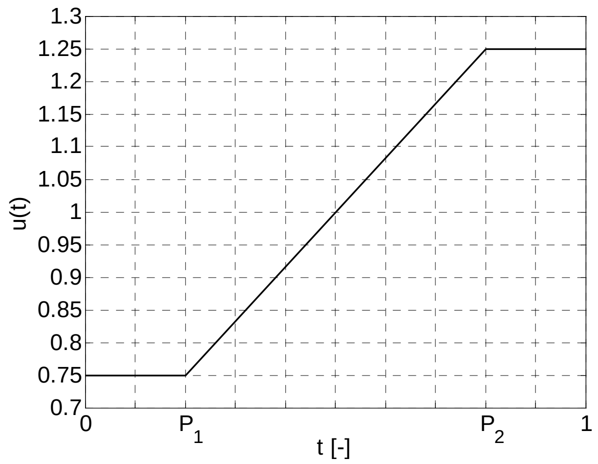

Figure C.1: *Illustration of the parameterization used to solve* ***Example 2****.*

The first one results from the initial and final state. For $0\le t\le P_1$, the following is true:

$$\dot{x}=u=0.75\Leftrightarrow x(t)=0.75t+A\xrightarrow{x(0)=0\Rightarrow A=0}x(t)=0.75t \qquad \text{(C-15)}$$

For $P_1\le t\le P_2$:

$$\dot{x} = u = \frac{0.5}{P_2 - P_1}(t - P_1) + 0.75$$
$$\Leftrightarrow x(t) = \frac{0.5}{P_2 - P_1}\frac{(t - P_1)^2}{2} + 0.75t + B \qquad \textit{(C-16)}$$

Equalizing (C-15) with (C-16) at $t = P_1$ yields:

$$0.75P_1 = 0.75P_1 + B \Leftrightarrow B = 0 \qquad \textit{(C-17)}$$

meaning that, for $P_1 \le t \le P_2$:

$$x(t) = \frac{0.5}{P_2 - P_1}\frac{(t - P_1)^2}{2} + 0.75t \qquad \textit{(C-18)}$$

Finally, for $P_2 \le t \le 1$:

$$\dot{x} = u = 1.25 \Leftrightarrow x(t) = 1.25t + C \qquad \textit{(C-19)}$$

Equalizing (C-18) with (C-19) at $t = P_2$ yields:

$$0.25(P_2 - P_1) + 0.75P_2 = 1.25P_2 + C \Leftrightarrow C = -0.25(P_1 + P_2) \qquad \textit{(C-20)}$$

meaning that, for $P_2 \le t \le 1$:

$$x(t) = 1.25t - 0.25(P_1 + P_2) \qquad \textit{(C-21)}$$

The condition $x(1) = 1$ implies that:

$$1.25 - 0.25(P_1 + P_2) = 1 \Leftrightarrow \boxed{P_1 + P_2 = 1} \qquad \textit{(C-22)}$$

Expression (C-22) is the first equation. The second one is a consequence of (5-4) and (5-5). Defining the Hamiltonian as explained in Section 5.1.4 yields:

$$H = \begin{cases} 0.5u^2 + x + \lambda u + \mu_1(-u + 0.75) & ,0 \le t \le P_1 \\ 0.5u^2 + x + \lambda u & ,P_1 \le t \le P_2 \\ 0.5u^2 + x + \lambda u + \mu_2(u - 1.25) & ,P_2 \le t \le 1 \end{cases} \qquad \textit{(C-23)}$$

From (5-4), it follows that:

$$\dot{\lambda} = -\frac{\partial H}{\partial x} = -1 \Leftrightarrow \lambda(t) = -t + D \quad ,0 \le t \le 1 \qquad \textit{(C-24)}$$

On the other hand, from (5-5) it follows that for $P_1 \le t \le P_2$:

$$\frac{\partial H}{\partial u}=0 \Leftrightarrow u+\lambda=0 \Leftrightarrow \lambda=-u \qquad \text{(C-25)}$$

Equalizing (C-24) with (C-25) at $t=P_1$ and at $t=P_2$ yields, respectively:

$$-P_1+D=-0.75 \qquad \text{(C-26)}$$

$$-P_2+D=-1.25 \qquad \text{(C-27)}$$

Eliminating D between (C-26) and (C-27) results in the second equation:

$$\boxed{P_2-P_1=0.5} \qquad \text{(C-28)}$$

The following subroutine was then implemented to solve (C-22) and (C-28) through HC:

$$P_1^{'}=1-P_2 \qquad \text{(C-29)}$$

$$P_1=P_1\cap P_1^{'} \qquad \text{(C-30)}$$

$$P_1^{'}=P_2-0.5 \qquad \text{(C-31)}$$

$$P_1=P_1\cap P_1^{'} \qquad \text{(C-32)}$$

$$P_2^{'}=1-P_1 \qquad \text{(C-33)}$$

$$P_2=P_2\cap P_2^{'} \qquad \text{(C-34)}$$

$$P_2^{'}=0.5+P_1 \qquad \text{(C-35)}$$

$$P_2=P_2\cap P_2^{'} \qquad \text{(C-36)}$$

Since this problem is very simple, introducing a shrinking phase would not improve the results.

The final state can be calculated, without being affected by dependency, through (C-21).

Calculating the performance index is more complex. The integral (C-1) needs to be divided into 5 different parts, each one corresponding to a different section of $u(t)$, as defined in (C-12). Writing out the different parcels of the performance index results in (C-37).

$$
\begin{aligned}
J &= \int_0^1 \left(0.5u^2 + x\right)dt = \\
&= \underbrace{\int_0^{\underline{P_1}} \left(0.5(0.75)^2 + (0.75t)\right)dt}_{1^{st}\ parcel} + \\
&\quad \underbrace{+\int_{\underline{P_1}}^{\overline{P_1}} \begin{pmatrix} 0.5\left\{0.75 \cup \left(\dfrac{0.5}{P_2 - P_1}(t - P_1) + 0.75\right)\right\}^2 + \\ + \left\{(0.75t) \cup \left(\dfrac{0.5}{P_2 - P_1}\dfrac{(t - P_1)^2}{2} + 0.75t\right)\right\} \end{pmatrix} dt}_{2^{nd}\ parcel} + \\
&\quad \underbrace{+\int_{\overline{P_1}}^{\underline{P_2}} \begin{pmatrix} 0.5\left(\dfrac{0.5}{P_2 - P_1}(t - P_1) + 0.75\right)^2 + \\ + \left(\dfrac{0.5}{P_2 - P_1}\dfrac{(t - P_1)^2}{2} + 0.75t\right) \end{pmatrix} dt}_{3^{rd}\ parcel} + \\
&\quad \underbrace{+\int_{\underline{P_2}}^{\overline{P_2}} \begin{pmatrix} 0.5\left\{\left(\dfrac{0.5}{P_2 - P_1}(t - P_1) + 0.75\right) \cup 1.25\right\}^2 + \\ + \left\{\left(\dfrac{0.5}{P_2 - P_1}\dfrac{(t - P_1)^2}{2} + 0.75t\right) \cup \left(1.25t - 0.25(P_1 + P_2)\right)\right\} \end{pmatrix} dt}_{4^{th}\ parcel} + \\
&\quad \underbrace{+\int_{\overline{P_2}}^{1} \left(0.5(1.25)^2 + \left(1.25t - 0.25(P_1 + P_2)\right)\right)dt}_{5^{th}\ parcel}
\end{aligned}
\qquad \text{(C-37)}
$$

The first parcel and the last were analytically calculated as follows:

1st parcel:

$$\int_0^{\underline{P_1}} \left(0.5(0.75)^2 + (0.75t)\right)dt = 0.5 \cdot 0.75^2 \cdot \underline{P_1} + \frac{0.75\underline{P_1}^2}{2} \qquad \text{(C-38)}$$

5th parcel:

$$
\begin{aligned}
&\int_{\overline{P_2}}^{1} \left(0.5(1.25)^2 + \left(1.25t - 0.25(P_1 + P_2)\right)\right)dt = \\
&= \frac{(P_1 + P_2)(\overline{P_2} - 1)}{4} - \frac{5\overline{P_2}^2}{8} - \frac{25\overline{P_2}}{32} + \frac{45}{32}
\end{aligned}
\qquad \text{(C-39)}
$$

The other ones were integrated using the rectangular method (3-39) described in Section 3.6. The integration limits of the 2nd, 3rd and 4th parcels were divided into N_{int} subintervals each.

No special care was taken to reduce the dependency effect in the calculation of (C-37).

C.1.3 Comparing the algorithms

Table C.1 shows all the inputs not yet mentioned.

Since the parameterization used here is different from the one used in [Chu, 2007], the accuracy of the two algorithms can only be compared in terms of the width of the "guaranteed bounds" on J^*. The stopping criteria were chosen accordingly.

Table C.1: *User inputs in* ***Example 2****.*

Designation	Symbol		Value
Number of time subintervals used for integration	N_{int}		500
New boxes per old box	N_{split}		$\begin{bmatrix}0 & 2\end{bmatrix}^T$
Stopping criteria	ε_P	ε_{P_1}	$+\infty$
		ε_{P_2}	$+\infty$
	ε_J		2.0e-3
Hull consistency stopping criteria	ε_{abs}		2
	ε_{rel}		1.01

The HC stopping criteria were chosen deliberately high to optimize the CPU time.

In this particular case, choosing $N_{split} = [-]$, in accordance to (5-41), would not work. At least one parameter must be split for the algorithm to converge.

In **Table C.2**, the two algorithms are compared.

Table C.2: *Comparative results for* ***Example 2****.*

			The developed algorithm		**Chu's algorithm**	
Designation	**Symbol**		**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	J^I		$[0.9630, 0.9641]$	1.0e-3	$[0.9623, 0.9650]$	2.6e-3
Bounds on the solution point	P^I	P_1^I	$[0.249, 0.251]$	0	$[0.820, 0.910]$	9e-2
		P_2^I	$[0.749, 0.751]$	0	$[0.200, 0.220]$	2e-2
		P_3^I	N/A	N/A	$[0.7784, 0.8000]$	2.2e-2
Bounds on the final state	X^I		$[1,1]$	0	$[0.9991, 1.0001]$	1.8e-3
CPU time	$\Delta t_{CPU}\,[s]$		0.98	N/A	7.32	N/A
Number of remaining boxes	$N_{remaining}$		2	N/A	2	N/A

Comparing Chu's result with (C-10) shows that P^* is not contained inside the hull of the solution set P^I. This proves, without a doubt, that the algorithm given in [Chu, 2007] is unable to give guaranteed bounds on the solution point.

On the other hand, the developed algorithm correctly bounds P^* and only needs approximately $1/7$ of the time to achieve the same accuracy.

The developed algorithm produces relatively wide bounds on J^I, especially considering that P^I is a point interval. This is due to the wrapping effect that affects the calculation of (C-37). A relatively big value of N_{int} was chosen in order to minimize this effect, with obvious consequences to the algorithm's speed. Only a new integration technique could improve these results.

This example served to show the capabilities of HC.

C.2 Example 4

The only difference between **Example 4** and **Example 3** is the state inequality constraint (C-47).

Performance index: $$J\left(x(t),u(t),t\right)=\int_0^1 0.5u^2(t)dt \qquad \text{(C-40)}$$

State dynamics: $$\dot{x}_1(t)=u(t) \qquad \text{(C-41)}$$

$$\dot{x}_2(t)=x_1(t) \qquad \text{(C-42)}$$

Initial state: $$x_1(0)=1 \qquad \text{(C-43)}$$

$$x_2(0)=0 \qquad \text{(C-44)}$$

Final state: $$x_1(1)=-1 \qquad \text{(C-45)}$$

$$x_2(1)=0 \qquad \text{(C-46)}$$

State inequality constraint: $$x_2(t)\le 0.2 \quad ,0\le t\le 1 \qquad \text{(C-47)}$$

The analytical solution is given in [Bryson Jr. and Ho, 1975]:

Optimal control function: $$u^*(t)=\begin{cases}4.8t-3.2 & ,0\le t\le 0.5\\ -4.8t+1.6 & ,0.5\le t\le 1\end{cases} \qquad \text{(C-48)}$$

Global minimum: $$J^*=2.24 \qquad \text{(C-49)}$$

Example 4 has a higher global minimum J^* than **Example 3** because of (C-47).

C.2.1 Description of Chu's algorithm

The following parameterization is suggested in [Chu, 2007]:

$$u(t)=\begin{cases} P_2t+P_3 & ,0\le t\le \underline{P_1} \\ (P_2t+P_3)\cap(P_4t+P_5) & ,\underline{P_1}<t<\overline{P_1} \\ P_4t+P_5 & ,\overline{P_1}\le t\le 1 \end{cases} \tag{C-50}$$

with:

$$P_{initial}=\begin{bmatrix}[0.45,0.55]\\ [3.0,5.0]\\ [-5.0,-2.0]\\ [-5.0,-3.0]\\ [0.0,2.0]\end{bmatrix} \tag{C-51}$$

and:

$$P^*=\begin{bmatrix}0.5 & 4.8 & -3.2 & -4.8 & 1.6\end{bmatrix}^T \tag{C-52}$$

Assume that $u^{(1)}(t)=P_2t+P_3$ and $u^{(2)}(t)=P_4t+P_5$ are represented by the blue and red regions illustrated in **Figure C.2**, respectively.

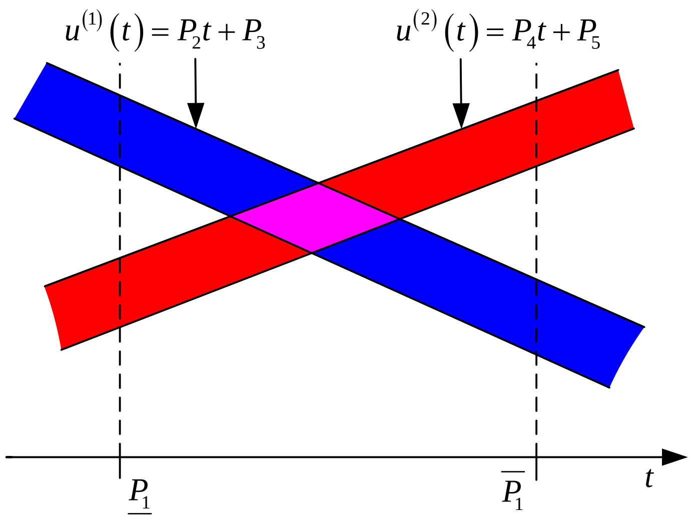

Figure C.2: $u(t)$ *for* $t\in P_1$.

For any $t\in P_1$ outside the pink area, $u(t)=\varnothing$. Obviously, this is not acceptable. The parameterization used in [Chu, 2007] is therefore incorrect.

This example was solved in [Chu, 2007] using the algorithm explained in Section 5.3.1.2 with the modifications enumerated in Section C.1.1.

C.2.2 Description of the developed algorithm

Without making further assumptions, the 5th order parameterization (C-50) can be rewritten as a 4th order parameterization as follows:

$$u(t)=\begin{cases}\dfrac{P_3-P_2}{P_1}t+P_2 & ,0\le t\le P_1\\ \dfrac{P_4-P_3}{1-P_1}(t-1)+P_4 & ,P_1\le t\le 1\end{cases} \tag{C-53}$$

or in a more detailed form:

$$u(t)=\begin{cases}\dfrac{P_3-P_2}{P_1}t+P_2 & ,0\le t\le \underline{P_1}\\ \left(\dfrac{P_3-P_2}{P_1}t+P_2\right)\cup\left(\dfrac{P_4-P_3}{1-P_1}(t-1)+P_4\right) & ,\underline{P_1}\le t\le \overline{P_1}\\ \dfrac{P_4-P_3}{1-P_1}(t-1)+P_4 & ,\overline{P_1}\le t\le 1\end{cases} \tag{C-54}$$

where:

$$P^*=\begin{bmatrix}0.5 & -3.2 & -0.8 & -3.2\end{bmatrix}^T \tag{C-55}$$

The meaning of P_1, P_2, P_3 and P_4 can be understood from **Figure C.3**, where they are assumed to be point interval variables.

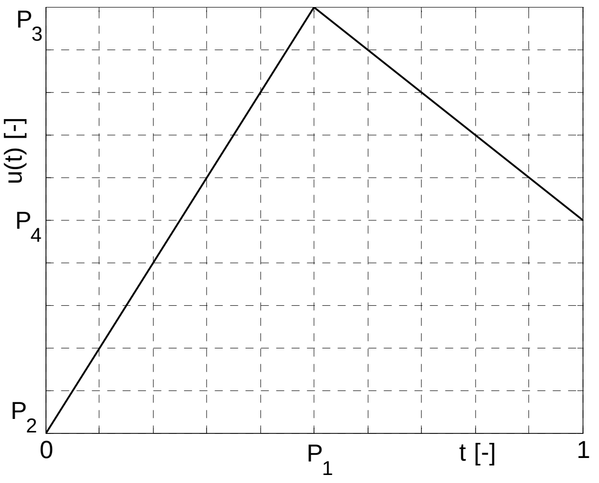

Figure C.3: *Illustration of the parameterization used to solve* ***Example 4****.*

By looking at (C-50) and (C-54), it is possible to relate Chu's parameterization (identified by the superscript ^{C}) with the one used in this thesis (superscript ^{F}).

$$P_1^F=P_1^C \tag{C-56}$$

$$u^F(t=0)=u^C(t=0)\Leftrightarrow P_2^F=P_3^C \tag{C-57}$$

$$u^F\left(t=P_1^F\right)=u^C\left(t=P_1^C\right)\Leftrightarrow P_3^F=P_2^C P_1^C+P_3^C \tag{C-58}$$

$$u^F\left(t=1\right)=u^C\left(t=1\right)\Leftrightarrow P_4^F=P_4^C+P_5^C \tag{C-59}$$

For the comparison with [Chu, 2007] to be valid, the initial box used in this thesis must be equivalent to Chu's initial box. Therefore, (C-51) was inserted into (C-56)-(C-59) and the result taken as the initial box used in this thesis:

$$P_{initial}=\begin{bmatrix}[0.45,0.55]\\ [-5,-2]\\ [-3.65,0.75]\\ [-5,-1]\end{bmatrix} \tag{C-60}$$

Two interval equations can be deduced from the state dynamics and the initial and final states. Integrating (C-41) yields:

$$\begin{cases}\dot{x}_1=u=\dfrac{P_3-P_2}{P_1}t+P_2 & ,0\le t\le P_1\\ \dot{x}_1=u=\dfrac{P_4-P_3}{1-P_1}\left(t-1\right)+P_4 & ,P_1\le t\le 1\end{cases}$$
$$\Leftrightarrow \tag{C-61}$$
$$\begin{cases}x_1^{(1)}\left(t\right)=\dfrac{P_3-P_2}{2P_1}t^2+P_2t+A & ,0\le t\le P_1\\ x_1^{(2)}\left(t\right)=\dfrac{P_4-P_3}{2\left(1-P_1\right)}\left(t-1\right)^2+P_4t+B & ,P_1\le t\le 1\end{cases}$$

The superscripts $^{(1)}$ and $^{(2)}$ are only used here to identify the different phases.

A and B can be determined by imposing (C-43) and (C-45):

$$x_1\left(0\right)=1\Leftrightarrow A=1 \tag{C-62}$$

$$x_1\left(1\right)=-1\Leftrightarrow B=-1-P_4 \tag{C-63}$$

The first equation follows from equalizing the two branches of (C-61) at $t=P_1$, i.e.:

$$x_1^{(1)}\left(P_1\right)=x_1^{(2)}\left(P_1\right) \tag{C-64}$$

resulting in:

$$\boxed{-\frac{P_2P_1}{2}-2-\frac{P_4}{2}+\frac{P_4P_1}{2}-\frac{P_3}{2}=0} \tag{C-65}$$

On the other hand, integrating (C-42) yields:

$$\begin{cases} \dot{x}_2 = x_1 = \dfrac{P_3 - P_2}{2P_1} t^2 + P_2 t + 1 & , 0 \le t \le P_1 \\ \dot{x}_2 = x_1 = \dfrac{P_4 - P_3}{2(1-P_1)} (t-1)^2 + P_4 t - 1 - P_4 & , P_1 \le t \le 1 \end{cases}$$

$$\Leftrightarrow \tag{C-66}$$

$$\begin{cases} x_2^{(1)}(t) = \dfrac{P_3 - P_2}{6P_1} t^3 + \dfrac{P_2}{2} t^2 + t + C & , 0 \le t \le P_1 \\ x_2^{(2)}(t) = \dfrac{P_4 - P_3}{6(1-P_1)} (t-1)^3 + \dfrac{P_4}{2} t^2 + (-1 - P_4) t + D & , P_1 \le t \le 1 \end{cases}$$

C and D can be determined by imposing (C-44) and (C-46):

$$x_2(0) = 0 \Leftrightarrow C = 0 \tag{C-67}$$

$$x_2(1) = 0 \Leftrightarrow D = 1 + \frac{P_4}{2} \tag{C-68}$$

The second equation follows from equalizing the two branches of (C-66) at $t = P_1$, i.e.:

$$x_2^{(1)}(P_1) = x_2^{(2)}(P_1) \tag{C-69}$$

resulting in:

$$\boxed{-\frac{P_2 P_1^2}{3} - 2P_1 \frac{P_4}{3} + \frac{P_3}{6} + \frac{P_4 P_1^2}{3} - \frac{2P_1 P_4}{3} - \frac{P_1 P_3}{3} + 1 = 0} \tag{C-70}$$

HC was applied to (C-65) and to (C-70) in the following way (the intermediate intersection steps are not shown here):

$$P_1^{'} = \frac{-2 - P_4\left(-\frac{1}{2} + \frac{P_1}{2}\right) - \frac{P_3}{2}}{\frac{P_2}{2}} \tag{C-71}$$

$$P_1^{'} = \frac{-\frac{P_2 P_1^2}{3} + P_4\left(\frac{1}{3} + \left(\frac{P_1}{3} - \frac{2}{3}\right) P_1\right) + \frac{P_3}{6} + 1}{2 + \frac{P_3}{3}} \tag{C-72}$$

$$P_2^{'} = \frac{-2 - \frac{P_3}{2} + P_4\left(-\frac{1}{2} + \frac{P_1}{2}\right)}{\frac{P_1}{2}} \tag{C-73}$$

$$P_2^{'} = \frac{P_3\left(\frac{1}{6}-\frac{P_1}{3}\right)+P_4\left(\frac{1}{3}+\left(\frac{P_1}{3}-\frac{2}{3}\right)P_1\right)-2P_1+1}{\frac{P_1^2}{3}} \qquad \text{(C-74)}$$

$$P_3^{'} = 2\left(P_4\left(-\frac{1}{2}+\frac{P_1}{2}\right)-2-\frac{P_2P_1}{2}\right) \qquad \text{(C-75)}$$

$$P_3^{'} = -6\left(\left(-\frac{P_3}{3}-\frac{P_2P_1}{3}-2\right)P_1+P_4\left(\frac{1}{3}+\left(\frac{P_1}{3}-\frac{2}{3}\right)P_1\right)+1\right) \qquad \text{(C-76)}$$

$$P_4^{'} = \frac{-\frac{P_2P_1}{2}-2-\frac{P_3}{2}}{\frac{1}{2}-\frac{P_1}{2}} \qquad \text{(C-77)}$$

$$P_4^{'} = \frac{P_3\left(\frac{1}{6}-\frac{P_1}{3}\right)+1+P_1\left(-2-\frac{P_1P_2}{3}\right)}{-\frac{1}{3}+P_1\left(-\frac{P_1}{3}+\frac{2}{3}\right)} \qquad \text{(C-78)}$$

The dependency effect was taken into account when writing (C-71)-(C-78). The form of the equations prevents the denominators from containing zero.

Since there are four parameters and just two equations, the procedure explained in Section 5.2.2.2.4 based on an upper bound on the minimum has to be used. The first step is to prove the feasibility of a box P with respect to the equality constraints. In order to apply **Theorem 4.1**, two interval parameters have to be fixed. Since the initial bounds on P_1 are relatively sharp, P_1 was chosen as one of those parameters. By also fixing P_2, feasibility can be proved in the following way:

1) $$P_{1,m} = m(P_1) \qquad \text{(C-79)}$$

2) $$P_{2,m} = m(P_2) \qquad \text{(C-80)}$$

3) $$P_3^{'} = 2P_4\left(-\frac{1}{2}+\frac{P_{1,m}}{2}\right)-4-P_{2,m}P_{1,m} \qquad \text{(C-81)}$$

4) $$P_4^{'} = \frac{P_3^{'}\left(\frac{1}{6}-\frac{P_{1,m}}{3}\right)+1+P_{1,m}\left(-2-\frac{P_{1,m}P_{2,m}}{3}\right)}{-\frac{1}{3}+P_{1,m}\left(-\frac{P_{1,m}}{3}+\frac{2}{3}\right)} \qquad \text{(C-82)}$$

5) If $P_3^{'} \subset P_3$ and $P_4^{'} \subset P_4$, then **Theorem 4.1** guarantees that there is a solution of (C-65) and of (C-70) inside $P^{'} = \left[P_{1,m} \quad P_{2,m} \quad P_3^{'} \quad P_4^{'}\right]^T$. Thus, $P^{'}$ can be used to update $\overline{J}$.

Note that (C-81) and (C-82) are not affected by dependency.

At this point, and following the suggestion given in Section 5.2.4, it was decided that P_1 and P_2 would be the parameters split during the branching phase.

Proving that $P^{'}$ is feasible with respect to the equality constraints (C-65) and (C-70) is only the first step. $P^{'}$ must also be proved feasible with respect to the inequality constraint (C-47). So sharp bounds on $x_2\left(t\right)$ could be calculated, or in other words, to reduce the dependency effect, (C-66) was rewritten as follows:

$$\begin{cases} x_2\left(T\right)=T\left(\left(\dfrac{P_3-P_2}{6P_1}T+\dfrac{P_2}{2}\right)T+1\right) & ,T=\left[0,\underline{P_1}\right] \\ x_2\left(t\right)=\left(\left(P_1\left(\dfrac{P_3}{6}+\dfrac{P_2}{3}\right)+1\right)P_1\right)\cup & \\ \qquad\cup\left(P_3\left(\dfrac{1}{6}+\left(\dfrac{P_1}{6}-\dfrac{1}{3}\right)P_1\right)+P_4\left(\dfrac{1}{3}+\left(\dfrac{P_1}{3}-\dfrac{2}{3}\right)P_1\right)-P_1+1\right) & ,t=P_1 \\ x_2\left(T\right)=\left(\left(\left(\dfrac{T}{6}-\dfrac{1}{2}\right)T+\dfrac{1}{2}\right)T-\dfrac{1}{6}\right)\dfrac{\left(P_4-P_3\right)}{\left(1-P_1\right)}+P_4\left(T\left(\dfrac{T}{2}-1\right)+\dfrac{1}{2}\right)-T+1 & ,T=\left[\overline{P_1},1\right] \end{cases} \tag{C-83}$$

where the main priority was to factor out P_3 and P_4. Since P_1 and P_2 are split during the branching phase, all the boxes should have sharper bounds on P_1 and P_2 than on P_3 and P_4. Moreover, P_1 and P_2 are replaced by thin intervals in (C-79) and (C-80). Thus, it is more important to factor out P_3 and P_4 than P_1 and P_2. In this way, sharper bounds on $x_2\left(t\right)$ can be calculated.

Factoring out P_3 and P_4 for $0\le t\le\underline{P_1}$ and $\overline{P_1}\le t\le 1$ has a problem though: the dependency due to T increases considerably. As explained in Section 5.2.2.1, to deal with this, $\left[0,\underline{P_1}\right]$ and $\left[\overline{P_1},1\right]$ are divided in N_{ineq} subintervals each.

(C-83) is used both for proving feasibility of a box $P^{'}$ and for proving infeasibility of a box P with respect to the inequality constraint.

The performance index is calculated through (C-84). The rectangular method was used to calculate the second parcel. The first and third parcels are calculated analytically via (C-85) and (C-86). P_3 and P_4 were factored out to reduce the dependency effect.

C.2.3 Comparing the algorithms

The inputs given to the developed algorithm are shown in **Table C.3**.

$$\begin{aligned} J &= \int_0^1 0.5u^2 dt = \\ &= \underbrace{\int_0^{\underline{P}_1} 0.5\left(\frac{P_3 - P_2}{P_1} t + P_2\right)^2 dt +}_{1^{st}\ parcel} \\ &\quad \underbrace{+\int_{\underline{P}_1}^{\overline{P}_1} 0.5\left(\left(\frac{P_3 - P_2}{P_1} t + P_2\right) \cup \left(\frac{P_4 - P_3}{1 - P_1}(t-1) + P_4\right)\right)^2 dt +}_{2^{nd}\ parcel} \\ &\quad \underbrace{+\int_{\overline{P}_1}^{1} 0.5\left(\frac{P_4 - P_3}{1 - P_1}(t-1) + P_4\right)^2 dt}_{3^{rd}\ parcel} \end{aligned} \qquad \text{(C-84)}$$

$$\int_0^{\underline{P}_1} 0.5\left(\frac{P_3 - P_2}{P_1} t + P_2\right)^2 dt = \frac{\underline{P}_1}{6P_1^2}\left(P_3\left(P_2\left(3P_1\underline{P}_1 - 2\underline{P}_1^{\,2}\right)\underline{P}_1^{\,2}P_3\right) + P_2^2\left(3P_1\left(P_1 - \underline{P}_1\right) + \underline{P}_1^{\,2}\right)\right) \qquad \text{(C-85)}$$

$$\int_{\overline{P}_1}^{1} 0.5\left(\frac{P_4 - P_3}{1 - P_1}(t-1) + P_4\right)^2 dt = \frac{\left(-\overline{P}_1 + 1\right)}{6\left(P_1 - 1\right)^2}\left(\begin{array}{l} P_4\left(\begin{array}{l} P_4\left(\left(3P_1 - 3\overline{P}_1 - 3\right)P_1 + \overline{P}_1^{\,2} + \overline{P}_1 + 1\right) + \\ +P_3\left(\left(3\overline{P}_1 - 3\right)P_1 - 2\overline{P}_1^{\,2} + \overline{P}_1 + 1\right) \end{array}\right) + \\ +P_3^2\left(\overline{P}_1 - 1\right)^2 \end{array}\right) \qquad \text{(C-86)}$$

Table C.3: *User inputs in* ***Example 4****.*

Designation	Symbol		Value
New boxes per old box	N_{split}		$\begin{bmatrix}2 & 2 & 0 & 0\end{bmatrix}^T$
Initial upper bound on the minimum	$\overline{J}$		$+\infty$
Number of time subintervals used for integration	N_{int}		1
Number of time subintervals used for inequality constraints	N_{ineq}		1000
Stopping criteria	ε_P	ε_{P_1}	$+\infty$
		ε_{P_2}	$+\infty$
		ε_{P_3}	$+\infty$
		ε_{P_4}	$+\infty$
	ε_J		6.3e-2
Hull consistency stopping criteria	ε_{abs}		1.0e-5
	ε_{rel}		0.05

The intended performance could only be achieved by choosing a relatively high value for N_{ineq}.

In **Table C.4**, the results produced by the developed algorithm are compared with the sharpest results presented in [Chu, 2007].

Table C.4: *Comparative results for **Example 4**.*

			The developed algorithm		Chu's algorithm	
Designation	**Symbol**		**Result**	**Width**	**Result**	**Width**
Bounds on the minimum	J^I		$[2.206, 2.256]$	4.9e-2	$[2.201, 2.265]$	6.4e-2
Bounds on the solution point	P^I	P_1^I	$[0.450, 0.550]$	1e-1	$[0.478, 0.513]$	3.4e-2
		P_2^I	$[-3.41, -3.08]$	3.2e-1	$[4.53, 5.00]$	4.7e-1
		P_3^I	$[-0.87, -0.71]$	1.5e-1	$[-3.26, -3.15]$	1.0e-1
		P_4^I	$[-3.43, -3.03]$	3.9e-1	$[-4.82, -4.32]$	4.8e-1
		P_5^I	N/A	N/A	$[1.24, 1.62]$	3.8e-1
CPU time	$\Delta t_{CPU}\,[s]$		554 (≈9m)	N/A	9994 (≈2h47m)	N/A
Number of remaining boxes	$N_{remaining}$		8345	N/A	1214	N/A

In this particular case, both algorithms correctly bound P^*. However, the algorithm developed in this thesis is approximately 18 times faster.

The following comments are in order:

1) The initial bounds on P_1 were not reduced. An experiment was made where ε_J was set to zero and the algorithm was allowed to run indefinitely. After approximately 24 hours, the width of J^I was 1.95e-2 but the bounds on P_1 were still unchanged. The reason why it is so difficult for the developed algorithm to reduce the initial bounds on P_1 is, in the author's opinion, due to the suboptimal form of (C-71) and (C-72). Ideally, these two HC equations designed for reducing the bounds on P_1 should not have linear terms of P_1 in the right hand side. However, to prevent the denominators from containing zero, these suboptimal forms had to be chosen. This makes it harder to reduce the bounds on P_1. If extended intervals analysis was available, the optimal forms could have been used.

2) No box P containing P^* can ever be proved feasible with respect to the inequality constraint. To understand why, consider the limit case: $P = P^*$. By replacing P with P^* in (C-83), one would obtain for $t = P_1^* = 0.5$:

$$x_2\left(t = P_1^* = 0.5\right) = \left[\underline{X}_2, \overline{X}_2\right] = [0.1999, 0.2001] \qquad \textit{(C-87)}$$

Since the number 0.2 cannot be exactly represented in a computer, outward rounding must be used. (C-87) is the sharpest non-thin interval containing 0.2. Since $\overline{X}_2 > 0.2$, $P = P^*$ cannot be proved feasible with respect to the inequality constraint. Moreover, due to fundamental theorem of interval analysis (see (3-42)), no box containing P^* can ever be proved feasible with respect to the inequality constraint. This makes calculating sharp bounds on J^* and P^* a lot harder.

3) Using the same inputs as before (see **Table C.3**) together with $N_{shrinking} = 100$ and $\varepsilon_{shrinking} = 0.1$, adding a shrinking phase results in an increase of the CPU time from approximately 554s to 1152s, for the same desired accuracy. Even with four parameters, it is still faster not to use a shrinking phase.

4) In **Table C.5**, Δt_{CPU} is given as a function of N_{split}. All other inputs are kept constant (see **Table C.3**).

Table C.5: *CPU time as a function of* N_{split} *in* ***Example 4****.*

Number of parameters being split	New boxes per old box N_{split}	CPU time $\Delta t_{CPU}\,[s]$
1	$\begin{bmatrix}2 & 0 & 0 & 0\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}0 & 2 & 0 & 0\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}0 & 0 & 2 & 0\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}0 & 0 & 0 & 2\end{bmatrix}^T$	No convergence after 800s
2	$\begin{bmatrix}2 & 2 & 0 & 0\end{bmatrix}^T$	554
	$\begin{bmatrix}2 & 0 & 2 & 0\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}2 & 0 & 0 & 2\end{bmatrix}^T$	481
	$\begin{bmatrix}0 & 2 & 2 & 0\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}0 & 2 & 0 & 2\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}0 & 0 & 2 & 2\end{bmatrix}^T$	No convergence after 800s
3	$\begin{bmatrix}2 & 2 & 2 & 0\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}2 & 2 & 0 & 2\end{bmatrix}^T$	595
	$\begin{bmatrix}2 & 0 & 2 & 2\end{bmatrix}^T$	No convergence after 800s
	$\begin{bmatrix}0 & 2 & 2 & 2\end{bmatrix}^T$	No convergence after 800s
4	$\begin{bmatrix}2 & 2 & 2 & 2\end{bmatrix}^T$	No convergence after 800s

As predicted by rule of thumb (5-41), splitting two parameters yields the fastest results.

On the other hand, in Section 5.2.4, it is suggested that, in this case, P_1 and P_2 would be the best parameters to split, as a result of (C-79) and (C-80). However, **Table C.5** shows that the best results are obtained by splitting P_1 and P_4. Why this is the case is hard to say. Nevertheless, dividing P_1 and P_2 is still a good initial guess.

5) The reader might have noticed that in **Table C.4**, for the first time, the guaranteed bounds on the final state are not given. In the previous examples, non-thin bounds on the final state meant, apart from dependency, that the equality constraints could not be exactly satisfied.

However, in this example, since the final state is used to determine B and D in (C-61) and (C-66), the only consequence of not exactly satisfying (C-65) and (C-70) will be that (C-64) and (C-69) will also not be exactly satisfied. Therefore, it would not be fair to compare the bounds on the final state obtained in this thesis with the bounds given in [Chu, 2007] (since the final state might be calculated in a different way). More details about this are given at the end of **Example 5**.