Hybrid Differential Dynamic Programming Algorithm for Low-Thrust Trajectory Design Using Exact High-Order Transition Maps

Msc. Thesis of:
Michele Maestrini
Registration No 852579

Supervisors:
Dr. Pierluigi Di Lizia PhD.
Politecnico di Milano
Dr. Roberto Armellin PhD.
University of Surrey

2017-2018
Abstract

Optimal orbital trajectories are obtained through the solution of highly nonlinear large scale problems. In the case of low-thrust propulsion applications, the spacecraft benefits from high specific impulses and, hence, greater payload mass. However, these missions require a high count of orbital revolutions and, therefore, display augmented sensitivity to many disturbances. Solutions to such problems can be tackled via a discrete approach, using optimal feedback control laws. Historically, differential dynamic programming has shown outstanding results in tackling these problems. A state of the art software that implements a variation of DDP has been developed by Whiffen [1] and is used by NASA’s Dawn mission. One of the latest techniques implemented to deal with these discrete constrained optimizations is the Hybrid Differential Dynamic Programming (HDDP) algorithm, introduced by Lantoine and Russell [2]. This method complements the reliability and efficiency of classic nonlinear programming techniques with the robustness to poor initial guesses and the reduced computational effort of DDP. The key feature of the algorithm is the exploitation of a second order state transition matrix procedure to propagate the needed partials, decoupling the dynamics from the optimization. In doing so, it renders the integration of dynamical equations suitable for parallelization. Together with the possibility to treat constrained problems, this represents the greatest improvement of classic DDP. Nevertheless, the major limitation of this approach is the high computational cost to evaluate the required state transition matrices. Analytical derivatives, when available, have shown a significant reduction in the computational cost and time for HDDP application.

This work applies differential algebra to HDDP to cope with this limitation. In particular, differential algebra is introduced to obtain state transition matrices as polynomial maps. These maps come directly from the integration of the dynamics of the system, removing the dedicated algorithmic step and reducing its computational cost. Moreover, by operating on polynomial maps, all the solutions of local optimization problems are treated through differential algebraic techniques. This approach allows us to deal with higher order expansions of the cost, without modifying the algorithm. The leading assumption of this work is that, treating higher than second order expansions, grants larger radii of convergence for the algorithm, improved robustness to initial guesses, hence faster rates of convergence. Examples are presented in this thesis to assess the performance of the newly constructed algorithm and to test the assumptions.
Contents

1 Introduction
   1.1 Low-Thrust Trajectory Design .................................. 1
   1.2 Literature Review ............................................. 2
       1.2.1 Indirect Methods ......................................... 2
       1.2.2 Direct Methods ........................................... 3
       1.2.3 Constrained Optimization ................................. 4
       1.2.4 Differential Dynamic Programming ..................... 6
       1.2.5 Empirical Control Laws .................................. 8
       1.2.6 Robust Control ........................................... 8
       1.2.7 Differential Algebra ..................................... 11
   1.3 Outline of This Work ........................................... 12

2 Theoretical Background ............................................ 13
   2.1 Mathematical Notation ......................................... 13
   2.2 Differential Dynamic Programming ............................ 15
       2.2.1 Backward Induction ...................................... 16
       2.2.2 Forward Propagation ..................................... 17
       2.2.3 Application to Differential Dynamic Programming ...... 17
       2.2.4 Limitations of Differential Dynamic Programming ...... 19
   2.3 Hybrid Differential Dynamic Programming .................... 20
       2.3.1 Problem Reformulation ................................... 20
       2.3.2 Quadratic Expansions ..................................... 21
       2.3.3 State Transition Matrix Approach ......................... 23
       2.3.4 Stage Optimization ....................................... 26
       2.3.5 Phase Optimization ....................................... 32
       2.3.6 Termination ................................................ 34
       2.3.7 Overview of the algorithm ................................. 34
3 Methodology
3.1 Differential Algebra .............................................. 37
  3.1.1 $D_1$ Differential Algebra ................................. 39
  3.1.2 $D_n$ General Case ........................................... 40
3.2 HDDP Based on Polynomial Maps ................................. 41
  3.2.1 Backward Sweep on Stages ................................. 43
  3.2.2 Constraints Handling Techniques .......................... 45
  3.2.3 Backward Sweep on Phases ................................. 45
  3.2.4 Trust Region Algorithm ...................................... 47

4 Applications ......................................................... 49
4.1 Linear Quadratic Problem ........................................ 49
  4.1.1 Results ...................................................... 50
4.2 Mono-dimensional Landing ....................................... 52
  4.2.1 Results ...................................................... 52
4.3 Earth-Mars Transfer .............................................. 56
  4.3.1 Results ...................................................... 57
4.4 Satellite Constellation Refueling ............................... 62
  4.4.1 Results ...................................................... 65

5 Conclusions .......................................................... 73

Bibliography ............................................................. 79

A Tuning Parameters .................................................. 89

B Differential Algebra Example ...................................... 93

C Computational Times ............................................... 95
1

Introduction

1.1 Low-Thrust Trajectory Design

Low thrust optimal control has found growing interest among researchers and practitioners thanks to its remarkable mass savings. With respect to chemical propulsion, ion-thrusters allow for higher specific impulses, yielding greater useful mass. For example, a solar sail mission could reduce the propellant mass stored virtually to zero, as the thrust is provided by solar wind.

The downsides in the design of these methods are twofold. First of all, they increase the size of the problem that needs to be solved due to the large amount of decision variables that must be chosen. Secondly, it causes an accumulation of errors due to the large actuation times needed to produce significant delta-V’s [3].

The aim of this thesis is to add to the current literature regarding optimal control for low-thrust trajectory design. In order to do so, the focus is placed on implementing Differential Algebraic techniques for the treatment of a well validated DDP-based technique: HDDP[2].

The work starts by addressing the question whether higher than second order expansions could lead to an improvement on convergence speed of the algorithm. This enhancement in speed can be seen as an augmented convergence region of the algorithm. In particular, HDDP is based on a quadratic trust region, whereas this work an attempts at using higher orders. The increase in order of the expansion allows also for the evaluation of nonlinear optimal feedback.

To these purposes, this dissertation starts by replicating HDDP algorithm on MATLAB® (preliminary) and C++ (final). The C++ code is improved by the addition of Differential Algebraic techniques to modify the standard HDDP and
obtain the desired results thanks to the software DACE.

Four test cases are provided at the end of this work in order of growing complexity: I) a validation Linear-Quadratic problem; II) a Mono-Dimensional Landing problem; III) an interplanetary Earth-Mars Transfer and, to conclude, IV) a Satellite Constellation Refueling Problem.

From these examples, it emerges that increasing the order of the expansions and of the feedback law does not yield a better convergence. Additionally, it causes numerical instability of the algorithm to arise, as well as a noticeable increase on computational time due to the number of polynomial coefficients that ought to be computed at each iteration.

Besides this problem, the thesis tries to implement an easier way of dealing with partial derivatives propagation necessary for HDDP. In this regard, the use of DA allows the user to avoid the derivation of analytical partials necessary for the State Transition Matrix approach implemented in HDDP.

The expected improvement of the algorithm is tested on the Satellite Constellation Refueling case. The dynamic model for the system is modified by adding the effect of Earth’s zonal harmonic perturbation $J_2$ to the system of dynamical equations. The algorithms achieves convergence without need for the user to compute and assemble complicated partials via a symbolic external software (e.g. MAPLE®).

Overall, this work provides a first attempt at the application of Differential Algebraic techniques in the Differential Dynamic Programming framework. By doing so, it creates the first building block for further researches in this field. Moreover, it adds to the current Differential Dynamic Programming literature by exploiting high order nonlinear optimal feedback controls, also dealing with constraints. Furthermore, it tries to improve the user’s capability of exploiting HDDP software by removing the tedious step of obtaining partials from symbolic external software and interfacing it with Fortan/C code.

This thesis starts by reviewing the literature and the current state of the art in optimal control to explain the motivations that lead to the introduction of DA in this field. Subsequently, an explanation of DA and the methodology of its application to HDDP is provided. To conclude, an analysis of the application cases is performed and the consequent conclusions are drawn.

1.2 Literature Review

In the past, several approaches have been implemented to find optimal trajectories. Among the many solutions, particularly challenging is the case of low-thrust propulsion. Approaches to these optimization challenges are classically divided in two main categories: indirect methods and direct methods [4].

1.2.1 Indirect Methods

These methods rely on calculus of variation or Pontryagin’s Maximum Principle [5] to retrieve the necessary optimality conditions. The problem is then reduced
Chapter 1. Introduction

to a two point boundary value problem (TPBVP) and solved. Indirect approaches introduce additional states, the so called 'co-states' (or adjoint states), that have little physical meaning most of the time. Additionally, the equations and gradients of co-states, needed to retrieve necessary conditions for optimality, are not easily formulated. Therefore, one of the main disadvantages of indirect methods, is the necessity of the user to have deep knowledge of the problem [5]. Their solution is obtained when, for each time instant, the adjoint states (Lagrange multipliers) extremize the Hamiltonian. Moreover, the boundary conditions together with the dynamics of state and adjoint states must be satisfied. The classical approaches to solution of these problems include single-shooting, multiple-shooting, finite differences and collocation techniques.

Single shooting methods solve a TPBVP with an first guess on the initial conditions, then this trial guess is updated as a function of the final error, details can be found in [6]. Keeping this in mind, it is clear that a first guess on Lagrange multipliers is also necessary, subsequently this guess is evaluated and iterated until optimality is achieved. The major downside of these techniques is that the first guess on Lagrange multipliers is not always obvious and the solution of the problem is strongly affected by it, as shown in [4],[7],[8].

Multiple shooting techniques attempt to solve the TPBVP by splitting the time interval in a succession of subintervals. Then, single-shooting is applied between the subintervals and continuity is then ensured via linking constraints. The main advantage of multiple shooting, is the reduction of sensitivity to bad initial conditions and the possibility to parallelize the computations, being the subintervals independent as shown by Betts and Huffman [9]. On the other hand, the number of decision variables increases quite rapidly with the number of subintervals, causing an increase of computational effort.

For many applications, also Finite Differences approaches are broadly used. These methods rely on a discretization of the domain in a grid of points, that provide an approximation of partial derivatives through finite differences schemes between these discretization sites. Hence, the values of variables on the grid points become the unknowns, and the solution is usually obtained by solving a linear system of equations (implicit or explicit), an overview of this method can be found in [10]. Finally, collocation techniques do not require integration of states and co-states. In these techniques, the time span is divided in subintervals, one at each collocation point, and simple functions are used to parametrize the unknowns at each segment as explained by Russell and Shampine in [11]. The typical choice for such functions are piecewise polynomials, whose coefficients become the new unknowns of the problem.

1.2.2 Direct Methods

In direct methods, the problem is reformulated from an optimal control problem to a nonlinear (often quadratic) programming problem. To reformulate it as NLP problem, the optimal control problem is discretized and parametrized with polynomials or other functions whose coefficients become the new decision variables. The idea behind this method, is that the research for the approximating function of the solution is restricted in a finite dimensional space of functions. Such ap-
proximations are usually piecewise polynomials as explained by Von Stryck and Bulirsch in [5]. Usually, direct methods rely on physical quantities rather than abstract Lagrange multipliers, and their solutions are only approximated. The reformulation is based on a selected decision vector and an iterative procedure which adjusts it during iterations until all convergence criteria are met. All these NLP methods can be solved provided that first order derivatives of the cost and of the constraints are given, as shown in [12],[13]. Some of the softwares that implement such methods, offer the possibility to exploit also informations on the second order derivatives (e.g. IPOPT [12]). The key step of the NLP procedures is the assembly of a Hessian containing all the second order pure and mixed partial derivatives of the cost function with respect to the decision variables. After it is assembled, the Hessian is inverted, however the size of the Hessian grows rapidly with the number of decision variables as explained in [2]. In fact, decision vectors for classic NLP increase linearly with the number of discretization points, while the size of the Hessian grows with the square of the discretized variables (sometimes with the cube). The proof of this trend is provided by Murray and Yakowitz [14]. This causes an increase in computational effort which is two fold. First, inverting the sparse Hessian matrix is notoriously computationally expensive. This happens even when tailored solutions to the particular sparse structure of the problem are applied, an example of these strategies is given by Franke in his software OMUSES [15]. Secondly, the process of assembling large Hessians is expensive per se. This is owed to the repeated chain rule applications needed to compute all the necessary sensitivities with respect to decision variables, and it was noted by Lantoine in [16]. Solution to this problem has been nonetheless found, for example by using the software Sparse Optimization Suite implemented by Betts [17],[18]. Regardless their broad usage, direct methods are in general less accurate than the indirect ones, and they may converge to a local minimum. In fact, discretized optimal control problems usually present several of them as shown by Kraft in [19].

The main advantage of these direct methods over indirect ones, is the reduced dependence of the solution on initial choice of adjoint variables. This allows for less expert users to tackle the problems, as reduced insight is needed.

An additional note should be made regarding Differential dynamic programming. This method is classified as a direct one, but if implemented in a first order version, it actually retrieves the same equations of calculus of variations, minimizing the Hamiltonian at each iteration. Proof of this has been provided by Dreyfus [20] as well as Bryson [21]. This is why DDP represents a link between direct and indirect methods: the necessary conditions for optimality may not be formulated by DDP as they were for indirect methods, however its solution is influenced by them.

1.2.3 Constrained Optimization

Methods requiring constraint handling, introduce Lagrange multipliers in their cost and new conditions necessary for optimality, i.e. Karush-Kuhn-Tucker condi-
Chapter 1. Introduction

These conditions make it possible to treat linear constraint, therefore nonlinear constraints must be linearized in order to be addressed. The typical solution to these equality constrained problems, are based on elimination methods (also called primal methods) whose description can be found in [23]. These approaches restrict the number of inputs in a subset, so that the optimization problem to be solved becomes unconstrained. The optimal solution is then found in this subset which is compliant with the constraints. Such methods are for example the null-space method and the range space method as described by Fletcher [24]. The main idea behind these methods comes from the projected gradient method developed by Rosen [25], which divides the decision variables in subset of dependent and independent. The independent variables are used for optimization while the dependent variables are adjusted so that they comply with the constraints. The approach of linearizing the constraints is widely used also in methods where the Lagrangian function has a quadratic order expansion such as the Sequential Quadratic Programming methods developed by Bazarra et al. [26]. Originally, their convergence properties were tested by Hann [27] and Powell [28]. An overview of such procedures can be found in Barclay et al. [29], but also modern solvers such as KNITRO [30] and SNOPT [13] use them. The requirement for these constrained problems is that the iterates should be feasible, which is not always possible due to the nonlinearity of the constraints that are only linearly approximated. Some methods implement a quadratic approximation of the constraints as in the work of Patel and Scheeres [31], but they require also quadratic control feedback implementation.

Another category of constraint handling procedures is Penalty Methods, which are only approximate solutions to the optimization problem. They include in the objective function a penalty term for the violation of constraints. In this way, the problem is treated as unconstrained and the quality of the constraint satisfaction is dictated by the entity of the penalty. Usual penalty functions are quadratic, as in Courant [32], but many other solutions have been found using different shapes. However, penalty methods require the penalty term to be raised to infinity to give exactly zero constraint violation. This characteristic of penalty methods, results in conditioning problems of the Hessian matrix as reported by Murray [33].

The methods described up until this point are particularly useful for equality constrained problems. However, the necessity to accurately treat also inequality constraints, has lead to many other solution approaches.

A first approach, is to add slack variables so that an inequality can be transformed into an equality, adding to the set of tuning parameters of the problem.

The second approach is, instead, to use an active set of constraints to determine whether a constraint is active or not during optimization as in [34],[24]. This second category is employed in SNOPT [13] and KNITRO [30].

An additional technique is the interior point method, which uses a strategy for enforcing constraints at each iteration. They include a function in the objective to be minimized which makes the cost go to infinity when the constraint is violated. Typical shapes for such functions are logarithmic as in Frisch [35] or inverse as in Carroll [36]. As one can imagine, this technique suffers from similar ill conditioning issues of penalty methods. Nonetheless, it is used also by IPOPT [12] and
Finally, an Augmented Lagrangian method can be used. This strategies were introduced by Powell [37] and Hestenes [38] and constitute a hybrid between the Lagrangian approach and the penalty function method. The gist of this method is that the Lagrangian is augmented with a penalty function which increases with the violation of constraints. The introduction of the Lagrange multipliers solves the conditioning problem posed by pure penalty methods, as shown by Bertsekas [39]. The strategy for solving these problems is by primal-dual approach. This means dividing the problem into an inner loop and an outer loop. The inner loop, is where the decision variables are updated to minimize the cost function. Whereas, the outer loop updates the Lagrange multipliers, after the internal loop has converged, to maximize the cost function. For this reason, these problems are known also as minimax problems and their formulation can be found in Bertsekas [39]. The convergence of the algorithm depends on the convergence of the inner loop as well as the convergence of the outer loop. Therefore, it is paramount to have a correct Lagrange multipliers update formulation. This update procedure, can be linear as well as nonlinear, as the update technique implemented by Lantoine and Russell in [2]. The inner loop rigorous convergence is not necessary for the complete algorithm to converge and some problems showed convergence also when the inner loop is only approximately solved as in [40]. To conclude, Bertsekas [39], suggests that augmented Lagrangian techniques could be superior to active set methods in treating path nonlinear constraints, as the determination of violation of such constraints can be difficult.

Even if historically these three approaches have been treated as different techniques, it has been proven by Giannessi [41] that they are only special cases of a broader approach named Image Space Analysis. The prevalence of one of these methods on another is yet to be proven, however the tendency of penalty function method to increase nonlinearity and slow down convergence rates is well known and documented by Bertsekas [39].

An improved method, based on a mixed approach of augmented Lagrangian and range space active set method, was implemented by Yakowitz [42] and applied to a multi-reservoir problem in his co-joined work with Murray [34]. This last method is reprised by Lantoine and Russell in HDDP [2].

### 1.2.4 Differential Dynamic Programming

To solve larger problems with an admissible amount of computational power, Differential Dynamic Programming (DDP) was introduced. DDP is a technique available for problems that can be reformulated as a dynamic optimization problem [43]. What DDP does is discretizing the optimization into smaller subproblems, dividing the decision vector in smaller sub vectors that only influence local and future optimization steps. Optimizing for each stage on the control variables of that stage, yields a linear scaling of the problem with the number of control variables, as proven by Jacobson and Mayne in [44]. To sum up, if the problem is discretized in $N$ stages, each with $m$ controls, a classic NLP method would have to invert an $(N \cdot m)^2$ Hessian matrix. On the other hand, DDP only requires $N$ times the inversion of a $m^2$ matrix. Proof of this linearly increasing computational cost
can be found in Liao and Shoemaker [45]. The founding assumption of DDP is Bellman’s Principle of Optimality (developed for Dynamic Programming), noting that discrete optimization can be seen as a series of optimal decisions made over time. The principle reads:

"An optimal policy has the property that whatever the initial states and initial decisions are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." [46]

Continuous time DP theory links Bellman’s Principle to Hamilton-Jacobi-Bellman equations, i.e. a system of first order spatial-temporal PDE equivalent to Pontryagin’s Maximum Principle whose discretized counterpart is the Bellman equation. The way these problems are solved is by optimizing the subproblems backwards in a process called "backward induction" until the first point is reached [47]. This solution method allows for the computation of an optimal feedback control law from the trajectory. The issue emerging from this solution is that the algorithm generates a field of extremals, which makes it possible to know all the optimal control strategies and all the costs for any initial conditions. Unfortunately, this requires prohibitive amount of computational power/storage, hence generating the so-called "curse of dimensionality".

DDP is, therefore, a development of pure Dynamic Programming to overcome the inherent "curse of dimensionality". The method is based on a quadratic expansion of the cost in the neighborhood of a reference trajectory, effectively rendering the minimization a localized problem, hence sacrificing globality. The reference trajectory is then updated iteratively and so are its second order expansions. A feedback law, based on the perturbations of states, is generated during a process called "backward-sweep" so as to improve the following iterate. Instead, during the "forward-sweep", the dynamics of the system subject to this feedback law is re-computed. Analytical quadratic expansions, moreover, allows for improved convergence with respect to previous linear problems or to numerically approximated second order problems.

However, DDP is mainly suitable for smooth, unconstrained problems. This is, in fact, the original form in which it was formulated by Jacobson and Mayne [44]. Several attempts were made at applying DDP also to constrained problems, and an overview of the spectrum of different approaches is presented by Yakowitz in [48]. Fundamentals contributions exploited Lagrangian multiplier methods, e.g. [49],[50], and augmented Lagrangian method as [51], as well as the Penalty method [52].

To treat constraints with DDP, modern methods have started to include well validated NLP techniques in DDP, for example in the work of Lantoine and Russell [2],[53]. The state of the art in low-thrust optimal control is Mystic software developed by Whiffen [1] and it is based on a DDP variant developed by the same author [52]. Mystic exploits an Hessian shifting technique to enforce convexity of the optimization problems, and has a penalty function for treating constraints. If the computational time of the algorithm exceeds the maximum allowed value, then a prescribed low-thrust control law (Q-law) is used.
Another method that relies on DDP is Hybrid Differential Dynamic Programming [54],[55],[2],[53] developed by Lantoine and Russell. This is a multi-phase, multi-stage method which uses augmented Lagrangian technique to treat phase constraints. On the other hand, each stage is constrained via null-space methods directly during optimization. Moreover, instead of a generic Hessian shifting technique, HDDP implements a trust region algorithm to guarantee boundedness of the solution and convexity of the cost function. Being this algorithm multiple-phase but non multiple-shooting, the subproblems are solved in a succession and not independently in parallel.

The attempt at applying multiple shooting principles to HDDP was made by Pellegini in his Ph.D. dissertation [56], where HDDP was applied to the single legs of the multiple shooting problem.

### 1.2.5 Empirical Control Laws

Besides direct and indirect solutions of the optimal control problem, also a third option is available: Empirical control laws. These are sub-optimal solution to the problem which exploit heuristics to reduce the complexity of the solution. Usually these kinds of laws are Lyapunov feedback control laws (to guarantee stability) and they must be carefully calibrated with heuristics to guarantee a (sub-optimal) solution.

Many versions of these problems have been introduced in the past, some include possibility of coasting periods. In order to do that, Petropolous [57] introduced a mechanism for estimating the optimal orbital position to change each of the classical orbital parameters based on effectivity of the maneuver. This algorithm is particularly important because its development lead to the implementation of the Q-Law [58]. Control values are obtained in a way that maximizes the Q-function's reduction. Indeed, Q is a function that captures the proximity to the target orbit.

Empirical control laws are particularly important for real mission applications. In fact, the assumption in optimal control solutions presented until this point, is that these laws must rely on exact dynamics, but the reality introduces various disturbances. Nonetheless, the system must be able to complete the mission under the presence of uncertainties. Taking this into consideration, a method called "constraint tightening" is usually applied as described by Rayman et al. in [59]. The main idea behind this technique, consists in retaining a "margin" during the optimization procedure, to guarantee the budget for future corrections.

The main advantage of these empirical laws is that, even if they represent sub-optimal solutions, they can yield excellent initial guesses for indirect optimization.

### 1.2.6 Robust Control

The application of heuristics to retain a margin is not always desirable. The control law obtained by neglecting disturbances, is generally different from the one including them. Especially in space applications, the uncertainties are many and they have a great impact on the mission’s outcome. The impact of unmodeled dynamical perturbations, as well as the scarce knowledge of the system initial
state, produces a very rapid error accumulation in the trajectory. Together with the accumulation of uncertainties on system parameters, non-deterministic constraints and terminal condition can be detrimental to the mission’s reference path as explained in [3]. Instead of using empirical margins to estimate the propellant necessary for corrections, there has been a growing interest in a robust optimal low-thrust trajectory design strategy. The main benefit coming from this technique would be the possibility to reduce the modeling cost for small satellites, allowing for some errors, as explained by Ozaki et al. in [60]. This would reduce the cost for research rendering it more accessible.

Classical approaches to the solution of optimization problems under uncertainties, rely on the assumption of linearity both of dynamics and feedback control. Orbital mechanics applications of such problems are extensively found in the literature, e.g. [61]. A significant attempt at optimizing with uncertainties was made by Olympio and Yam [62]. In their work, they developed a method based on indirect approaches which only focuses on "temporary" disturbances, in particular one engine failure.

Alternative solutions can be obtained again under the linearity assumption as explained by Diehl and Bjornberg in [63] by exploiting Model Predictive Control. To face also nonlinear problems, Theodorou et al. [64] have exploited Stochastic Differential Dynamic Programming (SDDP) to minimize the expected value of the cost function. Their work manages to make the DDP algorithm robust to model uncertainties and varying noise covariance, as it is already insensitive to additive fixed noise when the model is deterministic. Proof of this has been given by Jacobson and Mayne and can be found in [44]. In a previous work by Todorov and Tassa [65], this result was obtained for linearized systems, however DDP exploits a quadratic expansion of the cost, therefore the method is modified accordingly. Their approach was reprised also by Ozaki et al. [60] by mapping the expected value of the cost to go function using the unscented transform described in [66], and applying SDDP to minimize it.

A significant contribution to the field of robust optimal control has been given in the field of semi-analytical nonlinear control techniques. With these approaches, it is possible to treat uncertainties on initial and final states, uncertainties on model parameters and uncertainties on constraints and dynamical models. More often than not, in fact, the target conditions are not known a priori, and only to a certain degree of accuracy, e.g. by Vasile in [67]. These techniques rely on nonlinear optimal feedback control to guarantee that highly nonlinear dynamics such as orbital motion can be treated effectively. These laws require a solution of the Hamilton-Jacobi-Bellmann equation formulated in [47]. Being the HJB equation a nonlinear first-order PDE, it is difficult to solve it for practical applications. Numerous attempts at approximating the solution have been made, so that a sub-optimal nonlinear control feedback could be retrieved.

Bonnard and Caillau tried to give an alternative approach to solution of the equation in [68]. The problem is reformulated as a two-point boundary value problem and subsequently solved by numerical iteration.

In Bryson and Ho [44], the HJB is solved approximately by expanding the objec-
ative function to second order around a reference trajectory, the so called "field of extremals".

Relevant contributions have been obtained in the case of infinite horizon control, the most notable one being the State Dependent Riccati Equations approach detailed by Beeler [69]. The basic idea of this method is that the system is reformulated as "pseudo-linear", rendering the dynamic matrices dependent on current states. Then a quadratic formulation of the cost is implemented. Finally, the solution is retrieved by solving iteratively a Riccati equation on-line evaluating the matrices at the current states. However, this method may prove to be too computationally expensive for large problems.

As far as finite horizon approaches are concerned, there is paucity of studies. Although, some relevant results were obtained by Fax and Murray [70] for time scalable systems. With the proper scaling they obtained a reformulation of the HJB equation from spatial-temporal PDE to only spatial PDE. The solution of the PDE is then retrieved at a particular time instant, and then scaled back to any desired time.

Finally, a new set of optimization techniques is introduced. They do not rely on any statistical hypothesis on the uncertainties and they belong to the so-called semi-analytic methods. They try to add to this debate by introducing new mathematical tools to treat uncertainties.

One of the possible approaches is Interval Analysis, a mathematical tool developed by Moore [71]. The main idea of Interval Analysis is the substitution of real numbers with intervals of real numbers. Arithmetic and analysis of intervals are substituted to the usual ones acting on real numbers. Therefore, acting on the interval of all possible initial values will yield all possible final values as outputs. This technique can be exploited to propagate effectively errors and uncertainties, but it may as well result in an artificially high overestimation of the solution. This happens when the dynamical equations are numerically integrated, and it is the result of the "wrapping effect" described in [72].

Following this train of thoughts has lead to the development of a substitute for interval analysis that still retains the idea of propagating more than a single state vector. Differential algebra (DA) guarantees such an instrument.

Application examples in orbital mechanics can be found in Di Lizia et al. [73],[74]. DA can be used to efficiently treat optimal control problems. After reducing the optimization problem to a two-point boundary value problem, DA is used to expand the solution of the optimal problem with respect to initial and terminal conditions about a reference trajectory. In doing so, the optimal trajectory and feedback law are computed only once as polynomial maps, and the retrieval of the optimal feedback control policy is obtained as simple polynomial evaluation when different final or initial conditions are imposed, this has been shown in [74]. DA is, furthermore, used to manage saturation constraints and switching times of controls, once again as simple feedback evaluations of polynomials depending on local offset from the reference trajectory. Everything is valid in a rather large neighborhood of the reference trajectory and the feedback laws obtained with this approach can be evaluated to an arbitrarily high order as demonstrated in [74]. Successful applications of DA include the possibility of studying Mars aerocap-
ture techniques with an uncertain aerodynamic model of the spacecraft and of the atmosphere [75], application to three body problem [76] and finally asteroid encounters considering uncertain initial conditions derived from orbital determination [77].

1.2.7 Differential Algebra

In this framework, an introduction to differential algebra is paramount for the following work. The reason why DA was invented in the first place, was to obtain the solution to analytical problems applying algebraic techniques [78]. The field saw its start in Liouville’s work [79] and was significantly improved by Ritt [80] who obtained a complete algebraic theory for particular kind of differential equations. The algorithmic aspect of DA was treated in particular by Risch [81].

DA has found success in many application fields, differential algebraic equations being worth mentioning [82]. The application of DA was extended by Berz [83] for the solution of differential equations and partial differential equations. In particular, he applied the technique to expand differential equations in terms of initial conditions up to arbitrary order.

DA’s founding assumption is that it is possible to convey more information about functions than just their values at specific points, being this extra information the Taylor expansion of the function at a certain location up to arbitrary order. Historically, numerical treatment of functions was, in fact, based on floating point arithmetic operations at some specific evaluation points. To achieve the same goal for functions, computer programs were developed, so that they could implement operations between them in a similar way as they are implemented for real numbers. The key feature of DA is in fact its ability to efficiently represent functions in a computer environment in a way that they can be easily manipulated via usual arithmetic expressions. For each operation defined between functions, an equivalent one is coded to act on their Taylor expansions. In this way, the Taylor expansion of the result of an operation between functions is equivalent to applying a corresponding operator to the operands and vice versa.

The computer implementation of differential algebra, allows for the computation of a function’s values and its Taylor expansion coefficients up to an arbitrary order with a fixed amount of computational effort. Using the coefficients of the Taylor expansion of summand and factors (of product and scalar product operators), it is possible to obtain the coefficients of the function resulting from the operation. This makes it possible to define a truncated power series algebra.

Similarly to floating points numbers, elementary operators and algorithm to perform more complex tasks can be implemented in computer environment [78]. In particular: function compositions, inversions and common trigonometric functions treatment can be introduced. To complete the purpose of DA also integration and derivation operators were introduced by Berz [78]. This two operators allowed for precise extraction of arbitrarily high order multi-variable derivatives overcoming the difficulties encountered by classic numerical differentiation. Computer implementation of the algorithms was realized by Berz and Makino in the software COSY-Infinity [84].
1.3 Outline of This Work

In this work, the application of DA for the improvement of some aspects of HDDP is studied. DA is exploited for its ability to retrieve precise high order derivatives of functions along with the functions values. This properties are exploited to propagate the needed partials of the cost function and to obtain feedback laws as inversion of polynomial maps at arbitrarily high order. Additionally, this allows to consider perturbed dynamics with little to no effort.

In chapter 2, an overview of the necessary mathematical tools, is given. In particular, more details are given as far as DDP is concerned in section 2.2. Then, HDDP is presented into more detail in section 2.3. Indeed, some non trivial aspects regarding State Transition Matrices propagation are presented in section 2.3.3, and more details on the algorithm for trust region quadratic problem solution are provided in section 2.3.4.

After the theoretical setup is finished, the methodology is introduced in chapter 3, in which an introduction to details and advantages of DA is provided in section 3.1 as extracted from [78]. To conclude this chapter, a clear explanation of the application of DA techniques in HDDP is provided in section 3.2.

Following this chapter, example applications are provided in chapter 4, where models and their results are detailed. Finally, the analysis of the results is assessed in chapter 5, where further research directions are also suggested.
Theoretical Background

To introduce the developments of this work, a complete overview of the mathematical instruments exploited is given in the following sections. This include a brief overview of the standard DDP method that lead to the HDDP method exploited in this work.

2.1 Mathematical Notation

To introduce the developments of this work, an exhaustive overview of the mathematical instruments exploited is herein provided. First, a definition of all the used variables according to the notation of Lantoine and Russell [2] is provided. This is owed to the necessity of using only one notation to treat all the following problems. In fact, a general case of multi-stage, multi-phase problem is described. The number of phases is $M$ and the corresponding index is $i$ and the number of stages, which can be different for each phase, is $N_i$ with corresponding index $j$. This way of dividing the problem is characteristic of multiple shooting techniques [5]. Clearly this approach can be adapted for orbital mechanics problems. Stage constraints could be seen as boundaries on control magnitudes, while phase constraints are representative of targets. The structure itself of various phases could represent a multi-target mission, where also the dynamical equations change accordingly with the phase e.g. [53]. The problem structure is illustrated in Figure 2.1.

A list of symbols that will be used later on this chapter is here reported.

$x_{i,j} \in \mathbb{R}^{n_{x,i}}$ states of dimension $\mathbb{R}^{n_{x,i}}$ at phase $i$ and stage $j$;
2.1. Mathematical Notation

Figure 2.1: Basic formulation of the Multiphase Problem, adapted from [2].

\[ u_{i,j} \in \mathbb{R}^{n_{u,i}} \text{ dynamic controls of dimension } \mathbb{R}^{n_{u,i}} \text{ at phase } i \text{ and stage } j; \]

\[ w_i \in \mathbb{R}^{n_{\omega,i}} \text{ static controls of dimension } \mathbb{R}^{n_{\omega,i}} \text{ at phase } i \text{ and stage } j; \]

\[ \Gamma_i : \mathbb{R}^{n_{\omega,i}} \rightarrow \mathbb{R}^{n_{x,i}} \text{ functions describing initial states of phase } i; \]

\[ F_{i,j} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R}^{n_{\omega,i}} \rightarrow \mathbb{R}^{n_{x,i}} \text{ transition functions that propagate the states } \]

\[ \text{from stage } j \text{ to stage } j+1 \text{ of phase } i; \]

\[ L_{i,j} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R}^{n_{\omega,i}} \rightarrow \mathbb{R} \text{ stage cost functions at phase } i \text{ and stage } j; \]

\[ g_{i,j} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R}^{n_{\omega,i}} \rightarrow \mathbb{R}^{n_{g,i}} \text{ stage constraints at phase } i \text{ and stage } j; \]

\[ \psi_{i,j} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{\omega,i}} \times \mathbb{R}^{n_{x,i+1}} \times \mathbb{R}^{n_{\omega,i+1}} \rightarrow \mathbb{R}^{n_{\psi,i}} \text{ boundary phase constraints between } \]

\[ \text{phase } i \text{ and } i+1 \]

\[ \phi_{i,j} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{\omega,i}} \times \mathbb{R}^{n_{x,i+1}} \times \mathbb{R}^{n_{\omega,i+1}} \rightarrow \mathbb{R}^{n_{\phi,i}} \text{ terminal cost of phase between phase } \]

\[ i \text{ phase } i+1 \]

Therefore, it is possible to reformulate the full problem. The function that needs
to be minimized is expressed as Equation 2.1.
\[ J := \sum_{i=1}^{M} \left[ \sum_{j=1}^{N_i} \left( L_{i,j}(x_{i,j}, u_{i,j}, \omega_i) \right) + \phi_i(x_{i,N_i+1}, \omega_i, x_{i+1,1}, \omega_{i+1}) \right]; \] (2.1)

The problem must minimize Equation 2.1 while subject to constraints on the stages as Equation 2.4 and on the phases as Equation 2.5. The dynamical motion is described by Equation 2.3 with initial conditions derived from Equation 2.2; the controls (static and dynamic) have bounds described as in Equation 2.6 to 2.7.

\[ x_{i,1} = \Gamma_i(\omega_i); \] (2.2)

\[ x_{i,j+1} = F_{i,j}(x_{i,j}, u_{i,j}, \omega_i); \] (2.3)

\[ g_{i,j}(x_{i,j}, u_{i,j}, \omega_i) \leq 0; \] (2.4)

\[ \psi_i(x_{i,N_i+1}, \omega_i, x_{i+1,1}, \omega_{i+1}) = 0; \] (2.5)

\[ u_{i,j}^L \leq u_{i,j} \leq u_{i,j}^U; \] (2.6)

\[ \omega_{i,j}^L \leq \omega_{i,j} \leq \omega_{i,j}^U. \] (2.7)

The nature of Equation 2.3 corresponds to Equation 2.8.
\[ F_{i,j} = x_{i,j} + \int_{t_{i,j}}^{t_{i,j+1}} f_{i,j}(x, u_{i,j}, t) dt; \] (2.8)

The nonlinear dynamics of the system is represented by equation \( f_{i,j} \) that needs to be integrated in time using some integration scheme. Historically, the first way of treating this function was with a first order approximation method as explained by McReynolds [85].

The last operator that needs to be defined is the \( \bullet \) operator. It is extensively used for retaining matrix notation in the following sections, while treating tensor operations:
\[ C = A \bullet B: \text{ if } B \text{ is a tensor, and } A \text{ is a vector, then it is equivalent to } C(i, j) = \sum_{p=1}^{N} B(p, i, j) A(p). \text{ If } A \text{ is a matrix, } C(i, j, k) = \sum_{p=1}^{N} A(i, p) B(p, j, k) \]

### 2.2 Differential Dynamic Programming

A general overview of DDP has been given in section 1.2.4, several different approaches were reviewed by Yakowitz [48]. Here, some mathematical concepts are reported in order to help with the following development of an optimal control solution using HDDP and DA.

For simplicity, the general case of multi-phase control is reduced to a single phase...
2.2. Differential Dynamic Programming

\( M = 1 \). First, a discussion of the solution to classic Dynamic Programming (DP) is performed, so that it becomes possible to introduce the reason for DDP. The first step into tackling the problem is reducing the cost function in Equation 2.1 to a "cost-to-go". This function describes the remaining cost that needs to be added before reaching the last stage, as computed at stage \( k \).

\[
J_k = (x_k, u_k, \ldots, u_N) := \sum_{j=k}^{N} L_j(x_j, u_j) + \phi(x_{N+1}); \tag{2.9}
\]

The main procedure of DP can be better explained as divided into its two main phases, backward induction and forward propagation.

### 2.2.1 Backward Induction

The main idea is that an optimal control policy can be retrieved regardless of the initial conditions. At step \( k \), the past decisions will not influence the optimal decisions that need to be made from \( k \) to \( N \). At the final time instant \( N \), the optimization problem is recasted as Equation 2.10, by exploiting the dynamic transition function \( F \).

\[
\min_{u_N} (J_N) = \min_{u_N} (L(x_N, u_N) + \phi(x_{N+1})) = \min_{u_N} (L(x_N, u_N) + \phi(F_N(x_N, u_N))); \tag{2.10}
\]

Regardless of past choices, the optimal policy from \( N \) to \( N + 1 \) is only affected by \( u_N \). It is clear now that, if it is possible to compute an optimal feedback \( u^*_N(x_N) \), then retrieving this control policy makes it possible to compute the optimized \( J^*_N \).

\[
\min_{u_N} J_N = J^*_N(x_N); \tag{2.11}
\]

The idea behind the overall procedure is that the optimal \( J^*_{k+1} \) at a generic time instant \( k + 1 \), is only dependent on the state at that time instant. Then the cost-to-go at step \( k \) includes this term, plus the stage cost \( L_k \) as described by Equation 2.12.

\[
J_k = L_k(x_k, u_k) + J^*_{k+1}(x_{k+1}); \tag{2.12}
\]

Once again, the dynamic transition function can be exploited as in Equation 2.10 to retrieve:

\[
J_k = L_k(x_k, u_k) + J^*_{k+1}(F_k(x_k, u_k)); \tag{2.13}
\]

Then, the only input affecting the cost is the decision vector of \( u_k \), which can be computed in the same way as for the steps from \( N \) to \( k + 1 \) by optimizing this cost as in Equation 2.14.

\[
\min_{u_k} (J_k) = \min_{u_k} (L_k(x_k, u_k) + J^*_{k+1}(F_k(x_k, u_k))); \tag{2.14}
\]

If the optimal \( u^*_k(x_k) \) is obtained, then it can be substituted in the cost Equation 2.13. This iterative procedure can be reformulated in the more canonical form:

\[
\min_{u_k} (J_k) = \min_{u_k} (L(x_k, u_k) + \min_{u_{k+1}, \ldots, u_N} (J(x_{k+1}, u_{k+1}, \ldots, x_N))); \tag{2.15}
\]

This is why the procedure is called backward-sweep. Once a nominal control feedback policy is retrieved, the algorithm can move to the next step.
2.2.2 Forward Propagation

From the backward sweep, a sequence of inputs is available, which is represented by feedback functions describing the input for every possible state vector $x$.

$$u^*_1(x_1), u^*_2(x_2), \ldots, u^*_N(x_N);$$ (2.16)

Then, the algorithm can simply start from the first point, propagate the dynamics with Equation 2.3 utilizing the new states to evaluate the optimal law when there are perturbations. However, this method requires huge amounts of storage to solve this procedure, as both the states and the controls need discretization to obtain necessary and sufficient global conditions for optimality. Therefore, DDP is introduced, the region of optimality is restricted to a corridor in the neighborhood of the reference trajectory and global optimality is sacrificed in favor of local optimality.

2.2.3 Application to Differential Dynamic Programming

As DDP and DP were originally conceived without constraints, they will be removed for now. Application to DDP requires a reference trajectory to be defined, after that, the process consists in an iteration of "backward sweep" and "forward sweep" to improve the guess made.

The reference trajectory is defined as $\bar{x}_k$ and this trajectory is expanded as $x_k = \bar{x}_k + \delta x_k$, while the reference input history is $\bar{u}_k$ with $u_k = \bar{u}_k + \delta u_k$ expansion. This is necessary to define the second order expansion in the neighborhood of this trajectory of the dynamics and cost. Then, the expansion is applied to both terms on the right side of Equation 2.12. The first element is expanded as:

$$L_k(x_k, u_k) \simeq L_k + L_{x,k}\delta x_k + L_{u,k}\delta u_k + 0.5\delta x_k^T L_{xx,k}\delta x_k + 0.5\delta u_k^T L_{uu,k}\delta u_k + \delta x_k^T L_{xu,k}\delta u_k;$$ (2.17)

All the partial derivatives and reference value of $L_k$ are evaluated on the reference trajectory ($\bar{x}_k, \bar{u}_k$). The second element of Equation 2.12 must be expanded as:

$$J^*_k(x_{k+1}) \simeq J^*_{k+1} + \delta x_{k+1}^T J^*_{xx,k+1}\delta x_{k+1};$$ (2.18)

The dynamics can now be exploited in the same way as it was done in Equation 2.10. Therefore, the state is expressed as a new expansion of $x_{k+1}$:

$$\delta x_{k+1}(\delta x_k, \delta u_k) \simeq F_{x,k}\delta x_k + F_{u,k}\delta u_k + \delta x_k^T F_{xx,k}\delta x_k + 0.5\delta x_k^T F_{xx,k}\delta x_k + 0.5\delta u_k^T L_{uu,k}\delta u_k + \delta x_k^T L_{xu,k}\delta u_k;$$ (2.19)

Also the left term of Equation 2.12 must be expanded as in:

$$J^*_k(x_k) \simeq J^* + \delta x_k^T J^*_{xx,k}\delta x_k;$$ (2.20)

At the end of this process of expansion, it is possible to substitute the expansion of dynamics $\delta x_{k+1}$ of Equation 2.19 into the expansion of the second term $J^*_{k+1}$ as expressed in Equation 2.18. Then it must be added to the expansion of the first
2.2. Differential Dynamic Programming

Reference Trajectory

\[ X_0 \]

\[ X_N \]

\[ X_k \]

\[ X_{k+1} \]

\[ L_k(x_k, u_k) \]

\[ J_{k+1}^*(x_{k+1}) \]

\[ \delta X_k \]

\[ \delta u_k \]

\[ \delta u_k + \delta u_k \]

\[ \delta u_k \]

\[ \delta u_k \]

\[ \text{Optimization: } \delta u_k(\delta X_k) \]

\[ \text{Backward Sweep} \]

\[ \text{Backward Sweep scheme adapted from [86].} \]

Term \( L_k \) as reported in Equation 2.17.

At the end of these operations, the resulting equation Equation 2.21 is obtained.

\[
L_k(x_k, u_k) + J_{k+1}^*(x_{k+1}) \simeq L_k + J_{k+1}^* + \begin{bmatrix} \frac{L_{x,k} + J_{x,k+1}^* F_{x,k}}{L_{u,k} + J_{x,k+1}^* F_{u,k}} \end{bmatrix}^T \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} + 0.5 \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix}^T Q \begin{bmatrix} \delta x_k \\ \delta u_k \end{bmatrix} ;
\]

(2.21)

Where the matrix \( Q \) is defined as:

\[
Q = \begin{bmatrix} L_{xx,k} + J_{x,k+1}^* \cdot F_{xx,k} + F_{x,k}^T J_{x,k+1}^* F_{x,k} & L_{uu,k} + J_{x,k+1}^* \cdot F_{uu,k} + F_{x,k}^T J_{x,k+1}^* F_{u,k} \\ L_{xu,k} + J_{x,k+1}^* \cdot F_{xu,k} + F_{x,k}^T J_{x,k+1}^* F_{u,k} & L_{uu,k} + J_{x,k+1}^* \cdot F_{uu,k} + F_{x,k}^T J_{x,k+1}^* F_{u,k} \end{bmatrix} ;
\]

(2.22)

By imposing that the gradient of Equation 2.21 with respect to \( \delta u_k \) is equal to 0, an optimal control policy can be retrieved, provided that Equation 2.23 is a positive definite matrix.

\[
Q_{uu,k} = L_{uu,k} + J_{x,k+1}^* \cdot F_{uu,k} + F_{u,k}^T J_{x,k+1}^* F_{u,k} ;
\]

(2.23)

Such control policy is a linear feedback law, formulated as \( \delta u_k^* = A_k + B_k \delta x_k \) whose \( A_k \) and \( B_k \) matrices are reported in Equation 2.24:

\[
A_k = -[L_{uu,k} + J_{x,k+1}^* \cdot F_{uu,k} + F_{u,k}^T J_{x,k+1}^* F_{u,k}]^{-1}[L_{uu,k} + J_{x,k+1}^* F_{u,k}] ;
\]

\[
B_k = -[L_{uu,k} + J_{x,k+1}^* \cdot F_{uu,k} + F_{u,k}^T J_{x,k+1}^* F_{u,k}]^{-1}[L_{uu,k} + J_{x,k+1}^* \cdot F_{uu,k} + F_{u,k}^T J_{x,k+1}^* F_{u,k}]^T ;
\]

(2.24)

After retrieving this optimal control feedback law, one can substitute Equation 2.24 in \( J_k \) as expressed in Equation 2.21 to obtain \( J_{x,k}^* \) and proceed with the backward-sweep as illustrated in Figure 2.2. During the backward sweep, it is necessary to keep track of matrices \( A_k \) and \( B_k \) as they are needed for the forward...
Chapter 2. Theoretical Background

19

Reference Trajectory

\[ X_{k+1} = \begin{bmatrix} \delta X_k \\ \delta u_k \end{bmatrix} \]  

Forward Sweep

\[ X_0 \rightarrow X_N \]

Updated Trajectory

\[ \text{Figure 2.3: Forward Sweep scheme adapted from [86].} \]

sweep. After optimizing backward from final to initial point, the forward sweep starts and improves the reference guess trajectory.

Starting from the first point, where the state is known (\( \delta x_0 = 0 \)), a new \( u_0 = \overline{u}_0 + \delta u_0 \) is applied where \( \delta u_0 = A_0 + B_0 \delta x_0 \). The new state \( x_1 \) will differ in general from \( \overline{x}_1 \) reference. Therefore it will be possible to compute \( \delta x_1 = x_1 - \overline{x}_1 \). The following step of the algorithm, is then to use this \( \delta x_1 \) to obtain \( \delta u_1 = A_1 + B_1 \delta x_1 \). This makes it possible to compute \( u_1 = \overline{u}_1 + \delta u_1 \) and so on for all the stages from 0 to the final one. At each step \( k \) the new \( \overline{x}_{k+1} \) is obtained by applying a new optimal control policy: \( u_k = \overline{u}_k + \delta u_k^* \). \( \delta u_k^* \) is now given by \( A_k + B_k (x_k - \overline{x}_k) \) estimating \( \delta x_k \) as the difference between the old reference trajectory and the new computed \( x_k \) during the precedent step of the forward sweep. The process is illustrated in Figure 2.3.

2.2.4 Limitations of Differential Dynamic Programming

DDP as formulated in the above basic theory, and as explained in section 1.2.4, falls short in several aspects.

Constraints Handling

Solutions of DDP are particularly efficient for some categories of problems, described in depth by Liao and Schoemaker [87]. These problems are smooth and unconstrained, and they yield positive definite Hessians. Many techniques have been implemented to deal with this shortcoming, HDDP introduces a mixed approach where augmented Lagrangian is used to enforce phase constraints, which in general admit some slack. Whereas stage constraints, which are instead representative of physical limitations (e.g. thrust magnitudes) are enforced with constrained NLP techniques. This procedure was first implemented by Lin and
2.3 Hybrid Differential Dynamic Programming

The algorithm developed by Lantoine and Russell [2] is paramount to this work, therefore and overview of the algorithm is given in this section. The key steps are deeply investigated, and a brief overview of the procedure is given at the end.

2.3.1 Problem Reformulation

The problem structure introduced for DDP needs to be modified to comply with the multi-phase structure of HDDP. As mentioned in subsubsection 2.2.4, the phase constraints are treated with an augmented Lagrangian method. In order to do so, the final cost functions of phases $\phi_i$ are reformulated as Equation 2.25, where an augmented Lagrangian function is provided, exploiting the classical formulation of the quadratic penalty term.

$$
\tilde{\phi}_i(x_{i,Ni+1}, \omega_i, x_{i+1,1}, \omega_{i+1}, \lambda_i) := \phi_i(x_{i,Ni+1}, \omega_i, x_{i+1,1}, \omega_{i+1}) + \lambda_i^T \psi_i(x_{i,Ni+1}, \omega_i, x_{i+1,1}, \omega_{i+1}) + \sigma \| \psi_i(x_{i,Ni+1}, \omega_i, x_{i+1,1}, \omega_{i+1}) \|^2; \quad (2.25)
$$

The Lagrange multipliers are expressed as $\lambda_i \in \mathbb{R}^{n\psi_i}$ while $\sigma \in \mathbb{R}$ is a real penalty parameter that could be reformulated as a matrix in case different constraints show significantly different impact on the cost function. This approach was, in fact, implemented by Aziz et al. [86].

Inserting the newly implemented augmented cost into the formulation of the merit function in Equation 2.1, the problem finds its final shape. Thanks to Bertsekas in [39], the problem can be reformulated as the following minimax problem:

$$
\max_{\lambda_i} \min_{u_{i,j}, \omega_i} \sum_{i=1}^{M} \left[ \sum_{j=1}^{N_i} (L_{i,j}(x_{i,j}, u_{i,j}, \omega_i)) + \tilde{\phi}_i(x_{i,Ni+1}, \omega_i, x_{i+1,1}, \omega_{i+1}, \lambda_i) \right]; \quad (2.26)
$$
Chapter 2. Theoretical Background

Subject to the following conditions:

\[
\begin{cases}
  x_{i,1} = \Gamma_i(\omega_i); \\
  x_{i,j+1} = F_{i,j}(x_{i,j}, u_{i,j}, \omega_i); \\
  g_{i,j}(x_{i,j}, u_{i,j}, \omega_i) \leq 0; \\
  u_{i,j}^L \leq u_{i,j} \leq u_{i,j}^U; \\
  \omega_{i,j}^L \leq \omega_{i,j} \leq \omega_{i,j}^U;
\end{cases}
\] (2.27)

As described in [38] by Hestenes and Magnus, the classical approach to this problem is to split it in two nested loops. The first inner loop solves the problem exactly minimizing for the inputs \( u, \omega \), whereas the outer loop maximizes the Lagrange multipliers \( \lambda \). The new method proposed in HDDP by Lantoine and Russell is to update simultaneously at each iteration the control variables and the Lagrange multipliers. This method is an extension of methods that give inexact minimization of the augmented functions as developed by Bertsekas in [89]. Bertsekas also provided proof of convergence for such algorithms in [39]. According to the notation developed for DDP in section 2.2.3, the variation from the reference states, co-states and controls is defined as:

\[
x_{i,k} = x_{i,k} + \delta x_{i,k} u_{i,k} = u_{i,k} + \delta u_{i,k}, \omega_i = \omega_i + \delta \omega_i, \lambda_i = \lambda_i + \delta \lambda_i.
\]

### 2.3.2 Quadratic Expansions

The HDDP algorithm, exploits its multiphase structure to quadratically expand the cost function at each stage of a phase, and then extend the expansion at the end of the phase to the previous one. Using this strategy makes it possible to sweep from the last stage of the last phase all the way back to the initial condition of the first phase.

**Stage Expansion**

At a generic stage, the expansion of the cost function is formulated similarly to classical DDP according to section 2.2.3. Equation 2.12 can be reprised, and the two right hand side terms can be expanded according to the new variables per each stage \( k \) of phase \( i \). Being index \( i \) present per each variable, it is removed to improve readability.

\[
\delta L_k \simeq L_{x,k}^T \delta x_k + L_{u,k}^T \delta u_k + L_{\omega,k}^T \delta \omega \\
+ 0.5 \delta x_k^T L_{xx,k} \delta x_k + 0.5 \delta u_k^T L_{uu,k} \delta u_k + 0.5 \delta \omega^T L_{\omega \omega,k} \delta \omega \\
+ \delta x_k^T L_{xu,k} \delta u_k + \delta x_k^T L_{x\omega,k} \delta \omega + \delta u_k^T L_{uw,k} \delta \omega; 
\] (2.28)

\[
\delta J_{k+1}^* \simeq E R_{k+1} + J_{x,k+1}^* \delta x_{k+1} + J_{\omega,k+1}^* \delta \omega + J_{\lambda,k+1}^* \delta \lambda \\
+ 0.5 \delta x_{k+1}^T J_{xx,k+1}^* \delta x_{k+1} + 0.5 \delta \omega^T J_{\omega \omega,k+1}^* \delta \omega + 0.5 \delta \lambda^T J_{\lambda \lambda,k+1}^* \delta \lambda \\
+ \delta x_{k+1}^T J_{x\omega,k+1}^* \delta \omega + \delta x_{k+1}^T J_{x \lambda,k+1}^* \delta \lambda + \delta \omega^T J_{\omega \lambda,k+1}^* \delta \lambda; 
\] (2.29)

In Equation 2.29, \( ER \) represents the expected reduction of the cost function subject to the optimal control. Indeed it represents the term \( J_{k+1} - J_{k+1}^* \).
Finally, as in section 2.2.3, the state \( x_{k+1} \) is mapped with the dynamic transition function and expanded:

\[
\delta x_{k+1} \simeq F_{x,k} \delta x_k + F_{u,k} \delta u_k + F_{\omega,k} \delta \omega \\
+ 0.5 \delta x_k^T \bullet F_{xu,k} \delta x_k + 0.5 \delta u_k^T \bullet F_{uu,k} \delta u_k + 0.5 \delta \omega ^T \bullet F_{\omega u,k} \delta \omega \\
+ 0.5 \delta x_k^T \bullet F_{xu,k} \delta u_k + \delta x_k^T \bullet F_{x\omega,k} \delta \omega + \delta u_k^T \bullet F_{uu,k} \delta \omega ;
\]  

(2.30)

These expansions are combined in a similar way as it was done for standard DDP. First, the expansion of \( \delta x_{k+1} \) as in Equation 2.30 is substituted in the expansion of \( J_k^i \) as in Equation 2.29. Then, they are summed with expansion of \( L_k \) and equated with the expansion of \( J_k^* \), so that \( \delta J_k(\delta x_k, \delta u_k, \delta \lambda, \delta \omega) \) is obtained:

\[
\delta J_k \simeq -ER_{k+1} + J_{x,k}^T \delta x_k + J_{u,k}^T \delta u_k + J_{\omega,k}^T \delta \omega + 0.5 \delta x_k^T J_{x\omega,k} \delta x_k \\
+ 0.5 \delta u_k^T J_{uu,k} \delta u_k + 0.5 \delta \omega^T J_{\omega\omega,k} \delta \omega + 0.5 \delta \lambda^T J_{\lambda\omega,k} \delta \lambda + \delta x_k^T J_{x\lambda,k} \delta \lambda + \delta u_k^T J_{u\lambda,k} \delta \lambda + \delta \omega^T J_{\omega\lambda,k} \delta \lambda;
\]

(2.31)

**Phase Expansion**

At the end of each phase, it is necessary to express between phase \( i \) and phase \( i-1 \). To comply with the notation of HDDP as in [2], terms belonging to phase \( i \) are now renamed “+” and terms belonging to \( i-1 \) are now reformulated as “−”. The cost function to be expanded is now the cost optimized until phase \( i \), as expressed in Equation 2.31 i.e. \( \delta J_k(\delta \omega_+, \delta \lambda_+, \delta x_+), \) augmented with the final phase cost of phase \( i - 1 \), as expressed in Equation 2.25 i.e. \( \delta \phi_-(\delta \lambda_-, \delta \omega_-, \delta x_-, \delta \omega_+, \delta x_+) \). The partials are here reported, nevertheless for a full derivation of the partials the reader is directed to [2].

\[
\delta J_{i,0} \simeq ER_{i,1} + J_{x,x}^T \delta x_+ + J_{x,\omega}^T \delta \omega_+ + J_{x,\lambda}^T \delta \lambda_+ + J_{x,x}^T \delta x_- + J_{x,\omega}^T \delta \omega_- + J_{x,\lambda}^T \delta \lambda_- \\
+ 0.5 \delta x_+^T J_{x,x+x} \delta x_+ + 0.5 \delta \omega_+^T J_{x,\omega} \delta \omega_+ + 0.5 \delta \lambda_+^T J_{x,\lambda} \delta \lambda_+ \\
+ 0.5 \delta x_-^T J_{x,x-x} \delta x_- + 0.5 \delta \omega_-^T J_{x,\omega} \delta \omega_- \\
+ \delta x_+^T J_{x,x+\omega} \delta \omega_- + \delta x_-^T J_{x,x-\omega} \delta \omega_+ + \delta x_+^T J_{x,x+\lambda} \delta \lambda_- + \delta x_-^T J_{x,x-x} \delta x_- \\
+ \delta \omega_+^T J_{x,x+\omega} \delta \omega_- + \delta \omega_-^T J_{x,x-\omega} \delta \omega_+ + \delta \omega_+^T J_{x,x+\lambda} \delta \lambda_- + \delta \omega_-^T J_{x,x-x} \delta \lambda_- \\
+ \delta \omega_+^T J_{x,x+\lambda} \delta \lambda_- + \delta \omega_-^T J_{x,x-\lambda} \delta \lambda_-;
\]

(2.32)

Partials of the cost function (that needs to be computed analytically), are obtained by the partials of \( J_- = J_+ + \tilde{\phi}_- \). An important note to be made is that there are no mixed partials of \( \delta \lambda_+ \) and the terms indexed “−”. The reason being, \( \lambda_+ \) only entering in \( J \) through the cost at the end of phase \( i \) as in Equation 2.25. Finally, to express \( \delta x_+ \) as function of \( \delta \omega_+ \), the initial condition Equation 2.2 is exploited by expanding it to second order:

\[
\delta x_+ = \Gamma(\omega_+) - \Gamma(\omega_-) = \Gamma_\omega \delta \omega_+ + 0.5 \delta \omega_+^T \Gamma_{\omega\omega} \delta \omega_+;
\]

(2.33)
Chapter 2. Theoretical Background

Figure 2.4: Expansion between phases for HDDP, adapted from [2].

By inserting this equation into Equation 2.32, the dependency from $δx_+$ is removed. In the end of this process, Equation 2.34 remains.

$$δJ_{i,0} = J_{i,0}(δω_+, δω_-, δλ_+, δλ_-, δx_-);$$

(2.34)

To better understand the terms that influence the expansion of the cost at the end of a phase, Figure 2.4 is reported, as adapted from [2].

2.3.3 State Transition Matrix Approach

One of the most important features of HDDP is its STM approach to the propagation of the cost partials. These STMs are introduced by generating an augmented state and the corresponding augmented transition function:

$$X_k = \begin{bmatrix} x_k \\ u_k \\ ω_k \end{bmatrix};$$

(2.35)

$$\tilde{F}_k = \begin{bmatrix} F_k \\ 0_{n_u} \\ 0_{n_ω} \end{bmatrix};$$

(2.36)

The expansion of the states dynamics in Equation 2.30 can be rewritten thanks to the newly introduced augmented states as:

$$δX_{k+1} ≃ \tilde{F}_{X,k}δX_k + 0.5δX_k^T\tilde{F}_{XX,k}δX_k = Φ_k^1δX_k + 0.5δX_k^T • Φ_k^2δX_k;$$

(2.37)

Substituting this Equation 2.37 into the quadratic expansion of the stage reported in Equation 2.31 at the end of subsubsection 2.3.2, makes it possible to compute
the partials at a generic stage $k$ of phase $i$ by mapping them with the State Transition Matrix $\Phi_1^k$ and the State Transition Tensor $\Phi_2^k$.

This formulation decouples dynamics from the optimization. In fact, state transition matrices can be computed once the reference trajectory is picked. Then they are stored, and can be used during the backward sweep. If the backward sweep produces a good iteration, that is accepted, the reference is updated, therefore the STMs are recomputed. If the iteration is discarded, then the reference trajectory does not change, hence the STM are not recomputed. The mapping is expressed according to the following equation by simple combination of Equation 2.28, Equation 2.29 in which the newly found Equation 2.30 is substituted.

\[
\begin{bmatrix}
J_{x,k}^T \\
J_{u,k}^T \\
J_{\omega,k}^T
\end{bmatrix} =
\begin{bmatrix}
L_{x,k}^T \\
L_{u,k}^T \\
L_{\omega,k}^T
\end{bmatrix} +
\begin{bmatrix}
J_{x,k+1}^* \\
0_{n_u} \\
J_{\omega,k+1}^*
\end{bmatrix}^{T} \Phi_1^k;
\]  
\(2.38\)

\[
\begin{bmatrix}
J_{xx,k} & J_{ux,k} & J_{x\omega,k} \\
J_{ux,k} & J_{uu,k} & J_{u\omega,k} \\
J_{x\omega,k} & J_{u\omega,k} & J_{\omega\omega,k}
\end{bmatrix}
= 
\begin{bmatrix}
L_{xx,k} & L_{ux,k} & L_{x\omega,k} \\
L_{ux,k} & L_{uu,k} & L_{u\omega,k} \\
L_{x\omega,k} & L_{u\omega,k} & L_{\omega\omega,k}
\end{bmatrix}

+ \Phi_1^T_k
\begin{bmatrix}
J_{x,k+1}^* & 0_{n_u \times n_x} \\
0_{n_u \times n_x} & 0_{n_u \times n_u} \\
0_{n_u \times n_u} & J_{\omega,k+1}^*
\end{bmatrix}
\Phi_1^k
+ 
\begin{bmatrix}
J_{x,k+1}^* \\
J_{u,k+1}^* \\
J_{\omega,k+1}^*
\end{bmatrix}
\bullet \Phi_2^k;
\]  
\(2.39\)

Finally, according to this formulation, the partials of the Lagrange multipliers, which do not have dynamic equations, come by simple application of the chain rule:

\[
\begin{align*}
J_{\lambda,k} &= J_{\lambda,k}^*; \\
J_{\lambda\lambda,k} &= J_{\lambda\lambda,k}^*; \\
\left[ J_{x\lambda,k}^T, J_{u\lambda,k}^T, J_{\omega\lambda,k}^T \right] &= J_{X\lambda,k+1}^T \frac{\partial X_k}{\partial X_k} = 
\left[ J_{x\lambda,k+1}^*, 0_{n_u}, J_{\omega\lambda,k+1}^* \right] \Phi_1^k;
\end{align*}
\]  
\(2.40\)

**Partials Propagation**

The propagation of dynamical equations allows for the computation of partials which lies at the basis of every optimization method as described by Betts [6]. The accuracy of such computations can compromise the results of the optimization algorithm. In fact, they can lead to sub-optimal steps that slow down (or prevent entirely) convergence of the algorithm, therefore accurate integration is required. The computation of partials can represent a heavy computational load for the algorithm as explained by Pellegrini [90]. Several algorithms are available and their comparison can be found in the overview given by Fox [91]. This is a key step for the algorithm, as the STM formulation allows for parallel
computing. The usual methods of propagation are reported in Pellegrini’s Ph.D. thesis [90]. The most relevant are propagation of variational equations, finite differences methods and automatic differentiation. In particular, Pellegrini faced the problem of complex-step derivatives. Fornberg showed that finite differences approaches often yield round-off errors [92], accurate higher orders methods can be implemented, but they require more evaluations of the functions, hence more computational effort as reported by Pellegrini [90]. The HDDP algorithm however relies on propagation of the variational equations alongside the trajectory. These transition functions of the states can be, in fact, computed one stage independently from the other. The method for propagating them is proposed by Majji et al.[93] and Park et al.[94]. It requires the integration of the following system of differential equations:

\[
\begin{align*}
\dot{\Phi}_1^k &= f_X \Phi_1^k; \\
\dot{\Phi}_2^k &= f_X \cdot \Phi_2^k + \Phi_1^k \cdot f_{XX} \cdot \Phi_1^k; \\
\dot{x} &= f;
\end{align*}
\] (2.41)

subject to the following initial conditions:

\[
\begin{align*}
\Phi_1^k(t_k) &= I_{nx+nu+nw}; \\
\Phi_2^k(t_k) &= 0_{nx+nu+nw}; \\
x(t_k) &= x_k;
\end{align*}
\] (2.42)

This system of ODE is composed of \( N + N^2 + N^3 \) functions, therefore its computational cost increases very rapidly with the number of variables. Even if the sparsity and symmetry of the Hessian is considered, the computational cost is not significantly improved, as a loss of matrix notation imply more tedious implementation.

Most importantly, it must be noted that the required partials \( f_X \) and \( f_{XX} \) require a great deal of computational effort. Their derivation is often based on dedicated softwares to extract accurate partials. This process is removed in this work thanks to DA, as a faster way of propagating the cost partials is found.

Particular attention should be made while numerically integrating the STM, as they are representative of the partial derivatives of the approximated trajectory only for fixed step integrators, as explained by Pellegrini [90]. The practical approach of improved convergence as stated by Betts [6] is here verified by Pellegrini [90] and reported for clarity.

In the trivial case of first order Euler integrations:

\[
\begin{align*}
X_{i+1} &= X_i + f(X_i)\delta t; \\
\Phi_1^{i+1} &= \Phi_1^{i+1} + f_X|X_i\Phi_1^{i+1}\delta t; \\
\end{align*}
\] (2.43)

given \( \Phi_1^{i+1} = \frac{\partial X_{i+1}}{\partial X_0} \), if the partial derivative of the first equation of Equation 2.43 is taken with respect to the initial state \( X_0 \), Equation 2.44 is obtained.

\[
\frac{\partial X_{i+1}}{\partial X_0} = \frac{\partial X_i}{\partial X_0} + \frac{\partial f}{\partial X}|_{X_i} \frac{\partial X_i}{\partial X_0} \delta t = \Phi_1^i + f_X|X_i\Phi_1^i \delta t;
\] (2.44)
This equation and the second part of Equation 2.43 are identical, hence the propagation of the partial is correct. The same considerations can be extended to higher order methods and for higher order partials with some effort.

On the other hand, using a variable step integrator may compromise the results of the optimization if the tolerances required are not very strict [90]. The same example for variable step integrator yield some adjustments. In particular the $\delta t$ is a function of the error at the previous step, therefore $\delta t(X_i)$. Now, taking the partial of the dynamic function as in Equation 2.44 gives a different result, in particular:

$$
\frac{\partial X_{i+1}}{\partial X_0} = \frac{\partial X_i}{\partial X_0} + \frac{\partial f}{\partial X}_{X_i, \partial X_0} \delta t + f(X_i) \frac{\partial \delta t}{\partial X}_{X_i, \partial X_0} = \Phi_1^i + \left( f_X|_{X_i} \delta t + f(X_i) \frac{\partial \delta t}{\partial X}_{X_i} \right) \Phi_1^i;
$$

(2.45)

Hence, the variational equations are modified by a change in the step size. Differences and comparison of results are detailed in Pellegrini’s work [90].

### 2.3.4 Stage Optimization

Up until now, this section described the expansion process and the propagation of the system’s dynamics, together with the mapping of partials thanks to the STM approach. However, it is necessary to discuss how HDDP implements optimization procedures both on steps and phases.

The idea of this step is to exploit the expansion of the cost function obtained at the end of section 2.3.2, take its gradient with respect to $\delta u_k$ and equate it with 0. The aim of this procedure is to obtain a law for $\delta u_k$ such as the following:

$$
\delta u_k = -J^{-1}_{uu,k}(J_{u,k} + J_{ux,k}\delta x_k + J_{u\lambda,k}\delta \lambda + J_{uu,k}\delta \omega);
$$

(2.46)

As mentioned before, the algorithm must ensure constraints compliance and a descent direction, which are not always guaranteed if $J_{uu,k}$ is not positive definite. To deal with this issues, two main approaches were exploited in HDDP.

#### Trust Region Method

This method is implemented to guarantee positive definiteness of the Hessian matrix $J_{uu,k}$, and at the same time, to constrain the resulting step into a quadratic region of validity. The treatment of this trust region problem is relevant to the approach of the solution exploiting DA, therefore the basic form of the algorithm will be explained in detail here. The complete algorithm is described in Conn, Gould and Toint [95].

The problem to be solved is the following:

$$
\begin{cases}
\min_{s \in \mathbb{R}^n} q(s) = g^T s + 0.5 s^T H s; \\
||s|| \leq \Delta;
\end{cases}
$$

(2.47)

The solution is either interior to the trust region, or on the boundary. If $q(s)$ is convex and the Hessian $H$ is positive semidefinite, the solution may or may not
lie on the boundary. In the non-convex case, a solution must lie on the boundary of the trust region.

According to the theorem 7.2.1 in [95]:

Any minimizer of this problem subject to the equality constraint \( ||s||^2 - \Delta = 0 \) complies with Equation 2.48, given \( H(\lambda M) = H + \lambda M I \) positive semidefinite.

\[ H(\lambda M)s^M = -g; \tag{2.48} \]

Moreover, if \( H(\lambda M) \) is positive definite, \( s^M \) is unique.

In fact, it is possible to consider the minimization of the following Lagrangian:

\[ \mathcal{L} = g^T s + 0.5s^T Hs + \lambda ||s||^2 - \Delta^2; \tag{2.49} \]

If the gradient with respect to \( s \) is taken, Equation 2.48 is retrieved.

When the constraint is \( ||s||^2 \leq \Delta \), \( \lambda M \) represents the Lagrange multiplier of such constraint. Then \( \lambda M \) is either greater than 0, when the constraint is active, or equal to zero, when the constraint is not active (given \( H(\lambda M) \) is positive semi-definite, \( \lambda M \geq 0 \) and \( \lambda M(||s^M||^2 - \Delta) = 0 \)).

If a solution lies on the boundary, then \( H(\lambda M) \) is positive semi-definite and \( \lambda M(||s^M||^2 - \Delta) = 0 \) is obviously verified as \( ||s|| = \Delta \) and \( \lambda M \geq 0 \) is necessary for active constraints. On the other hand if the solution lies inside, then \( Hs^M + g = 0 \) must be verified (i.e. \( \nabla s q(s) = 0 \)) with \( H \) positive semi definite, therefore the conditions are the same as having \( \lambda M = 0 \) (inactive constraint).

To conclude, if \( H(\lambda M) \) is positive definite, the global minimizer is unique on the trust region boundary by theorem 7.2.1. Otherwise, if the global minimizer is inside the trust region, \( \lambda M \) is necessarily 0: this implies \( H \) itself is positive definite therefore \( s \) is again unique.

This technique represents a particular case of Hessian shifting, in particular, the shift given is equal to the Lagrange multiplier of the trust region constraint.

The Trust-Region Quadratic Programming Algorithm (TRQP) aims at finding this \( \lambda M \).

To find the Lagrange multiplier when the solution lies on the border, \( ||s(\lambda)||^2 - \Delta = 0 \) needs to be solved. For convenience, the definition of \( ||s(\lambda)||^2 \) is given:

\[ \psi(\lambda) = ||s(\lambda)||^2 = ||U^T(\Lambda + \lambda I)^{-1}Ug||^2 = \sum_{i=1}^{n} \frac{\gamma_i^2}{(\lambda_i + \lambda)^2}; \tag{2.50} \]

This equation has many poles, therefore it tends to infinity in those points where \( \lambda = -\lambda_i \). For \( \lambda \to \infty \), this function tends to zero. Therefore, for decreasing values of \( \lambda \), the pole which is encountered first is the negative of the smallest eigenvalue of \( H \), from here on called \( \lambda_1 \). Given Equation 2.50 is positive for every value of \( \lambda \), it is known that a solution to \( \psi(\lambda) = \Delta^2 \) will be found between \( \lambda \to \infty \) and \( \lambda \to -\lambda_1 \) where \( \psi(\lambda) \) goes to infinity. If \( \lambda_1 \) is smaller than zero (\( H \) non definite), then \( \lambda \in [-\lambda_1, \infty) \). This means that it is mandatory to constrain the research in this interval to guarantee that the \( H(\lambda) \) matrix is positive semi
definite. If, instead, \( H \) is positive definite, then \( \lambda \in [0, \infty) \) as it is only interesting to study those solutions where \( \lambda \geq 0 \). This confirms that when the problem is positive definite, the solution is unique and it can either lie on the inside or on the outside of the trust region. On the other hand, if the Hessian is not positive definite, the solution has to lie on the boundary of the trust-region and therefore \( \lambda > 0 \). In some particular cases, there is cancellation of \( \gamma_1 \), therefore the influence of the smallest pole is eliminated. If this pole is smaller than 0, then this may pose some difficulty. It is, in fact, obvious that the solution must lie between \( \lambda \in [-\lambda_1, \infty) \) now that \( \lambda_1 \) has no effect, but there is the added condition that \( H(\lambda) \) is positive semidefinite, and this imposes \( \lambda \leq -\lambda_1 \). This solution happens when the gradient has no component in the eigenvector set defined by \( \lambda_1 \). To solve this issue, consider \( H(-\lambda_1)u_1 = 0 \): this equation admits many solutions as \( H(-\lambda_1) \) is positive semidefinite and singular. This implies that the equation \( H(-\lambda_1)(s) = -g \) has many solutions, in the set \( H(-\lambda_1)(s + \alpha u_1) = -g \). In particular there will be a scalar \( \alpha \) that gives \( ||s + \alpha u_1||^2 = \Delta \).

**Root finding**

A smart reformulation of the problem is obtained by finding the solution of \( ||s(\lambda)||^2 - \Delta = 0 \) and exploiting the secular equation:

\[
\phi(\lambda) = \frac{1}{||s(\lambda)||^2} - \frac{1}{\Delta} = 0; \quad (2.51)
\]

This equation is based on the observation that \( \frac{1}{\psi(\lambda)} \) is better behaved than \( \psi(\lambda) \) as it has no finite poles, and the zeros in the location of the negative eigenvalues of \( H \). It is then easier to apply a Newton method, specifically, the derivative of \( \phi(\lambda) \) is easily derived as:

\[
\phi'(\lambda) = -\frac{s(\lambda)^T \nabla_{\lambda} s(\lambda)}{||s(\lambda)||^4}; \quad (2.52)
\]

\( \nabla_{\lambda} s(\lambda) = -H(\lambda)^{-1}s(\lambda) \) can be obtained by deriving the equation \( (H + \lambda I)s(\lambda) = -g \).

The way to apply the Newton iteration is to compute the factorization of \( H(\lambda) = LL^T \) for a guess \( \lambda > -\lambda_1 \). The Cholesky factorization is defined as the only interesting in values of \( \lambda \) are those for which the matrix positive definite. Then, solve \( LL^T s = -g \), and finally evaluate \( s^T H^{-1}s \) which is necessary to compute \( \nabla_{\lambda} s(\lambda) \).

The Newton iterate is simply \( \lambda^+ = \lambda - \frac{\phi(\lambda)}{\phi'(\lambda)} \). Some relevant properties of the secular function are that the function is strictly increasing and concave when \( \lambda > -\lambda_1 \).

**Safeguards for the Method**

Having stated concavity of the secular function, leads to showing that:

**H1:** All the Newton iterates starting in a \( \lambda > -\lambda_1 \) in which \( \phi(\lambda) < 0 \) (that is the constraint is violated) converge with a Q-quadratic rate, and every iteration point inherits these properties. If an iterate point can be found between \(-\lambda_1\) and \( \lambda^* \) optimal, then convergence is guaranteed.
Chapter 2. Theoretical Background

H2: If starting in a \( \lambda > -\lambda_1 \) in which \( \phi(\lambda) > 0 \) then the new iterate has two possible outcomes. In the first case, \( \lambda^+ \) is smaller than the optimal solution and larger than \( -\lambda_1 \) with \( \phi(\lambda^+) \leq 0 \). In the second case, \( \lambda^+ \) will be smaller than \( -\lambda_1 \). Therefore, the first case will assure safeguarding and convergence, in the second case the algorithm discards the iteration and picks a new value.

The overall structure of the safeguarding implementation requires the definition of three subsets of \( \lambda \):

\[
\begin{align*}
\mathcal{N} &= \{ \lambda | \lambda \leq \max[0, -\lambda_1] \}; \\
\mathcal{L} &= \{ \lambda | \max[0, -\lambda_1] < \lambda \leq \lambda^* \}; \\
\mathcal{G} &= \{ \lambda | \lambda > \lambda^* \};
\end{align*}
\]  

(2.53)

The extraction of the maximum between \( -\lambda_1 \) and 0 is necessary, as in the convex case, the interesting case remains \( \lambda > 0 \).

\( \mathcal{L} \) corresponds to the case where \( \lambda \) is smaller than the optimal value, \( \mathcal{G} \) corresponds to the case where \( \lambda \) is greater than the optimum value and \( \mathcal{N} \) corresponds to the case where \( H(\lambda) \) is non positive definite, therefore the solution is not acceptable. A representation of a typical case is depicted in Figure 2.5. Identifying at which set a certain iterate belongs is straightforward by simple evaluation of \( \phi(\lambda) \).

The fundamental idea of the TRQP algorithm is to construct an interval of \([\lambda_L, \lambda_U]\) so that \( \lambda_U \) always belongs to \( \mathcal{G} \) while \( \lambda_L \) always belongs to \( \mathcal{L} \). Then, this interval should be shrunk at each iteration until a certain tolerance is met. Thanks to the way these sets are built, starting from a value smaller than the upper bound, always leads to a point smaller than the upper bound. Whereas, starting from a point greater of the lower bound always leads to a point greater than the lower bound according to H1 and H2 (as observed from the arrows in Figure 2.5).

The desirable set, as stated before, is \( \mathcal{L} \): an iterate starting here guarantees Q-quadratic convergence and the algorithm can simply iterate (H1).

If the starting condition lies in \( \mathcal{G} \), the two possible outcomes are that the next iteration ends up in \( \mathcal{L} \) or in \( \mathcal{N} \) as explained in H2. In the first case, the best condition is retrieved again and all the following iterations will be in \( \mathcal{L} \). If the trial point is in \( \mathcal{N} \), the Newton iterate is not computed, and \( \lambda^+ \) is instead re-picked in the acceptable interval.

During iterations, if a \( \lambda \) in \( \mathcal{L} \) or in \( \mathcal{N} \) is found, then \( \lambda_L \) can be reset to the newly found value, effectively rising the left bound of the interval.

If the algorithm step leads to \( \mathcal{G} \), the iterate that lead to this point was not a Newton iterate, as the only possible outcomes of Newton iterates is having \( \phi(\lambda) < 0 \) or \( \lambda \in \mathcal{N} \) according to H1. Therefore, \( \lambda_U \) is reset to this value. The situation is better seen in the following Figure 2.5.

**Lower Bound Identification**

To guarantee the correctness of the lower bound \( \lambda_L \), it must be close to \( -\lambda_1 \). Having an improved lower bound can lead to a sensible speeding up
of the convergence process. According to the Rayleigh quotient rule, for a symmetric matrix and a nonzero \( p \) vector:

\[
\lambda_{\text{min}} \leq \frac{p^T H p}{p^T p} \leq \lambda_{\text{max}}; \quad (2.54)
\]

This grants that for every unit vector \( u^T H(\lambda) u \geq \lambda + \lambda_1 \), therefore by choosing an improved guess on the lower bound \( \lambda^B = \lambda - u^T H(\lambda) u \) makes it possible to choose \( \lambda^B \leq -\lambda_1 \). It is desirable to have the smallest possible \( u^T H(\lambda) u \) because at the limit, \( u \) is actually the eigenvector corresponding to the minimum eigenvalue, therefore the lower bound becomes exactly \( \lambda^B = -\lambda_1 \). If considering the case of \( H(\lambda) \) positive definite, it is possible to perform the Cholesky factorization \( LL^T \). Then a technique similar to power method for the estimation of eigenvalues and eigenvectors is exploited. In fact, multiplying a random vector \( v \) by \( H(\lambda)^{-1} v \) the result will amplify components in the eigenvector direction. The larger this product becomes, the closer to the actual eigenvector direction of \( \lambda_1 \). To generate a good approximation, the research space of \( v \) is restricted to a vector whose elements are \( \pm 1 \). Then, thanks to the factorization, making \( L^{-1} v \) large gives \( L^{-T} L^{-1} v \) large. While substituting forward, during the solution of the lower triangular system \( L \omega = v \), the \( \pm \) sign is picked depending on which value yields the higher coefficient. This method is based upon the LINPACK condition estimator.

Once \( \omega \) is found, \( u \) is simply estimated as the unit vector of direction \( L^{-T} \omega \).

Finally, if the Cholesky decomposition is not possible as \( H(\lambda) \) is indefinite, nonetheless the Cholesky factors up till \( k^{th} \) stage can be retrieved. It is pos-

![Figure 2.5: The division of domain in sub regions. Points starting in \( G \) follow the blue arrows, points in \( L \) follow the red arrow, whereas points in \( N \) are repicked in the acceptable interval according to the yellow arrow (adapted from [95]).](image)
sible to add a $\delta$ to this diagonal element so that the leading $k \times k$ submatrix of $H(\lambda) + \delta e_k e_k^T$ is singular ($e_k$ is the vector whose elements are 0 except for the $k^{th}$ element which is 1). Therefore, this result is exploited by applying $(H(\lambda) + \delta e_k e_k^T) v = 0$ while trying to find a vector which elements after index $k$ are 0, the element $k$ is 1 and all the other elements are computed by solving this system exploiting the available Cholesky factors. Then, it is possible to see that

$$v^T (H(\lambda) + \delta e_k e_k^T) v = v^T (H(\lambda) v + \delta v) = 0; \quad (2.55)$$

as $e_k v = 1$, and using the Rayleigh quotient in Equation 2.54:

$$0 = \frac{v^T H(\lambda) v}{v^T v} + \frac{\delta}{v^T v} \geq \lambda + \lambda_1 + \frac{\delta}{||v||^2}; \quad (2.56)$$

Also in this case, keeping $\lambda^B = \lambda + \frac{\delta}{||v||^2} \leq -\lambda_1$ makes it possible to approximate a better lower bound if the algorithm retains this new estimate when it is smaller than $\lambda^L$. This way it is possible to shrink the interval even in these cases.

The TRQP is applied to the Hessian and the gradient of the cost with respect to $\delta u_k$. Then the shift is obtained, and used to compute the shifted Hessian inverse. In particular, the previous algorithm is the particular case where $s = D \delta u_k$, $D$ is just a positive definite scaling matrix added to the set of tuning parameters to improve convergence of the TRQP. In fact, different variables will have different impacts on the cost and this scaling allows for a selection of an elliptical trust region, instead of a simple circular one. The same problems require $H = D^{-T} J_{uu,k} D^{-1}$ and $g = J_{u,k} D^{-1}$. The inverse of the shifted Hessian is computed with an efficient procedure exploiting an eigendecomposition of the $H$ matrix.

**Constraints Treatment**

To improve the TRQP with the addition of constraints, the inputs are bounded to move the states in the direction compliant with active sets of constraints. In this method, the treatment of constraints is efficiently accounted for, by means of a range-space method implemented by Yakowitz [42] and Murray and Yakowitz [34]. This concept is based on a linearization of active constraints and then the solution of a linearly constrained quadratic programming problem, as developed in Fletcher’s work [24].

The way this method works, is by computing the optimal step with the TRQP as if there was no change from the reference trajectory. This change is then used to estimate the set of active constraints. Afterwards, Those active constraints are linearized as $g_k = g_c + g_{u,k} \delta u_k + g_{x,k} \delta x_k + g_{\lambda,k} \delta \lambda_k = 0$. This set of active constraints must have a dimension $m_k$, smaller than the size of the input vector. Moreover, $g_k$ needs to be explicitly dependent on the controls. Therefore, taking the expansion of the cost in Equation 2.31 and substituting the shifted Hessian $\tilde{J}_{uu,k}$, allows for
formulation of the Lagrangian of the system with stage constraints:

\[
\delta J_k \simeq - E R_{k+1} + J_{x,k}^T \delta x_k + J_{u,k}^T \delta u_k + J_{\omega,k}^T \delta \omega + J_{\lambda,k}^T \delta \lambda + 0.5 \delta x_k^T J_{xx,k} \delta x_k + 0.5 \delta u_k^T J_{uu,k} \delta u_k + 0.5 \delta \omega^T J_{\omega\omega,k} \delta \omega + 0.5 \delta \lambda^T J_{\lambda\lambda,k} \delta \lambda + \delta x_k^T J_{xu,k} \delta u_k + \delta x_k^T J_{x\omega,k} \delta \omega + \delta x_k^T J_{x\lambda,k} \delta \lambda + \delta u_k^T J_{u\omega,k} \delta \omega + \delta u_k^T J_{u\lambda,k} \delta \lambda + \delta \omega^T J_{\omega\lambda,k} \delta \lambda \\
\nu_k^T (g_c + g_{x,k} \delta x_k + g_{\omega,k} \delta \omega + g_{\lambda,k} \delta \lambda);
\]

(2.57)

This formulation grants a straightforward solution thanks to Fletcher’s work [24]. Solving these problem requires the solution to the system:

\[
\begin{bmatrix}
J_{uu,k} & g_{u,k} \\
g_{u,k}^T & 0_{m_k \times n_u}
\end{bmatrix}
\begin{bmatrix}
\delta u_k \\
\nu_k
\end{bmatrix} =
\begin{bmatrix}
-J_{u,k} - J_{xu,k}^T \delta x_k - J_{u\omega,k} \delta \omega - J_{u\lambda,k} \delta \lambda \\
-g_c - g_{x,k}^T \delta x_k - g_{\omega,k}^T \delta \omega
\end{bmatrix};
\]

(2.58)

This problem is convex with respect to the optimization variable as the Hessian is shifted, moreover the constraints are linear. Therefore, sufficient and necessary conditions for optimality can be retrieved exploiting the Karush-Kuhn-Tucker conditions [22].

Many solutions exist in literature for this kind of linearly constrained equality optimization problems. Primal methods, [96] are based on the idea of reducing the set of decision variables and applying unconstrained optimization while satisfying the linear constraints. For example, HDDP implements optionally also one of these methods (even though for control bounds only): the null-space method. These methods are implemented also in state of the art solvers like SNOPT [13] for NLP problems, Pellegrini [90] uses them in MDDP. This procedure works by removing the inputs that violate the constraints from the set of decision variables. They are assigned a non feedback term so that they are fixed on their boundary. Then, the Hessian and gradient are reduced and the TRQP is recomputed on the reduced set of inputs. The process can be repeated many times until no violated constraints are found.

In both constrained and unconstrained cases, the obtained feedback law is:

\[
\delta u_k = A_k + B_k \delta x_k + C_k \delta \omega + D_k \delta \lambda;
\]

(2.59)

Substituting this feedback into the cost expansion of stage \( k \) in Equation 2.31, allows for the derivation of optimal partials of \( J_k \) with respect to \( x_k, \omega \) and \( \lambda \) which are necessary to continue the stage backward sweep, applying Equation 2.39, Equation 2.38 and Equation 2.40. The process is better illustrated with Figure 2.6.

It is important to notice that the constraints are linearized, during the forward run this may lead to their violation due to high order terms.

### 2.3.5 Phase Optimization

The techniques described in section 2.3.4 are used again in this step of the algorithm. The gradient of the cost between phases as in Equation 2.34 is now used. Noting that there are no mixed partials of \( \lambda_+ \) and the terms indexed ” - “, the problem can be decoupled and solved in two steps to make it easier. First the following feedback law is obtained

\[
\delta \lambda_+ = A_{\lambda+} + B_{\lambda+} \delta \omega_+;
\]

(2.60)
To guarantee that this step maximizes the cost, as required by Jacobson and Mayne [44], the TRQP is applied to the Hessian and gradient of the cost with respect to $\lambda_+$ changing their sign (due to the outcome of this algorithm being a positive definite matrix). Afterwards, this step is reinserted into the expansion across phases of Equation 2.34 which is now only a function of "-" terms and $\delta \omega_+$. After the application of the last TRQP to the Hessian of the cost with respect to $\delta \omega_+$, the gradient of this expansion is equated to 0, and the following feedback law is obtained:

$$
\delta \omega_+ = A_{\omega_+} + B_{\omega_+} \delta x_- + C_{\omega_+} \delta \omega_- + D_{\omega_+} \delta \lambda_-;
$$

To sum up, the minimization of across-phases cost follows this procedure:

- gradient of Equation 2.34 with respect to $\delta \lambda_+$ is equated to zero.
- TRQP is applied for $\delta \lambda_+$ and the feedback law Equation 2.60 is found.
- Equation 2.60 is substituted in Equation 2.34 and the gradient of this new function with respect to $\delta \omega_+$ is equated to zero.
- TRQP is applied for $\delta \omega_+$ and the feedback law Equation 2.61 is found.
- Equation 2.61 is substituted in the expansion resulting from previous substitution of Equation 2.60 in Equation 2.34.

At the end of this process, the partials obtained are exploited to extend the backward sweep between the two phases, by using them as initial conditions for the backward sweep on stages via Equation 2.39, Equation 2.38 and Equation 2.40.
2.3.6 Termination

Several ways of accepting trial iterates generated by this code have been made, Lantoine and Russell explicitly stated in [2] that these are just indicative criteria, exploited by many other authors. The acceptance criteria for a good iterate is the expected reduction at the end of the backward sweep to be similar to the actual one. The idea is inspired by Rodriguez et al. [97] and Whiffen in [52]. If the expected reduction from the quadratic expansion is similar to the real one, the quadratic approximation is good, therefore the trust region radius used in TRQP can be increased using heuristic criteria from Lin and Morè [98], otherwise it is reduced.

If an iteration yields an acceptable result, but the constraint violation is increased, then the penalty term will be updated to increase the cost of violation. The equation that regulates this increase in penalty is again one of the many possible, and it includes bounds to retain equilibrium between optimality and feasibility. These algorithmic aspects are better detailed in Appendix A, as they are not paramount for the following discussion.

2.3.7 Overview of the algorithm

Given the necessary mathematical tools and equations, it is now useful to give a brief overview of the algorithm, so that the contribute of this work can be put in perspective. This schematic division of the algorithm makes it possible to better understand where the improvement of the DA acts, and where the algorithmic changes implemented need to be plugged in.

**Step 0:** A first trial iterate is generated with values of controls and multipliers set to a random guess (often all 0 is a good trial). The system is propagated forward and the cost function is estimated. Go to Step 1.

**Step 1:** STM computation as in Equation 2.41 step given input and state history, often performed in parallel to improve speed. Go to Step 2.

**Step 2:** Backward propagation of the partials using Equation 2.39, Equation 2.38, Equation 2.40. At each stage the feedback law Equation 2.59 (either constrained or unconstrained as explained) is obtained and its gains stored. At the end of the phase, Equation 2.60 and Equation 2.61 are obtained and the feedback gains stored. These optimal laws are used to map optimal partials by substituting them in the respective expansions necessary for application of the backward mapping. The expected reduction at the initial point is estimated at the end of this procedure. Go to Step 3.

**Step 3:** Convergence is checked according to criteria specified in HDDP by Lantoine and Russell [2]. If the algorithm meets the convergence requirements Stop. Otherwise, proceed with the forward sweep (go to Step 4).
**Step 4:** Forward sweep to compute a new trajectory, a new input and multiplier history. Also new costs and constraint violations are estimated. Go to Step 5

**Step 5:** Based on the quality of the quadratic approximation as explained in section 2.3.6, the trust region radius is updated. If the iterate is rejected, the algorithm continues from Step 2, without recomputing STM, otherwise it goes to Step 6.

**Step 6:** the penalty parameter is updated if constraint violation worsen as explained in [2]. Go to Step 7.

**Step 7:** The input history, the nominal trajectory and the Lagrange multipliers are updated, together with the cost function and constraint violation. The new values are given from Step 4. Afterwards, the algorithm restarts from Step 1.
Methodology

As introduced during the Literature Review, Differential Algebra is the core of this work. It is used extensively in the algorithm and an overview of its capabilities is due in this chapter.

3.1 Differential Algebra

The main concept behind DA is to represent functions and operations between them on the computer, in an analogous way it is done for floating point numbers. To better represent this, the convention of Berz’s work [78] is adopted. The idea of these operators acting on equivalent classes is introduced in Figure 3.1 for floating point numbers, and extended for DA.

Applying an operator on $a, b \in \mathbb{R}$ gives a result $a * b \in \mathbb{R}$. Transforming $a, b \mapsto \overline{a}, \overline{b} \in \mathbb{FP}$ approximates these numbers due to truncation at a certain number of digits necessary to store them on the computer. The key step in ob-

![Figure 3.1: Simple transition diagram for real and floating point numbers, adapted from [78].](image-url)
3.1. Differential Algebra

Figure 3.2: Simple transition diagrams for functions and their equivalent class, adapted from [78].

The DA, operates similarly by applying this diagram (or an equivalent) on functions. First, the definition of $T$ is given, as the operator of extraction of Taylor expansion of a function up to arbitrary order $n$. Then, $T$ becomes an equivalence relation and the polynomials with equal coefficients up to order $n$ form the equivalent class. Starting from the Taylor coefficients of functions up to order $n$ of two equations, it is possible to obtain the Taylor coefficients up to order $n$ of their product, their product with a scalar and their sum. Therefore it is possible to create a Truncated Power Series Algebra [99]. Algorithmic equivalents of these operators, that act on a computer environment, were developed in a similar way as it was done for floating point numbers. Moreover, it was possible to extend this equivalence beyond basic operations, to derivatives and integrations of functions. The diagram in Figure 3.1 is now reformulated as the following diagram for functions in Figure 3.2.

The first structure from which DA stems, was that of Truncated Power Series Algebra (TPSA). Then, the concept of ordering was introduced, therefore TPSA was extended with differentiation and integration as it was possible to include a "differential" term. This extension was applied to the study of non Archimedean structures and this lead to the introduction of the Levi-Civita field [100]. Finally, automatic differentiation [101] is connected to DA as it was the first implementation of a code that would automatically add to the standard provided code, the generated code for evaluation of first order derivatives alongside it. Nonetheless, this discipline was restricted to first order differentiation only, as derivatives of higher order in many variables are problematic when they are numerically computed. The structure resulting from adding derivative and integration operators in TPSA is that of Differential Algebra.

To help with the understanding of the matter, the simplest Differential Algebra is proposed. The concept applied in the following section for functions of 1 variable and its Taylor equivalent class of order 1, can be readily extended (as
done in Berz [78]) to functions of order \( n \) in \( v \) variables.

### 3.1.1 \( 1D_1 \) Differential Algebra

This first nontrivial algebra is introduced by the set of pairs \( (q_0, q_1) \in \mathbb{R}^2 \). By redefinition of sum, scalar multiplication and vector multiplication as in Equation 3.1, \( 1D_1 \) structure is obtained.

\[
\begin{align*}
(q_0, q_1) + (r_0, r_1) &= (q_0 + r_0, q_1 + r_1); \\
t(q_0, q_1) &= (tq_0, tq_1); \\
(q_0, q_1)(r_0, r_1) &= (q_0r_0, q_1r_0 + q_0r_1).
\end{align*}
\] (3.1)

Given this set of operations, it can be observed that these three form an algebra. In fact, multiplication between vectors is commutative, associative and distributive with respect to addition. Moreover, the unity element of multiplication is found to be \((1, 0)\) (any number in \( 1D_1 \) multiplied by it stays the same). This algebra forms an extension of real numbers, as in the case of complex numbers. This can be verified by observing that \( \mathbb{R} \) elements are all contained in elements of \( 1D_1 \) with shape \((q_0, 0)\). Therefore, operations between these numbers as defined in Equation 3.1 remain in the subset of \( 1D_1 \) with shape \((q_0, 0)\).

Contrarily to complex numbers, \( 1D_1 \) is not a field as multiplicative inverse is only defined for particular cases in which \( q_0 \neq 0 \) and root is only defined if \( q_0 \geq 0 \). To verify these, one only needs to retrieve the solution to \((q_0, q_1)(x_0, x_1) = (1, 0)\) and finding \((x_0, x_1)\) corresponds to finding \((q_0, q_1)^{-1}\). Whereas, in the root case, simple combinations of the three basic operations lead to \( \sqrt{(q_0, q_1)} = (\sqrt{q_0}, \frac{q_1}{2\sqrt{q_0}}) \).

This structure is particularly similar to the one of complex numbers, where the only difference in operations would be having \((q_0, q_1)(r_0, r_1) = (q_0r_0 - q_1r_1, q_1r_0 + q_0r_1)\). In \( \mathbb{C} \) it is possible to define a particular number \( i \), which has specific properties. In \( 1D_1 \) it is possible to define another particular number \( d = (0, 1) \). This number has a special role and its properties can be explained easily after adding the concept of order to \( 1D_1 \).

The possibility of ordering numbers is what mainly differentiates this algebra from complex numbers, and gives it its important properties. The ordering criteria is total, hence given \((q_0, q_1)\) and \((r_0, r_1)\) only one of the following holds:

\[
\begin{align*}
(q_0, q_1) &< (r_0, r_1) \quad \text{if} \quad q_0 < r_0 \vee (q_0 = r_0 \wedge q_1 < r_1); \\
(q_0, q_1) &> (r_0, r_1) \quad \text{if} \quad (r_0, r_1) < (q_0, q_1); \\
(q_0, q_1) &= (r_0, r_1) \quad \text{if} \quad q_0 = r_0 \wedge q_1 = r_1.
\end{align*}
\] (3.2)

The last two important properties of this ordering is that it embeds the order of real numbers, and also that it is compatible with addition and multiplication (e.g. if \( \overline{q} < \overline{r} \implies \overline{q} + \overline{s} < \overline{r} + \overline{s} \) and \( \overline{q} < \overline{r}, \overline{s} > 0 \implies \overline{qs} < \overline{rs} \)).

After extending with the order, the properties of the number \( d \) become clearer. As a matter of fact, it is a number greater than 0 but smaller than any real number for the properties in Equation 3.2. Moreover its squared is 0 according to the defined multiplication in Equation 3.1. Therefore, \( d \) is infinitely small.
and it is called "differential". Any number in \( D_1 \) can now be represented as: \( q_0 + dq_1 = (q_0, 0) + (0, q_1) \), in other words real plus differential part. The possibility to define this differential grants the possibility to define exact derivatives:

\[
\partial : D_1 \rightarrow D_1 | \partial(q_0, q_1) = (0, q_1);
\]

\[
\partial((q_0, q_1)(r_0, r_1)) = \partial(q_0r_0, q_1r_0 + q_0r_1) = (0, q_1r_0 + q_0r_1) =
\]

\[
= (0, q_1)(r_0, r_1) + (q_0, q_1)(0, r_1) = \partial(q_0, q_1)(r_0, r_1) + (q_0, q_1)\partial(r_0, r_1);
\]

\[
(3.3)
\]

\[
\partial((q_0, q_1) + (r_0, r_1)) = \partial(q_0 + r_0, q_1 + r_1) = (0, q_1 + r_1) =
\]

\[
= (0, q_1) + (0, r_1) = \partial(q_0, q_1) + \partial(r_0, r_1);
\]

Adding \( \partial \) operator to \( D_1 \) transforms it into a differential algebra.

This useful introduction is given to understand that by applying \( D_1 \) operations to functions, automatic derivatives are found. In fact, by storing \( f \) in \( q_0 \) and \( f' \) in \( q_1 \) (using notation \([f] = (f, f')\)), it is possible to repeat the fore mentioned results:

\[
\partial((f, f')(g, g')) = \partial(fg, f'g + g'f) = (0, f'g + g'f) =
\]

\[
= (0, f')(g, g') + (f, f')(0, g') = \partial(f, f')(g, g') + (f, f')\partial(g, g') = f'g + g'f;
\]

\[
(3.4)
\]

\[
\partial((f, f') + (g, g')) = \partial(f + g, f' + g') = (0, f' + g') =
\]

\[
= (0, f') + (0, g') = \partial(f, f') + \partial(g, g') = f' + g';
\]

If this formulation is used, applying operations on \([f] \) and \([g] \) yields results that contain the derivative of the function resulting from the operation. Therefore complicated derivatives can be now taken by applying simple algebraic rules (namely Equation 3.1). An example of this is given in Appendix B.

### 3.1.2 \( nD_v \) General Case

The differential algebra \( D_1 \) can be extended with some efforts to \( nD_v \) as performed by Berz [78]. As we had derivatives of order 1 for functions of 1 variable in the simplest differential algebra, this will perform ideally the same task, giving us the coefficients of the Taylor expansion of a function in \( v \) variables, up to order \( n \). The equivalence class that is now used ideally in Figure 3.2 is the class of all the functions whose Taylor expansion coefficients up to order \( n \) in \( v \) variables agree and are the same. These equivalence classes are stored as DA vectors. The number of coefficients to be stored increases dramatically with the order and the number of variables. The dependency is the same: observe this by considering a quadratic function in 1 variable (3 coefficients) and a bilinear function in 2 variables (still 3 coefficients). The number of coefficients of the Taylor expansion than can be stored are limited mainly by memory allocation.

In a similar way as it was possible for \( D_1 \), \( nD_v \) can be totally ordered, with rules that are compatible with addition and multiplication. Its first coefficient is still the number that maps the reals in the new algebra, and any equivalent class
Chapter 3. Methodology

that has null leading term, is infinitesimal. The way the order is checked in this new generalized case, is by checking the leading term, and then proceeding from low order to high order until a discrepancy in values is found (lexicographic order).

After the concept of order and infinitesimal have been introduced, it is possible to verify that the most important operators in $nD_v$ are contracting on this set (or on one of its subsets). A contracting operator is one for which the derivatives of some elements in $nD_v$ agree to an higher order than before the application of such operator.

The important consequence brought by this, is that these operators satisfy a fixed point theorem, that makes it possible to state that all these operators (sum, multiplication, inverse, anti-derivative etc.) have a unique solution in $nD_v$ and they converge to it in finitely many steps. This fixed point is an element $a$ in $nD_v$ for which a contracting operator $O$ gives $O(a) = a$. Given that the derivatives agree to an higher order after every iteration, this number is at most $n + 1$.

This formulation is particularly useful considering that it makes it possible to formulate algorithms implementation of these operators, knowing the effort necessary for the computation a priori.

As explicitly stated in the introduction to this work, the modified code extensively exploits DA through DACE software. This is a software with DA core, based on COSY Infinity [101]. The order of the computation and the number of variables must be set at the beginning of the computer execution.

3.2 HDDP Based on Polynomial Maps

The general outline of the modifications implemented to HDDP is given in this section.

The differential algebra introduced in section 3.1 makes it possible to compute the derivatives of a function $f$ in $v$ variables up to order $n$ along with the computation of the function. This has important consequences when the dynamical function that maps the states from stage $k$ to stage $k+1$ is obtained via numerical integration. Notation taken from Di Lizia et al. [73] is adopted in the following dissertation.

Without a loss of generality, one can express the expansion of an ODE in one variable as:

$$\begin{cases} \dot{x} = f(x); \\ x(t_0) = x_0; \end{cases} \quad (3.5)$$

The solution of this ODE equation requires algebraic operations to be performed, together with the evaluation of $f$ at several time instants. Thanks to this, if the initial point is initialized as its constant part plus the DA identity (i.e. the Taylor expansion of its identity function) $[x_0] = x_0 + \delta x_0$, then the Taylor expansion of the solution at each integration step is obtained as a function of variation from reference initial conditions. The procedure is described with a first order Euler integration scheme but any ODE scheme ideally exploits the same algebraic
3.2. HDDP Based on Polynomial Maps

operations.

Using a first order scheme as $x_{k+1} = x_k + f(x_k)\Delta t$, then by the properties of DA, described in section 3.1, this new value can be expressed as $[x_{k+1}] = [x_k] + f([x_k])\Delta t$. The extraction of the operator $f([x_k])$ gives the Taylor expansion of the function $f$ in the neighborhood of the starting condition $x_k$ as a function of $\delta x_k$, expressed as $f(x_k) + \mathcal{M}_f(\delta x_k)$ (constant and differential part of the expansion).

The remaining algebraic operations help to compute $[x_{k+1}] = x_{k+1} + \mathcal{M}_{x_k}(\delta x_k)$ which represents the expansion of $x_{k+1}$ with respect to the initial value $\delta x_k$ about the reference point $x_{k+1}$. If the procedure is continued, then at the step $k + 2$:

$$
[x_{k+2}] = [x_{k+1}] + \Delta tf([x_{k+1}]) = x_{k+1} + \mathcal{M}_{x_{k+1}}(\delta x_k) + \Delta tf(x_{k+1} + \mathcal{M}_{x_{k+1}}(\delta x_k));
$$

$$
[x_{k+2}] = x_{k+2} + \mathcal{M}_{x_{k+2}}(\delta x_k);
$$

This procedure gives, $x_{k+2} = x_{k+2} + \mathcal{M}_{x_{k+2}}(\delta x_k)$ and can be repeated per each integration step from $\delta x_0$ up to the desired integration interval $N$, so that the final result is: $[x_N] = x_N + \mathcal{M}_{x_N}(\delta x_0)$.

This result comes from the fact that propagating ODE system in the DA framework only requires evaluations of the right hand side, algebraic operations and composition of DA polynomials. In particular, this result holds its validity for multi-variable functions propagated through higher order ODE schemes.

In HDDP, this is exploited for estimating the partials removing the STM computation step (i.e. Step 1 of the algorithm in section 2.3.7).

During the forward sweep of the algorithm (Step 4 of the algorithm in section 2.3.7), the variables at step $k$ are initialized as their value on the reference trajectory plus their DA identity, one per each independent variable needed (i.e controls, states, Lagrange multipliers).

$$
[x_k] = \bar{x}_k + \delta x_k;
$$
$$
[u_k] = \bar{u}_k + \delta u_k;
$$
$$
[\omega] = \bar{\omega} + \delta \omega;
$$
$$
[\lambda] = \bar{\lambda} + \delta \lambda;
$$

Then the dynamic equation of the system as in Equation 2.3 is applied to these DA objects and the result is:

$$
[x_{k+1}] = F([x_k], [u_k], [\omega]) = x_{k+1} + \mathcal{M}_{x_{k+1}}(\delta x_k, \delta u_k, \delta \omega);
$$

Notice how, regardless of the shape of $F$, being it analytical or numerically integrated via ODE, this expression retains its value and guarantees the correct partials.

This equation represents the nominal value of the new state and its high order polynomial expansion $\mathcal{M}_{x_{k+1}}$ as a high order polynomial of $\delta x_k, \delta u_k, \delta \omega$ about the reference starting condition at instant $k$. At each stage $k$ these maps are stored, they contain all the information on partial derivatives needed for propagation of partials of the cost function.
3.2.1 Backward Sweep on Stages

When treating the backward sweep across the stages of a phase, the index of the phase is removed to reduce notation complexity.

At the beginning of the backward sweep, the state $x_{N+1}$ is available as well as the value of Lagrange multipliers $\lambda$, and they are augmented with their DA identity functions generating $[x_{N+1}]$ and $[\lambda]$. Without loss of generality, the final cost can be expressed exploiting Equation 2.25 as:

$$\tilde{\phi} = \tilde{\phi}([x_{N+1}], [\lambda], [\omega]) = \tilde{\phi} + M_{\phi}(\delta x_{N+1}, \delta \lambda, \delta \omega); \quad (3.9)$$

In the forward sweep, the transition maps $\delta x_{k+1} = M_{x_{k+1}}(\delta x_k, \delta u_k, \delta \omega)$ were stored, therefore it is possible to retrieve the last map $M_{x_{N+1}}$ for usage. To start the backward sweep, this polynomial map is composed with that of $\tilde{\phi}$, allowing for the expression of:

$$[J_{N+1}] = \tilde{\phi} + M_{\phi}(\delta x_{N+1}, \delta \lambda, \delta \omega) = \tilde{\phi} + M_{\phi}(M_{N+1}(\delta x_N, \delta u_N, \delta \omega), \delta \lambda, \delta \omega); \quad (3.10)$$

To the last term of this equation, the last stage cost (provided there is one) must be added $[L_N] = L_N + M_{L_N}(\delta x_N, \delta u_N, \delta \omega)$.

$$[J_N] = [J_{N+1}] + [L_N] = J_N + M_{J_N}(\delta x_N, \delta u_N, \delta \omega, \delta \lambda); \quad (3.11)$$

The next step of the algorithm, is to solve the optimization problem: the gradient of $J_N$ with respect to $\delta u_N$ must be set to 0. Therefore, a control law can be retrieved by exploiting DA maps inversion.

The derivatives of $J_N$ with respect to $\delta u_N$ can be taken in the neighborhood of $\tilde{u}_k$ thanks to the differentiation operator available in DACE, and then they can be set to zero:

$$[J_{u,N}] = J_{u,N} + M_{J_{u,N}}(\delta x_N, \delta u_N, \delta \omega, \delta \lambda) = 0; \quad (3.12)$$

The feedback law $\delta u_N(\delta x_N, \delta \omega, \delta \lambda)$ is now obtained via map inversion as in Equation 3.14. To guarantee the feasibility of this inversion, some identity polynomial maps have been introduced in order to exploit the inversion operator available in DACE. Such operator is based on reducing the inversion problem to a fixed-point problem.

$$\begin{pmatrix} \delta J_{u,N} \\ \delta x_N \\ \delta \omega \\ \delta \lambda \end{pmatrix} = \begin{pmatrix} M_{J_{u,N}} \\ I_{x_N} \\ I_{\omega} \\ I_{\lambda} \end{pmatrix}^{-1} \begin{pmatrix} \delta x_N \\ \delta u_N \\ \delta \omega \\ \delta \lambda \end{pmatrix}; \quad (3.13)$$

$$\begin{pmatrix} \delta x_N \\ \delta u_N \\ \delta \omega \\ \delta \lambda \end{pmatrix} = \begin{pmatrix} M_{J_{u,N}} \\ I_{x_N} \\ I_{\omega} \\ I_{\lambda} \end{pmatrix} \begin{pmatrix} \delta J_{u,N} \\ \delta x_N \\ \delta \omega \\ \delta \lambda \end{pmatrix}; \quad (3.14)$$
3.2. HDDP Based on Polynomial Maps

\[ \delta u_N = \delta u_N(\delta J_{u,N}, \delta x_N, \delta \omega, \delta \lambda); \]  (3.15)

The polynomials relative to \( \delta u_N \) are extracted and their coefficients stored for usage in the forward sweep, after their evaluation in \( \delta J_{u,N} = -J_{u,N} \) to guarantee that the feedback law extremizes the cost. The final input expression is, therefore, Equation 3.16.

\[ \delta u^*_N = \delta u^*_N(\delta x_N, \delta \omega, \delta \lambda); \]  (3.16)

Recovering now the DA function \( J_N \), it is possible to compose the polynomial \( J_N \) with these maps of \( \delta u_N \) and obtain the optimized cost as Equation 3.17.

\[
\begin{align*}
J^*_N &= J_N + M J_N (\delta x_N, \delta u^*_N(\delta x_N, \delta \omega, \delta \lambda), \delta \omega, \delta \lambda) = J^*_N + M J^*_N (\delta x_N, \delta \omega, \delta \lambda); \\
ER_N &= [J_N] - [J^*_N] = J_N - J^*_N; \\
\end{align*}
\]  (3.17)

Once the step \( N \) is performed, the cost is propagated backward:

\[
\begin{align*}
[J_{N-1}] &= L_{N-1} + M L_{N-1} (\delta x_{N-1}, \delta u_{N-1}, \delta \omega) + J^*_N = J_{N-1} + M J^*_N (\delta x_{N-1}, \delta u_{N-1}, \delta \omega, \delta \lambda) = J_{N-1} + M J_{N-1} (\delta x_{N-1}, \delta u_{N-1}, \delta \omega, \delta \lambda); \\
\end{align*}
\]  (3.19)

Having showed how to start the iteration process, at the generic step \( k \) the process performs the following steps:

- \([J_k] = [L_k] + [J^*_{k+1}] \) is composed with the dynamic mappings obtained during the forward sweep and stored \( \delta x_{k+1} = M x_{k+1} (\delta x_k, \delta u_k, \delta \omega, \delta \lambda) \) to obtain \( [J_k] = J_k + M J_k (\delta x_k, \delta u_k, \delta \omega, \delta \lambda) \).

- Derivatives of \([J_k] \) are extracted and the gradient with respect to the controls is equated to zero as in Equation 3.12.

- The gradient \( \delta J_{u,k} \) is augmented with identity maps to guarantee inversion and inverted as in Equation 3.14.

- The feedback law in polynomial form is evaluated in \( \delta J_{u,k} = -J_{u,k} \) and the feedback polynomial law is substituted in \( J_k \) to obtain \( J^*_k \) as in Equation 3.17.

- The expected reduction is obtained by extracting the difference between \( J_k \) and \( J^*_k \) on the reference trajectory (i.e. \( \delta x_k = 0, \delta \omega = 0, \delta \lambda = 0 \)), the algorithm proceeds to \( k - 1 \).

The process can be repeated per each step until step 0 of the phase. It is important to notice how this method, when the arbitrary order is set to \( n = 2 \), retrieves exactly the same partials mapping and linear feedback law as pure HDDP, without the cumbersome formulation of partial derivatives of dynamic functions, or integration of a system of ODE of size \( N + N^2 + N^3 \).
3.2.2 Constraints Handling Techniques

The constraint handling techniques are the same as HDDP but with slight modifications. To enforce null-space methods as in the standard HDDP for control bounds, the nominal polynomial feedback is evaluated on the reference trajectory (i.e. $\delta x_k = 0, \delta \omega = 0, \delta \lambda = 0$). If a constraint is violated, the polynomial function is substituted with a constant $\delta u_k^*$ that fixes the controls at the next iteration on the control bounds, the control is removed from the set of active controls and the process is repeated with the remaining constraints (e.g. Mono-Dimensional Landing Problem in section 4.2).

The treatment of nonlinear constraints requires the introduction of other DA variables, the Lagrange multipliers of stage nonlinear active constraints $\nu_k$. The set of active constraints are estimated by checking if the nominal $\delta u_k^*$ violates them exactly as in HDDP. Then, the Lagrangian $L$ is constructed adding the term $\nu_k (g_k)$ to the cost function, and it is minimized with respect to $\delta u_k$ and $\delta \nu_k$.

\[
\begin{bmatrix}
\delta L_{u,k} \\
\delta L_{v,k} \\
\delta x_k \\
\delta \omega \\
\delta \lambda \\
\delta \nu_k 
\end{bmatrix} = \begin{bmatrix}
\mathcal{M}_{L_{u,k}} \\
\mathcal{M}_{L_{v,k}} \\
I_x \\
I_\omega \\
I_\lambda \\
-1
\end{bmatrix} \begin{bmatrix}
\delta x_k \\
\delta \omega \\
\delta \lambda \\
\delta \nu_k 
\end{bmatrix}; \quad (3.20)
\]

\[
\begin{bmatrix}
\delta x_k \\
\delta u_k \\
\delta \omega \\
\delta \lambda \\
\delta \nu_k 
\end{bmatrix} = \begin{bmatrix}
\mathcal{M}_{L_{u,k}} \\
\mathcal{M}_{L_{v,k}} \\
I_x \\
I_\omega \\
I_\lambda 
\end{bmatrix}^{-1} \begin{bmatrix}
\delta L_{u,k} \\
\delta L_{v,k} \\
\delta x_k \\
\delta \omega \\
\delta \lambda 
\end{bmatrix}; \quad (3.21)
\]

The optimal constrained feedback is obtained by evaluating $\delta u_k$ in $\delta L_{u,k} = -L_{u,k}$ and $\delta L_{v,k} = -L_{v,k}$. The process can then continue as before, noting that there is no need to store these Lagrange multipliers as they are not needed during the iteration.

3.2.3 Backward Sweep on Phases

To extend the backward sweep across phases after all the stages on phase $i$ have been minimized, another composition of maps is necessary. At the end of a phase, it is necessary to initialize other variables as DA variables, adding their identity polynomials. In particular, there are $\delta \omega_i, \delta \omega_{i-1}, \delta \lambda_i, \delta \lambda_{i-1}$, and $\delta x_i$.

The notation is here simplified again, values indexed $-$ refer to multipliers, controls and states pertinent to phase $i-1$, whereas the index $+$ refers to the same quantities at phase $i$. First of all, the initial condition of a stage are exploited. At the end of the sweep on phase $i$, the cost is:

\[
[J_+^*] = J_+^* + \mathcal{M}_{J_+^*} (\delta x_+, \delta \omega_+, \delta \lambda_+); \quad (3.22)
\]
To this cost, it is necessary to add the termination cost of phase $i - 1$ as in Equation 2.25 which, thanks to DA, is expressed as:

$$[	ilde{\phi}_-] = \tilde{\phi}_- + \mathcal{M}_{\tilde{\phi}_-}(\delta x_- , \delta \omega_- , \delta x_+ , \delta \omega_+ , \delta \lambda_-);$$  

(3.23)

Therefore, in a similar way as for HDDP, applying $[\Gamma(\omega_+)] = [x_+]$ and composing the cost function with it, gives:

$$[J_-] = \tilde{J}_+ + \tilde{\phi}_- + \mathcal{M}_{\tilde{\phi}_-}(\delta x_- , \delta \omega_- , \delta x_+ , \delta \omega_+ , \delta \lambda_-) + \mathcal{M}_{\tilde{J}_+}(\delta x_+ , \delta \omega_+ , \delta \lambda_+);$$  

(3.24)

$$[J_-] = \tilde{J}_+ + \mathcal{M}_{\tilde{J}_-}(\delta x_- , \delta \omega_- , \delta \omega_+ , \delta \lambda_- , \delta \lambda_+);$$  

(3.25)

This substitution effectively removes the dependency from $\delta x_+$. To retain the same scheme of HDDP, and obtaining the same results when $n = 2$, the problem is uncoupled as it was done in section 2.3.5.

It is clear that the derivatives with respect to $\delta \lambda_\pm$ will only present mixed terms with $\delta \omega_\pm$ as the only component of Equation 3.25 in which $\delta \lambda_\pm$ appears is Equation 3.22. Therefore, the following solution is setup by equating the gradient of this equation with respect to $\delta \lambda_\pm$ to zero.

$$\begin{bmatrix}
(\delta J_{\lambda_-, \pm})_+ \\
(\delta \lambda_+) \\
\delta \omega_+
\end{bmatrix} = \begin{bmatrix}
\mathcal{M}_{\lambda_+, \pm} \\
\mathcal{I}_{\omega_+}
\end{bmatrix} \begin{bmatrix}
(\delta \lambda_+) \\
(\delta \omega_+)_-
\end{bmatrix};$$  

(3.26)

$$\begin{bmatrix}
(\delta J_{\lambda_+, \pm})_+ \\
(\delta \lambda_+) \\
\delta \omega_+
\end{bmatrix} = \begin{bmatrix}
\mathcal{M}_{\lambda_+, \pm} \\
\mathcal{I}_{\omega_+}
\end{bmatrix}^{-1} \begin{bmatrix}
(\delta J_{\lambda_-, \pm})_+ \\
(\delta \lambda_+) \\
\delta \omega_+
\end{bmatrix};$$  

(3.27)

This equation retrieves a feedback law for $\delta \lambda_+ (\delta \omega_+)$ by evaluating it in $\delta J_{\lambda_-, \pm} = -J_{\lambda_+, \pm}^*$. By composing this polynomial map with Equation 3.25, Equation 3.28 is retrieved.

$$[J^*_1] = J^*_1 + \mathcal{M}_{J^*_{1+}}(\delta x_- , \delta \omega_- , \delta \omega_+ , \delta \lambda_-);$$  

(3.28)

Once again, the feedback law for $\delta \omega_\pm$ is obtained by equating to zero the gradient of Equation 3.28 with respect to these variables:

$$\begin{bmatrix}
(\delta J_{\omega_-, \pm})_1 \\
(\delta \omega_+) \\
\delta x_- \\
\delta \omega_- \\
(\delta \lambda_-)
\end{bmatrix} = \begin{bmatrix}
\mathcal{M}_{\omega_+, \pm} \\
\mathcal{I}_{x_-} \\
\mathcal{I}_{\omega_-} \\
\mathcal{I}_{\lambda_-}
\end{bmatrix} \begin{bmatrix}
(\delta \omega_+) \\
(\delta x_-) \\
(\delta \omega_-) \\
(\delta \lambda_-)
\end{bmatrix};$$  

(3.29)

$$\begin{bmatrix}
(\delta J_{\omega_+, \pm})_1 \\
(\delta \omega_+) \\
\delta x_- \\
\delta \omega_- \\
(\delta \lambda_-)
\end{bmatrix} = \begin{bmatrix}
\mathcal{M}_{\omega_+, \pm} \\
\mathcal{I}_{x_-} \\
\mathcal{I}_{\omega_-} \\
\mathcal{I}_{\lambda_-}
\end{bmatrix}^{-1} \begin{bmatrix}
(\delta J_{\omega_-, \pm})_1 \\
(\delta \omega_+) \\
(\delta x_-) \\
(\delta \omega_-) \\
(\delta \lambda_-)
\end{bmatrix}. $$

(3.30)

This is the final step in the phase optimization procedures, as now the feedback law of $\delta \omega_\pm (\delta x_- , \delta \omega_- , \delta \lambda_-)$ can be retrieved by imposing $\delta J_{\omega_-, \pm}^{*1} = -J_{\omega_+, \pm}^{*1}$. If this feedback law is substituted into Equation 3.28, the optimal function at the beginning of phase $i - 1$ is found, and the stages backward sweep can re-start:

$$[J^*_1] = J^*_1 + \mathcal{M}_{J^*_{1+}}(\delta x_- , \delta \omega_- , \delta \omega_+ (\delta x_- , \delta \lambda_- , \delta \omega_- , \delta \lambda_-) = J^*_1 + \mathcal{M}_{J^*_{1-}}(\delta x_- , \delta \omega_- , \delta \lambda_-);$$  

(3.31)
Chapter 3. Methodology

The expected reduction of this step can still be computed as the difference between Equation 3.28 and Equation 3.28 after substituting the optimal $\delta \omega_+$ feedback law and evaluating it on the reference trajectory (all variations set to 0).

$$ER_- = [J_-] - [J^*_-] = J_- - J^*_-$; \quad (3.32)$$

Moreover, the feedback law for $\delta \omega_+$ is now available, and the updating scheme of the Lagrange multipliers can be obtained as composition of:

$$\delta \lambda_+ (\delta \omega_+); \quad (3.33)$$
$$\delta \omega_+ (\delta x_-, \delta \lambda_-, \delta \omega_-); \quad (3.34)$$
$$\delta \lambda_+ = \delta \lambda_+ (\delta x_-, \delta \lambda_-, \delta \omega_-)). \quad (3.35)$$

3.2.4 Trust Region Algorithm

For the treatment of Trust Region limitation and to guarantee a descent direction, the same approach as standard HDDP is used. The TRQP algorithm is applied on the Hessians and gradient. Afterwards, the cost function is modified by adding a shift of $\gamma$ to the coefficients of second order of the cost function $J$, so that the new modified cost $J_m$ has the same Hessian as the shifted one. To conclude, the gradient of the modified cost is taken equal to 0, and the feedback laws are retrieved.

For example to shift the input Hessian the procedure is the following:

- Starting from $[J_k]$, the derivatives with respect to $\delta u_k$ are stored in the gradient and the Hessian matrix.
- These matrices are used in the TRQP procedure.
- The $[J_m]$ function is obtained: $[J_m] = J_k + \gamma \delta u_k^T \delta u_k$.
- This modified $J$ is used to obtain the feedback law, by imposing the gradient to be zero.
- The algorithm proceeds by substituting this feedback into the cost function $[J_k]$ to obtain $[J^*_k]$.

This algorithm poses a limitation on the size of the quadratic trust-region, even for higher orders. Future research directions aim at modifying this step to exploit the full potential of DA and higher orders.
Applications

To test the outcome of the modified algorithm, four examples are presented in order of increased complexity. The first case is a Linear Quadratic problem to test convergence of the algorithm. The second case is a mono-dimensional landing problem. Subsequently, an Earth-Mars transfer is studied. At last, A Peer-to-Peer satellite constellation refueling maneuver is studied.

4.1 Linear Quadratic Problem

The solution of this exercise is implemented according to Lantoine and Russell [53] with some minor modifications. This example is known to converge for HDDP, the preliminary test is to assess whether DA introduction spoiled convergence. This kind of problems is linear in the controls and quadratic in cost, moreover it has linear constraints. This particular structure makes it possible to retain quadratic augmented cost, thanks to the linearity of the constraints. Therefore, these problems should converge in only one iteration for methods based on augmented Lagrangian.

This result is obtained by Powell in [37].

Similarly to the test case in [53], this example exploits 2 phases \((M = 2)\) and 5 stages per each phase \((N_1 = N_2 = 5)\). The transition functions \(F_{i,j}\) are defined as:

\[
x_{i,j+1} = F_{i,j}(x_{i,j}, u_{i,j}) = \begin{bmatrix} r_{i,j+1} \\ v_{i,j+1} \end{bmatrix} = \begin{bmatrix} r_{i,j} + v_{i,j} \\ v_{i,j} + u_{i,j} \end{bmatrix};
\]  

\(4.1\)
The stage constraints $g_{i,j}$ are not present, on the contrary the phase constraints $\psi_1$ at the end of phase 1 and $\psi_2$ at the end of phase 2 are defined as follows:

$$\psi_1 = x_{2,1} - x_{1,6} = 0;$$
$$\psi_2 = x_{2,6} - x_t = 0;$$

(4.2)

The final objective state here is defined by $x_t$ and it is a targeted point at the end of the second phase.

The first phase constraint is just imposing continuity between the last stage of the first phase and the starting stage of the second phase.

The constant controls $\omega$ are defined only on the second phase, as in the first phase every parameter is fixed. $\omega_2$ are used to parametrize the initial conditions of the second phase as $x_{2,1} = \Gamma(\omega_2) = \omega_2$.

Finally the stage cost function is formulated as $L_{i,j} = ||u_{i,j}||^2$. The first guess of this algorithm needed in Step 0 of HDDP in section 2.3.7 is provided in the following results section.

### 4.1.1 Results

The algorithm converges in exactly one iteration.

The problem is set up in a way to permit the user modification of number of phases and stages in the process. The results are shown with 2 phases and 5 stages each, but this number can be modified accordingly. As the time step is fixed, increasing $N$ or $M$ corresponds to increasing the total time of the process, yielding smaller input and therefore smaller cost function.

The target point is $x_t = [2.0, 4.0, 1.0, -0.5, 1.5, -2.5]$, whereas the initial point of the algorithm is set to $x_0 = [1.0, 1.0, 1.0, 1.0, 1.0, 1.0]$, while the first guess for inputs and Lagrange multipliers is all zeros. Also the initial conditions for constant controls on phase 2, that corresponds to the initial condition of the states of phase 2, are set to zero. As it can be observed in Figure 4.1, the continuity constraints between phases are respected, moreover the final target is achieved in 1 iteration only. The result is consistent if the number of stages or phases is changed and the convergence is achieved in exactly 1 iteration. Several trials with different target points, different $M$ and $N$ have reported the same result. Finally, the input trajectory is reported in the following Figure 4.2.

For completeness, also the value of the Lagrange multipliers for the phase constraints are reported in Table 4.1. To conclude this validation example, also the constant controls of phase 2 are reported in Table 4.2.

Augmenting the order of the example does not add any benefit to the formulation of the problem, as it is already converging in 1 iteration.

### Table 4.1: Lagrange multipliers’ values for the Liner Quadratic Problem.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Lagrange Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0.0545455 0.224242 -0.139394 0.327273 0.0121212 0.630303]</td>
</tr>
<tr>
<td>2</td>
<td>[0.0545455 0.224242 -0.139394 0.0545455 -1.10909 1.32727]</td>
</tr>
</tbody>
</table>
Chapter 4. Applications

Figure 4.1: Evolution of states for the Linear Quadratic Problem.

Figure 4.2: Evolution of controls for the Linear Quadratic Problem.

Table 4.2: Constant controls values for the Linear Quadratic Problem.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Constant Controls $\omega$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$[3.5455 2.5758 4.9394 -0.0909 -0.1515 0.1212]$</td>
</tr>
</tbody>
</table>
4.2 Mono-dimensional Landing

The dynamic formulation of this problem is reprised from Lantoine and Russell [55], and the same problem is presented also by Pellegrini [90]. The objective function is formulated as $J_1 = -m_f$, hence with phase cost $\phi = -m(t_f)$ and no stage cost $L_k$. For comparison purposes, the same problem is solved for $J_2 = \sum_{i=1}^N ||u_i||^2$, i.e. no phase cost but stage cost $L_k = ||u_k||^2$. The dynamics that need to be integrated are:

$$
\begin{bmatrix}
\dot{x} \\
\dot{v} \\
\dot{m}
\end{bmatrix} =
\begin{bmatrix}
v \\
-g + \frac{T}{m} \\
-\frac{T}{g_0I_{sp}}
\end{bmatrix}; \quad (4.3)
$$

There is only one phase considered ($M = 1$), with $N = 10$ stages. The stage constraints $g_i$ are control bounds $u^L \leq u_i < u^U$ with $u^L = 0$ and $u^U = 1.227$. They can be treated with null space method or with range space active set method as explained in section 2.3.4. The phase constraint $\psi$ is:

$$
\psi = \begin{bmatrix} x(t_f) \\ v(t_f) \end{bmatrix} = 0; \quad (4.4)
$$

Finally the initial conditions and parameters are:

$$
x(0) = 1; \quad v(0) = -0.783; \quad m(0) = 1; \quad TOF = 1.397; \quad g_0I_{sp} = 2.349; \quad g = 1; \quad (4.5)
$$

No constant controls $\omega$ are introduced in this example. The provided model of the system is already scaled for improved convergence. To conclude, the set of tuning parameters for this example are reported in Appendix A. The integration scheme used for the dynamics is a direct Euler first order method, which is sufficient for this example.

4.2.1 Results

First of all, the results for second order are reported, which should yield the same results as in [55] and [90].

The final norm of constraint violation is $f = 7.6952 \cdot 10^{-6}$, while the final mass is $m_f = 0.392$ and the results are consistent with the ones found in literature. The evolution of the height $h$ and the velocity $v$ are represented in Figure 4.3, whilst the mass evolution $m$ is represented in Figure 4.5.

The next step is to analyze the behavior of the solution for different orders of expansion. This task is performed in Table 4.3. Orders higher than 6, do not converge for this problem. Heuristics and experience indicate that an high-order feedback controller behaves better when close to the solution of the problem. Bearing in mind that the starting position is far enough from the optimal solution, it is possible to observe that order 2 behaves better than the others. While at the beginning of the algorithm the solver is far from the solution, higher-order
Chapter 4. Applications

Figure 4.3: Evolution of adimensional states for the Mono-Dimensional Landing Problem.

Figure 4.4: Evolution of adimensional control for the Mono-Dimensional Landing Problem.
4.2. Mono-dimensional Landing

Figure 4.5: Evolution of the normalized mass for the Mono-Dimensional Landing Problem.

Table 4.3: High order effects on the Mono-Dimensional Landing Problem.

<table>
<thead>
<tr>
<th>Order</th>
<th>Outcome</th>
<th>Iterations</th>
<th>Iterations</th>
<th>Constraint Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Accepted</td>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Success</td>
<td>87</td>
<td>124</td>
<td>$7.6952 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>3</td>
<td>Success</td>
<td>91</td>
<td>137</td>
<td>$8.64365 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>4</td>
<td>Success</td>
<td>71</td>
<td>83</td>
<td>$8.47425 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>5</td>
<td>Aborted</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Success</td>
<td>213</td>
<td>258</td>
<td>$8.85748 \cdot 10^{-6}$</td>
</tr>
</tbody>
</table>

exhibits greater violation of constraints. On the other hand, after some iterations, the higher order tend to have smaller violation of constraints, due to the proximity to the optimal trajectory. For really high orders, numerical difficulties arise, spoiling the convergence. As it can be already seen for order 6, feedback terms of high order tend to render the constraint violation reduction more noisy. Probably, order 6 is used outside the convergence radius of the polynomial, resulting in this behavior. On the contrary, the examples up to order 4 show quite a good result. To conclude, if the polynomials are used properly (inside their region of convergence), some improvements with respect to order 2 may arise. However, these enhancements were not significant in this particular case, therefore the use of higher than second order is not justified.
Figure 4.6: Norm of constraint violation $f$ for the Mono-Dimensional Landing Problem for orders 2(a), 3(b), 4(c) and 6(d).
4.3 Earth-Mars Transfer

The problem is again reprised from Lantoine and Russell [53]. The dynamical model for this system is inspired by Sims-Flanagan model [102].

Sims-Flanagan Model

The main idea behind this model, is to lump the effect of continuous low-thrust arcs as concentrated impulses at the discretization points, and afterwards the propagation of orbit exploits simple coast arcs with $f$ and $g$ procedure developed by Bate et al. [103]. In doing so, numerical integrations can be avoided. To include perturbations, it is possible to add their integrated contribution to the $\Delta v$ caused by the thrusters. However, this model is intended for use with near-Keplerian problems only. Validity of the state transition matrices has been assessed by comparison with their analytical counterparts. For the analytical derivation, the work done by Pitkin has been retrieved [104].

Constant parameters for the problem are herein reported in Equation 4.6.
\[
T_{\text{max}} = 0.5 \text{ N}; \\
TOF = 348.79 \text{ d}; \\
I_{sp} = 2000 \text{ s}; \\
\mu_{\odot} = 1.327 124 400 18 \times 10^{11} \text{ km}^{3}\text{s}^{-2};
\tag{4.6}
\]

The problem studied is single phase($M = 1$), and has $N = 40$ stages. According to Lantoine and Russell, the state vector is composed of the Cartesian components of position $\mathbf{r}$, Cartesian components of velocity $\mathbf{v}$ and mass $m = [\mathbf{r}, \mathbf{v}, m]$. The starting date as reported by [53] is the 10th of April 2007. Initial data were obtained using JPL ephemerides DE405 model, and are reported in Equation 4.7.
\[
\mathbf{r}_0 = \begin{bmatrix} -140699693 \\ -51614428 \\ 980 \end{bmatrix} \text{ km}; \\
\mathbf{v}_0 = \begin{bmatrix} 9.774596 \\ -28.07828 \\ 4.337 725 \times 10^{-4} \end{bmatrix} \text{ km s}^{-1};
\tag{4.7}
\]
\[
m_0 = 1000 \text{ kg};
\]

The final target state is the position and velocity of Mars at the end of the transfer, again obtained via DE405 model and reported in Equation 4.8.
\[
\mathbf{r}_M(t_f) = \begin{bmatrix} -172682023 \\ 176959469 \\ 7948912 \end{bmatrix} \text{ km}; \\
\mathbf{v}_M(t_f) = \begin{bmatrix} -16.427384 \\ -14.860506 \\ 9.214 86 \times 10^{-2} \end{bmatrix} \text{ km s}^{-1};
\tag{4.8}
\]

The state transition function applied is obtained via the $f, g$ approach formulated by Bate et al.[103], and it is reported in Equation 4.9.
\[
\begin{bmatrix} \mathbf{r}_{k+1} \\ \mathbf{v}_{k+1} \\ m_{k+1} \end{bmatrix} = \begin{bmatrix} f(\mathbf{r}_k, \mathbf{v}_k, \Delta \mathbf{v}_k)\mathbf{r}_k + g(\mathbf{r}_k, \mathbf{v}_k, \Delta \mathbf{v}_k)(\mathbf{v}_k + \Delta \mathbf{v}_k) \\ \dot{f}(\mathbf{r}_k, \mathbf{v}_k, \Delta \mathbf{v}_k)\mathbf{r}_k + \dot{g}(\mathbf{r}_k, \mathbf{v}_k, \Delta \mathbf{v}_k)(\mathbf{v}_k + \Delta \mathbf{v}_k) \\ m_k \exp(-\frac{||\Delta \mathbf{v}_k||}{g_0 I_{sp}}) \end{bmatrix};
\tag{4.9}
\]
As a consequence, the controls of the problem are the Cartesian components of the impulsive $\Delta v$.

The phase terminal constraints are reported in Equation 4.10, they try to enforce the final state to be equal to the target one.

$$\psi = \begin{bmatrix} r_f - r_M(t_f) \\ v_f - v_M(t_f) \end{bmatrix} = 0; \quad (4.10)$$

Furthermore, the control bounds are formulated as a nonlinear constraint on the amplitude of input $\Delta v$. In [54] the formulation of this constraint as Equation 4.11 can be found.

$$g_k = ||\Delta v_k|| - \frac{T_{max}}{m_k} \leq 0; \quad (4.11)$$

This constraint is based on the observation that using $m_k$ for the whole impulse, instead of decreasing it during the thrust time as would happen in real life, makes it possible to maintain a conservative bound that retains feasibility even if it leads to a more conservative solution. The initial guess of controls and multipliers is set to 0.

Moreover, some modifications have been introduced to the example proposed in [53] to improve convergence. To this aim, the problem is scaled: distances are scaled with $R = 1$ ua, velocities are scaled with the velocity of circular orbit of radius $R$ ($V = \sqrt{\mu/R}$), this implies a time scaling constant such that $2\pi$ corresponds to a full revolution. Ultimately, the mass is scaled with its starting value $M = 1000$ kg.

Also the cost function to be minimized is changed slightly, instead of minimizing $J = -m(t_f)$ (i.e. maximum final mass), the choice was to minimize $J = \sum_{k=0}^{N} ||u_k||^2$ (i.e. minimum energy). In fact this particular choice gives a smoother cost function, therefore easier to be optimized.

### 4.3.1 Results

The obtained final mass is 566.34 kg and the final constraint violation is $f = 9.56957 \times 10^{-6}$.

The final error predicted with the tolerances as specified in Appendix A, is $e_x = 329.79$ km on the position, and $e_v = 2.7736 \times 10^{-4}$ km s$^{-1}$ on the velocity.

The transfer trajectory can be observed in Figure 4.7, and the trajectory follows the typical shape of minimum-energy transfers, with the transfer path crossing the starting orbit. In addition, also the evolution of the mass is represented in Figure 4.10. Finally, for completeness, the values of the multipliers for the phase constraints are reported in Table 4.4.

It is particularly interesting to observe the plot of the control amplitude. In Figure 4.9, the $\Delta v$ controls are represented as impulses. These impulses are representative of continuous thrust arcs, and therefore the thrust magnitude is instead depicted as piecewise constant. The constraint on thrust is the maximum value $T_{max} = 0.5$ N and it is constant. On the contrary the constraint on the impulsive $\Delta v$ varies with the mass as described by Equation 4.11. The controls are compliant with the constraint in magnitude in the two formulations, and the controls
4.3. Earth-Mars Transfer

Figure 4.7: Computed optimal transfer path between Earth and Mars. 3D representation on the left, projection in the x-y plane on the right.

Table 4.4: Lagrange multipliers’ values for the Earth-Mars Transfer Problem.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Lagrange Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[0.0103995 -0.0289793 -0.00281961 0.000447461 0.0473245 0.00229082]</td>
</tr>
</tbody>
</table>

laying on the constraints are the same in both cases.

To conclude, a depiction of the directions of such impulses during the trajectory is reported in Figure 4.8.

As far as the behavior of the solution to higher orders is concerned, this work is not free of limitations and future developments. In fact, for this particular problem, even order 4 resulted to be too high to achieve convergence, probably due to the large nonlinearities that cause divergence of the higher order feedback terms.

The results obtained are reported in Table 4.5. Note that for \( n = 3 \) the violation of the phase constraints improves: the error becomes \( e_x = 44.935 \text{ km} \) and \( e_v = 1.2096 \times 10^{-5} \text{ km s}^{-1} \). On the other hand, far from the optimal trajectory, the constraint violation is greater for higher orders, as can be seen in Figure 4.12 where the plots of Figure 4.11 are superimposed. However, the norm violation is quickly reduced and the violation of the constraints for order 3 remains lower throughout the remainder of the iterations (exception made for the iterate at which order 2 converges). Nonetheless, the higher order controller struggles to converge (as observed also in the previous example) and the number of iterations is significantly increased.
Figure 4.8: Computed optimal transfer path between Earth and Mars with control directions. 3D representation on the left, projection in the x-y plane on the right.

Table 4.5: High order effects on the Earth-Mars Transfer Problem.

<table>
<thead>
<tr>
<th>Order</th>
<th>Outcome</th>
<th>Iterations Accepted</th>
<th>Iterations Total</th>
<th>Constraint Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Success</td>
<td>172</td>
<td>271</td>
<td>9.56957 · 10^{-6}</td>
</tr>
<tr>
<td>3</td>
<td>Success</td>
<td>538</td>
<td>930</td>
<td>5.05122 · 10^{-7}</td>
</tr>
<tr>
<td>4</td>
<td>Aborted</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4.3. Earth-Mars Transfer

Figure 4.9: Computed optimal controls for the Earth-Mars Transfer Problem.

Figure 4.10: Evolution of mass for Earth-Mars transfer.
Chapter 4. Applications

Figure 4.11: Norm of constraint violation $f$ for the Earth-Mars Transfer Problem for orders 2(a), 3(b).

Figure 4.12: Norm of constraint violation $f$ for the Earth-Mars Transfer Problem compared for different orders.
4.4 Satellite Constellation Refueling

The last application of modified HDDP, is to the case of large satellite constellations. The selected constellation for this study is the IRIDIUM-NEXT constellation. In particular, IRIDIUM 131 is considered for the data relative to the orbit. The classical approaches to orbit refueling for satellite constellations are mainly two. The first one consists in having a tank vehicle, servicing the satellites of the constellation. Conversely, the second strategy is a peer-to-peer refueling strategy, and this is the case taken as a reference. A comparison between the two different strategies is given by Tsiotras and De Nailly in [105].

The IRIDIUM NEXT constellation is composed of a total of 80 satellites (comprising active and spare), expected to be in orbit by the end of 2018. IRIDIUM-NEXT 131 is part of the latest 10 satellites launched into orbit on the 12th of December 2017 on board SpaceX’s Falcon IX rocket.

The source of Keplerian elements for this satellite was the Air Force Space Command. In particular the elements extracted for this satellites make it possible to compute the initial condition. The satellites of the constellation are distributed on 6 orbital planes, each with 11 active satellites, as represented in Figure 4.13. The case studied is the refueling of two adjacent satellites.

The initial guess for the algorithm was created using MATLAB®.

The time of flight guess was provided by using a Lambert arc approach. Indeed, a sampling of the orbit in 100 points was made. Then, for each of these sampled positions, a Lambert arc was created, with a TOF in a range between 0 and the orbital period. The final target position could be evaluated propagating in time the position of the target satellite for a time equal to the TOF considered plus the time of departure.

A Porkchop graph was produced as in Figure 4.14, where only one period in the departure time range is considered. In particular, to estimate the minimum $\Delta v$ budget necessary to reach one satellite, a maneuver of two impulses was studied and their total contribute is reported.

The initial guess for the time of flight obtained is $TOF = 1.516$ h. This initial guess was introduced in the modified HDDP algorithm. The problem was scaled with the same scheme as the Earth-Mars Transfer case. The number of phases selected for the problem is $M = 1$ and the number of stages is $N = 100$. The
Chapter 4. Applications

Figure 4.13: IRIDIUM satellites on orbital plane number 2 of the constellation.
dynamics that need to be integrated are reported in Equation 4.12

\[
\begin{align*}
\dot{x} &= \dot{x}; \\
\dot{y} &= \dot{y}; \\
\dot{z} &= \dot{z}; \\
\ddot{x} &= -\frac{\mu}{r^3}x + \frac{T_x}{m}; \\
\ddot{y} &= -\frac{\mu}{r^3}y + \frac{T_y}{m}; \\
\ddot{z} &= -\frac{\mu}{r^3}z + \frac{T_z}{m}; \\
\dot{m} &= -\frac{T}{I_{sp}g_0};
\end{align*}
\] (4.12)

The integration scheme used is a 7/8 Dormand-Prince (8th order solution for propagation, 7th order solution for step size control) Runge-Kutta scheme and the initial conditions to this problem, as obtained from the preliminary Porkchop graph study, are reported in Equation 4.13 to Equation 4.15.

\[
\begin{align*}
\mathbf{r}_0 &= \begin{bmatrix} 1085.028 & -241.357 & -7071.888 \end{bmatrix} \text{ km}; \quad (4.13) \\
\mathbf{v}_0 &= \begin{bmatrix} -5.922 & 4.407 & -1.0743 \end{bmatrix} \text{ km s}^{-1}; \quad (4.14) \\
\mathbf{x}_0 &= \begin{bmatrix} \mathbf{r}_0 \\ \mathbf{v}_0 \end{bmatrix}; \quad (4.15)
\end{align*}
\]
The target position of the satellite is reported in Equation 4.16 to Equation 4.18.

\[ \mathbf{r}_t = \begin{bmatrix} 1444.3563 \\ -509.9122 \\ -6992.1188 \end{bmatrix} \text{ km}; \quad (4.16) \]

\[ \mathbf{v}_t = \begin{bmatrix} -5.8468 \\ 4.3874 \\ -1.5241 \end{bmatrix} \text{ km s}^{-1}; \quad (4.17) \]

\[ \mathbf{x}_t = \begin{bmatrix} \mathbf{r}_t \\ \mathbf{v}_t \end{bmatrix}; \quad (4.18) \]

The cost function is again an augmented Lagrangian one, which minimizes the energy as formulated in Equation 4.19.

\[ J = \sum_{k=0}^{N} (||\mathbf{u}_k||^2) + \lambda^T (\mathbf{x}(t_f) - \mathbf{x}_t) + \sigma_0 ||\mathbf{x}(t_f) - \mathbf{x}_t||^2; \quad (4.19) \]

The phase constraint function is the violation of the target \( \psi = \mathbf{x}(t_f) - \mathbf{x}_t \), while there is no final phase cost. Additionally, the stage cost from Equation 4.19 is \( L_k = ||\mathbf{u}_k||^2 \).

The maximum thrust available for the system is \( T_{\text{max}} = 310 \text{ N} \) to guarantee sufficient thrust to achieve the target in the selected TOF.

Once again the constraint to be imposed on stages is a nonlinear one, it requires that the amplitude of the thrust is smaller than the maximum allowed value.

\[ ||\mathbf{u}_k|| < T_{\text{max}}; \quad (4.20) \]

After optimizing this first case, a second case is developed to show the flexibility of this algorithm. This case considers the presence of the \( J_2 \) perturbation effect in the dynamic model, which, as a consequence, is reformulated as Equation 4.21.

\[
\begin{align*}
\dot{x} &= \dot{x}; \\
\dot{y} &= \dot{y}; \\
\dot{z} &= \dot{z}; \\
\ddot{x} &= -\frac{\mu}{r^3} x + \frac{T_x}{m} - \frac{3J_2\mu R_2^2}{2r^5}(1 - 5\frac{z^2}{r^2})x; \\
\ddot{y} &= -\frac{\mu}{r^3} y + \frac{T_y}{m} - \frac{3J_2\mu R_2^2}{2r^5}(1 - 5\frac{z^2}{r^2})y; \\
\ddot{z} &= -\frac{\mu}{r^3} z + \frac{T_z}{m} - \frac{3J_2\mu R_2^2}{2r^5}(3 - 5\frac{z^2}{r^2})z; \\
\dot{m} &= -\frac{T}{I_{\text{sp}, g0}} .
\end{align*}
\]

### 4.4.1 Results

After optimizing this first case, a second case is developed to show the flexibility of this algorithm. This case considers the presence of the \( J_2 \) perturbation effect in the dynamic model, which, as a consequence, is reformulated as Equation 4.21.
Figure 4.15: Initial relative position between Target satellite and Tank satellite.

Table 4.6: Lagrange multipliers’ values for the Satellite Constellation Refueling Problem.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Lagrange Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[-0.860903 0.509705 1.52568 1.47022 -1.03679 -0.469851]</td>
</tr>
</tbody>
</table>
Chapter 4. Applications

Figure 4.16: Optimal transfer path for the Satellite Constellation Refueling Problem.

Figure 4.17: Optimal control evolution for the Satellite Constellation Refueling Problem.
4.4. Satellite Constellation Refueling

![Graph showing mass evolution during optimal transfer for the Satellite Constellation Refueling Problem.](image)

**Figure 4.18:** Mass evolution during optimal transfer for the Satellite Constellation Refueling Problem.

**Table 4.7:** High order effects on the Satellite Constellation Refueling Problem.

<table>
<thead>
<tr>
<th>Order</th>
<th>Outcome</th>
<th>Iterations Accepted</th>
<th>Iterations Total</th>
<th>Constraint Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Success</td>
<td>197</td>
<td>338</td>
<td>7.50309 \cdot 10^{-6}</td>
</tr>
<tr>
<td>3</td>
<td>Success</td>
<td>173</td>
<td>293</td>
<td>4.585 \cdot 10^{-6}</td>
</tr>
<tr>
<td>4</td>
<td>Success</td>
<td>199</td>
<td>333</td>
<td>4.80822 \cdot 10^{-6}</td>
</tr>
<tr>
<td>5</td>
<td>Aborted</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Following up, the effect of an increasing order is studied, the results are again consistent with the ones found in the previous examples, and they are reported in Table 4.7. The behavior of the norm of constraint violation with the number of iteration can be observed in Figure 4.19.
Figure 4.19: Norm of constraint violation $f$ for the Satellite Constellation Refueling Problem for orders 2(a), 3(b) and 4(c).
After the solution has been retrieved, the perturbation \( J_2 \) is included in the dynamics, and the study is repeated. The solution retrieved is extremely close to the unperturbed one, as the time of flight (\( \sim 2 \) h) is not sufficiently large for the effects of the perturbation to appear on the final solution. The only modification necessary to run this example with respect to the previous one is the change in dynamics for the ODE propagation, as DA will automatically compute the higher order derivatives. The final constraint violation reached is \( f = 4.75122 \cdot 10^{-6} \), and the final mass obtained is \( m_f = 976.5 \) kg. The main differences can be observed in the thrust magnitude in Figure 4.20. Moreover, a discrepancy can be observed in the different Lagrange parameters reached at convergence, by comparing Table 4.6 and the newly obtained Table 4.8.

To replicate the previous results, also a study of the high order effect on convergence is herein reported. The results of constraint violation with the number of iterations is illustrated in Figure 4.21 and it is similar to the unperturbed case. For the sake of completeness, also the number of iterations is reported in Table 4.9.

![Figure 4.20: Difference in control for perturbed and unperturbed cases of the Satellite Constellation Refueling Problem.](image-url)
Chapter 4. Applications

Table 4.8: Lagrange multipliers’s values for the Satellite Constellation Refueling Problem perturbed with $J_2$.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Lagrange Multipliers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>[-0.938731 0.554697 1.6653 1.61168 -1.12907 -0.512154]</td>
</tr>
</tbody>
</table>

Table 4.9: High order effects on Satellite Constellation Refueling Problem with additional $J_2$ perturbation.

<table>
<thead>
<tr>
<th>Order</th>
<th>Outcome</th>
<th>Iterations Accepted</th>
<th>Iterations Total</th>
<th>Constraint Violation</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Success</td>
<td>184</td>
<td>311</td>
<td>$4.75122 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>3</td>
<td>Success</td>
<td>205</td>
<td>351</td>
<td>$3.29792 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>4</td>
<td>Success</td>
<td>212</td>
<td>361</td>
<td>$2.54479 \cdot 10^{-6}$</td>
</tr>
<tr>
<td>5</td>
<td>Aborted</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.21: Norm of constraint violation $f$ for orders 2(a), 3(b) and 4(c) when the disturbance of $J_2$ is introduced in the Satellite Constellation Refueling Problem.
Conclusions

This thesis investigated the problem of Low-Thrust optimal control through the combination of Differential Algebra and Differential Dynamic Programming. The use of DA in HDDP grants the possibility of computing higher order feedback and approximations in the neighborhood of the reference trajectory. The hypothesis formulated at the beginning of this study is that high order feedback and expansions could enlarge the convergence region of the algorithm, leading to a faster convergence with respect to standard second order methods. Nonetheless, this hypothesis seems to be confuted in chapter 4. Through the examples, a clear improvement in the number of iterations for higher orders is not verified. In addition, the higher order feedback terms seem to worsen numerical errors of the algorithm. In particular, the constraint violation is smoother for lower orders and becomes noisy for higher orders. Moreover, even though in some cases the number of iterations necessary for convergence of the algorithm does not change substantially or even reduces, the computational time increases (relative results are reported in Appendix C).

Having addressed this issue, the thesis improves however the HDDP algorithm by relieving some of the workload of the user. The difficulty in implementing the model for perturbed dynamics arises when the necessary partial derivatives of the dynamic function $f$ need to be computed. In particular, the problems are stressed by Pellegrini in his Ph.D. thesis [90]. For complicated dynamical models, the partials required for the propagation of the dynamics must be computed with a symbolic manipulator software and sometimes they are not even possible to obtain. Moreover, these partials need to be converted into useful code (i.e. C++,Fortran etc.). Usually, these symbolic manipulators yield inefficient code, largely due to insufficient factoring. The use of DA to re-
duce the user’s effort is demonstrated in section 4.4, and completely removes these difficulties. Indeed, once a problem is set-up, the user is only asked to code a system of ODEs, reducing the workload on the user to interface this software with a symbolic manipulator (e.g. MAPLE®) as DA will automatically take care of computing the higher order partials. An example is given in this study by introducing the $J_2$ dynamical perturbation, which is handled by the same algorithm as the unperturbed case.

As any preliminary study, this work may suffer from a number of limitations. First of all, as assessed also during section 3.2.4, the algorithm still relies on an Hessian shifting technique to guarantee a convex objective function. This technique is based on a quadratic trust region procedure. Secondly, the algorithm does not improve the standard limitations of HDDP as far as tuning is concerned. In fact, a lot of tuning parameters must be set accurately, in order for this solver to converge to a solution. Finally, the scaling is performed via a non automatic procedure, which slows down solution considerably.

On the other side of the coin, the thesis provides several contributions. First, it provides a first attempt to apply Differential Algebraic techniques in the Differential Dynamic Programming. By doing this, it creates the first building block for further research in this direction. In addition, it adds to the current Differential Dynamic Programming literature by exploiting high order nonlinear feedback controls, capable of dealing with constraints. And lastly, this dissertation improves the user’s experience with HDDP software by removing the tedious step of obtaining partials from symbolic external software and interfacing it with Fortan/C code.

Finally, taking into consideration the above-mentioned limitations and contributions, future researches might obtain enhanced results, taking this thesis as a starting point. For example, an alternative to the Hessian shifting problem could be formulated exploiting high-orders. Indeed, the cost function is not quadratic, therefore, an optimal expansion point where the Hessian is positive definite could be located. Afterwards, the trajectory could be re-expanded about this new reference condition, so that a descent direction is guaranteed.

Furthermore, the high-order terms could be better exploited by using the convergence radius of the cost function to estimate the elliptical region of convergence automatically. This improvement could, at the same time, remove several tuning parameters from the algorithm (namely, $D$ scaling matrix).
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Basic formulation of the Multiphase Problem, adapted from [2].</td>
<td>14</td>
</tr>
<tr>
<td>2.2</td>
<td>Backward Sweep scheme adapted from [86].</td>
<td>18</td>
</tr>
<tr>
<td>2.3</td>
<td>Forward Sweep scheme adapted from [86].</td>
<td>19</td>
</tr>
<tr>
<td>2.4</td>
<td>Expansion between phases for HDDP, adapted from [2].</td>
<td>23</td>
</tr>
<tr>
<td>2.5</td>
<td>The division of domain in sub regions. Points starting in ( G ) follow the blue arrows, points in ( L ) follow the red arrow, whereas points in ( N ) are repicked in the acceptable interval according to the yellow arrow (adapted from [95]).</td>
<td>30</td>
</tr>
<tr>
<td>2.6</td>
<td>Backward sweep on the stages of a phase. The lines about the reference state represent the trust region of the quadratic approximation (adapted from [2]).</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Simple transition diagram for real and floating point numbers, adapted from [78].</td>
<td>37</td>
</tr>
<tr>
<td>3.2</td>
<td>Simple transition diagrams for functions and their equivalent class, adapted from [78].</td>
<td>38</td>
</tr>
<tr>
<td>4.1</td>
<td>Evolution of states for the Linear Quadratic Problem.</td>
<td>51</td>
</tr>
<tr>
<td>4.2</td>
<td>Evolution of controls for the Linear Quadratic Problem.</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>Evolution of adimensional states for the Mono-Dimensional Landing Problem.</td>
<td>53</td>
</tr>
<tr>
<td>4.4</td>
<td>Evolution of adimensional control for the Mono-Dimensional Landing Problem.</td>
<td>53</td>
</tr>
<tr>
<td>4.5</td>
<td>Evolution of the normalized mass for the Mono-Dimensional Landing Problem.</td>
<td>54</td>
</tr>
<tr>
<td>4.6</td>
<td>Norm of constraint violation ( f ) for the Mono-Dimensional Landing Problem for orders 2(a), 3(b), 4(c) and 6(d).</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>Computed optimal transfer path between Earth and Mars. 3D representation on the left, projection in the x-y plane on the right.</td>
<td>58</td>
</tr>
</tbody>
</table>
4.8 Computed optimal transfer path between Earth and Mars with control directions. 3D representation on the left, projection in the x-y plane on the right. ...................................................... 59
4.9 Computed optimal controls for the Earth-Mars Transfer Problem. . 60
4.10 Evolution of mass for Earth-Mars transfer. ............................... 60
4.11 Norm of constraint violation $f$ for the Earth-Mars Transfer Problem for orders 2(a), 3(b). ........................................... 61
4.12 Norm of constraint violation $f$ for the Earth-Mars Transfer Problem compared for different orders. .................................. 61
4.13 IRIDIUM satellites on orbital plane number 2 of the constellation. 63
4.14 Computed optimal impulsive $\Delta v$ budget for the Satellite Constellation Refueling Problem. ....................................... 64
4.15 Initial relative position between Target satellite and Tank satellite. 66
4.16 Optimal transfer path for the Satellite Constellation Refueling Problem. ................................................................. 67
4.17 Optimal control evolution for the Satellite Constellation Refueling Problem. ............................................................... 67
4.18 Mass evolution during optimal transfer for the Satellite Constellation Refueling Problem. .................................................. 68
4.19 Norm of constraint violation $f$ for the Satellite Constellation Refueling Problem for orders 2(a), 3(b) and 4(c). ..................... 69
4.20 Difference in control for perturbed and unperturbed cases of the Satellite Constellation Refueling Problem. ......................... 70
4.21 Norm of constraint violation $f$ for orders 2(a), 3(b) and 4(c) when the disturbance of $J_2$ is introduced in the Satellite Constellation Refueling Problem. ........................................... 71
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Lagrange multipliers’ values for the Liner Quadratic Problem.</td>
<td>50</td>
</tr>
<tr>
<td>4.2</td>
<td>Constant controls values for the Liner Quadratic Problem.</td>
<td>51</td>
</tr>
<tr>
<td>4.3</td>
<td>High order effects on the Mono-Dimensional Landing Problem.</td>
<td>54</td>
</tr>
<tr>
<td>4.4</td>
<td>Lagrange multipliers’ values for the Earth-Mars Transfer Problem.</td>
<td>58</td>
</tr>
<tr>
<td>4.5</td>
<td>High order effects on the Earth-Mars Transfer Problem.</td>
<td>59</td>
</tr>
<tr>
<td>4.6</td>
<td>Lagrange multipliers’ values for the Satellite Constellation Refueling Problem.</td>
<td>66</td>
</tr>
<tr>
<td>4.7</td>
<td>High order effects on the Satellite Constellation Refueling Problem.</td>
<td>68</td>
</tr>
<tr>
<td>4.8</td>
<td>Lagrange multipliers’s values for the Satellite Constellation Refueling Problem perturbed with $J_2$.</td>
<td>71</td>
</tr>
<tr>
<td>4.9</td>
<td>High order effects on Satellite Constellation Refueling Problem with additional $J_2$ perturbation.</td>
<td>71</td>
</tr>
<tr>
<td>A.1</td>
<td>Tuning Parameters for HDDP with their description.</td>
<td>90</td>
</tr>
<tr>
<td>A.2</td>
<td>Tuning Parameters for the Mono-Dimensional Landing Problem.</td>
<td>90</td>
</tr>
<tr>
<td>A.3</td>
<td>Tuning Parameters for the Earth-Mars Transfer Problem.</td>
<td>91</td>
</tr>
<tr>
<td>A.4</td>
<td>Tuning Parameters for the Satellite Constellation Refueling Problem.</td>
<td>92</td>
</tr>
<tr>
<td>C.1</td>
<td>Computational times for various orders for the different test cases.</td>
<td>95</td>
</tr>
</tbody>
</table>
Bibliography


80 Bibliography


Appendix A

Tuning Parameters

One of the difficulties of the HDDP algorithm, is the necessity to tune many parameters to achieve convergence. The most important parameters are described in the following table and their values for the examples are reported in their respective sections.
Tuning parameters for the Mono-dimensional Landing Problem are not exploited as the algorithm converges in just one iteration, therefore it is not deemed useful to report them.
Table A.1: Tuning Parameters for HDDP with their description.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon_{opt}$</td>
<td>Expresses the value of the minimum Expected Reduction for which the algorithm converges.</td>
</tr>
<tr>
<td>$\epsilon_{feas}$</td>
<td>Expresses the value of tolerance admitted for phase constraint violation.</td>
</tr>
<tr>
<td>$\Delta_0$</td>
<td>Sets the starting value of Trust Region radius.</td>
</tr>
<tr>
<td>$\sigma_0$</td>
<td>Initial cost violation penalty in the Augmented Lagrangian function.</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>Coefficient used to adjust the Trust Region radius, if the iterate is accepted, $\Delta_{p+1} = (1 + \kappa)\Delta_p$ otherwise $\Delta_{p+1} = (1 - \kappa)\Delta_p$.</td>
</tr>
<tr>
<td>$\epsilon_1$</td>
<td>Expresses the maximum deviation from 1 of the ratio between Expected Reduction and the computed Reduction.</td>
</tr>
<tr>
<td>$\kappa_\sigma$</td>
<td>Parameters used to increase the penalty cost $\sigma$ if the constraint is violated. In particular: $\sigma_{p+1} = \max(\min(0.5\frac{\kappa_\sigma}{\kappa_\sigma}, \sigma_p), \sigma_p)$, where $f$ is an estimate of the norm of constraint violation and $h$ is the cost estimate(without penalty term $f$). The value is safeguarded to keep the balance between feasibility and optimality of the solution.</td>
</tr>
<tr>
<td>$\kappa_{easy,hard}$</td>
<td>Parameters used to set convergence of the TRQP problem.</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>Parameter used to choose a value of $\lambda$ inside the boundary values when the Newton iteration step leads to an unfeasible point.</td>
</tr>
</tbody>
</table>

A.1 Mono-Dimensional Landing

many of the proposed parameters have been retrieved directly from [55] and [90].

Table A.2: Tuning Parameters for the Mono-Dimensional Landing Problem.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon_{opt}$</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>$\epsilon_{feas}$</td>
<td>$10^{-4}$</td>
</tr>
<tr>
<td>$\Delta_0$</td>
<td>0.01</td>
</tr>
<tr>
<td>$\sigma_0$</td>
<td>0.001</td>
</tr>
<tr>
<td>$\kappa$</td>
<td>0.25</td>
</tr>
<tr>
<td>$\epsilon_1$</td>
<td>0.01</td>
</tr>
<tr>
<td>$\kappa_\sigma$</td>
<td>1.1</td>
</tr>
<tr>
<td>$\kappa_{easy,hard}$</td>
<td>0.1, 0.2</td>
</tr>
<tr>
<td>$\theta_1$</td>
<td>0.01</td>
</tr>
</tbody>
</table>

A particular note goes to the scaling matrix $D$ of the Trust Region Quadratic Problem. The values were not clearly reported in the reference literature for this
example, therefore its diagonal elements were used as tuning parameters for the whole algorithm. The matrix exploited are hereby reported:

\[
D_u = \begin{bmatrix} 2.0 \\ 0.01 & 0.0 \\ 0.0 & 0.02 \end{bmatrix}
\]  
(A.1)

\[
D_\lambda = \begin{bmatrix} 0.001 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.001 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.001 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.001 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.001 \end{bmatrix}
\]  
(A.2)

The values of the diagonal entries in \(D_u\) are dictated by the maximum value of the Hessian of cost with respect to input (which was approximately 2 during the simulations). The same reasoning was then applied to the Hessian with respect to the Lagrange multipliers and they were adjusted until convergence was achieved.

### A.2 Earth-Mars Transfer

Table A.3: Tuning Parameters for the Earth-Mars Transfer Problem.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\epsilon_{opt})</td>
<td>(10^{-8})</td>
</tr>
<tr>
<td>(\epsilon_{feas})</td>
<td>(10^{-5})</td>
</tr>
<tr>
<td>(\Delta_0)</td>
<td>0.01</td>
</tr>
<tr>
<td>(\sigma_0)</td>
<td>0.001</td>
</tr>
<tr>
<td>(\kappa)</td>
<td>0.25</td>
</tr>
<tr>
<td>(\epsilon_1)</td>
<td>0.01</td>
</tr>
<tr>
<td>(\kappa_\sigma)</td>
<td>1.1</td>
</tr>
<tr>
<td>(\kappa_{easy,ehard})</td>
<td>0.1, 0.2</td>
</tr>
<tr>
<td>(\theta_1)</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[
D_u = \begin{bmatrix} 100.0 & 0.0 & 0.0 \\ 0.0 & 100.0 & 0.0 \\ 0.0 & 0.0 & 100.0 \end{bmatrix}
\]  
(A.3)

\[
D_\lambda = \begin{bmatrix} 0.001 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.001 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.001 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.001 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.001 \end{bmatrix}
\]  
(A.4)

Once again, the \(D_\lambda\) matrix was tuned to achieve convergence, while the terms in \(D_u\) were selected so that the \(\delta u\) is penalized, thus helping to achieve compliance of the input constraints. In fact, having large diagonal terms in \(D_u\) reduces the \(\delta u\) output of the Trust Region Quadratic Problem. Without loss of generality,
if there is only 1 dimension, \( \delta u < \frac{\Delta}{D_u} \), therefore the maximum \( \delta u \) is smaller for higher \( D_u \) and vice versa.

### A.3 Satellite Constellation Refueling

**Table A.4:** Tuning Parameters for the Satellite Constellation Refueling Problem.

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \epsilon_{\text{opt}} )</td>
<td>( 10^{-8} )</td>
</tr>
<tr>
<td>( \epsilon_{\text{feas}} )</td>
<td>( 10^{-4} )</td>
</tr>
<tr>
<td>( \Delta_0 )</td>
<td>0.01</td>
</tr>
<tr>
<td>( \sigma_0 )</td>
<td>0.001</td>
</tr>
<tr>
<td>( \kappa )</td>
<td>0.25</td>
</tr>
<tr>
<td>( \epsilon_1 )</td>
<td>0.01</td>
</tr>
<tr>
<td>( \kappa_0 )</td>
<td>0.25</td>
</tr>
<tr>
<td>( \kappa_{\text{easy}, \text{hard}} )</td>
<td>0.1, 0.2</td>
</tr>
<tr>
<td>( \theta_1 )</td>
<td>0.01</td>
</tr>
</tbody>
</table>

\[
D_u = \begin{bmatrix}
1000.0 & 0.0 & 0.0 \\
0.0 & 1000.0 & 0.0 \\
0.0 & 0.0 & 1000.0
\end{bmatrix}
\quad \text{(A.5)}
\]

\[
D_\lambda = \begin{bmatrix}
0.01 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.01 & 0.0 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.01 & 0.0 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.01 & 0.0 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.01 & 0.0 \\
0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.01
\end{bmatrix}
\quad \text{(A.6)}
\]
Appendix B

Differential Algebra Example

This example serves the purpose of showing how DA techniques can be applied to obtain automatic information about the derivative of a function, while computing the function value. The following expressions are useful for the development of the solution and are here reported:

\[(q_0, q_1)(x, y) = (1, 0) \rightarrow x = \frac{1}{q_0}, y = -\frac{q_1}{q_0} \rightarrow (q_0, q_1)^{-1} = \left(\frac{1}{q_0}, -\frac{q_1}{q_0}\right)\];

\([f]^{-1} = (f, f')^{-1} = \left(\frac{1}{f}, -\frac{f'}{f^2}\right) = \left[\frac{1}{f}\right]\) \hspace{1cm} (B.1)

Keeping this in mind, we assume \(f = x\), therefore \([f] = (x, 1)\).

\[h = \frac{3 + 7x - \frac{2}{x^2}}{\frac{1}{2+x} - 4} \hspace{1cm} (B.2)\]

The unknown is the derivative of \(h\), therefore \([h]\) is sought.

- \([x][x] = (x, 1)(x, 1) = (x^2, 2x) = [x^2];\)
- \(7[x] + 2[x^2]^{-1} = (7x, 7) + 2\left(\frac{1}{x^2}, -\frac{2x}{x^4}\right) = (7x + \frac{2}{x^2}, 7 - \frac{4x}{x^3});\)
- \(3 + [7x + 2/x^2] = (3, 0) + (\frac{2}{x^2}, 7 - \frac{4x}{x^3}) = (3 + 7x + \frac{2}{x^2}, 7 - \frac{4x}{x^3});\)
- \(2 + [x] = (2, 0) + (x, 1) = (2 + x, 1);\)
- \([2 + x]^{-1} = \left(\frac{1}{2+x}, -\frac{1}{(2+x)^2}\right);\)
- \([\frac{1}{2+x}] - 4 = \left(\frac{1}{2+x}, -\frac{1}{(2+x)^2}\right) - (4, 0) = \left(\frac{1}{2+x} - 4, -\frac{1}{(2+x)^2}\right);\)
- \([\frac{1}{2+x} - 4]^{-1} = \left(\frac{1}{2+x} - 4, -\frac{1}{(2+x)^2}\right);\)
\[ \left[ \frac{1}{2 + x} - 4 \right]^{-1} [3 + 7x + 2/x^2] = \left( \frac{3 + 7x + 2}{2 + x - 4}, \frac{7 - 4x}{2 + x - 4} - \frac{1}{(2 + x - 4)^2} (3 + 7x + 2/x^2) \right) \]

The result at the end of this procedure is obtained via simple application of the rule described in subsection 3.1.1. The result is nonetheless the same as \([h] = (h, h')\).
Appendix C

Computational Times

Computational times for the various examples are reported in Table C.1.

Table C.1: Computational times for various orders for the different test cases.

<table>
<thead>
<tr>
<th></th>
<th>Computational Time [s]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Order 2</td>
</tr>
<tr>
<td>Mono-Dimensional Landing</td>
<td>3.7</td>
</tr>
<tr>
<td>Earth-Mars Transfer</td>
<td>15.8</td>
</tr>
<tr>
<td>Satellite Constellation Refueling</td>
<td>75.0</td>
</tr>
<tr>
<td>Perturbed Satellite Constellation Refueling</td>
<td>114.9</td>
</tr>
</tbody>
</table>