TECHNISCHE UNIVERSITÄT
ILMENAU

Pontificia Universidad Católica del Perú

Escuela de Posgrado

Permutation-based enhancement of factorization methods for the fast solution of KKT systems

Tesis para obtener el grado académico de Maestro en Ingeniería de Control y Automatización que presenta:

John Victor Leon Meza

Asesor PUCP (PUCP): *Dr. Carlos Gustavo Pérez-Zuñiga*

Co-Asesor de la Universidad no PUCP: *Dr. habil. Pu Li*

Lima, 2025

Informe de Similitud

Yo, **Carlos Gustavo Pérez-Zuñiga**, docente de la Escuela de Posgrado de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado **Permutation-based enhancement of factorization methods for the fast solution of KKT systems**, del autor:

John Victor Leon Meza

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de ….7..%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 29/01/2025
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:
…… Lima, 29/01/2025 ……..

Apellidos y nombres del asesor / de la asesora: **Pérez-Zuñiga, Carlos Gustavo**	
DNI: 41864666	Firma: 
ORCID: 0000-0001-5946-1395	

Acknowledgments

I express my sincere gratitude to all those who have supported me throughout the process of writing this thesis.

First of all, I would like to express my gratitude to my professors, whose expertise and dedication have been invaluable throughout my academic journey and the writing of this thesis. In particular, I am deeply grateful to M.Sc. Bernd Juris for his continuous guidance, encouragement, and invaluable feedback throughout my research. Their expertise and insights were instrumental in shaping the direction of my work and ensuring its success.

I am deeply grateful to my family and friends for their unwavering support and understanding during this challenging yet rewarding journey. Special thanks to Maria Esther Meza Olivera, my mother, for her constant encouragement and emotional support. Although you are not here to see it, your presence helped me through my studies and allowed me to reach the end of this work.

Resumen

Esta tesis explora las posibles mejoras en el tiempo de procesamiento para la resolución de problemas de control óptimo mediante el uso de las condiciones de Karush-Kuhn-Tucker (KKT) que dan lugar a una serie de ecuaciones lineales. El objetivo principal de este trabajo era desarrollar y probar métodos que redujeran el esfuerzo computacional y el tiempo necesario para resolver estos sistemas, lo cual es crítico en aplicaciones en tiempo real como el Control Predictivo de Modelos (MPC). Se propusieron dos enfoques para abordar el problema: el primero se centra en el uso de un algoritmo de Grado Mínimo Aproximado (AMD) calculado fuera de línea para reducir los rellenos al factorizar la matriz de coeficientes, y el segundo aprovecha la información relacionada con cómo cambian las entradas de la matriz a través de la iteración de los solvers para identificar los sectores constantes que no es necesario calcular en cada paso del solver. Ambos enfoques muestran resultados prometedores en la resolución del problema, con el de AMD mostrando una mejora más concreta en comparación con otros métodos similares, mientras que el segundo mostró potencial para sistemas más grandes, aunque son necesarias más optimizaciones en la codificación. En general, la investigación proporciona valiosos conocimientos para mejorar la eficiencia de la resolución de problemas de control óptimo y contribuir a estrategias de control en tiempo real más eficaces.

Abstract

This thesis explores the potential improvements in processing time for solving optimal control problems by using the Karush-Kuhn-Tucker (KKT) conditions which result in a series of linear equations. The primary objective of this work was to develop and test methods that would reduce the computational effort and time needed for solving these systems, which is critical in real-time applications such as Model Predictive Control (MPC). Two approaches were proposed to tackle the problem: the first focuses on the use of an offline calculated Approximate Minimum Degree (AMD) algorithm to reduce the fill-ins when factorizing the coefficient matrix, and the second takes advantage of information related to how the entries of the matrix change through the solvers iteration to identify the constant sectors that do not need to be calculated at each solver step. Both approaches show promising results in solving the problem, with the AMD one showing a more concrete improvement compared to other similar methods, while the second one showed potential for larger systems, though further optimizations on the coding are necessary. In general, the research provides valuable insight into improving the efficiency of solving optimal control problems and contributing to more effective real-time control strategies.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Optimization problems are widely used in many applications to achieve goals that are feasible according to the conditions needed. Its solution involves firstly defining the objective to be achieved while considering a certain model, and the limitations of its states in the form of algebraic constraints. Then a solution method is selected which can include static methods by using a discretization of the problem to obtain a quasi-static optimal control problem, this leads to the definition of a set of nonlinear equations called Karush-Kuhn-Tucker(KKT) conditions, which structure depends on the method selected. The problem can then be solved numerically using, for example, Newton steps that generate a linear system of the form $\boldsymbol{Ax} = \boldsymbol{b}$ at each iteration, which implies the computational complexity of the solution being $\mathcal{O}(n^3)$ with n being the dimension of the state vector $\boldsymbol{x}$ (Nocedal and Wright, 1999). This can be expected to be the most complex part of the process, and thus many techniques and investigations were made to calculate them faster. This is especially true when a large number of variables are involved in the optimization due to a specific prediction horizon or time sample.

The control of dynamic systems has to take into consideration, as one of the most important aspects, the maximum feasible response time for the type of process to be controlled. Processes with fast changes need fast updates to the control law, thus calculations must be done in a way that optimizes the time needed. Restricting the time horizon or widening the time step used for discretization could be helpful in accelerating the calculations, but it could lead to suboptimal results after the total time of operation. Also, physical conditions can increase the complexity of the solution of these problems, as the presence of nonlinear ones can greatly restrict the methods that can be used.

Applications of this can be found in the field of self-driving vehicles, as advances in real-time object detection made them reach high frame rates depending on the model and hardware installed (Katkade, Bagal, Manza, and Yannawar, 2023), so that those can be avoided to prevent accidents or collisions (Ekatpure, 2023) (Sprodowski, 2021), thus it is of great importance that vehicle control can be done at a speed that allows for the correct function of these systems.

Other industries that use this type of control include the petrochemical with maximization of profit while responding to market variations with minimal capital investment due to the unpredictability of the market (Morari and H. Lee, 1999), or its compensation for dead times and unusual dynamic behaviors (Cutler and Ramaker, 1980). Also in predictors for the control of distillation columns in oil refineries and other similar processes in energy plants (Richalet, Rault, Testud, and Papon, 1978) (Holkar and Waghmare, 2010).

The applications show the need to use as much information as possible in the calculation of solution of optimization problems, which has raised the interest to develop methods that can accelerate the KKT systems computation. This work focuses on two concepts that accomplish these tasks and can be implemented for the control of dynamic systems.

1.2 Objective

This work analyses the effects different approaches have on the computational effort needed to solve KKT-conditions for optimal control problems, with the objective of finding one that potentially improves its calculation time to make them applicable for fast dynamic systems. To achieve this, a multivariable double integrator model is used as the process to be controlled with an interior-point method that can be customized to accept permutations on the KKT-matrix. Also algorithms that generate those reorderings are developed based on two concepts that enhance the calculation time needed for the solution of the KKT systems, with the first concept based on a fill-ins generation reduction when inverting a matrix for the solution of linear equation systems, and the second one including information regarding the rate of change of certain sectors of matrices which can then prefactorized.

1.3 Outline

This work is divided into 5 chapters. The first one is the State of the Art, where a summary of the latest advancements made by other researchers is provided, aiming to offer a clear overview of the current state of the problem. It includes an insight into the state of the optimal control research and the software needed for the development of the thesis.

The second one is the Theoretical Framework which includes the revised theories and key definitions that guided the design of both the testing environment and proposed solutions. It also served as a reference point for evaluating the results of the research. The topics to be discussed are factorization methods, permutation based on an approximate minimum degree, and ground knowledge in system optimization.

The Design chapter presents the problem statement, followed by the development and implementation of the proposed solutions. It includes a discussion on the reasons why the reordering based on minimum degree and a factorization including information about the rate of changes in a matrix are presented as possible solutions.

The Evaluation chapter collects and analyses the results of the methods discussed in the design part. Later, a comparison of concepts between the solutions and the benchmark will be used to have a detailed assessment of their performance. This is complemented with a discussion of the results, where the theoretical framework is revised with practical implementation, and future work is suggested.

Finally, the conclusion of the last chapter presents the final results of the analysis made in the previous chapters with the aim of highlighting the most significant contributions of the research and providing suggestions for future work that could enhance or build upon the findings.

2 State of the Art

The present chapter provides a comprehensive overview of the existing methods and approaches relevant to the topic of the thesis. Discussion in the topic of Optimal Control is done to delve into the methodologies and innovations made in this field to contextualize the work and problematic. In addition, a detailed description of the software currently used is done to identify what can be used in the development of the solution approaches.

2.1 Optimal Control

A control problem aims to achieve the desired behavior of a dynamic system, usually following a trajectory of its states that was pre-calculated and found to have a good performance. When other goals are taken into account, an objective function can be constructed that factors and weights them according to their priority. This, combined with the limitations normally present in processes, constitutes the basis of an Optimal Control Problem (OCP).

An OCP can be solved with many methods; some of them, such as the simplex method (Nocedal and Wright, 1999), exploit the limits of the feasible regions of the variables, as the optimal solutions are usually found there. Others, such as interior point methods (IPM) (Nesterov and Nemirovskii, 1994), use the inner zone of the limitations to achieve faster performance. These have in common that the calculations are done by an extensive use of linear equation systems, which can be computationally expensive when a large number of variables or specially complex coefficient structure are involved. In order to speed up the calculations, interior point methods have been widely researched and analyzed.

In (Cohen, Lee, and Song, 2018) it is stated that the fastest algorithm that solves an arbitrary linear program with n variables and d constraints has a set complexity which depends exponentially on both factors and the number of nonzeros present in the coefficients of the equality constraints. In more generic cases, it is dominated by the cost per iteration and the number of iterations that are represented as $n^{2.5}$. This has motivated researchers to find ways to reduce the number of iterations needed while maintaining cost with success in some specific cases. The authors present a stochastic version of the short-step central path method to address the general case, which maintains the same running time even when using smaller steps due to their complexity and size decreasing proportionally. Another attempt to reduce the overall computation times is made in (Lee and Sidford, 2015), where it is noted that the main matrices involved in solution algorithms do not change much between iterations, which can be exploited to

maintain an approximate inverse of them that would reduce the average cost per iteration. This is done by using low-rank updates to the coefficient matrix at each iteration using a diagonal matrix. Additionally, (Jiang, Kathuria, Lee, Padmanabhan, and Song, 2020) presents a solver for cases where the decision variables are given in the form of semidefinite matrices and thus called semidefinite programs. It can be considered a more generalized version of linear programming and has new on-going application cases. Here, it can be seen that the time to invert the Hessian and slack matrices is a barrier for improvements, something that has to be considered also in the simpler cases. In (Schulze Darup and Book, 2021) an alternative for faster solutions of a model predictive controller is proposed. This type of practical applications has termination conditions that stops the solver when the computational effort needed to continue would not deliver a significantly higher optimality result. In some cases, due to the dynamics of the systems, this condition only allows a small number of iterations to be made. Therefore, in the article, a real-time alternating-direction method of multipliers is proposed, which was proven to achieve an nearly optimal result even when reducing the allowed iterations. Although real-world applications were not discussed, this approach has the potential to achieve accurate and fast controllers.

As mentioned above, the number of nonzero coefficients has a great impact on the performance of optimization solvers. They can be arranged into a matrix for simplicity in the moment of solving OPCs, especially when a high number of equations is considered, such as in the case of KKT conditions. When the majority of its entries are empty, or zero, they can be called sparse matrices, and their structures are exploited to allow faster calculations to be made. In (Srivastava, Jin, Liu, Albonesi, and Zhang, 2020) and (Zhang, Wang, Han, and Dally, 2020) it is mentioned that generalized sparse matrix multiplication (SpGEMM) is a key computational task used in many engineering and scientific applications. This has increased motivation to improve the speed with which it is done. The first work introduces a row-wise multiplication approach, which allows for parallel calculations of the output while requiring less on-chip memory in comparison to the traditional method, as it does not need to load the entire matrix at each calculation, just the rows involved. The second work focuses on the reuse of the input and output data to build an acceleration architecture. This is done by first producing partial matrices in a multiply stage, which are then merged. Other storage types for sparse matrices are discussed in (Shahnaz, Usman, and Chughtai, 2005), (Langr and Tvrdík, 2016) and (Smailbegovic, Gaydadjiev, and Vassiliadis, 2005) where sparsity is taken advantage to allow lower memory requirements and faster access to the data for calculations. Programming environments have also proposed and developed their own sparse storage type. For example, Matlab includes a special datatype that can be used in many of its functions and operations with the benefit of high time savings (Gilbert, Moler, and Schreiber, 1997).

A matrix can also be factorized into a set of triangular matrices that have the structure needed to calculate equation systems faster (Haddad, 2009). Nonsingular square matrices that are also positive definite can be reduced using Cholesky factorization, which is similar to the square root in real numbers $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{L}^T$ with a further reduction to avoid calculating square roots as $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$. When the matrix is not positive definite, the LU factorization can be used $\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$ (Davis, 2006). One problem that arises with using these methods is the possible loss of sparsity and the generation of previously

non-existent nonzero entries. To avoid this, a reordering step can be done, as it can be proven that if the matrix is less dense on its upper left region, the amount of fill-ins is reduced. This is the basis for minimum degree algorithms (Roos, Terlaky, and Vial, 2006).

The nonzero pattern of a sparse symmetric matrix can be represented using graph theory, where each row represents a node and each nonzero value is an edge between nodes, the amount of edges a single node has is defined as its degree. When a node is removed, new edges appear corresponding to the number of new nonzero entries generated. The problem with this approach is that calculating the exact degree at each time is computationally costly; however, algorithms that use upper bound approximations were developed (Amestoy, Davis, and Duff, 1996).

2.2 Software

The main simulation and calculation portion of this work was performed using Matlab version 9.12.0.2039608 (R2022a) Update 5, which includes a number of algorithms that allow faster execution and testing of the methods that will be discussed (Inc., 2022). The number of functions that accept and process sparse matrices makes it a suitable environment to test the approaches proposed in this work. This subsection introduces the methods used and serves as the informatics basis from which the design is done.

Matlab's backslash "\" operator or mldivide is used to solve the system of linear equations $\boldsymbol{Ax}=\boldsymbol{b}$ using the structure and characteristics of the coefficient matrix to minimize the computation time needed by choosing an adequate solver. It allows the input of sparse matrices achieving a significantly lower processing time. It simplifies the selection of a permutation and factorization method to need just one line of code $\boldsymbol{x} = \boldsymbol{A} \setminus \boldsymbol{b}$. The solvers included in this method relate to the structure of the matrix, including least squares, triangular, banded, permuted triangular, LU, LDL, and Cholesky. This approach allows better performance when solving problems with many variables, such as for the accurate prediction and control of a fast dynamic system. Related to the work of this document, more attention goes to the LDL and Cholesky solvers, as they allow faster calculation times for generic sparse matrices and the main condition for its use is symmetry, which can be obtained by introducing minor changes of variables when defining the equation system. These solvers also have the quality of using block methods, which makes them perform faster.

With the Optimization and Model Predictive Control Toolboxes, new functions are included that solve optimization problems using the interior point method. The characteristics of the system to be solved in this work allow for the use of the functions *fmincon* and *mpcInteriorPointSolver*. The first one is used to find the minimum of a multivariable

nonlinear function subjected to constraints and bounds, defined as in equation 2.1

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & c(x) < 0 \\ & ceq(x) = 0 \\ & A.x \leq b \\ & Aeq.x = beq \\ & lb \leq x \leq ub \end{aligned} \tag{2.1}$$

where $\boldsymbol{f(x)}$, $\boldsymbol{c(x)}$ and $\boldsymbol{ceq(x)}$ are user defined functions with the possibility of being nonlinear, $\boldsymbol{b}$, $\boldsymbol{beq}$,$\boldsymbol{lb}$ and $\boldsymbol{ub}$ are vectors for our problem, and the pair $\boldsymbol{A}$ $\boldsymbol{Aeq}$ are matrices. It also has a *options* parameter where the algorithm can be chosen, as well as the maximum number of iterations or function evaluations, to improve the performance of the solver, by default it is set on Interior Point. This solver is recommended to be used when the number of variables to be optimized is low because the way it calculates them tends to take a lot of iterations and thus time.

The other function, *mpcInteriorPointSolver*, is included in the Model Predictive Control Toolbox to be used in the Optimization step. It uses a primal-dual interior-point algorithm with a Mehrotra predictor-corrector for the solution of quadratic programming problems. Because it is meant to be used on MPC applications, it provides good performance even in the presence of large prediction horizons, which means that it can handle more variables than the previous function in less computational time. Similarly to the previous functions, equation 2.2 shows the form the problem needs to be solved in this way

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2}x^T Hx + f^T x \\ \text{s.t.} \quad & A.x \leq b \\ & Aeq.x = beq \end{aligned} \tag{2.2}$$

where $\boldsymbol{f}$, $\boldsymbol{b}$ and $\boldsymbol{beq}$ are vectors, and $\boldsymbol{H}$, $\boldsymbol{A}$ and $\boldsymbol{Aeq}$ are matrices. Another parameter is a *mpcInteriorPointOptions* object that sets the number of iterations or the tolerance in the solution of the problem. For both functions, an initial condition vector $\boldsymbol{x_0}$ is needed to give the solver the information on the number of variables to optimize.

Matlab also includes native permutation functions. These were developed to exploit the special structures produced by the reordering of the elements of the matrices. For the reduction of the bandwidth of a matrix, it includes the reversed Cuthill-McKee, which groups the entries around the diagonal allowing the use of banded solvers when calculating equation systems. This permutation is not very effective with systems similar to the one being analyzed in this work. Other reordering algorithms were made to maintain the sparsity of the matrix even after factorization, including Nested Dissection (ND) and Approximate Minimum Degree (AMD). Both use graph theory to find a configuration where the number of new entries when solving an equation system is minimized. The first divides the complete matrix into block submatrices and then applies the reordering, while the second does it in its entirety. This results in ND being more effective when the system has a structure that can be easily divided. Their syntax in Matlab is *dissect()* and *amd()*, respectively. These functions will be used to compare the performance of the first proposed approach, as they have the same working principle and objective.

3 Model predictive control

Model Predictive Control (MPC) is a powerful and versatile control strategy that has gained significant attention across various industries. This chapter delves into the fundamentals of MPC, exploring its structure, benefits, and practical applications. MPC relies on a model of the system or process to predict future behavior and make control decisions that optimize performance over a specified time horizon. By integrating optimization techniques, MPC can achieve objectives such as minimizing tracking error or reducing energy consumption while handling constraints that ensure safe and efficient operation.

The chapter will also address the formulation of optimal control problems and the role of solvers in executing MPC. Optimal control theory provides the mathematical foundation for MPC, enabling the design of control actions that achieve the best possible outcome under defined constraints and objectives. To solve these control problems, various optimization solvers are employed, each with unique characteristics tailored to different types of systems and constraints. This combination of prediction, optimization, and real-time adaptability makes MPC a robust tool for managing complex processes with high efficiency and precision.

By the end of this chapter, the reader will have a comprehensive understanding of MPC and its application to real-world systems, as well as insights into the optimization techniques that support this advanced control approach.

3.1 MPC Framework

Model predictive controllers use the model of the operating plant or process system to estimate future outputs at determined time instants, enabling the use of optimization solvers to achieve specific objective functions (Camacho and Bordons, 2004). These objectives are often centered on minimization of the error between the estimated output and the desired trajectory, or to enhance the efficiency of energy consumed.

MPC is widely used in the industry because of its simplicity regarding control theory concepts, as it is very intuitive and easy to tune, allowing less experienced staff to implement it effectively. In addition, a wide range of processes can be controlled without greatly increasing their complexity, making them suitable for systems with long delays, inherited instability, or systems with multiple input and output variables. Another advantage of its use is the introduction of the concept of a feedforward control strategy for the compensation of disturbances, which is something traditional controllers such as PID struggle with. Finally, its open methodology allows for extensions, including those regarding complex constraints in the design process.

One of the primary disadvantages of this type of control is the computational load required since a new solution is made at each sampling time. This is especially troublesome when the system requires faster updates due to its innate dynamics or when they depend on a large number of variables, producing a complex optimal problem that increases the minimum computational power required by the controller. Another drawback that comes with the use of these strategies is the necessity of a highly accurate and appropriate process model, as the design algorithm depends greatly on it. Any discrepancies between the real process and model can considerably impact the performance of the control. To avoid this effect, a degree of robustness has to be incorporated into the controller, which can be done by analyzing the more frequent inaccuracy factors present on the system, such as the delay uncertainty in presence of high-frequency responses.

The basic structure of an MPC controller consists on three main components. First, there is the model part, which is generated by using the identified process model to predict the system's behavior at future time instants. This is followed by the error calculation between the desired trajectory and the predicted values, which gives the performance degree the current set of control variables has. Finally, an optimizer uses this error in conjunction with the objective function and the system's constraints to recalculate the controllers output, achieving a better final performance at each instant and compensating for disturbances. Figure 3.1 presents a diagram of the controller structure, including the signals that go in and out of it.

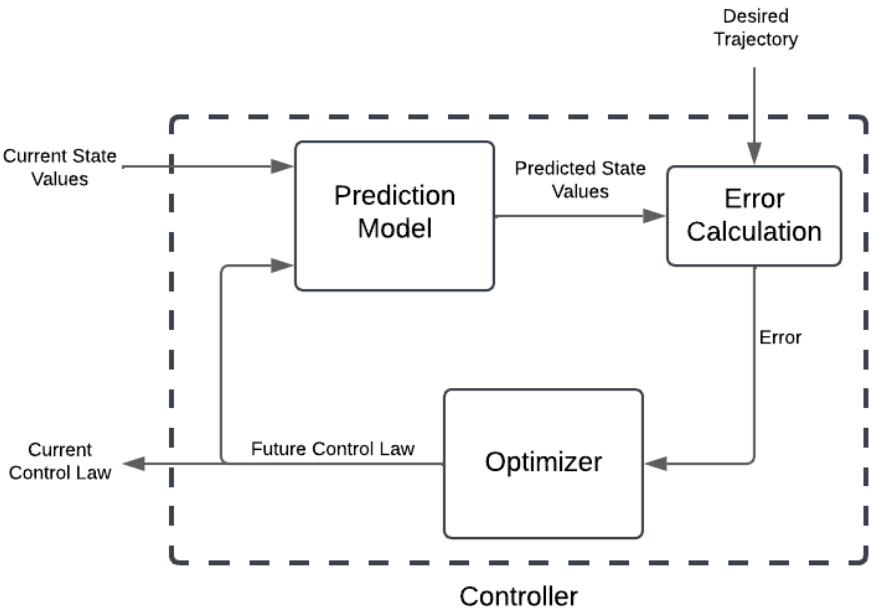


Figure 3.1: Diagram of Basic Components of an MPC Controller

The next sections discusses in more detail how optimization problems are formulated and solved, which establishes the foundations of MPC including its mathematical formulation and the problem that needs to be solved at each control step. Then, one of the most used solvers of MPC problems is introduced, which uses the inner region of the constraints to find the optimal trajectory of the decision variables. Due to this condition,

the methods included on this type of solvers are called interior-point methods. Together, these sections provide a comprehensive overview of both the theoretical foundation and practical considerations necessary for an effective implementation of the MPC.

3.2 Optimal Control Problem

Optimization could be understood as the configuration of decision variables in a system to achieve a certain goal. It is used in many fields, like in Economics, where this could mean the election of which investments to make in order to achieve the maximum profit possible in a certain amount of time, or in Biology, where systems that utilize the minimum amount of resources to get energy tend to survive the longest.

The definition of a goal and the constraints that a system must follow to achieve an optimized state is called the *optimization problem (OP)*. Mathematically it can be defined in a general form as in equation 3.1. Here, $\boldsymbol{f(x)}$ is defined as the *objective function* that will be minimized, $\boldsymbol{h(x)}$ as the equality constraints and $\boldsymbol{g(x)}$ as the inequality ones.

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & h(x) = 0 \\ & g(x) \leq 0 \end{aligned} \tag{3.1}$$

In terms of control engineering, the objective could be to minimize the energy given to the system while reaching a certain state in a limited amount of time. Usually the equality constraints are the dynamics of the system while the inequalities are their physical limitations like input signal saturation. Equation 3.2 shows how it can be defined mathematically for a general $\boldsymbol{\dot{x}=f(x,u,t)}$ system.

$$\begin{aligned} \min_{u} \quad & J(x,u,t) \\ \text{s.t.} \quad & \dot{x} - f(x,u,t) = 0 \\ & u_{inf} \leq u \leq u_{sup} \\ & x \in \mathbb{R}^n, u \in \mathbb{R}^m \end{aligned} \tag{3.2}$$

The solution can be found in different ways, the one used for this work is the *Interior-Point Method*, which finds a solution with a trajectory inside the feasible region.

3.2.1 System discretization

A dynamic system can be discretized to allow the usage of static methods to solve the optimal problem (Kaya and Martínez, 2007). This means that it is necessary to estimate its behavior and calculate the values of its states after each time step $\boldsymbol{\Delta t}$ in a horizon. To choose the appropriate parameters, the characteristics of the process have to be considered.

If the system has a fast dynamic and the time step chosen is wide, the discretized version would divert a lot from the continuous one with the possibility of being unstable.

However, when too small, many variables will be involved in the problem, leading to more time used for the solution calculation, making it unfeasible in the cases of fast-adapting control algorithms.

In a discretized system, the differential equation in time $\dot{x}=f(x,u,t)$ is converted into a difference form $x_k=\hat{f}(x_{k-1},u_{k-1},k)$, where k refers to the step number and $\hat{f}$ depends on the selected time step. The relation between time and step is $t=t_k=k\Delta t$.

Two of the most common approaches to the discretization step are *Euler's Explicit* and *Implicit Methods* which uses and approximation of the differential equation by using small time steps Δt (Chapra and Canale, 2015).

1. *Euler's Explicit Method* or *Forward Method* uses the current information regarding the states values to calculate the approximation of the differential function by using the current derivative as the slope value of a line, so then the next state is calculated as

   $$x_{k+1} = x_k + \Delta t f(x_k)$$

   This ensures an simpler way of calculating the approximation, as all the values needed for the discretization are set on the current time instant.

2. *Euler's Implicit Method* or *Backward Method* is similar to the Explicit Method, being the main difference that it uses the derivative of the next time instant as the slope of the discretized step

   $$x_{k+1} = x_k + \Delta t f(x_{k+1})$$

   Because x_{k+1} is still unknown, to calculate the derivative of the variable at time instant $k+1$, numeric solvers can be used when the system is nonlinear.

The main difference between both types of discretization methods is the calculation simplicity the Forward Method has in comparison with the Backward Method. However, this is also the reason why, in the presence of a curve, the first one tends to increase its error to the external part of the curve, which in the case of processes dynamics means that sometimes it will appear to be unstable even when the original system is naturally stable (Biswas, Chatterjee, Mukherjee, and Pal, 2013). In the second method, the error tendency is to go to the inside sector of the curve, which gives more numerical stability. Figure 3.2 shows the comparison between the trajectory over time of a continuous system in black, the explicit method in blue, and the implicit methods in red.

Due to this behavior, when the system is linear, the implicit method is often selected for the discretization, as an analytical solution can be found without needing numerical methods. To accomplish this, the following equation can be used.

$$x_k = (I_n - \Delta t A)^{-1}(x_{k-1} + \Delta t B u_{k-1}) \tag{3.3}$$

where A and B belong to the original linear system and I_n is an identity matrix of size n. Then, the optimization problem could be defined as in equation 3.4.

$$\begin{aligned} \min_{\overline{x},\overline{u}} \quad & \hat{J}(\overline{x},\overline{u}) \\ \text{s.t.} \quad & \overline{x} - \hat{f}(\overline{x},\overline{u}) = 0 \\ & u_{inf} \leq \overline{u} \leq u_{sup} \end{aligned} \tag{3.4}$$

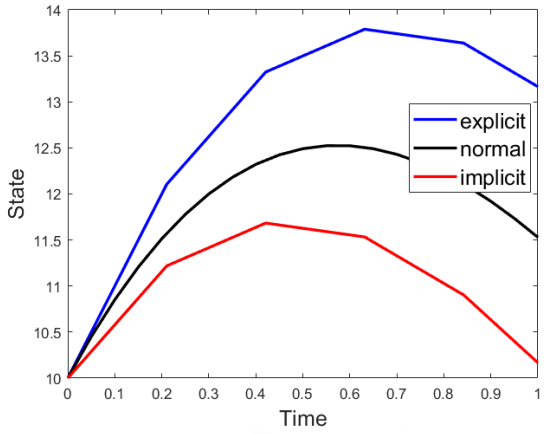


Figure 3.2: Comparison between Discretization Methods and Continuous System

where $\overline{\boldsymbol{x}} = [\boldsymbol{x_0}, \boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_N}]^T$ and $\overline{\boldsymbol{u}} = [\boldsymbol{u_0}, \boldsymbol{u_1}, \boldsymbol{u_2}, \ldots, \boldsymbol{u_N}]^T$ are the vectors of discretized variables $\boldsymbol{x}$, $\boldsymbol{u}$ in a period of time $\boldsymbol{T}=\boldsymbol{N}\boldsymbol{\Delta t}$. Also consider that the objective function has to be discretized, for example, if it involves an integration, it should be converted into a Riemann sum. In this way, a dynamic optimization problem is expressed in a quasi-static form which can be solved using static methods like Interior-Point.

Applied to the MPC case, some special considerations have to be made. The general MPC optimal control problem (Hrovat, Di Cairano, Tseng, and Kolmanovsky, 2012) has the objective of minimizing an objective function $\boldsymbol{J}$, which can be defined as the accumulation of functions depending on the state variables $\boldsymbol{x}$, output $\boldsymbol{y}$, and control $\boldsymbol{u}$, from the current time instant $\boldsymbol{t}$ through the prediction horizon $\boldsymbol{N}$. This is done by achieving an optimal set of values for the control signal, thus having the objective formulation as

$$\min_{u} \sum_{k=0}^{N} J(x(k|t), y(k|t), u(k|t))$$

In (Faulwasser, Mehrez, and Worthmann, 2021) an analysis is made with relation of stability of a control system using MPC without defining the terminal states as constraints or as penalty. Its stated that the stability directly depends on the prediction horizon selected. Additionally, the system dynamics and conditions are used as constraints in the following way:

1. Discrete-time dynamic differential equation:

   $$x(k+1|t) = f(x(k|t), u(k|t))$$

   where $\boldsymbol{f}$ indicates the variation of the state variables from one time instant to the next. It can be linear or non-linear, with a simpler solution being obtained by linearizing it into a matrix configuration.

2. System output:

   $$y(k|t) = h(x(k|t), u(k|t))$$

   where h represents the function that links the output to the state and the input variables.

3. Variable limits:

$$\begin{aligned} x_{min} \leq x(k|t) \leq x_{max}, \quad k = 1, \ldots, N_c \\ y_{min} \leq y(k|t) \leq y_{max}, \quad k = 0, \ldots, N_c \\ u_{min} \leq u(k|t) \leq u_{max}, \quad k = 0, \ldots, N_{cu} \end{aligned}$$

   which can represent physical or operational constraints the variables are subjected to. Here, it is useful to define a different time horizon N_c for the state and output variable, and N_{cu} for the control signal because at some time step a terminal control law can be applied.

4. Initial conditions:

$$x(0|t) = x(t)$$

   which sets the initial state of of the optimization to the current measured states (at time t), so that a degree of feedback control is introduced to increase the robustness.

5. Terminal control law:

$$u(k|t) = \kappa(x(k|t)), \quad k = N_u, \ldots, N-1$$

   which applies a control law κ depending on the state variables from after the control horizon N_u through the end of the prediction horizon. This is done to ensure a certain behavior as it approximates the steady state.

Finally, the optimal control problem for MPC can thus be formulated as

$$\begin{aligned} \min_{u} \quad & \textstyle\sum_{k=0}^{N} J(x(k|t), y(k|t), u(k|t)) \\ \text{s.t.} \quad & x(k+1|t) = f(x(k|t), u(k|t)), \\ & y(k|t) = h(x(k|t), u(k|t)), \\ & x_{min} \leq x(k|t) \leq x_{max}, && k = 1, \ldots, N_c \\ & y_{min} \leq y(k|t) \leq y_{max}, && k = 0, \ldots, N_c \\ & u_{min} \leq u(k|t) \leq u_{max}, && k = 0, \ldots, N_{cu} \\ & x(0|t) = x(t), \\ & u(k|t) = \kappa(x(k|t)), && k = N_u, \ldots, N-1 \end{aligned} \tag{3.5}$$

which is then solved to calculate specially the first value of the set of input variables $u(0|t)$ that is inputted to the control system, completing the cycle that starts with the measurement of the current state values.

3.2.2 KKT conditions

An OP in its general form(Equation 3.1) is solved when the objective function has its minimum value while fulfilling the constraints conditions. If it did not have restrictions, a solution x^* could be found by finding the gradient of its objective function and calculating the Hessian or second derivative to ensure that the solution found is a minimum. When constraints are included, this is not possible because it could result in unfeasible solutions. To avoid this, the Karush-Kuhn-Tucker (KKT) conditions are needed (Ghojogh, Ghodsi, Karray, and Crowley, 2021).

Firstly, the Lagrangian function of the OP is defined as

$$L(x) = f(x) + \lambda_h^T h(x) - \lambda_g^T g(x)$$

that aggregates the objective function with the constraints using a new set of variables $\boldsymbol{\lambda}$ that correspond to each constraint and that are always positive. Then our OP changes to minimize $\boldsymbol{L(x)}$, thus making the first condition:

$$\nabla_x L(x^*, \lambda^*) = 0$$

Then, the equality constraints have to be followed, so the second condition is

$$h(x^*) = 0$$

. The third condition uses the concept of active constraints, as $\boldsymbol{\lambda_{g_i}} \geq 0$ when $g_i(x) = 0$, which means that this condition is followed as if it were an equality constraint, otherwise $\boldsymbol{\lambda_{g_i}} = 0$, leading to the last condition being

$$\lambda_g^{*T} g(x^*) = 0$$

.

Finally, to solve these conditions, a numeric solver like Newton's method can be used, ending in the following equation.

$$\begin{bmatrix} \nabla_{xx}^2 L(x_k, \lambda_k) & \nabla_x h(x_k) & \nabla_x g(x_k) \\ \nabla_x h(x_k)^T & 0 & 0 \\ \nabla_x g(x_k)^T \lambda_{g(k)} & 0 & g(x_k)^T \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda_h \\ \Delta \lambda_g \end{bmatrix} = - \begin{bmatrix} \nabla_x L(x_k, \lambda_{h(k)}) \\ h(x_k) \\ \lambda_{g(k)}^T g(x_k) \end{bmatrix}$$

The resulting linear system could change with the introduction of slack variables, which can transform inequality constraints into equality ones. This is helpful when trying to find solutions by doing steps inside of the feasible region instead of working on the limits.

3.3 Numerical Solution (interior Point Methods)

As stated in (Nocedal and Wright, 1999), the motivation to introduce interior-point methods was introduced from the complexity found in some cases when solving optimization problems using the *simplex method*. The differences between both include the amount of iterations needed to find a solution, where the simplex method uses many inexpensive ones, and that the interior-point methods approach a solution without lying on the constraints imposed, while simplex tests the vertices of its boundaries. Figure 3.3 shows an example of this type of behavior. Here, the blue arrows represent how a traditional simplex method algorithm would check each boundary point of the green-colored feasible region, which takes more iterations than an interior-point algorithm represented by black arrows that end up in the same optimal solution marked by an asterisk in red.

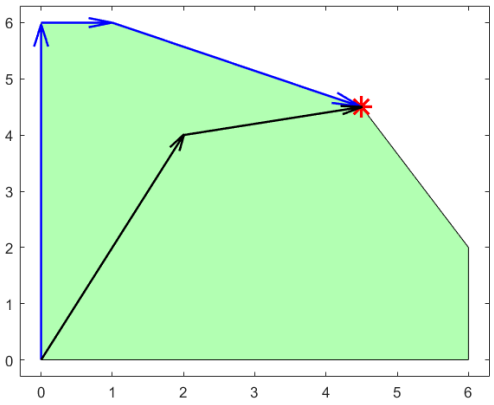


Figure 3.3: Comparison between Simplex and Interior-point Methods

Interior-point methods work by following a path made up of strictly feasible points. For this, a new set of variables that can be used as the measure of the distance between variables and boundaries are needed. Therefore, we introduce *slack variables*, which are always positive and correspond to each inequality constraint to convert them into equality ones. For example, let $\boldsymbol{g(x) \leq 0, g(x_k) : \mathbb{R}^n \rightarrow \mathbb{R}^m}$ be a restriction of an OP, $\boldsymbol{s_k \in \mathbb{R}^{m+}}$ can be defined so that the equality constraint $\boldsymbol{h_g(x_k, s_k) = g(x_k) + s_k = 0}$ can be added to the problem, as well as $\boldsymbol{s_k \geq 0}$. Also, $\boldsymbol{\mu}$ is defined as the parameter that generates a central path and that will be updated at each iteration $\boldsymbol{\mu_k = \tau \frac{\lambda_g^T s}{n_s}}$ where $\boldsymbol{\tau \in [0, 1]}$ and $\boldsymbol{n_s}$ is the number of slack variables introduced; this allows the solution to approach the boundaries after a number of calculations.

With these new variables and parameters, the linear system that allows us to find a solution would be

$$\begin{bmatrix} \nabla^2_{xx} f & A^T_{eq} & 0 \\ A_{eq} & 0 & \hat{I}_s \\ 0 & \hat{S} & \Lambda_g \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta s \end{bmatrix} = - \begin{bmatrix} \nabla_x L \\ \hat{h} \\ \lambda_g \odot s \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \mu e \end{bmatrix} \tag{3.6}$$

where $\boldsymbol{A_{eq} = [\nabla_x h(x_k)^T, \nabla_x h_g(x_k)^T]^T}$, $\boldsymbol{\hat{h}(x, s) = [h(x), g(x) + s]^T}$, $\boldsymbol{\lambda = [\lambda_h, \lambda_g]^T}$, $\boldsymbol{\hat{I}_s = [0, I_{n_s}]^T}$, $\boldsymbol{S = diag(s)}$, $\boldsymbol{\hat{S} = [0, S]}$, $\boldsymbol{\Lambda_g = diag(\lambda_g)}$, $\boldsymbol{e = [1, 1, \ldots 1]^T}$ and $\boldsymbol{\lambda_g \odot s}$ is the element wise multiplication of $\boldsymbol{\lambda_g}$ and $\boldsymbol{s}$.

Something to be noted is that the systems matrix is not symmetric, but the pattern of nonzero entries is, which means that it could be possible to do transformations on the variables for the system to gain symmetry. To accomplish this while avoiding divisions by zero, a new step variable $\boldsymbol{\Delta \tilde{s} = S^{-1} \Delta s}$ can be defined so that the only changes present in the new conditions equation would be the symmetrization of the matrix ending up in

$$\begin{bmatrix} \nabla^2_{xx} f & A^T_{eq} & 0 \\ A_{eq} & 0 & \hat{S}^T \\ 0 & \hat{S} & \Lambda_g S \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \tilde{s} \end{bmatrix} = - \begin{bmatrix} \nabla_x L \\ \hat{h} \\ \lambda_g \odot s \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \mu e \end{bmatrix} \tag{3.7}$$

This new system is now suitable for the use of special factorizations that have as a condition that the matrix must be symmetric, depending on whether it is positive definite

or not, respectively. This means that the newton steps would require less computation to be calculated, as discussed previously in the factorization section.

The system of equation 3.6 can be further reduced to its *augmented form* which uses a smaller symmetric matrix, thus reducing the amount of computational effort needed to solve it.

$$\begin{bmatrix} \nabla^2_{xx} f & A^T_{eq} \\ A_{eq} & -\hat{I}_s \Lambda_g^{-1} \hat{S} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} \nabla_x L \\ \hat{h} + \hat{I}_s \Lambda_g^{-1} (\mu e - \lambda_g \odot s) \end{bmatrix} \tag{3.8}$$
$$\Delta s = \Lambda_g^{-1} (\mu e - \lambda_g \odot s - \hat{S} \Delta \lambda)$$

Another reduction is also possible reaching the *normal-equations form*, where each variable's steps will be calculated separately.

$$\begin{aligned} \Delta \lambda &= D^{-1} (\overline{F_2} - A_{eq} (\nabla^2_{xx} f)^{-1} \overline{F_1}) \\ \Delta x &= (\nabla^2_{xx} f)^{-1} (\overline{F_1} - A^T_{eq} \Delta \lambda) \\ \Delta s &= \Lambda_g^{-1} (\mu e - \lambda_g \odot s - \hat{S} \Delta \lambda) \end{aligned} \tag{3.9}$$

with the previously undefined variables as

$$\begin{aligned} \overline{F_1} &= -\nabla_x L \\ \overline{F_2} &= -\hat{h} + \hat{I}_s \Lambda_g^{-1} (\mu e - \lambda_g \odot s) \\ D &= -\hat{I}_s \Lambda_g^{-1} \hat{S} - A_{eq} (\nabla^2_{xx} f)^{-1} A^T \end{aligned}$$

In (Roos et al., 2006), a preprocessing step is advised, this includes the detection of redundancies in the constraints and elimination of linear dependencies. Also, it states that with permutations, like minimum-degree algorithms, the solution of each step could make the implementation more efficient. Another thing to consider is the feasibility of the solutions, as the calculation of Newton steps can lead some variables to exit the boundaries imposed by the problem; this is solved by adding scaling so that their steps remain always in the feasible region.

4 Theoretical Framework

This chapter is meant to be used to recollect and summarize the theoretical background needed to develop the algorithms used in the design stage. It starts with the concept of factorization, which is widely used when solving linear equation systems because it results in triangular matrices that can be solved as the results of Gaussian elimination with low computational costs. Then a reordering algorithm based on the minimum degree from graph theory is introduced, which serves as the base used in the first solution proposal. Finally, an overview on system optimization is done, including the basics of interior-point methods, which will then be used as the testing environment for the strategies in the design.

4.1 Factorization

A linear system of the form $\boldsymbol{Ax} = \boldsymbol{b}$ where $\boldsymbol{A} \in \mathbb{R}^{\boldsymbol{n}\times\boldsymbol{n}}$, $\boldsymbol{b} \in \mathbb{R}^{\boldsymbol{n}}$ can be solved using Gaussian elimination, reducing the system in a way that allows the $\boldsymbol{A}$ matrix to have an upper triangular structure (Haddad, 2009). By using a factorization method, it is possible to decompose a matrix into the product of a lower and upper-triangular matrices. The selection of which method to use depends on the structure and characteristics of the original matrix, with a focus on symmetry and positive definiteness. When matrices have dimensions where the element-by-element factorization method can be too computationally costly, block methods were developed to take advantage of parallel computing on modern hardware, such as a multicore Graphics Processing Unit (GPU) (D'Azevedo and Hill, 2012). This greatly increases computational efficiency, but can also lead to unstable results when the structure of the matrix is not considered; for example, the presence of singular blocks makes the resulting matrices undefined.

This section presents the most common factorization methods used: Cholesky, LDL, and LU, with their block forms. Each has different requirements for their use and also has different computation complexity, which will be an important decision point when implementing them in the system solution.

4.1.1 Cholesky Factorization

Similar to how real positive numbers have a square root, square-symmetric positive definite matrices can be decomposed into the product of an upper-triangular matrix and its transpose (Davis, 2006, Chapter 4). This is called Cholesky factorization and the reduced form is shown in equation 4.1.

$$A = LL^T \tag{4.1}$$

Equations 4.2 and 4.3 can be used to calculate the entries for the decomposed matrices, the first for the elements in the diagonal and the second for the rest. Here, the presence of square roots can be noted, which is why a positive definite matrix is needed.

$$l_{i,i} = \sqrt{a_{i,j} - L_{\neq i,i} L^T_{\neq i,j}} \tag{4.2}$$

$$l_{i,j} = \frac{1}{l_{i,i}} (a_{i,j} - L_{\neq i,i} L^T_{\neq i,j}) \tag{4.3}$$

This type of factorization is the least complex one and, due to the result in a product of two triangular matrix, is also the most time-efficient when solving linear system equations.

A further improvement in the efficiency of computation can be made by dividing the matrix $\boldsymbol{A}$ into submatrices called blocks, of size $\boldsymbol{d}$, which can then be loaded directly into the computers cache, thus reducing the access time of the data (Chen, Jin, Shi, Qiu, and Liu, 2013). Then, to calculate the block Cholesky factorization, the following steps can be followed:

1. Divide the original matrix into 4 blocks, where $\boldsymbol{A}_{11}$, $\boldsymbol{A}_{22}$ and $\boldsymbol{A}_{21}$ are of sizes $\boldsymbol{d}$, $\boldsymbol{n - d}$ and $(\boldsymbol{n - d}) \times \boldsymbol{n}$, respectively. The triangle matrix $\boldsymbol{L}$ follows the same dimensions for its blocks.

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}$$

2. Solve $\boldsymbol{A_{11} = L_{11} L_{11}^T}$ using normal Cholesky method.
3. Solve $\boldsymbol{L_{21}^T = L_{11}^{-1} A_{21}^T}$.
4. Define a new matrix $\boldsymbol{A' = A_{22} - L_{21} L_{21}^T = L_{22} L_{22}^T}$.
5. If the size of $\boldsymbol{A'}$ is larger than $\boldsymbol{d}$, repeat the procedure from Step 1. If not, calculate the last block with traditional Cholesky.

This procedure can be repeated in other factorizations considering minor changes in the structure of the resulting matrices.

4.1.2 LDL Factorization

A square symmetric matrix that is either negative definite or indefinite can be factorized by using the LDL decomposition (Golub and Van Loan, 2013). This allows the complexity to be equal to Cholesky's with $O(n^3/3)$, but with the disadvantage of resulting in three matrices, which means that it takes longer to solve linear system equations. Equations 4.4 show how a matrix $\boldsymbol{A}$ with the characteristics previously established can be decomposed into a lower triangular matrix whose diagonals are all unitary entries $\boldsymbol{L}$ and a diagonal matrix $\boldsymbol{D}$.

$$A = LDL^T \tag{4.4}$$

Equations 4.5 and 4.6 can be used to calculate each entry for both $\boldsymbol{L}$ and $\boldsymbol{D}$. The order to calculate these entries is to first get the diagonal matrix entries that are used to calculate the ones in the lower triangular matrix.

$$d_{i,i} = a_{i,i} - \sum_{k=1}^{i-1} d_{k,k} l_{i,k}^2 \tag{4.5}$$

$$l_{i,j} = \frac{1}{d_{i,i}} \left(a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} d_{k,k} l_{j,k} \right) \tag{4.6}$$

This factorization can also be converted to block form, applying the same steps as in the previous case, but using the following configuration and replacing each traditional Cholesky factorization with an LDL procedure:

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} D_{11} & 0 \\ 0 & D_{22} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}$$

Another way of applying this method is shown in other works such as (Higham, 1999), where D is a block diagonal matrix with the following configuration:

$$A = \begin{bmatrix} A_{11} & A_{21}^T \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I_d & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} D_{11} & 0 \\ 0 & D_{22} \end{bmatrix} \begin{bmatrix} I_d & L_{21}^T \\ 0 & L_{22}^T \end{bmatrix}$$

with the solution being calculated by following the steps:

1. Set the value of $\boldsymbol{D_{11} = A_{11}}$.
2. Solve $\boldsymbol{L_{21} = A_{21} D_{11}^{-1}}$.
3. Define a new matrix $\boldsymbol{A' = A_{22} - L_{21} D_{11} L_{21}^T = L_{22} D_{22} L_{22}^T}$.
4. If the size of $\boldsymbol{A'}$ is bigger than $\boldsymbol{d}$, divide it similar to the initial block configuration and go to step 1. If its size would accept only one block, then set $\boldsymbol{L_{22}}$ as an identity matrix of the appropriate size, and then the last step changes to solving $\boldsymbol{D_{22} = A_{22} - L_{21} D_{11} L_{21}^T}$.

This method is less complex than the previous one, but it can take more time to solve the equation system, because the matrix $\boldsymbol{D}$ is block diagonal. This is why selecting an appropriate size $\boldsymbol{d}$ for the submatrices is important to allow the use of banded solvers.

4.1.3 LU Factorization

When the matrix is neither symmetric nor positive definite, a more complex factorization can be used that also allows the use of triangular matrices to describe it. This reduction is called LU and is shown in Equation 4.7

$$A = LU \tag{4.7}$$

where $\boldsymbol{L}$ is a lower-triangular matrix and $\boldsymbol{U}$ is an upper-triangular with the same dimension as $\boldsymbol{A}$. The matrix $\boldsymbol{A}$ only needs to be square and non-singular for this factorization

method to be applied. Equations 4.8 and 4.9 can be used to calculate the entries from the decomposed matrices.

$$l_{i+1,j} = \frac{u_{i+1,j}}{u_{j,j}} \tag{4.8}$$

$$U_{j+1:n,j:n} = U_{j+1:n} - L_{j+1:n,j} U_{j,j:n} \tag{4.9}$$

The matrix $\boldsymbol{L}$ has to be initialized as identity, and $\boldsymbol{U}$ as the original $\boldsymbol{A}$. A permutation step can also be introduced to guarantee the presence of nonzero values on the diagonal of $\boldsymbol{A}$ and to avoid mathematical errors that could appear in the calculation.

Compared to the other factorization methods, this one is more complex with $O(n^3)$ (Lu, Luo, Lian, Jin, and Liu, 2021), which is the reason it should be avoided if the characteristics of the matrix are suitable for the other reductions.

This method can also be calculated in a block form in a similar way as Cholesky an additional step to calculate the asymmetry of the matrices (J. Demmel, Higham, and Schreiber, 2000). Then, with the configuration

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{bmatrix} \begin{bmatrix} U_{11} & U_{21} \\ 0 & U_{22} \end{bmatrix}$$

the solution steps can be set as:

1. Solve $\boldsymbol{A_{11} = L_{11} U_{11}}$ using normal LU factorization.
2. Solve $\boldsymbol{L_{21} = A_{12} U_{11}^{-1}}$ and $\boldsymbol{U_{12} = L_{11}^{-1} A_{21}}$
3. Define new matrix $\boldsymbol{A' = A_{22} - L_{21} U_{12} = L_{22} U_{22}}$.
4. If the size of $\boldsymbol{A}'$ is larger than $\boldsymbol{d}$, repeat Step 1. If not, calculate the last block with traditional LU.

Although there is still the need of doing normal LU factorizations, the size of the submatrices are smaller and therefore more efficient compared to doing it on the complete matrix.

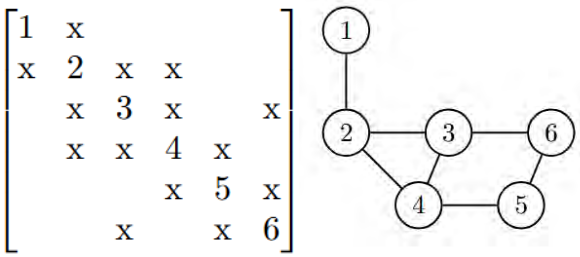
One thing to note is that, unlike normal LU, the block form has been shown to be unstable even when the matrix is symmetric positive definite (J. W. Demmel and Higham, 1992), therefore the algorithms that use this method must be tested for their specific use.

4.2 Degree based Permutation

In the last section, factorization methods were presented to reduce a matrix to the product of two triangular matrices to solve a linear system. Although this strategy is useful, when the original matrix is large and sparse, the resulting factorization can result in matrices that are less sparse and require more memory than the original system. To avoid the generation of fill-ins, which are the new values that appear on the factorization, a reordering step is done before the reduction.

As stated in (Amestoy et al., 1996), the nonzero pattern of a sparse symmetric matrix can be represented using graph theory, this can be seen in Figure 4.1 where each row

represents a node and each value is an edge between nodes, the number of edges a single node has is defined as its degree. The graphs allow us to select nodes that when pivoted show the amount of fill-ins generated, thus presenting a way to find the order of permutations desired.



(a) Matrix (b) Graph representation

Figure 4.1: Graph Representation of a Matrix

In this section, two alternatives of finding the pivot nodes are presented, the main difference between both the use of heuristics on approximate minimum degree due to the complexity of finding a suitable solution in the case of true minimum degree.

4.2.1 Elimination and Quotient Graphs

A sparse symmetric matrix represented as an elimination graph is divided at each iteration into two sets, one of the nodes $\boldsymbol{V^k} = \{1, 2, \ldots, n\}$ and one of the edges $\boldsymbol{E^k} = \{(i, j)\}$ where $\boldsymbol{i, j} \in \boldsymbol{V^k}$. After selecting a pivot node $\boldsymbol{p}$, it should be removed from the node set and the edges between its adjacent set of nodes should be updated at each iteration $\boldsymbol{Adj_k(p)}$.

It can be noted that, at each iteration of pivoting on an elimination graph, the number of edges present would be equal to or higher than before, thus needing more memory. To solve this and get a representation where the storage requirement is equal to or less than before, a quotient graph can be used. The main similarity between both graphs is the nodes(now called *variables*) and edges sets, but also the eliminated nodes will be stored in a new set of *elements* $\overline{\boldsymbol{V}}^{\boldsymbol{k}}$, and the edges between variables and elements are also stored in a new set $\overline{\boldsymbol{E}}^{\boldsymbol{k}}$.

In order for the algorithm to use a quotient graph, the following sets are useful to define. Consider $\boldsymbol{i}$ as a variable in $\boldsymbol{V^k}$

- $A_i^k = \{j : (i, j) \in E^k\}$: set of *variables* adjacent to node $\boldsymbol{i}$.
- $\mathcal{E}_i^k = \{e : (i, e) \in \overline{E}^k\}$: set of *elements* adjacent to node $\boldsymbol{i}$
- $L_e^k = \{i : (i, e) \in \overline{E}^k\}$: set of *variables* adjacent to *element* $\boldsymbol{e}$.

To calculate the true degree of a node or variable $\boldsymbol{i}$, equation 4.10 can be used.

$$d_i^k = \left| \left(A_i^k \cup \bigcup_{e \in \mathcal{E}_i^k} L_e^k \right) \setminus \{i\} \right| \tag{4.10}$$

Two variables i, j can be called indistinguishable and can merge into a *supervariable* s_i when $\boldsymbol{A_i^k \cup E_i^k \cup \{i\} = A_j^k \cup E_j^k \cup \{j\}}$, this means that the variables and elements of two nodes are the same, so both can be counted as just one pivot in the next iteration using its external degree as:

$$d_{s_i}^k = |A_i \setminus s_i| + \left| \bigcup_{e \in \mathcal{E}_i^k} L_e^k \setminus s_i \right| \tag{4.11}$$

The algorithm 1 shows the process for updating the sets $\boldsymbol{A_{s_p}^k}$, $\boldsymbol{\mathcal{E}_{s_p}^k}$ and $\boldsymbol{L_{s_p}^k}$ from the supervariable s_p in the $\boldsymbol{k}$-th iteration. First, as the pivot changes from a variable to an element, its set L has to be computed as the variables adjacent to it united with the set of variables connected to its adjacent elements minus the pivot itself. Then the sets of variables connected to the pivot have to be updated to remove the pivot from its set A and add it to its set $\boldsymbol{\mathcal{E}}$.

Algorithm 1 Quotient Graph Update Algorithm

Select a pivot supervariable s_p
$L_{s_p}^k = \left(A_{s_p}^{k-1} \cup \bigcup_{e \in \mathcal{E}_{s_p}^{k-1}} L_e^{k-1} \right) \setminus s_p$
for i in $L_{s_p}^k$ **do**
  $A_i^k = \left(A_i^{k-1} \setminus L_{s_p}^{k-1} \right) \setminus s_p$
  $\mathcal{E}_i^k = \left(\mathcal{E}_i^{k-1} \setminus \mathcal{E}_{s_p}^{k-1} \right) \cup s_p$
  Calculate d_i^k
end for

In the next sections, two strategies will be presented in order to select the pivot and to update the degrees of the variables.

4.2.2 Minimum Degree

The Minimum Degree Algorithm uses the exact external degree (equation 4.11) of all nodes, including supervariables, to find those who have the lowest degree. This is done because the amount of new edges generated depends on the quantity of adjacent nodes the pivot has, thus the degree is a good indicator on how many fill-ins will be generated after each elimination, then to minimize their formation, the pivot should have the lowest possible degree. This Algorithm has a high solution calculation complexity and takes more time to be solved because it has to calculate the exact external degree at each iteration. Therefore, some methods were used to use approximate degrees' calculations to reduce the computation costs.

4.2.3 Approximate Minimum Degree

In (Amestoy et al., 1996) the use of upper bounds $\overline{d}_{s_i}^k$ to approximate the degree is presented. Those include worst-case scenarios, such as a fully connected node where

the degree would be $\left|V^k\right| - 1$ or the most number of fill-ins generated in one iteration as $\overline{d}_{s_i}^{k-1} + \left|L_{s_p}^k \setminus s_i\right|$. It also uses set theory to set the following boundary:

$$\left|\bigcup_{e \in \mathcal{E}_i^k} L_e^k \setminus s_i\right| \leq \left|L_{s_p} \setminus s_i\right| + \sum_{e \in \mathcal{E}_i^k \setminus s_p} \left|L_e \setminus L_{s_p}\right|$$

thus the external degree would be bounded like in equation 4.12.

$$\overline{d}_{s_i}^k = min \begin{cases} \left|V^k\right| - 1, \\ \overline{d}_{s_i}^{k-1} + \left|L_{s_p}^k \setminus s_i\right|, \\ \left|A_{s_i}^k \setminus s_i\right| + \left|L_{s_p}^k \setminus s_i\right| + \sum_{e \in \mathcal{E}_i^k \setminus s_p} \left|L_e^k \setminus L_{s_p}^k\right| \end{cases} \tag{4.12}$$

In the same work two other approximations from different authors are presented:

- $\hat{d}_{s_i} = |A_{s_i} \setminus s_i| + \sum_{e \in \mathcal{E}_{s_i}} |L_e \setminus s_i|$
- $\tilde{d}_{s_i} = \begin{cases} d_{s_i} & if |\mathcal{E}_{s_i}| = 2 \\ \hat{d}_{s_i} & otherwise \end{cases}$

Compared, $\hat{d}$ was shown to be the most relaxed bound, while $\overline{d}$ is the tightest. Another thing to be considered is that in the cases of having just two or fewer elements adjacent to any supervariable, its bounded and exact degrees would be equal.

The advantages of using these types of algorithms to reduce the calculation times for the solutions of equation systems can be seen in the field of power distribution networks (Guo et al., 2021), usually used in combination with factorization methods (Selman, 2016). This can also be extended to the field of optimal control, as the number of variables and equations to be solved are vast, making these methods necessary for the feasibility of the solution for specific dynamic systems.

Figure 4.2 shows the steps in which the method would be applied to the matrix presented before to obtain a permutation order by calculating the degrees. The first image (a) shows the graph representation of the matrix, and here the node with lower degree is evidently 1, as it only presents one edge, thus this is selected and eliminated from the graph. In the second image (b), there are 3 possible pivots, because nodes 2, 5 and 6 present the same amount of edges. For illustrative purposes, node 6 was chosen, so it had to be eliminated, meaning that a new edge between 3 and 5 is created. Then, the third (c) and fourth (d) images show how a supervariable is separated. Here, it is evident that nodes 3 and 4 have the same edges to other variables, and between themselves, thus they can be considered a supernode which is labeled $S_{3,4}$. The next pivot is then selected to be 2, as it has the same degree as 5, but lower than the supervariable. In the fifth image (e), any node could be selected as a pivot, with the sixth image (f) being the leftover. This means that for the matrix presented in 4.1, the order in which it obtains a reduction in the amount of fill-ins when factorizing is $1, 6, 5, 2, 5, 3, 4$. This can then be extended for larger matrices, including the ones produced by the KKT conditions of optimal control problems.

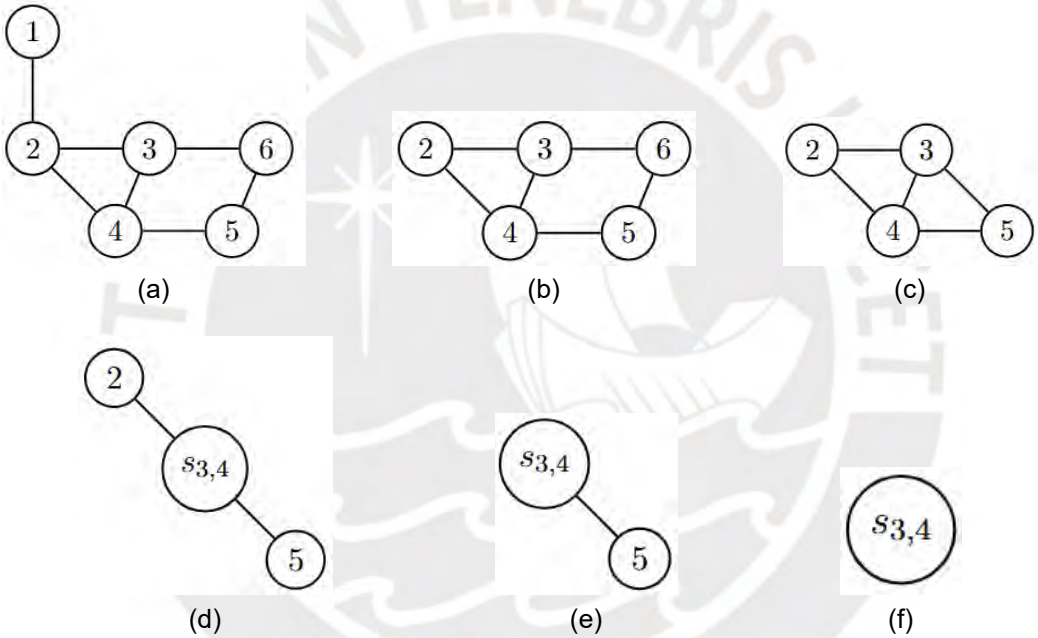


Figure 4.2: AMD Example Procedure

5 Design

In this chapter, the main problematic of the work is presented based on the analysis of the previous chapters. Then, the theoretical background will be used to propose solutions that could solve it, having a detailed explanation on their functionality and how it was implemented in the testing environment.

5.1 Problem Definition

One of the biggest challenges when developing optimal control algorithms is managing the computational complexity of the number of variables involved present. These depend directly on the time horizon, discretization method and sample time, as longer prediction periods with more frequent sampling can produce an increased amount of decision variables and constraints.

This is especially important in systems with fast dynamics, like those that require near-real-time updates on the control law, as the solution must provide quick responses while maintaining a certain prediction accuracy. Although some methods have been developed to solve optimal control problems efficiently, such as Model Predictive Control (MPC), the scalability of the problem becomes an obstacle.

That is why there is a need to develop methods that can accelerate the solution of optimal control problems without sacrificing general performance. One of the aspects that can be improved is the enhancement of the solutions for the KKT condition equations that can be presented as simple linear systems.

5.2 Proposed Solutions

After analyzing previous chapters for strategies to accelerate the calculation of the solution of the KKT condition equation systems, two strategies were selected and tested with the potential to accomplish this objective. Both involve taking advantage of the structure and information of the KKT matrix to skip some calculations that slow the computation of the optimal solution to the problem.

The first one focuses on the use of an approximate minimum-degree reordering to minimize the apparition of fill-ins when inverting the KKT matrix. This method takes advantage of the effect the internal topology of a matrix has on the amount of entries that its inverse has which, in addition to the capacity of sparse matrix calculations to ignore empty spaces, allows for faster factorization and then solution of linear equation

systems. Additionally, the generation of the permutation order can be done offline in cases where the KKT matrix has a constant structure, which applies to the systems used in this work, avoiding the computer complexity that comes with the use of these procedures. To achieve the goals of this strategy, an approximate minimum degree algorithm was first developed based on the work of (Amestoy et al., 1996). Then, an interior-point solver was made, considering their capacity to solve optimal control problems with a large amount of variables while being time efficient. This, combined with the offline calculation of the permutation order, would show the effects this method has compared with the non-permuted case.

The second strategy consists of making the factorization step used when solving equation systems faster, by using information regarding the changes that the KKT matrix has at each iteration of the interior point solver. A factorization step is usually made to solve a set of equations because it allows faster computation as triangular systems of equations are easier to calculate. By knowing which regions it has that are more subjected to variations, a special permutation can be done to allocate those entries on the upper left side of the matrix. Block factorization methods can also be used to take advantage of their ability to calculate the resulting triangular matrices faster. This, combined with the information about the more constant sections and their allocation in the first blocks to be processed, allows the factorization step to just update the resulting matrices that could be defined offline, reducing in this way the computational complexity of this procedure. This method was tested by creating a block LDL factorization function according to the symmetry of the KKT matrix of the dynamic system to be used in the first iteration, as well as a constant information one that would update the values of the triangular matrices while the optimization problem is solved. Then, a change in the parameters can be made to analyze how the size of the constant sector or blocks used to operate affect the calculation time.

5.3 Testing Environment

The previously mentioned approaches need an environment where they can be tested. This was made possible by developing an algorithm using an interior-point method using a test model to be controlled. For this, a linear dynamical system is introduced in the form of a double integrator

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

that is chosen due to its simplicity and because of its similarity to real kinematics models. In this regard, $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ could be understood as the position coordinates of a vehicle, while $\boldsymbol{x}_2$ and $\boldsymbol{x}_4$ would be the velocities related to such variables, while both control variables can be seen as accelerations. This could also be the result of the application of advanced control techniques in nonlinear systems such as exact feedback linearization (Estrada, Li, and Cai, 2021).

Then, to use it in the context of a quasi-static optimal problem, a discretization step must be done. To do this, a sampling time was selected based on the characteristics of the system and by testing, with a suitable range found in $\boldsymbol{\Delta t} \leq 0.2s$. This together with the time horizon $\boldsymbol{T} = 15s$ will generate the sufficient number of variables to evaluate the performance of the solution proposals.

The discretization method selected is Euler's implicit method, due to its higher stability and the systems' linearity avoiding the need of mathematical methods for prediction of future state values. This generated the following difference equation when the sampling time was set at $0.1s$

$$\begin{bmatrix} x_1(k+1) \\ x_2(k+1) \\ x_3(k+1) \\ x_4(k+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(k) \\ x_2(k) \\ x_3(k) \\ x_4(k) \end{bmatrix} + \begin{bmatrix} 0.01 & 0 \\ 0.1 & 0 \\ 0 & 0.01 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} u_1(k) \\ u_2(k) \end{bmatrix}$$

which will be used to define the equation constraints of the variables in the optimization problem as:

$$x(k+1) - A_d x(k) - B_d u(k) = 0$$

Also, to fully define the system, initial conditions were set at $\boldsymbol{x}_0 = [5, 10, 5, 5]^T$ as it was shown to be a suitable starting point for adequately testing the solutions.

Having the discretized model of the system, the next step is the definition of the goal to achieve. Defining the objective of the control to be either a tracking or regulation problem would help with the objective function. In the case of this test environment, a tracking objective function was selected, so the first goal is to minimize the total error through the horizon, which can be represented as $\sum \boldsymbol{q}_x(\boldsymbol{x} - \boldsymbol{x}^*)^2$. Furthermore, this should be achieved by using the least amount of energy possible, for which the minimization of the input variable was set as the second goal $\sum \boldsymbol{q}_u \boldsymbol{u}^2$. This results in the following matrix formulation of the objective function

$$J(x, u) = \begin{bmatrix} x - x^* & u \end{bmatrix} Q \begin{bmatrix} x - x^* \\ u \end{bmatrix}$$

where $\overline{\boldsymbol{x}}$ and $\overline{\boldsymbol{u}}$ represent the discretized values of the states and input variables in the complete time horizon, respectively, and $\boldsymbol{Q}$ is the diagonal weight matrix.

In terms of inequality constraints, the first ones considered were the limitations on each control variable $-5 \leq \boldsymbol{u} \leq 5$ which can be understood as a saturation to prevent damage to a controller output. When testing, another set of constraints was considered to check the performance of the approaches in the presence of nonlinear ones, for this, a limitation on the resultant velocity was established $\sqrt{\boldsymbol{x}_2^2 + \boldsymbol{x}_4^2} \leq 12$. The values selected for both inequality constraints were arbitrarily determined considering the initial conditions. In the implementation of the interior-point method as discussed in chapter 4 Theoretical Framework, slack variables have to be introduced to change the inequality into equality constraints to get

$$\begin{aligned} u - s_1 &= -5 \\ u + s_2 &= 5 \\ \sqrt{x_2^2 + x_4^2} + s_3 &= 12 \end{aligned}$$

The linear constraints can be grouped into a concise matrix form

$$\hat{h} = A_{eq}\tilde{x} + S_{eq}s + C = 0$$

where $\tilde{\boldsymbol{x}} = [\boldsymbol{x}(k), \boldsymbol{u}(k)]^T$ and $s = [\boldsymbol{s}_1, \boldsymbol{s}_2]^T$. Then, the matrices can be defined as follows:

$$A_{eq} = \left[\begin{array}{cccc|cc} I & 0 & 0 & \dots & 0 & 0 & \dots \\ -A_d & I & 0 & \dots & -B_d & 0 & \dots \\ 0 & -A_d & I & \dots & 0 & -B_d & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\ \hline 0 & 0 & 0 & \dots & e_2 & 0 & \dots \\ 0 & 0 & 0 & \dots & 0 & e_2 & \dots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \end{array}\right]$$

where $\boldsymbol{I}$ is a 4×4 identity matrix and $\boldsymbol{e}_2$ is the vector $[1, 1]^T$,

$$S_{eq} = \left[\begin{array}{ccc} 0 & 0 & \dots \\ 0 & 0 & \dots \\ 0 & 0 & \dots \\ \vdots & \vdots & \ddots \\ \hline \hat{I} & 0 & \dots \\ 0 & \hat{I} & \dots \\ \vdots & \vdots & \ddots \end{array}\right]$$

where $\hat{I}$ is the matrix $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$,

$$C = \left[\begin{array}{c} x_0 \\ 0 \\ 0 \\ \vdots \\ \hline 10\hat{e}_2 \\ 10\hat{e}_2 \\ \vdots \end{array}\right]$$

where $\hat{e}_2$ is the vector $[-1, 1]^T$. Applying this conversion into a matrix form will make the calculation of gradients easier and thus making the KKT matrix easier to define. Equation 5.1 represent the definition of the optimal control problem.

$$\begin{aligned} \min_{x,u,s} \quad & \begin{bmatrix} x - x^* & u \end{bmatrix} Q \begin{bmatrix} x - x^* \\ u \end{bmatrix} \\ \text{s.t.} \quad & A_{eq}\tilde{x}(k) + S_{eq}s(k) + C = 0 \\ & \sqrt{x_2^2 + x_4^2} + s_3 = 12 \\ & s_1, s_2, s_3 \geq 0 \\ & k = 0, 1, 2, \dots N - 1 \end{aligned} \tag{5.1}$$

Having defined the OCP, the next step is to generate the KKT conditions matrix from the Newton steps to solve it. Firstly, it will be shown how it is set when the nonlinear constraint is not present. To do this, the first term is defined to be

$$\nabla^2_{xx} L(x(k), \lambda(k)) = Q$$

then, the second term related to the gradient of the constraints by the state and control variables

$$\nabla_x \hat{h}(x(k)) = A^T_{eq}$$

Finally, the KKT conditions matrix ends up being

$$KKT_{matrix} = \begin{bmatrix} 2Q & A^T_{eq} & 0 \\ A_{eq} & 0 & S^T_{eq} \\ 0 & S_{eq} & \Lambda_g S \end{bmatrix} \tag{5.2}$$

and it allows to use Newton steps to find the variation in the decision variables per iteration solving the equation system from 3.7. Then, to finish defining the linear equation system, the right side has to be established. To do this, $\nabla_x \boldsymbol{L}$ can be defined as

$$\nabla_x L = 2Q \begin{bmatrix} x - x^* \\ u \end{bmatrix} + A_e q^T \lambda$$

Now, to make sure that the slack variables fulfill their positive condition, a step size adjustment has to be make. For this, a resize factor $\boldsymbol{\alpha}$ is introduced to ensure that the update to each variable is limited to a 80% reduction in the value of the lesser slack variables.

$$\alpha = 0.8 \min(1, -\frac{s_k}{\Delta s_k}), \forall \Delta s_k < 0$$

Matlab was chosen as the development platform for this and the following algorithms due to its robust capabilities for numerical computation, which is fundamental when trying to operate with matrices of varying sizes and shapes. Additionally, their built-in functions for handling sparse matrices make them well suited for the performance of the different approaches that will be tested.

Figure 5.1 shows the iterative trajectories of states $\boldsymbol{x}_1$ and $\boldsymbol{x}_3$ solved by the algorithm. The path of the variables is depicted in blue, whereas the final point is marked in red. After each iteration, the solver makes the trajectory converge to the desired target at the origin, demonstrating the effectiveness of the optimization algorithm. In particular, the final optimal solution shows a smooth behavior, meaning that no abrupt changes happen in the states, which is expected because the control variables are constrained to a specific range of values.

Furthermore, Figure 5.2 shows the time evolution of the variables in the final iteration of the solver. Highlight its effectiveness by showing how the variables approach their steady target. In addition, the control variables satisfy their constraints in the complete evaluated time range. In particular, $\boldsymbol{u}_1$ starts with values that are in the proximity of its limitation, reflecting the influence the initial conditions have on the solution. This behavior depicts the solver's ability to operate in the defined boundaries while driving the solution towards optimal performance.

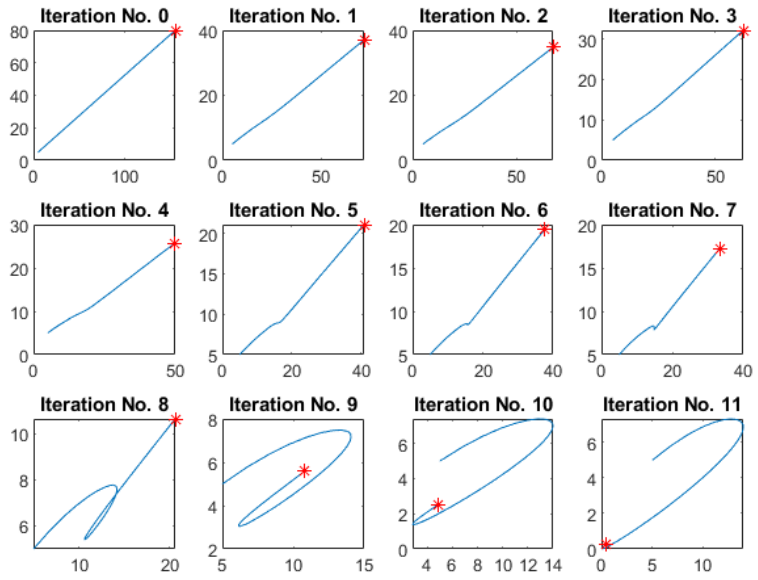


Figure 5.1: Trajectories of the Solution of the Solver

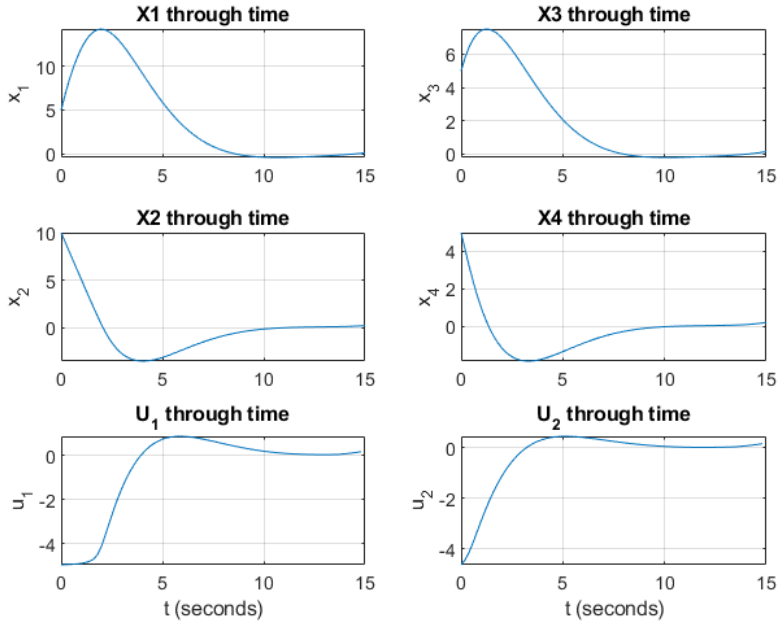


Figure 5.2: Values of the variables at the optimal solution

In the case of the additional nonlinear constraint, the first block of the KKT matrix has to add another that is the result of its second derivation related to the variables. Additionally, the block A_{eq} would add columns with the values of the first derivation of the constraint. This increases the size of the matrix, its complexity of solution and its topology, which influences the performance of the approaches, as will be seen later.

In summary, the testing environment consists of an interior-point algorithm that solves optimal control problems. To accomplish this, first, the problem is established as in

Equation 5.1, then the KKT equation system is defined, including the matrix and the result vector, and the variable steps can be calculated. This is the framework needed to test the approaches, as it gives real case values for the KKT matrix and allows their implementation without changing much of the solution algorithm.

5.4 Permutation Approach

The internal structure of the linear equation matrix significantly influences the factorization step of a linear system solver, due to the determination of the number of nonzero values present on the resulting triangular matrices. This, in turn, means that it also has a great impact on its computational complexity, since denser systems require more iterations to be solved. To reduce this effect, a reordering of the matrix is essential. Following what was discussed in the previous chapter, an Approximate Minimum Degree algorithm was developed as an effective strategy for optimizing this process.

The AMD developed is based on the work of Amestoy (Amestoy et al., 1996) and was implemented using Matlab because it has built-in functions that were optimized considering the operation of large sparse matrices. This makes the iterative calculations easier to process and allows for a relatively fast solution in the presence of the amount of variables the OPC has in comparison with dense-matrices methods.

The algorithm expects a square matrix and checks for symmetry, adding a symmetrization step of the form $Q_s = Q+Q^T$ when needed, as the implementation uses the concept of undirected graphs, which are represented by symmetric square matrices with every entry representing the edges between nodes. Then, an initial calculation of the degree is done by counting the number of edges a node has, which also reveals which of them have the same connected variables. With these established, the quotient graph can be defined by initializing its sets of variables $\boldsymbol{A}_i$, $\boldsymbol{L}_e$, and elements $\mathcal{E}_i$ as stated in the theoretical framework.

After being initialized, the algorithm makes a number of iterative calculations that depend on the number of nodes and supervariables present, the last being updated at each evaluation due to the possible creation of new ones. At each iteration, the lowest degree node is found, which is selected as the next pivot and moved from the variable to the element set, updating at the same time the relation sets for the other nodes. Then, the bounds for the updated degrees are calculated using equation 4.12 so that the exact minimum degree calculation is avoided and they can be used for the next iteration without greatly increasing the computer complexity of the algorithm. After the last node is evaluated, a vector with pivot order is generated which indicates the optimal permutation of the matrix.

The performance of the algorithm in relation to the number of fill-ins generated after factorization was tested using matrices from the SuiteSparse Matrix Collection[1] and those obtained from the OPC simulation. The main focus of the comparison is to reflect how the algorithm produces a permutation order that gives less nonzero entries with

[1] Formerly the University of Florida Sparse Matrix Collection https://sparse.tamu.edu/ (Davis and Hu, 2011) Visited last time on: 20.12.2024

respect to other similar functions. Table 5.1 shows the matrices compared to their sizes as n, initial nonzero entries as nz and after factorizing as nz_{fact}, also metrics for this algorithm as $oAMD$ and one implemented as a native Matlab function as $mAMD$. The first three matrices are the ones needed to inverse for the OP when using the complete system, the augmented form, or the normal-equations, in the same order.

Matrix	n	nz	nz_{fact}	nz_{oAMD}	nz_{mAMD}
dF_comp	2708	8112	551708	11402	12284
dF_aug	2108	6912	360104	9602	10352
dF_ne	1204	6604	103752	7244	5100
1138_bus	1138	4054	58651	5148	5283
G54	1000	11832	951347	505023	582174
sherman1	1000	3750	49880	50753	53554
bp_1200	822	4726	16264	8054	13241
494_bus	494	1666	7806	2319	2685
bcspwr05	443	1623	7583	2250	2499
mesh2e1	306	2018	24545	10145	11439
tols90	90	1746	318	267	1185
bcspwr01	39	131	293	156	164

Table 5.1: Comparison of AMD algorithms

The test results show that, in most cases, the amount of nonzero entries after factorization (LU factorization) is decreased when using the developed algorithm, even having less than when using the native function. Only exceptions are the matrices *sherman1*, where just factorization is better than both options, and *dF_ne*, where the Matlab function has less nonzero values than the algorithm.

In the cases of the complete and augmented forms of the OPC, the reduction in the number of fill-ins of approximately 7.21%, which means that it is a viable option to implement. Specifically for the case of the complete system, Figure 5.3 illustrates the sparsity pattern of the matrix in combination with the permuted by both the proposed algorithm and Matlab's native function, and the resulting factorized matrices, in the case of the system with only linear constraints. Here, it is clear that the use of an AMD algorithm greatly reduces the number of nonzero entries in the resulting matrices and that the pattern affects their structure.

These results are of particular interest for solving the problem due to the significant reduction in the number of calculations required when solving a linear equation system. For example, in the case of the complete system shown in the figure, the reduction of nonzeros is approximately 95.95%, which translates to a proportional reduction in computational effort and a substantial decrease in processing time.

Although the benefits of this reduction are clear, generating the permutation order itself introduces additional computational complexity. Although the AMD algorithm is efficient, it can become time-consuming for large-scale problems. For example, the largest system considered here involves matrices of size 2708×2708 without nonlinear constraints, increasing to 3010×3010 when such constraints are included. In these

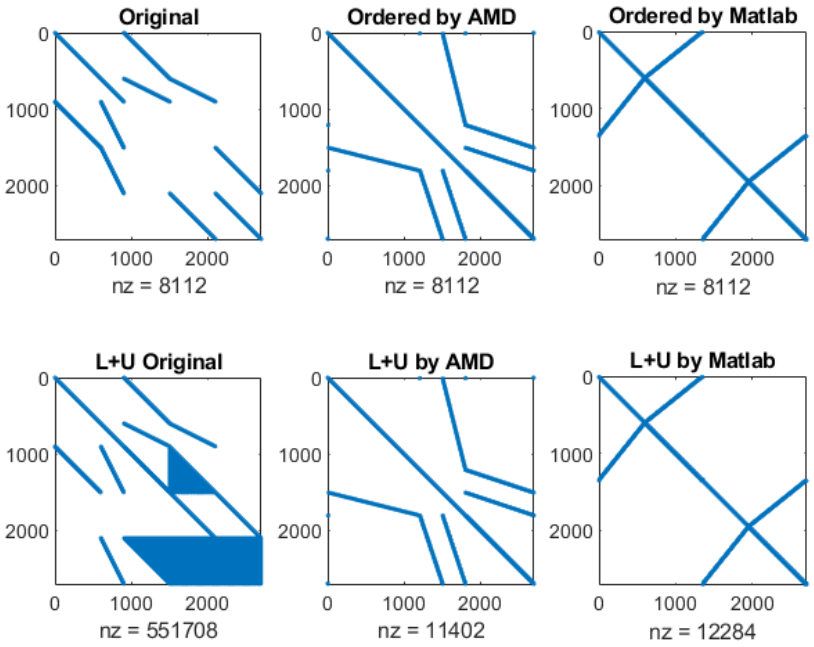


Figure 5.3: Sparsity patterns of Matrix dF_comp

cases, the time spent generating the AMD reordering may outweigh the time saved during the factorization step, potentially negating the overall efficiency gains.

To address this issue, the constant (or nearly constant, in the case of nonlinear constraints) structure of the KKT matrix can be exploited. By calculating the permutation offline, prior to the solver's execution, the precomputed order can be applied directly to the KKT matrix during each iteration. This approach preserves the efficiency advantages of the AMD algorithm while avoiding its runtime increase. The only additional computational effort is to apply the permutation to the matrix, which Matlab handles efficiently using its built-in sparse matrix operations.

Another advantage of this method is that, since it is applied before solving the linear equation system, it can be effectively combined with other accelerating strategies, such as the alternative approach discussed in this work or the ones included with Matlab's native functions. Although the possibility of a combination between the approaches has not been explored in the present work, it represents a promising direction for future studies. The influence of the AMD method on computational efficiency will be analyzed and discussed with the results presented in the following chapter.

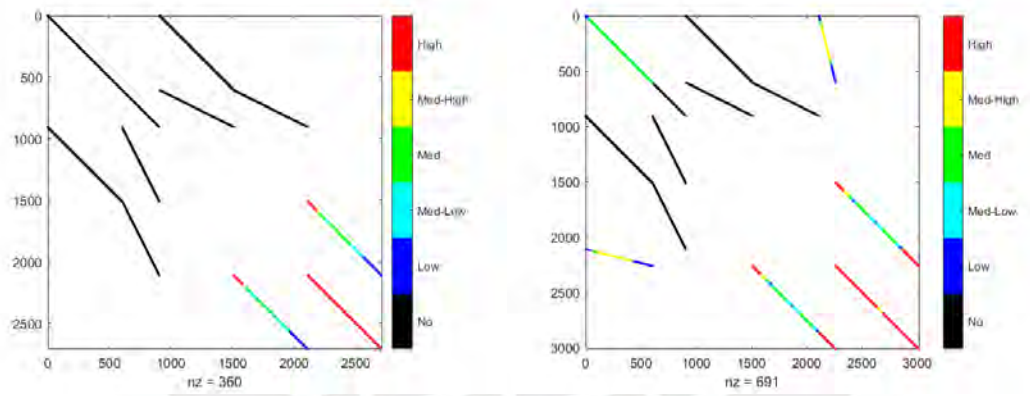
5.5 Constant Factorization Approach

The other significant point of interest in this work is the factorization process itself, which plays a critical role in solving linear equation systems by dividing it into less computationally complex problems to solve because of its structure.

The traditional approach for these methods includes an entry-per-entry calculation of the resulting matrices, which can lead to time-intensive processing in the cases of large systems, such as the one being analyzed. To mitigate this, the introduction of a block processing approach can be done, with the selection of the sub-block sizes being the principal source of the acceleration. The general rule is that larger blocks will decrease the number of calculations needed. However, this can also impact the overall process due to the factorized matrices partially lacking the special structures that the traditional methods have that can be exploited by different solvers to optimize the calculation of the equation systems.

Recognizing this limitation, a specialized method was developed to reduce processing times for normal and block factorization. Each iteration of the solver produces a new equation system, whose coefficients can be arranged into the KKT matrix, such as in the positions reserved for the equality constraints or the Lagrange multipliers λ and the slack variables s. Having mapped the sectors according to the rate of change they have through the operation helps to detect the parts that are more constant.

Figure 5.4 shows a color map that represents this mapping for the case where only linear constraints are considered (a) and for the case with nonlinearities (b).



(a) Linear constraints (b) Nonlinear constraints

Figure 5.4: Color map of changes in the KKT Matrix

The colors represent the changes, with red being the most variable and blue nearly constant. The black entries mean that they remain unchanged throughout the duration of the solution. It can be seen that there are zones in the graphs that are constant, so there is the potential of saving processing time if their factorization is calculated beforehand. Therefore, a method that uses offline calculated factorized matrices as the base for the ones processed online to avoid redundant calculations of the sectors that remain constant during the operation.

The proposed strategy takes advantage of the positioning of the constant sector. When it is found in the upper left corner as in the case of figure (a), no further pre-processing is needed. On the other hand, when the constant entries are found in varying positions throughout the matrix as in Figure (b), it is useful to apply a permutation that locates those on the desired position. It is important to ensure this because the calculations start with that area.

Due to the iterative nature of the calculation of the factorized matrices in both the traditional and block cases, the processing of the factorization starts on the upper right corner and calculates the entries or matrices found there and on the same column for the triangular matrices, and follows diagonally in the direction of the lower down section. This means that the first sector of the factorized matrices is only dependent on the same area on the KKT matrix, while the last one depends on the entirety of the matrix.

This is the behavior that is being exploited. If the first section is constant, the same location will be constant for the factorized matrices, which means that it would be redundant to calculate it at each solver's iteration. To avoid this redundancy an offline complete factorization can be done in the initialization stage of the complete algorithm can be done. Then, at each iteration, the calculation will be limited to the variable sections, saving processing time.

The main setback this approach has is the dependence on the constant sector of the analyzed matrix, since the time saved is directly proportional to its size. In those cases, it would be important to identify the degree of change of the least variable areas, where it could be possible to apply the method by introducing a constantly changing diagonal matrix, which would not increase the complexity of the calculations by much. This modification to the approach is not discussed in further detail in this work, but indicates an initial point for further studies.

6 Evaluation

Having introduced the approaches, this chapter focuses on presenting the results obtained from their practical implementation in the testing environment. Their performance is compared to the traditional methods are discussed in the context of the research objectives, addressing their significance, limitations, and potential applications. First, the data is presented with their interpretation, which helps to understand the impact the approaches have on the main problematic. Then a discussion of the results is done, combining the theoretical knowledge with the practice, where a deeper insight on the potential effect of the approaches on other systems is given.

6.1 Results

The first test carried out had the objective of comparing the performance of native functions Matlab when solving the OPC using interior-point methods with the one developed in this work. The chosen functions are *fmincon*, which serves as a generic optimization solver that lacks some optimizations made for an MPC application, and *mpcInteriorPointSolver*, which was developed with the goal of solving complex MPC systems. The proposed solvers include a simple direct IPM algorithm and one which includes the AMD approach. Table 6.1 summarizes the result of this test. The test system is the one that includes only linear constraints for a simpler implementation and to compare the fastest calculation times for each method. The values were obtained using a test sample size of 1000 solutions, except in the case of *fmincon* where only 10 were used due to the fact that it took considerably longer than the other solutions.

Functions	Time
fmincon	69.1225 s
mpcInteriorPointSolver	1.0240 s
Own Interior Point Alg.	0.0353 s
Own Interior Point with AMD Alg.	0.0397 s

Table 6.1: Comparison between Matlab's functions and own algorithm

Here, it can be observed that the developed algorithms reduce the calculation time in 96.55% compared to the MPC Toolbox solver, this is probably due to the fact that they are made specifically for the analyzed linear system. The worst performing function is *fmincon*, as it is meant for a wider number of generic optimization problems where an accurate solution is more important than the calculation time. These results indicate that the developed IPM algorithm has the potential for practical application with expected

good performance. In addition, they serve as a baseline for the performance of the approaches proposed in this thesis.

The first approach is tested by comparing other permutation algorithms with the developed AMD. Here, the tested system includes non-linear constraints to review how it changes from the other case and to see the performance of the proposal in a more complex case. The permutations included are the proposed AMD, and Matlab's native AMD and nested dissections (ND) functions. Table 6.2 shows the summary of the results of this test. Here, the complete and augmented forms of the KKT matrix are shown separately, as well as the symmetric and non-symmetric cases. Due to the sparsity loss when using the normal-equations form that made it take considerably longer in comparison with the others to calculate, it was not considered for this comparison. Those results were obtained by solving the OPC using a time step of $0.1s$, a horizon of $15s$, and 1000 samples for each method, then, the average values of the duration times for the solutions were calculated.

Method	**Time**
Complete Non-Symm	0.0891 s
Complete AMD Non-Symm	0.0969 s
Complete Symm	0.0644 s
Complete AMD Symm	0.0677 s
Complete Matlab's AMD Symm	0.0689 s
Complete Matlab's ND Symm	0.0714 s
Augmented	0.0607 s
Augmented AMD	0.0633 s
Augmented Matlab's AMD Symm	0.0648 s
Augmented Matlab's ND Symm	0.0640 s

Table 6.2: Summary of the Methods used for the Problem with Nonlinear Constraints

The symmetric case performs 38.35% better than the non-symmetric case, because the solver uses symmetry to solve the system with LDL factorization instead of LU, which is proven to be faster due to the resultant matrices. In terms of the permutations, the results show that ND performs slightly worse in comparison with the AMD's, for which the one developed in this work has a 1.74% faster calculation time for the complete matrix form. For the augmented case, ND performs better than Matlab's AMD, but it is still slightly worse than the thesis one by 1.09%. However, the base solution, without permutations, needs 6.05% less time to calculate. This would mean that the structure of the matrix does not need to be permuted to achieve low calculation times, which will be revised in the discussion section of this chapter, as a potential explanation is mentioned. The importance of this test is to show that the concept of using an AMD permutation can achieve faster calculations times when the reordering is done in such a way that the number fill-ins is minimized.

The results of the second approach testing are presented in Table 6.3, where the summary of the average calculation time for the factorization step is given. The methods compared are Matlab's native LDL function, and the developed functions for both entry-per-entry and block LDL, and ones that incorporate constant information. The test was done using the KKT matrix in the case of linear constraints due to having a clear dis-

tinction between the constant and variable parts, which was then used on the modified LDL functions. The results were obtained by taking the average of the calculation time after 1000 samples.

Algorithm	Time
Matlab's LDL	0.0017 s
LDL	17.661 s
Constant	7.1046 s
BLDL	0.5306 s
Constant BLDL	0.3436 s

Table 6.3: Summary of the Methods used for the Problem with Nonlinear Constraints

These values show that the Matlab function for LDL has an optimized performance, with a really low amount of time to produce the factorized matrices. The developed functions are really slow in comparison with it, so the analysis will be done by comparing the performance of the modified factorizations with constant information with their traditional form. First, the block methods save around 97% of the calculation time, which is a great reduction compared to the traditional case and is the reason why they are preferred in problems where the number of variables involved is high, such as the one being analyzed. Then, a reduction of 59.77% of the calculation time is achieved by using the constant variant of the normal function, while the block case reaches a reduction of 35.24%. These indicators show the high effectiveness of using modified functions with constant information, which is the main objective of the second approach.

6.2 Discussion

The results presented in the previous section provide a valuable insight into the effect the proposed approaches have on the solution of optimal control problems, with the aim of those approaches being modifications on the KKT conditions matrix or its factorization. Initially, the results seem to be not successful, but that may be caused by other factors that are external to the concepts presented in this thesis. This subsection focuses on those discussions, as well as giving a deeper insight on the potential benefits the approaches have for practical applications of these methods.

The first discussion topic is the results from the use of the interior-point method algorithms. First, it is evident that the algorithm developed has better performance compared to those native to Matlab. The reason behind these results are the code optimizations made to the IPM algorithm to better fit the system studied, a multivariable double integrator. Its KKT conditions matrix structure was known beforehand, including the mapping of constant values, which meant that at each time iteration it just needed to be updated, saving some processing time that the other algorithms could not do due to the lack of the information.

The controlled system serves as a good base for testing new methods or modifications with the objective of achieving faster calculations because of its simplicity. This model

is one of the most used for control applications because it is able to represent many different dynamics, especially in the mechanical field. Even in cases where the system has nonlinear behavior, by using linearization techniques, the model can represent the real dynamics very accurately, given that the values remain inside a defined range. That is why it is commonly used to test and develop new control methods as seen in (Rao and Bernstein, 2001), (Ferrari-Trecate, Galbusera, Marciandi, and Scattolini, 2009), and (Huba, Vrancic, and Bistak, 2022). This means that despite being optimized to work on this system, a direct application of the methods presented in the work will have an impact on real controlling systems.

The first approach was proven to produce faster calculation times when using its permutation order, when comparing it to other similar algorithms like another AMD and ND. A better result in this case would have been to also be faster than the non-permuted case, as the theory indicates that a permutation which maintains the sparsity after factorization of a matrix should take less processing time due to the zero entries being omitted when calculating. This is proven when analyzing the results of the other methods in comparison with the one proposed, the last producing fewer fill-ins and results in a decrease in calculation time.

However, the direct solution of the linear equation system seems to be faster than the AMD permuted one, even though it makes the matrix denser when factorizing. The reason behind this unexpected behavior is the algorithm Matlab's backslash function uses for the solution of linear equation systems. As mentioned in the State of the Art chapter, this function is not limited to simply inverse a matrix for the solution, instead, it runs the system through a series of conditions to choose the best performing algorithm. For the case studied, because the matrix is symmetrized by using a change of variable and not being positive definite, the function uses the LDL solver. This solver then checks again for the structure of the matrix, as a direct application of any factorization method can result in undefined calculations due to it relying on divisions of entries or matrices that could be zero or singular. If it is found that one cannot directly apply, a permutation step is done. The time spent on this action depends on the structure; if it is found to be nearly optimal for factorization, it is greatly reduced. That is why the AMD presented that produces less fill-ins has a better performance than the other methods. However, the way in which the approach was implemented indicates that two attempts to reorder the matrix are done, which, even though it takes less time for the second permutation, makes it take longer in comparison to the non-permuted case.

The potential feasibility of these methods was a key consideration in the development of the algorithms, with calculation and sampling times being critical factors. Given a sampling time of $0.1s$, the processing time for any of the methods must remain below this threshold to ensure the potential applicability in real time. The calculation times achieved indicate that the IPM algorithm, combined with the proposed method, is potentially feasible, as it operates within this limit. However, the feasibility of practical implementation would depend on the specific characteristics of the controller, as its processing power must be sufficient, and the demands of the application, as the sampling time could be increased causing less variables to be involved in the calculations leading to faster solutions. Furthermore, reducing the calculation times even further is essential to allow computational resources to be used in other critical processes. For example, in the case of trajectory following, it can be considered that two different OPC

could be solved, one to indicate what is the optimal trajectory of the systems states through an environment that could include obstacles that need to be avoided, while the other would be focused on control tasks by calculating the law for the input of the plant. Other things to consider are communication between the components of the control system, as delays and transfer times should also be taken into account.

In conclusion of this result, it has been proven that the concept behind the proposed method helps to solve the problem of reducing the calculation times of the optimal control problem. Additionally, the calculation times achieved are lower than the sample time, which makes them potentially feasible for real time applications. However, the conditions to allow for this implementation must be considered, and thus a combination with the other approach could be useful.

The next discussion point is the result of the tests from the second approach. The direct deduction from these is that to achieve faster performance times for factorization methods, a block approach that exploits the mapping of the changing entries to identify the constant sectors can be used. The results were obtained from the testing using the studied case; this means that those big improvements seen in the calculation times are caused by the broad constant sections present in its KKT matrix. Because of this, this approach is more useful when applying the new factorizations to systems that allow this characteristic to be present.

Another obvious result is the broad superiority of the Matlab algorithm compared to the ones developed. This is mainly due to the fact that its functions are optimized by the efforts of a group of developers, which allowed them to achieve such short processing times. This means that a similar implementation of the algorithm that incorporates the modifications made for the approach could have a greater impact on the performance. Additionally, Matlab has a symbolic toolbox where the calculation of the matrices could be done faster by only assigning the needed values to an already set structure, because its changes are minimal. However, this suggestion has to be carefully analyzed, because in the moment the toolbox cannot handle in a time efficient manner big matrices like the one gotten from the studied case.

In the present work, a variation of the system was introduced by adding a nonlinear constraint, which made some of the previously constant parts of the matrix lose this feature. This was caused by making more of these sectors depend on the current values of the system leaving only the equation constraints part that is related to the dynamic system constant because it has linear behavior. When nonlinear dynamics are considered, most of the KKT conditions matrix would be depending on the state values, meaning that the constant parts will be greatly reduced, which lowers the effectiveness of the constant approach. The evaluation of these cases could be the next step for a more generalized method, such as introducing intermediate matrices that have a low update cost, which could serve as mathematical corrections for the factorized ones.

Both approaches have the potential to decrease the processing time for the solution of optimal control problems. This also opens the possibility of a combined method, where an approximate minimum degree also uses the variation mapping of the matrix to reorder the matrix to be able to apply a constant factorization. However, this is a complex task, as a traditional AMD algorithm uses the undirected graph representation of the matrix to estimate the degree of each node, which represents the values of the

diagonal, with the other entries being represented by edges. Here, a modification could be made to give more weight to the edges where its value is more constant and to the nodes so that they are also considered. This has to be carefully done, because if the degree is varied by a great amount, the permuted matrix would lose the structure that allowed it to maintain its sparsity when solving the equation system. This opens the possibility for future work to tackle this approach and achieve better performance.

7 Conclusions and Recommendations

This thesis had the primary objective of finding potential improvements in the calculation time for the solution of the linear equation system generated by the KKT-conditions when solving optimal control problems. This was analyzed and tested by using a multivariate double integrator system, where an algorithm of the interior-point method is used for the calculation of the control law for the stabilization around the origin case.

To do this, two approaches were proposed, one that focuses on using an approximate minimum degree algorithm to reduce the amount of fill-ins of the factorized matrices resultant from the coefficient matrix, and the other that uses information on how it changes through the iterations of the solvers. Both aim to avoid as much calculations as possible, reducing in this way the number of operations done at each solver's iteration. This chapter presents the conclusions and recommendations derived from the development of the work.

7.1 Conclusions

- The approximate minimum degree (AMD) algorithm was proven to achieve improved computational efficiency when used as a reordering for the solution of linear equation systems. By taking advantage of the constant structure the KKT conditions matrix has through the iterations, a single offline calculation for the permutation strategy is needed to reduce the computational effort during the solution when compared to other similar permutations. This is caused due to its capacity of minimizing the number of fill-ins produced during the factorization, maintaining the sparsity that can be used for faster, less complex operations.
- Block factorizations that use a change map of the matrix through iterations have the potential to significantly reduce processing times. This was shown when comparing them to similar methods that lacked some of the characteristics of the developed one. Although the results are promising for the solution of the problem, further optimizations should be done in implementation to achieve practical viability.
- The methods were applied and show improved performance on a linear system, which implies that they lack generality for a large number of real dynamics, like the nonlinear cases. However, many applications of optimal control, like model predictive control (MPC), consider linear system approximation in an operation range, thus meaning that in this field the approaches serve as foundational steps toward more efficient solutions.

- Although the approaches show potential as solution concepts for the problem, further research and development efforts should be considered for practical application. This means that the methods presented could be taken as starting points for future work that attempts to improve the solution of the problem.

7.2 Recommendations

- A combined approach using an AMD permutation with constant information for the KKT matrix could achieve better performances when compared to each of them separately. Then, future work should analyze their individual impact and develop strategies that can balance them for optimal results, as they seem to tend to interfere with each other.
- Using a symbolic approach for the factorizations and matrix inversions should be explored. The computational effort of recalculating the factorized or inverse matrices is the key component of the overall solvers performance. By having precalculated matrices which are updated, this step could decrease its complexity and achieve faster results. This approach should be analyzed in future work using an appropriate coding environment, as Matlab's is not suited for big-dimensioned matrices.
- An extension of the presented methods for the nonlinear system case could be done. This would give them more generality, allowing them to be implemented on a larger number of real systems.
- Further optimizations of the coding are required in the approaches to achieve superior performance. This should be done to ensure successful real-time applications, as they have additional processes that are needed for the complete control of the system.

Bibliography

Amestoy, P. R., Davis, T. A., & Duff, I. S. (1996). An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, *17*(4), 886–905. doi:10.1137/S0895479894278952

Biswas, B., Chatterjee, S., Mukherjee, S., & Pal, S. (2013). A discussion on euler method: A review. *Electronic Journal of Mathematical Analysis and Applications*, *1*(2), 2090–2792. Retrieved from https://ejmaa.journals.ekb.eg/article_309811_cb4e94d217da1b8e29e3c9294ccd8e0a.pdf

Camacho, E. F., & Bordons, C. (2004). *Model predictive control* (2. ed.). Archivierung/-Langzeitarchivierung gewährleistet PEBW pdager DE-31. London: Springer.

Chapra, S. C., & Canale, R. P. (2015). *Numerical methods for engineers* (7th). New York: McGraw-Hill Education.

Chen, J., Jin, Z., Shi, Q., Qiu, J., & Liu, W. (2013). Block algorithm and its implementation for cholesky factorization. *ICCGI 2013*, 245.

Cohen, M. B., Lee, Y. T., & Song, Z. (2018). Solving linear programs in the current matrix multiplication time. *CoRR*, *abs/1810.07896*. arXiv: 1810.07896. Retrieved from http://arxiv.org/abs/1810.07896

Cutler, C. R., & Ramaker, B. L. (1980). Dynamic matrix control - a computer control algorithm. *Joint Automatic Control Conference*, *17*, 72. doi:10.1109/JACC.1980.4232009

D'Azevedo, E., & Hill, J. (2012). Parallel lu factorization on gpu cluster. *Procedia Computer Science*, *9*, 67–75. Proceedings of the International Conference on Computational Science, ICCS 2012. doi:https://doi.org/10.1016/j.procs.2012.04.008

Davis, T. A. (2006). *Direct methods for sparse linear systems*. doi:10.1137/1.9780898718881. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898718881

Davis, T. A., & Hu, Y. (2011). The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software 38, 1, Article 1*. doi:10.1145/2049662.2049663

Demmel, J., Higham, N., & Schreiber, R. (2000). Block lu factorization.

Demmel, J. W., & Higham, N. J. (1992). Stability of block algorithms with fast level-3 blas. *ACM Trans. Math. Softw.*, *18*(3), 274–291. doi:10.1145/131766.131769

Ekatpure, R. (2023). Machine learning for enhancing vehicle safety and collision avoidance systems in automotive development: Techniques, models, and real-world applications. *Journal of Computational Intelligence and Robotics*, *3*(2), 1–43. Retrieved from https://thesciencebrigade.com/jcir/article/view/257

Estrada, M., Li, S., & Cai, X. (2021). Feedback linearization of car dynamics for racing via reinforcement learning. arXiv: 2110.10441 [math.OC]. Retrieved from https://doi.org/10.48550/arXiv.2110.10441

Faulwasser, T., Mehrez, M., & Worthmann, K. (2021). Predictive path following control without terminal constraints. In T. Faul-wasser, M. A. Müller, & K. Worthmann (Eds.), *Recent advances in model predictive control: Theory, algorithms, and applications* (pp. 1–26). doi:10.1007/978-3-030-63281-6_1

Ferrari-Trecate, G., Galbusera, L., Marciandi, M. P. E., & Scattolini, R. (2009). Model predictive control schemes for consensus in multi-agent systems with single- and double-integrator dynamics. *IEEE Transactions on Automatic Control*, *54*(11), 2560–2572. doi:10.1109/TAC.2009.2031208

Ghojogh, B., Ghodsi, A., Karray, F., & Crowley, M. (2021). Kkt conditions, first-order and second-order optimization, and distributed optimization: Tutorial and survey. arXiv: 2110.01858 $[\mathrm{math.OC}]$. Retrieved from https://arxiv.org/abs/2110.01858

Gilbert, J., Moler, C., & Schreiber, R. (1997). Sparse matrices in matlab: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, *13*. doi:10.1137/0613024

Golub, G., & Van Loan, C. (2013). *Matrix computations*. Johns Hopkins University Press. Retrieved from https://books.google.de/books?id=X5YfsuCWpxMC

Guo, J., Liang, H., Ai, S., Lu, C., Hua, H., & Cao, J. (2021). Improved approximate minimum degree ordering method and its application for electrical power network analysis and computation. *Tsinghua Science and Technology*, *26*(4), 464–474. doi:10.26599/TST.2020.9010019

Haddad, C. N. (2009). Cholesky factorization. In C. A. Floudas & P. M. Pardalos (Eds.), *Encyclopedia of optimization* (pp. 374–377). doi:10.1007/978-0-387-74759-0_67

Higham, N. J. (1999). Stability of block ldlt factorization of a symmetric tridiagonal matrix. *Linear Algebra and its Applications*, *287*(1), 181–189. doi:https://doi.org/10.1016/S0024-3795(98)10074-5

Holkar, K., & Waghmare, L. M. (2010). An overview of model predictive control. *International Journal of control and automation*, *3*(4), 47–63.

Hrovat, D., Di Cairano, S., Tseng, H., & Kolmanovsky, I. (2012). The development of model predictive control in automotive industry: A survey. In *2012 ieee international conference on control applications* (pp. 295–302). doi:10.1109/CCA.2012.6402735

Huba, M., Vrancic, D., & Bistak, P. (2022). Reference model control of the time delayed double integrator. *IEEE Access*, *10*, 39282–39298. doi:10.1109/ACCESS.2022.3165645

Inc., T. M. (2022). Matlab version: 9.13.0 (r2022b). Natick, Massa-chusetts, United States: The MathWorks Inc. Retrieved from https://www.mathworks.com

Jiang, H., Kathuria, T., Lee, Y. T., Padmanabhan, S., & Song, Z. (2020). A faster interior point method for semidefinite programming. In *2020 ieee 61st annual symposium on foundations of computer science (focs)* (pp. 910–918). doi:10.1109/FOCS46700.2020.00089

Katkade, S. N., Bagal, V. C., Manza, R. R., & Yannawar, P. L. (2023). Advances in real-time object detection and information retrieval: A review. *Artificial Intelligence and Applications*, *1*(3), 139–144. doi:10.47852/bonviewAIA3202456

Kaya, C. Y., & Martínez, J. M. (2007). Euler discretization and inexact restoration for optimal control. *Journal of Optimization Theory and Applications*, *134*(2), 191–206. doi:https://doi.org/10.1007/s10957-007-9217-x

Langr, D., & Tvrdík, P. (2016). Evaluation criteria for sparse matrix storage formats. *IEEE Transactions on Parallel and Distributed Systems*, *27*(2), 428–440. doi:10.1109/TPDS.2015.2401575

Lee, Y. T., & Sidford, A. (2015). Efficient inverse maintenance and faster algorithms for linear programming. arXiv: 1503.01752 [cs.DS]

Lu, Y., Luo, Y., Lian, H., Jin, Z., & Liu, W. (2021). Implementing lu and cholesky factorizations on artificial intelligence accelerators. *CCF Transactions on High Performance Computing*, *3*(3), 286–297. doi:https://doi.org/10.1007/s42514-021-00075-8

Morari, M., & H. Lee, J. (1999). Model predictive control: Past, present and future. *Computers I& Chemical Engineering*, *23*(4), 667–682. doi:https://doi.org/10.1016/S0098-1354(98)00301-9

Nesterov, Y., & Nemirovskii, A. (1994). *Interior-point polynomial algorithms in convex programming*. SIAM.

Nocedal, J., & Wright, S. J. (1999). *Numerical optimization* (P. Glynn & S. M. Robinson, Eds.). Springer.

Rao, V., & Bernstein, D. (2001). Naive control of the double integrator. *IEEE Control Systems Magazine*, *21*(5), 86–97. doi:10.1109/37.954521

Richalet, J., Rault, A., Testud, J., & Papon, J. (1978). Model predictive heuristic control: Applications to industrial processes. *Automatica*, *14*(5), 413–428. doi:https://doi.org/10.1016/0005-1098(78)90001-8

Roos, C., Terlaky, T., & Vial, J.-P. (2006). *Interior point methods for linear optimization* (Second). Springer.

Schulze Darup, M., & Book, G. (2021). On closed-loop dynamics of admm-based mpc. In T. Faulwasser, M. A. Müller, & K. Worthmann (Eds.), *Recent advances in model predictive control: Theory, algorithms, and applications* (pp. 107–134). doi:10.1007/978-3-030-63281-6_5

Selman, A. H. (2016). Lu factorization algorithm with minimum degree ordering in power distribution network problems. *Southeast Europe Journal of Soft Computing*, *4*(2). doi:http://dx.doi.org/10.21533/scjournal.v4i2.91

Shahnaz, R., Usman, A., & Chughtai, I. R. (2005). Review of storage techniques for sparse matrices. In *2005 pakistan section multitopic conference* (pp. 1–7). doi:10.1109/INMIC.2005.334453

Smailbegovic, F., Gaydadjiev, G. N., & Vassiliadis, S. (2005). Sparse matrix storage format. In *Proceedings of the 16th annual workshop on circuits, systems and signal processing* (pp. 445–448).

Sprodowski, T. (2021). Collision avoidance for mobile robots based on an occupancy grid. In T. Faulwasser, M. A. Müller, & K. Worthmann (Eds.), *Recent advances in model predictive control: Theory, algorithms, and applications* (pp. 219–244). doi:10.1007/978-3-030-63281-6_9

Srivastava, N., Jin, H., Liu, J., Albonesi, D., & Zhang, Z. (2020). Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product. In *2020 53rd annual ieee/acm international symposium on microarchitecture (micro)* (pp. 766–780). doi:10.1109/MICRO50266.2020.00068

Zhang, Z., Wang, H., Han, S., & Dally, W. J. (2020). Sparch: Efficient architecture for sparse matrix multiplication. In *2020 ieee international symposium on high performance computer architecture (hpca)* (pp. 261–274). doi:10.1109/HPCA47549.2020.00030